End-to-End Deep Neural Network for Automatic Speech Recognition

William Song
willsong@stanford.edu
Department of Computer Science
Stanford University

Jim Cai
jimcai@stanford.edu
Department of Computer Science
Stanford University

Abstract

We investigate the efficacy of deep neural networks on speech recognition. Specifically, we implement an end-to-end deep learning system that utilizes mel-filter bank features to directly output to spoken phonemes without the need of a traditional Hidden Markov Model for decoding. The system will comprise of two variants of neural networks for phoneme recognition. In particular, we utilize convolutional for frame level classification and recurrent architecture with Connectionist Temporal Classification loss for decoding the frames into a sequence of phonemes. We carry out our experiments on the TIMIT dataset. We have managed to obtain 22.1% for the frame error rate with our CNN, which to our knowledge, closely matches the state-of-the-art. Our decoded phone sequence achieves an error of 29.4%.

1 Introduction

The problem of automatic speech recognition has been an important research topic in the machine learning community since as early as the 70s [13]. Most standard ASR systems delineate between phoneme recognition and word decoding[11][13]. Before the emergence of deep learning, researchers often utilized other classification algorithms on highly specialized features such as MFCC in order to arrive at a distribution of possible phonemes for each frame. During the decoding phase, a Hidden Markov Model (HMM) with a pre-trained language model is used to find the most likely sequence of phones that can be mapped to output words. Earlier applications of deep learning in speech also separates the two tasks; there are many successful hybrid systems that take advantage of DNN's discriminative power for phone recognition but leave the decoding for the HMM. We are interested in extending the recent developments as done in [9][10] that have produced state-of-the-art results with recurrent neural networks that can handle recognition and decoding simultaneously.

With the advent of utilizing GPUs to train deep neural networks (DNNs), many DNN architectures have performed extremely well in a variety of machine learning problems. Even though typically there is little to no feature engineering in the process, these neural networks have managed to consistently beat benchmarks on various speech tasks. In fact, most of the state-of-the-art in automatic speech recognition are a result of DNN models [4].

However, many DNN speech models, including the widely used Google speech API, use only densely connected layers [3]. While such models have great learning capacity, they are also very prone to overfitting and it is also difficult for it to learn from features that have local correlations. Further, most such systems use a separate paradigm of model to perform decoding. With this insight, we'd like to combine the recent advancement in both CNN and RNN to produce an end-to-end speech recognition system using purely neural networks. The motivation to use CNN is inspired by the recent successes of convolutional neural networks (CNN) in many computer vision applications, where the input to the network is typically a two-dimensional matrix with very strong local correlations [5]. While the reason for using RNN and CTC is to replace the HMM in order to truly achieve an end-to-end deep learning system.

2 Related Work

For a previous course, we experimented with a speech recognition architecture consisting of a hybrid deep convolutional neural network (CNN) for phoneme recognition and a hidden Markov model (HMM) for word decoding. Specifically, we implemented a GPU-based CNN and applied it on the logged Mel filter-bank features to determine the likelihood distribution of the phones per frame. We then used these as emission probabilities and applied Viterbi decoding to compute the most likely phone sequence. However, due to the limited tools and resources, we were only be able to experiment with small CNN architectures. Our best model achieved an accuracy of 26.3% frame error on the standard core test dataset for TIMIT. In this project, we seek to extend the results of our previous project and remove the HMM decoding entirely, utilizing recurrent neural networks with CTC loss.

Novel neural network architectures have been shown to be successful in both decoding words from phonemes and identifying phonemes from speech. O. Abdel-Hamid, et al. have done seminal work in speech recognition with CNN and has demonstrated the state of the art results at the time it was published [14] and [15]. His approach still separates decoding (with an HMM) from the training of the network predicting the distribution of phonemes; the CNN also does not leverage longer-range temporal features. A. Graves et al. have attempted to combine CNN with LSTM-type RNN architecture and use beam search for word decoding, which also produced the state of the art result. J. Chorowski et al used a CNN network with maxout activations for phoneme recognitions and used a bidirectional RNN for word decoding.

Connectionist Temporal Classification[9] has seen success in the last decade since it was developed for decoding language. Hannun, et al.[7] utilized it for the decoding step in Baidu's deep speech network. Graves, et al.[17] utilized it as their objective function in their deep bi-directional LSTM ASR system. Most recently, A. Maas and Xie, et al. leveraged it to perform lexicon-free speech recognition[16].

3 Approach

3.1 Framewise Classification with CNN

Our approach consists of two main components – framewise classification and phone decoding. The diagram shown in Figure 1 illustrates a sample joint network of one of our experimented models. In the first part, we train a classifier that can independently predict each frame separated by 10 milliseconds intervals with high accuracy. A typical English sentence is 200-500 frames. We use Mel logged-filter bank features as input; when shown in 2D, it displays a transformed intensity of frequencies over time. Since such input closely resembles natural images, we decide to use CNN to perform training. CNNs are exceptionally good at capturing high level features in spatial domain and have demonstrated unparalleled success in computer vision related tasks. One natural advantage of using CNN is that it's invariant against translations of the variations in frequencies, which are common observed across speaker with different pitch due to their age or gender.

For each frame, we generate the actual input to the CNN by taking a window of frames surrounding it. Each input instance is a small one-channel image patch. Our CNN architecture closely resembles many of architectures seen in recent years of research CITE. It consists of 4 convolutional layers where the first two layers have max pooling. After the convolutions, it's followed by two densely connected layer and finally a softmax layer. ReLU is used for all activation functions. One thing we do differently is that instead of using the typical square convolution kernel, we use rectangular kernels since given a short window of frames, much of the information is stored across the frequency domain rather than the time domain.

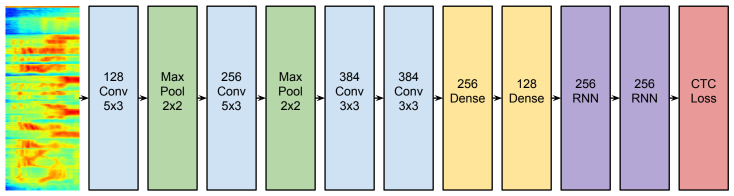

Figure 1: A fully connected layout of one of our many experimented architecture.

3.2 Decoding with CTC Network

In the second part, we attempt to replace the traditional HMM phone decoder with a suitable network capable of outputting sequential labels on un-aligned input data. Despite the fact that the actual input for a whole sentence contains hundreds of frames, it typically contain of no more than fifty phones. Even if our CNN could accurately predict most frames individually, we are more interested in the final output sequence of phones. We draw inspiration from papers using Connectionist Temporal Classification (CTC) as their loss function [9]. In [1] and [2], the authors have also extended their CTC with RNN transducer method, where a language model is added in conjunction with the CTC model. Using the embeddings or the probability distributions learned by the CNN, we would then use a CTC loss layer to finally output the phone sequence.

First we would like to describe the paradigm for decoding utilizing CTC loss in a RNN for decoding[9][16]. For a given input sequence, we would like to minimize the average edit distance from our predicted sequence to the actual sequence. For a given input sequence x_i, output sequence z_i and predictor h, we would like to find h that best minimizes

$$\frac{1}{n}\sum_{i}^{n}\frac{\text{edit}(h(\mathbf{x_i}),\mathbf{z_i})}{|\mathbf{z_i}|} \tag{1}$$

To obtain the most likely prediction for this evaluation metric, one natural goal is to maximize the probability of the correct output sequence to be predicted from the input sequence. More formally, given input: **x**= $(x_1, x_2, \dots, x_T)$ and output **z**=$(z_1, z_2, \dots, z_U)$, we want to maximize $p(\mathbf{z}|\mathbf{x})$ in our model. We assume

$$p(\mathbf{z}|\mathbf{x}) = \prod_{u} p(z_u|\mathbf{x}) \tag{2}$$

In other words, the individual phones are conditionally independent given the input sequence. CTC is usually associated with a neural network architecture; because of this, the output is usually interpreted as **y**, an {L x n} matrix of activations, where n is the number of frames in the output sequence and L is the number of labels. We also introduce a 'blank' label to the vocabulary. If the input and output sequence have the same number of elements,

$$p(\mathbf{z}|\mathbf{x}) = \prod_{t=0}^{T} y_{z_t}^{t} \tag{3}$$

To account for variable length sequences, the authors define a mapping $\beta(\mathbf{z}) = \mathbf{z}'$ that concatenate repeated frame predictions into one character. For example, $\beta(caaaat\ \ sattt) = cat\ sat$. The authors use β to allow for the input sequence and the output sequence to have different lengths, which lends itself naturally to the original problem of aligning audio features to shorter phoneme sequences. Another advantage of β is being able to distinguish between repeated characters and repeated characters followed by a space.

The most probable path reduces to taking the maximum activation at each point, whereas the most probable resolved path is not as easy to calculate. For a given labeling **t**, the probability of seeing labels $\mathbf{l}_{1:s}$ at time t is

$$\alpha_t(s) = \sum_{\pi:\beta(\pi_{1:t})=\mathbf{l}_{1:s}} \prod y_{\pi_t} \tag{4}$$

Naively, this is intractable, but [9] describes a dynamic programming formulation that efficiently calculates these probabilities. The authors insert a blank token in between each label for more

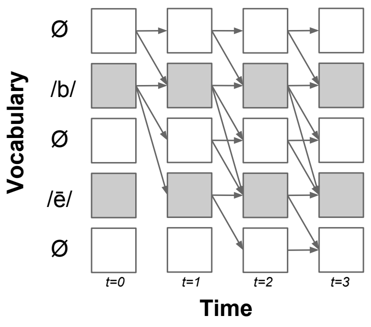

Figure 2: The possible paths for the phoneme sequence /b/ /ē/, corresponding to 'bee'

degrees of freedom in the segmentation. The augmented path length is 2|**l**|+1. Each blank label can transition to itself (continuation) or the next non-blank label. Each non-blank is allowed to transition to itself, the next blank label, or the next non-blank label in the sequence. We illustrate an example in 2.

The dynamic program formulation is straightforward from the diagram. Define

$$\bar{\alpha}_t(s) = \alpha_{t-1}(s) + \alpha_t(s-1) \tag{5}$$

We can see that if the current label is blank or if the current label is the same as it was two steps ago (continuation), there are only two incoming edges. Thus, the recurrence is:

$$\alpha_t(s) = \bar{\alpha}_t(s) y^t_{l_s} \tag{6}$$

For all other cases, there are three incoming edges, and the relation is

$$\alpha_t(s) = (\bar{\alpha}_t(s) y^t_{l_s} + \alpha_{t-1}(s-2)) y^t_{l_s} \tag{7}$$

The final probability is calculated as

$$p(\mathbf{l}|\mathbf{x}) = \alpha_T(|\mathbf{l}|) + \alpha_T(|\mathbf{l}|-1) \tag{8}$$

with the second term accounting for the lack of the final blank. We utilize the typical forward-backward algorithm to back-propagate errors. The forward variables are as defined previously, and the backward β variables are defined similarly, but for backward paths. It is essential to normalize α_t, β_t at every step to prevent underflow. The calculated final loss is then

$$ln(p(\mathbf{l}|\mathbf{x})) = \sum_{t=1}^{T} \sum_{s} ln(\alpha_t(s)) \tag{9}$$

We seek to minimize the negative of the function above through gradient descent and consider each sample separately. Given $\alpha_t(s), \beta_t(s)$ defined as above,

$$\frac{\alpha_t(s)\beta_t(s)}{y^t_{l_s}} = \sum_{\pi \in \beta^{-1}(l)} p(\pi|\mathbf{x}) \tag{10}$$

through algebraic manipulation. Summing over all s, we obtain $p(\mathbf{l}|\mathbf{x})$. Differentiating this w.r.t. y^t_k, we only consider paths at time t that go through label k.

$$\frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y^t_k} = \frac{1}{y^{t^2}_k} \sum_{s \in \text{pos}(k,l)} \alpha_t(s)\beta_t(s) \tag{11}$$

pos(k,l) gives the set of positions where k occurs in l. Thus the objective function's gradient is

$$\frac{\partial}{y^t_k}(-ln(p(\mathbf{l}|\mathbf{x})) = \frac{1}{p(\mathbf{l}|\mathbf{x})} \frac{1}{(y^t_k)^2} \sum_{s \in \text{pos}(k,l)} \alpha_t(s)\beta_t(s) \tag{12}$$

Through the softmax layer, we take the derivative w.r.t. the pre-normalized outputs u_k^t. This is the error signal sent to the lower RNN layers for back propagation.

$$\frac{\partial}{u_k^t}(-ln(p(\mathbf{l}|\mathbf{x})) = y_l^t - \frac{1}{y_k^t Z_t} \sum_{s \in \mathrm{pos}(k,\mathbf{l})} \frac{\alpha_t(s)\beta_t(s)}{y_{l_s}^t} \tag{13}$$

where Z is the normalization factor.

Our prediction algorithm tied to the CTC loss, h , takes the form of an RNN as in [16] due to its success in predicting character-level predictions that would hopefully translate into phone-level prediction success.

4 Experiments

4.1 Data

We perform our work on a commonly used dataset for speech recognition: TIMIT. TIMIT comprises of 630 speakers with 6300 utterances in 8 dialects [6]. Each audio clip consists of a phoneme transcription aligned with the audio as well as the expected sentence transcription.

As is standard with TIMIT evaluations, we remove the spoken dialect examples. Our train/validation/test breakdown is by speakers with 462/144/24 speakers used for each split. TIMIT contains 61 phone labels that we subsequently collapse into 39 during phoneme prediction by common practice [6] due to allophones and various phones being acoustically indistinguishable (but are differently demarcated based on context). We will keep the entire set of labelled phones during training and for decoding as well.

4.2 Implementation

We implement our CNN models using *Caffe*, a highly optimized tool that takes advantage of GPU's parallelization to speed up training [12]. It offers highly optimized kernels for convolutional layers.

For the CTC decoding network, we choose to use a Python library written by SAIL[16] to perform this task since Caffe does not directly support temporal layers. Since we'd like to take advantage of the efficiency of Caffe to experiment multiple potentially bigger and deeper networks, we decided to separate the training into two stages.

In this scheme, we first train a CNN that can predict frames with high accuracy. Then we feed in the last embedding layer of CNN into our CTC network to train to predict actual phone sequences. This approach would be equivalent to training a CNN first and then freezing the CNN parameters for fine-tuning the network on a separate task.

All feature generation, data loading and pre-processing of the acoustic data are implemented ourselves in a mixture of Python/C++ code.

4.3 Training

We first train a convolutional network with softmax layer on top for frame classification with shuffled frame data. We use a mini-batch size of 256 frames, where each frame is a 1-channel image patch of size $40 \times w$, where w is the context window size ranging from $15 - 25$. We find that using context window size larger than 25 does not seem to make improvements since most of the non-silence phones in speech are shorter than 20 frames, or equivalently, 200 milliseconds. We then remove the softmax layer and attach the CTC portion of the network atop the CNN and freeze the CNN weights to train in fashion similar to what are described in [10] and [9]. When feeding in data into the new CTC network, we group the frames by sentences, and all frames in a sentence are passed in its natural order. We train the CTC network with both standard densely connected neural network and recurrent neural network architectures. With the recurrent neural network, we perform back propagation through time algorithm and the cost associated with each frame is accumulated over the time. We choose to use Nesterov's accelerated gradient descent method for training [18].

Table 1: Frame error rate with CNN models

CNN Model	Val Err	Test Err
25 frame window, 128-256 Conv, 256-128 Dense	25.4%	26.3%
25 frame window, 128-256-384 Conv, 256-128 Dense	23.5%	23.8%
15 frame window, 128-256-384-384 Conv, 256-128 Dense	23.8%	23.9%
25 frame window, 128-256-384-384 Conv, 256-128 Dense	22.6%	22.9%
35 frame window, 128-256-384-384 Conv, 1024-512 Dense	22.6%	22.9%
25 frame window, 128-256-384-384 Conv, 1024-512 Dense	**21.3**%	**22.1**%

Table 2: Phone error rate with CTC models

CTC Model	Val Err	Test Err
MFCC Input, 1024-1024 RNN	38.2%	38.1%
MFCC Input, 2048-1024 RNN	35.6%	35.8%
MFCC Input, 2048-2048 RNN	34.7%	35.3%
CNN Input, 2×1024	33.5%	34.1%
CNN Input, 3×1024	32.7%	33.2%
CNN Input, 4×2048	30.5%	30.7%
CNN 2nd last layer Input, 4×2048	30.5%	30.7%
CNN Input, $4 \times 2048, \lambda = 1e - 3$	**28.9**%	**29.4**%
CNN Input, $4 \times 2048, \lambda = 1e - 4$	29.1%	29.7%
CNN Input, 2048-2048 RNN	33.1%	33.3%
CNN Input, $4 \times 2048, \lambda = 1e - 4$	32.0%	31.4%

For both networks, we train until the model begin to overfit on the training set and the dev accuracy begins to fall. Much of the training is done on SAIL's Deep clusters, which uses nVidia GTX780 GPUs.

4.4 Evaluation

The two main evaluation metrics are the frame error rate (FER) and phone error rate (PER). The FER simply reflects the inaccuracy of our CNN network on each individual frame. The FER is the percentage of frames that are predicted incorrectly by our classifier when using the collapsed 39 phonemes as the labels. The phone error rate can be computed after we've decoded the entire sentence into a sequence of phones with our CTC network. We then compute the Levenshtein (or edit) distance between our predicted sequence and the ground truth sequence to obtain the number of mistakes made. The average number of mistakes over the length of the phone sequence would be our PER. We cross-validate the hyper-parameters of our model, such as filter size, number of filters, depth, learning rate and regularization using the dev set as described above, and evaluate on the test set in the end.

4.5 Results

We have run many experiments with varying model architectures for both the CNN and the CTC network separately and shown some of the representative results in Table 1 and 2. All results in Table 1 are obtained through various CNN models. Note that the first result in the FER table are taken from our previous work in a different class, which was implemented with a Python framework called Theano. We were only able to try very small architecture in comparison to Caffe due to Theano's inefficiency. With Caffe, we are able to quickly experiment many much deeper and larger architectures with larger context window sizes to greatly boost the performance. To our knowledge, the FER we obtained is comparable to the best we've seen in the literature [8].

In the PER table, it demonstrates our result for using the CTC network to decode input over time to an actual phoneme sequence. Due to the rather long training time of RNN's, most of the models we trained are non-RNN's. We have been rather unsuccessful at training RNN's with the CNN input, and it made marginal improvement over tbe baseline of using simply the MFCC input.

Table 3: Comparison of several phoneme sequences between the ground truth (GT) and the predictions (Pred) of our best model of a few short sentences in the core test set of TIMIT sorted in ascending order of phone error rate (or in this case, the relative edit distance). A blank in the GT/Pred indicates an insertion/deletion in the prediction, while red indicates a substitution in the prediction.

Type	**Phoneme Sequences of a select set of sentences**
GT	`sil b r aw n ay z ay sil b r aw m ah s sil t ae sh sil`
Pred	`sil b r    n ay z ay sil b r    m ah s sil t ae sh sil`
GT	`sil b uh sil k s ah f ah sh sil n uh sil k s sil`
Pred	`sil b uh sil k s ah f ah sh sil l    sil   s sil`
GT	`sil sh uh sil sh iy w   ey sil k ih m sil`
Pred	`sil sh ih sil sh iy w r ey sil k ih m sil`
GT	`sil dh ey hh eh sil s l ae sil dh eh    r th ay z  sil`
Pred	`sil dh ey eh    sil s l ae sil dh ey ey r f  ay ay sil`
GT	`sil dh iy ow ey s ah s sil w ah z ah m er aa       sh    sil`
Pred	`sil dh iy y  ey s ih s sil w ah z ih m er aa ay ay sh jh sil`
GT	`sil   eh n w ah dx ay z dh ey w er sil`
Pred	`sil w ah n          ay s dh    w er sil`
GT	`sil ow n l ih l   oy er z l ah v m ih y ih n ih r  s sil`
Pred	`sil    n l iy l l ey er   l ah v n iy   ih n    er s sil`

4.6 Analysis

Overall, we are satisfied with our CNN performance for the task of framewise classification, but the result for CTC is rather underwhelming. We believe that we owe our success with CNN thanks to the efficient implementation of Caffe, which allowed us to train large models with speed.

For the CTC network, the raw phone accuracy is not very satisfying. However, upon closer inspection to the actual phone sequence output of a few short sentences in the test set as shown in Table 3, we find that many of the errors are understandable. For instance, most of the phonemes in red, which represent a wrongly predicted phone, in fact, do sound phonetically similar to the correct phonemes. For instance, the pair `ey` and `eh`, or `th` and `f`, all have very similar spectrograms. We think that much of the red errors shown in the table are due to the incapability of the CNN classifier. One very common source of error we can find are repeated phonemes. For instance, it has occurred on three of the last four sentences. We believe that inperfect training of the CTC network contribut directly to such errors since a blank label should not have been emitted between two of the same phones. While of course one very simple heuristic to fix such problem is to simply eliminate all duplicated phonemes (while indeed, duplicated phoneme rarely occurs in actual data), it contradicts the spirit of this project of achieving end predictions with solely neural networks.

5 Conclusion

Through this project we have explored advancements in recent literature regarding the replacement of the GMM-HMM based automatic speech recognition system with a deep neural networks. We have achieved good results frame accuracy with our CNN-classifier that leverages strong local correlation in speech signals; however, we were not able to leverage the output of this network successfully in our CTC-RNN.

We believe that one blunder we've made is not experimenting with the RNN models earlier on. We spent too much time trying to perfect the framewise classification with CNN, and underestimated the difficulty and the time involved to train a good RNN model with CTC loss. In the future, we'd like to investigate thoroughly the reasons for RNN's poor convergence, and perform more hyper parameter tuning. We are also interested in independently using CTC to train a phoneme language model to use it as initial weights before taking actual CNN input.

Acknowledgement

We'd like to thank Andrew Maas and Ziang Xie for providing us access to their CTC implementation. We'd also like to thank SAIL for providing us the necessary machines and GPUs to carry out our experiment. Finally, we'd like to thank Billy Jun's contribution to the Theano implementation of framewise classification.

References

[1] Graves, Alex. "Sequence transduction with recurrent neural networks." *arXiv preprint* arXiv:1211.3711 (2012).

[2] Graves, Alex, A-R. Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks." *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013.

[3] G. E. Hinton, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". *Signal Processing Magazine, IEEE*, 29(6):8297, 2012.

[4] Deng, Li, et al. "Recent advances in deep learning for speech research at Microsoft." *ICASSP*, 2013.

[5] A. Krizhevsky, et al., "ImageNet Classification with Deep Convolutional Neural Networks." *NIPS*, 2012.

[6] A. K. Halberstadt, "Heterogeneous acoustic measurements and multiple classifiers for speech recognition." *Ph.D. Dissertation at MIT*, 1998.

[7] Hannun, Awni, et al. "DeepSpeech: Scaling up end-to-end speech recognition." *arXiv preprint arXiv:1412.5567* (2014).

[8] Graves, Alex, and Jrgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural Networks* 18.5 (2005): 602-610.

[9] Graves, Alex, et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks." *Proceedings of the 23rd international conference on Machine learning. ACM*, 2006.

[10] Graves, Alex, Navdeep Jaitly, and A-R. Mohamed. "Hybrid speech recognition with deep bidirectional LSTM." *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on. IEEE*, 2013.

[11] Bourlard, H., and N. Morgan. "Connectionist speech recognition: a hybrid approach. 1994."

[12] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." *Proceedings of the ACM International Conference on Multimedia*. ACM, 2014.

[13] Bengio, Yoshua. "Markovian Models for Sequential Data." (1996).

[14] O. Abdel-Hamid, et al., "Applying CNN Concepts to Hybrid NN-HMM model for speech recognition." *IEEE ICASSP*, 2012.

[15] T. N. Sainath, et al., "Deep Convolutional Neural Networks for LVCSR." *IEEE ICASSP*, 2013.

[16] Maas, et al."Lexicon-Free Conversational Speech Recognition with Neural Networks." *NAACL*, 2015.

[17] Graves, Alex, and Navdeep Jaitly. "Towards end-to-end speech recognition with recurrent neural networks." Proceedings of the 31st International Conference on Machine Learning (ICML-14). 2014

[18] Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." Proceedings of the 30th international conference on machine learning (ICML-13). 2013.