Question Answering with Deep Learning

Alex Auvolat
alex.auvolat@ens.fr

Thomas Mesnard
thomas.mesnard@ens.fr

Étienne Simon
esimon@esimon.eu

February 29, 2016

Abstract

In this report, we describe our approach to the question answering problem by first reducing it to a reading comprehension task. Our models parse the question as well as a document likely to contain the answer we are looking for and generate their predictions from both of them. We combine common neural networks techniques with more advanced deep learning stemming from recent natural language processing research, and we provide the first publicly available implementation of an attention model applied to reading comprehension.

Contents

1 Problem

We became interested in the question answering problem in the scope of a Kaggle competition: The Allen AI Science Challenge. In this competition we are given a dataset of multiple choice questions and answers from a standardized 8th grade[1] science exam. Two examples of question which could have been found in the dataset[2] are:

> Which of the following is an example of a physical change but not a chemical change?
>
> A – A log gives off heat and light as it burns.
>
> B – A tree stores energy from the Sun in its fruit.
>
> C – A penny lost in the grass slowly changes color.
>
> **D** – A water pipe freezes and cracks on a cold night.

> Lichens are symbiotic organisms made of green algae and fungi. What do the green algae supply to the fungi in this symbiotic relationship?
>
> A – carbon dioxide
>
> **B** – food
>
> C – protection
>
> D – water

Our approach to solve this problem, is two folds: first we starts by doing a search of the question on Wikipedia to select an article likely to contain the answer we are looking for, then we apply a reading comprehension technique recently introduced in [11] which parses the selected article and ranks the possible answers accordingly. In the remaining of the report, we will mostly focus on the deep learning model used for reading comprehension.

Reading comprehension can be seen as a supervised learning task with two inputs: given a document d and a question q we want to model the conditional probability $p(a|d,q)$ of the possible answers a. Since we want to apply deep learning to this problem, we need large amount of data. Sadly, the lack of big dataset was a major limitation of previous works, as an example, the recently introduced Memory Networks [10] were first trained and evaluated on a synthetic dataset.

In [11], the authors propose a novel strategy to generate a large dataset of triplet (d, q, a) from news article. On the CNN website, news articles are accompanied by "story highlights", a bullet point list of small sentences summarising the article. These summaries are not important extract from the article, but actual abstractive rewriting of information contained in the article. For our problem, the question–answer pairs of the dataset were generated from these summaries by transforming them into Cloze [20] style questions: one of the entities found in the summary is hid and we ask the question of finding it. By following this strategy we were able to extract a 2.5Go dataset of 387329 triplets, which should be enough to train our models satisfactorily.

With this model, our goal is to comprehend the given document. However, as it is, the model will learn world information during training, for

[1]In a normal curriculum, students in 8th grade are 13 to 14 years old.

[2]Because of license restrictions, we can't reproduce questions from the actual dataset in this report.

Original Version	Anonymised Version
Lee Min-bok didn't laugh once when he watched "The Interview" the North Korea defector calls the Hollywood comedy "vulgar" admitting he couldn't even watch the whole film. Yet he is still sending thousands of copies across the border from South Korea to North Korea in balloons, determined his people will see the movie in which the leader Kim is assassinated on screen...	*@entity0* didn't laugh once when he watched "*@entity4*" the *@entity6* defector calls the *@entity8* comedy "vulgar" admitting he couldn't even watch the whole film. Yet he is still sending thousands of copies across the border from *@entity2* to *@entity6* in balloons, determined his people will see the movie in which the leader *@entity15* is assassinated on screen...
Lee Min-bok says he finds the movie vulgar, but sends it anyway	*@entity0* says he finds the movie vulgar, but sends it anyway

Figure 1: Example of anonymisation of the CNN article "Defector sends thousands of 'The Interview' DVDs to North Korea".

example if the model encounter the question "which country ran a nuclear test last month?", it might have learned that North Korea is the usual culprit and does not even need to parse the document to answer. To prevent this potentially harmful process, we anonymise the documents and questions. An example of anonymisation is provided in figure 1, we do not want the model to learn that Lee Min-bok finds the movie vulgar, we want it to be able to extract it from the document irregardless of the name of the person or its sentiment about the movie. Furthermore, the entities are randomly shuffled during training, so the model will see the anonymised version of figure 1 with a random permutation of its entities.

2 Architecture

Deep learning is an approach born in the 2000s with [5] which revamped neural networks. Neural networks are powerful predictors, by composing alternatively linear and non-linear operators they offer universal approximation [8] but they were often difficult to train. Deep learning developed techniques in order to train networks composing a high number of non-linearity.

In this section we present the basic bricks from which our models were built and how we composed them together in order to model $p(a|d,q)$.

2.1 Neural Language

Our problem involve a lot of text processing, however neural networks need to take distributed representation as input. A distributed representation is a many-to-many relationship between two types of representation (such as words and neurons). Each word is represented by many neurons. Each neuron participates in the representation of many words.

For natural language processing, the first time distributed representations were used somewhat successfully was in the neural language model introduced in 2003 by [22]. The idea is to embed words in a vector space, that is, to each word v we associate an embedding $L(v)$ which is a real valued vector of fixed dimension.

In order to do this, we have a lookup table of size $|V| \times d$ where $|V|$ is the size of the vocabulary and d the size of the embeddings. The i-th word of the vocabulary is represented by the i-th row of the matrix.

We can also see this as a normal neural network layer where we represent the input word as a one-hot vector (a binary vector where a single value is set to one), taking the dot product with the weight matrix is then equivalent to taking the i-th row of the matrix.

There are several ways to train the embedding matrix [17], but in our models it is simply learned as an additional parameter together with the rest of the model. The representation learned by the embedding matrix has been shown [14] to capture semantic information in an interesting way and this technique is now used in most deep learning application in natural language processing.

2.2 Softmax

Our model not only read words at its input, but also outputs words, thus we need a distributed representation at the end of the network too. When predicting a class (e.g. predicting one word), we can output as many values as the number of classes (e.g. $|V|$ values, one for each word) thus assigning a score to each class. Predicting the correct class would be equivalent to taking the maximum over these values. However, the max function is not differentiable, to circumvent this problem, we use the softmax function defined as follow:

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The exponential is used to make all values positives, the denominator then normalize the values such that $\sum_i \text{softmax}(x)_i = 1$.

The output of the softmax can be interpreted as a probability distribution, for example, when predicting the next word in sentence, the softmax assign a probability to each word in the vocabulary. The normalizer, force the computation of e^{x_i} for all i, which makes this function computationally expensive.

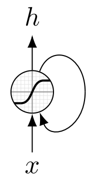
All of our models end with a softmax over the set of anonymised entities, which are taken to be the set of possible answers.

2.3 Recurrent Neural Networks

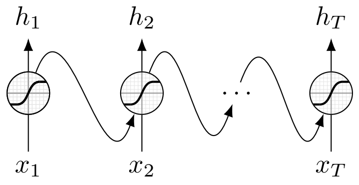
The word embedding technique enable us to represent words and take them as input, but our problem involve the processing of whole sentences and pages. Since [13], the dominant approach used in natural language processing to process variable-length sequences has been to use recurrent neural networks. A RNN is a neural network with a loop that carries a hidden state through several time steps as illustrated by figure 2, it can be summarized by the equation:

$$h_{t+1} = \tanh(Wx_t + Uh_t + b)$$

Looking at figure 2a, this might seems like a very shallow network, however once unrolled as in figure 2b, we can see that these networks actually compose a very high number of non-linearity thus being dubbed the deepest of all networks. Composing a high number of non-linearity



(a) Illustration of a RNN as a recurrent loop.



(b) Illustration of a RNN unrolled in time.

Figure 2: Schematic representation of recurrent neural network cells. Both figures represent the same thing, but the one on the right make the temporal relations explicit.

cause the gradient to dissipate, this is known as the vanishing gradient problem [21], the learning algorithm become unable to assign credit and the parameter can not be updated correctly. This makes RNN very difficult to train, especially when the sequence being processed is as long as it is in our problem.

2.4 Long Short-Term Memory

Long Short-Term Memory [18] are a type of RNN often used when processing long sequences since they are less susceptible to the vanishing gradient problem. LSTM redefine the recurrence of RNN by adding multiplicative gates as illustrated by figure 3, it is governed by the following set of equation[3]:

$$\begin{aligned}
x'_t &= [x_t, h_{t-1}] && \text{Recurrent input}\\
\tilde{c}_t &= \tanh(W_c x'_t + b_c) && \text{Cell candidate}\\
i_t &= \sigma(W_i x'_t + U_i c_{t-1} + b_i) && \text{Input gate}\\
f_t &= \sigma(W_f x'_t + U_f c_{t-1} + b_f) && \text{Forget gate}\\
c_t &= i_t \tilde{c}_t + f_t c_{t-1} && \text{New cell}\\
o_t &= \sigma(W_o x'_t + U_o c_t + b_o) && \text{Output gate}\\
h_t &= o_t \tanh(c_t) && \text{Hidden layer output}
\end{aligned}$$

One peculiarity of LSTM, is the presence of multiple gates i, f, and o, they are used as mask or mixing factor in the unit. LSTM units are interpreted as having an internal cell memory c_t which is an additional (internal) state alongside h_t and is used as input of the cell alongside x_t and h_{t-1}. When computing its activation, we first compute a cell candidate $\tilde{c}_t$ which is the potential successor to c_t. Then, the multiplicative gates come into play, the cell c_t is partially updated with a mix of c_{t-1} and $\tilde{c}_t$ controlled by the input and forget gates i_t and f_t. Finally, the output of the unit is masked by the output gate o_t[4].

It has been theorized [7] that the gates are what make LSTM so powerful, the multiplications allow the model to learn to control the flow of information in the unit thus counteracting the vanishing gradient problem.

[3] σ is the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

[4] Note that the output gate o_t has its value computed from the new cell value c_t instead of c_{t-1} as it is the case for i_t and f_t.

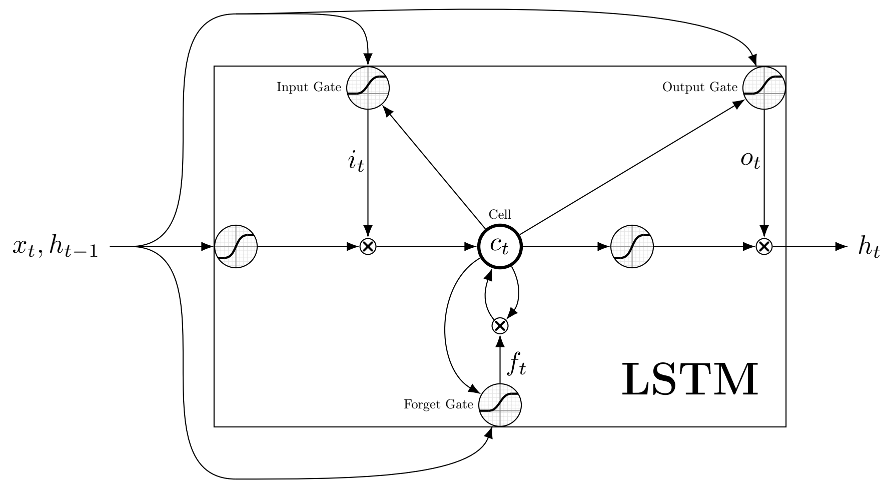


Figure 3: Long Short-Term Memory. The input x is fed to a tanh layer to produce a candidate value for the cell. The cell c is partially updated with the candidate, controlled by the input and forget gates i and f. The value of the cell is then fed to a tanh and masked by the output gate o. All gates use a sigmoid activation function in order to output a value between 0 and 1. The input and forget gate are fed the previous value of the cell c_{t-1}, whereas the output gate receive the new value c_t.

2.5 The Deep LSTM Reader

The first model we applied to the reading comprehension task is the Deep LSTM Reader presented in figure 4. This model runs a LSTM over a concatenation of the document and the question separated by a marker (`<SEP>`), this is a (cheap) way to enable interaction between the document and the question.

Even though LSTMs enable longer sequences to be processed, the path the gradient must go through between the cost at the output and the first word of the document is quite long, in order to cut down this distance we use a bidirectional LSTM, that is, we run the LSTM in both direction: the first one read the words in the natural order, the second one in the reverse order. This also enable the LSTM to select information in the document depending on the question it reads, indeed the forward LSTM can only selectively parse the question with its knowledge of the document (but not the opposite), thus it must entirely sum up the document into a single fixed-size vector not knowing which part of the document the question will be about[5].

Furthermore, in order to have a more powerful model, we stacked two bidirectional LSTMs on top of each other thus making the model deeper. To train efficiently this model, we added skip connections, that is we connected the second recursive layer directly to the input. Often used with deep RNNs, these additional connections enable a smoother training by adding a shorter path between the input and the second LSTM layer.

After the question and the document have been encoded with LSTMs,

[5]The opposite problem exists for the reverse LSTM, a more satisfying solution will be given with our second model.

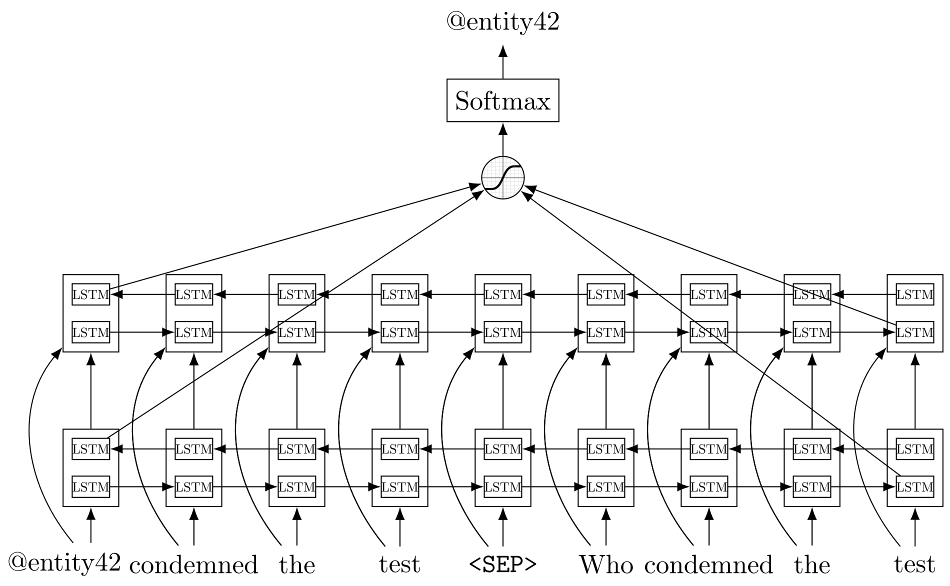


Figure 4: The Deep LSTM Reader: the concatenation document–question is parsed in both direction producing a first representation of the input pair. A second bidirectional LSTM is then run over this representation and the raw input to produce a higher level representation. The last states of each LSTM is then fed to a MLP with a softmax to produce the model's prediction.

we use the concatenation of the last hidden state of each LSTMs as a representation for the document–question pair and feed it to a 2-layers MLP. The activation function of the first layer is the usual tanh, the one of the last layer is the previously mentioned softmax function over the set of anonymised entities, giving us the prediction of the model.

2.6 The Attentive Reader

Even though bidirectional LSTMs enable us to process long sequences, the path between the middle of the document and the output is still quite long. Furthermore, at each time step LSTMs embed a variable-length sequence into a fixed-dimensional space this cause an unavoidable loss of information. In order to prevent this information loss, we must move away from model keeping a fixed-size vector at each time step. The next model we tried was inspired by recent advances in machine translation [3] and vision [12]: the Attentive Reader.

Similarly to the Deep LSTM Reader, the Attentive Reader start by encoding the input document and context, except that it encodes them separately (the interaction between the two being handled latter in the model). Once again, the encoding is done by a stack of bidirectional LSTMs. The difference is that, this time, we consider the whole sequence of hidden states of the document to be its representation (instead of looking at the final states only).

In order to compile this variable-length representation into a fixed-size vector for our output MLP, we use the so-called attention mechanism. The Attentive Reader is illustrated by figure 5 and can be summed up by

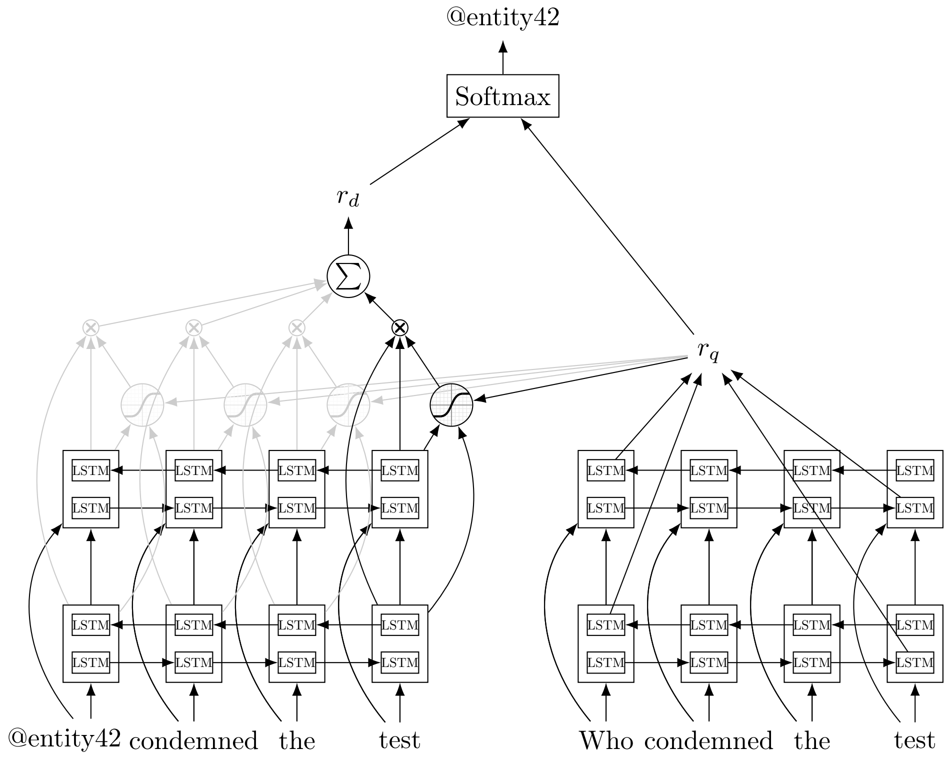


Figure 5: The Attentive Reader. For readability, the attention mechanism was only detailed for the last word of the document, and the softmax over the energies e was omitted. The document and question are first preprocessed by a stack of bidirectional LSTM in a similar way of what was done by the Deep LSTM Reader. For each word–context representation of the document, an energy is computed by a tanh layer which also has access to the question representation. The document representation is built as a sum of each word–context representation weighted by its energy. The prediction is built from the document and question representations as before.

the following set of equation:

$$\begin{array}{rcll}
{}^{d}\overrightarrow{h}^{1}_{t} & = & \mathrm{LSTM}(d_t, {}^{d}\overrightarrow{h}^{1}_{t-1}) & \text{1st forward LSTM processing of the document} \\
{}^{d}\overleftarrow{h}^{1}_{t} & = & \mathrm{LSTM}(d_t, {}^{d}\overleftarrow{h}^{1}_{t+1}) & \text{1st backward LSTM processing of the document} \\
{}^{d}\overrightarrow{h}^{2}_{t} & = & \mathrm{LSTM}(d_t, {}^{d}\overrightarrow{h}^{1}_{t}, {}^{d}\overrightarrow{h}^{2}_{t-1}) & \text{2nd forward LSTM processing of the document} \\
\ldots & = & \ldots & \\
m_t & = & [{}^{d}\overrightarrow{h}^{1}_{t}, {}^{d}\overleftarrow{h}^{1}_{t}, {}^{d}\overrightarrow{h}^{2}_{t}, {}^{d}\overleftarrow{h}^{2}_{t}] & \text{Representation of the } t\text{-th word of the document.} \\
r_q & = & [{}^{q}\overrightarrow{h}^{1}_{T}, {}^{q}\overleftarrow{h}^{1}_{1}, {}^{q}\overrightarrow{h}^{2}_{T}, {}^{q}\overleftarrow{h}^{2}_{1}] & \text{Representation of the question} \\
e_t & = & W_e \tanh(W_m m_t + W_r r_q + b_a) + b_e & \text{Energy of each document word} \\
a & = & \mathrm{softmax}(e) & \text{Weight of each document word} \\
r_d & = & \sum_t^T a_t m_t & \text{Representation of the whole document} \\
y & = & \mathrm{softmax}(W_d r_d + W_q r_q + b_y) & \text{Prediction}
\end{array}$$

Note that the question is handled as before since it is often short, the LSTMs handle it correctly and we can use it to control the attention on the document.

The interpretation of the attention mechanism is as follow: It gives an energy e_t to each word in the document, this energy is computed given the word and the context in which the word is used m_t as well as the question being asked r_q. Then, this energy is passed through a softmax giving a normalized weights to each word in the document: the attention a_t. The fixed-size representation of the document r_d is then computed as the weighted sum of each word–context representation. The prediction y is once again obtained by a softmax over the set of entities given the representation of the document and the question.

3 Training algorithm

Our models were trained with variants of the stochastic gradient descent algorithm, first of all, let's write the equation of the classic SGD for a parameter θ as follow:

$$g_{\theta^t} = \frac{\partial \ \mathrm{cost}(a^t, d^t, q^t, \theta^t)}{\partial \theta^t}$$

$$\theta^t \leftarrow \theta^{t-1} - \lambda g_{\theta^t}$$

Like previously said, all our models end with a softmax over the set of entities. Therefore, we are using the standard categorical cross entropy as a cost function, which, following a softmax, is equivalent to the negative log likelihood. Let a be the answers, d the documents, and q the questions:

$$\mathrm{cost}(a, d, q, \theta^t) = -\sum_i \mathbb{1}_{\{a=i\}} \log\left(\mathrm{model}_\theta(d, q)\right)_i = -\log\left(\mathrm{model}_\theta(d, q)\right)_a$$

A first (and classic) improvement over SGD is the use of momentum [16]. For each parameter θ, a velocity v_θ accumulates the gradients over time with a decay rate α. When updating a parameter, we use the velocity instead of using the raw gradient:

$$v_\theta^t \leftarrow \alpha v_\theta^{t-1} + \lambda g_{\theta^t}$$

$$\theta^t \leftarrow \theta^{t-1} - v_\theta^t$$

Momentum allows to speed up the training when succeeding gradients are in the same direction, and slow it down when they are in opposite directions.

3.1 RMSProp

RMSProp[19] is an algorithm that utilizes the magnitude of recent gradients to normalize the current one. The normalized gradients are then used with momentum as previous.

$$M_\theta^t \leftarrow \gamma M_\theta^{t-1} + (1-\gamma) g_{\theta^t}^2$$

$$v_\theta^t \leftarrow \alpha v_\theta^{t-1} + \frac{\lambda}{\sqrt{M_\theta^t}}\, g_{\theta^t}$$

$$\theta^t \leftarrow \theta^{t-1} - v_\theta^t$$

The name comes from the fact that the gradient is normalized by something akin to the Root Mean Square of the previous gradients. This normalization factor can be seen as the learning rate, thus RMSProp is an adaptive learning rate technique.

After experimentation, we found that RMSProp was what functioned best with our models, thus most of them were trained with it.

3.2 AdaDelta

For some model, we found that training initially with RMSProp then fine tuning by switching to AdaDelta [23] gave better results. It can be summarized by the following set of equation:

$$M_\theta^t \leftarrow \gamma M_\theta^{t-1} + (1-\gamma) g_{\theta^t}^2$$

$$\Delta_\theta^t \leftarrow -\frac{\sqrt{O_\theta^{t-1} + \varepsilon}}{\sqrt{M_\theta^t + \varepsilon}} g_{\theta^t}$$

$$O_\theta^t \leftarrow \gamma O_\theta^{t-1} + (1-\gamma) {\Delta_\theta^t}^2$$

$$\theta^t \leftarrow \theta^{t-1} + \Delta_\theta^t$$

If we reason in term of unit, the derivative g_{θ^t} can be seen as having unit 1/ unit of θ, thus SGD and momentum are adding two quantities with different units, AdaDelta tries to palliate to this by dividing by a gradient normalizer and multiplying by an update normalizer, thus having Δ_θ in "the good unit". Again, the goal is to protect against sudden change in the gradient by dividing the update by an average of past gradients (note that the numerator lag behind by a time step). A nice property of AdaDelta is that it is quite insensitive to hyperparameters setting, the algorithm will quickly converge for most values.

3.3 Regularization

The number of parameters of our models is quite high, in order to train them satisfactorily, we need to regularize them.

The use of a validation set as an early stopper indicator can be considered a basic form of regularization. Indeed, before implementing other forms of regularization this metric allowed us to stop the training before overfitting. Figure 6 shows an example of convergence plot for one of our early model that suffered from a lack of regularization, clearly early stopping is not enough.

The other form of regularization we used is dropout [6]. In dropout, the activation of neurons are randomly dropped and set to zeros with a

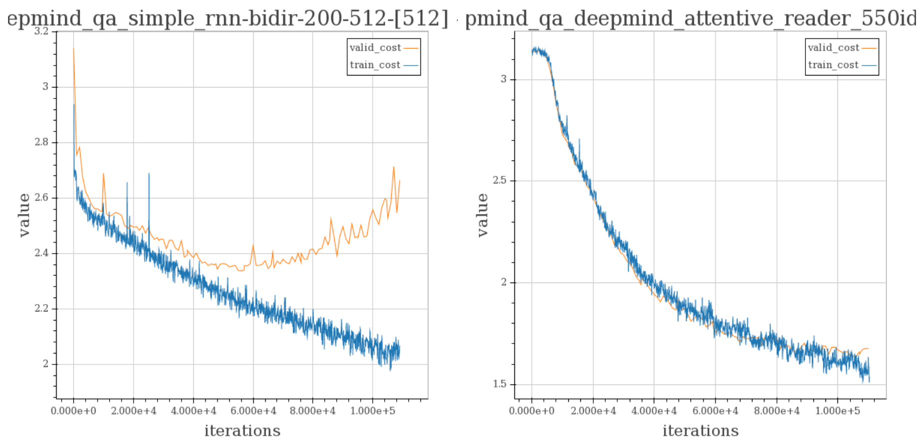


Figure 6: Negative log likelihood as a function of the number of iterations. On the left is the convergence for one of our early model (a single layer bidirectional LSTM) not using dropout or entity shuffling[6]. The model is clearly overfitting: the training cost continue to decrease while the validation cost increase. On the right is the convergence for our Attentive Reader which despite having more parameters, is less prone to overfitting thanks to the use of dropout and entity shuffling.

probability p. This has the effect of preventing a neuron from completely depending on the value of another neuron ("co-adaptation").

During training, a node is sometimes dropped out by the multiplication by a binomial, during testing, since there are more node (node are not dropped out at test time), we need to multiply each node by p to have a similarly valued input (which is equivalent to dividing them by p during training).

Dropout has been shown to be a cheap alternative to model ensembling, indeed, each possible subnetwork can be seen as a model in an ensemble.

Lastly, we tried various form of noise injection, either in the weight matrices [15] or in the activations [2]. Noise have been known to sometime dramatically increase generalization, however we did not observe any improvement.

After experimentation, we observed that dropout performed very well, we decided to combine it with early stopping only.

4 Experiments

Our model is publicly available on Github at:

`https://github.com/ejls/Deep-Question-Answering`

The code was written in Python with the help of Theano [4, 9], this library allows us to define a computation graph for our model and then

[6]As presented in the first section, the document and question are anonymised, during training the anonymised entities are shuffled, further preventing the network from overfitting the training set.

provide tools to perform automatic differentiation on it, thus greatly reducing the quantity of code to be written and the risk of mistake in the backpropagation routine. Furthermore, Theano enable us to use GPUs which greatly increased the speed of our algorithm. This is non-negligible since even with Nvidia Titan GPUs, training took around 4 days. We also used the blocks and fuel library [1] developed at MILA (ex-LISA) in Montreal alongside Theano, these libraries propose basic building blocks for deep learning model, once again making our work easier and allowing us to concentrate on model selection and hyperparameter tunning.

We extracted from the CNN website a 2.5Go dataset of 387329 triplets. For evaluation purpose, we divided this dataset into the three usual train, valid and test sets as follow:

Train	Valid	Test
380207	3924	3198

We tried several different hyperparameters settings, in the end the one with which we obtained the best performances were tuned as follow: For both models, the word embeddings were of dimension 200. For the Deep LSTM Reader, the size of the LSTM cells was 128 and the size of the second output MLP layer was 20. For the Attentive Reader, the size of the LSTM cells was 256 and the size of the second attention MLP layer was 100. The batch were of size 32 and were sorted by group of 20 into documents of increasing length. The reason for sorting the documents is that when training on a batch of variable-length sequences, the matrix processed on GPU has the same width as the longest sequence in the batch. Thus, by sorting sequences by length on a small number of consecutive batches, the gradient is not much affected but the training speed improve.

A convergence plot of the Attentive Reader is given in figure 6. In the end the models compared as follow:

Model	Validation error rate	Test error rate
Deep LSTM Reader	40.63%	38.93%
Attentive Reader	40.24%	38.38%

Despite being quite larger and slower to train, the Attentive Reader didn't performed much better. Similar results were already reported in [11], so this was to be expected. Nonetheless, an error rate under 40% means that our models were able to answer over 60% of the questions in the test set by reading the corresponding news article.

The additional computational expense of the attentive reader could be justified by the fact that it also produces an attention map on the document as presented in figure 7. In numerous fields, deep learning and neural networks are not used because they usually do not offer an explanatory output alongside their prediction. With attention mechanisms, it is a bit easier for us to give an explanation as to why our models generated a given output.

5 Conclusion

We obtained results which are at a state-of-start level on the reading comprehension task and provide the first publicly available implementation of an attention model for it. We also showed that on this task, the way

Document:

Gov. *@entity0* has long made it clear that he holds sharp disdain for *@entity3*. In fact, he finds it so unappealing, he'd rather drown himself than serve in *@entity6*. Speaking at the *@entity12* *@entity11* conference on saturday, the second term *@entity8* governor emphasized that he never plans on running for office in *@entity12* again. Most think *@entity6* is worst in their lifetime "the only job left for me to run for is *@entity17*, and let me just say this : I would rather die than be in the *@entity17*. Okay? I would be bored to death," *@entity0*, who's considering a 2016 presidential bid, said to laughs from the audience." Can you imagine me bangin' around that chamber with 99 other people? Asking for a motion on the amendment in the subcommittee? Forget it. It would be over, everybody. You'd watch me just walk out and walk right into the *@entity31* and drown. That'd be it." *@entity0* says again he'd love to run the *@entity35* *@entity0* gets compassionate as he eyes 2016

Question:

Gov. *@placeholder* says he's not running for office again in *@entity12*.

Answer: *@entity0*

Anonymisation table:

@entity0	Chris Christie	*@entity12*	New Jersey
@entity3	Washington	*@entity17*	U.S. Senate
@entity6	Congress	*@entity31*	Potomac River
@entity8	Republican	*@entity35*	Mets
@entity11	NAACP		

Figure 7: Visualization of the attention map on a document from our training set. The intensity of the red highlight is proportional to the attention weight, that is the weight that this word and its context was given in the representation of the document as a whole.

we implemented the attention mechanism doesn't produce a much better result than a deep stack of bidirectional LSTMs, however it provides an interesting explanatory output.

Further work could focus on a way to include world data in these models, indeed deep question answering models either have access to world data and rely to much on it, or try to ignore it completely as it is with the models we presented in this report.

References

[1] Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and Fuel: Frameworks for deep learning. *ArXiv e-prints*, June 2015.

[2] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, abs/1409.0473, 2014.

[4] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[5] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[6] Hinton E Geoffrey, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[7] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[8] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[9] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[10] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[11] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692, 2015.

[12] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[13] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[15] Alan F Murray and Peter J Edwards. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *Neural Networks, IEEE Transactions on*, 5(5):792–802, 1994.

[16] Boris Teodorovich Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[17] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[20] Wilson L Taylor. Cloze procedure: a new tool for measuring readability. *Journalism and Mass Communication Quarterly*, 30(4):415, 1953.

[21] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[22] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[23] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.