MLP Coursework 2: Learning rules, BatchNorm, and ConvNets

s1773005

Abstract

The experiment investigates the influence of different learning rules and batch normalization by using deep neural networks on EMNIST Balanced. We first construct a baseline model and compare the effect of RMSProp and Adaptive Moment Estimation(Adam) and batch normalization. Then we implement convolutional networks and compare the performance with deep neural networks.

1. Introduction

We explore the classification of images of handwritten digits using deep neural networks and convolutional networks with the EMNIST Balanced dataset (Cohen et al., 2017). The EMNIST Balanced dataset includes training dataset with shpae of (100000, 784), validation dataset with shape of (15800, 784)and test dataset with shape of (15800, 784). The training dataset is used to train our model and the validation dataset is used to modify our parameters in order to get a reliable and generalizable results. At last, we use test dataset to test the performance with the parameters we modified. We reset random number generator and data provider states on each run to ensure the reproducibility of our results.

At fist we implement deep neural networks. The report is aim to investigate performance of different activation functions and learning rate. We choose the simple and quick deep neural network architecture with stochastic gradient descent learning rule as our baseline model. Then we compared the accuracy of models with different learning rules, such as RMSProp (Tieleman & Hinton) and Adaptive Moment Estimation(Adam) (Kingma & Ba, 2014) with baseline model. Later, we implement batch normalization in our model in order to investigate the impact of using it. We use the test set at the end to assess the accuracy of the deep neural network architecture that we judge to be the best.

Then we implement convolutional networks with convolutional layers and max-pooling layers. The report is aim to investigate performance of different number of convolutional layers.

All the models are contain 784 dimensions of inputs, 100 dimensions of hidden layer and 47 dimensions of outputs. The performance of the model is compared by the error and accuracy of the training set and the validation set.

2. Baseline systems

In this section, we need to compare different activation functions and modify learning rate in order to get a simple and quick model architecture with not bad performance. This kind of network architecture with stochastic gradient descent learning rule will be our baseline model.

First, we explore the performance of the model with less sensitive parameters like activation functions. We compare Restricted Linear Unit(ReLU) (Nair & Hinton, 2010), Leaky ReLU (Maas et al., 2013), Exponential Linear Unit(ELU) (Clevert et al., 2015) and Scaled Exponential Linear Unit(SELU) (Klambauer et al., 2017) with error and accuracy of validation dataset. The running time of each activation functions is the running time of each epoch. In order to ensure the speed of the model, we set the learning rate to 0.1. The result can be seen by the Table 1 and Figure 1.

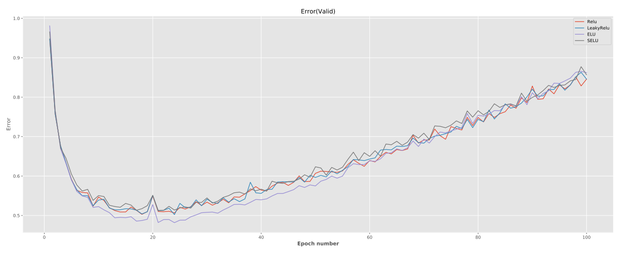

Figure 1. Error of validation dataset with different activation functions

Activation Function	Error	Acc	Time(s)
Relu	.823	.818	3.03
LeakyRelu	.846	.815	3.63
ELU	.856	.823	5.01
SELU	.857	.815	4.36

Table 1. Error and accuracy of validation dataset with different activation functions

From the Table 1, we can find that the ELU has the highest accuracy but cost the longest time. The Relu has a not bad accuracy with the shortest running time. The Elu has higher error that Relu. The others are all very slow and have low accuracy. Therefore, we choose Relu as our baseline model activation function.

Also, from the Figure 1, we can find that all of the activation function has over fitting. It may because we have a high learning rate. Therefore, we modify our learning rate in the following experiments.

We compare the performance of models with different learning rate with 0.1, 0.05, 0.01 and 0.001. The results can be seen by Figure 2 and Table 2.

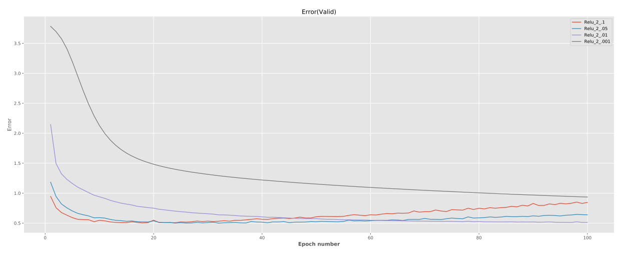

Figure 2. Error of validation dataset with different learning rate

Learning Rate	Error	Acc	Time(s)
0.1	.823	.818	3.03
0.05	.647	.826	4.28
0.01	.507	.836	2.97
0.001	.936	.733	2.82

Table 2. Error and accuracy of validation dataset with different learning rate

From the Figure 2, we can find that when we decrease the learning rate, it fixed the over fitting problem. Especially for the 0.01 and 0.001 learning rate, the error of validation dataset is pretty well with nearly no over fitting.

Then, from the Table 2, we can find the 0.01 learning rate has the best accuracy of validation dataset and also very short running time. Also the 0.01 learning rate has shorter running time, it has lower accuracy and higher error. This is because when the learning rate is very low and have no over fitting, we need to have more number of epochs to train our model in order to get higher accuracy. However, this will increase the running time.

Therefore, we choose 0.01 learning rate with 2 hidden layers and 100 hidden units as our baseline model. The learning rule is stochastic gradient descent and activation function is Relu.

3. Learning rules

We used two learning rules during this experiments: RMSProp and Adaptive Moment Estimation(Adam), then we compared these two learning rules with the baseline models stochastic gradient descent. As to SGD, the learning rate is constant during training. However, as to RMSProp and Adam, the learning rate will change with different gradients, which will decrease the running time and avoid getting stuck in local minimum.

The SGD learning rule has the following form:

Algorithm 1 Stochastic gradient descent

```
**Require:** Learning rate $\varepsilon_k$, Initial parameter *theta*
**while** stopping criterion not met **do**
    Sample a minibatch of $m$ examples from the training set $x^{(1)}, ..., x^{m}$ with corresponding targets $y^{(i)}$.
    Compute gradient estimate:
    $\hat{g} \leftarrow +\frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$.
    Apply update: $\theta \leftarrow \theta - \varepsilon\hat{g}$
**end while**
```

Although the SGD enables our model to jump to new and potentially better local minima, it has trouble navigating ravines and very easy to stuck in this kind of local minima. Therefore, we can add some momentum to it and adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters, which can give two other learning rules named RMSProp and Adam. The RMSProp learning rule has the following form:

Algorithm 2 RMSProp

```
**Require:** Global learning rate $\varepsilon$, decay rate $\rho$
**Require:** Initial parameter $\theta$
**Require:** Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers
Initialize accumulation variable $r = 0$
**while** stopping criterion not met **do**
    Sample a minibatch of $m$ examples from the training set $x^{(1)}, ..., x^{m}$ with corresponding targets $y^{(i)}$.
    Compute gradient: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$.
    Accumulate squared gradient: $r \leftarrow \rho$r+$(1-\rho)$g$\bigodot$g.
    Compute parameter update: $\triangle\theta = -\frac{\varepsilon}{\sqrt{\delta + r}} \bigodot g$.
    Apply update: $\theta \leftarrow \theta + \triangle\theta$.
**end while**
```

We can see that the RMSProp not only add momentum to the gradient but also divides the learning rate by an exponentially decaying average of squared gradients. We can see that if we just divides the learning rate by the sum of the squares of the gradients, it will accumulate the squared gradients in the denominator, which will cause the learning rate become infinitesimally small during training.

Here is another learning rules named Adam. The relation between Adam and RMSProp is that RMSProp with momentum generates its parameter updates using a momentum on the rescaled gradient, whereas Adam updates are directly estimated using a running average of first and second moment of the gradient (Kingma & Ba, 2014). The Adam learning rule has the following form:

```
Algorithm 3 Adam
  Require: Step size $\varepsilon$ (default: 0.001)
  Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in [0,1). (default: 0.9 and 0.999 respectively)
  Require: Small constant $\delta$ used for numerical stabilization (default: $10^{-8}$)
  Initial parameters $\theta$ Initialize 1st and 2nd moment variables s = 0, r = 0 Initialize time step t = 0
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $x^{(1)}, ..., x^{m}$ with corresponding targets $y^{(i)}$.
    Compute gradient: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$.
    $t \leftarrow t + 1$
    Update biased first moment estimate: $s \leftarrow \rho_1 s + (1 - \rho_1) g \odot g$
    Update biased second moment estimate: $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
    Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
    Compute parameter update: $\Delta\theta = -\varepsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$
    Apply update: $\theta \leftarrow \theta + \Delta\theta$.
  end while
```

We then use 2 hidden layers and 100 hidden units model with learning rate equal to 0.01 to compare the performance of the three learning rules.

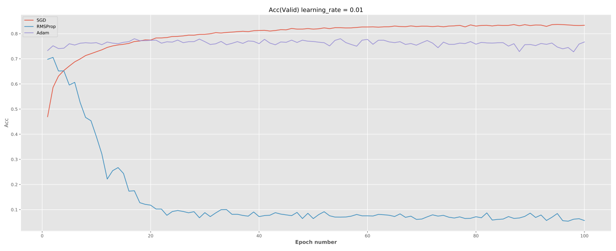

Figure 3. Accuracy of validation dataset with different learning rules

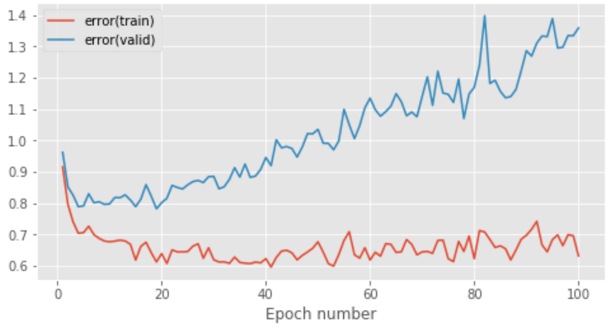

Figure 4. Error of validation dataset with Adam learning rules

From Figure 3, we can find that the SGD has the highest accuracy and the RMS even has a worser result than beginning. From Figure 4, we can find that the amplitude of the error of Adam learning rule is very large. This is because, as to RMSProp and Adam, 0.01 learning rate is large. The RMSProp learning rule with 0.1 learning rate cannot even find the local minima. The Adam learning rule with 0.1 learning rate is hard to find the local minima as it always cross the minima with the large learning rate. Therefore, we need to have a smaller learning rate so that the model will not cross any possible local minimum. We choose 0.001 learning rate for RMSProp and Adam learning rules and compare the error and accuracy of validation dataset with the model using SGD learning rule and 0.01 learning rate.

Learning Rules	Error	Acc	Learning Rates
SGD	.507	.836	.01
RMSProp	4.27	.072	.01
RMSProp	1.55	.801	.001
Adam	1.36	.758	.01
Adam	1.24	.811	.001

Table 3. Error and accuracy of validation dataset with different learning rate

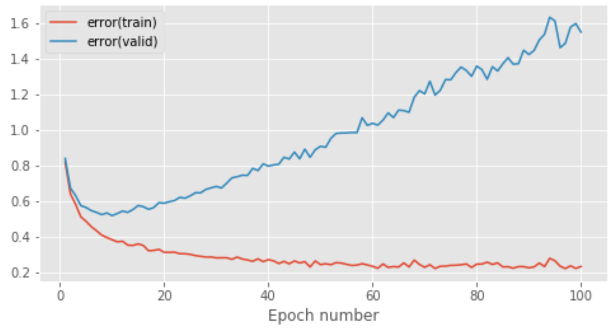

Figure 5. Error of validation dataset with RMSProp learning rules

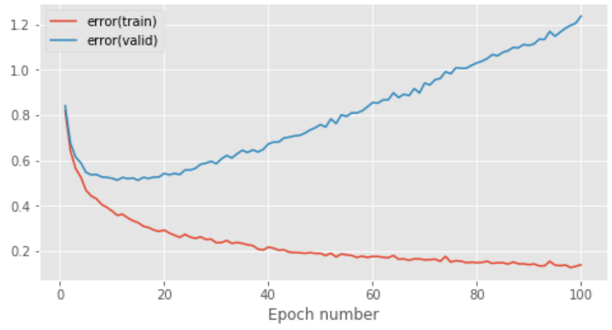

Figure 6. Error of validation dataset with Adam learning rules

From Table 3 we can find that when we change the learning rate to 0.001, the error and accuracy of validation dataset are much better for RMSProp and Adam learning rules. The error lines are smoother than before and the RMSProp can find the local minima. However, the error is still larger and the accuracy is still smaller than baseline model with SGD learning rule. From Figure 5 and Figure 6 we can find that, with 0.001 learning rate, the two learning rules are all over fitting, which may causes the accuracy of RMSProp and Adam smaller than baseline model. Also, from the baseline experiments, we can find that 0.01 learning rate is the best for our model, so we need to find some other methods to fix the problem of RMSProp and Adam learning rules with 0.01 learning rate. It seems that we need to use dropout at this situation. However, dropout will significantly increase the running time by careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. Therefore, we can use batch Normalization, which allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout (Ioffe & Szegedy, 2015).

4. Batch normalisation

Here we use batch normalisation to our baseline model and also the RMSProp and Adam learning rule model with 0.01 learning rate. The batch normalisation has the following form:

Algorithm 4 Batch normalisation

```
    **Input:** Values of $x$ over a mini-batch: $B = x_1 ... m$
    Parameters to be learned: $\gamma, \beta$
    **Output:** $y_i = BN_{\gamma,\beta}(x_i)$
    $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
    $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
    $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
    $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$
```

The batch normalisation uses minibatch statistics to normalise activations of each layer which can help over fitting problem. It also allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout (Ioffe & Szegedy, 2015). Therefore, we can use 0.01 learning rate for both RMSProp and Adam learning rules. The result can be seen by the following figures:

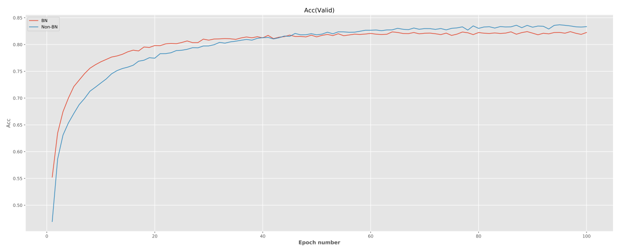

Figure 7. Accuracy of validation dataset with or without batch normalisation of baseline model

From the Figure 7, we can find that batch normalisation helps less to Stochastic gradient descent as the baseline model without batch normalisation has better accuracy of validation dataset. This is because baseline model do not have over fitting problem and batch normalisation will condense the information from the input image, which influences the training process of baseline model.

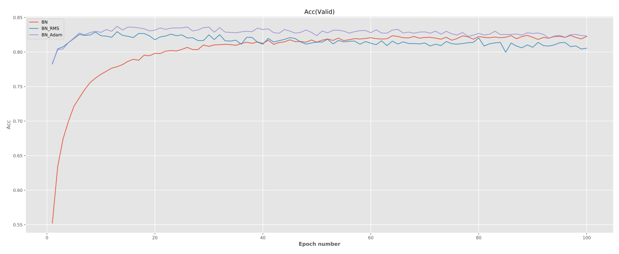

Figure 8. Accuracy of validation dataset with different learning rules

However, from the Figure 8 we can find that batch normalisation helps a lot to RMSProp and Adam learning rules with 0.01 learning rate. The accuracy of RMSProp is much more normal than before and the accuracy of Adam is also become better, even greater than baseline model. This is because batch normalisation help more to the over fitting problem than briefly decrease learning rate.

5. Convolutional networks

We implement convolutional networks in order to increase the accuracy of our model. We choose one and two convolutional layers with max-pooling layers to train our model. The convolutional layers is aim to extract features from the input, which can decrease the number of parameters and the amount of calculation as they share weights of the same feature. As to the max-pooling layers, they condense information from the previous layer with the max value.

We used the similar model architecture as before, with 2 hidden layer, 100 hidden units and 0.01 learning rate to implement the convolutional networks. We set convolutional kernels of dimension 5*5 with stride equals to 1 and pooling regions of 2*2 with stride equals to 2. As to two convolutional layers, we use 5 feature maps in the first convolutional layer and 10 feature maps in the second convolutional layer.

The results can be seen by the following figures:

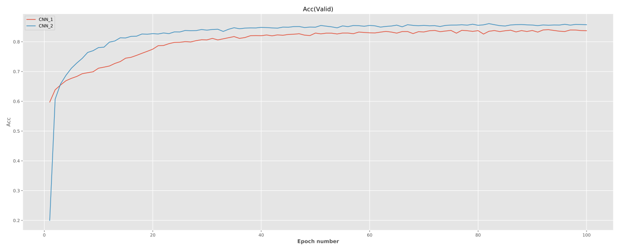

Figure 9. Accuracy of validation dataset with 1 convolutional layer

From the Figure 9 we can find that the 1 convolutional layer has much better accuracy of validation dataset than before, and there is no over fitting. There are two important ideas behind the convolutional neural networks. The first one is sparse interactions. Traditional neural network layers use the parameters describing the interaction between each input unit and each output unit, which will cause a large number of parameters to train. However, the convolutional networks use meaningful features with only tens or hundreds of pixels instead of the original input image with thousands or millions of pixels. This means that we only need to train a small number of parameters, which can improve its statistical efficiency. The second one is parameter sharing. Traditional neural network only use each weights once when computing the output of a layer. Nevertheless, as to convolutional neural network, the weights can be learned from only one set and can shared these weights with the weights of the same features. This can improve the efficiency of calculation.

Also, we can find that with 2 convolutional layers, the model has better accuracy and lower error of validation dataset than 1 convolutional layer model. This is because the model can extract much more features from the input image with 2 convolutional layers, which can help the model distinguish very similar images.

6. Test results

We first choose the best deep neural network on the EMNIST test set. We choose the model with 2 hidden layers, 100 hidden units, 0.01 learning rate and Adam learning rule with batch normalisation. This is because it has the best accuracy with no over fitting. The result can be seen by the following figure.

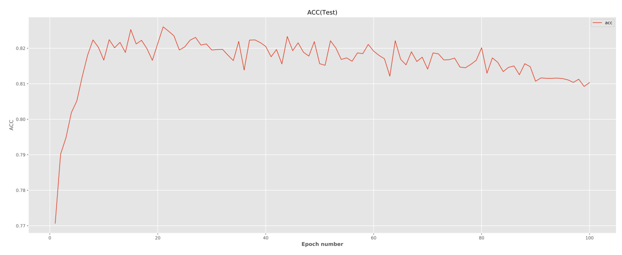

Figure 10. Accuracy and error of test dataset with Adam learning rules

We can see that the model has nearly .82 accuracy with nearly no over fitting. This is a good accuracy comparing models with other parameters. However, the amplitude of the accuracy is very large, which means the result is not very stable and it may because the 0.01 learning rate is still a little bit large. If we want have better accuracy, we can choose convolutional neural networks.

Then we choose the best convolutional network. We choose the 2 convolutional layers with max-pooling layers model and set the same learning rate and same number of hidden layers and hidden units as deep neural network for the same reason. The result can be seen by the following figure.

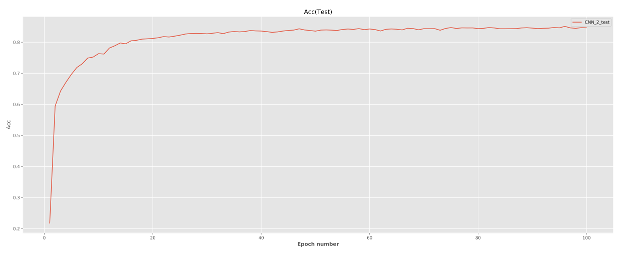

Figure 11. Accuracy of test dataset with 2 convolutional layers

We can see that the model has nearly .85 accuracy with absolutely no over fitting. This is a better accuracy than before. Also, the accuracy is very smooth, which means more stable and reliable result. This is because the convolutional neural network has less parameters to train so that they can have 0.01 learning rate, which is the best learning rate for this networks architecture.

7. Conclusions

In conclusion, comparing with other activation functions, the ReLU layer has the better performance to be a baseline model. Although the Elu layer has better accuracy, it cost longer time to finish the training. Furthermore, as there is an over fitting problem for 0.1 learning rate, we choose 0.01 learning rate as our baseline model after comparing some different learning rate.

Then, we compared the performance of the model with different learning rules. As to 0.01 learning rate, it is too large for the model with RMSProp learning rule to find even local minima. Therefore, we choose 0.001 learning rate for the RMSProp and Adam learning rule. However, with 0.001 learning rate, there is over fitting problem for both the learning rule and the accuracy of both models are all smaller than baseline. The results tell us that 0.01 learning rate is proper for this network architecture and we need to find other methods to fix both amplitude problem and over fitting problem. Therefore, we choose batch normalisation in the next experiments.

We choose 0.01 learning rate with 2 hidden layers and 100 hidden units as our basic model architecture. We compare the performance of stochastic gradient descent, RMSProp and Adam learning rules with or without batch normalisation. We find that batch normalisation helps a lot to RMSProp and Adam learning rules but helps less to stochastic gradient descent. This is because the over fitting only happened to the RMSProp and Adam learning rules. Batch normalisation will condense the information from the input image, which influences the training of the stochastic gradient descent model.

We also implement convolutional neural networks in order to increase the accuracy of the validation dataset. Both 1 convolutional layers and 2 convolutional layers have better accuracy than deep neural networks and 2 convolutional layers has the best accuracy over all. This is because sparse interactions and parameter sharing and 2 convolutional layers can extract much more features from the input image than 1 convolutional layers.

At last, we use test dataset to test the best deep neural model and convolutional neural model. The accuracy of convolutional neural model is better than the deep neural model.

In further experiments, we can add some activation function to the convolutional neural model to find that weather it can increase the accuracy. Also, we need to have some better algorithm to increase the speed of the model.

References

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Tieleman, T and Hinton, G. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report, Technical report, 2012. 31.