Interspeech 2018
2-6 September 2018, Hyderabad

End-to-End Speech Command Recognition with Capsule Network

Jaesung Bae, Dae-Shik Kim

School of Electrical Engineering,
Korea Advanced Institute of Science and Technology (KAIST)
bjsd3, daeshik@kaist.ac.kr

Abstract

In recent years, neural networks have become one of the common approaches used in speech recognition(SR), with SR systems based on Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) achieving the state-of-the-art results in various SR benchmarks. Especially, since CNNs are capable of capturing the local features effectively, they are applied to tasks which have relatively short-term dependencies, such as keyword spotting or phoneme-level sequence recognition. However, one limitation of CNNs is that, with max-pooling, they do not consider the pose relationship between low-level features. Motivated by this problem, we apply the capsule network to capture the spatial relationship and pose information of speech spectrogram features in both frequency and time axes. We show that our proposed end-to-end SR system with capsule networks on one-second speech commands dataset achieves better results on both clean and noise-added test than baseline CNN models.

Index Terms: speech recognition, capsule network, routing-by-agreement, convolutional neural network

1. Introduction

Early-stage studies using neural network in automatic speech recognition were mostly hybrid speech recognition (SR) systems incorporating with Hidden Markov Models/Gaussian Mixture Models (HMMs/GMMs) [1, 2]. In the hybrid SR systems, the neural networks accept features generated from HMMs/GMMs and predict frame-level targets. However, the hybrid systems have the issue of having to set hyperparameters and train the networks separately.

Therefore, recent neural network based SR systems aim for an end-to-end system. Owing to the property of recurrent neural networks (RNNs) that show strengths in capturing long time dependency and that of convolutional neural networks (CNNs) in detecting the local features, a RNN-variant Long Short-Term Memory (LSTM) model combined with a Connectionist Temporal Classification (CTC) [3] model is achieving the-state-of-the-art performance in common SR systems [4, 5, 6]. However, training RNNs takes a longer time than CNNs due to the computational expensiveness and is trickier due to the gradient vanishing/exploding problem. Therefore, for tasks that have short time dependency such as keyword spotting or phoneme-level recognition, CNN based SR systems are still taking an active part of research and show competitive performance [7, 8, 9, 10].

However, the fundamental problem of CNNs is that it is not able to capture the spatial relationships of low-level features. For example, face detection using CNN can extract the low-level features such as eyes, nose, and mouth using edge or color gradient information and combine them with weighted sum at a higher level and decide whether the object is a face or not. During this process, CNN does not consider any spatial relationship or orientation of features: for instance, it disregards where the eyes and noses are placed. The problem is often handled by including max-pooling or building a deep network, making each high layer neuron have a larger receptive field. However, max-pooling leads to the loss of valuable information by ignoring all but the neurons with the maximum activation value, and there exists an upper limit in increasing the receptive field by the max-pooling and deep network.

We conjectured that in the speech domain, the spatial relationship among the speech features would also play a crucial role. In the conventional speech analysis, valuable information such as pitch and formant frequencies [11, 12] is gained from the spectrogram converted from the raw speech waveform. The position and the relationship of these features in frequency and time axes decide what speech the spectrogram represents. However, because the existing CNN based SR systems do not consider the spatial relationship of these features, there would be a limitation in performance, which can be overcome by applying the capsule network [13] instead, which considers the spatial relationship and pose information. Inspired by this, we applied the capsule network in SR systems and compared its performance with CNN based systems. For comparison, we used a one-second speech command recognition task [14] to evaluate the performance in clean and noisy environments. Because the one-second speech command dataset is developed for consumer and robotic applications, we sought to reduce the models parameter number. With this goal, we ran a set of experiments while varying factors such as the kernel size, absence or presence of decoder, channel size, input and output capsules vector length to find the best capsule network model. Comparing the best capsule network and CNN model, the capsule network had 11.6% less error rate (ER) in clean test, and 10.4% less ER in noisy test.

In section 2, we will discuss the capsule network in depth. In section 3, we will describe the architectures of baseline CNN and the capsule network. Then in section 4, the dataset, training and evaluation techniques, and the result will be described. Finally, we will discuss the results and suggest future work.

2. Capsule Network

Hinton et al. [13] suggested the idea of 'capsule' and the iterative routing-by-agreement mechanism. In this scheme, one converts a scalar output neuron to a vector output neuron called a capsule. By using the vector neuron, the capsule network can capture the various pose information such as translation and rotation of objects and contain it in a small vector. The original authors designed the capsule to encode the probability of entity exists as its length and the pose information as its orientation. Then, with the routing-by-agreement between capsules, the network can learn the hierarchical relationship between these pose information capsules contained.

The capsule network replaces the scalar neuron, which most of previous neural networks including Deep Neural Networks (DNNs) and CNNs use, to the vector neuron which is a capsule. Assume that there is a capsule $\boldsymbol{x}_i^l$ which represent the ith capsule of lth layer and $\boldsymbol{x}_j^{(l+1)}$ which represent the jth capsule of $(l+1)$th layer. Then from $\boldsymbol{x}_i^l$ we define $\boldsymbol{x}_j^{(l+1)}$ as

$$\boldsymbol{x}_j^{(l+1)} = squash\left(\sum_i^N c_{ij}\boldsymbol{W}_{ji}\boldsymbol{x}_i^l\right) \quad (1)$$

where $\boldsymbol{W}_{ji}$ is a weight matrix between capsule $\boldsymbol{x}_i^l$ and $\boldsymbol{x}_j^{(l+1)}$ and c_{ij} is a coupling coefficient. We could compute c_{ij} by applying softmax function to b_{ij} as

$$c_{ij} = \frac{\exp b_{ij}}{\sum_k \exp b_{ik}} \quad (2)$$

b_{ij} represents the log prior probability that capsule i in layer l should be coupled to capsule j in layer $(l+1)$, and it will be learned by routing algorithm. b_{ij} is initially set to 0. Then we could compute the $x_j^{(l+1)}$ with Equation1 and update the b_{ij} with Equation3.

$$b_{ij} \leftarrow b_{ij} + \boldsymbol{W}_{ji}\boldsymbol{x}_i^l \bullet \boldsymbol{x}_j^{l+1} \quad (3)$$

Then we compute the output capsule with updated b_{ij} again and keep updating b_{ij} for r routing times. This is called the routing-by-agreement because by dot product of $\boldsymbol{x}_i^l$ and $\boldsymbol{x}_j^{(l+1)}$ in Equation3, it computes the agreement between the child capsule and parent capsule and update the coupling coefficient with it.

They use the squashing function [13] for training the capsule network, which makes short vectors have a length close to zero and long vectors to a length close to one.

$$squash(\boldsymbol{x}) = \frac{||\boldsymbol{x}||^2}{1+||\boldsymbol{x}||^2}\frac{\boldsymbol{x}}{||\boldsymbol{x}||} \quad (4)$$

Finally, for the loss function, margin loss is used to train the model. L_c is the margin loss of class c, and $\boldsymbol{v}_c$ is a final output capsule in class c. $T_c = 1$ iff the target class is c, $m^+ = 0.9$, $m^- = 0.1$, and $\lambda = 0.5$.

$$\begin{aligned} L_c = {} & T_c \max\left(0, m^+ - ||\boldsymbol{v}_c||\right)^2 \\ & + \lambda(1-T_c)\max\left(0, ||\boldsymbol{v}_c|| - m^-\right)^2 \end{aligned} \quad (5)$$

In the paper [13], authors also built the decoder network that reconstructs the input image from the output capsules. It has three layers of fully connected (FC) layer with ReLU activation for the first two and sigmoid for the last, and is trained by computing mean squared error (MSE) between its output and the training image. Scaled-down MSE is then added to margin loss in Equation 5, which also works as a regularization term.

3. Model

3.1. Baseline CNN

For the baseline CNN models structure, we used a standard CNN, which has three convolutional layers at the front and followed by two FC layers. Each of the convolutional layers has a stride of 1 without any padding, and has ReLU for activation function. In precedent researches that tried to adjust CNNs for speech recognition tasks, they added max-pooling after the first convolutional layer only over the frequency-axis [10]. Therefore, similarly, we added max-pooling after the first convolutional layer with the kernel size and the stride of (1×3). After the last FC layer, one more FC layer with size 30 classes followed. Dropout [15] was applied in the layer before the output layer, and softmax was applied at the output layer. In the experiment, we changed the kernel and the channel size of convolutional layers and channel of two FC layers.

3.2. Capsule Network

We used the basic capsule network architecture [13] as Figure1. It has one convolutional layer, one input capsule layer, and one output capsule layer. The first convolutional layer is followed by ReLU non-linear activation function with stride of 1 and 256 channels without any padding. To make an input capsule, we convolved the output of the first convolutional layer with channel size of 'input capsule vector length' and stride of 2 for 'capsule channel' time. Here, instead of convolving for 'capsule channel' times, we convolved only once with channel of [capsule channel × input vector length] and then reshaped it to make the input capsule. From this input capsule layer, we can compute the output capsules with the routing algorithm as we described in section 2. The number of capsules in the output capsule layer will be the same as the class number.

To find the optimal capsule network architecture for one-second SR task, which achieves good performance with less parameters, we changed various hyperparameters in the capsule network and compared the results. We set the five hyperparameters as 1)the kernel size of convolution before the first convolution layer and the input capsule layer 2)presence and absence of a decoder network which reconstructs input image 3)capsule networks channel 4)input capsules vector length and 5)output capsules vector length. From the combination of 1)to 5), we searched for the best hyperparameters systematically and finally obtained the best model.

4. Experiment

4.1. Data

The speech commands dataset [14] consists of 64,727 audio files. The audio files were collected by crowdsourcing. They are labeled with 30 commands, which are divided into 20 main commands and 10 sub-commands recorded to help recognizing unrecognized words. Additionally, the dataset provides the background noise audio files with white and pink noise and some everyday life noise. The details of commands can be found in Table1.

Since this dataset was produced for consumer and robotic application purposes, they did not give strict guidelines for quality during crowdsourcing, other than to record in a closed room. Therefore, we assumed that the audio files in the dataset already contain a little noise, and when we added the background noise, the actual SNR would be lower than we expected. Each 10% of dataset was split for validation and testing.

We converted the raw audio files into 40-dimensional log mel-filter-bank coefficients with deltas and delta-deltas. They were computed with 20ms window, 10ms stride and normalized to have zero mean and unit variance.

4.2. Training and Evaluation

For training, we trained on a clean dataset with 30 labels (main and sub commands in Table 1). We used margin loss

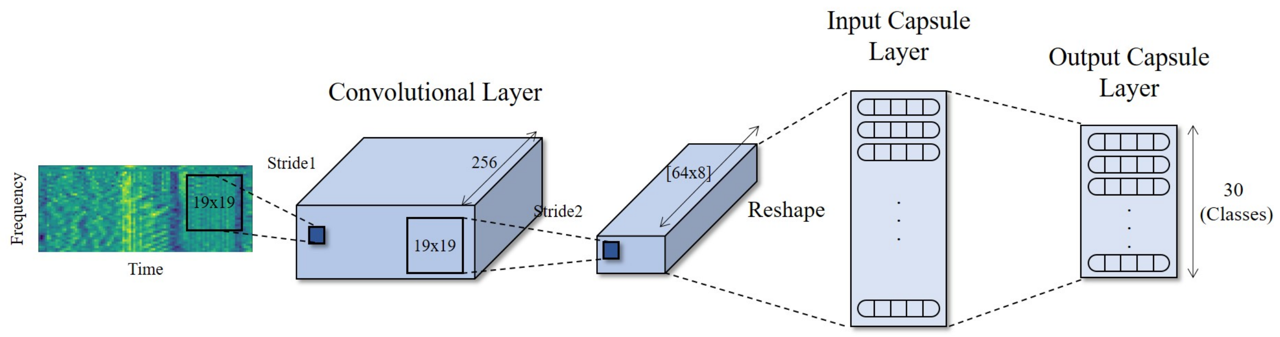

Figure 1: *The capsule network architecture with kernel size 19, 64 channels, input vector length of 8 and output vector length of 16. It is first convoluted with stride 1 and channel of 256, and then convoluted with stride 2 with (64× 8) channels which is channel size and input vector length. Then it is reshaped into [4096, 8] size array which makes input capsule layer. The routing algorithm is used to update coupling coefficient between input and output capsule layers. The output capsule layer has 30 columns because we train on 30 command labels.*

Table 1: *Speech Commands Dataset.*

Type	**Commands**
Main commands	down, eight, five, four, go, left, nine, no, off, on, one, right, seven, six, stop, three, two, up, yes, zero
Sub-commands	bed, bird, cat, dog, happy, house, marvin, sheila, tree, wow
Background noise	doing the dishes, dude miaowing, exercise bike, pink noise, running tap, white noise

(Equation5) for the capsule networks, and added scaled-down decoder MSE when the decoder network was used. CNNs were trained with cross entropy loss. We used the Adam optimizer [16] with learning rate 0.001 and 3 routing times in capsule network routing. For capsule networks, weight initialization was quite important for convergence. We used the normal distribution initializer with standard deviation of 0.01 for capsule layers and the Xavier initializer[17] for CNNs.

During evaluation, we used 20 main command labels plus one unrecognition label, which was used instead of sub-command labels, in both clean and noisy datasets. To make the audio files noisy, we randomly picked one type of noise from the provided background noise files and randomly clipped one second. Then we added it to a clean audio file at 5dB SNR. Because of this randomness, we evaluated on the noisy dataset for five times and averaged to get reliable results.

4.3. Result

For the baseline CNN models, we maintained the architecture described in section 3 and tested various sizes of kernels and channels of convolutional layers and the size of FC layers to find the best model. For capsule networks, we searched the best hyperparameters sequentially in the order of kernel size, decoders presence, channel size, input vector length, and output vector length. The starting model was kernel size 9, with decoder, 64 channels, 8 input-vector length, and 16 output-vector length.

4.3.1. Baseline CNN

Maintaining the standard architecture determined in section 3, we varied the kernel size and output channel of both convolutional layers and FC layers. The 'CNN-1' model had the kernel size (5,5) and 128, 256, 256 channels for convolutional layers and two 1024 channels for FC layers. 'CNN-2' had (5,5) kernel and 128, 64, 64 channels and 512, 512 FC channels. 'CNN-3' had (5,5) kernel for the first two convolutional layers and (7,7) for the last convolutional layer, the channels are 32, 64, 64 and 512, 256 for FC layer. In our experiment, when we reduced the number of parameters less than 'CNN-3', the CNNs did not converge or had significantly high test ER. The 'CNN-3' was the best model with clean ER 22.1% and noisy ER 55.1%.

4.3.2. Capsule Network

Starting from kernel size 9, we increased the kernel size and observed the number of parameter (NP), the clean and noisy ER in Table2. Up until kernel 19, the NP kept decreasing. However, with kernel size larger than 19, the size of image after the second convolutional layer became negative. To make the image size positive, we added zero-padding after the first convolutional layer, so the NP increased. The kernel 19 performed the best in clean, and kernel 25 was best in noisy test data. Even though the kernel 25 got least noisy ER, we chose kernel 19 because it had much less NP than 25 and had the least clean ER and second-to-the-last least noisy ER.

In this work, we focused on building well performing SR systems, with less emphasis on making the decoder converge. We compared the result after removing the decoder from 'Caps-kernel19', and the model without the decoder showed similar but a little worse performance in the clean and noisy test. However, we could take advantage of parameter reducing without the decoder, and we found that in most cases the decoder network did not train well. Therefore, in further experience we used a model without the decoder.

With the number of channels, 'Caps-channel32' had the least NP, clean ER, and noisy ER. For input and output vectors, they both had least clean and noisy ER with vector length 4, so we chose vector length 4 for both input and output capsule. The final best capsule network architecture we found through the experiment was 'Caps-outputvec4' which has kernel size 19, without decoder, channel of 32, and input and output cap-

Table 2: *Clean and noisy Error Rate(ER) of various baseline CNN and capsule network(Caps) models trained on clean dataset. 'NP' is the number of parameter. The model details are in section 4.3.*

Model	NP	clean ER	noisy ER
CNN-1	94.8M	24.2%	59.5%
CNN-2	12.0M	24.1%	57.2%
CNN-3	**6.0M**	**22.1%**	**55.1%**
Caps-kernel9	139.3M	20.8%	58.9%
Caps-kernel11	119.0M	20.2%	55.0%
Caps-kernel13	101.7M	21.6%	58.8%
Caps-kernel15	87.5M	19.0%	61.1%
Caps-kernel17	76.2M	16.3%	55.9%
Caps-kernel19	**68.0M**	**12.7%**	49.7%
Caps-kernel21	161.1M	26.2%	57.3%
Caps-kernel25	161.6M	13.1%	**45.6%**
Caps-kernel30	174.6M	19.6%	56.5%
Caps-NoDecoder	63.1M	13.9%	52.5%
Caps-channel32	**31.6M**	**11.3%**	**47.4%**
Caps-channel96	94.7M	15.1%	56.1%
Caps-channel128	131.0M	15.2%	55.9%
Caps-inputvec2	**8.0M**	12.2%	47.4%
Caps-inputvec4	15.9M	**11.6%**	**47.3%**
Caps-inputvec16	63.1M	12.3%	48.6%
Caps-outputvec2	**12.4M**	11.1%	48.4%
Caps-outputvec4	12.9M	**10.5%**	**44.7%**
Caps-outputvec8	13.9M	11.4%	50.9%

sules vector length of 4. Comparing the best capsule network model and best CNN model, the best capsule network model had 11.6% and 10.4% less ER in clean and noisy dataset each.

Since the best CNN model had less parameters than the best capsule network model, we ran additional experiments for comparison (Table 3). To reduce the number of parameters from the best capsule network model, we compared the results with various input and output vector length while fixing the kernel size to 19, channel to 16 and without a decoder. The capsule network model with input vector length 2 and output vector length 16 ('Caps-inputvec2' in Table3) had about 4.0M NP, which was about 67% of the NP of the best CNN model, but had 9.1% less clean ER and 2.9% less noisy ER.

5. Discussion

Almost every capsule network based model performed better than the baseline CNN models, and even with less NP, they got significantly better results than the CNNs. From this, we could conclude that the capsule networks could capture the speech features very efficiently. We suggested this was because, by using capsule and routing-by-agreement algorithm, the capsule network could contain pose information and spatial relationship of speech features which CNNs could not, as we originally hypothesized.

When we compared 'Caps-kernel9' and 'Caps-kernel25', they both had large NP but 'Caps-kernel25' had comparable performance to the 'Caps-kernel19' and even got best result in noisy test where 'Caps-kernel9' has poor performance. This suggested that capsule networks could capture the features well

Table 3: *Clean and noisy ER of the capsule network with 16 channels to reduce the NP. In the input vector length experiment, the output vector length was 16, and in the output vector length experiment, the input vector length was 4*

Model	NP	clean ER	noisy ER
Caps-inputvec2	**4.0M**	13.0%	52.2%
Caps-inputvec4	8.0M	11.3%	52.5%
Caps-inputvec16	31.6M	**11.1%**	**46.2%**
Caps-outputvec2	**6.3M**	11.6%	53.3%
Caps-outputvec4	6.5M	11.9%	48.5%
Caps-outputvec8	7.0M	**10.8%**	**48.0%**

even with a big kernel size. We suggested that this is because by using vector neuron, the amount of information that each neuron could contain becomes much bigger.

When we focused on NP, the performance got better when we decreased the NP until a certain level. This might be because the network could avoid over-fitting with less NP, but after a certain level the performance became lower. Therefore, finding the proper number of parameters for the model was important. For example, 'Caps-inputvec2' in Table2 had insufficient NP to contain the features information and 'Caps-inputvec16' in Table2 was over-fitted to the train data.

6. Conclusion and Future Work

In this work, we first applied the capsule networks in the speech domain in place of CNNs to capture the pose information and spatial relationships among features in frequency and time axes. We built an end-to-end one-second speech command recognition system and found the best capsule network architecture through experimentation. Compared to CNN models, the capsule network based systems achieved much better results in both clean and noisy data.

Because it was the first attempt to apply capsule network in speech domain, there is a plenty of room for further work. For example, we could apply the network to longer time dependency task, such as phoneme-level recognition. Also, we trained only on the clean dataset in the present study, but it is possible to train on noisy speech, to make model more noise-robust. Lastly, it should be examined whether adding regularization techniques such as dropout improves the test performance.

7. Acknowledgements

This work was supported by the Engineering Research Center of Excellence (ERC) Program supported by National Research Foundation (NRF), Korean Ministry of Science & ICT (MSIT) (Grant No. NRF-2017R1A5A1014708), the ICT R&D program of MSIP/IITP. [R-20161130-004520, Research on Adaptive Machine Learning Technology Development for Intelligent Autonomous Digital Companion], the KIST Institutional Program. (Project No. 2E27810-18-P034), the ETUDE CORPORATION. (Project No. G01180051), and Brain Korea 21+ Project, BK Electronics and Communications Technology Division, KAIST in 2018. The authors also thank KOLON Corporation, Korea, for providing funding for this research through the KOLON-KAIST Lifestyle Innovation Center Project. (LSI17-ITKDS0001)

8. References

[1] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.

[4] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[5] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[6] X. Tian, J. Zhang, Z. Ma, Y. He, J. Wei, P. Wu, W. Situ, S. Li, and Y. Zhang, "Deep lstm for large vocabulary continuous speech recognition," *arXiv preprint arXiv:1703.07090*, 2017.

[7] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[8] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Acoustics, speech and signal processing (icassp), 2014 ieee international conference on*. IEEE, 2014, pp. 4087–4091.

[9] L. Tóth, "Combining time-and frequency-domain convolution in convolutional neural network-based phone recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 190–194.

[10] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, C. L. Y. Bengio, and A. Courville, "Towards end-to-end speech recognition with deep convolutional neural networks," *arXiv preprint arXiv:1701.02720*, 2017.

[11] P. Ghahremani, B. BabaAli, D. Povey, K. Riedhammer, J. Trmal, and S. Khudanpur, "A pitch extraction algorithm tuned for automatic speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2494–2498.

[12] L. Welling and H. Ney, "Formant estimation for speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 6, no. 1, pp. 36–48, 1998.

[13] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, 2017, pp. 3859–3869.

[14] P. Warden, "Speech commands: A public dataset for single-word speech recognition." *Dataset available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz*, 2017.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.