Cyprus University of Technology | Faculty of Engineering and Technology

Doctoral Dissertation

Natural Language Processing with Deep Neural Networks

Tolias Kyriakos

Limassol, September 2020

CYPRUS UNIVERSITY OF TECHNOLOGY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING,

COMPUTER ENGINEERING AND INFORMATICS

Doctoral Dissertation

Natural Language Processing with Deep Neural Networks

Tolias Kyriakos

Limassol, September 2020

Approval Form

Doctoral Dissertation

Natural Language Processing with Deep Neural Networks

Presented by

Kyriakos Tolias

Supervisor: Dr. Sotirios Chatzis, Assistant Professor

Signature: _____

Member of the committee: Dr. Sergios Theodoridis, Professor

Signature: _____

Member of the committee: Dr. Nicolas Tsapatsoulis, Professor

Signature: _____

Cyprus University of Technology

Limassol, September 2020

Copyrights

The approval of the dissertation by the Department of Electrical Engineering, Computer Engineering and Informatics does not imply necessarily the approval by the Department of the views of the writer.

I would like to express my sincere gratitude to my advisor Assistant Prof. Dr. Sotirios Chatzis for the continuous support of my PhD study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Also, I would like to thank from the bottom of my heart Dr. Sotos Voskarides who inspired me to chase my academic dream.

In my father's memory

Abstract

Since the very first days of the Computer Era, machines provided us the ability to collect and store vast amount of information. It, soon, became obvious that harvesting information was an entirely different task, much more complicated and demanding. Many solutions developed to make it possible for humans to communicate with computers, aiming data mining. Databases, query languages and search engines were for decades the most prevalent solution. At first, special skills were required such as query language knowledge or search engine's syntax to be able to perform advanced tasks. In order to be adopted by users, search should be easy and human friendly. Nowadays, search engines are using simple language syntax and have made significant progress towards natural language path.

Still, search engines fall sort on combining information from different parts producing a synthetic answer. Computers are actually computational machines, therefore are excellent at manipulating syntax and calculating words' frequency but are weak in recognizing concepts behind the words. A traditional search engine, is not able to draw conclusions or realize the context of a dialogue.

Machine Learning has been proven strong in dealing with such concepts. One of the most challenging fields of machine learning is the Natural Language Processing (NLP), especially its component Natural Language Understanding (NLU). The crest of NLP are the question-answering and summarization tasks, in sense that strong cognitive ability is required in order the conceptual context to be extracted. Supervised learning of deep neural networks, is currently the best available tool for these tasks. Despite the rapid advances in the field of Machine Learning, their performance remains poor when dealing with hard NLU and NLP tasks, such as abstractive summarisation and question answering.

This dissertation aims to offer substantive and measurable progress in both these areas, by ameliorating a key problem of modern machine learning techniques: The need for dense and large data corpora for effective model training. This is an especially hard task in the context of such applications. To this end, we leverage arguments from the field of Bayesian inference. This allows for better dealing with the modelling uncertainty, which is the direct outcome of data sparsity, and results in poor modelling/generalisation performance. Our approaches are founded upon solid and elaborate statistical inference arguments, and are evaluated using challenging popular benchmarks. As we show, they offer tangible performance advantages over the state-of-the-art.

Keywords: Deep Learning, Machine Learning, Bayesian Inference, Variational Bayes, Natural Language Processing

Contents

[1] We have developed our source codes in Python, using the TensorFlow library [1]. We run our experiments on a server with 64GB RAM and an NVIDIA Tesla K40 GPU.

List of Figures

List of Tables

List of Abbreviations

AI: Artificial Intelligence
AVI: Amortized Variational Inference
BoW: Bag-of-Words
BP: BackPropagation
BPTT: Back Propagation Through Time
DOF: Degrees Of Freedom
DNC: Differentiable Neural Computer
ELBO: Evidence Lower Bound
GD: Gradient Descent
GRU: Gated Recurrent Unit
KL: Kullback Leibler (divergence)
LSTM: Long Sort-Term Memory
MANN: Memory-Augmented Neural Networks
MC: Monte Carlo
ML: Maximum Likelihood
NTM: Neural Turing Machine
pdf: probability density function
PE: Position Encoding
ReLU: Rectified Linear Unit
RN: Random Noise
RNN: Recurrent Neural Network
SBG: Shannon-Boltzmann-Gibbs (entropy)
SGD: Stochastic Gradient Descent
TE: Temporal Encoding
VAEs: Variational AutoEncoders
VB: Variational Bayes

List of Publications

TOLIAS, Kyriakos; CHATZIS, Sotirios P. t-Exponential Memory Networks for Question-Answering Machines. IEEE transactions on neural networks and learning systems, 2018, 30.8: 2463-2477.

1 SUPERVISED MACHINE LEARNING

Supervised learning is the machine learning task of inferring a function from a training dataset. The **dataset** includes examples, each of which is a pair of an input object and a desired output value (label). The training starts by **initializing** the network's weights. Subsequently, the efficiency of a neural network, for a set of examples, is evaluated through a **loss function**. The derivative of the loss function is computed by the **backpropagation** method and it is used by an **optimization** (learning) algorithm to minimize loss of the inferred function, hence acquire knowledge. The function has to able to **generalize**, meaning that it should be able to produce the expected output for unseen input data. **Deep learning** allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [3].

1.1 Loss functions

The core concept in machine learning is the derivation of a function that would be a good estimator for a problem, given some known data. The most popular principle for this function evaluation is the **Maximum likelihood**. Considering an unknown, data generating, distribution $p_{data}(x)$, a set of m examples $\mathrm{X} = \{x^{(i)}, x^{(2)}, ..., x^{(m)}\}$ that are drawn independently from this distribution, and a parametric family of probability distributions $p_{data}(x;\theta)$ over the same space indexed by θ, then the maximum likelihood estimator for θ is:

$$\theta_{ML} = \underset{\theta}{arg\,max}\, p_{model}(\mathrm{X};\theta) = \underset{\theta}{arg\,max} \prod_{i=1}^{m} p_{model}(x^{(i)};\theta)$$

The existence of the product $\prod$ introduces possible underflow issues, for this reason the logarithmic expression of maximum likelihood, also known as **maximum log likelihood**, is most commonly used:

$$\theta_{ML} = \underset{\theta}{arg\,max} \sum_{i=1}^{m} log\, p_{model}(x^{(i)};\theta),$$

which can also be expressed as an expectation with respect to the empirical distribution p_{data}

$$\theta_{\mathrm{ML}} = \underset{\theta}{arg\,max} \mathbb{E}_{x\sim\hat{p}_{data}} log\, p_{model}(x;\theta)$$

Maximum likelihood estimation can be thought of the minimum dissimilarity between the empirical distribution and the model distribution.

This property is exploited by the **loss function**. The loss or cost function is an important part of supervised learning. It is used to measure the difference

between ground-truth y and the predicted value $\hat{y}$. Its value is always non negative and as it approaches to zero, the accuracy of the model is improved.

Cross-entropy is one of the most commonly used loss functions in artificial neural networks. It is based on *Shannon entropy* which expresses the amount of the uncertainty in a probability distribution $P(\mathrm{x})$:

$$H(x) = \mathbb{E}_{x\sim P}[I(\mathrm{x})] = -\mathbb{E}_{x\sim P}[logP(\mathrm{x})],$$

where $I(x)$ is defined as the self-information of an event $x = \mathrm{x}$.

In case of two individual probability distributions over the same variable x, the amount of their dissimilarity can be measured by the *Kullback–Leibler (KL) divergence*:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[log\frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[logP(x) - logQ(x)]$$

Cross-entropy, $H(P, Q)$ is similar to KL divergence, expressing the difference between the two probabilities, but rather simpler. Their difference lies on the *Entropy*, term $\mathbb{E}_{x\sim P}[logP(x)]$, that is missing in cross-entropy.

The goal of supervised learning is the minimization of the divergence between the two probability distributions with respect to the inferred distribution $Q(\mathrm{x})$. The Entropy ($\mathbb{E}_{x\sim P}[logP(x)]$) is not depended on Q_x, hence its omission does not affect the optimization.

The **cross-entropy** is defined as:

$$H(P, Q) = -\mathbb{E}_{x\sim P}logQ(\mathrm{x})$$

Other well known loss functions are the *Mean Squared Error* [4], *Mean Squared Logarithmic Error, L2, Mean Absolute Error, Mean Absolute Percentage Error, Negative Logarithmic Likelihood, Poisson, Cosine Proximity, Hinge and Squared Hinge.*

Considering the above, maximum likelihood is equivalent to the minimization of the negative log likelihood or the loss function. For this reason, in supervised training, the goal is to minimize the loss function by altering the network parameters, in order to model a probability distribution function (empirical) that better describes the actual distribution dictated by the training data.

1.2 Optimization algorithms

Gradient descent [5] is one of the most popular optimization algorithms of deep neural networks. As described in the previous section, in supervised learning, training is equivalent to optimizing a function $f(\theta)$ by altering θ. Training aims to minimize the loss function $f(\theta)$, which expresses the difference between the predicted $\hat{y}$ and the real values y (labels).

In most cases the model defines a distribution $p(y|\mathrm{x};\theta)$ and the **cross-entropy** is used as the loss function in pursuit of **maximum log likelihood**. Given that $f(\theta)$ is differentiable and $f(\theta) = y$, the derivative of the loss function, $f'(\theta) = \frac{dy}{d\theta}$, defines the slope of $f(\theta)$ at the point θ. In a sense, the derivative suggests the direction of θ values that leads to smaller values of $f(\theta)$.

So, gradient descent, as an optimization algorithm, is used to minimize a function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. More precisely, **gradient descent** is an indicator that shows the path to the minimum of the objective function $J(\theta)$, where $J(\theta)$ is parameterized by a model's parameters θ [6]. Minimum loss can be reached by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ with regards to the parameters. The size of steps taken towards minimum are defined by the *learning rate* η.

Gradient descent algorithms can be distinguished into three main categories, based on the amount of data used to compute the gradient of the objective function. As the amount of data used grows, the algorithm gets more precise, meaning that the accuracy of the parameter update increases, but the time required to perform an update is increases as well. The first variant, the **batch gradient descent**, uses the entire dataset to compute the gradient of the cost function w.r.t. the parameters.

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

This approach may become very slow because it needs to read the entire dataset for each update. At the same time, it may suffer from memory issues, since the entire dataset needs to be stored in memory. Furthermore, it cannot be trained on-the-fly with new data. When new training examples appear, the training must be started from the beginning.

The **stochastic gradient descent (SGD)**, is designed to perform a parameter update for each individual training example of the dataset. It is called stochastic because each example gives a noisy estimate of the average gradient over all examples. Considering the input of the example as $x^{(i)}$ and the output (label) as the $y^{(i)}$, the parameter update is

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

SGD is dealing better with data redundancy, since it computes the update for each example and therefore is much faster than batch gradient descent.

Additionally, this feature (single example update) allows the online training of the model. A potential drawback of SGD is that the convergence around minimum can be very complex, but experience has shown that it can be addressed by decreasing the learning rate η.

The last variant is coined **mini-batch gradient descent** and combines and combines concepts of the two previous variants. In many cases the amount of dataset examples is huge and the time to complete a single step of gradient calculation may be prohibitively long. The gradient can be thought as an expectation that can be approximated by a small set of samples. On every step a mini batch of examples is drawn from the dataset. So, an update for every mini-batch of n training examples is performed as

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

In this way, stability of the algorithm, in respect to convergence, is attained, since the variance of the parameter update decreases. Due to its performance and stability, the mini-batch gradient descent has became one of the standard algorithms used in deep supervised learning.
The learning process through gradient descent algorithm is presented in Fig. 1.

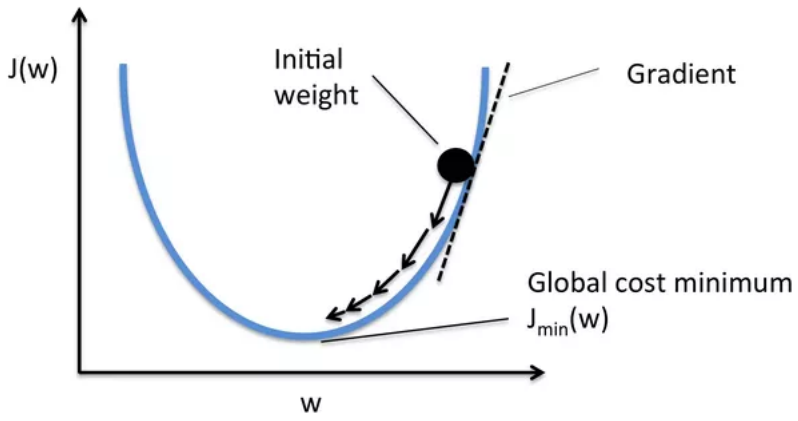


Figure 1: Gradient descent optimization process

During the last years, several extensions of SGD have been proposed in order to increase the accuracy and speed of convergence, such *Nesterov accelerated gradient (NAG)* [7], *Adagrad* [8], *Adadelta* [9], *Adam* [10], *AdaMax* [10] *and Nadam* [11], to name a few.

Adagrad is a gradient descent algorithm suitable for dealing with sparse data, exploiting the fact that its learning rate η can adapt to the parameters. It is thus able to perform large updates for infrequent parameters, while smaller updates are being effected for frequent ones. As mentioned Adagrad uses a different learning rate η for every parameter θ_i at every time step t. Let $g_{t,i}$ be the gradient of the objective function w.r.t. the parameter θ_i at time step t:

$$g_{t,i} = \nabla_{\theta t} J(\theta_{(t,i)})$$

Conventionally, the update for each parameter, at each time step t becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

Under Adagrad, at each time step t, the learning rate η for every parameter θ_i is updated by taking the past gradients of θ_i into account:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

where G_t is a diagonal matrix, in which the i, i element is the sum of squares of the gradients w.r.t. θ_i, up to time step t and ϵ is a constant (smoothing term) that prevents division by zero. Finally the update becomes:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

where $\odot$ is the element-wise multiplication.

Apart from the ability to deal with sparse data, Adagrad, takes care of the learning rate, making its tuning, less human demanding. On the other hand, the learning rate tends to shrink, and in some cases may become infinitesimally small, causing the model to cease acquiring any further knowledge.

1.3 Backpropagation

In forward propagation, a feed-forward neural network is fed with information x, which is then forwarded through the hidden units of each layer, until a prediction $\hat{y}$ is produced. **Backpropagation** [12] is a technique that computes the error introduced by every individual unit of the network.

At this point, it is essential to be clarified that the backpropagation is exclusively used to compute the gradient of the function, while the optimization

(learning) algorithm is responsible for converting gradient to knowledge, through loss minimization. Information about the error (cost), meaning the difference between the ground-truth and the predicted value, is propagated backwards through the network units in order the gradient to be computed. Following, the gradient is used by the optimization algorithm to adjust the weights of the network in such a way that minimizes the loss. The backpropagation process in a deep neural network is illustrated in Fig. 2 (the forward computation is shown with gray arrows).

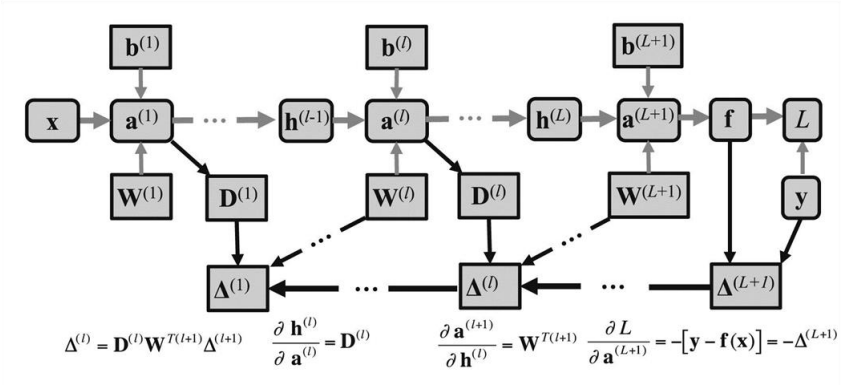


Figure 2: Backpropagation in a deep neural network

Backpropagation takes advantage of the **Chain Rule of Calculus** [13] (L'Hôpital , 1696), which states that the derivative of a function formed by composing other functions, can be computed using the derivatives of these functions, which (derivatives) are known. In that way the derivative computation becomes feasible and efficient (fast). Suppose that $\chi \epsilon \mathbb{R}^{\mathrm{m}}$, $y \epsilon \mathbb{R}^{\mathrm{n}}$, g maps from $\mathbb{R}^{\mathrm{m}}$ to $\mathbb{R}^{\mathrm{n}}$ and f maps from $\mathbb{R}^{\mathrm{n}}$to $\mathbb{R}$, then if $y = g(x)$ and $z = f(x)$ the derivative of z with respect to x_i is defined as:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

1.4 Parameter initialization

In order for the training process to commence, an initialization of the network's weights should be effected. In many cases of simple neural networks (single layered), assignment of very low values to the weights is enough. On the contrary, in deep networks, as the network layers increase, this initialization approach is not very efficient.

X. Glorot & Y. Bengio (2010) [14] suggested that the initialization of the weights is crucial to the stability of the network, by observing the effect of the random initialization of deep neural networks. They found out that gradient descent from random initialization is very poor on deep neural networks, especially those with non linear activation functions (e.g. sigmoid or RELU), and devised a new method coined **Glorot.**

The main idea is that each weight is initialized with a low Gaussian value with zero mean and variance based on the hidden nodes connected to that weight. More specifically,

$$W_{i\sim j} = U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}],$$

where $U[-a, a]$ is the uniform distribution in the interval $[-a, a]$ and n the size of the previous layer. They also noticed that this approach causes the variance of the back-propagated gradient to be dependent on the layer, so they proposed the addition of a normalization factor, which resulted in the **normalized initialization**, that is considered to be the standard initialization method in deep learning.

$$W_{i\sim j} = U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}]$$

1.5 Regularization

An important feature of a neural network is generalization, meaning the ability of the model to perform "as expected" on new unseen input data. Minimizing the loss function through training does not guarantee that the model will have the same behaviour under test examples. **Underfitting** is the case where the model performs poorly on training data and cannot generalize on new data, while **overfitting** is the case where it performs well under training data but cannot generalize its performance with new data. The latter happens usually when the model, during training, learns the data "too" well, so it ends up modelling the noise along with "core concepts" found in the training data.

The methods used to reduce the test errors of a model can be grouped under the term "regularization". One of the first methods was the **Parameter Norm Penalties**, in which a penalty term $\Omega(\theta)$ is added to the objective function J. The new regularized objective function $\tilde{J}$ is defined as:

$$\tilde{J}(\theta; \mathrm{X}, y) = J(\theta; \mathrm{X}, y) + \alpha\Omega(\theta),$$

where α is a non negative real number that controls the extend of regularization via the penalty norm Ω. Several penalty norms have been proposed such

as the $L^1$and the L^2, also known as **weight decay**.

Another regularization strategy is **data augmentation**. It is a very effective strategy but it is really difficult to be implemented. It requires the creation of a vast amount of training data which in many cases is not feasible.

In some cases, **noise injection** is being employed, forming a sort of data augmentation [15]. The injection of random noise to the input or the hidden units of a model, during training, is able to increase its robustness [16]. Furthermore, noise can be injected to the weights as proposed by Jim et al. (1996) [17] and Graves et. al (2011)[18]. Under a Bayesian view the weights of a model are considered uncertain and representable via a probability distribution that reflects this uncertainty. So, the noise injection to the weights can be thought as a practical, stochastic way to reflect this uncertainty. The output units could also be injected with noise in a process called **label smoothing** [19].

Dropout is another popular regularization technique, where non-output units are randomly removed during training time. This way, we reduce the effective number of parameters trained on each iteration of stochastic gradient descent, yet we eventually train all the postulated model parameters.

2 NEURAL NETWORKS FOR NLP

2.1 Sequence to sequence models

Many NLP tasks require processing of long sequences, such as sentences or phrases, in order for another output sequence to be produced. A typical example of such task is the Q&A, where facts and questions in the form of sentences (sequence of words) are fed to the input of a neural network, while the answer is expected in the output, in the form of a word or sentence. A second case is the Summarization task in which a text in the form of a set of sentences is analyzed by the model in order to extract a shorter comprehensive version of the given text, again in the form of a sequence of words.

Since 1986, a special family of neural networks have been developed for dealing with sequences. Recurrent neural networks or **RNNs** [12] were explicitly designed to handle sequence-to-sequence tasks. Input data in traditional neural networks are considered to be independent from each other. This approach is not well suited for sequences where each information is highly dependent on its predecessors. So, the network should employ a kind of "memory" to remember information already passed to the model. The idea that revolutionized the neural network world and lies behind the success of RNNs is **parameters sharing**. It actually functions as a short-memory enabling the network to take advantage of prior knowledge, by sharing the same weights across several time steps.

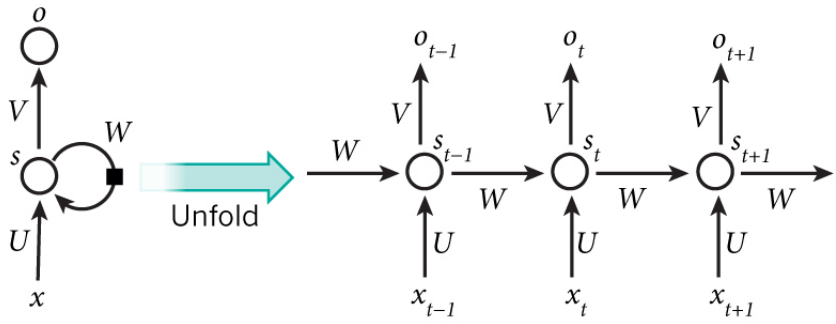


Figure 3: A recurrent neural network and the unfolding in time of the computation involved in its forward computation

An RNN is equivalent to a traditional neural network unfolded in time as depicted in Fig. 3. Given a sequence of k number of items (words), then the RNN would be unrolled into a $k-$layer neural network. The x_t is the input and the s_t is the hidden state, in time step t. As shown in eq. 1, s_t is computed

based on the current input and the state at the previous time step.

$$s_t = f(Ux_i + W_{t-1}) \tag{1}$$

It is worthwhile to mention that the network shares parameters W, U and V. In practice s_t functions as the "memory" of the sequence, since s_{t-1} was computed taking into account s_{t-2} and consequently all previous inputs and states. Typically the function f is non-linear (sigmoid, RELU etc.) and the output o_t is a soft selection from a probabilities distribution vector over the available dictionary:

$$o_t = softmax(Vs_t) \tag{2}$$

The RNN's training processes is gradient descent based, similarly to the traditional neural networks. The gradient calculation is made by using a special kind of backpropagation, the **Back Propagation Through Time (BPTT)** and cross-entropy as the loss function. The cost error is calculated for the entire sequence as the sum of errors at each time step.

Unfortunately, the standard RNN is not suitable for capturing long-term dependencies due to either the **exploding** or the **vanishing gradient problem** [20, 21, 22].

The exploding gradient problem was introduced in Bengio et al. (1994)[22]. It refers to the large increase in the norm of the gradient during training time. This situation can be caused by the exponential growth of long-term components of the network. The rescaling of the norms has been proposed to address this issue [23].

On the other hand, there are situations where the long-term components are heading exponentially fast to norm zero, preventing the model to understand relations between distant events in time [23]. The introduction of a regularization term that prefers parameters, the gradients of which neither explode or vanish was suggested to ameliorate the problem. Further remedies for the vanishing gradient are considered to be the proper weights initialization and the use of RELU as the non-linear activation function instead of sigmoid. The most popular solution though was provided by the invention of the **Long Sort-Term Memory** networks, typically referred to as LSTM [24, 25].

The main difference of the RNN and the LSTM is that the latter has a method to control the network's state, thus providing the ability to decide what information is useful to remember and what to forget. This feature is realized through three gates included in an LSTM block, **the forget, the input and the update gate**.

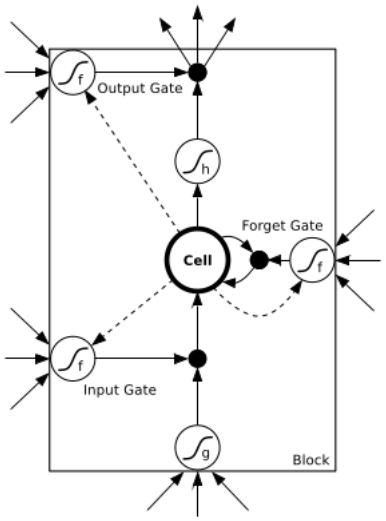


Figure 4: LSTM architecture

The forget gate is fed with the input x_t and the previous output h_{t-1} and through a non-linear (sigmoid) function produces (eq. 3) a number between zero and one. This number is the "percentage" of the previous state to keep while discarding the rest.

$$f_t = \sigma \cdot (W_f \cdot [h_{t-1}, x_t]) + b_f \tag{3}$$

Further, the cell has to decide about what new information should be stored in memory. This is a two-step process. First the model determines which 'memories' (memory blocks) should be replaced (eq. 4) and then decides what information should take their place (eq. 5).

$$i_t = \sigma \cdot (W_i \cdot [h_{t-1}, x_t]) + b_i \tag{4}$$

$$\tilde{C}_t = \tanh \cdot (W_C[h_{t-1}, x_t]) + b_C \tag{5}$$

The cell state then is updated (eq. 6), discarding information based on the forget gate and keeping information according to the input gate.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{6}$$

Finally, the output (eq. 7) is based on a filtered version of the computed cell state

$$h_t = \sigma \cdot (W_o[h_{t-1}, x_t] + b_o) * \tanh(C_t) \tag{7}$$

The **Gated Recurrent Unit (GRU)** [26, 27, 28, 29, 30] is a simpler version of LSTM, in which, basically the input and the forget gate have been incorporated into one single "update" gate and the cell state has been integrated with the hidden state. Note, that both the LSTM and GRU belong to the family of **gated RNNs**.

2.2 Encoder - Decoder architecture

In 2014, the idea of using recurrent networks to map sequences to other sequences emerged by two independent scientific research teams [26, 31]. The encoder-decoder or sequence-to-sequence architecture was initially used in machine translation where sequences of variable length needed to be mapped to other sequences, also of variable length. Its foundation lies in the ability of RNNs to model sequential dependencies. The main principle is the based on the idea of an intermediate **context vector** C which is "shared" among the two RNN modules of the model. The encoder converts the input sequence to context and the decoder based on the context produces the output sequence. The Encoder-Decoder architecture is depicted in Fig. 5.

The main advantage of this approach is that the input and output sequence length can be of different size. The main drawback of this paradigm is that, for computational reasons, the size of the context vector C must be rather limited [32]. As such, it does not possess the capacity to encode dependencies over long temporal horizons, which is critical for the success of a multitude of NLP tasks.

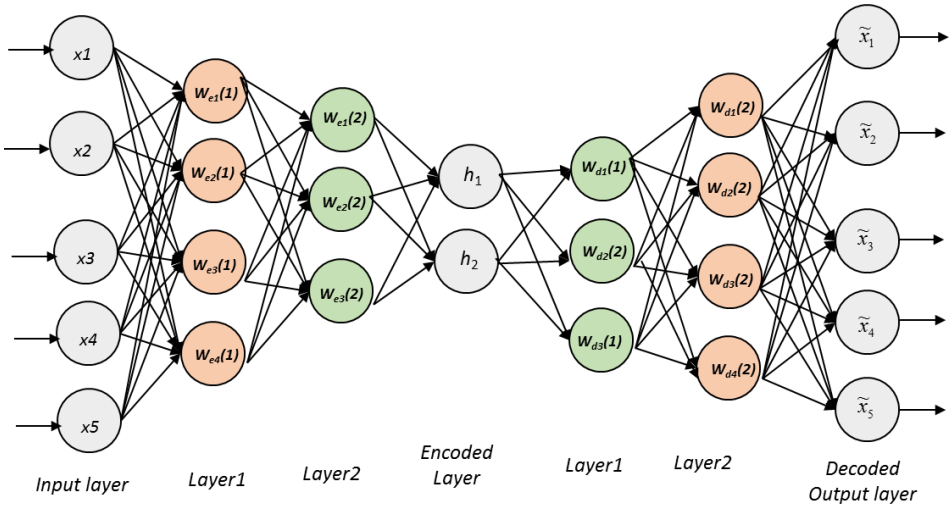


Figure 5: Encoder - Decoder architecture

2.3 Memory augmented networks

Memory is one of the main key components of NLP (Natural Language Processing) neural networks. As mentioned in the previous section, RNNs are not suitable for modelling long sequences. Although recurrent neural networks are endowed with memory, encoded by hidden states and weights, they suffer from two main issues; capacity and structure. Traditional neural networks, even more sophisticated ones, have limited size memory the structure of which, in some cases, tends to be unified, meaning that there is a lack of compartmentalization. These facts adversely affect their performance in tasks where long-term dependencies are important, such as NLP (Natural Language Processing).

In an effort to provide read and write memory functionality. several neural memory models have been developed since the early 1990s. The **neural network pushdown automaton** (Das et al., 1992) [33] provided memory push and pop actions, using a neural network state controller connected to a continuous external stack memory trained by gradient descent. Schmidhuber (1992) [34] proposed another gradient-based system dealing with temporal sequences, consisted by two feed-forward networks; one for producing, context dependent, weights changes and the second for storing these changes. The latter acted as a kind of a parameterized memory, able to quickly perform read and write operations.

Associative memory networks (Haykin, 1994) [35, 36] provided content-addressable memory, operation of which was driven from the transformation of a key vector to a value vector along with its references. The memory was unified, storing the weights of the model in the same location. In contrast, in memory-based learning models such as nearest neighbor, data were structured and stored into memory slots.

Memory Neural Networks (Weston et. al., 2014) [37], also referred as **MemNN**, combine long-term memory with inference modules achieving extraction of the most salient information given a context. The use of large memory implies expensive, time consuming computations. The search is accelerated by tricks such as hashing words or clustering word embeddings. In more detail, when hashing words is applied, memory is divided into slots, as many as the size of the dictionary. Then the input sentence is hashed into memory slots. A slot in the memory is taken into account during the search process only if it shares at least one word with the sentence. Alternatively, when clustering-based word embeddings are inferred, initialization is performed via k-means. In this case search queries only the buckets that share at least an exact word with the input sentence, along with its synonyms. The basic limitation of MemNN was the need of strong supervision during training, where along with the labeled data, supporting facts were also required. Even though MemNN scored very promising results in the experiments, it was very difficult to be used with real world data, due to this demerit.

Similar to MemNN, **Neural Turing Machine** (Graves et. al., 2014) [38], uses large, addressable memory with read and write functionality. It contains a neural network component (LSTM controller, sequential Input and delayed Output) and a differentiable memory bank. **NTM** is equipped with "attention" which allows the model to selectively search only a small portion of memory, based on contextual relevance. Attention has two complementary addressing mechanisms for producing network weights. The content-based mechanism which is based on content similarity; and the location-based mechanism where variables are addressed by location. The main focus of the model is on sorting, copying and recall of memory data. In this way, similar to a Turing Machine, NTM emulates a generic computation model that is able to learn any algorithm, by the data. In contrast to MemNN which has only a read head, NTM employs both read and write heads. Also, the addressing mechanism is trained in an unsupervised fashion, making it easier to be used with real world data. On the other hand, this unsupervised learning may become very complex and difficult, so NTM was largely used for relatively easy tasks. A strong point of NTM is that, since it can be trained to learn the algorithm behind the data, it is not application-specific, in contrast to the MemNN which was developed exclusively for Question and Answering tasks.

The **RNNSearch**, uses another kind of memory, also able to cope with long sequence representations. It was developed by Bahdanau et al. (2014) [32] and uses a model that automatically (soft-)searches words of the input sentence that are important for the prediction. It uses soft-alignment, meaning that the alignment mechanism is able to identify and build non-trivial, non-monotonic alignment between words, allowing the model to jump over words in memory in order to improve performance especially on long sentences.

The work of Grefenstette et. al. (2015) [39], coined **Neural DeQue** extended NTM, creating a new restricted class of models especially focused on language transduction tasks. It consists of a recurrent neural network (LSTM) enhanced by an unbounded differentiable memory. The memory incorporates a Neural Stack, Neural Queue and Neural DeQue that are used to produce the next state and output from input and previous state, with the help of a Controller.

At the same time, Sukhbaatar et. al. (2015) [40, 41] developed the **End-to-End Memory Network**, an extension of MemNN and RNNSearch, that is able to learn Q&A tasks with weak supervision and demonstrates the effects of a long-term memory integrated into neural network models for training end-to-end systems. The model is an encoder – decoder network enhanced with an external memory and an attention mechanism. It uses a feed forward network (RNN) as the Controller, and has a differentiable soft-attention Input, and Delayed Output as Interfaces. Even though the model is focused on Q&A tasks, its memory mechanism, which is content based, is rather generic and can be

used for any long-term dependencies modelling task. Memory has multiple layers (hops), so the model is able to handle multiple searches through time. It is implemented by an embedding function that captures similarities, enabling the model to be trained with weak supervision. Additionally, the lookup operation incorporates specific time features in order to maintain temporal order.

A multiple stacks memory was also used in **Stacked RNN**, published by A. Joulin & T. Mikolov (2015) [42] in an effort to overcome some of Deep Learning limitations by utilizing a model with a more complex structure. They showed that recurrent neural network's performance can be enhanced in difficult algorithm recognition tasks (Counting, Memorization and Binary Addition) by the increase of memory stacks in a structured way. The model consists of a Controller (RNN), and three interfaces, the sequential input, the differentiable memory stack and the sequential output.

Another extension of NTM was introduced in Reinforcement Learning Neural Turing Machine – Revised (W. Zaremba et. al., 2016) [43] where the model learns to interact with a discrete interface, such as database or search engine, instead of memory which is continuous and differentiable. Discrete interfaces cannot be trained through back-propagation, so Reinforcement Learning algorithm was used instead. The model has a Controller (LSTM) and three interfaces called Input Tape, the Memory Tape, and the Output Tape.

A new machine learning model called a **Differentiable Neural Computer (DNC)** was introduced by Graves et. al. (2016) [44]. DNC consists of a neural network equipped with an external memory. It has an LSTM as the Controller, and a Sequential Input, a differentiable Memory Queue, and the Sequential Output as Interfaces. It can perform read and write actions to the external memory matrix, which is used for handling complex data such as graphs, while the neural network (controller) learns how to do so. It is analogous to a conventional computer but instead of being programmed for each task, it can be trained. Information in memory contains the data as well as its associative temporal links. Both of these are used in memory search, in a bidirectional fashion, in order to recall the appropriate information. The Controller outputs are used to parameterize the distribution (weights) over the locations in memory. An introspective attention model is used for focusing on selective memory locations which consist of three separate attention mechanisms based on content, memory and temporal order. The Controller interpolates through these mechanisms using scalar gates.

Training of neural networks demands vast amount of examples, in contrast to humans who are able to rapidly adapt to new stimulations. Previously mentioned memory networks, such as NTM [38] or MemNN [40, 41], require large datasets in order to be able to model a credible output distribution function. Furthermore, their ability to adapt to new information is rather limited, hence increasing the need for iterative learning. Santoro et.al. (2016) [45, 46] suggested **Memory-Augmented Neural Networks (MANN)** to address these

issues, especially for cases where instant inference is required based on limited data that rapidly change. As the name implies, MANN consists of a neural network augmented with external memory. As opposed to the other networks, it is equipped with an additional controller coined Least Recently Used Access (LRUA). This is a special controller module designed to access memory and is focused on encoding relevant information. Specifically, it is designed to focus on memory content instead of memory locations. Depending on the relevance between input data and information stored in memory, LRUA functions as a memory writer which, depending on content, decides where to store incoming data. Irrelevant data are stored to rarely used locations, in contrast to relevant data, which are placed to the last used location, as a kind of memory update [45, 46].

2.4 Memory networks with attention mechanism

The invention of the notion of neural attention, along with the development of external differentiable memory modules, could be considered as major breakthroughs in Deep Learning. As already mentioned in the previous section, many state-of-the-art NLP models were implemented using this kind of scheme (external memory with attention). Likewise that many memory variations have been developed to serve specific applications, there are several attention mechanisms proposed in the past few years.

The idea of an attention mechanism has been originally studied in Neuroscience and Computational Neuroscience [47, 48]. The fact that humans, as well as animals, can focus on specific parts of their sight, as well as to specific types of past stored memories, helped scientists to realize that there must be a visual attention mechanism in the process. This idea then was inherited to neural networks, because it was the right fit to the memory extension. The external memory is very large, compared to the neural network internal memory, and contains lots of information, only a part of which is relevant to the neural response at a given time point. The attention mechanism is able to select the most salient information from the memory, making the network significantly more efficient. Attention-based neural networks have excelled in neural machine translation [32, 49], image captioning [50], speech recognition [51, 52], question answering [41, 40, 53], and algorithm-learning [37, 54].

Attention is implemented as a hidden layer which computes a distribution to make a selection over source elements, as depicted in Fig. 6.

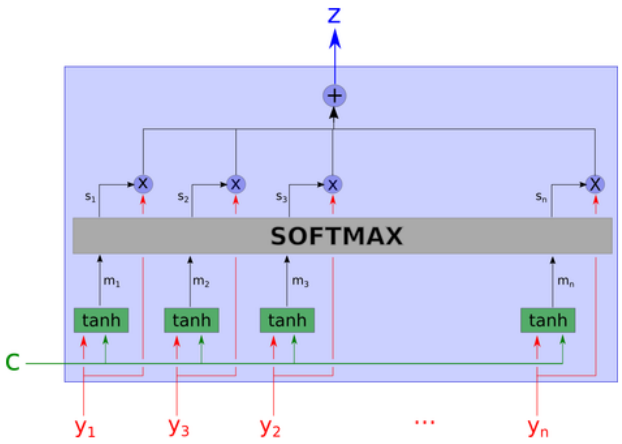
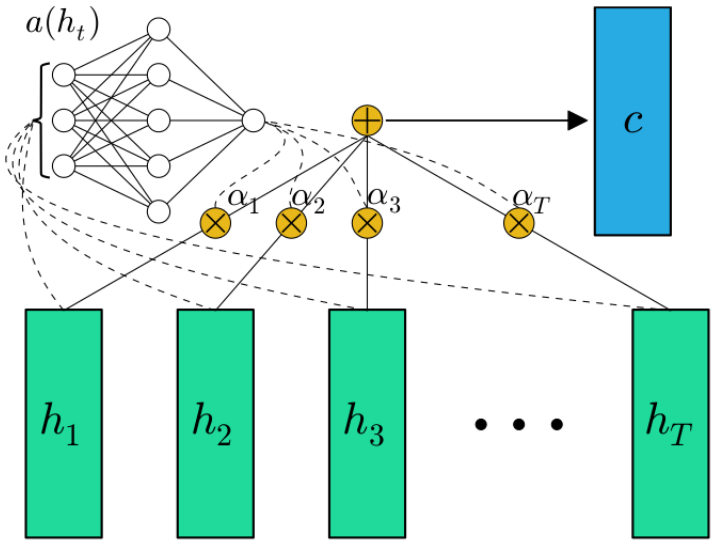


Figure 6: Soft Attention mechanism

Given that an attention mechanism takes a set of facts and the context as input, the output (of the attention) is a shorter representation of the input, containing only important information. This mechanism consists of three layers. The first layer employs a non linear function, usually tanh or dot product, which combines each individual element of the input with the context. The second layer is a softmax layer. The obtained softmax values are probabilities proportionate to the relevance of each fact to the context. Finally, the output is the weighted mean of the values, where weights represent importance. This is a deterministic process and since all layers are differentiable, the mechanism can be directly connected to any neural network that affords gradient-based training. This type of Attention, which is one of the most popular, is known as **Soft Attention** [50]. In the same work, a **Hard Attention** mechanism is also proposed. Therein, the softmax layer is replaced by "hard max" that selects only one memory position, as opposed to producing probabilities of the model attending to each one of the available memory positions.



Vectors h_1,h_2,...,h_t represent hidden states through time. These are passed to a function $\alpha(h_t)$ to derive the probability vector α. The output vector c is the weighted average of h_t, where weights express how relevant each h is to α.

Figure 7: Feed-forward attention mechanism.

Let us consider the feed-forward attention network, depicted in Fig. 7. Therein, the vectors h_1,h_2,...,h_t represent the hidden states obtained through time by an RNN encoder module that is presented with the input data. These are passed to a function $\alpha(h_t)$ to derive the probability vector α. The output vector c is the weighted average of h_t, where weights express how relevant each h is to α. At each time step a "context" vector is computed as the weighted mean of the state sequence, providing a closer correlation between model hidden states through time (context). The attention mechanism reduces the need for remembering the entire input sequence and allows the decoder to focus on different parts of the input at different time steps. Inspired by Bahdanau et al. (2014) [32], feed-forward attention [55, 56] was developed as a simplification to the attention mechanism. A single context vector is produced for the entire sequence instead of the multiple vectors, one for each individual time step.

In [49] (Luong et. al., 2015) attention mechanisms are classified into two major categories, distinguished by the portion of the source that is being used. **Global Attention** mechanisms infer the weight vector based on the current state and the entire set of source states. Then, the model computes the context vector (global) as a weighted average over all source states. The fact that all states of the sequence are fed to the attention mechanism, requires expensive computations for longer sequences, hence introducing scalability issues. In

contrast, **Local attention mechanisms** limit computation of the attention vectors to a limited subset of source states (window), around a selected (predicted) position. Since only a part of the entire context is taken into account, it is clear that the computation expensiveness is diminished. Additionally, the subset of the source states is also differentiable; this makes it easy to be directly attached to a gradient-based network, in contrast to hard attention [50].

W. Wang et. al. (2017) [57, 58] experimented with a **Gated Attention**, a novel attention mechanism enhanced by an additional gate. In a question answering model, a non-linear (sigmoid) gate is added to the input of an RNN, responsible for identifying the most important set of facts (passage parts) according to the question. Contrary to LSTM and GRU, this gate is based on current word (of the examined fact) and the attention vector, which is able to focus on the most pertinent parts of the question and the current word. Even though knowledge of the context is of paramount importance, only a small portion is being used. To address this issue, authors proposed the Self-Matching Attention, that directly matches the representation of passage-question pair against itself. This way, the gate is presented with the representation of the context derived based on the current word, and the question is able to focus on the most important information, even over very long sequences.

Structured Attention Networks [59] offer a new perspective especially in cases of complex synthetic tasks where serious structural dependencies may exist. They have been introduced as a generalization of categorical soft attention mechanisms. Soft attention layers are not designed to model structural properties. In this case structure has to be learned through heavy training, where the size of the dataset plays a key role. The importance of modelling structural dependencies was demonstrated in many publications [60, 61, 62, 63, 64, 65, 66, 67]. In a sense, the idea of structured attention is analogous to the work of Bahdnau et. al., (2014) [32]. In the same way that the network learns how to align words in a translation task, structured attention is able to learn how to segment sentences or parse a tree from the input. It is specified as a graphical model over multiple latent variables, the edges of which hold information about the structure of the data. Attention is computed by inferring the expectation of the context vector over a set of structures. The whole process is differentiable, making able to train the model in an end-to-end fashion, through gradient-based algorithms.

Another model that exploits the merits of graphs was proposed by Tan et. al. (2017) [68] as an alternative for summarization tasks. **Graph-Based Attention** mechanism is explicitly designed for tasks where mapping the input to output is conceptually difficult. A Graph is constructed for sentence ranking, where the Vertices of the Graph are the sentences and Edges represent the relationship (similarity) between them. The saliency score is calculated based on the relation of the current sentence hidden state against the hidden state of all the other sentences. The attention model, additionally, incorporates a distrac-

tion mechanism initially presented by Chen et. al. (2016) [69], which penalizes sentences that have already attracted focus. Attention is finally determined based on the saliency scores of the sentences and expresses the most globally salient sentences which are in the same time important during decoding the specific hidden state. The graph-based attention function is fully differentiable.

Similar to Graph-Based Attention, **Hierarchical Attention** mechanisms have also been designed as extensions of attention for traditional summarization models. In a Hierarchical Document model, a document is regarded as a composition of words, sentences, paragraphs and even larger units. Nallapati et. al. (2016) [70] introduced a hierarchical encoder with hierarchical attention. The model tries to encode information importance in sentence as well as in word level. The weight context vector is computed in a two-step process. Initially, the word-level attention is calculated and then it is re-scaled based on the sentence-level attention probabilities. Alternatively, in [71], during the extractive summarization process, each sentence and all the document words are attended by the decoder through a softmax layer, in each time step, in order for the model to derive the probability of a word to be found next in the summary.

A. F. Martins and R. F. Astudillo (2016) [72] observed that the traditional softmax layer is designed to produce dense weight representations, meaning that each part of the input has a proportionate contribution to the neural response. In an effort to create an activation layer that would be able to tackle sparse and dense, they developed a **sparsemax** transformation. By using Euclidean projections onto the simplex, this transformation can produce sparse posterior distributions, in which some of the output variables are assigned with zero values. Sparsemax can be leveraged to developed a new selective attention mechanism, that has showed very promising results in multi-label classification and natural language inference tasks [72]. Following sparsemax, V. Niculae and M. Blondel (2017) [73], devised two new attention mechanisms, coined **fusedmax** and **oscarmax**; both address issues emanating from sparse representations, as well as the need to better model hierarchical structures. Fusedmax, based on fused LASSO [74], instructs the neural network to focus on contiguous segments of the input in order to produce the output,. Oscarmax utilizes OSCAR [75] penalties to enable paying attention to non-contiguous groups of items (words) of the input.

Lately, in 2018, L. Hou et. al. [76], suggested that a major issue in abstractive neural summarization is the repetition of phrases or words in the generated summary. To address this issue, they developed a **Joint-Attention mechanism**. Specifically, the network postulates an integration of two separate attention mechanisms, found in the input (encoder) and the output (decoder) of the network. The first one employs a dynamically changing context vector, which is able to focus on the most important information of the input according to each time step. The latter is responsible of reducing the repetition of words or phrases. In contrast to other attentional functions, the output of which depends

only on a few of the last decoder steps, this implementation combines previous decoder outputs with the last decoder state to infer the context by a three-layer process (non-linear, softmax, aggregation).

3 QUESTION – ANSWERING

One long-term goal of machine learning research is to produce methods that are applicable to reasoning and natural language, in particular building an intelligent dialogue agent. This agent has to be able to comprehend natural language and reason data in order to answer a question. Recent advances in deep learning have brought to the fore models that can make multiple computational steps in the service of completing a task; these are capable of describing long-term dependencies in sequential data. Novel recurrent attention models over possibly large external memory modules constitute the core mechanisms that enable these capabilities.

The existing theory of MEM-NN, first introduced in [37] is considered to be the cornerstone for many state-of-the-art models in Q&A tasks. The Memory Network, as already described, was difficult to be trained via back-propagation and had a strong need for supervision. This demerit was alleviated in a recent end-to-end-trainable extension of MEM-NN, presented in [41], which can also be thought as an extension of [32]. That variant enjoys the advantage of requiring much less supervision during training, which is of major importance in real-world question-answering scenarios. Additionally, it has the ability to perform multiple computational steps before responding to queries, enabling it to better model complex synthetic tasks.

The model input comprises a set of *facts,* $\{\boldsymbol{x}_i\}_{i=1}^N$*,* that are to be stored in the memory, as well as a *query* $\boldsymbol{q}$; given these, the model outputs an *answer* a. Each of the facts*,* $\boldsymbol{x}_i$*,* as well as the query, $\boldsymbol{q}$, contain symbols coming from a dictionary with V words. Specifically, they are represented by vectors that are computed by concatenating the one-hot representations of the words they contain. The latter are obtained on the basis of the available dictionary comprising V words. The model writes all $\boldsymbol{x}_i$ to the memory, up to a fixed buffer size, and then finds a continuous variable encoding for both the $\boldsymbol{x}_i$ and the $\boldsymbol{q}$. These continuous representations are then processed via multiple hops, so as to generate the output a; this essentially constitutes one (selected) symbol from the available dictionary. This modelling scheme allows for establishing a potent training procedure, which can perform multiple memory accesses back to the input.

Specifically, let us consider one layer of the MEM-NN model. It comprises three main functional components:

Input memory representation: Let us consider an input set of facts*,* $\{\boldsymbol{x}_i\}_{i=1}^N$*,* to be stored in memory. This entire set is converted into *memory vectors*, $\{\boldsymbol{m}_i\}_{i=1}^N$*,* $\boldsymbol{m}_i \in \mathbb{R}^\delta$*,* computed by embedding each $\boldsymbol{x}_i$ in a continuous space, using a linear embedding matrix $\boldsymbol{A}$; that is, $\boldsymbol{m}_i = \boldsymbol{A}\boldsymbol{x}_i$. The query $\boldsymbol{q}$ is also embedded in the same space; this is performed via a distinct embedding matrix $\boldsymbol{B}$, and yields an *internal state vector* $\boldsymbol{u}$; that is, $\boldsymbol{u} = \boldsymbol{B}\boldsymbol{q}$. On this basis,

MEM-NN proceeds to compute the match between the submitted query, $\boldsymbol{q}$, and each one of the available facts, by exploiting the salient information contained in their inferred embeddings; that is, the state vector, $\boldsymbol{u}$, and the memory vectors, $\{\boldsymbol{m}_i\}_{i=1}^N$, respectively. Specifically, it simply takes their inner product followed by a softmax:

$$\varpi_i = \text{softmax}(\boldsymbol{u}^T \boldsymbol{m}_i) \tag{8}$$

where

$$\text{softmax}(\boldsymbol{\zeta}_i) \triangleq \frac{\exp(\boldsymbol{\zeta}_i)}{\sum_j \exp(\boldsymbol{\zeta}_j)} \tag{9}$$

In essence, $\boldsymbol{\varpi} = [\varpi_i]_{i=1}^N$, is a probability vector over the facts, which shows how strong their affinity is with the submitted query. This vector is the inferred *attention* vector.

Output memory representation: In addition to the inferred memory vectors, MEM-NN also extracts from each fact, $\boldsymbol{x}_i$, a corresponding *output vector embedding,* $\boldsymbol{c}_i$, via another embedding matrix $\boldsymbol{C}$, i.e. $\boldsymbol{c}_i = \boldsymbol{C}\boldsymbol{x}_i$. These output vector embeddings are considered to encode the salient information included in the presented facts that can be used for output (answer) generation. To achieve this goal, we leverage the inferred attention vector $\boldsymbol{\varpi}$, by using it to weight each fact (encoded via its inferred output vector embedding) with the corresponding computed probability value. It holds

$$\boldsymbol{o} = \sum_{i=1}^{N} \varpi_i \boldsymbol{c}_i \tag{10}$$

Generating the final prediction: MEM-NN output layer is a simple softmax layer, which is presented with the computed output vector, $\boldsymbol{o}$, as well as the internal state vector, $\boldsymbol{u}$. It estimates a probability vector over all possible predictions, $\hat{a}$, that is all the entries of the considered dictionary of size V. It holds

$$\hat{a} = softmax(\boldsymbol{W}(o + u)) \tag{11}$$

where $\boldsymbol{W}$ is the weights matrix of the output layer of the network, whereby we postulate

$$a = \arg\max(\hat{a})$$

Multiple hops: The model is capable to perform multiple hops in memory by simply stacking multiple such layers[2]. In case of $\boldsymbol{k}$ hops, each layer, except for the first one, takes as input the sum of the output o^k and the input u^k from layer k:

[2]The number of hops performed in memory is a model hyperparameter, that has to be selected in a heuristic manner. Naturally, there is no point in this number exceeding the number of facts presented to the model each time.

$$u^{k+1} = u^k + o^k$$

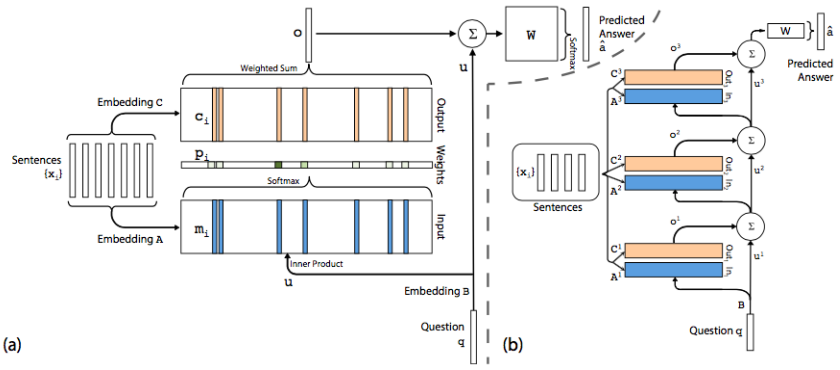
The weights matrix $\boldsymbol{W}$ is fed with a combination of the top memory layer input and output representations

$$\hat{\boldsymbol{\alpha}} = softmax(\boldsymbol{W} u^{k+1}) = softmax(\boldsymbol{W}(o^k + u^k))$$

There can be two types of weight tying within the model:

1. **Adjacent:** The input embedding of each layer is the output embedding of the previous one (below in memory), meaning that $A^{k+1} = C^k$. Two constraints are being applied. The answer prediction matrix is constrained to the size of final output embedding ($W^T = C^k$) and the question embedding is matched to the input embedding of the first layer ($B = A^1$).

2. **Layer-wise:** The input and output embeddings are the same across different layers $A^1 = A^2 = ... = A^k$ and $C^1 = C^2 = ... = C^k$. A graphical illustration of the considered end–to-end trainable MEM-NN model is provided in Fig. 8.



(a) A single-layer version of the considered model. (b) A 3-layer version, obtained via stacking.

Figure 8: End-to-End Memory Network

Data encoding: Three types of encoding are proposed. The traditional **bag-of-words representation (BoW)**, where a sentence embedding is just the sum of all word embeddings

$$m_i = \Sigma_j A x_{i,j}$$
$$c_i = \Sigma_j C x_{i,j}$$
$$u_i = \Sigma_j B q_{i,j}$$

Due to the fact that in some tasks the order of words within the sentence is important, a second method, coined **Position Encoding (PE)**, is proposed. Under this rationale, the expressions become

$$m_i = \Sigma_j l_j \cdot A x_{i,j}$$
$$c_i = \Sigma_j l_j . C x_{i,j}$$
$$u_i = \Sigma_j l_j \cdot B q_{i,j}$$

where l_j is a column vector

$$l_{kj} = (1 - \frac{j}{J}) - (\frac{k}{d})(1 - \frac{2j}{J})$$

and $\cdot$ is element-wise multiplication. The amount of words in the sentence are represented by J, while d is the embedding size. Under PE the embeddings are affected by the position of each word in a sentence.

In order to give a temporal notion to the embeddings, **Temporal Encoding (TE)** is introduced. In some tasks, understanding the order of events (sentences) in a story is mandatory. For this reason, memory and output vector are altered to

$$m_i = \Sigma_j A x_{i,j} + T_A(i)$$

$$c_i = \Sigma_j C x_{i,j} + T_C(i)$$

where T_A, T_C are special vectors, that are learned during training and hold temporal information. In order for the relative distance between the question and each sentence to be reflected, sentences are indexed in reverse order.

Regularization: The authors suggested a new technique that improved the performance of the model, referred as Random Noise (RN). It involves the injection random noise to the input (stories). Specifically, during training time, 10% of empty memories are added to stories, a fact that regularizes the, previous described, temporal vector T_A.

Training - Evaluation: During training, all three embedding matrices A, B and C, as well as W are jointly learned by minimizing a standard cross-entropy loss between the predicted value $\hat{\boldsymbol{\alpha}}$ and the true label $\boldsymbol{\alpha}$, using stochastic gradient descent algorithm (SGD) and Glorot initialization.

The model is trained on the bAbI dataset developed by J. Weston et. al. (2015) [77], which comprises 20 proxy tasks that evaluate reading comprehension via question answering. Tasks are independent from each other and aim to individually examine one aspect of intended behaviour. For each task, a training set is provided that contains the facts, the questions and the right answers as well as

the supporting facts used to derive the answer. In case of MEM-NN, supporting facts are ignored, reducing the need for strong supervision. The data represent common activities between actors and objects that move and interact between each other. The answers are limited to one word or a specific list of words, so evaluation is clear and easy. A separate training set is provided along with a test set for each different task.

In **Task 1 (Single Supporting Fact)**, the answer can be simply inferred by locating the correct fact from the input data (set of facts), in contrast to **Task 2 (Two Supporting Facts)** and **Task 3 (Three Supporting Facts)** where multiple facts must be chained, two and three respectively, in order for the question to be answered. **Task 4 (Two Argument Relations)** and **Task 5 (Three Argument Relations)** were designed to evaluate the model's ability to distinguish objects from subjects. Task 4 includes cases where the answer is determined by the word's order while stories in Task 5 contain interaction between actors and objects.

Task 6 (Yes/No) constitutes the simplest form of true/false questions. The counting ability is tested via **Task 7 (Counting)**, where the model needs to count the multitude of an object held by an actor, disregarding irrelevant information. In a similar way, in **Task 8 (Lists/Sets)** the answer contains a list of different items held by an actor. A more complex natural language construct is explored in **Task 9 (Simple Negation)**, where some facts have negative meaning, hence challenging the model to understand that the statement is false. Another complex task is examined in **Task 10 (Indefinite Knowledge)** in which the answer cannot be defined by the facts that, in many cases, are phrases connected by a disjunctive "or". A sample of statements and questions from Tasks 1 to 10 are depicted in Fig. 9.

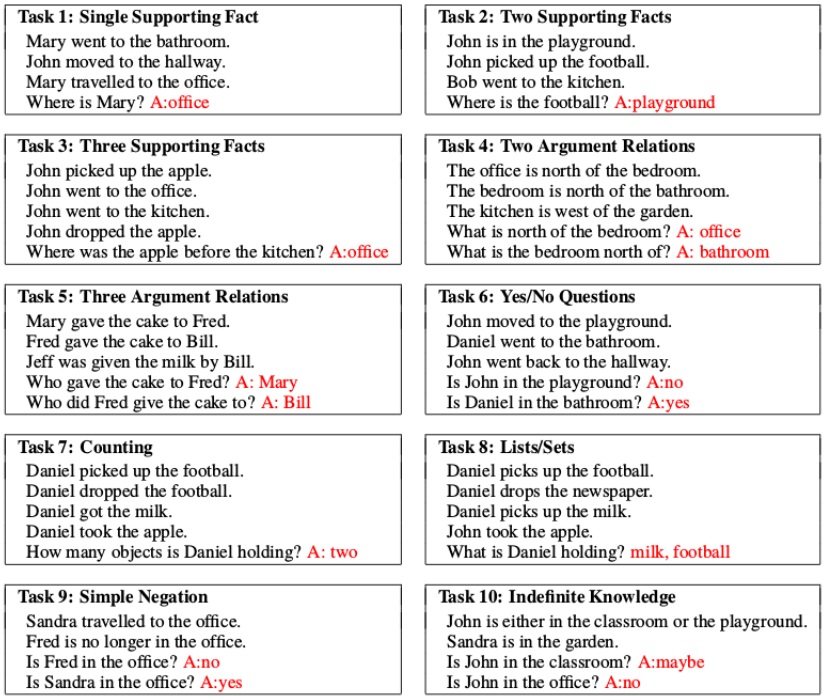


Figure 9: Sample statements and questions from Tasks 1 to 10

As the name implies, **Task 11 (Basic Coreference)** tests the ability to locate the latest reference of an actor in order to pinpoint her current location. **Task 12 (Conjunction)** is very similar to the previous task, and differentiates only on the fact that many actors are included in the same sentence. **Task 13 (Compound Coreference)** also tests the coreference, but ii this case a pronoun can refer to multiple actors. The time comprehension is evaluated through **Task 14 (Time Reasoning)**, where facts contain time expressions, in contrast to all previous tasks that time sequence was corresponding to the order of sentences.

Deduction and induction are tested in **Task 15 (Basic Deduction)** and **Task 16 (Basic Induction)**, respectively, through inheritance of properties. Spatial reasoning is examined in Task **17 (Positional Reasoning)**, where the relative position of a colored block is being asked. The ability of the model to understand relations between object sizes is examined in **Task 18 (Size Reasoning)**, where it is asked to determine whether an object can fit in another, based on the provided facts. **Task 19 (Path Finding)**, challenges the model to find the directions in order to move from one location to another, given a description of the relation between locations. Finally, the **Task 20 (Agent's Motivation)**, examines the ability of the model to learn the reason behind the

actions of an actor. A sample of statements and questions from Tasks 11 to 20 are depicted in Fig. 10.

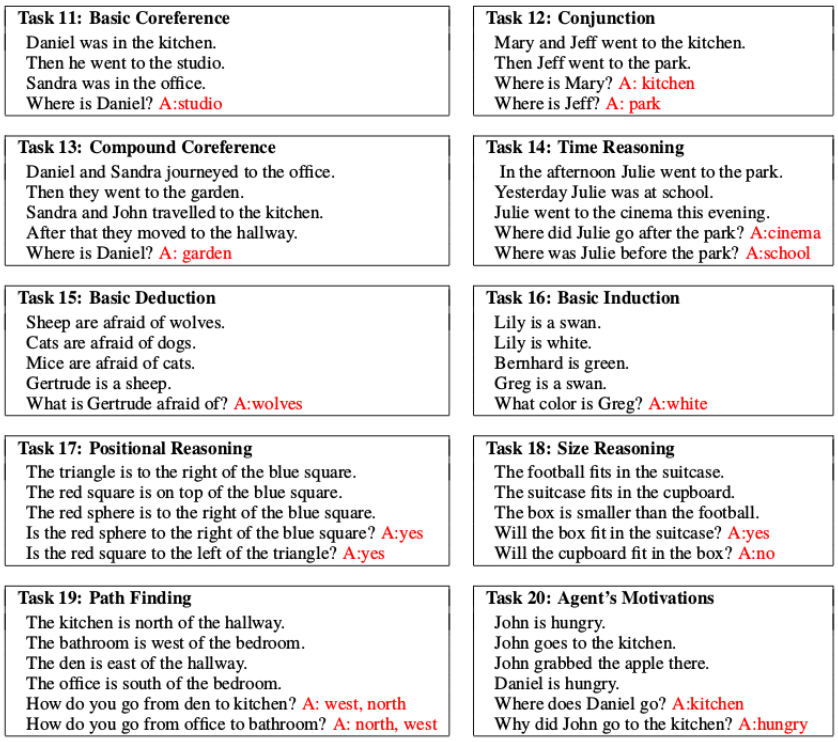


Figure 10: Sample statements and questions from Tasks 11 to 20

The experimental evaluation of the model showed that some tasks (1, 12, 20) are relatively easy to be learned, while some others (17, 19) are much harder. Furthermore, on some tasks (e.g. 15) a large variance of the performance was detected, pointing that the model was highly depending on the initialization. Finally, there are considerations about the model's behaviour in large and sparse data as those found in real-data applications.

4 TEXT SUMMARIZATION

The revolution brought by Internet and World Wide Web enabled humans to access, in a daily basis, an enormous amount of information, of any kind. It is in fact impossible to make use of all these information, given the fact that reading, extracting the most interesting parts and understanding the concept hidden behind the data, is a very demanding and time consuming process. Over the years, it became clear that some kind of a summarization tool would be necessary to serve these needs. Summarization is the process of creating a shorter version of the original text, preserving the most salient information, without altering the basic concepts.

Due to the fact that the amount of information is vast, the data are most likely unstructured and the diversity of natural language is large, the required cognitive load for reading and understanding documents is colossal. For this reason, AI text summarization is an open field, still posing several challenges.

There are two approaches to text summarization; the **extractive**, in which sentences or words found in the original text are used to construct the summary and the **abstractive**, which includes the generation of new sentences or novel words during the summarization process. Initially, researchers were attracted [78, 79, 80, 81] by the extractive method, since it was much easier. With the introduction of the sequence-to-sequence architecture [31] the abstractive method was made more feasible and attractive. Currently, state-of-art abstractive models [82, 83, 84, 85, 86] are performing equally well to extractive ones [87], and alongside are able to produce more human-like results.

The encoder-decoder scheme, with the use of recurrent neural networks (RNNs), enabled models to read and generate text, through word alignment. It was initially used for machine translation [32, 49] and then the idea was inherited to text summarization. Although the performance of the earlier abstractive models [82, 83, 84, 85] was very promising, they showed difficulty in rendering facts, they had problem dealing with out-of-vocabulary words (OOV) and suffered from word or phrase repetition.

In 2017, Manning et. al. [86] developed a **hybrid point-generator** model in an effort to address these issues and achieved state-of-the-art performance. It is a hybrid network employing a **pointer-generation model** along with a **coverage mechanism**. The point-generation model is a sophisticated amalgam of a *sequence-to-sequence* attentional model and a *pointer network*.

Sequence-to-sequence model is based on the work of Nallapati et. al. (2016) [88] and comprises an encoder, which is a single-layer bidirectional LSTM, a decoder, which is a single-layer unidirectional LSTM and an attention mechanism adopted by Bahdanau et. al. (2014) [32]. The main intuition behind the function of this module, is that the model initially focuses on the most salient

words of the source according to the input word, the previous word and the encoder-decoder states. Then, the context vector, that holds information about what has been already read by the network, is computed based on the attention information (distribution) and the encoder state. Finally the context vector along with the decoder state is used to compute the probability distribution over all the words of the vocabulary, which is the final distribution, from which, words are predicted.

More specifically, at each time step t, the encoder produces the hidden state h_i according to the input words of sentence w_i, that are fed sequentially to it. The decoder has state s_t and takes as input the embedding of the previous word. These are used by the attention mechanism to compute the attention distribution.

$$e_t^i = v^T \tanh(W_h h_i + W_s s_t + b_{attn}) \tag{12}$$

$$a^t = softmax(e^t) \tag{13}$$

where v, W_h , W_s and b_{attn} are the weights and bias of the model.

Following the context vector h^* is calculated based on the encoder state and the attention distribution

$$h^* = \sum a^t h_i \tag{14}$$

The context vector combined with the decoder state are taken into account to produce the vocabulary distribution P_{vocab}

$$P_{vocab} = softmax(V^{'}(V[s_t, h_t^*] + b) + b^{'}) \tag{15}$$

where V , $V^{'}$, b and $b^{'}$ are learnable parameters.

$$P(w) = P_{vocab}(w) \tag{16}$$

The **pointer network** was originally developed by Vinyals et. al. (2015) [54]. A problem observed by the authors was that traditional sequence-to-sequence models require the size of the output dictionary to be fixed and pre-defined. They proposed a new variation of the attention mechanism [32] that is purposed to create pointers to input items, so to handle problems where the output dictionary was dependent on the input sequence length.

Given a training pair, (P, C^P), the encoder hidden states $(e_1, ..., e_n)$ and the decoder hidden states $(d_1, ..., d_{m(P)})$ the pointer network computes the conditional probability $p(C^P|P;\theta)$ using the attention vector, that is essentially a pointer to input elements.

$$u_j^i = v^T \tanh(W_1 e_j + W_2 d_i), j\epsilon(1, ..., n)$$

$$p(C_i, ..., C_{i-1}, P) = softfmax(u^i)$$

where u^i is the output distribution over the dictionary of inputs and v, W_1, W_2 are learnable parameters of the output. As mentioned the point-generator is a combination of the sequence-to-sequence attentional model and the pointer-network. The generation of words is made by either copying from the input using pointers as in pointer-network or by selecting from a fixed dictionary, as in encoder-decoder architecture. Additionally to the attention distribution $a^t$and the context vector h^*, the generation probability is computed.

$$p_{gen} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr})$$

where x_t is the decoder input, s_t the decoder state b_{ptr}, w_h, w_s and w_x are trainable parameters. The generation probability P_{gen} functions as a alternator that select between copying and generating words.

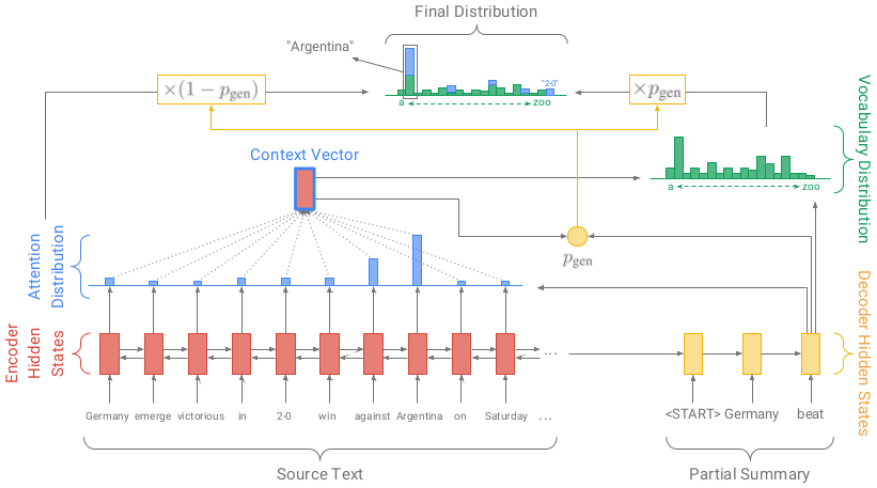


Figure 11: Pointer generator model

More specifically, when the model selects to copy a word from the input sequence, it samples the attention distribution a^t while in order to generate a new word found in the global vocabulary it samples the global vocabulary distribution P_{vocab}. The global vocabulary is the concatenation of the vocabulary with all the words of the current input.

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \Sigma_{i:w_i=w} a^t \tag{17}$$

Additionally, the eq. 17 can be thought as a trick to deal with out of vocabulary (OOV) words. When an OOV word appears, the model selects to copy a word from the input, since $P_{vocab}(w) = 0$.

Training: The SGD algorithm with Glorot initialization is used for model's training. The loss function at each timestep and the overall loss during training are defined, respectively:

$$loss_t = -logP(w_t^*)$$

$$loss = \frac{1}{T}\sum_{t=0}^{T} loss_t$$

In order to reduce word repetition, another trick was utilized. A **coverage mechanism** was adopted from the work of Tu et. al. (2016) [89] to penalize words that have already taken attention. A *coverage vector* that acts as global attention memory was added to the network. In details, the coverage vector c^t is an aggregation of all passed attention distributions and is defined as:

$$c^t = \sum_{t'=0}^{t-1} a^{t'}$$

Accordingly, the encoder state was altered to take into account the coverage vector (attention memory)

$$e_t^i = v^T \tanh(W_h h_i + W_s s_t + w_c c_i^t + b_{attn}) \tag{18}$$

Experimentation showed that another term would be useful for further eliminating repetitive words. The loss changed to include a penalization of overlapping between attention distributions and the global attention memory

$$loss_t = -logP(w_t^*) + \lambda \sum_i min(a_i^t, c_i^t) \tag{19}$$

where λ is a hyperparameter and $\sum_i min(a_i^t, c_i^t)$ is the **coverage loss**.

Summary generation: ***Beam search*** is employed for realizing the summary generation. Finding the (most likely) output sequence involves searching through all the possible output sequences based on their likelihood, as defined by $P(w)$. The size of the vocabulary consists of thousands or even millions of words. Given the fact that search complexity is exponential to the length of the output sequence, the calculation can be easily become intractable. Therefore, an approximation method as the beam search is necessary.

It is a heuristic, path-based, search method [90] that instead of calculating only the next step, it expands all k next steps, where k is the *beam size,* and keeps the k most probable candidates for each step, while rejecting and pruning

all other values. Subsequently, the next k steps are calculated, but only for the k candidate values from the previous step. The process is iterated until the special "EOS" (End Of Sentence) token is selected, or the maximum sequence length is reached. The beam size can be thought as the number of parallel searches in the sequences of probabilities, where useful information is exchanged between these parallel searches.

Experiments - evaluation: The model was tested using the *CNN/Daily Mail dataset* [91]. The dataset is a collection of about 300k articles and their summaries from the CNN (April 2007 - April 2015) and the Daily Mail (June 2010 - April 2015) web site. The average length of articles and summaries is 781 and 56 tokens respectively. Detailed corpus statistics are displayed in Fig. 12.

	CNN			**Daily Mail**		
	train	valid	test	train	valid	test
# months	95	1	1	56	1	1
# documents	90,266	1,220	1,093	196,961	12,148	10,397
# queries	380,298	3,924	3,198	879,450	64,835	53,182
Max # entities	527	187	396	371	232	245
Avg # entities	26.4	26.5	24.5	26.5	25.5	26.0
Avg # tokens	762	763	716	813	774	780
Vocab size	118,497			208,045		

Figure 12: CNN - Daily Mail corpus statistics

The results were evaluated using two different metrics; the **ROUGE** (ROUGE-1, ROUGE-2 and ROUGE-L) [92] and **METEOR** (exact match and full mode) [93]. Based on the former metric, the Pointer Generator with Coverage outperforms all the state-of-the-art abstractive model [88] while performing equally to the top performing extractive model [87]. Considering the latter metric, the model also presents the best performance among abstractive models, but lacks in accuracy comparing to the extractive model (about 3 METEOR points).

A sample summary is illustrated in Fig. 13. It can been observed that the Point-Gen model is able to accurately copy factual data from the original text and produce a comprehensive summary. Furthermore, the model shows excellent behaviour when dealing with OOV words. Nevertheless, it suffers from repetitions, fact that is ameliorated in Point-Gen with Coverage. Still, both models couldn't produce many novel words, following the safe path of copying. Hence, this approach can be considered to be in between of abstractive and extractive method.

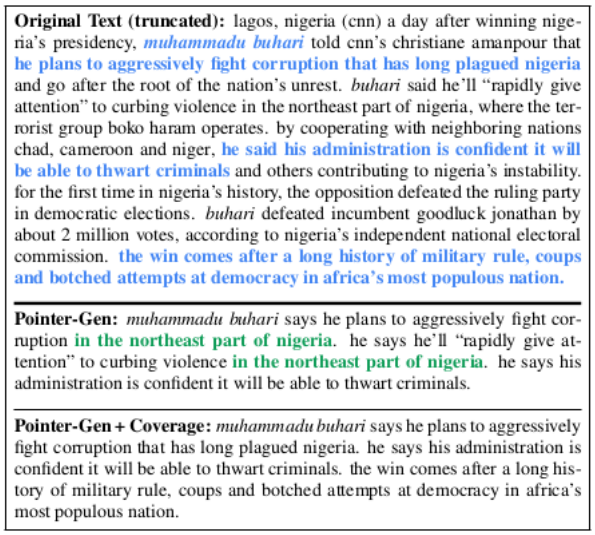


Figure 13: Pointer generator and Pointer generator with coverage sample summary

5 BAYESIAN INFERENCE

5.1 Frequentist vs Bayesian

In the previous chapters the state-of-the-art DNNs (Deep Neural Networks) for Natural Language Processing (NLP) tasks, especially for Q&A tasks and Text Summarization, were studied. These solutions, as described, aim to predict point-estimates, based on the maximum likelihood paradigm, for the parameters θ of the model. Then all predictions are made based on this single value of θ. In essence, the true parameter value θ is treated as fixed but unknown and the point estimate $\hat{\theta}$ as a random variable that is a function of the dataset. This also known as the **frequentist** approach.

The key concept behind this school of thought is that only repeatable random events have probabilities, which (probabilities) are connected with the frequency of occurrence of an event observed via sampling. It should be pointed that probabilities are depended on the amount of data that needs to be large and, additionally, should include repetitive events. It is impossible to predict the probability of an event occurring when this event hasn't been observed before. Furthermore, there is no provision for the incorporation of prior knowledge to the prediction.

These characteristics are major drawbacks of this approach, that make it unsuitable for problems where predictions have to made based on limited amount of data or in cases that prior belief may exist. Furthermore, it is not possible to assign probabilities to an event that hasn't been occurred before. All the aforementioned are common cases in supervised learning.

According to the **Bayesian** approach, probabilities represent uncertainty or the degree of belief about a given event. Probabilities reflect knowledge and experience, so it is possible to assign probabilities to any event, even to unseen, rare or non repetitive ones. Additionally, belief can be updated at any time making the prediction process more accurate and flexible. Opposite to the frequentist approach, the Bayesian approach predicts a full probability distribution instead of point-estimates for the postulated model parameters.

5.2 Bayes essentials

In 1763, Reverend T. Bayes in his essay [94] provided the mathematical foundation for updating belief based on new evidences. He defined that

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{20}$$

where A and B are events and $P(B) \neq 0$

Unlike to frequentist paradigm, the Bayesian inference approach considers that the dataset is not random, since it can be directly observed. On the other hand, the true parameter θ is unknown or uncertain, hence θ is a random variable.

Bayesian inference refers to the process in which prior evidence and observations are used to infer the posterior probability $p(\theta|x)$ of the random variables θ given the observations x. The prior knowledge about parameter θ is represented by the prior probabilities distribution, known as **prior** $p(\theta)$. It is usually a high entropy distribution reflecting the uncertainty before observing the data.

After observing a set of m data samples, belief about θ can be updated to $p(\theta|\{x^{(1)}, x^{(2)}, ..., x^{(m)}\})$ based on the eq. 20

$$p(\theta|\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}) = \frac{p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}|\theta) \cdot p(\theta)}{p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\})} \tag{21}$$

This is the **posterior** probability distribution, that expresses the additional knowledge (probabilities) acquired by the observed data sample. The term $p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}|\theta)$ renders the probability of data given θ, also known as **likelihood function**.

It can be observed that Bayesian makes predictions based on the probability distribution over parameter θ, in contrast to frequentist that make predictions based on a fixed value of θ.

After observing the next data sample $x^{(m+1)}$ the distribution over $x^{(m+1)}$ is rendered as

$$p(x^{(m+1)}|\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}) = \int p(x^{(m+1)}|\theta) \cdot p(\theta|\{x^{(1)}, x^{(2)}, ..., x^{(m)}\} d(\theta) \tag{22}$$

The term $p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ is the **marginal likelihood**, representing the evidence as observed by the data

$$p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}) = \int p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}|\theta) \cdot p(\theta) d(\theta) \tag{23}$$

In order to be able to model complex data we need to introduce additional data dependencies on some unobserved latent variable$z$. So the marginal likelihood can be redefined conditionally to the hidden or latent variables z. Based on eq.

23, the marginal likelihood can be expressed as

$$p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}) = \int p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}, z, \theta) dz d\theta =$$
$$= \int p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}|z, \theta) p(z) p(\theta) dz d\theta \tag{24}$$

Then, once a prior over the latent variables is defined as $p(x|z)$, the posterior can be computed

$$p(z, \theta|\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}) = \frac{p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}|z, \theta) \cdot p(z, \theta)}{\int p(\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}|z, \theta) p(z) p(\theta) dz d\theta} \tag{25}$$

As eq. 25 dictates, in order to derive the posterior distribution we should be able to compute the marginal likelihood, which is the denominator. Even though it may seem a simple task, in most cases the calculation of the integral is either difficult or intractable. **Variational Bayes** is focused on providing methods to approximate the posterior probability distribution.

5.3 Variational Bayes

The approximation of the posterior, according to variational Bayes, is a two step process. First, the posterior is being approximated with a simpler family of distributions and then the fittest distribution, amongst this family, is selected. The selection is made based on the maximization of the **Evidence Lower Bound (ELBO)**, also known as variational free energy.

Given that x observations have been made, z are the latent variables, and $q(z)$ is a distribution to approximate the true posterior distribution $p(z|x)$, then the log-probability of the observations, using the Jensen's inequality $f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$, becomes

$$\begin{aligned} log\, p(x) &= log \int_z p(x, z) = log \int_z p(x, z) \frac{q(z)}{q(z)} = \\ &= log\, (\mathbb{E}_q[\frac{p(x, z)}{q(z)}]) = log\, \mathbb{E}_q[log \frac{p(x, z)}{q(z)}] \geq \\ &\geq \mathbb{E}_q[log \frac{p(x, z)}{q(z)}] = \mathbb{E}_q[log\, p(x, z)] + H[z] \\ &= \mathbb{E}_q[log\, p(x|z)] + \mathbb{E}_q[log\, p(z)] + H[z] \end{aligned} \tag{26}$$

where $H[z]$ is the Shannon Entropy

$$H_z = -\mathbb{E}_q[log\, q(z)]$$

On this basis, and by utilizing the definition of the Kullback-Leibler (KL) divergence:

$$KL[q(z)||p(z)] = -(\mathbb{E}_q[log\, p(z)] + H[z]) \tag{27}$$

it is straightforward to obtain that:

$$log\, p(x) \geq \mathbb{E}_q[log\, p(x|z)] - KL[q(z)||p(z|x)] \tag{28}$$

Hence, we have obtained a lower-bound to the log-evidence of the model, $log\, p(x)$, which is expressed as a functional of the sought approximate (variational) posterior, $q(z)$. Thus, maximization of this functional over $q(z)$ allows for approximating infinitely well the actual posterior $p(z|x)$, with the approximation improving the more tighter this bound becomes.

5.4 Variational Bayes in Deep Learning

The main idea of applying variational Bayesian inference to deep learning models consists in calculating a posterior distribution over the network weights given the training data. The benefit of such a learning algorithm setup is that the so-obtained posterior distribution answers predictive queries about unseen data by taking expectations: Prediction is made by averaging the resulting predictions for each possible configuration of the weights, weighted according to their posterior distribution. This allows for accounting for uncertainty, which is prevalent in tasks dealing with sparse training datasets.

Specifically, let us consider a training dataset $\mathcal{D}$. A deep network essentially postulates and fits to the training data a (conditional) likelihood function of the form $p(\mathcal{D}|\boldsymbol{w})$, where $\boldsymbol{w}$ is the vector of network weights. In the case of Bayesian treatments of neural networks, an appropriate prior distribution, $p(\boldsymbol{w})$, is imposed over $\boldsymbol{w}$, and the corresponding posterior is inferred from the data [95]. This consists in introducing an approximate posterior distribution over the network weights, $q(\boldsymbol{w};\boldsymbol{\phi})$, and optimizing it w.r.t. a lower bound to the network log-marginal likelihood $\log p(\mathcal{D};\boldsymbol{\phi})$, commonly referred to as the evidence lower bound (ELBO), $\mathcal{L}(\boldsymbol{\phi})$ [96]; it holds

$$\begin{aligned}\log p(\mathcal{D};\boldsymbol{\phi}) \geq \mathcal{L}(\boldsymbol{\phi}) =& \mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\log p(\mathcal{D}|\boldsymbol{w}) \\ &+ \log p(\boldsymbol{w}) - \log q(\boldsymbol{w};\boldsymbol{\phi})]\end{aligned} \tag{29}$$

where $\mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\cdot]$ is the expectation of a function w.r.t. the random variable $\boldsymbol{w}$, drawn from $q(\boldsymbol{w};\boldsymbol{\phi})$. This is *equivalent to minimizing a KL divergence measure* between the inferred approximate variational density and the actual underlying distribution.

Turning to the selection of the imposed prior $p(\boldsymbol{w})$, one may opt for a fixed-form *isotropic* Gaussian:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0, \sigma_0^2 \boldsymbol{I}) \tag{30}$$

where $\boldsymbol{I}$ is the identity matrix, and $\mathcal{N}(\cdot|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a multivariate Gaussian with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. On the other hand, the sought variational posterior $q(\boldsymbol{w}; \boldsymbol{\phi})$ is for simplicity and efficiency purposes selected as a diagonal Gaussian of the form:

$$q(\boldsymbol{w}; \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \tag{31}$$

where $\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$, and $\text{diag}(\boldsymbol{\sigma}^2)$ is a diagonal matrix with the vector $\boldsymbol{\sigma}^2$ on its main diagonal.

An issue with the above formulation is that the entailed posterior expectation $\mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\log p(\mathcal{D}|\boldsymbol{w})]$ is analytically intractable; this is due to the non-conjugate nature of deep networks, stemming from the employed nonlinear activation functions. This prohibits taking derivatives of $\mathcal{L}(\boldsymbol{\phi})$ to effect derivation of the sought posterior $q(\boldsymbol{w}; \boldsymbol{\phi})$. In addition, approximating this expectation by simply drawing Monte-Carlo (MC) samples from the weights posterior is not an option, due to the prohibitively high variance of the resulting estimator.

To address this issue, one can resort to a simple reparameterization trick: We consider that the MC samples $\boldsymbol{w}^{(s)}$ used to approximate the expectation $\mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\log p(\mathcal{D}|\boldsymbol{w})]$ are functions of their posterior mean and variance, as well as a random noise vector, $\boldsymbol{\epsilon}$, sampled from a standard Gaussian distribution [97, 98, 99]. This can be effected by introducing the transform:

$$\boldsymbol{w} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{32}$$

where $\odot$ denotes the elementwise product of two vectors, and the $\boldsymbol{\epsilon}$ are distributed as $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. By substituting this transform into the derived ELBO expression, the entailed posterior expectation is expressed as an average over a standard Gaussian density, $p(\boldsymbol{\epsilon})$. This yields an MC estimator with low variance, under some mild conditions [99].

Under this rationale, inference is performed by drawing a number of MC samples from the inferred posteriors over the model parameters, $q(\cdot; \boldsymbol{\phi})$, and obtaining the average predictive value of the model that corresponds to these drawn parameter values (samples). As a result, in variational inference, the uncertainty in the data is taken into account, which is a significant issue when dealing with sparse datasets, which are prevalent in real-world.

6 t-Exponential Memory Networks for Q&A Machines

Recent advances in deep learning have brought to the fore models that can make multiple computational steps in the service of completing a task; these are capable of describing long-term dependencies in sequential data. Novel recurrent attention models over possibly large external memory modules constitute the core mechanisms that enable these capabilities. Our work addresses learning subtler and more complex underlying temporal dynamics in language modeling tasks that deal with sparse sequential data. To this end, we improve upon these recent advances, by adopting concepts from the field of Bayesian statistics, namely variational inference. Our proposed approach consists in treating the network parameters as latent variables with a prior distribution imposed over them. Our statistical assumptions go beyond the standard practice of postulating Gaussian priors. Indeed, to allow for handling outliers, which are prevalent in long observed sequences of multivariate data, multivariate t-exponential distributions are imposed. On this basis, we proceed to infer corresponding posteriors; these can be used for inference and prediction at test time, in a way that accounts for the uncertainty in the available sparse training data. Specifically, to allow for our approach to best exploit the merits of the t-exponential family, our method considers a new t-divergence measure, which generalizes the concept of the Kullback-Leibler divergence. We performed an extensive experimental evaluation of our approach, using challenging language modeling benchmarks, and illustrate its superiority over existing state-of-the-art techniques.

6.1 Introduction

Recent developments in machine learning have managed to achieve breakthrough improvements in modeling long-term dependencies in sequential data. Specifically, the machine learning community has recently witnessed a resurgence in models of computation that use explicit storage and a notion of attention [100, 101, 102, 103]. As it has been extensively shown, the capability of effectively manipulating such storage mechanisms offers a very potent solution to the problem of modeling long temporal dependencies. Its advantages have been particularly profound in the context of question-answering bots. In such applications, it is required that the trained models be capable of taking multiple computational steps in the service of answering a question or completing a related task.

This work builds upon these developments, seeking novel treatments of Memory Networks (MEM-NNs) [101, 102] to allow for more flexible and effective learning from sparse sequential data with heavy-tailed underlying densities. Indeed, both sparsity and heavy tails are salient characteristics in a large variety of real-world language modeling tasks. Specifically, the earliest solid empirical evidence that any sufficiently large corpus of natural language utterances entails heavy-tailed distributions with power-law nature dates back to 1935 [104].

Hence, we posit that the capability of better addressing these data properties might allow for advancing the state-of-the-art in the field. Our inspiration is drawn from recent developments in approximate Bayesian inference for deep learning models [105, 106, 107, 108, 109]. Bayesian inference in the context of deep learning models can be performed by considering that the network parameters are stochastic latent variables with some prior distribution imposed over them. This inferential framework allows for the developed network to account for the uncertainty in the available (sparse) training data. Thus, it is expected to yield improved predictive and inferential performance outcomes compared to the alternatives.

Existing Bayesian inference formulations of deep networks postulate Gaussian assumptions regarding the form of the imposed priors and corresponding (inferred) posterior distributions. Then, inference can be performed in an approximate, computationally efficient way, by resorting to variational Bayes [110]. This consists in searching for a proxy in an analytically solvable distribution family that approximates the true underlying distribution. To measure the closeness between the true and the approximate distribution, the relative entropy between these two distributions is used. Specifically, under the aforementioned Gaussian assumption, one can use the Shannon-Boltzmann-Gibbs (SBG) entropy, whereby the relative entropy yields the well known Kullback-Leibler (KL) divergence [111].

Despite these advances, in problems dealing with long sequential data comprising multivariate observations, such assumptions of normality are expected to be far from the actual underlying densities. Indeed, it is well-known that real-world multivariate sequential observations tend to entail a great deal of outliers (heavy-tailed nature). This fact gives rise to significant difficulties in data modeling, the immensity of which increases with the dimensionality of the data [112]. Hence, replacing the typical Gaussian assumption with alternatives has been recently proposed as a solution towards the amelioration of these issues [113].

Our work focuses on the t-exponential family[3], which was first proposed by Tsallis and co-workers [114, 115, 116], and constitutes a special case of the more general ϕ-exponential family [117, 118, 119]. Of specific practical interest to us is the Students'-t density, which has been extensively examined in the literature of generative models, such as hidden Markov models [120, 121, 122]. The Student's-t distribution is a bell-shaped distribution with heavier tails and one more parameter (degrees of freedom - DOF) compared to the normal distribution, and tends to a normal distribution for large DOF values [123]. Hence, it provides a much more robust approach to the fitting of models with Gaussian assumptions. On top of these merits, the t-exponential family also gives rise to a new t-divergence measure; this can be used for performing variational inference in a fashion that better accommodates heavy-tailed data (compared to standard KL-based solutions) [124].

Under this rationale, our proposed approach is founded upon the funda-

[3]Also referred to as the q-exponential family or the Tsallis distribution.

mental assumption that the imposed priors over the postulated MEM-NN parameters are Student's-t densities. On this basis, we proceed to infer their corresponding Student's-t posteriors, using the available training data. To best exploit the merits of the t-exponential family, we effect variational inference by resorting to a novel algorithm formulation; this consists in minimizing the t-divergence measure [124] over the sought family of approximate posteriors.

The contribution of this work can be summarized as follows:

1. The proposed approach, dubbed t-exponential Memory Network ($t-$MEM-NN), is the first ever attempt to derive a Bayesian inference treatment of MEM-NN models for question-answering. Our approach imposes a prior distribution over model parameters, and obtains a corresponding posterior; this is in contrast to existing approaches, which train simple point-estimates of the model parameters. By obtaining a full posterior density, as opposed to a single point-estimate of the model parameters, our approach is capable of coping with uncertainty in the trained model parameters.

2. We consider imposition of Student's-t priors, which are more appropriate for applications dealing with modeling heavy-tailed phenomena, as is the case with large natural language corpora [104]. This is the first time that explicit heavy-tailed distribution modeling is considered in the literature of MEM-NNs. Eventually, by making use of the trained posteriors, one can perform inference by drawing multiple alternative samples of the model parameters, and averaging the predictive outcomes pertaining to each sample. Thus, our predictions do not rely on the "correctness" of just a single model estimate; this way, the effects of model uncertainty are considerably ameliorated.

3. Model training is performed by maximizing a t-divergence-based objective functional, as opposed to the commonly used objectives that are based on the KL divergence. This allows for making the most out of the heavy tails of the obtained Student's-t distributions. Our work is the first one that performs approximate inference for deep latent variable models on the grounds of a t-divergence-based objective functional.

6.2 Methodological Background

6.2.1 End-to-end Memory Networks

Our proposed approach extends upon the existing theory of MEM-NNs, first introduced in [101]. Specifically, we are interested in a recent end-to-end-trainable extension of MEM-NNs, presented in [102]. That variant enjoys the advantage of requiring much less supervision during training, which is of major importance in real-world question-answering scenarios. The model input comprises a set of *facts,* $\{\boldsymbol{x}_i\}_{i=1}^N$, that are to be stored in the memory, as well as a *query* $\boldsymbol{q}$; given these, the model outputs an *answer* a. Each of the facts, $\boldsymbol{x}_i$, as well as the

query, $\boldsymbol{q}$, contain symbols coming from a dictionary with V words. Specifically, they are represented by vectors that are computed by concatenating the one-hot representations of the words they contain. The latter are obtained on the basis of the available dictionary comprising V words. The model writes all $\boldsymbol{x}_i$ to the memory, up to a fixed buffer size, and then finds a continuous variable encoding for both the $\boldsymbol{x}_i$ and the $\boldsymbol{q}$. These continuous representations are then processed via multiple hops, so as to generate the output a; this essentially constitutes one (selected) symbol from the available dictionary. This modeling scheme allows for establishing a potent training procedure, which can perform multiple memory accesses back to the input.

Specifically, let us consider one layer of the MEM-NN model. This is capable of performing a single memory hop operation; multiple hops in memory can be performed by simply stacking multiple such layers[4]. It comprises three main functional components:

Input memory representation: Let us consider an input set of facts, $\{\boldsymbol{x}_i\}_{i=1}^N$, to be stored in memory. This entire set is converted into *memory vectors*, $\{\boldsymbol{m}_i\}_{i=1}^N$, $\boldsymbol{m}_i \in \mathbb{R}^\delta$, computed by embedding each $\boldsymbol{x}_i$ in a continuous space, using a position embedding procedure [102] with embedding matrix $\boldsymbol{A}$. The query $\boldsymbol{q}$ is also embedded in the same space; this is performed via a position embedding procedure with embedding matrix $\boldsymbol{B}$, and yields an *internal state vector* $\boldsymbol{u}$. On this basis, MEM-NN proceeds to compute the match between the submitted query, $\boldsymbol{q}$, and each one of the available facts, by exploiting the salient information contained in their inferred embeddings; that is, the state vector, $\boldsymbol{u}$, and the memory vectors, $\{\boldsymbol{m}_i\}_{i=1}^N$, respectively. Specifically, it simply takes their inner product followed by a softmax:

$$\varpi_i = \text{softmax}(\boldsymbol{u}^T \boldsymbol{m}_i) \tag{33}$$

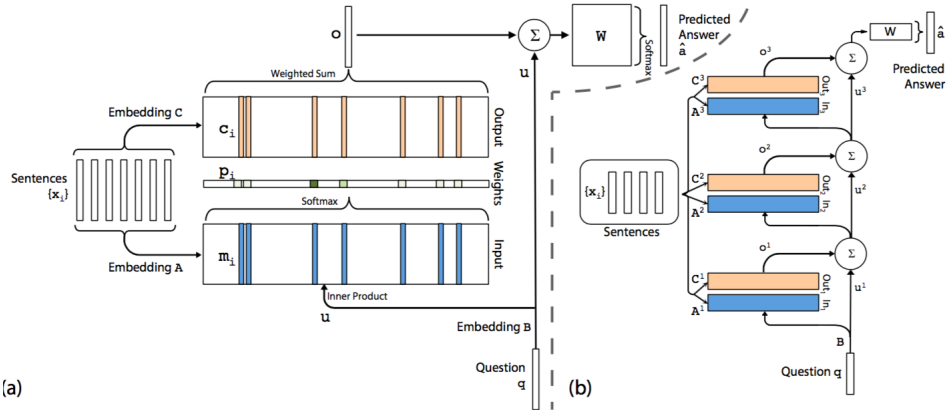
where

$$\text{softmax}(\boldsymbol{\zeta}_i) \triangleq \frac{\exp(\boldsymbol{\zeta}_i)}{\sum_j \exp(\boldsymbol{\zeta}_j)} \tag{34}$$

In essence, $\boldsymbol{\varpi} = [\varpi_i]_{i=1}^N$, is a probability vector over the facts, which shows how strong their affinity is with the submitted query. We will be referring to this vector as the inferred *attention* vector.

Output memory representation: In addition to the inferred memory vectors, MEM-NN also extracts from each fact, $\boldsymbol{x}_i$, a corresponding *output vector embedding*, $\boldsymbol{c}_i$, via another position embedding procedure [102] with embedding matrix $\boldsymbol{C}$. These output vector embeddings are considered to encode the salient information included in the presented facts that can be used for output (answer) generation. To achieve this goal, we leverage the inferred attention vector $\boldsymbol{\varpi}$, by using it to weight each fact (encoded via its inferred output vector embedding)

[4]The number of hops performed in memory is a model hyperparameter, that has to be selected in a heuristic manner. Naturally, there is no point in this number exceeding the number of facts presented to the model each time.



(a) A single-layer version of the considered model. (b) A 3-layer version, obtained via stacking (adopted from [102]).

Figure 14: MEM-NN model

with the corresponding computed probability value. It holds

$$\boldsymbol{o} = \sum_{i=1}^{N} \varpi_i \boldsymbol{c}_i \tag{35}$$

Generating the final prediction: MEM-NN output layer is a simple softmax layer, which is presented with the computed output vector, $\boldsymbol{o}$, as well as the internal state vector, $\boldsymbol{u}$. It estimates a probability vector over all possible predictions, $\hat{\boldsymbol{a}}$, that is all the entries of the considered dictionary of size V. It holds

$$\hat{\boldsymbol{a}} = \text{softmax}(\boldsymbol{W}(\boldsymbol{o} + \boldsymbol{u})) \tag{36}$$

where $\boldsymbol{W}$ is the weights matrix of the output layer of the network, whereby we postulate

$$a = \arg\max(\hat{\boldsymbol{a}})$$

A graphical illustration of the considered end–to-end trainable MEM-NN model, that we build upon in this work, is provided in Fig. 14. Our exhibition includes both single-layer models, capable of performing single memory hop operations, as well as multi-layer ones, obtained by stacking multiple singe layers, which can perform multiple hops in memory. Note that, to save parameters, and reduce the model's overfitting tendency, as well as its memory footprint, we tie the corresponding embedding matrices across all MEM-NN layers, as suggested in [102].

6.2.2 Variational Bayes in Deep Learning

The main idea of applying variational Bayesian inference to deep learning models consists in calculating a posterior distribution over the network weights given

the training data. The benefit of such a learning algorithm setup is that the so-obtained posterior distribution answers predictive queries about unseen data by taking expectations: Prediction is made by averaging the resulting predictions for each possible configuration of the weights, weighted according to their posterior distribution. This allows for accounting for uncertainty, which is prevalent in tasks dealing with sparse training datasets.

Specifically, let us consider a training dataset $\mathcal{D}$. A deep network essentially postulates and fits to the training data a (conditional) likelihood function of the form $p(\mathcal{D}|\boldsymbol{w})$, where $\boldsymbol{w}$ is the vector of network weights. In the case of Bayesian treatments of neural networks, an appropriate prior distribution, $p(\boldsymbol{w})$, is imposed over $\boldsymbol{w}$, and the corresponding posterior is inferred from the data [109]. This consists in introducing an approximate posterior distribution over the network weights, $q(\boldsymbol{w};\boldsymbol{\phi})$, and optimizing it w.r.t. a lower bound to the network log-marginal likelihood $\log p(\mathcal{D};\boldsymbol{\phi})$, commonly referred to as the evidence lower bound (ELBO), $\mathcal{L}(\boldsymbol{\phi})$ [125]; it holds

$$\begin{aligned}\log p(\mathcal{D};\boldsymbol{\phi}) \geq \mathcal{L}(\boldsymbol{\phi}) =& \mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\log p(\mathcal{D}|\boldsymbol{w}) \\ &+ \log p(\boldsymbol{w}) - \log q(\boldsymbol{w};\boldsymbol{\phi})]\end{aligned} \tag{37}$$

where $\mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\cdot]$ is the expectation of a function w.r.t. the random variable $\boldsymbol{w}$, drawn from $q(\boldsymbol{w};\boldsymbol{\phi})$. This is *equivalent to minimizing a KL divergence measure* between the inferred approximate variational density and the actual underlying distribution.

Turning to the selection of the imposed prior $p(\boldsymbol{w})$, one may opt for a fixed-form *isotropic* Gaussian:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0, \sigma_0^2 \boldsymbol{I}) \tag{38}$$

where $\boldsymbol{I}$ is the identity matrix, and $\mathcal{N}(\cdot|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a multivariate Gaussian with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. On the other hand, the sought variational posterior $q(\boldsymbol{w};\boldsymbol{\phi})$ is for simplicity and efficiency purposes selected as a diagonal Gaussian of the form:

$$q(\boldsymbol{w};\boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)) \tag{39}$$

where $\boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$, and $\mathrm{diag}(\boldsymbol{\sigma}^2)$ is a diagonal matrix with the vector $\boldsymbol{\sigma}^2$ on its main diagonal.

An issue with the above formulation is that the entailed posterior expectation $\mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\log p(\mathcal{D}|\boldsymbol{w})]$ is analytically intractable; this is due to the non-conjugate nature of deep networks, stemming from the employed nonlinear activation functions. This prohibits taking derivatives of $\mathcal{L}(\boldsymbol{\phi})$ to effect derivation of the sought posterior $q(\boldsymbol{w};\boldsymbol{\phi})$. In addition, approximating this expectation by simply drawing Monte-Carlo (MC) samples from the weights posterior is not an option, due to the prohibitively high variance of the resulting estimator.

To address this issue, one can resort to a simple reparameterization trick: We consider that the MC samples $\boldsymbol{w}^{(s)}$ used to approximate the expectation $\mathbb{E}_{q(\boldsymbol{w};\boldsymbol{\phi})}[\log p(\mathcal{D}|\boldsymbol{w})]$ are functions of their posterior mean and variance, as well

as a random noise vector, $\boldsymbol{\epsilon}$, sampled from a standard Gaussian distribution [108, 107, 105]. This can be effected by introducing the transform:

$$\boldsymbol{w} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{40}$$

where $\odot$ denotes the elementwise product of two vectors, and the $\boldsymbol{\epsilon}$ are distributed as $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. By substituting this transform into the derived ELBO expression, the entailed posterior expectation is expressed as an average over a standard Gaussian density, $p(\boldsymbol{\epsilon})$. This yields an MC estimator with low variance, under some mild conditions [105].

6.2.3 The Student's-t Distribution

The adoption of the multivariate Student's-t distribution provides a way to broaden the Gaussian distribution for potential outliers [123]. The probability density function (pdf) of a Student's-t distribution with mean vector $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$, and $\nu > 0$ degrees of freedom is [126]

$$t(\boldsymbol{y}_t; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) = \frac{\Gamma\left(\frac{\nu+\delta}{2}\right) |\boldsymbol{\Sigma}|^{-1/2} (\pi\nu)^{-\delta/2}}{\Gamma(\nu/2)\{1 + d(\boldsymbol{y}_t, \boldsymbol{\mu}; \boldsymbol{\Sigma})/\nu\}^{(\nu+\delta)/2}} \tag{41}$$

where δ is the dimensionality of the observations $\boldsymbol{y}_t$, $d(\boldsymbol{y}_t, \boldsymbol{\mu}; \boldsymbol{\Sigma})$ is the squared Mahalanobis distance between $\boldsymbol{y}_t, \boldsymbol{\mu}$ with covariance matrix $\boldsymbol{\Sigma}$

$$d(\boldsymbol{y}_t, \boldsymbol{\mu}; \boldsymbol{\Sigma}) = (\boldsymbol{y}_t - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{y}_t - \boldsymbol{\mu}) \tag{42}$$

and $\Gamma(s)$ is the Gamma function, $\Gamma(s) = \int_0^\infty e^{-t} z^{s-1} dz$.

It can be shown (see, e.g., [126]) that, in essence, the Student's-t distribution corresponds to a Gaussian scale model [127] where the precision scalar is a Gamma distributed latent variable, depending on the degrees of freedom of the Student's-t density. That is, given

$$\boldsymbol{y}_t \sim t(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) \tag{43}$$

it equivalently holds that [126]

$$\boldsymbol{y}_t | \xi_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}/\xi_t) \tag{44}$$

where the precision scalar, ξ_t, is distributed as

$$\xi_t \sim \mathcal{G}\left(\frac{\nu}{2}, \frac{\nu}{2}\right) \tag{45}$$

and $\mathcal{G}(\alpha, \beta)$ is the Gamma distribution.

A graphical illustration of the univariate Student's-t distribution, with $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ fixed, and for various values of the degrees of freedom ν, is provided in Fig. 15. As we observe, as $\nu \to \infty$, the Student's-t distribution tends to a Gaussian with the same $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. On the contrary, as ν tends to zero, the tails of the

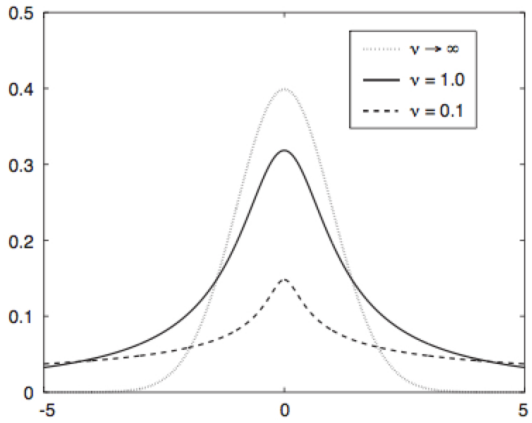


Figure 15: Univariate Student's-t distribution $t(\boldsymbol{y}_t; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu)$, with $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ fixed, for various values of ν [2].

distribution become longer, thus allowing for a better handling of potential outliers, without affecting the mean or the covariance of the distribution. Thus, by exploiting the heavier tails of the Student's-t distribution, a probabilistic generative model becomes capable of handling, in a considerably enhanced manner, outliers residing in the fitting datasets. That is, if the modeled phenomenon is actually heavy-tailed, the inferred Student's-t model will be capable to cope, by yielding a value for the fitted degrees of freedom parameter, ν, e.g., close to 1. On the other hand, if no such heavy-tailed nature does actually characterize the data, the fitted degrees of freedom parameter, ν, will yield a value close to infinity (practically, above 100). In the latter case, the model essentially reduces to a simpler Gaussian density [123].

6.2.4 The t-Divergence

As discussed in Section 6.2.2, conventional variational inference is equivalent to minimization of a KL divergence measure, which is also known as the relative SBG-entropy. Motivated from these facts, and in order to allow for making the most out of the merits (heavy tails) of the t-exponential family, the t-divergence was introduced in [124] as follows:

Definition 1. The t-divergence between two distributions, $q(\boldsymbol{h})$ and $p(\boldsymbol{h})$, is defined as

$$D_t(q||p) = \int \tilde{q}(\boldsymbol{h}) \log_t q(\boldsymbol{h}) \mathrm{d}\boldsymbol{h} - \tilde{q}(\boldsymbol{h}) \log_t p(\boldsymbol{h}) \mathrm{d}\boldsymbol{h} \tag{46}$$

where $\tilde{q}(\boldsymbol{h})$ is called the escort distribution of $q(\boldsymbol{h})$, and is given by

$$\tilde{q}(\boldsymbol{h}) = \frac{q(\boldsymbol{h})^t}{\int q(\boldsymbol{h})^t \mathrm{d}\boldsymbol{h}}, \; t \in \mathbb{R} \tag{47}$$

Importantly, the divergence $D_t(q||p)$ preserves the following two properties:

- $D_t(q||p) \geq 0,\ \forall q, p$. The equality holds only for $q = p$.
- $D_t(q||p) \neq D_t(p||q)$.

In the seminal work of [124], it has been shown that by leveraging the above definition of the t-divergence, $D_t(q||p)$, one can establish an advanced variational inference framework, much more appropriate for modeling data with heavy tails. We exploit these benefits in developing the training and inference algorithms of the proposed t-MEM-NN model, as explained in the following Section.

6.3 Proposed Approach

6.3.1 Model Formulation

t-MEM-NN extends upon the model design principles discussed previously, by building on the solid theory of variational inference based on the t-divergence. It does so by introducing a novel formulation that renders MEM-NN amenable to Bayesian inference.

To effect our modeling goals, we first consider that the postulated embeddings matrices are Student's-t distributed latent variables. Specifically, let us start by imposing a simple, zero-mean Student's-t prior distribution over them, with tied degrees of freedom:

$$p(\boldsymbol{A}) = t(\text{vec}(\boldsymbol{A})|\boldsymbol{0}, \boldsymbol{I}, \nu) \tag{48}$$

$$p(\boldsymbol{B}) = t(\text{vec}(\boldsymbol{B})|\boldsymbol{0}, \boldsymbol{I}, \nu) \tag{49}$$

$$p(\boldsymbol{C}) = t(\text{vec}(\boldsymbol{C})|\boldsymbol{0}, \boldsymbol{I}, \nu) \tag{50}$$

where $\text{vec}(\cdot)$ is the matrix vectorization operation, and $\nu > 0$ is the degrees of freedom hyperparameter of the imposed priors. On this basis, we seek to devise an efficient means of inferring the corresponding posterior distributions, given the available training data. We postulate that the sought posterior $q(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}; \boldsymbol{\phi})$ factorizes over $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ (mean-field approximation [118]); the factors are considered to approximately take the form of Student's-t densities with means, *diagonal* covariance matrices, and degrees of freedom inferred from the data. Hence, we have:

$$q(\boldsymbol{A}; \boldsymbol{\phi}) = t(\text{vec}(\boldsymbol{A})|\boldsymbol{\mu}_{\boldsymbol{A}}, \text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{A}}), \nu_{\boldsymbol{A}}) \tag{51}$$

$$q(\boldsymbol{B}; \boldsymbol{\phi}) = t(\text{vec}(\boldsymbol{B})|\boldsymbol{\mu}_{\boldsymbol{B}}, \text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{B}}), \nu_{\boldsymbol{B}}) \tag{52}$$

$$q(\boldsymbol{C}; \boldsymbol{\phi}) = t(\text{vec}(\boldsymbol{C})|\boldsymbol{\mu}_{\boldsymbol{C}}, \text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{C}}), \nu_{\boldsymbol{C}}) \tag{53}$$

where $\boldsymbol{\phi} = \{\boldsymbol{\mu}_i, \boldsymbol{\sigma}^2_i, \nu_i\}_{i \in \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}\}}$, and $\nu_i > 0, \forall i$.

On this basis, to perform model training in a way the best exploits the heavy tails of the developed model, we minimize the t-divergence between the sought

variational posterior and the postulated joint density over the observed data and the model latent variables. Thus, the proposed model training objective becomes

$$\begin{aligned} & q(\boldsymbol{A};\boldsymbol{\phi}), q(\boldsymbol{B};\boldsymbol{\phi}), q(\boldsymbol{C};\boldsymbol{\phi}), \boldsymbol{W} \\ & \quad = \underset{q,\boldsymbol{W}}{\arg\min}\, D_t\left(q(\boldsymbol{A};\boldsymbol{\phi}), q(\boldsymbol{B};\boldsymbol{\phi}), q(\boldsymbol{C};\boldsymbol{\phi}) || p(a;\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{W})\right) \end{aligned} \tag{54}$$

where $p(a;\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{W}) = p(a;\boldsymbol{W})p(\boldsymbol{A})p(\boldsymbol{B})p(\boldsymbol{C})$. Then, following the derivations rationale of [124], and by application of simple algebra, the expression of the t-divergence in (47) yields

$$\begin{aligned} & D_t\big(q(\boldsymbol{A};\boldsymbol{\phi}), q(\boldsymbol{B};\boldsymbol{\phi}), q(\boldsymbol{C};\boldsymbol{\phi}) || p(a;\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{W})\big) = \\ & \quad = D_t\left(q(\boldsymbol{A};\boldsymbol{\phi}) || p(\boldsymbol{A})\right) + D_t\left(q(\boldsymbol{B};\boldsymbol{\phi}) || p(\boldsymbol{B})\right) \\ & \quad + D_t\left(q(\boldsymbol{C};\boldsymbol{\phi}) || p(\boldsymbol{C})\right) - \mathbb{E}_{\tilde{q}(\cdot;\boldsymbol{\phi})}[\log p(a;\boldsymbol{W})] \end{aligned} \tag{55}$$

where $\tilde{q}(\cdot;\boldsymbol{\phi})$ is the escort distribution of the sought variational posterior, and $p(a;\boldsymbol{W})$ is a Multinoulli parameterized via the probability vector $\hat{\boldsymbol{a}}$, given by (33).

Following [124], and based on (41)-(46), we obtain that the t-divergence expressions in (48) can be written in the following form:

$$\begin{aligned} D_t\left(q(\boldsymbol{\Theta};\boldsymbol{\phi}) || p(\boldsymbol{\Theta})\right) = \sum_{l=1}^{\delta V} \Bigg\{ & \frac{\Psi_{ql}}{1-t}\left(1 + \frac{1}{\nu_{\boldsymbol{\Theta}}}\right) \\ & - \frac{\Psi_p}{1-t}\left(1 + \frac{[\boldsymbol{\sigma}^2_{\boldsymbol{\Theta}}]_l + [\boldsymbol{\mu}_{\boldsymbol{\Theta}}]_l^2}{\nu}\right) \Bigg\} \end{aligned} \tag{56}$$

where $\boldsymbol{\Theta} \in \{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}\}$, $[\boldsymbol{\xi}]_l$ is the lth element of a vector $\boldsymbol{\xi}$, we denote

$$\Psi_{ql} \triangleq \left(\frac{\Gamma(\frac{\nu_{\boldsymbol{\Theta}}+1}{2})}{\Gamma(\frac{\nu_{\boldsymbol{\Theta}}}{2})(\pi\nu_{\boldsymbol{\Theta}})^{1/2}[\boldsymbol{\sigma}_{\boldsymbol{\Theta}}]_l} \right)^{-\frac{2}{\nu_{\boldsymbol{\Theta}}+1}} \tag{57}$$

$$\Psi_p \triangleq \left(\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})(\pi\nu)^{1/2}} \right)^{-\frac{2}{\nu+1}} \tag{58}$$

δ is the dimensionality of the embeddings, V is the vocabulary size, ν is the degrees of freedom hyperparameter of the prior, and the free hyperparameter t is set as [124]

$$t = \frac{2}{1+\nu_{\boldsymbol{\Theta}}} + 1 \tag{59}$$

6.3.2 Training Algorithm Configuration

As we observe from the preceding discussion, the expectation of the conditional log-likelihood of the model, $\mathbb{E}_{\tilde{q}(\cdot;\boldsymbol{\phi})}[\log p(a;\boldsymbol{W})]$, is computed with respect to the

escort distribution of the sought posterior, $\tilde{q}(\cdot;\boldsymbol{\phi})$. Based on (44)-(46), it is easy to show that this escort distribution yields a factorized form, with [124]

$$\tilde{q}(\boldsymbol{\Theta};\boldsymbol{\phi}) = t\left(\text{vec}(\boldsymbol{\Theta})|\boldsymbol{\mu}_{\boldsymbol{\Theta}}, \frac{\nu_{\boldsymbol{\Theta}}}{\nu_{\boldsymbol{\Theta}}+2}\text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\Theta}}), \nu_{\boldsymbol{\Theta}}+2\right) \quad \forall \boldsymbol{\Theta} \in \{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}\} \tag{60}$$

Despite this convenient escort distribution expression, though, this posterior expectation cannot be computed analytically; hence, its gradient becomes intractable. This is due to the nonconjugate nature of t-MEM-NN, which stems from its nonlinear assumptions. Apparently, approximating this expectation using a set of S MC samples, $\{\boldsymbol{\Theta}_s\}_{s=1}^{S}$, drawn from the escort densities (28), would result in estimators with unacceptably high variance.

In this work, these issues are resolved by adopting the reparameterization trick ideas described in Section 6.2.2, adapted to the t-exponential family. Specifically, we perform a smart reparameterization of the MC samples from the Student's-t escort densities (28) which yields:

$$\boldsymbol{\Theta}_s = \boldsymbol{\mu}_{\boldsymbol{\Theta}} + \left(\frac{\nu_{\boldsymbol{\Theta}}}{\nu_{\boldsymbol{\Theta}}+2}\right)^{1/2} \boldsymbol{\sigma}_{\boldsymbol{\Theta}}\boldsymbol{\epsilon}_s \tag{61}$$

where $\boldsymbol{\epsilon}_s$ is random Student's-t noise with unitary variance:

$$\boldsymbol{\epsilon}_s \sim t(\boldsymbol{0}, \boldsymbol{I}, \nu_{\boldsymbol{\Theta}}+2) \tag{62}$$

Then, the resulting (reparameterized) t-divergence objective (48) can be minimized by means of any off-the-shelf stochastic optimization algorithm. For this purpose, in this work we utilize Adagrad; this constitutes a stochastic gradient descent algorithm with adaptive step-size [128], and fast and proven convergence to a local optimum. Adagrad updates the trained posterior hyperparameter set, $\boldsymbol{\phi}$, as well as the output layer weights, $\boldsymbol{W}$, by utilizing the gradient $\nabla_{\boldsymbol{\phi},\boldsymbol{W}} D_t\left(q(\boldsymbol{A};\boldsymbol{\phi}), q(\boldsymbol{B};\boldsymbol{\phi}), q(\boldsymbol{C};\boldsymbol{\phi})||p(a;\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{W})\right)$.

On each Adagrad iteration, this gradient is computed using only a small subset (minibatch) of the available training data, as opposed to using the whole training dataset. This allows for computational tractability, no matter what the total number of training examples is. To facilitate convergence, on each algorithm iteration a different minibatch is selected, in a completely random fashion.

In this context, it is important to appropriately select the number of MC samples drawn from (55) *during training*. In our work, we opt for the computationally efficient solution of drawing *just one MC sample during training*. One could argue that using only one MC sample is doomed to result in an approximation of limited quality. However, it has been empirically well-established that drawing just one MC sample is sufficient when Adagrad is executed with a small minibatch size compared to the size of the used training dataset [109, 113]. Indeed, this is the case with our experimental evaluations in Section 6.4. In all cases, network initialization is performed by means of the Glorot uniform

Table 1: Considered benchmark dataset: Indicative training data samples from three of the entailed types of tasks.

Sam walks into the kitchen.	Brian is a lion.	Sandra got the milk.
Sam picks up an apple.	Julius is a lion.	Sandra journeyed to the garden.
Sam walks into the bedroom.	Julius is white.	Sandra went back to the bathroom.
Sam drops the apple.	Bernhard is green.	Sandra put down the milk.
Q: Where is the apple?	Q: What color is Brian?	Q: Where was the milk before the bathroom?
A: Bedroom	A: White	A: Garden

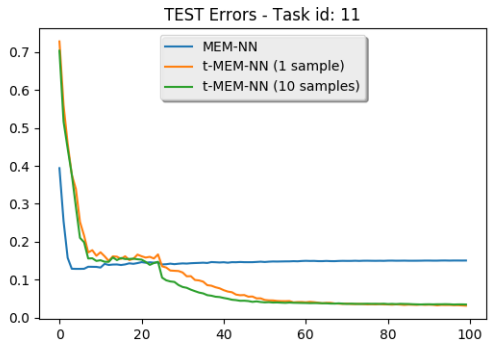


Figure 16: Test error per epoch: Task type #11.

scheme, except for the degrees of freedom hyperparameters; these are initialized at high values ($\nu = 100$), which essentially reduce the initial Student's-t densities of our model to simpler Gaussian ones (as discussed in Section II.C) [129].

6.3.3 Inference Procedure

Having obtained a training algorithm for our proposed t-MEM-NN model, we can now proceed to elaborate on how inference is performed using our method. As briefly hinted in Section 6.2.2, this consists in drawing a number of MC samples from the inferred posteriors over the model parameters, $q(\cdot; \boldsymbol{\phi})$, and obtaining the average predictive value of the model that corresponds to these drawn parameter values (samples). According to the related deep learning literature, drawing a set of $S = 10$ samples should be enough *for inference purposes* [109, 113]. We investigate the impact of the number of drawn MC samples to the eventually obtained performance of the inference algorithm of our model in our experiments that follow.

Table 2: Quantitative assessment: Accuracy results in the test set.

Test Accuracy (%)			
	Baseline	*t*-MEM-NN	
Task type	MemN2N	1 sample	10 samples
1: 1 supporting fact	100	100	100
2: 2 supporting facts	84	77	84
3: 3 supporting facts	55	55	**57**
4: 2 argument relations	96	94	96
5: 3 argument relations	88	87	88
6: yes/no questions	92	93	**97**
7: counting	83	81	**85**
8: lists/sets	87	87	**90**
9: simple negation	90	88	**92**
10: indefinite knowledge	78	81	**84**
11: basic coreference	85	98	**98**
12: conjunction	100	100	100
13: compound coreference	89	93	**95**
14: time reasoning	92	92	**96**
15: basic deduction	100	100	100
16: basic induction	45	45	**46**
17: positional reasoning	51	51	**53**
18: size reasoning	87	89	**91**
19: path finding	12	12	**14**
20: agent's motivation	100	100	100

Table 3: Attention in task type #1 - story #202.

Facts
1. mary moved to the hallway
1. daniel travelled to the office
2. john went back to the hallway
3. john moved to the office
4. sandra journeyed to the kitchen
5. mary moved to the bedroom

Question	Answer	Supporting Facts
where is daniel?	office	daniel travelled to the office

(a) Predicted answers.

MemN2N	*t*-MEM-NN (1 sample)	*t*-MEM-NN (10 samples)
bedroom	bedroom	office

(b) Model attention per hop.

MemN2N		*t*-MEM-NN (1 sample)		*t*-MEM-NN (10 samples)	
hop 1	daniel travelled to the office	hop 1	daniel travelled to the office	hop 1	daniel travelled to the office
hop 2	daniel travelled to the office	hop 2	daniel travelled to the office	hop 2	daniel travelled to the office
hop 3	mary moved to the bedroom	hop 3	mary moved to the bedroom	hop 3	daniel travelled to the office

Table 4: Attention in task type #11 - story #3.

Facts
1. john journeyed to the hallway
2. after that he journeyed to the garden
3. john moved to the office
4. following that he went to the hallway
5. sandra travelled to the bedroom
6. then she moved to the hallway
7. mary travelled to the hallway
8. afterwards she went to the bathroom

Question	Answer	Supporting Facts
where is sandra?	hallway	sandra travelled to the bedroom then she moved to the hallway

MemN2N	t-MEM-NN (1 sample)	t-MEM-NN (10 samples)
bathroom	hallway	hallway

(a) Predicted answers.

MemN2N		t-MEM-NN (1 sample)		t-MEM-NN (10 samples)	
hop 1	afterwards she went to the bathroom	hop 1	sandra travelled to the bedroom	hop 1	sandra travelled to the bedroom
hop 2	afterwards she went to the bathroom	hop 2	then she moved to the hallway	hop 2	then she moved to the hallway
hop 3	afterwards she went to the bathroom	hop 3	then she moved to the hallway	hop 3	then she moved to the hallway

Table 5: Attention in task type #14 - story #22.

Facts
1. mary went back to the kitchen this morning
2. mary travelled to the school yesterday
3. yesterday fred travelled to the bedroom
4. yesterday bill moved to the park
5. this afternoon bill went back to the park
6. bill went to the school this morning

Question	Answer	Supporting Facts
where was bill before the park?	school	this afternoon bill went back to the park bill went to the school this morning

(a) Predicted answers.

MemN2N	t-MEM-NN (1 sample)	t-MEM-NN (10 samples)
park	park	school

(b) Model attention per hop.

MemN2N		t-MEM-NN (1 sample)		t-MEM-NN (10 samples)	
hop 1	yesterday bill moved to the park	hop 1	yesterday bill moved to the park	hop 1	yesterday bill moved to the park
hop 2	yesterday bill moved to the park	hop 2	this afternoon bill went back to the park	hop 2	this afternoon bill went back to the park
hop 3	yesterday bill moved to the park	hop 3	yesterday bill moved to the park	hop 3	bill went to the school this morning

6.4 Synthetic Question&Answering Tasks

In this Section, we perform a thorough experimental evaluation of our proposed t-MEM-NN model. We provide a quantitative assessment of the efficacy, the effectiveness, and the computational efficiency of our approach, combined with deep qualitative insights into few of its key performance characteristics. To this end, we utilize a publicly available benchmark, which is popular in the recent literature, namely the set of synthetic question-answering (QA) tasks defined in [130] (bAbI). Specifically, we consider the *English-Language* tasks of the bAbI dataset that comprise 1K training examples (*en-1K* tasks). This dataset, as described in Chapter 3, comprises 20 different types of tasks, which are characterized by different qualitative properties. Some of them are harder to be learned, while some others are much easier.

The idea of all the types of tasks entailed in this dataset is rather simple. Each task consists of a set of statements (facts), a question, and an answer. The answer comprises only one word from the available vocabulary. Given the facts, a question is asked and an answer is expected. Then, model performance can be evaluated on the basis of the percentage of generated answers that match the available groundtruth. To allow for a better feeling of what our dataset looks like, we provide indicative samples from three of the entailed types of tasks in Table 1. Note that the available dataset also provides additional supporting facts that may be made use of by the trained models.

To provide some comparative results, we evaluate two variants of our method, namely one where *inference is performed* using only a single MC sample (drawn from the model posteriors), and another one where 10 MC samples are used. In all cases, *training* is performed by drawing just one MC sample. In addition, we compare to the state-of-the-art alternative that is the closest related to our approach, namely the MemN2N method of [102]. Our source codes have been developed in Python, using the TensorFlow library [1], as well as open-source software published by Dominique Luna[5]. Our experiments are run on an Intel Xeon server with 64GB RAM and an NVIDIA Tesla K40 GPU.

6.4.1 Experimental Setup

Model training is performed by utilizing the training dataset provided in the used *en-1K* bAbI benchmark. This comprises 1000 examples from each type of task; from these, we randomly select 900 samples to perform training, and retain the remainder 100 for validation purposes. Each training example comprises the full set of data pertaining to the task, including the correct answer (which we expect the system to generate), apart from the corresponding question and available statements (facts). On this basis, a trained model is evaluated by presenting it with the facts and the questions pertaining to each example in the test set, and running its inference algorithm to obtain a predicted answer. The available test set comprises 1000 cases from each task type, which are completely unknown to the trained models.

[5] `https://github.com/domluna/memn2n`

Since the used benchmark comprises a multitude of task types, we train a different model (of each evaluated method) for each task type. An obvious advantage of such a modeling setup consists in the fact that it allows for the trained models to be finely-tuned to data with very specific patterns. On the other hand, the imposed weight tying across model layers is a strong safeguard against possible model overfitting.

Turning to the selection of the hyperparameters of the training algorithms, *we emphasize that we adopt exactly the same configuration for both our model and the baseline.* Specifically, we perform training for 100 epochs, as also suggested in [102]. Adagrad is carried out by splitting our training data into 32 minibatches. The learning rate is initialized at $\eta = 0.01$, and is annealed every 25 epochs by $\eta/2$, until the maximum number of epochs is reached (similar to [102]). Glorot initialization for all trained models is performed via a Gaussian distribution with zero mean and $\sigma = 0.1$. The postulated models employ an external memory size of 50 sentences. Nil words are padded with zero embedding (zero one-hot encodings). The embedding space size, δ, is set to 20; this is shown in [102] to work best for the MemN2N model. In order to calculate the output (predicted answer) for each problem, 3 computational steps (*hops*) are performed, similar to the suggestions of [102].

6.4.2 Quantitative Assessment

In this Section, the accuracy of the model-generated predictions is measured and reported. In order for a trained model to be considered successful in some type of task, we stipulate that a 95% accuracy must be reached, similar to [102]. The test-set accuracy results obtained under our prescribed experimental setup are provided in Table 2. Expectably, increasing the number of MC samples drawn to perform inference improves performance in the most challenging of the task types, while retaining performance in the rest. As we observe, our approach manages to pass our set success threshold of 95% accuracy in 9 task types (# 1, 4, 6, 11, 12, 13, 14, 15, 20), thus outperforming the MemN2N baseline which passes the threshold in only 5 cases (# 1, 4, 12, 15, 20). Another characteristic finding is that our approach outperforms the baseline in most tasks, and achieves the same performance in the few rest. In our perception, this finding vouches for the better capacity of our approach to learn the underlying distributions in the modeled dataset. Of course, one also observes that our method does not offer significant improvements on the types of tasks for which the baseline has low accuracy. However, this apparently constitutes an inherent weakness of the whole learning paradigm adopted by MEM-NN networks; this cannot be rectified by introducing better inference mechanisms, as we do in this work.

Further, to show how the test error of the considered approaches converges over the training algorithm epochs, in Fig. 16 we depict the evolution of the test error for an indicative task type, in one execution of our experiments. We observe that both variants of our approach (i.e., using one or ten MC samples for performing inference) converge gradually and consistently over the training algorithm epochs. In contrast, the baseline MemN2N approach appears to reach

its best performance early-on during training, and subsequently remains almost stable.

6.4.3 Qualitative Assessment

To provide some qualitative insights into the inferred question-answering rationale of our approach, and how this compares to the original MemN2N, in Tables 3 - 5 we illustrate what the inferred attention vectors look like in three indicative test cases. More specifically, we record which fact each model mostly focuses its attention on, on every hop; further, we compare this result to the supporting facts included in the dataset. Our so-obtained results indicate that the proposed approach manages to better focus on the most salient information (sentences), as indicated by the provided supporting facts. This outcome offers a strong intuitive explanation of the reasons why t-MEM-NN appears to outperform the baseline, in most of the considered task types.

6.4.4 Computational Times

Apart from inferential accuracy, the computational costs of a devised method constitute another aspect which affects its efficacy. To allow for objectively examining this aspect, we have developed all the evaluated algorithms using the same software platform, and executed them on the same machine (each time without concurrently running any other user application).[6] Then, we recorded the resulting wall-clock times, for both model training and inference.

As we have observed, baseline MemN2N training requires an average of 146.5 msec per minibatch, while our approach imposes a negligible increase, requiring an average of 149.8 msec. This is reasonable, since training of our model entails the same set of computations as baseline MemN2N, with the only exception being the computation of the t-divergence terms pertaining to the degrees of freedom parameters, which are of linear complexity. Thus the observed slight increase in computational times, which is clearly worth it for the improved model performance.

Turning to the computational costs of the inference algorithm, we observe that our approach requires computational times comparable to MemN2N in order to generate one answer. This is clearly reasonable, since both models entail the same set of feedforward computations. On the other hand, it is significant to underline that the extra computational costs of t-MEM-NN that arise from an increase in the number of MC samples drawn to perform inference (from just one to ten) are completely negligible. Indeed, an average increase of 0.1 msec was observed. This was well-expected, since the extra matrix multiplications that arise from the use of multiple drawn MC samples are completely parallelizable over commercially available, modern GPU hardware.

[6] Our source codes have been developed in Python, using the TensorFlow library [1]. We run our experiments on an Intel Xeon server with 64GB RAM and an NVIDIA Tesla K40 GPU.

Hence, to summarize, our approach only imposes a small increase in the total training time compared to the baseline MemN2N, which is more than worth it for the offered increase in modeling accuracy. In addition, using multiple MC samples to perform inference is clearly auspicious, since it allows for improved accuracy while imposing negligible increases in computational times.

6.5 Further Insights

6.5.1 Effect of the number of hops

In the previous experimental evaluations, we performed three memory hops, following the suggestions of [102]. Yet, it is desirable to know how t-MEM-NN performance may be affected if we change this number. To get a proper answer to this question, we repeat our experiments considering only one memory hop, as well as an increased number of five hops. We provide the so-obtained results in Tables 6 and 7. As we observe, conducting only one hop in memory results in a significant performance impairment in the vast majority of the considered task types. This way, our model manages to pass the success threshold in only two of the considered task types (#1 and 12), as opposed to the eight task types attained when performing three memory hops. On the other hand, a further increase of the number of hops from three to five seems to undermine the obtained performance. Indeed, t-MEM-NN passes the success threshold in only six task types, namely task types #1, 4, 11, 12, 15, 20. We posit that these outcomes are due to the structure of the used dataset, as it also becomes obvious from the number of available supporting facts, which is more than one in most cases, but it seldom exceeds three.

Table 6: t-MEM-NN accuracy in the test set, performing just one memory hop.

Test Accuracy (%)		
Task type	1 sample	10 samples
1: 1 supporting fact	100	100
2: 2 supporting facts	34	**35**
3: 3 supporting facts	23	23
4: 2 argument relations	79	**80**
5: 3 argument relations	87	87
6: yes/no questions	69	69
7: counting	52	**56**
8: lists/sets	33	**35**
9: simple negation	77	**80**
10: indefinite knowledge	44	**47**
11: basic coreference	25	**28**
12: conjunction	96	**99**
13: compound coreference	92	**93**
14: time reasoning	21	**23**
15: basic deduction	57	**75**
16: basic induction	44	**46**
17: positional reasoning	48	48
18: size reasoning	85	**86**
19: path finding	9	9
20: agent's motivation	83	**84**

Table 7: t-MEM-NN accuracy in the test set, increasing the number of memory hops to five.

Test Accuracy (%)		
Task type	1 sample	10 samples
1: 1 supporting fact	100	100
2: 2 supporting facts	78	**82**
3: 3 supporting facts	56	**57**
4: 2 argument relations	93	**96**
5: 3 argument relations	87	87
6: yes/no questions	73	**78**
7: counting	78	**81**
8: lists/sets	87	**89**
9: simple negation	83	**86**
10: indefinite knowledge	82	**84**
11: basic coreference	96	**97**
12: conjunction	98	**99**
13: compound coreference	90	90
14: time reasoning	87	**89**
15: basic deduction	94	**96**
16: basic induction	44	**46**
17: positional reasoning	50	**51**
18: size reasoning	89	**90**
19: path finding	11	**14**
20: agent's motivation	100	100

6.5.2 Altering the embedding space dimensionality

In addition, it is interesting to examine what the effect of the embedding space dimensionality is on the performance of our approach. Indeed, it is reasonable to expect that the larger the embedding space the more potent a postulated model is. However, the entailed increase in the trainable model parameters does also come at the cost of considerably higher overfitting tendencies. These might eventually undermine the obtained accuracy profile of t-MEM-NN.

To examine these aspects, we repeat our experiments considering a smaller embeddings size than the one suggested in [102], specifically $\delta = 10$, as well as a much larger one, specifically $\delta = 50$. The outcomes of this investigation are depicted in Tables 8 and 9, respectively. As we observe, decreasing the postulated embedding space dimensionality to $\delta = 10$ results in worse model performance, since the model passes the success threshold in only four task types (#1, 11, 12, 20). Similarly interesting are the findings pertaining to an increase of the embedding space size to $\delta = 50$. In this case, average model performance over all the considered task types remains essentially stable. Thus, it seems that model performance reaches a plateau as we continue to increase the size of the embeddings. Note also that postulating either $\delta = 10$ or $\delta = 50$ results in t-MEM-NN passing the success threshold in exactly the same set of

task types as when we postulate $\delta = 20$.

To summarize, increasing the embedding space size does not appear to be worth the extra computational costs. Indeed, t-MEM-NN requires an extra 2.2 msec to generate one answer when δ increases from 20 to 50, which represents an average increase by 62%. On the other hand, predictive accuracy does not yield any considerable increase.

Table 8: t-MEM-NN accuracy in the test set, reducing the embedding size to $\delta = 10$.

Test Accuracy (%)		
Task type	1 sample	10 samples
1: 1 supporting fact	100	100
2: 2 supporting facts	55	**56**
3: 3 supporting facts	31	**34**
4: 2 argument relations	79	**81**
5: 3 argument relations	83	83
6: yes/no questions	60	**61**
7: counting	77	77
8: lists/sets	85	85
9: simple negation	70	70
10: indefinite knowledge	71	**75**
11: basic coreference	96	**97**
12: conjunction	98	98
13: compound coreference	92	92
14: time reasoning	83	83
15: basic deduction	70	**74**
16: basic induction	45	45
17: positional reasoning	50	50
18: size reasoning	87	87
19: path finding	10	10
20: agent's motivation	100	100

Table 9: t-MEM-NN accuracy in the test set, increasing the embedding size to $\delta = 50$.

Test Accuracy (%)		
Task type	1 sample	10 samples
1: 1 supporting fact	100	100
2: 2 supporting facts	73	**78**
3: 3 supporting facts	52	**55**
4: 2 argument relations	90	**91**
5: 3 argument relations	85	**86**
6: yes/no questions	72	**78**
7: counting	79	**81**
8: lists/sets	86	**89**
9: simple negation	86	**87**
10: indefinite knowledge	80	**82**
11: basic coreference	95	**96**
12: conjunction	99	99
13: compound coreference	93	**95**
14: time reasoning	81	**84**
15: basic deduction	98	**100**
16: basic induction	44	**45**
17: positional reasoning	50	**52**
18: size reasoning	89	**91**
19: path finding	11	**13**
20: agent's motivation	100	100

6.5.3 Joint task modeling

In all the previous experiments, we have trained a distinct model on each one of the 20 types of QA tasks included in the considered *en-1K* bAbI benchmark. However, one could also consider jointly training one single model on data from all the included task types. Clearly, one may argue that this alternative approach might make it more difficult for the trained model to distinguish between fine patterns. However, it is also the case that, by training a joint model on all task types, we also allow for a significantly reduced overfitting tendency (by increasing the effective number of training data). Hence, we consider training a single model on all the task types; we train for 60 epochs, and anneal the learning rate every 15 epochs.

Our results, obtained by setting the latent space dimensionality equal to $\delta = 20$, and by employing 3 memory hops, are depicted in Table 10. It is evident that, similar to the single-task setup, inference using 10 MC samples yields better average performance than using just one. Considering the set success threshold of 95% accuracy, we obtain that our method succeeds in 9 task types (# 1, 6, 9, 10, 12, 13, 14, 15, 20); this way, it outperforms baseline MemN2N by one task. Note also that t-MEM-NN performance is greater than MemN2N in all these tasks.

Table 10: Joint-modeling setup: Accuracy results in the test set.

Test Accuracy (%)			
	Baseline	t-MEM-NN	
Task type	MemN2N	1 sample	10 samples
1: 1 supporting fact	99	100	**100**
2: 1 supporting facts	**86**	61	79
3: 3 supporting facts	**71**	45	60
4: 2 argument relations	85	77	85
5: 3 argument relations	86	82	86
6: yes/no questions	96	99	**100**
7: counting	84	84	**85**
8: lists/sets	89	88	89
9: simple negation	96	99	**99**
10: indefinite knowledge	92	91	**95**
11: basic coreference	**94**	90	91
12: conjunction	98	99	**100**
13: compound coreference	97	93	**98**
14: time reasoning	88	91	**96**
15: basic deduction	98	97	**100**
16: basic induction	46	46	**48**
17: positional reasoning	55	55	**58**
18: size reasoning	59	67	**71**
19: path finding	10	14	**17**
20: agent's motivation	100	100	100

6.5.4 Experimental evaluation with the rest of the available tasks

Further, for completeness sake, we examine how the performance of our model compares to the competition when it comes to considering the rest of the tasks available in the bAbI dataset. That is, we report results on the 20 QA tasks that are developed in the Hindi language, that comprise both 1K as well as 10K training examples, or employ random shuffling. These are denoted as *en-10K, hn-1K, hn-10K, shuffle-1K,* and *shuffle-10K,* respectively. Our findings, obtained by setting the latent space dimensionality equal to $\delta = 20$, and by employing 3 memory hops, are depicted in Tables 11-15. As we observe, in all cases our approach exceeds the 95% success threshold in more tasks than the baseline. This is yet another result that vouches for the validity of our theoretical claims, and the efficacy of our algorithmic construction and derivations.

Table 11: Accuracy results in the Hindi/1k test set.

Test Accuracy (%)			
	Baseline	t-MEM-NN	
Task type	MemN2N	1 sample	10 samples
1: 1 supporting fact	99	100	**100**
2: 1 supporting facts	85	85	**88**
3: 3 supporting facts	53	50	**55**
4: 2 argument relations	97	96	**98**
5: 3 argument relations	88	85	**89**
6: yes/no questions	89	89	89
7: counting	83	83	**84**
8: lists/sets	88	87	**90**
9: simple negation	89	89	**92**
10: indefinite knowledge	78	75	**82**
11: basic coreference	84	90	**98**
12: conjunction	99	99	99
13: compound coreference	89	93	**95**
14: time reasoning	93	92	**96**
15: basic deduction	100	98	100
16: basic induction	45	45	**47**
17: positional reasoning	49	45	**51**
18: size reasoning	86	86	**93**
19: path finding	13	12	13
20: agent's motivation	100	100	100

Table 12: Accuracy results in the Shuffled/1k test set.

Test Accuracy (%)			
	Baseline	t-MEM-NN	
Task type	MemN2N	1 sample	10 samples
1: 1 supporting fact	100	100	100
2: 1 supporting facts	79	75	79
3: 3 supporting facts	49	50	**54**
4: 2 argument relations	95	93	95
5: 3 argument relations	86	85	86
6: yes/no questions	91	90	**94**
7: counting	80	80	**84**
8: lists/sets	87	87	**89**
9: simple negation	89	89	**94**
10: indefinite knowledge	79	80	**83**
11: basic coreference	83	92	**100**
12: conjunction	99	99	**100**
13: compound coreference	89	90	**95**
14: time reasoning	91	91	**96**
15: basic deduction	100	99	100
16: basic induction	45	45	**46**
17: positional reasoning	50	49	**52**
18: size reasoning	86	88	**91**
19: path finding	12	12	**13**
20: agent's motivation	100	100	100

Table 13: Accuracy results in the en/10k test set.

Test Accuracy (%)			
	Baseline	t-MEM-NN	
Task type	MemN2N	1 sample	10 samples
1: 1 supporting fact	100	100	100
2: 1 supporting facts	98	97	**100**
3: 3 supporting facts	83	85	**89**
4: 2 argument relations	100	98	100
5: 3 argument relations	99	99	99
6: yes/no questions	100	99	100
7: counting	95	95	**97**
8: lists/sets	97	98	**100**
9: simple negation	99	98	99
10: indefinite knowledge	96	96	**99**
11: basic coreference	91	94	**100**
12: conjunction	100	100	100
13: compound coreference	94	94	**98**
14: time reasoning	97	95	**100**
15: basic deduction	100	100	100
16: basic induction	47	47	47
17: positional reasoning	57	57	57
18: size reasoning	89	88	**92**
19: path finding	33	33	**36**
20: agent's motivation	100	100	100

Table 14: Accuracy results in the Hindi/10k test set.

Test Accuracy (%)			
	Baseline	t-MEM-NN	
Task type	MemN2N	1 sample	10 samples
1: 1 supporting fact	100	100	100
2: 1 supporting facts	97	97	**100**
3: 3 supporting facts	83	81	**91**
4: 2 argument relations	99	99	99
5: 3 argument relations	98	98	**99**
6: yes/no questions	100	100	100
7: counting	94	93	**97**
8: lists/sets	97	97	**99**
9: simple negation	98	98	**99**
10: indefinite knowledge	93	93	**97**
11: basic coreference	94	96	**100**
12: conjunction	100	100	100
13: compound coreference	96	97	**100**
14: time reasoning	98	98	**100**
15: basic deduction	100	100	100
16: basic induction	46	43	**47**
17: positional reasoning	56	56	**57**
18: size reasoning	91	91	**94**
19: path finding	31	32	**35**
20: agent's motivation	100	100	100

Table 15: Accuracy results in the Shuffled/10k test set.

Test Accuracy (%)			
	Baseline	t-MEM-NN	
Task type	MemN2N	1 sample	10 samples
1: 1 supporting fact	100	100	100
2: 1 supporting facts	97	98	**100**
3: 3 supporting facts	83	83	**88**
4: 2 argument relations	100	100	100
5: 3 argument relations	99	98	99
6: yes/no questions	100	100	100
7: counting	95	95	**97**
8: lists/sets	96	96	**99**
9: simple negation	99	99	**100**
10: indefinite knowledge	97	97	**99**
11: basic coreference	92	94	**100**
12: conjunction	100	100	100
13: compound coreference	96	96	**100**
14: time reasoning	98	95	**100**
15: basic deduction	100	100	100
16: basic induction	47	47	**48**
17: positional reasoning	56	56	**57**
18: size reasoning	90	90	**92**
19: path finding	35	35	**38**
20: agent's motivation	100	100	100

6.5.5 Do we actually need to infer heavy-tailed posteriors?

As we have already discussed, the central assumption in the formulation of our model that the imposed posteriors are of multivariate Student's-t form allows to account for heavy-tailed underlying densities, with power-law nature. However, a question that naturally arises is whether this assumption actually addresses an existing problem. That is, whether the underlying densities are actually heavy-tailed. To address this question, we can leverage some attractive properties of the Student's-t distribution. Specifically, as we have explained in Section 6.2.3, the degrees of freedom parameter of a Student's-t density controls how heavy its tails are. This way, a model employing Student's-t densities effectively modifies, through model fitting, how heavy its tails are, to account for the actual needs of the application at hand.

Therefore, examining the degrees of freedom values of the fitted model posteriors is a natural means of deducing whether the imposition of Student's-t densities is actually worthwhile, or a simpler Gaussian assumption would suffice. Our findings can be summarized as follows: In all the experimental cases reported above, the posteriors over the input embedding matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, given in (19) and (20), yield values $\nu_{\boldsymbol{A}}, \nu_{\boldsymbol{B}} \leq 5$, while for the output embeddings $\boldsymbol{C}$, given by (21), we have $\nu_{\boldsymbol{C}} \leq 20$. These findings imply that all our

fitted models end up requiring degrees of freedom parameter values low enough to account for quite heavy tails. Thus, our empirical experimental findings vouch for the efficacy of our assumptions.

6.6 Guess the Number

We conclude our experimental investigations by considering a setup that allows for us to evaluate whether our model is capable of learning latent abstract concepts by engaging in conversation with a teacher. Specifically, our devised experimental setup emulates a kid's game named "Guess the number." This well-known game is played by two entities, the teacher and the student; on each round, the teacher picks an integer number between given boundaries, and the student tries to guess which number the teacher has originally selected. When the student guesses a number different than the target, the teacher provides them a hint whether the guessed number is greater or less than it. On the sequel, the student has to make another guess, following the limits dictated in the preceding conversation (i.e., all previous guesses and provided hints). The game continues until either the student guesses the target number or we reach the maximum allowed numbers of tries.

Under this experimental rationale, we have constructed datasets that correspond to two different scenarios; in these, the chosen numbers lie between: (a) 0 and 10; and (b) 0 and 100. The maximum number of tries is set to 100 for both scenarios. In addition, we perform evaluation with a diverse number of training examples including 100, 1K, and 10K, in order to assess the effect of the training dataset size. The latent space dimensionality of all the evaluated models is set equal to $\delta = 20$, while we employ 3 memory hops, similar to Section 6.4.2. Model evaluation is performed on 100 distinct test games, in all cases. Inference for t-MEM-NN is run with the number of drawn MC samples set to 1 or 10; training is performed by drawing just one MC sample.

For the purpose of quantitative performance evaluation of the trained models, we have defined and use three metrics: (i) *accuracy*, which describes the average percentage of correct decisions; a guess is considered correct when the guessed number is within the limits defined by the conversation's history; (ii) *success*, which describes the average percentage of games where the target number was correctly guessed within the preset limit of 100 tries; and (iii) *rounds*, the average number of guesses made before the target was found. The last metric obviously concerns only successful games.

Our so-obtained results are depicted in Tables 16 and 17. To allow for the reader to get an insight into the construction of the considered game, as well as the generated outputs of MemN2N and our proposed approach, we provide two characteristic output samples of the evaluated models in Table 18. According to the outcome of this assessment, it is obvious that our proposed approach outperforms the baseline model in all metrics for all scenarios.

Table 16: Guess the number: selected numbers take values between 0 and 10.

100 Training examples			
	Baseline	t-MEM-NN	t-MEM-NN
Metric	MemN2N	1 sample	10 samples
Accuracy (%)	77	87	**91**
Success (%)	86	95	**98**
Rounds	4.3	3.9	**3.6**
1k Training examples			
	Baseline	t-MEM-NN	t-MEM-NN
Metric	MemN2N	1 sample	10 samples
Accuracy (%)	99	100	**100**
Success (%)	100	100	**100**
Rounds	4.2	3.8	**3.6**
10k Training examples			
	Baseline	t-MEM-NN	t-MEM-NN
Metric	MemN2N	1 sample	10 samples
Accuracy (%)	97	99	**100**
Success (%)	98	100	**100**
Rounds	4.0	3.8	**3.7**

Table 17: Guess the number: selected numbers take values between 0 and 100.

100 Training examples			
	Baseline	t-MEM-NN	t-MEM-NN
Metric	MemN2N	1 sample	10 samples
Accuracy (%)	17	19	**21**
Success (%)	20	28	**31**
Rounds	7.7	5.1	**3.2**
1k Training examples			
	Baseline	t-MEM-NN	t-MEM-NN
Metric	MemN2N	1 sample	10 samples
Accuracy (%)	30	49	**55**
Success (%)	45	61	**72**
Rounds	8.7	8.2	**7.7**
10k Training examples			
	Baseline	t-MEM-NN	t-MEM-NN
Metric	MemN2N	1 sample	10 samples
Accuracy (%)	63	78	**85**
Success (%)	68	89	**97**
Rounds	12.6	9.1	**8.5**

Table 18: t-MEM-NN: Sample output of "Guess the number" game.

Model: **t-MEM-NN**
min: **0**
max: **100**
Train examples: **1000**

**** TESTING MODEL STARTS ****

Select a number between 0 and 100
Round: 1

Selection: 76
Correct: Selection within Bounds!
Accuracy: = 1.0
min_num: -1
max_num: 101
Hint: Target is a smaller number
Round: 2

Selection: 100
Wrong: Selection Out of Bounds!
Accuracy: = 0.5
min_num: -1
max_num: 76
Hint: Target is a smaller number
Round: 3

Selection: 1
Correct: Selection within Bounds!
Accuracy: = 0. 66
min_num: -1
max_num: 76
Hint: Target is a larger number
Round: 4

Selection: 59
Correct: Selection within Bounds!
Accuracy: = 0.75
min_num: 1
max_num: 76
Hint: Target is a smaller number
Round: 5

Selection: 11
Correct: Selection within Bounds!
Accuracy: = 0.8
min_num: 1
max_num: 59
Hint: Target is a larger number
Round: 6

Selection: 56
Correct: Selection within Bounds!
Accuracy: = 0.83
min_num: 11
max_num: 59
Hint: Target is a larger number
Round: 7

Selection: 54
Wrong: Selection Out of Bounds!
Accuracy: = 0.71
min_num: 56
max_num: 59
Hint: Target is a larger number
Round: 8

Selection: 57
Correct: Selection within Bounds!
Accuracy: = 0.75
min_num: 56
max_num: 59

****

Congratulations, the target is 57
You found the correct answer after 8 rounds
Accuracy: 0.75

6.7 Conclusions

In this work, we attacked the problem of modeling long-term dependencies in sequential data. Specifically, we focused on question-answering bots; these inherently require the ability to perform multiple computational steps of analyzing observed patterns over long temporal horizons, and on multiple time-scales. To achieve this goal, one may resort to the paradigm of neural attention models that operate over large external memory modules. This is a recent development in the field of machine learning, yielding state-of-the-art performance in challenging benchmark tasks.

In this context, the core contribution of our work was the provision of a novel inferential framework for this class of models, which allows to account for the uncertainty in the modeled data. This is a significant issue when dealing with sparse datasets, which are prevalent in real-world the considered tasks. In addition, our method was carefully crafted so as to best accommodate data with heavy-tailed distributions, which are typical in multivariate sequences.

To achieve these goals, we devised a novel Bayesian inference-driven algorithmic formulation of end-to-end-trainable MEM-NNs. Specifically, we considered a stochastic model formulation, where the trainable parameters (embedding matrices) of the network are imposed appropriate prior distributions, and corresponding posteriors are inferred by means of variational Bayes. To allow for accommodating heavy-tailed data, we postulated latent variables belonging to the t-exponential family; specifically, we considered multivariate Student's-t densities. In the same vein, and in order to allow for reaping the most out of the data modeling power of Student's-t densities, we performed variational inference for our model under a novel objective function construction. This was based on a t-divergence criterion, which offers an attractive alternative to the KL divergence (that is minimized in conventional variational Bayes), tailored to heavy-tailed data.

We performed an extensive experimental evaluation of our approach using challenging question-answering benchmarks. We provided thorough insights into the inferential outcomes of our approach, and how these compare to the competition. We also illustrated that our proposed approach achieves the reported accuracy improvement without undermining computational efficiency, both in training time and in prediction generation time.

7 Amortized Context Vector Inference for Sequence-to-Sequence Networks

Neural attention (NA) is an effective mechanism for inferring complex structural data dependencies that span long temporal horizons. As a consequence, it has become a key component of sequence-to-sequence models that yield state-of-the-art performance in as hard tasks as abstractive document summarization (ADS) and video captioning (VC). NA mechanisms perform inference of context vectors; these constitute weighted sums of *deterministic* input sequence *encodings*, adaptively sourced over long temporal horizons. However, recent work in the field of amortized variational inference (AVI) has shown that it is often useful to treat representations generated by deep networks as *latent random variables.* This allows for the models to better explore the space of possible representations. Based on this motivation, in this work we introduce a novel regard towards a popular NA mechanism, namely *additive soft-attention* (ASA). Our approach consists in treating the *context vectors* generated by ASA models as Gaussian latent variables, the posteriors of which are inferred by employing AVI. Both the means and the covariance matrices of the inferred posteriors are parameterized via deep network mechanisms similar to those employed in the context of standard ASA. To illustrate our method, we implement it in the context of two state-of-the-art sequence-to-sequence models with attention, one used for performing ADS, and one used to effect VC. We conduct an extensive experimental evaluation using challenging ADS and VC benchmarks, and show how our approach compares to the baselines.

7.1 Introduction

Sequence-to-sequence (*seq2seq*) or encoder-decoder models [31] constitute a novel solution to inferring relations between sequences of different lengths. They are broadly used for addressing tasks including machine translation (MT) [32, 49], abstractive document summarization (ADS), descriptive caption generation (DCG) [50], and question answering (QA) [41], to name just a few. *Seq2seq* models comprise two distinct RNN models: an *encoder* RNN, and a *decoder* RNN. Their main principle of operation is based on the idea of learning to infer an intermediate *context vector representation,* $\boldsymbol{c}$, which is "shared" among the two RNN modules of the model, i.e., the encoder and the decoder. Specifically, the encoder converts the source sequence to a context vector (e.g., the final state of the encoder RNN), while the decoder is presented with the inferred context vector to produce the target sequence.

Despite these merits, though, baseline *seq2seq* models cannot learn temporal dynamics over long horizons. This is due to the fact that a single context vector $\boldsymbol{c}$ is capable of encoding rather limited temporal information. This major limitation has been addressed via the development of neural attention (NA) mechanisms [32]. NA has been a major breakthrough in Deep Learning, as it enables the decoder modules of *seq2seq* models to adaptively focus on temporally-

varying subsets of the source sequence. This capacity, in turn, enables flexibly capturing long temporal dynamics in a computationally efficient manner. As such, it is no coincidence that state-of-the-art NLP algorithms, including models addressing DCG and ADS, are based on *seq2seq*-type models that are founded upon NA mechanisms [70, 82, 68, 32, 49].

Among the large collection of recently devised NA variants, the vast majority build upon the concept of *Soft Attention* [50]. Under this rationale, the NA mechanism consists of three layers. The first layer employs a nonlinear function; this produces a similarity estimate between the encoder outputs (encodings) pertaining to each individual element of the source sequence, and the *current state vector* of the decoder RNN. The second layer is a softmax function that uses the obtained similarity estimates to assign each source sequence encoding a weight. These weights are the *inferred attention probabilities,* which express the relevance of each encoding to the current state of the decoder. Finally, the output layer computes a weighted mean of the source sequence encodings, where the employed weights are the inferred attention probabilities. We eventually use this weighted mean to update the context vector of the postulated *seq2seq* model, and drive generation of the next element in the target sequence.

As we observe from the preceding discussion, at each sequence generation (decoding) step, NA-obtained context vectors essentially constitute deterministic representations of the dynamics between the source sequence and the decodings obtained thus far. However, recent work in the field of amortized variational inference (AVI) [97, 99] has shown that it is often useful to treat representations generated by deep networks as *latent random* variables. Indeed, it is now well-understood that, under such an inferential setup, the trained deep learning models become more effective in exploring the space of possible representations, instead of getting trapped to poor solutions. Then, model training reduces to inferring posterior distributions over the introduced latent variables; we resort to variational inference approximations, whereby the sought variational posteriors are parameterized via appropriate deep networks.

Motivated from these research advances, in this work we consider a novel formulation of Soft Attention. Specifically, we propose an NA mechanism formulation where the generated context vectors are considered random latent variables over which AVI is performed. We dub our approach amortized context vector inference (ACVI). To exhibit the efficacy of ACVI, we implement it in the context of two popular *seq2seq* model variants. Specifically, we consider: (i) Pointer-Generator Networks [86], which constitute a state-of-the-art approach for addressing ADS tasks; and (ii) baseline *seq2seq* models with additive soft-attention (ASA), applied to the task of VC.

7.2 Methodological Background

7.2.1 Abstractive Document Summarization

Abstractive document summarization consists in not only selecting words or sentences to copy from an original document, but also learning to generate new

sentences or novel words during the summarization process. The introduction of *seq2seq* models has rendered ADS both feasible and effective [82, 85]. Dealing with out-of-vocabulary (OOV) words was one of the main difficulties that early ADS models were confronted with. Word and/or phrase repetition was a second major issue. The pointer-generator model presented in [86] constitutes one of the most comprehensive efforts towards ameliorating these issues.

In a nutshell, this model comprises one bidirectional LSTM [24] (BiLSTM) encoder, and a unidirectional LSTM decoder, which incorporates an ASA mechanism [32]. The word embedding of each token, $\boldsymbol{x}_i,\ i \in \{1, \ldots, N\}$, in the source sequence (document) is presented to the encoder BiLSTM; this obtains a representation (encoding) $\boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}}_i; \overleftarrow{\boldsymbol{h}}_i]$, where $\overrightarrow{\boldsymbol{h}}_i$ is the corresponding forward LSTM state, and $\overleftarrow{\boldsymbol{h}}_i$ is the corresponding backward LSTM state. Then, at each generation step, t, the decoder LSTM gets as input the (word embedding of the) previous token in the target sequence. During training, this is the previous word in the available reference summary; during inference, this is the previous generated word. On this basis, it updates its internal state, $\boldsymbol{s}_t$, which is then presented to the postulated ASA network. Specifically, the employed attention mechanism computes the attention distribution, $\boldsymbol{a}_t$, as follows:

$$e_t^i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_h \boldsymbol{h}_i + \boldsymbol{W}_s \boldsymbol{s}_t + \boldsymbol{b}_{attn}) \tag{63}$$

$$\boldsymbol{a}_t = \text{softmax}(\boldsymbol{e}_t),\ \boldsymbol{e}_t = [e_t^i]_i \tag{64}$$

where the $\boldsymbol{W}_\cdot$ are trainable weight matrices, $\boldsymbol{b}_{attn}$ is a trainable bias vector, and $\boldsymbol{v}$ is a trainable parameter vector of the same size as $\boldsymbol{b}_{attn}$.

Then, the model updates the obtained context vector, $\boldsymbol{c}_t$, by taking the weighted average of all the source token encodings; the used weights are the inferred attention probabilities:

$$\boldsymbol{c}_t = \sum_i a_t^i \boldsymbol{h}_i \tag{65}$$

The context vector is combined with the decoder state to obtain the *predictive distribution* of the next *generated* word:

$$P_t^{vocab} = \text{softmax}(\boldsymbol{V}' \tanh(\boldsymbol{V}[\boldsymbol{s}_t; \boldsymbol{c}_t] + \boldsymbol{b}) + \boldsymbol{b}^{'}) \tag{66}$$

where $\boldsymbol{V}$ and $\boldsymbol{V}'$ are trainable weight matrices, while $\boldsymbol{b}$ and $\boldsymbol{b}'$ are trainable bias vectors.

In parallel, the network also computes an additional probability, p_t^{gen}, which expresses whether the next output should be *generated* by sampling from the predictive distribution, P_t^{vocab}, or the model should simply *copy* one of the already available s*ource sequence tokens*. This mechanism allows for the model to cope with OOV words; it is defined via a simple sigmoid layer of the form:

$$p_t^{gen} = \sigma(\boldsymbol{w}_c^T \boldsymbol{c}_t + \boldsymbol{w}_s^T \boldsymbol{s}_t + \boldsymbol{w}_x^T \boldsymbol{x}_t + \boldsymbol{b}_{ptr}) \tag{67}$$

where $\boldsymbol{x}_t$ is the decoder input, while the $\boldsymbol{w}_\cdot$ and $\boldsymbol{b}_{ptr}$ are trainable parameter vectors. The probability of copying the ith source sequence token is considered

equal to the corresponding attention probability, a_t^i. Eventually, the obtained probability that the next output word will be β (found either in the vocabulary or among the source sequence tokens) yields:

$$P_t(\beta) = p_t^{gen} P_t^{vocab}(\beta) + (1-p_t^{gen}) \sum_{i:\beta_i=\beta} a_t^i \tag{68}$$

Finally, a *coverage mechanism* may also be employed [89], as a means of penalizing words that have already received attention in the past, to prevent repetition. *Coverage* acts as a global attention aggregator that is added to the attention layer (63). Specifically, the coverage vector, $\boldsymbol{k}_t$, is defined as:

$$\boldsymbol{k}_t = [k_t^i]_{i=1}^N = \sum_{\tau=0}^{t-1} \boldsymbol{a}_\tau \tag{69}$$

Using the so-obtained coverage vector, expression (63) is modified as follows:

$$e_t^i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_h \boldsymbol{h}_i + \boldsymbol{W}_s \boldsymbol{s}_t + \boldsymbol{w}_k k_t^i + \boldsymbol{b}_{attn}) \tag{70}$$

where $\boldsymbol{w}_k$ is a trainable parameter vector of size similar to $\boldsymbol{v}$.

Model training is performed via minimization of the categorical cross-entropy between the generated predictions and the available groundtruth. A coverage term is also added to the loss function, to facilitate repetition prevention; it takes the form

$$\lambda \sum_i \sum_t \min(a_t^i, c_t^i) \tag{71}$$

Here, λ controls the influence of the coverage term; in the remainder of this work, we set $\lambda = 1$.

7.2.2 Video Captioning

Video captioning constitutes one of the first reported applications of *seq2seq* models with attention to data stemming from diverse modalities. In this work, we consider a simple *seq2seq* model with attention that comprises a BiLSTM encoder, an LSTM decoder, and an output distribution of the form (4). The used encoder is presented with visual features obtained from a *pretrained* convolutional neural network (CNN). Our use of a pretrained CNN serves the purpose of evaluating ACVI in the context of a model which does not entail any sophisticated assumption apart from plain LSTM encoders and decoders. This allows for us to better scrutinize the benefits obtained by using ACVI (we elaborate on the specific model configuration in Section 7.4.2).

7.3 Proposed Approach

To motivate our proposed ACVI scheme, we focus on the definition of the ASA network described in Section 7.2.1. However, note that it is straightforward

to extend the rationale of ACVI in the context of SA schemes that employ alternative score functions, e.g. dot-product or tensor-product ones [49].

We begin by introducing the core assumption that the computed context vectors, $\boldsymbol{c}_t$, constitute latent random variables. Further, we assume that, at each time point, t, the corresponding context vector, $\boldsymbol{c}_t$, is drawn from a distribution associated with one of the available source sequence encodings, $\{\boldsymbol{h}_i\}_{i=1}^N$. Let us introduce the set of latent indicator variables, $z_t \in \{1, \ldots, N\}$, with $z_t = i$ denoting that the context vector $\boldsymbol{c}_t$ is drawn from the ith density, that is the density associated with the ith encoding, $\boldsymbol{h}_i$. Then, we postulate the following hierarchical model:

$$\boldsymbol{c}_t | z_t = i; \mathcal{D} \sim p(\boldsymbol{\theta}(\boldsymbol{h}_i)) \tag{72}$$

$$z_t = i | \mathcal{D} \sim \pi_t^i(a_t^i) \tag{73}$$

where $\mathcal{D}$ are the source and target sequences, $\boldsymbol{\theta}$ denotes the parameters set of the context vector conditional density, and π_t^i denotes the probability of drawing from the ith conditional at time t. Notably, we assume that the assignment probabilities π_t^i are functions of the attention probabilities, a_t^i. This is reasonable, since higher affinity of the decoder state, $\boldsymbol{s}_t$, with the ith encoding, $\boldsymbol{h}_i$, at time t, should result in higher probability that the context vector be drawn from the corresponding conditional density. Similarly, we consider that the parameters set $\boldsymbol{\theta}$ is a function parameterized by the encodings vector it is associated with, $\boldsymbol{h}_i$.

Having defined the hierarchical model (72)-(73), it is important that we examine the resulting expression of the posterior density $p(\boldsymbol{c}_t; \mathcal{D})$. By marginalizing over (72) and (73), we obtain

$$p(\boldsymbol{c}_t; \mathcal{D}) = \sum_{i=1}^{N} \pi_t^i(a_t^i)\, p(\boldsymbol{\theta}(\boldsymbol{h}_i)) \tag{74}$$

In other words, we obtain a *finite mixture model posterior over the context vectors,* with mixture conditional densities associated with the available source sequence encodings, and mixture weights associated with the corresponding attention vectors. In addition, it is also interesting to compare this expression to the definition of context vectors under the conventional ASA scheme. From (65), we observe that conventional ASA is merely a special case of our proposed model, obtained by introducing two assumptions: (i) that the postulated mixture component assignment probabilities are identity functions of the associated attention probabilities, i.e.

$$\pi_t^i(a_t^i) = a_t^i \tag{75}$$

and (ii) that the conditional densities of the context vectors have all their mass concentrated on $\boldsymbol{h}_i$, that is they collapse onto the single point, $\boldsymbol{h}_i$:

$$p(\boldsymbol{c}_t | z_t = i; \mathcal{D}) = \delta(\boldsymbol{h}_i) \tag{76}$$

Indeed, by combining (74) - (76), we yield

$$p(\boldsymbol{c}_t; \mathcal{D}) = \sum_{i=1}^{N} a_t^i \delta(\boldsymbol{h}_i) = \delta\Big(\sum_{i=1}^{N} a_t^i \boldsymbol{h}_i \Big) \tag{77}$$

whence we obtain (65) with probability 1.

From the above discussion, it becomes apparent that our approach replaces the simplistic conditional density expression (76) with a more appropriate family $p(\boldsymbol{\theta}(\boldsymbol{h}_i))$, as in (74). Such a consideration may result in significant advantages for the postulated *seq2seq* model. Specifically, it introduces a principled way of accounting for uncertainty in either our model formulation or the available training data. This way, our trained model becomes more agile in searching for effective context representations, as opposed to getting trapped to poor local solutions. Indeed, recent developments in deep learning research have provided strong evidence of these advantages in a multitude of deep network configurations and addressed tasks, e.g. [97, 99, 131].

In the following, we examine conditional densities of Gaussian form. Adopting the inferential rationale of AVI, we consider that these conditional Gaussians are parameterized via the postulated BiLSTM encoder. Specifically, we assume

$$p(\boldsymbol{c}_t|z_t = i; \mathcal{D}) = \mathcal{N}\big(\boldsymbol{c}_t|\boldsymbol{\mu}(\boldsymbol{h}_i), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{h}_i))\big) \tag{78}$$

where

$$\boldsymbol{\mu}(\boldsymbol{h}) = \mathrm{MLP}_{\boldsymbol{\mu}}(\boldsymbol{h}) \tag{79}$$

$$\log \boldsymbol{\sigma}^2(\boldsymbol{h}) = \mathrm{MLP}_{\boldsymbol{\sigma}}(\boldsymbol{h}) \tag{80}$$

$\mathrm{MLP}(\cdot)$ are trainable MLPs comprising two layers of size $\dim(\boldsymbol{h})$, and the encodings, $\boldsymbol{h}_i$, are obtained from a BiLSTM encoder, similar to conventional models.

On this basis, and by adopting the assumption (75), we eventually yield the posterior density:

$$p(\boldsymbol{c}_t; \mathcal{D}) = \sum_{i=1}^{N} a_t^i \mathcal{N}\big(\boldsymbol{c}_t|\boldsymbol{\mu}(\boldsymbol{h}_i), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{h}_i))\big) \tag{81}$$

Thus, we have arrived at a full statistical treatment of the context vectors, $\boldsymbol{c}_t$. The corresponding density is defined as a Gaussian mixture model; its means and log-covariance diagonals are parameterized via the encoder BiLSTM module of the *seq2seq* model. This concludes the formulation of the proposed ACVI mechanism.

7.3.1 Training Algorithm

To perform training of a *seq2seq* model equipped with the ACVI mechanism, we resort to maximization of the resulting evidence lower-bound (ELBO) expression. To this end, we need first to introduce some prior assumption over the context latent variables, $\boldsymbol{c}_t$. In the following, we consider

$$p(\boldsymbol{c}_t) = \mathcal{N}\big(\boldsymbol{c}_t|\boldsymbol{0}, \boldsymbol{I}\big) \tag{82}$$

This selection both serves the purpose of simplicity and offers a valid way to effect model regularization.

On the grounds of these assumptions, it is easy to show that the resulting ELBO expression becomes

$$\mathcal{L} = \sum_t \left\{ \mathbb{E}_{p(\boldsymbol{c}_t;\mathcal{D})}[-J_t] \; - \mathrm{KL}[p(\boldsymbol{c}_t;\mathcal{D})||p(\boldsymbol{c}_t)] \right\} \tag{83}$$

where $J_t = J(\boldsymbol{c}_t)$ is the error function at decoding time t of the *seq2seq* model, and the KL divergence term yields

$$\begin{aligned} \mathrm{KL}[p(\boldsymbol{c}_t;\mathcal{D})||p(\boldsymbol{c}_t)] = & -\frac{1}{2}\left[\sum_{i=1}^{N} a_t^i \boldsymbol{\mu}(\boldsymbol{h}_i)\right]^T \left[\sum_{i=1}^{N} a_t^i \boldsymbol{\mu}(\boldsymbol{h}_i)\right] \\ & + \frac{1}{2}\mathrm{trace}\left(\mathrm{diag}\left(1 + \log \sum_{i=1}^{N} (a_t^i)^2 \boldsymbol{\sigma}^2(\boldsymbol{h}_i) - \sum_{i=1}^{N} (a_t^i)^2 \boldsymbol{\sigma}^2(\boldsymbol{h}_i)\right)\right) \end{aligned} \tag{84}$$

On the other hand, to approximate the analytically intractable posterior expectation $\mathbb{E}_{p(\boldsymbol{c}_t;\mathcal{D})}[-J_t]$, we draw Monte-Carlo (MC) samples from the amortized posteriors $p(\boldsymbol{c}_t;\mathcal{D})$. We ensure that the resulting MC estimators will be of low variance, by adopting the reparameterization trick. From (81), we directly obtain the reparameterized MC samples [132]:

$$\boldsymbol{c}_t \leftarrow \mathbb{E}[\boldsymbol{c}_t;\mathcal{D}] + \sqrt{\mathbb{V}[\boldsymbol{c}_t;\mathcal{D}]} \circ \boldsymbol{\epsilon} \tag{85}$$

where

$$\mathbb{E}[\boldsymbol{c}_t;\mathcal{D}] = \sum_{i=1}^{N} a_t^i \boldsymbol{\mu}(\boldsymbol{h}_i) \tag{86}$$

$$\mathbb{V}[\boldsymbol{c}_t;\mathcal{D}] = \sum_{i=1}^{N} \left(a_t^i\right)^2 \boldsymbol{\sigma}^2(\boldsymbol{h}_i) \tag{87}$$

and $\boldsymbol{\epsilon}$ is white random noise with unitary variance, i.e. $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

ELBO maximization can be performed by resorting to any modern, off-the-shelf, stochastic gradient optimizer. In this work, we adopt Adam with its default settings [133].

7.3.2 Inference Algorithm

To perform target decoding by means of a *seq2seq* model that employs the ACVI mechanism, we resort to *Beam search* [90], as usual in the literature of *seq2seq* models. Note, though, that in the case of our proposed approach, the resulting predictive probability gets averaged over the context vector latent variables, $\boldsymbol{c}_t$. That is, the predictive distribution expression (4) now becomes

$$\begin{aligned} P_t^{vocab} = & \mathbb{E}[\mathrm{softmax}(\boldsymbol{V}'\tanh(\boldsymbol{V}[\boldsymbol{s}_t;\boldsymbol{c}_t] + \boldsymbol{b}) + \boldsymbol{b}')] \\ \approx & \frac{1}{K}\sum_{k=1}^{K} \mathrm{softmax}(\boldsymbol{V}'\tanh(\boldsymbol{V}[\boldsymbol{s}_t;\boldsymbol{c}_t^{(k)}] + \boldsymbol{b}) + \boldsymbol{b}') \end{aligned} \tag{88}$$

where K is the number of MC samples, $\boldsymbol{c}_t^{(k)}$, drawn from the trained amortized posterior (81). In our experiments, we set $K = 10$ in all cases; *beam width* is set to five.

7.4 Experimental Evaluation[7]

7.4.1 Abstractive Document Summarization

For transparency and replicability purposes, we perform an extensive quantitative and qualitative assessment, adopting the setup described in [86]. Our experiments are based on the *non-anonymized CNN/Daily Mail* dataset, which comprises 287,226 training pairs of documents and reference summaries, 13,368 validation pairs, and 11,490 test pairs. In this dataset, the average article length is 781 tokens; the average summary length is 3.75 sentences, with the average summary being 56 tokens long. In all our experiments, we restrict the used vocabulary to the 50K most common words in the considered dataset. Note that this is significantly smaller than typical in the literature [70].

To obtain some comparative results, we experimentally evaluate the performance of a pointer-generator network employing both our ACVI mechanism and the standard ASA mechanism considered in [86]. This allows for directly assessing the benefits of introducing a latent variable regard towards the inferred context vectors, which is amenable to AVI. Following the suggestions in [86], and to allow for the maximum possible comparability of the reported outcomes, we evaluate both our approach and the considered baseline with LSTMs that comprise 256-dimensional states. These are mapped to 256-dimensional features via the matrices $\boldsymbol{W}_h$ and $\boldsymbol{W}_s$ in Eq. (63).

The observations presented to the encoder module of each model constitute 128-dimensional word embeddings of the original 50K-dimensional one-hot-vectors of the source tokens. Similarly, the observations presented to the decoder modules are 128-dimensional word embeddings pertaining to the summary tokens (reference tokens during training; generated tokens during inference). Both these embeddings are trained, as part of the overall training procedure of the evaluated models. We use **ROUGE**[8] (ROUGE-1, ROUGE-2 and ROUGE-L) [92] and METEOR[9] [93] as our performance metrics. METEOR is evaluated both in exact match mode (rewarding only exact matches between words) as well as full mode (which additionally rewards matching stems, synonyms and paraphrases).

To allow for faster convergence of model training, we split it into five distinct phases. On each phase, we employ a different number of maximum encoding steps for the evaluated models (i.e., the size of the inferred attention vectors), as well as for the maximum allowed number of decoding steps. We provide the related details in Table 1. During these phases, we train the employed models

[7] We have developed our source codes in Python, using the TensorFlow library [1]. We run our experiments on a server with 64GB RAM and an NVIDIA Tesla K40 GPU.

[8] `pypi.python.org/pypi/pyrouge/`.

[9] `www.cs.cmu.edu/~alavie/METEOR`.

Table 19: Abstractive Document Summarization: Training phases.

Phase	Iterations	Max encoding steps	Max decoding steps
1	0 - 71k	10	10
2	71k - 116k	50	50
3	116k - 184k	100	50
4	184k - 223k	200	50
5	223k - 250k	400	100

with the coverage mechanism being disabled; that is, we set $\boldsymbol{w}_k = \mathbf{0}$. We enable this mechanism only after these five training phases conclude. Specifically, we perform a final 3K iterations of model training, during which we train the $\boldsymbol{w}_k$ weights along with the rest of the model parameters. We use gradient clipping with a maximum gradient norm of 2, but do *not* use any form of regularization.

We provide some indicative examples of the generated summaries in Tables 21-24). It is apparent that our model is capable of yielding high quality outcomes, with satisfactory performance when it comes to OOV words. Our quantitative evaluation is provided in Table 2. To allow for deeper insights, we show therein how performance of the evaluated models varies before and after we introduce the coverage mechanism (trained for 3K iterations, as described above). For completeness sake, we also cite the performance of alternative state-of-the-art approaches on the same data. These include ADS and Extractive Summarization models. The latter simply rely on copying from the source document, as opposed to learning to generate anew; this may produce lower quality summaries, but substantially less prone to grammatical or syntactic errors. As we observe, utilization of ACVI outperforms all the alternatives by a large margin. Finally, it is interesting to examine whether ACVI increases the propensity of a trained model towards generating *novel words,* that is words that *are not found* in the source document, as well as the capacity to adopt OOV words. The related results are provided in Table 3. We observe that ACVI increases the number of generated novel words by 7 times compared to baseline ASA. In a similar vein, ACVI appears to help the model better cope with OOV words.

Below, we provide some indicative examples of summaries produced by a pointer-generator network with coverage, employing the ACVI mechanism. We also show what the initial document has been, as well as the available reference summary used for quantitative performance evaluation. In all cases, we annotate OOV words in italics, we highlight novel words in purple, we show contextual understanding in bold, while article fragments also included in the generated summary are highlighted in blue.

Table 20: Abstractive Document Summarization: ROUGE scores on the test set

Method	ROUGE			METEOR	
	1	2	L	Exact Match	+ stem/syn/para
abstractive model* [6]	35.46	13.30	32.65	-	-
seq2seq with ASA (150K vocabulary)	30.49	11.17	28.08	11.65	12.86
seq2seq with ASA (50K vocabulary)	31.33	11.81	28.83	12.03	13.20
pointer-generator (ASA)	36.44	15.66	33.42	15.35	16.65
pointer-generator + coverage (ASA)	39.53	17.28	36.38	17.32	18.72
pointer-generator + ACVI	**39.96**	**17.41**	**36.40**	**16.68**	**17.84**
pointer-generator + ACVI + coverage	**42.71**	**19.24**	**39.05**	**18.47**	**20.09**
lead-3 baseline [11]	40.34	17.70	36.57	20.48	22.21
lead-3 baseline* [22]	39.2	15.7	35.5	-	-
extractive model* [22]	39.6	16.2	35.3	-	-

*Models and baselines in the top section are abstractive, while those in the bottom section are extractive. Those marked with * were trained and evaluated on the* anonymized CNN/Daily Mail *dataset.*

Table 21: Abstractive Document Summarization - Example 223.

0.99|X|

Article

lagos , nigeria -lrb- cnn -rrb- a day after winning nigeria 's presidency , muhammadu buhari told cnn 's christiane amanpour that he plans to aggressively fight corruption that has long plagued nigeria and go after the root of the nation 's unrest . buhari said he 'll " rapidly give attention " to curbing violence in the northeast part of nigeria , where the terrorist group boko haram operates . by cooperating with neighboring nations chad , cameroon and niger , he said his administration is confident it will be able to thwart criminals and others contributing to nigeria 's instability . for the first time in nigeria 's history , the opposition defeated the ruling party in democratic elections . buhari defeated incumbent goodluck jonathan by about 2 million votes , according to nigeria 's independent national electoral commission . the win comes after a long history of military rule , coups and botched attempts at democracy in africa 's most populous nation . in an exclusive live interview from abuja , buhari told amanpour he was not concerned about reconciling the nation after a divisive campaign . he said now that he has been elected he will turn his focus to boko haram and " plug holes " in the " corruption infrastructure " in the country . " a new day and a new nigeria are upon us , " buhari said after his win tuesday . " the victory is yours , and the glory is that of our nation . " earlier , jonathan phoned buhari to concede defeat . the outgoing president also offered a written statement to his nation . " i thank all nigerians once again for the great opportunity i was given to lead this country , and assure you that i will continue to do my best at the helm of national affairs until the end of my tenure , " jonathan said . " i promised the country free and fair elections . (...)

Reference Summary

muhammadu buhari tells cnn 's christiane amanpour that he will fight corruption in nigeria . nigeria is the most populous country in africa and is grappling with violent boko haram extremists . nigeria is also africa 's biggest economy , but up to 70 % of nigerians live on less than a dollar a day .

Generated Summary

muhammadu buhari talks to cnn 's christiane amanpour **about the nation 's unrest** . for the first time in nigeria , opposition defeated incumbent goodluck jonathan by about 2 million votes. *buhari* : " the victory is yours , and the glory is that of our nation "

Table 22: Abstractive Document Summarization - Example 89.

0.99|X|

Article

lrb- cnn -rrb- eyewitness video showing white north charleston police officer michael slager shooting to death an unarmed black man has exposed discrepancies in the reports of the first officers on the scene . slager has been fired and charged with murder in the death of 50-year-old walter scott . a bystander 's cell phone video , which began after an alleged struggle on the ground between slager and scott , shows the five-year police veteran shooting at scott eight times as scott runs away . scott was hit five times . if words were exchanged between the men , they 're are not audible on the tape . it 's unclear what happened before scott ran , or why he ran . the officer initially said that he used a taser on scott , who , slager said , tried to take the weapon . before slager opens fire , the video shows a dark object falling behind scott and hitting the ground . it 's unclear whether that is the taser . (...)

Reference Summary

more questions than answers emerge in controversial s. c. police shooting . officer michael slager , charged with murder , was fired from the north charleston police department .

Generated Summary

video **shows** white north charleston police officer michael *slager* shooting to death . *slager* has been charged with murder in the death of 50-year-old walter scott . the video shows a dark object falling behind scott and hitting the ground .

7.4.2 Video Captioning

Our evaluation of the proposed approach in the context of a VC application is based on the Youtube2Text video corpus [134]. This corpus is appropriate for addressing the problem of training and evaluating an automatic video description generation model. It comprises 1,970 video clips, each associated with multiple natural language descriptions. This results in a total of approximately 80,000 video / description pairs; the used vocabulary comprises approximately 16,000 unique words. The constituent topics cover a wide range of domains, including sports, animals and music. We split the available dataset into a training set comprising the first 1,200 video clips, a validation set composed of 100 clips, and a test set comprising the last 600 clips in the dataset.

We preprocess the available descriptions with the *wordpunct tokenizer* from the NLTK toolbox[10]. We do not perform any other type of preprocessing, such as lowercasing and rare word elimination. After the said preprocessing, the number of unique words reduces to 15,903. On the other hand, to reduce the entailed memory requirements, we process only the first 240 frames of each video. To obtain some initial video frame descriptors, we employ a pretrained GoogLeNet CNN [135] (implementation provided in Caffe [136]). Specifically,

[10] `http:/s/www.nltk.org/index.html`.

Table 23: Abstractive Document Summarization - Example 1305.

0.99|X|

Article

andy murray came close to giving himself some extra preparation time for his wedding next week before ensuring that he still has unfinished tennis business to attend to . the world no 4 is into the semi-finals of the miami open , but not before getting a scare from 21 year-old austrian dominic thiem , who pushed him to 4-4 in the second set before going down 3-6 6-4 , 6-1 in an hour and three quarters . murray was awaiting the winner from the last eight match between tomas berdych and argentina 's juan monaco . prior to this tournament thiem lost in the second round of a challenger event to soon-to-be new brit aljaz bedene . andy murray pumps his first after defeating dominic thiem to reach the miami open semi finals . muray throws his sweatband into the crowd after completing a 3-6 , 6-4 , 6-1 victory in florida . murray shakes hands with thiem who he described as a ' strong guy ' after the game . (...)

Reference Summary

british no 1 defeated dominic thiem in miami open quarter finals . andy murray celebrated his 500th career win in the previous round . third seed will play the winner of tomas berdych and juan monaco in the semi finals of the atp masters 1000 event in key biscayne

Generated Summary

the world no 4 is into the semi-finals of the miami open . ***murray*** **is still ahead of his career through the season** . **andy** *murray* was awaiting the winner from the last eight match . *murray* throws his sweatband into the crowd after a 6-4 6-1 victory in florida .

Table 24: Abstractive Document Summarization - Example 1710.

0.99|X|

Article

steve clarke afforded himself a few smiles on the touchline and who could blame him ? this has been a strange old season for reading , who are one win away from an fa cup semi-final against arsenal but have spent too long being too close to a championship relegation battle . at least this win will go some way to easing that load . they made it hard for themselves , but they had an in-form player in jamie mackie who was able to get the job done . he put reading in front in the first half and then scored a brilliant winner just moments after chris o'grady had levelled with a penalty -- one of the only legitimate chances brighton had all night , even if clarke was angry about the decision . reading frontman jamie mackie fires the royals ahead against brighton in tuesday 's championship fixture . mackie -lrb- centre -rrb- is congratulated by nathaniel chalobah and garath mccleary after **netting** reading 's opener . reading -lrb- 4-1-3-2 -rrb- : federici ; gunter , hector , cooper , chalobah ; akpan ; mcleary , williams -lrb- keown 92 -rrb- , robson-kanu -lrb- pogrebnyak 76 -rrb- ; blackman , mackie -lrb- norwood 79 -rrb- . subs not used : cox , yakubu , andersen , taylor . scorer : mackie , 24 , 56 . booked : mcleary , pogrebnyak . brighton -lrb- 4-3-3 -rrb- : stockdale ; halford , greer , dunk , bennett ; ince -lrb- best 75 -rrb- , kayal , forster-caskey ; ledesma -lrb- bruno 86 -rrb- , o'grady , lualua . subs not used : ankergren , calderon , hughes , holla , teixeira . **scorer :** o'grady -lrb- pen -rrb- , 53 . booked : ince , dunk , bennett , greer . ref : andy haines . attendance : 14,748 . ratings by riath al-samarrai . (...)

Reference Summary

reading are now 13 points above the championship drop zone . frontman jamie mackie scored twice to earn royals all three points . chris o'grady scored for chris hughton 's brighton from the penalty spot . niall keown - son of sportsmail columnist martin - made reading debut .

Generated Summary

jamie *mackie* opened the scoring against brighton in tuesday 's championship fixture . chris *o'grady* and *garath mccleary* both scored . **jamie *mackie* and *garath mccleary* were both involved in the game** .

Table 25: Abstractive Document Summarization: Novel words generation rate and OOV words adoption rate obtained by using pointer-generator networks with ASA or ACVI.

	ASA	ACVI
Rate of Novel Words	0.05	0.38
Rate of OOV Words Adoption	1.16	1.25

Table 26: Video Captioning: Performance of the considered alternatives.

Method	ROUGE: Valid. Set	ROUGE: Test Set	CIDEr: Valid. Set	CIDEr: Test Set
ASA	0.5628	0.5701	0.4575	0.421
ACVI	0.5968	0.5766	0.6039	0.4375

we use the features extracted at the *pool5/7x7_s1 layer* of this pretrained model. We select 24 equally-spaced frames out of the first 240 from each video, and feed them into the prescribed CNN to obtain a 1024 dimensional frame-wise feature vector. These are the visual inputs eventually presented to the trained models. We set the maximum number of encoding steps to 24, and the maximum number of decoding steps to 20.

We yield some comparative results by evaluating *seq2seq* models configured as described in Section 7.2.2; we use either ACVI or the conventional ASA mechanism. All the employed LSTMs comprise 1000-dimensional states. These are mapped to 100-dimensional features via the matrices $\boldsymbol{W}_h$ and $\boldsymbol{W}_s$ in Eq. (63). The postulated decoders are presented with 256-dimensional word embeddings, obtained in a fashion similar to our ADS experiments. We perform Dropout regularization of the employed LSTMs, as suggested in [137]; we use a dropout rate of 0.5.

Our quantitative evaluation is performed on the grounds of the ROUGE-L and CIDEr [138] scores, on both the validation set and the test set. The obtained results are depicted in Table 25; they show that our method outperforms the baseline by an important margin. Finally, we provide some indicative examples of the generated results in Figs. 17-24). These vouch for the capacity of our approach to detect salient visual semantics, as well as subtle correlations between related lingual terms (e.g., hamster–>small animal, a car drives –> several people drive, meat–>pork).

In the following, we provide some characteristic examples of video descriptions generated by a trained *seq2seq* model employing ACVI, and a rival *seq2seq* model using conventional ASA. In the captions of the figures that follow, we annotate minor deviations with blue color, e.g. replacing "a man" with "a woman"; we highlight synonyms with green color, and use red color to indicate major mistakes which imply wrong perception of the scene.

Figure 17: Video Captioning Example 1
ACVI: a man is firing a gun
ASA: a man is firing a gun
Reference Description: a man is firing a gun at targets

Figure 18: Video Captioning Example 2
ACVI: a woman is cutting a piece of pork
ASA: a woman is putting butter on a bed
Reference Description: someone is cutting a piece of meat

Figure 19: Video Captioning Example 3
ACVI: a small animal is eating
ASA: a small woman is talking
Reference Description: a hamster is eating

Figure 20: Video Captioning Example 4
ACVI: the lady poured the something into a bowl
ASA: a woman is cracking an egg
Reference Description: someone is pouring something into a bowl

Figure 21: Video Captioning Example 5
ACVI: a woman is riding a horse
ASA: a woman is riding a horse
Reference Description: a woman is riding a horse

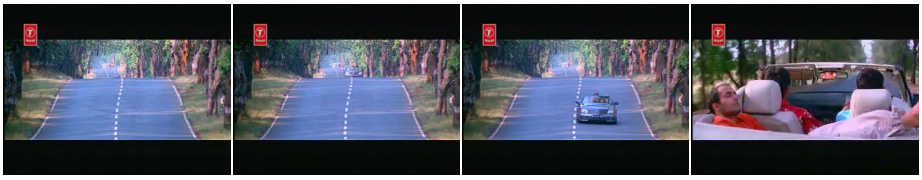


Figure 22: Video Captioning Example 6
ACVI: several people are driving down a street
ASA: a boy trying to jump
Reference Description: a car is driving down the road

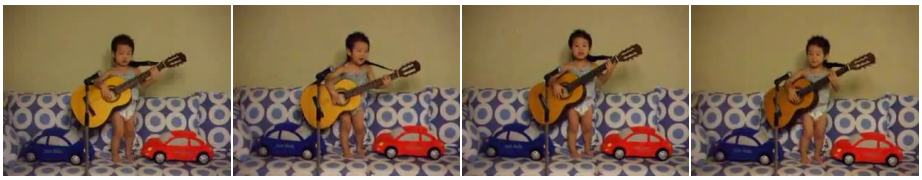


Figure 23: Video Captioning Example 7
ACVI: a man is playing the guitar
ASA: a high man is dancing
Reference Description: a boy is playing the guitar

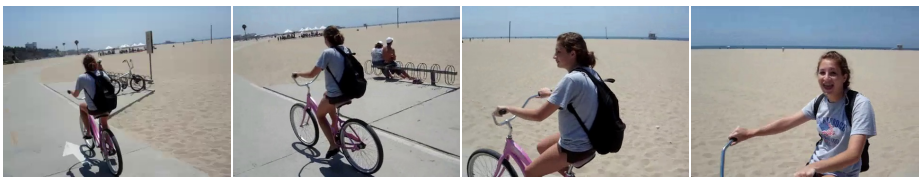


Figure 24: Video Captioning Example 8
ACVI: the man is riding a bicycle
ASA: a man rides a motorcycle
Reference Description: a girl is riding a bicycle

7.5 Conclusions

In this work, we cast the problem of context vector computation for *seq2seq*-type models into amortized variational inference. We made this possible by considering that the sought context vectors are latent variables following a Gaussian mixture posterior. Specifically, we assumed that each component density of the sought posterior is associated with one of the available source sequence encodings. That is, we considered that both the mean and the (diagonal) covariance matrix of this mixture posterior are parameterized by the postulated model encoders. On the same vein, we used the inferred attention probabilities associated with each encoding as the mixture component weights.

To illustrate our method, we implemented it in: (i) common *seq2seq* models with attention, comprising a BiLSTM encoder and a simple LSTM decoder; and (ii) the pointer-generator network with coverage, proposed in [86]. We evaluated the former on a VC task, and the latter on an ADS task; we used benchmark datasets in both cases. As we showed, our approach significantly outperforms the existing paradigm.

Finally, we underline that our proposed approach induces only negligible computational overheads compared to conventional ASA. Specifically, the only extra trainable parameters that our approach postulates are those of the MLPs employed in Eqs. (72)-(73); these are of extremely limited size compared to the overall model size, and correspond to merely few extra feedforward computations at inference time. Hence, it is unequivocal that our approach offers significant modeling performance benefits without undermining computational efficiency.

8 Future Endeavors

One research direction that we have not considered concerns the possibility of imposing nonelliptical or skewed distributions on the postulated latent variables. Indeed, many researchers in the past have shown that conventional generative models for sequential data, e.g. hidden Markov models, can yield significant benefits by considering nonelliptically contoured latent state densities, such as the multivariate normal inverse Gaussian (MNIG) distribution [139]. On the other hand, the efficacy and the potential advantages of introducing skewed latent variable assumptions in the context of DL models was empirically demonstrated in [113]. Nevertheless, such assumptions certainly come at the cost of increased computational complexity. Hence, we reckon that progressing beyond the elliptical class of distributions for formulating the assumptions of our model is a worthwhile future research direction. It might allow for even higher modeling performance, but requires novel theoretical developments to ensure retainment of the method's computational efficiency. Thus, these opportunities remain to be explored in our future research.

Another research path we consider is the extention of our work into multimodal tasks. Specifically, in the context of the EU funded project dubbed "aiD - aRTIFICIAL iNTELLIGENCE for the Deaf" (G.A. no 872139) we will have the opportunity to experiment in developing multimodal Sequence- to-Sequence models with Attention for text generation from Sign Language video sequences and speech generation from the obtained text. Specifically, we intend to build upon and extend the state-of-the-art Transformer [140] architecture in a fashion that allows for: (i) Detecting and processing lips, hands, arms, and face in video sequence as separate modalities; (ii) Fusing salient temporal dynamics across modalities in a way that allows for inferring interactions and structure; (iii) Combining the extracted temporal dynamics to uncover salient patterns by means of novel neural inference mechanisms using Amortized Variational Inference[141, 142].

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[2] M. Svensén and C. M. Bishop, "Robust Bayesian mixture modelling," *Neurocomputing*, vol. 64, pp. 235–252, 2005.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[4] D. A. Harville and D. R. Jeske, "Mean squared error of estimation or prediction under a general linear model," *Journal of the American Statistical Association*, vol. 87, no. 419, pp. 724–731, 1992.

[5] A. Cauchy, "Méthode générale pour la résolution des systemes dâéquations simultanées," *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

[6] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[7] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)," in *Doklady AN USSR*, vol. 269, 1983, pp. 543–547.

[8] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[9] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[11] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," vol. 323, pp. 533–536, 1986.

[13] G. W. Leibniz, "Memoir using the chain rule," in *cited in tmme 7:2&3 p 321-332, 2010*, 1676.

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[15] J. Sietsma and R. J. F. Dow, "Creating artificial neural networks that generalize," vol. 4, pp. 67–79, 1991.

[16] B. Poole, J. Sohl-Dickstein, and S. Ganguli, "Analyzing noise in autoencoders and deep networks," *arXiv preprint arXiv:1406.1831*, 2014.

[17] K.-C. Jim, C. L. Giles, and B. G. Horne, "An analysis of noise in recurrent neural networks: convergence and generalization," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1424–1438, Nov. 1996.

[18] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems*, 2011, pp. 2348–2356.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015, pp. 448–456.

[20] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen [in german] diploma thesis," *TU Münich*, 1991.

[21] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.(2001)," *Cited on*, p. 114, 2001.

[22] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[23] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks."

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[26] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[27] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[28] ——, "Gated feedback recurrent neural networks," in *International Conference on Machine Learning*, 2015, pp. 2067–2075.

[29] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International Conference on Machine Learning*, 2015, pp. 2342–2350.

[30] G. Chrupała, A. Kádár, and A. Alishahi, "Learning language through pictures," *arXiv preprint arXiv:1506.03694*, 2015.

[31] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[32] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate."

[33] S. Das, C. L. Giles, and G.-Z. Sun, "Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory," in *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, 1992, p. 14.

[34] J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," *Neural Computation*, vol. 4, no. 1, pp. 131–139, 1992.

[35] S. Haykin and R. Lippmann, "Neural networks, a comprehensive foundation," *International journal of neural systems*, vol. 5, no. 4, pp. 363–364, 1994.

[36] S. Haykin, "Neural networks: a comprehensive foundation, 1999," *Mc Millan, New Jersey*, 2010.

[37] J. Weston, S. Chopra, and A. Bordes, "Memory networks."

[38] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines."

[39] E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom, "Learning to transduce with unbounded memory."

[40] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "Weakly supervised memory networks," *CoRR, abs/1503.08895*, vol. 2, 2015.

[41] S. Sukhbaatar, J. Weston, R. Fergus *et al.*, "End-to-end memory networks," in *Advances in neural information processing systems*, 2015, pp. 2440–2448.

[42] A. Joulin and T. Mikolov, "Inferring algorithmic patterns with stack-augmented recurrent nets," in *Advances in neural information processing systems*, 2015, pp. 190–198.

[43] W. Zaremba and I. Sutskever, "Reinforcement learning neural turing machines-revised," *arXiv preprint arXiv:1505.00521*, 2015.

[44] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, p. 471, 2016.

[45] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*, 2016, pp. 1842–1850.

[46] ——, "One-shot learning with memory-augmented neural networks," *arXiv preprint arXiv:1605.06065*, 2016.

[47] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.

[48] R. Desimone and J. Duncan, "Neural mechanisms of selective visual attention," *Annual review of neuroscience*, vol. 18, no. 1, pp. 193–222, 1995.

[49] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[50] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention."

[51] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 577–585. [Online]. Available: http://papers.nips.cc/paper/5847-attention-based-models-for-speech-recognition

[52] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell."

[53] K. M. Hermann, T. KoÄiskÃœ, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend."

[54] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2692–2700.

[55] S. K. SÃžnderby, C. K. SÃžnderby, H. Nielsen, and O. Winther, "Convolutional lstm networks for subcellular localization of proteins."

[56] C. Raffel and D. P. W. Ellis, "Feed-forward networks with attention can solve some long-term memory problems."

[57] S. Wang and J. Jiang, "Machine comprehension using match-lstm and answer pointer."

[58] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou, "Gated self-matching networks for reading comprehension and question answering," 2017.

[59] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, "Structured attention networks."

[60] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," vol. 86, pp. 2278–2324, 1998.

[61] J. Peng, L. Bo, and J. Xu, "Conditional neural fields," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1419–1427. [Online]. Available: http://papers.nips.cc/paper/3869-conditional-neural-fields.pdf

[62] T. Do and T. Artieres, "Neural conditional random fields," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 177–184. [Online]. Available: http://proceedings.mlr.press/v9/do10a.html

[63] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch."

[64] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep structured output learning for unconstrained text recognition."

[65] L.-C. Chen, A. Schwing, A. Yuille, and R. Urtasun, "Learning deep structured models," in *International Conference on Machine Learning*, 2015, pp. 1785–1794.

[66] G. Durrett and D. Klein, "Neural crf parsing."

[67] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition."

[68] J. Tan, X. Wan, and J. Xiao, "Abstractive document summarization with a graph-based attentional neural model," in *Proc. ACL*, 2017.

[69] Q. Chen, X. Zhu, Z. Ling, S. Wei, and H. Jiang, "Distraction-based neural networks for document summarization."

[70] R. Nallapati, B. Zhou, C. N. dos santos, C. Gulcehre, and B. Xiang, "Abstractive text summarization using sequence-to-sequence rnns and beyond."

[71] J. Cheng and M. Lapata, "Neural summarization by extracting sentences and words."

[72] A. F. T. Martins and R. F. Astudillo, "From softmax to sparsemax: A sparse model of attention and multi-label classification."

[73] V. Niculae and M. Blondel, "A regularized framework for sparse and structured neural attention."

[74] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight, "Sparsity and smoothness via the fused lasso," vol. 67, pp. 91–108, 2005.

[75] H. D. Bondell and B. J. Reich, "Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with oscar," vol. 64, pp. 115–123, 2008.

[76] L. Hou, P. Hu, and C. Bei, "Abstractive document summarization via neural model with joint attention," in *National CCF Conference on Natural Language Processing and Chinese Computing*. Springer, 2017, pp. 329–338.

[77] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van MerriÃ«nboer, A. Joulin, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks."

[78] M. Kågebäck, O. Mogren, N. Tahmasebi, and D. Dubhashi, "Extractive summarization using continuous vector space models," in *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*, 2014, pp. 31–39.

[79] C. Li, X. Qian, and Y. Liu, "Using supervised bigram-based ilp for extractive summarization," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2013, pp. 1004–1013.

[80] K. Filippova and Y. Altun, "Overcoming the lack of parallel data in sentence compression," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 1481–1491.

[81] H. Saggion and T. Poibeau, "Automatic text summarization: Past, present and future," in *Multi-source, multilingual information extraction and summarization*. Springer, 2013, pp. 3–21.

[82] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[83] R. Nallapati, B. Xiang, and B. Zhou, "Sequence-to-sequence rnns for text summarization," 2016.

[84] S. Chopra, M. Auli, and A. M. Rush, "Abstractive sentence summarization with attentive recurrent neural networks," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 93–98.

[85] W. Zeng, W. Luo, S. Fidler, and R. Urtasun, "Efficient summarization with read-again and copy mechanism," *arXiv preprint arXiv:1611.03382*, 2016.

[86] C. Manning, "Get to the point: Summarization with pointer-generator networks. arxiv preprint," *arXiv preprint arXiv:1704.04368*, 2017.

[87] R. Nallapati, F. Zhai, and B. Zhou, "Summarunner: A recurrent neural network based sequence model for extractive summarization of documents." in *AAAI*, 2017, pp. 3075–3081.

[88] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[89] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li, "Modeling coverage for neural machine translation," *arXiv preprint arXiv:1601.04811*, 2016.

[90] S. Russel and P. Norvig, "Artificial intelligence: A modern approach, 2003," *EUA: Prentice Hall*, vol. 178.

[91] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend," in *Advances in Neural Information Processing Systems*, 2015, pp. 1693–1701.

[92] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," *Text Summarization Branches Out*, 2004.

[93] M. Denkowski and A. Lavie, "Meteor universal: Language specific translation evaluation for any target language," in *Proceedings of the ninth workshop on statistical machine translation*, 2014, pp. 376–380.

[94] M. Bayes, "An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs philosophical transactions, 53: 370–418," *URL http://rstl. royalsocietypublishing. org/content/53/370. short, http://rstl. royalsocietypublishing. org/content/53/370. full. pdf+ html*, 1763.

[95] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *arXiv preprint arXiv:1505.05424*, 2015.

[96] H. Attias, "A variational baysian framework for graphical models," in *Advances in neural information processing systems*, 2000, pp. 209–215.

[97] D. Jimenez Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, 2015.

[98] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[99] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[100] A. Graves, G. Wayne, and I. Danihelka, "Neural Turing machines," in *Proc. NIPS*, 2014.

[101] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *Proc. ICLR*, 2015.

[102] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Proc. NIPS*, 2015.

[103] A. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proc. ACL*, 2015.

[104] G. K. Zipf, *The psychology of language.* Houghton-Mifflin, 1935.

[105] D. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. ICLR'14*, 2014.

[106] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," in *Proc. NIPS'14*, 2014.

[107] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proc. ICML*, 2014.

[108] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. ICML*, 2015.

[109] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proc. ICML*, 2015.

[110] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul, "An introduction to variational methods for graphical models," in *Learning in Graphical Models*, M. Jordan, Ed. Dordrecht: Kluwer, 1998, pp. 105–162.

[111] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[112] A. Kosinski, "A procedure for the detection of multivariate outliers," *Computational Statistics and Data Analysis*, vol. 29, pp. 145–161, 1999.

[113] H. Partaourides and S. P. Chatzis, "Asymmetric deep generative models," *Neurocomputing*, vol. 241, pp. 90–96, 2017.

[114] C. Tsallis, "Possible generalization of Boltzmann-Gibbs statistics," *J. Stat. Phys.*, vol. 52, pp. 479–487, 1998.

[115] A. Sousa and C. Tsallis, "Student's t- and r-distributions: Unified derivation from an entropic variational principle," *Physica A*, vol. 236, pp. 52–57, 1994.

[116] C. Tsallis, R. S. Mendes, and A. R. Plastino, "The role of constraints within generalized nonextensive statistics." *Physica A*, vol. 261, pp. 534–554, 1998.

[117] J. Naudts, "Deformed exponentials and logarithms in generalized thermostatistics," *Physica A*, vol. 316, pp. 323– 334, 2002.

[118] ——, "Generalized thermostatistics and mean-field theory," *Physica A*, vol. 332, pp. 279–300, 2004.

[119] ——, "Estimators, escort proabilities, and ϕ-exponential families in statistical physics," *Journal of Inequalities in Pure and Applied Mathematics*, vol. 5, no. 4, 2004.

[120] S. P. Chatzis and D. Kosmopoulos, "A Variational Bayesian Methodology for Hidden Markov Models utilizing Student's-t Mixtures," *Pattern Recognition*, vol. 44, no. 2, pp. 295–306, Feb. 2011.

[121] S. Chatzis, D. Kosmopoulos, and T. Varvarigou, "Signal modeling and classification using a robust latent space model based on t distributions," *IEEE Trans. Signal Processing*, vol. 56, no. 3, March 2008.

[122] ——, "Robust sequential data modeling using an outlier tolerant hidden Markov model," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 31, no. 9, pp. 1657–1669, 2009.

[123] G. McLachlan and D. Peel, *Finite Mixture Models*. New York: Wiley Series in Probability and Statistics, 2000.

[124] N. Ding, S. N. Vishwanathan, and Y. Qi, "t-divergence based approximate inference," in *Proc. NIPS*, 2011.

[125] H. Attias, "A variational Bayesian framework for graphical models," in *Proc. NIPS'00*, 2000.

[126] C. Liu and D. Rubin, "ML estimation of the t distribution using EM and its extensions, ECM and ECME," *Statistica Sinica*, vol. 5, no. 1, pp. 19–39, 1995.

[127] D. Andrews and C. Mallows, "Scale mixtures of normal distributions," *J. Royal Stat. Soc. B*, vol. 36, pp. 99–102, 1974.

[128] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *JMLR*, vol. 12, pp. 2121–2159, 2010.

[129] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. AISTATS*, 2010.

[130] J. Weston, A. Bordes, S. Chopra, and T. Mikolov, "Towards AI-complete question answering: A set of prerequisite toy tasks," in *Proc. ICLR*, 2016.

[131] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, "Ladder variational autoencoders," in *Proc. NIPS*, 2016.

[132] G. McLachlan and D. Peel, *Finite Mixture Models*. New York: Wiley Series in Probability and Statistics, 2000.

[133] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015.

[134] L. Yao, A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville, "Describing videos by exploiting temporal structure," in *Proc. ICCV*, 2015.

[135] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. CVPR*, 2015.

[136] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," arXiv:1408.5093, 2014.

[137] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," in *Proc. ICLR*, 2015.

[138] R. Vedantam, C. L. Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation," in *Proc. CVPR*, 2015.

[139] S. Chatzis, "Hidden Markov Models with Nonelliptically Contoured State Densities," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2297–2304, Dec. 2010.

[140] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need."

[141] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. NIPS*, 2013.

[142] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. ICML*, 2015.