Using Deep Neural Networks for Automated Speech Recognition

Elie Michel

August 28th, 2015

Internship from April 27st to October 27st, 2015 in the Watson Speech and Language Technologies team at Interactions LLC under the supervision of Patrick Haffner.

1 Introduction

Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) have been studied for decades and are still addressing challenging problems whose variety requires the collaboration of a wide range of scientific domains. Recognizing words from an audio signal, and then understanding the meaning of the sentences they form mobilizes a lot of prior knowledge, even for the human brain.

Moreover, those technologies can benefit to a lot of products. They leverage the use of new hand free or remote ways to interact with devices or query databases, through *virtual assistants*. The commercial release of such assistants is very recent and many big companies invest in it[1]. They find promising applications in cars, home automation, wearable devices, etc.

But the reliability of those automated systems is still an issue. Even the state-of-the-art of fully automated speech recognition is unable to reach the human accuracy, especially in noisy environments, and detecting the intent is even harder.

To address this problem, Interactions has developed a method to couple both automated and human speech recognition known as Adaptive-Understanding[2], in which the machine can decide to ask for human help if it is not confident enough.

This way, the company has been able to develop a reliable product, but it has still a lot of interest in researching for more accurate systems since the use of human understanding is expensive.

Section 2 presents the company and shows why its organization requires a research team and what are the main challenges it addresses. It also briefly presents the team, which has an independent history. Then Section 3 explains more technically the state-of-the-art solutions used for speech and language processing. Section 4 focuses on a class of models whose use is growing in this pipeline, namely Deep Neural Networks. In Section 5, the tools and implementations available are described. Given the background of all the previous sections, section 6 presents my work and contributions during this internship, both scientifically and technically.

2 Presentation of the company

2.1 Overview

Founded in 2004, Interactions started with humans acting as mechanical turks to replace the ASR in a directed dialog system. While this sounds counter-intuitive, it provided extraordinary productivity im-

[1] Main examples of virtual assistants are *Siri* (Apple, October 2011), *Google Now* (Google, July 2012), *Cortana* (Microsoft, April 2014), *Echo* (Amazone, November 2014) and the very recently released *M* (Facebook, August 26th, 2015)

[2] http://www.interactions.net/product/adaptive-understanding/

provements over an explicit human-human dialog, while also delivering a much better accuracy than pure ASR systems. Thus it was seen as a good compromise for many large US companies to handle the first prompts of their customer service.

More recently, the company grew its business around the key idea of Adaptive-Understanding which couples both automated methods and human-based oracles to be able to target high accuracies and hence develop real applications of the state of the research in ASR and NLU.

2.2 Adaptive-Understanding

The overall idea of Adaptive-Understanding is to dynamically request a human oracle when the degree of confidence provided by automated system along with its output is too low.

The definition of *too low* is given by a target accuracy, while the trade-off is on the balance between the use of automated system or atomic recognition queries sent to human Intern Analysts. The evolution of the overall accuracy is measured to find the appropriate confidence threshold (See Figure 1).

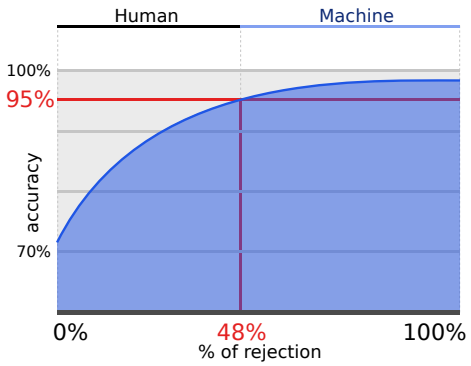

Figure 1: Accuracy of the final system as a function of the rejection rate (hand-drew curve). When there is no rejection, the accuracy is the one of a fully automated system. A rejection of 100% means that we use only humans. For any accuracy in between, a rejection rate can be found.

The company's whitepaper[3] reports that to reach a target accuracy of 95% on open ended questions, 52% of the input utterances is handled by the machine and so 48% is rejected to humans. Since the company's technology is under permanent development, those figures may have changed since then.

Note that the final error rate regroups both false positives from automation, where the machine output both a wrong answer and a high confidence score, and the human error, that is of course not null.

2.2.1 User benefits

Thanks to this idea, the imperfections of the state of the research does not reflect on the product itself, but on the maintenance price it costs to the company. This is the key that brings the research to a viable product.

This method is also particularly efficient to handle cases that are really hard for a machine like out-of-grammar responses due to proper nouns or requiring cultural knowledge.

Plus, waiting for an available human agent, what can be fatal to a good user experience, is not needed any more.

2.2.2 Human analysts benefits

Compared with a usual call center where human agents handle full conversations with users, many feel more comfortable with just having to handle single utterances. For instance, they do not need to fake kindness all the day face to users that themselves are not always kind. They do not require any explicit knowledge besides knowing the interface and a local mistake has less bad consequences. To motivate the agent, the whole process has been gamified.

[3] http://www.interactions.net/resources/whitepapers/#ufh-i-13423679-whitepaper-broadening-the-conversation

2.2.3 Company benefits

As the majority of cases is handled automatically, the company does not need as many human agents as with a all-human call center and so it is less expensive to maintain and more flexible.

2.2.4 Research benefits

One of the most interesting byproducts of this method is that it generates labeled data every day. Given how important datasets are in machine learning, this is really helpful.

Furthermore, once the Human Analyst setup is implemented, a part of it can be used for research and development.

2.3 Applications

The Adaptive-Understanding is used to assist both ASR and NLU and the main application is to maintain Customer Care call centers for other companies.

It builds tailored Virtual Assistants to help the user find answer to their questions related to the company in a natural way.

It is more than an interactive menu since the agent remembers any meaningful piece of information about the call to avoid prompting for an information that was a side answer to a previous question for instance.

2.4 Technical needs

Given its business model, the company has a deep interest in fostering research on ASR and NLU to get the automated part more accurate and so request less interventions of human analysts, who are more expensive than machines.

Many of the challenges it addresses are also research problems.

The ASR needs to be less sensitive to background noise and word-level noise such as “um”, “ah” or more robust to difficult caller accents. The NLU and intent detection needs to be more robust to complicated sentence constructions or out-of-vocabulary words.

Furthermore, there is a growing need for multilingual systems where the lack of data, compared to the size of English datasets, can be a problem.

For a given accuracy, other features also needs to be taken into account. The system has to be real time, so the models cannot be arbitrarily big. As for a lot of Machine Learning task, the training time can be really important. Given the huge amount of training data and the complexity of the problem, the time for an end-to-end training is many weeks!

For all those reasons, Interactions acquired in the end of the year 2014 the AT&T Watson Speech and Language Technologies (SLT) team[4].

2.5 The Watson Research Team

The essential structure of the Watson team dates from 1996, after AT&T spun off the infamous Bell Laboratories into Lucent Technologies, but many of its members where working in the formerly AT&T Bell Labs before and Watson platform relies on previous works.

Work for customer care before: Hom May I Help You, launched in 2001[5]

virtual assistant develop interface of things

Conclusion: Mix between dynamism of a young company and the expertise of an experimented team.

3 Speech and Language Technologies

Speech and Language Technologies regroup both speech analysis and synthesis, as shown on Figure 2,

[4] http://www.interactions.net/press-releases/interactions-closes-strategic-deal-acquire-att-watson-speech-recognition-natural-language-understanding-platform/

[5] http://www.corp.att.com/attlabs/reputation/timeline/01hmihy.html

and both can operate at several scales. In the following part I will focus on the analysis side.

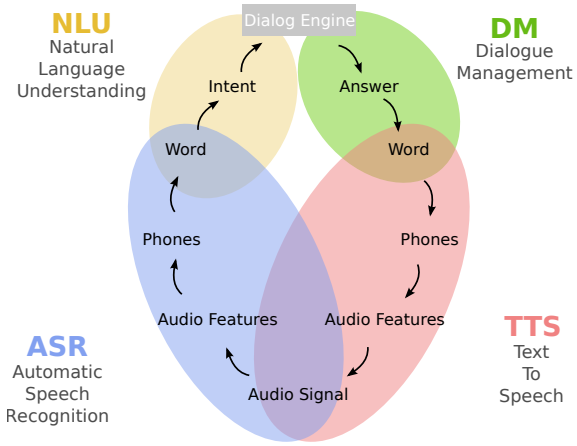

Figure 2: Overview of the Speech and Language Technologies. The left-hand part is speech analysis and the right-hand part is speech synthesis.

There are two main steps in speech analysis, namely Automated Speech Recognition (ASR) and Natural Language Understanding (NLU). The Subsection 3.1 presents briefly NLU and then the Subsection 3.2 presents ASR, which is what I worked on, in more details.

3.1 Natural Language Understanding

The role of NLU is to extract the user intent from a text. Sentences are parsed into semantic representations. Although it can be helped with additional information provided by the speech analysis such as word prominence, it does not assume that the text comes from speech and is hence trained separately.

The practical challenges are to be able to handle the multiple ambiguities of the natural language as well as the out-of-vocabulary words that can occur. Furthermore, a natural language grammar is too complex to be fully hand-crafted and rule-based system are at least helped unless replaced by learned models.

But a more fundamental issue is also the way to represent semantics and many different theories has been explored by linguistic and computing sciences.

It can be applied to a wide range of problems, like text summarizing, semantic search, and of course virtual assistants and chat bots.

3.2 Automatic Speech Recognition

ASR operates at the very lower level, taking the raw audio signal as input, and writing a transcript for it, which is basically a sequence of word.

Note that we could consider an augmented definition of word that embed additional information extracted from the audio signal like word prominence or emotion[6]. It may then be useful for the NLU.

In a more formal way, let us consider the input utterance of speech $\mathcal{S}$. The ASR task is then to determine the word W^* that is the more likely to be the transcript of $\mathcal{S}$:

$$W^* = \underset{W}{\operatorname{argmax}}\ P(W|\mathcal{S}) \tag{1}$$

The later probability is hard to determine by itself, so it is decomposed using Bayes' law:

$$P(W|\mathcal{S}) = \frac{P(\mathcal{S}|W)P(W)}{P(\mathcal{S})} \tag{2}$$

The quantity we are now looking for are easier to handle.

3.2.1 Language Model

$P(W)$ is the probability that the word W occurs. The most simple way to model it would be to measure the word statistics over a dataset. But more advanced methods are generally used, where the context is taken into account to influence the probability, using learned or rule-based grammars.

[6] {reference needed}

The model can hence be represented as a probabilistic automaton, a.k.a. Weighted Finite State Acceptor (Mohri, Pereira, and Riley 2002) where the transitions are labeled with words and weighted with probabilities, as shown by the example in Figure 3.

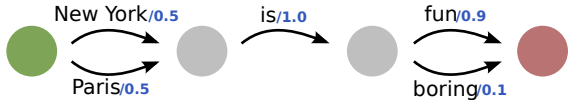

Figure 3: Example of minimalistic language model. When the speech recognizer is in a given state, the model gives a probability of hearing a word and the new internal state in which this operation would lead the recognizer to. Transition with null weight are not represented.

3.2.2 Acoustic Model

$P(\mathcal{S}|W)$ would be the probability of hearing the signal $\mathcal{S}$ if we knew that the word that the speaker intended to pronounce was W.

This translation from words to sounds is modeled though atomic speech units called phones (See (Livescu, Fosler-Lussier, and Metze 2012) for a survey of sub-word models). The sequence of words is converted into a sequence of phones using a pronunciation dictionary, and the emission of audio signal from phones is modeled as a Hidden Markov Model (HMM).

3.2.2.1 Pronunciation Dictionary The pronunciation dictionary specifies one or many pronunciations for each word (See Figure 4). It can be generated from the spelling using specific rules in a first step and then be bootstrapped to learn refined pronunciations from the training set.

Typically, the number of phones is around 30 to 100 for English, and they should be invariant to the presence of background noise and the properties of the voice like its pitch.

```
aware..... ax w ey r
awareness.. ax w ey r n ih s
away...... ax w ey
awe....... ao
awesome.... ao s ax m
awful..... ao f ax l
awfully.... ao f ax l iy
awfulness.. ao f ax l n ih s
```

Figure 4: Extract of the pronunciation dictionary provided with Switchboard corpus. The phones are represented by short names close to how they sound. Note that we have here only one pronunciation per word.

As well as for the LM, the AM dictionary fits well into the framework of Finite State Transducers (FST) (Mohri, Pereira, and Riley 2002). This provides a way to reflect the multiplicity of pronunciations of a word and its probabilities and also enable the possibility to compose AM and LM into a single FST to improve the decoding speed (See Section 3.2.4).

The whole model can hence be represented as a single FST along with the GMM of DNN used for emission prediction.

3.2.2.2 Hidden Markov Model Given this phonetic dictionary, the generation of speech can be modeled as a Hidden Markov Model (HMM), as shown in Figure 5. This reflects the fact that we cannot measure the phones themselves but an audio signal – which are phenomena resulting from those phones being pronounced – that a given observable speech frame is likely to be caused by several phones.

In fact, some refinement of the phone decomposition is performed. A phone is still of a higher level than the emitted sound and so sub-phone states are actually used (see Figure 6).

Furthermore, the emission is modeled using context phones since the actual pronunciation of a phone usually depends on the phones pronounced before and after it. The number of HMM state is then so high that they need to be grouped by similarities, using decision trees (Young, Odell, and Woodland 1994).

Self-loops are finally added to each state of the FST

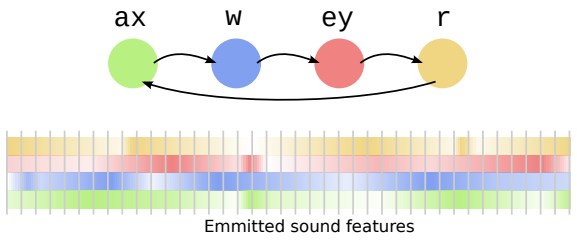

Figure 5: Extract of HMM. The dots are hidden states labeled with phones. The arrows represent the possible sequences of phones (here the dictionary contains only one word). The feature bands bellow correspond to the same color hidden state and roughly represent the continuous emission of the states

Figure 6: Decomposition of phones into phone states on a simple example.

resulting of all those techniques to encode the difference of time scale between the sequence of phones and the sequence of speech frames.

3.2.2.3 Emission Probabilities Distribution Functions We need, to complete this model, a measure of the probability that a given frame has been emitted by a given phone. This is particularly difficult since the emission is a continuous signal and so each phone has its own continuous Probability Distribution Function (PDF) defining what it emits.

During a long time, the state-of-the-art[7] model used to define the PDFs of HMM states emission was Gaussian Mixture Model (GMM). The probability that a state s emit a feature vector X is the weighted sum of covariant Gaussians:

$$P(X|s) = \sum_{k=1}^{M} \alpha_{ks} \mathcal{N}(X, \mu_k, \sigma_{ks}) \tag{3}$$

[7] {reference needed}

where α, μ and σ are learned parameters.

The new state-of-the-art speech recognition systems now use Deep Neural Network. A neural network fed with the feature vectors and outputting the current HMM state. See (G. Hinton et al. 2012) for an history and reflection about this evolution. For more information about DNNs, see Section 4.

Another method that still gives good result is the joint use of GMM and DNN, training the GMM on the output of a DNN (Hermansky, Ellis, and Sharma 2000).

3.2.3 Prior probability

Note that $P(\mathcal{S})$ is supposed to be constant, which means that any sound signal is equally likely to happen. This is a reasonable assumption given that the input signal is actually provided as normalized features.

Those features also provide more invariance to the speaker, the presence of noise, etc. A lot of different types of features have been investigated (Dimitriadis, Maragos, and Potamianos 2005), but the use of DNNs led some people to even completely get rid of the feature engineering (Deng et al. 2010), considering that the first layers of the network have already learned the best features.

3.2.4 Decoding

The input of the ASR task is a sequence of speech frame, so a sequence of HMM's emission states. The determination of the sequence of hidden states that is the most likely to have generated those emissions is known as HMM *decoding* and is performed using the Viterbi algorithm (Viterbi 1967).

The result is a sequence of HMM states then converted. Since those states result from the composition of the previous FSTs (Language Model, Pronunciation dictionary), it then output a sequence of words.

Finally, the main challenges of ASR are improving the accuracy, reducing the training time and increasing

the runtime speed.

4 Deep Neural Networks

Neural Networks, lately refurbished under the name of Deep Learning, are a very generic Machine Learning modeling tool that is widely used, especially on tasks that used to be known as signal processing, replacing heavy feature engineering work. And speech processing is one of them.

While inspired by and named after a model of real neuron interactions, a neural network has to remain a useful computing tool and a tractable mathematical object.

Its definition is very general since it is more a framework than a single object and a lot of variants can be found.

4.1 General Definition

A neural network is an acyclic weighted oriented graph $\mathcal{N}$, where a node n is called *neuron*, or *unit*, and is associated a activation function f_n.

A network also defines a subset of neurons as the *input layer* I and another one is the *output layer* O. The term *layer* comes from the typical network architectures.

4.1.1 Running

Running a neural network $\mathcal{N}$ on an input valuation $X = (X_n, n \in I)$ is the operation consisting in associating to each neuron n a scalar value a_n called *activity* by applying recursively the following rules:

- If $n \in I$, then $a_n = X_n$
- If not, a_n is defined by the sources $i_1, \dots, i_k$ of the connections of $\mathcal{N}$ whose n is a destination, also known as n's input units, and the respective weights of those connections $w_{ni_1}, \dots, w_{ni_k}$ by:

$$a_n = f_n(w_{ni_1}, a_{i_1}, \dots, w_{ni_k}, a_{i_k}) \tag{4}$$

Note that in a vast majority of cases, (4) can be written as follow:

$$a_n = f_n(\sum_{j=1}^{k} w_{ni_j} a_{i_j}) \tag{5}$$

Thus an output valuation $Y = (a_n, n \in O)$ is defined.

4.1.2 Learning

The network without its weights is described statically and considered as an input. On the contrary, the weights are determined during a training step.

This is why although weights could be included into the f_n functions, they are not. This is to clearly distinguish static parameters from learned parameters.

The unweighted network, along with the training constraints, form the *architecture* of the network.

4.2 A more practical definition

While this definition is theoretically interesting, the practical representation of neural networks does not strictly reflect it. Both running and learning actually require the network to be topologically sorted to be efficient, since computing the valuation of a node relies on the valuation of its inputs.

Hence, they are grouped by topological distance to the network inputs (See Figure 7). Such groups are called *layers* $(\mathcal{L}_n)_n$. We can, without loss of generality in the functions $X \mapsto Y$ defined by the running operation, assume that all the inputs of a given node are in the same layer (if not, it is easy to add identity nodes in between).

The layer representation is very powerful since it enables to compute all the sums $\sum_{j=1}^{k} w_{ni_j} a_{i_j}$ in (5)

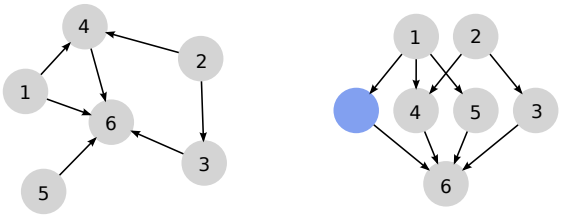

Figure 7: Equivalent representations of a neural network. The right-hand one is more computation-friendly although it contains one more node not to break the layer representation.

of a given layer $\mathcal{L}_n$ at the same time. They are indeed the results of the product of the weight matrix $W_n = (w_{ij}, i \in \mathcal{L}_{n-1}, j \in \mathcal{L}_n)$ by the activities $(a_i, i \in \mathcal{L}_{n-1})$ of the layer $n-1$.

This way, any linear algebra tool can be used to compute or analyze a neural network.

4.3 Typical architectures building blocks

Given this second definition of a network, it is convenient to describe an architecture by its layers.

4.3.1 Activation Function

An important parameter of a layer is the activation function of its units. It is worth noting that if this function is the identity, the weight matrices can be composed and the network is thus always equivalent to a single layer network. This is why the activation function is sometimes called *non-linearity* and is at the core of the power of representation of the networks. See Firgure 8 for examples.

A common interpretation of the way a DNN transforms a signal is that the activation of each layer is a higher-level non-linear representation of the previous one. Then the more layers the more abstraction can be encoded and learned.

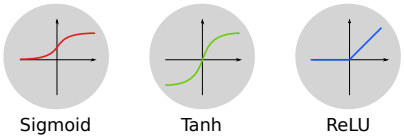

Figure 8: The most used non linearities are the *logistic sigmoid* ($\sigma(x) = \frac{1}{1+e^{-x}}$), the *hyperbolic tangent* and, more recently, the *Rectified Linear Units*, defined by $\text{ReLU}(x) = \max(0, x)$ (Nair and Hinton 2010)

The most basic network architecture is made of a sequence of layers where all the units have the same activation function, and all the *hidden layers* (layers which are neither input nor output) have the same size.

A different activation function can be used for the output layer. For instance, estimating the current hidden state of the ASR HMM requires to output probability (their logarithm, in fact), since the Viterbi algorithm actually explores many solutions, and so there is a need for a normalization.

Other applications such as extraction of bottleneck features requires an output distribution that is incompatible with the statistical behavior of the ReLU.

4.3.2 Weight sharing

Constraints can be applied to the space in which weights are picked. The most simple kind of constraint is *weight sharing*, which means that two or more connections can learn the same weight. It is mainly motivated by the need to ensure some invariance of the data processing.

Of course, the learning algorithm must be able to handle those constraints. In the common case of learning by gradient descent, weight sharing is performed by summing or averaging the error backpropagated though each connection. Generally, sharing weights reduces the number of parameters and hence eases the learning, although it does not improves runtime speed.

The main application of weight sharing is the Convolutional Neural Networks (CNN) (See Subsection 4.3.3).

It can also be a way to represent unfolded Recurrent Neural Networks (RNN) (See Subsection 4.3.4).

4.3.3 Convolutional Neural Networks

A convolutional layer behaves as if the same small network was applied locally, on a windows sliding over all the input which hence needs to be provided some geometrical organization. See Figure 9

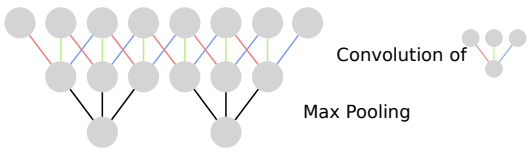

Figure 9: Illustration of a CNN. The colors on the connections represent the weight sharing. The second layer is a max pooling (Zhang et al. 2014) where the units get the maximum activation of their inputs. This is widely used with convolutional layers to ensure some invariance and subsample the data.

The idea of the CNN is to build a set of representations of the input which is invariant to the geometrical dimensions. It is especially useful when analyzing images since we need to recognize an object wherever it is on the image. Those applications have been used in the last state-of-the-art image classifiers like (Szegedy et al. 2014).

CNNs have also been applied on video (3D signal) or speech. In the case of speech, the convolution can be either applied on the temporal dimension only or on both temporal and spectral dimensions (Abdel-Hamid et al. 2012) (Saon et al. 2015).

Any sequential input can benefit of the robustness of CNN, even the discrete signal used in NLU (words or characters) (Kalchbrenner, Grefenstette, and Blunsom 2014) (Zhang and LeCun 2015).

Another advantage of CNNs is that they can benefit very efficiently from the hardware primitives implemented in common GPUs (Vasilache et al. 2014). This can be used both for training and runtime.

4.3.4 Recurrent Neural Networks

RNNs add to DNNs a whole new capability: *memory*. While a CNN learns invariant filters to apply at any point of the input signal, the RNN do the opposite by letting the whole network having a different behavior at any time.

The architecture of a RNN is not different from the one of a DNN, except that along with its usual input – the one provided by the training set and then by the runtime – the RNN is also fed with some part of the output of the previous run of itself.

Those new inputs and outputs are unsupervised and can be considered as some kind of memory. The main drawback is that the training must also learn this memory, and so generally need to perform the backpropagation through time (Werbos 1990).

RNNs provide a unique a way to handle varying size sequential inputs, as speech, textual or handwritten sentences. A challenge that RNNs does not solve by themselves is to map sequences of different size, like a speech utterance to its transcript. Additional methods has been developed like the CTC (Graves et al. 2006), used for ASR in (Hannun et al. 2014), then generalized as RNN transducer [Graves12]. Other methods include attention-based mechanism (Chorowski et al. 2014) or sentence embedding (Palangi et al. 2015), especially for translation.

RNNs also gave interesting results on sequence generation, using networks with no input but the recurrent state (Graves 2013).

A known issue of the training of such networks is the problem of *vanishing gradient* (Bengio, Simard, and Frasconi 1994). After some iterations of the backpropagation through time, the meaning of the gradient fades out and as a result the RNN is unable to learn long term dependencies.

The design of LSTMs (Hochreiter and Schmidhuber 1997) makes RNNs more robust by modeling a gate

system preventing the contributions of the gradient at steps where a memory has not been used. Other work like RNNs running at multiple timescales (Koutník et al. 2014) or Memory Networks (Weston, Chopra, and Bordes 2014) have also been recently presented.

Finally, RNNs can actually be applied on any recursive structure, such as syntactic trees (Tai, Socher, and Manning 2015). Any recurrent constructor of the structure on which the RNN is applied can use different weights as soon as it uses the same number of memory units.

4.4 Training Method

Another very important choice in the use of DNNs is the method used to train their weights. Basically, all the state-of-the-art methods more or less use Stochastic Gradient Descent (SGD), given some variations.

The efficiency of the SGD highly relies on the choice of some hyper-parameters and notably the learning rate. Tunning the learning rate is still an art that require intuition (Glorot and Bengio 2010). (Senior et al. 2013) even present an "empirical study" about it, while (Bottou 2012) published a list of tricks and recipes.

Many attempts to reduce the sensibility of hyper-parameters like Adagrad (Duchi, Hazan, and Singer 2011) or Adadelta (Zeiler 2012) have been investigated.

In order to help the convergence, layerwise unsupervised pretraining can be used (Bengio et al. 2007) or momentum (Sutskever et al. 2013) can be added to the SGD. The preparation of training data is also important, from utterance order randomization to more advanced methods like batch normalization (Ioffe and Szegedy 2015).

Some other tricks can avoid overfitting, like weight decay or dropout (Hinton et al. 2012).

To conclude, tunning a DNN is more or less an art and the efficiency of the diverse tricks depends on the application, the dataset, the architecture, etc. For a more exhaustive presentation, see (Bengio, Goodfellow, and Courville 2015).

5 Research tools and implementations

As DNN training and designing is hard to study in a purely theoretical way, finding a flexible framework to run experimentations is worth spending some time on it.

Furthermore, the team was not stick to a given tool so I had the liberty to chose the one I felt the most comfortable with. This choice takes into account the efficiency (save machine time), the flexibility (save human time) and also the community behind a tool. Working with the same tools as other researchers is useful to directly share work and experiment other people ideas.

5.1 DNN tools

The two leading open source tools for Deep Learning are Theano (Bergstra et al. 2010) and Torch. Successive benchmarks have reported similar performances, so the choice is more about the comfort of use, which might be different for different persons.

I believe that comparing Theano and Torch is similar to comparing respectively the theoretical and the practical definitions of neural networks given in sections 4.1 and 4.2. They are almost equivalent, but provide two different points of view, with their strengths and weaknesses.

Theano is entirely built around the idea of computation graphs and is actually quite agnostic to the idea of neural network if used alone. It provides powerful tools to prune, factorize and compile such graphs, either on CPU or GPU.

On the other hand, Torch is designed as a lightweight but complete numerical computing library using transparently CPU and GPU and providing a wide range

of models of DNN layers and a set of training optimization algorithms.

On a more technical side, Theano is a Python library and Torch a Lua package. The Python library can benefit from the huge Python community and the tools it developed. Lua provides a light C interface that makes it very easy to integrate models built in experimentations into a production system.

Furthermore, Torch does not need the graph compilation step used by Theano. This step can be a source of many architecture specific issues, obscure error messages not caught by Python, and compilation latency.

Both tools have a serious base of users. Torch is highly used and even maintained by companies like Facebook, Google or Twitter. Theano is maintained by a lot of academic contributors, especially from Y. Bengio's laboratory at Université de Montréal.

Although I tried both, the majority of my experiments have been driven using Torch.

5.2 ASR tools

Talking about tools, it is also interesting to present the Automatic Speech Recognition tools, although the main one I worked on is not Open Source and so I cannot give any detail about it.

But there is an Open Source reference for ASR that I have also looked at and which is developed by academics, called Kaldi (Povey et al. 2011). This provides a complete ASR pipeline reported to be close to or even beat the state-of-the-art (Vesely et al. 2013) (Povey, Zhang, and Khudanpur 2014) and has been designed to be easily modified for research purpose.

The private tool I worked on is called Watson and is the product of the work of the team during several decades. Some lines of code are even older than I am! It has been used in production for a long time by AT&T and now Interactions with successful results. Public information about Watson has been communicated through research papers such as (Bocchieri, Caseiro, and Dimitriadis 2011) or (Dimitriadis, Bocchieri, and Caseiro 2011).

There are of course other existing tools, but here are the one I have had the occasion to work with.

6 Personal work and contributions

6.1 MNIST

As shown in Section 4, designing and tunning DNNs for a given task is an art that requires some intuition about the behavior of the different theoretical tools available.

Facing this huge exploration space, I needed to experiment the basic ideas on a dataset of manageable size. Thus, it was possible to test models in a reasonable amount of time and so to test a lot of models (hundreds).

This is why I did all my first experiment of the MNIST dataset (see Figure 10). This is a set of 60,000 handwritten digits on which a lot of state-of-the-art methods has been applied, providing reference baselines[8].

Figure 10: Sample of MNSIT inputs

The task associated to this dataset is to classify those images into 10 classes corresponding to the 10 decimal digits. This made me understand better the DNNs, and the optimization algorithms used to train them.

This was also an occasion to practice Machine Learning in general, learning to recognize overfitting or to use validation sets, etc.

[8] http://yann.lecun.com/exdb/mnist/

6.2 DNN training parallelization

Beside this practical approach to Deep Learning, I did some reading and preliminary work about the main problematic of my internship: speeding up the DNN training by parallelizing it over many machines.

6.2.1 Initial Problem

As for any time consuming task, it is tempting to run its different components at the same time, on different machines. But this requires the system to be separated into components that does not communicate too much. Indeed, the time won by parallelization can easily be lost in term of communication latency. The usual training of neural network addresses two problems to parallelization.

The first one is that all the units communicate very often and with too many other units, so distributing a given network over many machines is difficult.

The second issue is that even if the network is trained by batches of data and so updated only every 256 inputs (for instance), the time required to communicate the updated weights is not negligible compared with the time spent computing it.

On the other hand, given that a network accuracy increases with the size of the dataset, and that there are indeed big datasets available, the training is really time consuming, even using GPUs whose memory can be limited.

6.2.2 Existing solutions

Distributing the usual algorithms "as is", without any fundamental modification, is impossible, or not efficient. What we need is to modify the learning more deeply so that several machines could train on their own while synchronizing less often.

Heavy software infrastructures have been developed, like for instance Google's *DistBelief* (Dean et al. 2012) which relies on an asynchronous parameter server and the distribution of both data and units. But this is more useful to train bigger models than to train them faster.

A more fundamental way to parallelize the training would be to find an efficient way to combine networks with the same architecture but trained on different data into a new network, better than any of the initial ones.

This idea is explored by (Povey, Zhang, and Khudanpur 2014). The reduction step is simply an averaging of the weights across the many parallel models, but the first results are not good. This becomes to be efficient when the training uses a different training method known as *Natural Gradient*. The results on the implementation provided in Kaldi are promising, although this does not scale well beyond 4 parallel machines, but those results are essentially empirical and require further investigation.

6.2.3 Theoretical intuition

The first step is to understand why the basic weight averaging does not work. As for a lot of theoretical Machine Learning problems, studying the trajectory of an error rate in a very general way is challenging, but a possible understanding relies in the unsupervised specialization of the units.

As shown by 11, the parallel jobs of the same training can lead to very different models. Indeed, although the inputs and outputs of the network are imposed, the choice of the role of hidden units is left to the initiative of the learning algorithm because many of them have a symmetric description.

As a consequence, averaging the weights of the units means mixing the properties of classifiers that do not aim at recognizing the same patterns!

6.2.4 Natural Gradient

The natural gradient can be considered as a way to prevent the units from changing their role (Fukumizu and Amari 2000). It makes the critical points of the parameters space where two units switch their roles

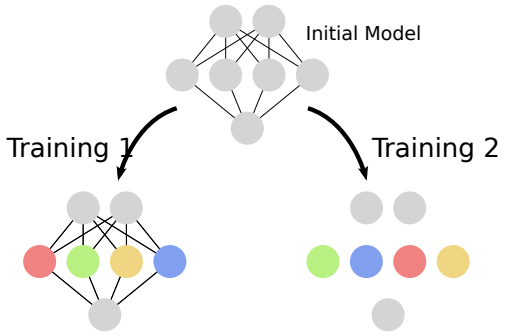

Figure 11: Problem of the unsupervised unit specialization. Many trainings of the same architecture can lead to very different self-organization of the hidden units.

more repulsive than with a standard SGD. So once those roles have been assigned, for instance by the first iterations of training, the natural gradient can be used for parallelization.

A reason for this improved robustness is that natural gradient is a second-order method while the standard SGD is a first order method. Note that in a more practical point of view, D. Povey presents this to deep learners as a way to apply an adapted learning rate at each unit, while a machine learner calls it a second-order method. These are two valuable points of view.

The main reason why natural gradient is not as efficient in practice is that it cannot be implemented exactly. The amount computation it theoretically requires is way too large, as it is in $\mathcal{O}(n^3)$ with a number n of neurons typically around 10,000, wile the SGD is about $\mathcal{O}(n^2)$.

Natural Gradient can be approximated by algorithms of backpropagation with hessian-free second order SGD (Pearlmutter 1994) (Martens 2010), and strong theoretical work on their application to neural network has been explored (Ollivier 2013) (Pascanu and Bengio 2013), even very recently (Martens and Grosse 2015).

But considering the considerable effort to implement it, I first explored another simpler idea for parallelization: Feedback Alignment (FA).

6.2.5 Feedback Alignment

Addressing the problem of constraining the unsupervised hidden units specialization, using for instance target propagation (Bengio 2014), my advisor found the work of (Lillicrap et al. 2014) on what is presented as a potentially more biologically plausible way to train a neural network.

6.2.5.1 Motivations and parallelization The basic idea of FA is presented in Figure 12. It has originally not been developed for parallelization, but as a more realistic learning algorithm, given that the usual backpropagation is very likely to not be how the human brain works.

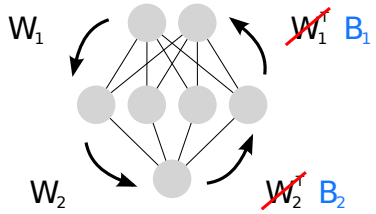

Figure 12: Feedback Alignment: Instead of backpropagating the output error using the transposed weight matrix, a random matrix B is used. This feedback matrix remains the same during all the training.

Here the backpropagation is highly influenced by the feedback matrix B. The guess was that this matrix somehow assigns a role to the units and hence training several models in parallel with the same feedback matrix should lead to very similar networks.

6.2.5.2 Reproduction of the results As it was a relatively new and unknown method, FA had no released public implementation. The first step was hence to implement it – which I did using Torch – and reproduce the article's result.

The presented results happen to be on the MNSIT dataset, but the backpropagation baseline they compare FA to is actually not the state of the art. Although I have been able to reproduce the result about FA, it is still less accurate than a traditional backpropagation (1.7% of error vs 1.5% while the paper reports 2.4%).

6.2.5.3 Further investigation Anyway, I continued investigating the FA since the results remain good for such a simplification of the learning algorithm.

I explored variants for the random drawing of the feedback matrix. Lillicrap draws B uniformly in $[-1, 1]$. Noticing that the sign of the forward weights progressively aligns to the sign of the feedback weights during the training, I assumed that only the sign was important and used $\{-1, 1\}$ instead, leading to similar results. I even changed the proportion of 1s and -1s and even a repartition of 10-90 percents works.

This idea of caring about the sign of the error only led me to try another variant then called *Discrete Backpropagation* in which the feedback matrix is build using only the sign of the forward weights. This gives intermediate results, between usual backpropagation and FA.

6.2.5.4 Application to parallelization Finally, I tried to apply FA to parallelization, which was the original goal, but unfortunately it did not succeed. Freezing the feedback does not seem to assign a precise role to the units.

This result is actually still very interesting since it might show that the role is essentially determined during the forward pass. This means that the units chose their role quite independently, given the representation made available by the previous layer, instead of requesting a given role to its inputs during the error backpropagation.

However, I then turned to the application of my new expertise in DNN tunning to ASR as it is what the team develop.

6.3 Experiments on ASR

6.3.1 Bootstrapping

The DNN in ASR associates to each frame of typically 10ms of sound a distribution of the probability to be in a given hidden state of the HMM. In fact, it uses a sliding window of several 10ms frames as input (11 frames in my case).

This means that the training data must provide a target state, or *label*, for each frame. But such dataset is highly dependent on the architecture of the HMM. Moreover, a dataset as the one I used contained millions of frames, so no human can afford labeling it manually.

Instead, a prior AM is used to provide an initial labeling, bootstrapping the learning process.

6.3.2 Transition from MNIST

The nature of the input, as well as the targets, is very different from MNIST task, and so some tricks that do well for the later might not work for the former.

For instance, the number of outputs for ASR is several thousands, and sometimes even close to 20,000. While the output layer is almost negligible with MNIST, it takes an important part of the computation. This makes method applicable only to the last layers worthwhile.

However, the experiments on MNIST where important to provide me a sensibility that is useful for the tunning of any DNN. DNN training for ASR takes more time and so allow less dummy exploration. My own learning of DNN tunning as somehow been bootstrapped as well!

6.3.3 Usual datasets

Dataset are of a critical importance. Their quality and relevance have a huge impact on the resulting models. They are also a way to confront to the published results.

Another interest in using common datasets is that their transcripts are regularly improved and enriched with other metadata such as speaker informations.

Among the most used datasets in publications are DARPA's Resource Management and TIMIT, Wall Street Journal, and Switchboard. Switchboard is the biggest and maybe the most used in the last publications. Hence the more challenging.

But the team have also access to other data from its previous work and collaboration, and thanks to Interactions' Adaptive-Understanding. Those data eventually fit better the final application of the system developed and their diversity makes them more challenging.

I used for a majority of my experimentations a dataset designed to be at the same time of e reasonable size (74,000 sentences, typically 8 hours of training per model) and diversified enough to reflect the complexity of bigger datasets used in production. This way, a method successful for this dataset is likely to scale well to production ones.

I also lately used Switchboard to get comparison with published results.

6.3.4 Label and Word accuracy

I initially ran experiments as a pure machine learner, evaluating the accuracy of my models on a test set of the same nature as the training set, i.e. a set of speech frames and associated labels.

But those labels are actually not an absolute information since they have been produced by another model. The measured accuracy was hence more a measure of discrepancy between two models.

To really get an idea of the accuracy of the model, it has to be evaluated in a pipeline that outputs data comparable with the provided with the original dataset, namely text transcripts. This leads us to the *word accuracy* – or its complementary, the Word Error Rate (WER).

6.3.5 Decoding

Measuring the WER requires to run the decoder on the output of the DNN. The decoder also high sensitivity to its hyper-parameters, especially in the combination of AM and LM whose provided probabilities can be exponentially scaled or floored for instance.

So for all the models that performed well on label accuracy, I had to run the decoder many times (10 to 100 times) to find the best decoding parameters (simply by grid-searching).

The fact of using a different measure of error to train and evaluate models can be disturbing, but backpropagating error through the Viterbi decoder is not an easy task, especially given the optimizations it can use like pruning of underused arcs. This problem can be solved by using sequential training as it is the case in some of Kaldi's recipes (Vesely et al. 2013).

6.3.6 Subsampling

Given that I had to run the decoder a lot, I investigated the tricks used to speed it up. There can be about the implementation of the Viterbi decoding itself, but other interesting ideas can be found. DNN subsampling is one of them (Vanhoucke, Devin, and Heigold 2013).

Instead of running the decoder for each time step, and so every 10ms in our case, a subsampled network generates for a given input window of frames two labels. One for the current frame, and one for the next frame. Actually the original paper outputs up to 4 frames at the same time.

This gave me results close to the traditional pipeline while being faster to compute. I obtained word accuracy of 74.5%, and then 75.3% using the $\mathcal{S}$ method (See Subsection 6.3.7), versus 76.5% for the traditional model.

Note that the larger the output is, the less the subsampling is efficient.

model	word accuracy
RNN-$\mathcal{S}$	76.4%
RNN	76.8%
SUB-$\mathcal{S}$	75.3%
SUB	74.5%
DNN-$\mathcal{S}$	76.4%
DNN	76.5%

Table 1: Examples of results obtained for speech recognition on our custom dataset. The models whose name ends with a "-$\mathcal{S}$" have been trained using the $\mathcal{S}$ method.

6.3.7 $\mathcal{S}$ method

Still worried about the training time, I designed and tested a new training method. Unfortunately, this is currently being patented and patent process is not as flexible as scientific publication regarding disclosure, which means I am unable to provide a preprint version. Hence the obscure designation of $\mathcal{S}$ *method* used in this report.

This method divided the training time by a factor of more than 3.5 on our training set and is being tested on Switchboard for a more comparable result. Furthermore, the consequences of this method on the word accuracy are small and sometimes beneficial! Some key results are presented in Table 1.

6.4 Comparison of Kaldi and Watson

As an open source tool, Kaldi benefits from the contributions of a wider audience. It hence provides training algorithms that Watson does not. But Watson has been especially optimized for runtime as it is used in commercial product that requires reactivity and low latency.

Looking at Kaldi is a way to test some methods to see if it is worth implementing it in Watson. And reading two implementation of very similar pipelines has been really helpful for my understanding of those tools.

7 Conclusion

The experience provided by this internship has been incredibly instructive. In a nutshell, almost all what is presented in this document has been learned during the internship.

I discovered the full and complex pipeline of ASR, and it get me experimented about DNNs, which can be used in a lot of research fields. Working on several subproblems gave me a very valuable overview.

I had the occasion to meet people working on a wide range since the team has been developing jointly all the components of the Speech and Language Technologies for a long time.

Interactions has built an effective business model around speech technologies that are difficult to use as a standalone tool. This gives a great opportunity to be able to do research and see a direct application.

Furthermore, the organization was quite unique since I lived the spinning-off of the team from AT&T and its move to Interactions, which is a very interesting step.

Not to mention that living and working in the US for many months, and especially in New York City, dramatically improved my English skills and was a very nice experience!

Acknowledgment

I wish to thank the whole team, for its patience and for making me feel useful and enjoying my work. I especially thank Patrick Haffner, my advisor who have closely and wisely followed and steered my researches (and reviewed this report).

Bibliography

Abdel-Hamid, Ossama, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. 2012. “Applying Convolutional Neural Networks Concepts to Hybrid NN-HMM Model for Speech Recognition.” In *Acoustics, Speech*

and Signal Processing (ICASSP), 2012 IEEE International Conference on, 4277–80. IEEE.

Bengio, Yoshua. 2014. "How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation." *ArXiv Preprint ArXiv:1407.7906*.

Bengio, Yoshua, Ian J. Goodfellow, and Aaron Courville. 2015. "Deep Learning." http://www.iro.umontreal.ca/~bengioy/dlbook.

Bengio, Yoshua, Pascal Lamblin, Dan Popovici, Hugo Larochelle, and others. 2007. "Greedy Layer-Wise Training of Deep Networks." *Advances in Neural Information Processing Systems* 19. MIT; 1998: 153.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. "Learning Long-Term Dependencies with Gradient Descent Is Difficult." *Neural Networks, IEEE Transactions on* 5 (2). IEEE: 157–66.

Bergstra, James, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. "Theano: A CPU and GPU Math Expression Compiler." In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 4:3. Austin, TX.

Bocchieri, Enrico, Diamantino Caseiro, and Dimitrios Dimitriadis. 2011. "Speech Recognition Modeling Advances for Mobile Voice Search." In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, 4888–91. IEEE.

Bottou, Léon. 2012. "Stochastic Gradient Descent Tricks." In *Neural Networks: Tricks of the Trade*, 421–36. Springer.

Chorowski, Jan, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. "End-to-End Continuous Speech Recognition Using Attention-Based Recurrent NN: First Results." *ArXiv Preprint ArXiv:1412.1602*.

Dean, Jeffrey, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc aurelio Ranzato, et al. 2012. "Large Scale Distributed Deep Networks." In *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, 1223–31. Curran Associates, Inc. http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf.

Deng, Li, Michael L Seltzer, Dong Yu, Alex Acero, Abdel-rahman Mohamed, and Geoffrey E Hinton. 2010. "Binary Coding of Speech Spectrograms Using a Deep Auto-Encoder." In *Interspeech*, 1692–95. Citeseer.

Dimitriadis, Dimitrios, Enrico Bocchieri, and Diamantino Caseiro. 2011. "An Alternative Front-End for the AT&T WATSON LV-CSR System." In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, 4488–91. IEEE.

Dimitriadis, Dimitrios, Petros Maragos, and Alexandros Potamianos. 2005. "Robust AM-FM Features for Speech Recognition." *Signal Processing Letters, IEEE* 12 (9). IEEE: 621–24.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *The Journal of Machine Learning Research* 12. JMLR. org: 2121–59.

Fukumizu, Kenji, and Shun-ichi Amari. 2000. "Local Minima and Plateaus in Hierarchical Structures of Multilayer Perceptrons." *Neural Networks* 13 (3). Elsevier: 317–27.

Glorot, Xavier, and Yoshua Bengio. 2010. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In *International Conference on Artificial Intelligence and Statistics*, 249–56.

Graves, Alex. 2013. "Generating Sequences with Recurrent Neural Networks." *ArXiv Preprint ArXiv:1308.0850*.

Graves, Alex, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks." In *Proceedings of the 23rd International Conference on Machine Learning*, 369–76. ACM.

Hannun, Awni, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, et al. 2014. "DeepSpeech: Scaling up End-to-End Speech Recognition." *ArXiv Preprint ArXiv:1412.5567*.

Hermansky, Hynek, Daniel W Ellis, and Shantanu Sharma. 2000. "Tandem Connectionist Feature Extraction for Conventional HMM Systems." In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, 3:1635–38. IEEE.

Hinton, Geoffrey E, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors." *ArXiv Preprint ArXiv:1207.0580*.

Hinton, Geoffrey, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, et al. 2012. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups." *Signal Processing Magazine, IEEE* 29 (6). IEEE: 82–97.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9 (8). MIT Press: 1735–80.

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ArXiv Preprint ArXiv:1502.03167*.

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. 2014. "A Convolutional Neural Network for Modelling Sentences." *ArXiv Preprint ArXiv:1404.2188*.

Koutník, Jan, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. 2014. "A Clockwork Rnn." *ArXiv Preprint ArXiv:1402.3511*.

Lillicrap, Timothy P, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. 2014. "Random Feedback Weights Support Learning in Deep Neural Networks." *ArXiv Preprint ArXiv:1411.0247*.

Livescu, Karen, Eric Fosler-Lussier, and Florian Metze. 2012. "Subword Modeling for Automatic Speech Recognition: Past, Present, and Emerging Approaches." *Signal Processing Magazine, IEEE* 29 (6). IEEE: 44–57.

Martens, James. 2010. "Deep Learning via Hessian-Free Optimization." In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 735–42.

Martens, James, and Roger Grosse. 2015. "Optimizing Neural Networks with Kronecker-Factored Approximate Curvature." *ArXiv Preprint ArXiv:1503.05671*.

Mohri, Mehryar, Fernando Pereira, and Michael Riley. 2002. "Weighted Finite-State Transducers in Speech Recognition." *Computer Speech & Language* 16 (1). Elsevier: 69–88.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified Linear Units Improve Restricted Boltzmann Machines." In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–14.

Ollivier, Yann. 2013. "Riemannian Metrics for Neural Networks." *ArXiv Preprint ArXiv:1303.0818*.

Palangi, Hamid, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2015. "Deep Sentence Embedding Using the Long Short Term Memory Network: Analysis and Application to Information Retrieval." *ArXiv Preprint ArXiv:1502.06922*.

Pascanu, Razvan, and Yoshua Bengio. 2013. "Revisiting Natural Gradient for Deep Networks." *ArXiv Preprint ArXiv:1301.3584*.

Pearlmutter, Barak A. 1994. "Fast Exact Multiplication by the Hessian." *Neural Computation* 6 (1). MIT Press: 147–60.

Povey, Daniel, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, et al. 2011. "The Kaldi Speech Recognition Toolkit." In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. Hilton Waikoloa Village, Big Island, Hawaii, US: IEEE Signal Processing Society.

Povey, Daniel, Xiaohui Zhang, and Sanjeev Khudanpur. 2014. "Parallel Training of Deep Neural Networks with Natural Gradient and Parameter Averaging." *ArXiv Preprint ArXiv:1410.7455*.

Saon, George, Hong-Kwang J Kuo, Steven Rennie, and Michael Picheny. 2015. "The IBM 2015 English

Conversational Telephone Speech Recognition System." *ArXiv Preprint ArXiv:1505.05899*.

Senior, Alan, Georg Heigold, Marc'Aurelio Ranzato, and Ke Yang. 2013. "An Empirical Study of Learning Rates in Deep Neural Networks for Speech Recognition." In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 6724–28. IEEE.

Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton. 2013. "On the Importance of Initialization and Momentum in Deep Learning." In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 1139–47.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. "Going Deeper with Convolutions." *ArXiv Preprint ArXiv:1409.4842*.

Tai, Kai Sheng, Richard Socher, and Christopher D Manning. 2015. "Improved Semantic Representations from Tree-Structured Long Short-Term Memory Networks." *ArXiv Preprint ArXiv:1503.00075*.

Vanhoucke, Vincent, Matthieu Devin, and Georg Heigold. 2013. "Multiframe Deep Neural Networks for Acoustic Modeling." In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 7582–85. IEEE.

Vasilache, Nicolas, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. 2014. "Fast Convolutional Nets with Fbfft: A GPU Performance Evaluation." *ArXiv Preprint ArXiv:1412.7580*.

Vesely, Karel, Arnab Ghoshal, Lukás Burget, and Daniel Povey. 2013. "Sequence-Discriminative Training of Deep Neural Networks." In *INTERSPEECH*, 2345–49.

Viterbi, Andrew J. 1967. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm." *Information Theory, IEEE Transactions on* 13 (2). IEEE: 260–69.

Werbos, Paul J. 1990. "Backpropagation Through Time: What It Does and How to Do It." *Proceedings of the IEEE* 78 (10). IEEE: 1550–60.

Weston, Jason, Sumit Chopra, and Antoine Bordes. 2014. "Memory Networks." *ArXiv Preprint ArXiv:1410.3916*.

Young, Steve J, Julian J Odell, and Philip C Woodland. 1994. "Tree-Based State Tying for High Accuracy Acoustic Modelling." In *Proceedings of the Workshop on Human Language Technology*, 307–12. Association for Computational Linguistics.

Zeiler, Matthew D. 2012. "ADADELTA: An Adaptive Learning Rate Method." *ArXiv Preprint ArXiv:1212.5701*.

Zhang, Xiang, and Yann LeCun. 2015. "Text Understanding from Scratch." *ArXiv Preprint ArXiv:1502.01710*.

Zhang, Xiaohui, Jan Trmal, Daniel Povey, and Sanjeev Khudanpur. 2014. "Improving Deep Neural Network Acoustic Models Using Generalized Maxout Networks." In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, 215–19. IEEE.