Performance Measure of Multilayer Perceptron in Handwritten Character Recognition

Devyash Sanghai
Department of Computer and Information Science
University of Florida
Gainesville, United States of America
devyashsanghai@gmail.com

***Abstract*—This paper provides a review of the application of multiplayer perceptron in character recognition using the MNIST dataset. The MNIST dataset consists of 70,000 images of digits 0-9. We implement a 2 & 1 hidden layer neural network and Compare the performance of the trained network in various different Hyper parameter settings. We provide empirical results regarding the number of Perceptron in the hidden layer. We apply regularization technique such as dropout and early stopping to avoid overfitting. We further have a look at procedures such as PCA algorithm and Down Sampling in the context of increasing the performance measure of the neural network. We further compare the performance of the multilayer perceptron in Stochastic Gradient Descent and Stochastic Gradient Descent with Momentum.**

***Index Terms*—MNIST. Multilayer Perceptron, Stochastic Gradient Descent, Regularization, Dropout technique, Stochastic Gradient Descent Momentum,**

I. Introduction

A. Multi-Layer Perceptron

A multilayer perceptron also called as a neural network is a supervised learning algorithm. It learns a function by learning from the training data set.

$$f(\cdot): R^m \rightarrow R^o$$

m: Dimension of the input dataset. MNIST has 28X28=784 dimensions

o: Dimensions of the output.

f(.): Nonlinear Function

MLP learns a nonlinear function approximator and hence can be used for both classification and regression. It consists of multiple perceptrons. It has mainly three layers:

- Input layer
- Hidden Layer
- Output Layer. Input and the output layer, there can be one or more non-linear layers, called hidden layers[1].
-

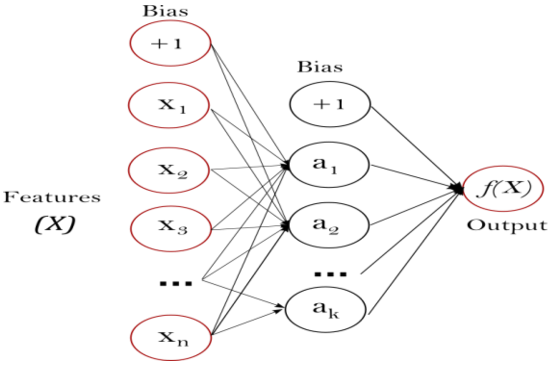


Fig. 1. A Multi-Layer Perceptron

The input layer is made up of a set of neurons representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + \ldots + w_mx_m$, followed by a nonlinear activation function $g(\cdot): R \rightarrow R$ - like the Relu function. As implemented in the project. The output layer receives the values from the last hidden layer and transforms them into output values. [1]

The advantages of Multi-Layer Perceptron are:

- Ability to learn non-linear models.
- Ability to learn models in an online manner using partial fit methodology.

The disadvantages of Multi-Layer Perceptron (MLP) include:

- MLP with hidden layers has a non-convex loss function where there exists more than one local minimum. Therefore, different random weight initializations can lead to different validation accuracy.
- MLP needs tuning of hyper parameters such as the number of hidden neurons, layers, and epochs.
- MLP is dependent on feature scaling. [1]

B. MNIST Data Set

The MNIST database of handwritten digits has a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from MNIST. The digits have been size-normalized and centered in a fixed-size image.

Fig: Different Images of the data set as contained in the MNIST dataset.

II. REGULARIZATION

Regularization is a technique used to avoid overfitting data on the learning data. By default, every MLP tries to learn convoluted complex inter-relations between data, while training. But becomes a big problem as an ever subset of real world data has its own complex inter-relations that may not be present in the real world data, synonymous to noise.

Regularization technique is used to avoid the above-stated problem. In this paper, we look the performance of the MLP under various regularization methods.

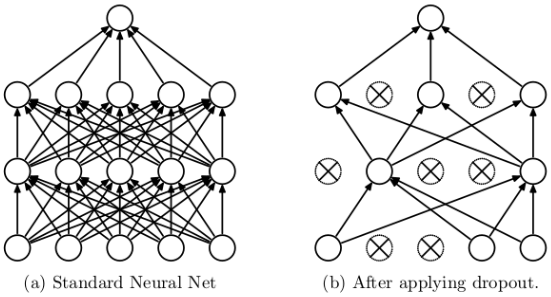
a) Validation-based early stopping (Holdout-based early stopping)

We apply Lutz Prechelt Early Stopping Technique with modifications as shown below, Early Stopping Technique refers to a method to separate the training dataset into 2 parts, Training and Validation and perform validation after few iterations to determine when the data is overfitting. We have broken our MNIST data set into 55,000 images for training and 5,000 images for Validation. The generalization error is calculated on the validation set to determine to overfit. We are checking for generalization error or in our case accuracy after every **5 epochs and since we are using stochastic gradient descent algorithm, We have kept a margin of 0.001** to keep room for wiggle of the stochastic nature of the algorithm. We have implemented the modified Prechelt naive implementation of holdout-based early stopping in the following steps[2]:

1. Separate the original dataset into validation and training. We have broken out MNIST dataset into training: validation ratio as 55 :5.
2. Training on the training set and then evaluate based on validation set after some epochs. We have taken 5 epochs to reduce running time.
3. Stop the training when the accuracy on validation starts decreasing with the margin of 0.001..
4. Use the weights the network had in that previous step as the result of the training run.

b) Dropout Technique

Dropout technique is basically randomly dropping a certain number of perceptrons from the neural network while training. This prevents the neural network from co-adapting. The whole network while training is thinned multiple times, and at testing, the effect of approximate of multiple networks is summarized Dropout introduces a significant amount of gradient noise.[3]



(a) Standard Neural Net

(b) After applying dropout.

Fig: The standard Neural Network and After applying Dropout.

c) Other Regularization Techniques.

Other Types of Regularization Techniques mainly consist of introducing various different weight penalties during training. Like L1 and L2 regularization and soft weight sharing (Nowlan and Hinton, 1992).

With unlimited computation, the best way to “regularize” a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by its posterior probability given the training data. [3]

III. IMPROVING THE TIME COMPLEXITY

a) Down Sampling

A down sampling - pooling layer is inserted in between the Neural Network architecture to reduce the size of the input parameters and hence control the model from overfitting. Maximum pooling works by running a square filter of length N over an image and then taking the maximum in that area. It is mainly used to reduce the feature map dimensionality for computational efficiency, hence improve actual performance.

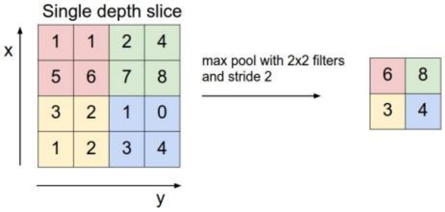


Fig: Running a 2X2 Max pool filter on an image slice.

Major benefits of down sampling can be summarized below as:

- Reduction in the parameters to ensure better computational time.
- It ensures translational equivariance meaning that the model can tolerate small translational changes in the input.

b) PCA Algorithm

PCA algorithm is used to reduce the number of dimensions in the input data. It reduces the dimensions by calculating the eigenvectors and eigenvalues and then reducing the dimensions to the direction of maximum variance. It is a statistical procedure that uses an orthogonal transformation that converts input observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The principal components are then decreased. The conversion is done is such a way that the first component has the maximum variance and this variability is across all the components.

IV. STOCHASTIC GRADIENT DESCENT ALGORITHM

Stochastic Gradient descent algorithm is a stochastic approximation of the gradient descent optimization method for minimizing an objective function.which is written as a sum of differentiable functions. Stochastic gradient descent tries to find minimums by iteration.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

Stochastic gradient descent algorithm (SGD) performs a parameter update for each training example x(i) and label y(i):

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.
SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, it has been shown that when we slowly decrease the learning rate, SGD shows the same convergence behavior as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

Its code fragment simply adds a loop over the training examples and evaluates the gradient w.r.t. each example.

c) Momentum

If the objective has the form of a long shallow ravine leading to the optimum and steep walls on the sides, standard SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains. Momentum is one method for pushing the objective more quickly along the shallow ravine. The momentum update is given by,

$$v\theta=\gamma v+\alpha\nabla\theta J(\theta;x(i),y(i))=\theta-vv=\gamma v+\alpha\nabla\theta J(\theta;x(i),y(i))\theta=\theta-v$$

In the above equation, vv is the current velocity vector which is of the same dimension as the parameter vector $\theta\theta$. The learning rate $\alpha\alpha$ is as described above, although when using momentum $\alpha\alpha$ may need to be smaller since the magnitude of the gradient will be larger. Finally $\gamma\in(0,1]\gamma\in(0,1]$ determines for how many iterations the previous gradients are incorporated into the current update. Generally, $\gamma\gamma$ is set to 0.5 until the initial learning stabilizes and then is increased to 0.9 or higher.[5]

V. SIMULATION RESULTS AND ANALYSIS

a) Selection of Number of Perceptrons & Hidden Layer

i) Input Layer

The input layer of the neural network depends on the number of features in the data. We can add another perceptron node for a bias term, although it is not compulsory. In our model, we would be using 784(+1 bias) perceptrons in the input layer.

ii) The Output Layer

The number of perceptron in the output layer is dependent on the on the model configuration Like the Input layer, every NN has exactly one output layer. Determining its size (number of neurons) is simple; it is completely determined by the chosen model configuration. There can be many modes, such as Machine Mode: Return (Yes/No) or Regression Mode: Return a value. In machine mode, the output layer has only 1 node
If the neural network is used as a classifier, it has one perceptron per class label. In MNIST data set we would be using 10, as we have 10 digits to recognize.

iii) The Hidden Layer

For most situations having 2 or 1 hidden layer is sufficient for the correct training of the Multi-layer perceptron. As it causes an increase in computational time complexity.

The next natural question is how many perceptrons we have to select? Again there are a lot of theory as to the selection of the number of perceptrons, it is best that we select our hyperparameters based on the empirical result.

I have run the Stochastic Gradient Descent algorithm for 5 epochs just to get an approx. value of accuracy on the testing data. (Learning rate as 0.01)

Number of neurons in hidden layer 1	Number of neurons in hidden layer 2	Accuracy	Time taken
650	650	94.27	87.34s
256	256	92.65	35.36s
10	10	10.33	8.38s
650	N-A	89.67	63.59s
256	N-A	88.2	32.47s
10	N-A	0.323%	7.9s

From the above results, we can make out there is a tradeoff in the selection of neurons and the time required to train the neural network.

We can also infer that even for just 5 epochs the tradeoff between 2 hidden layers vs 1 hidden layer is close to **~5%.** Whereas the time required is **~37%** more.

For Section of a number of neurons in 1 hidden layer the time required is reduced drastically, however, the point to recognize here is that since we are running the training only for 5 epochs the Cost function must not have converged. hence we can get better accuracy if we increase the number of epochs. But at the same time, we would need to determine the stopping criteria to avoid overfitting.

b) Improvement of Generalization using the Dropout

Dropout technique as described in the Regularization section was applied on the data set. (With dropout probability = 0.9) It was observed that the accuracy increased to 97.56% , but the time required to reach the accuracy

c) Down Sample the Data Set

After applying the down sampling in the neural network for the following hyperparameter:

Learning Rate:	0.01
Number of neurons in hidden layer 1	650
Number of neurons in hidden layer 2	650
Maxpoolsize	14X14=196
Epoch	74(Selected after early stopping)

The accuracy of the Network was about 97.6% The accuracy of the Data after down sampling data. Which is less than the accuracy found without down sampling.

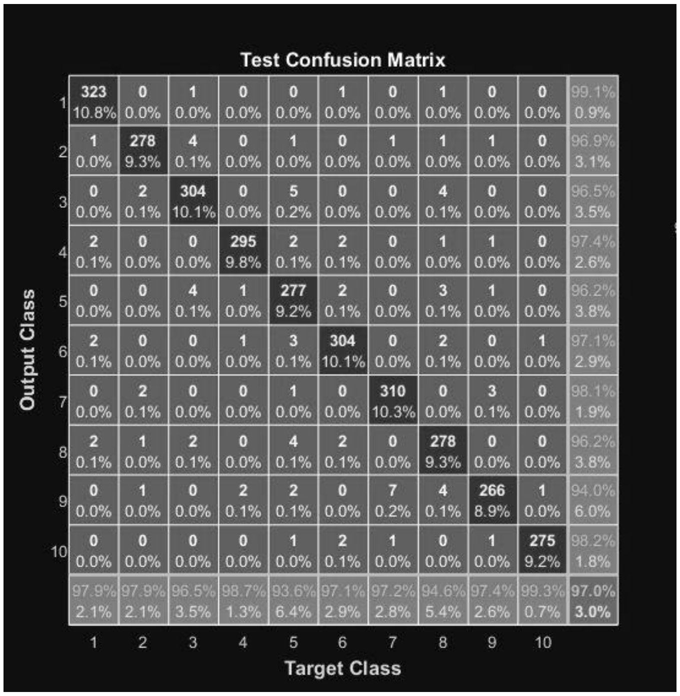
c) Improvement using Early Stopping Technique

Without early stopping technique it is very difficult to predict at which epoch the MLP would give the best output, as well as avoid overfitting the data.

Number of neurons in hidden layer 1	Number of neurons in hidden layer 2	Accuracy	Time taken
650	650	97.36	7:12 mins
256	256	97.12	8:23 mins
10	10	91.45	10:32 mins
650	N-A	97.3	5:57 mins

d) Confusion Matrix for Testing Dataset

Below is the confusion matrix on the Test data of 10,000 images. I have a single hidden layer with 650 perceptron using Stochastic Gradient descent algorithm.



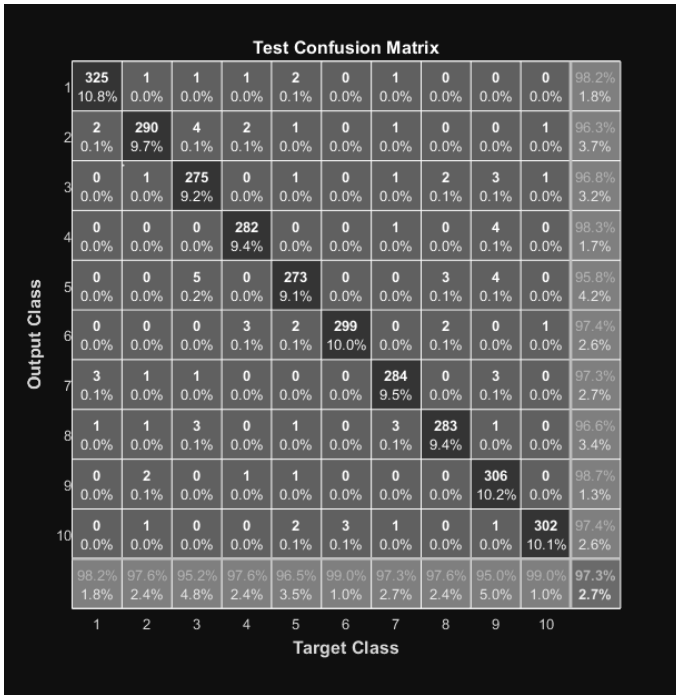
Test Confusion Matrix

	1	2	3	4	5	6	7	8	9	10	
1	323 10.8%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	99.1% 0.9%
2	1 0.0%	278 9.3%	4 0.1%	0 0.0%	1 0.0%	0 0.0%	1 0.0%	1 0.0%	1 0.0%	0 0.0%	96.9% 3.1%
3	0 0.0%	2 0.1%	304 10.1%	0 0.0%	5 0.2%	0 0.0%	0 0.0%	4 0.1%	0 0.0%	0 0.0%	96.5% 3.5%
4	2 0.1%	0 0.0%	0 0.0%	295 9.8%	2 0.1%	2 0.1%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	97.4% 2.6%
5	0 0.0%	0 0.0%	4 0.1%	1 0.0%	277 9.2%	2 0.1%	0 0.0%	3 0.1%	1 0.0%	0 0.0%	96.2% 3.8%
6	2 0.1%	0 0.0%	0 0.0%	1 0.0%	3 0.1%	304 10.1%	0 0.0%	2 0.1%	0 0.0%	1 0.0%	97.1% 2.9%
7	0 0.0%	2 0.1%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	310 10.3%	0 0.0%	3 0.1%	0 0.0%	98.1% 1.9%
8	2 0.1%	1 0.0%	2 0.1%	0 0.0%	4 0.1%	2 0.1%	0 0.0%	278 9.3%	0 0.0%	0 0.0%	96.2% 3.8%
9	0 0.0%	1 0.0%	0 0.0%	2 0.1%	2 0.1%	0 0.0%	7 0.2%	4 0.1%	266 8.9%	1 0.0%	94.0% 6.0%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	2 0.1%	1 0.0%	0 0.0%	1 0.0%	275 9.2%	98.2% 1.8%
	97.9% 2.1%	97.9% 2.1%	96.5% 3.5%	98.7% 1.3%	93.6% 6.4%	97.1% 2.9%	97.2% 2.8%	94.6% 5.4%	97.4% 2.6%	99.3% 0.7%	97.0% 3.0%

Output Class

Target Class

Fig: Test Confusion Matrix for Stochastic Gradient Descent Algorithm



Test Confusion Matrix

	1	2	3	4	5	6	7	8	9	10	
1	325 10.8%	1 0.0%	1 0.0%	1 0.0%	2 0.1%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	98.2% 1.8%
2	2 0.1%	290 9.7%	4 0.1%	2 0.1%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	1 0.0%	96.3% 3.7%
3	0 0.0%	1 0.0%	275 9.2%	0 0.0%	1 0.0%	0 0.0%	1 0.0%	2 0.1%	3 0.1%	1 0.0%	96.8% 3.2%
4	0 0.0%	0 0.0%	0 0.0%	282 9.4%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	4 0.1%	0 0.0%	98.3% 1.7%
5	0 0.0%	0 0.0%	5 0.2%	0 0.0%	273 9.1%	0 0.0%	0 0.0%	3 0.1%	4 0.1%	0 0.0%	95.8% 4.2%
6	0 0.0%	0 0.0%	0 0.0%	3 0.1%	2 0.1%	299 10.0%	0 0.0%	2 0.1%	0 0.0%	1 0.0%	97.4% 2.6%
7	3 0.1%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	284 9.5%	0 0.0%	3 0.1%	0 0.0%	97.3% 2.7%
8	1 0.0%	1 0.0%	3 0.1%	0 0.0%	1 0.0%	0 0.0%	3 0.1%	283 9.4%	1 0.0%	0 0.0%	96.6% 3.4%
9	0 0.0%	2 0.1%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	306 10.2%	0 0.0%	98.7% 1.3%
10	0 0.0%	1 0.0%	0 0.0%	0 0.0%	2 0.1%	3 0.1%	1 0.0%	0 0.0%	1 0.0%	302 10.1%	97.4% 2.6%
	98.2% 1.8%	97.6% 2.4%	95.2% 4.8%	97.6% 2.4%	96.5% 3.5%	99.0% 1.0%	97.3% 2.7%	97.6% 2.4%	95.0% 5.0%	99.0% 1.0%	97.3% 2.7%

Output Class

Target Class

Fig: Test Confusion Matrix for Stochastic Gradient Descent Algorithm with momentum

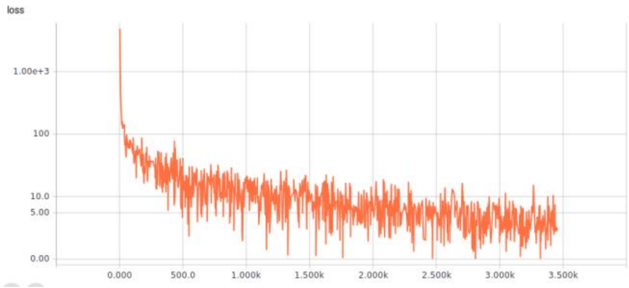
A confusion matrix is used to describe the performance of a classification model on test data set for which the true values of labels are already known.

We can gain information that there is more than one way, digits can be written due to a person's different writing styles. Our model breaks the image into unique image classes. As seen, the gained clusters are labeled in terms of truth, based upon the how large frequency with which a particular cluster is attached with 1 digit.

We have plotted a confusion matrix which quantifies the probability that a given digit is clustered “properly”, and if not “improperly” in the sense that it is in a cluster dominated by the same digit type. It quantifies the “purity” of the clusters, in the view, the image is associated with the same digit type.[4]

As we can see the accuracy of SGD is 97% for 600 perceptron in 1 hidden layer and the accuracy with SGD momentum is close, but better at 97.3%.

e) Learning Curve for Training

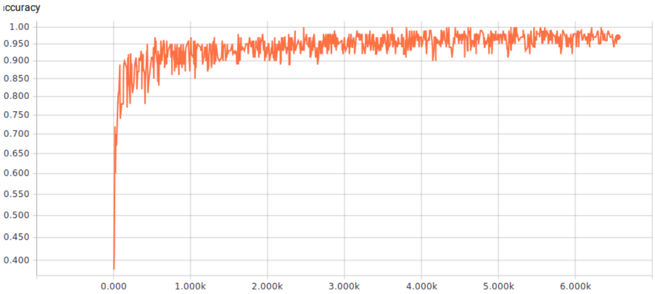


x_axis: Iterations
y_axis: Cost
Fig: Learning curve for Training as run on Tensor Flow

The learning curve above shows the performance of the neural network during testing as well as validation. We can infer from the plot that during training the cost function is decreasing during all the epochs. While in Validation as soon as the cost begins to increase , we apply early stopping technique.

The learning curve is set to not smooth to show the stochastic nature of the gradient descent algorithm.

f) Accuracy Curve for Validation



x_axis: Iterations
y_axis: Cost
Fig: Accuracy curve for Training as run on Tensor Flow

The accuracy curve shows that since this is a stochastic gradient descent algorithm . The curve is set to not smooth, to show the stochastic nature of the algorithm. As shown in the confusion matrix, it can be seen that the accuracy reaches to 97.0.

VI. Conclusion

We have implemented a Multilayer perceptron with multiple different regularization techniques. Furthermore, we compared the performance of the trained network in various configurations, We applied stochastic gradient descent algorithm, without and regularization, Then showed that accuracy increased after applying drop out. We later on compared the accuracy with Stochastic gradient descent algorithm with and without momentum, to conclude that with the momentum the accuracy increases a bit. The best accuracy was reached with SGD momentum with dropout and early stopping at 98.54%.

VII. REFERENCES

[1] 1.17. Neural Network Models (supervised)¶." 1.17. Neural Network Models (supervised) — Scikit-learn 0.18.1 Documentation. N.p., n.d. Web. 12 Dec. 2016.

[2] Prechelt, Lutz; Geneviève B. Orr (2012-01-01). "Early Stopping — But When?". In Grégoire Montavon, Klaus-Robert Müller (eds.). *Neural Networks: Tricks of the Trade*. Lecture Notes in Computer Science. Springer Berlin Heidelberg. pp. 53–67. ISBN 978-3-642-35289-8. Retrieved 2016-12-11.

[3] Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Nitish Srivastava, Geoffrey Hinton,Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov -Department of Computer Science.

[4] On the Integration of Topic Modeling and Dictionary Learning Lingbo Li Mingyuan Zhou Guillermo Sapiro† Lawrence Carin Department of Electrical and Computer Engineering, Duke University, Durham, NC

[5] "Unsupervised Feature Learning and Deep Learning Tutorial." Unsupervised Feature Learning and Deep Learning Tutorial. N.p., n.d. Web. 12 Dec. 2016.