Multi-class Classification using Multi-layer Neural Network

Chen Chen Mingxuan Li
480458339 470325230

17th April 2019

Abstract

*This project numerically compares the robustness of two nonnegative matrix factorisation (*NMF*) algorithm [?] based on multiplicative update rules when contaminated by large magnitude Gaussian, Poisson and Salt & Pepper noise. Section ?? briefly reviews relevant work in the field of* NMF*, which found these multiplicative update rules are very sensitive to initialisation. Section 2 describes the two algorithms and their theoretical properties, proposes a solution to make the multiplicative update rules less sensitive to initialisation, and details the statistical tools we used to compare the robustness of the two algorithm. Our simulation results in Section 3 agree with the theoretical properties of the two algorithms. Future work involves embedding the proposed multiple initialisations into the* NVIDIA GPU *based parallel-programming model.*

Contents

1 Introduction

The aim of this study is to build a multi-layer neural network application to perform a multi-class classification and use various optimisation approaches to improve the prediction accuracy on the given dataset. The task provides a training dataset and a testing dataset, the training dataset given for this task consists of 128 features for 60,000 samples, the test dataset contains 10,000 samples with same amount of features as training dataset, Given the

naive multi-layer neural network implementation in the early stage of our research, we also tried various optimisation methodologies to further improve the prediction accuracy, these methods include different activation functions, weight decay, stochastic gradient descent methods, mini-batch training, batch normalisation and dropout.

As we all know a Neural Network is a computational learning system that uses a network of functions in multiple layers to understand and translate a data input of one form into a desired output, usually in another form (Dee [1999]). It learns from processing many labelled examples that are supplied during training and using this answer key to learn what non-linear and complex characteristics of the input are needed to construct the correct output. In this supervised training, a network processes the inputs and compares its actual outputs against the expected outputs. For each iteration the cost will be propagated back through the network, to adjust the weights in each layer. This process is repeated until the best accuracy is achieved; This supervised learning algorithm is often referred to as a back-propagation algorithm, which is useful for training multi-layer Neural Networks. Since Neural Networks have shown a performance breakthrough in the area of object detection and scene classification, so this study will focus on identifying the best network and optimisation methodologies for this purpose (Lecun et al. [1998]).

In this report, we will first introduce all these optimisation methodologies and their common usages, followed by the experiment setup and results for the research as well as a detailed comparison of results by applying various methodologies. In the end we will also discuss importance of all these methodologies and how our neural network model benefits from them.

2 Methods

We build a fully connect feed-forward neural network for the classification task with a cross-entropy loss and L2 weight decay regularisation. Our neural networks implements a Xavier initialisation and six hidden layers.

For each of the three activation layers $i = 1, 2, 3$, this report uses $\vec{t}_i = (t_{i1}, t_{i2}, \ldots)$ to denote the the inputs of the ith activation layer[1], and uses $\vec{z}_i = (z_{i1}, z_{i2}, \ldots)$ to denote the outputs of those. We apply batch normalisation to normalise the input data from $\vec{t}_1$ to $\tilde{\vec{t}}_1$. This is followed by a nonlinear activation function. After the first hidden layer, our neural network applies

[1] We use ( ) to denote column vectors, and [ ] to denote row vectors.

an additional batch normalisation, followed by a drop out to the normalised outputs. We then apply an densely connected layer with a ReLU activation function, an additional drop out layer. The neural network finishes with a Softmax activation layer.

2.1 Cross validation score and early stopping prevents overfit

We partition our dataset into $50,000$ training samples and $10,000$ test samples. For each epoch, we first train our model using the training samples and then compute the cross-validation accuracy using the test samples. After 200 epoch, we find the number of iteration `n_iter` corresponding to the maximum cross-validation accuracy as our early stopping point. Using all the $60,000$ samples, we then train our model again `n_iter` epoches to obtain our final neural network model.

2.2 Cross-entropy with weight decay improves the model robustness

Define $\vec{y}_k := (y_{1k}, y_{2k}, \ldots, y_{10k})$ as the one hot encoding of the training label for the kth sample and $\vec{p}_k := (p_{1k}, p_{2k}, \ldots, p_{10k})$ as the corresponding predicted probabilities. To train the neural network, we apply the cross-entropy loss

$$L(\vec{p}|\vec{y}) = -\sum_{k}\sum_{j=1}^{10} y_{jk} \log p_{jk}. \tag{1}$$

Although the structure of our neural network is relative simple, it still has too many parameters. For example, if we set the number of hidden nodes in each hidden layer to be 160, then the number of parameters to estimate in the weights W_1, W_2 and W_3 is $128 * 160 + 160 * 160 + 160 * 10 = 47,680$. The total number of parameters including biases b_i and batch normalisation parameters γ_i and β_i is more than $47,680$. As we only have $50,000$ training samples, allowing entries of the weights W_i to be arbitrary real number overfits the dataset. To encounter this ill-proposed problem, we penalise the complexity of our model by a L2 regularisation term as proposed by Hoerl

and Kennard [1970]

$$L = -\sum_{k}\sum_{j=1}^{10} y_{jk}\log p_{jk} + \lambda\sum_{i=1}^{3}|W_i|^2\,, \tag{2}$$

where hyperparameter λ is a positive real number. In comparison with loss function (1), the regularised loss function (2) penalises the model to have large weight matrices W_i and hence reduces the model complexity.

The reason to choose cross-entropy loss lies in its derivative. Taking the derivative of loss function (2)

$$\frac{\partial L}{\partial z_{3j}} = \sum_{k} p_{jk} - y_{jk} + \lambda\frac{\partial}{\partial z_{3j}}\sum_{i=1}^{3}|W_i|^2\,.$$

This derivative function has a key property that many other loss functions like mean square loss do not have—the larger the errors $\sum_k p_{jk} - y_{jk}$ is, the faster all the neurons will learn.

2.3 Xavier initialisation speeds up convergence

We initialise the weight matrix W_i for $i = 1, 2, 3$ using Monte Carlo method from a uniform distribution bounded by $\pm\sqrt{6/\left[\dim(z_i) + \dim(t_i)\right]}$ as suggest by Glorot and Bengio [2010]. We use 0s as the initial condition for bias vector $\vec{b}$. As a result, the weight decay term helps to prevent the neural network from overfitting.

We also have experimented to use the normal distribution initialisation suggested by Glorot and Bengio [2010], but it is outperformed in accuracy and speed. So, it is not discussed here.

Xavier initialisation makes sure the initialised weights and biases are not too far away from the optimal weights and biases [Glorot and Bengio, 2010]. This is essential because a poorly initialised optimisation problem usually either ends up with a solution that is far away from global optimum, or even diverges.

2.4 Batch normalisation prevents internal covariance shift

We normalise the inputs t_{ij} for layers $i = 1, 2$. When training using mini-batch B as defined in Section 2.7.1, we use $\mu_B^{(j)}$ and $\sigma_B^{(j)^2}$ to denote the sample mean and sample variance for the mini-batch B. Here, index j denotes one dimension of the input data. Moreover, we define γ_{ij} and β_{ij} $(i = 1, 2)$ as real parameters to be learnt by backward proporgation. The batch normalisation layer iteratively normalise each dimensions j of the input batch as

$$\tilde{t}_{ij} = \gamma_{ij} \frac{t_{ij} - \mu_B^{(j)}}{\sqrt{\sigma_B^{(j)^2} + 10^{-8}}} + \beta_{ij} \text{ where } i = 1, 2. \tag{3}$$

When predicting test labels, we use the training means $\frac{1}{|B|} \sum_B \mu_B^{(j)}$ and training variances $\frac{1}{|B|} \sum_B \frac{|B|}{|B|-1} \sigma_B^{(j)^2}$ to normalise the test inputs. Our neural network applies this batch normalisation layer to prevents internal covariance shift [Ioffe and Szegedy, 2015]. Later on, Santurkar et al. [2018] find that batch normalisation makes the optimisation landscape considerably smoother.

2.5 Drop out performs model averaging

We apply drop out to the inputs of the 2nd and 3rd densely connected layers $\vec{t}_2$ and $\vec{t}_3$. When training our neural network, the drop out layer randomly ignores neurons in vectors $\vec{t}_2$ and $\vec{t}_3$ with a given probability hyperparameter p. When predicting labels for cross validation samples and testing samples, we multiply weights W_i where layer index $i = 2, 3$ by hyperparameter p. This drop out layer is equivalent to model averaging, and forces our neural network to learn a more robust set of parameters that are useful in conjunction with many distinct random subsets of the neurons [Hinton et al., 2012].

2.6 Activation functions map input to desired output range

Our neural network model has three densely connected layers. These three layers respectively implements a nonlinear activation function, a ReLU activation function, and a Softmax activation function.

2.6.1 Nonlinear activation

The first densely connected layer is chosen to be either of a hyperbolic function

$$\vec{z}_1(\tilde{\vec{t}}_1) = \tanh(W_1\tilde{\vec{t}}_1 + \vec{b}_1) \tag{4}$$

or a sigmoid function

$$\vec{z}_1(\tilde{\vec{t}})_1 = \vec{1} \Big/ \left[\vec{1} + \exp\left(W_1\tilde{\vec{t}}_1 + \vec{b}_1\right)\right] \tag{5}$$

where matrix W_1 is a 160×128 weighting matrix and vector $\vec{b}$ is an 160-dimensional bias vector. Here, the division / is an elementwise operator. This layer provides non-linearity to our neural network.

2.6.2 ReLU activation function

The second densely connected layer is ReLU

$$\vec{z}_2(\tilde{\vec{t}}_2) = \max(0, W_2\tilde{\vec{t}}_2 + \vec{b}_2).$$

ReLU guarantees the output of the layer is sparse. The nonlinear activation functions (4) and (5) do not have this property. The sparseness makes the neural network more computationally efficient [Livni et al., 2014].

2.6.3 Softmax activation function

The Softmax activation functions takes a vector of real numbers as the input, and convert it into a categorical distribution as the output. Define matrix $\mathbf{1}$ to be a 10×10 matrix of ones so that multiplying by matrix $\mathbf{1}$ is equivalent to summing up all rows. The activation function is the probability mass function of this categorical distribution

$$\vec{z}_3(\vec{t}_3) = \exp(W_3\vec{t}_3 + \vec{b}) \Big/ \left[\mathbf{1}\exp\left(W_3\vec{t}_3 + \vec{b}\right)\right], \tag{6}$$

which sums up to one. We assign the index of probability vector $\vec{z}_3(\vec{t}_3)$ corresponding to the largest probability as the classification result for the corresponding sample.

2.7 Stochastic gradient descent finds the optimal weights

We use gradient descent with mini-batch and momentum to iteratively find optimise our model.

2.7.1 Mini-batch improves the speed of computation

Traditional batch gradient descent uses all samples to estimate the gradient of the loss function. Estimating such a gradient using this way may be a very computationally expensive procedure. Alternatively, stochastic gradient descent uses only one sample to estimate the gradient. Although it economises the computational cost, the non-smooth nature makes the convergent much slower. We applied mini-batch as a compromise between the true gradient descent and the stochastic gradient descent methods. Instead of using either all samples or only using one sample to estimate the gradient, we use $1,000$ training samples (i.e. a "mini-batch" B) at each step. In comparison with stochastic gradient descent, mini-batch naturally takes advantage of the vectorisation nature of the `numpy` library and converges much faster [Goyal et al., 2017].

2.7.2 Momentum accelerates rate of convergence

We use η to denote the learning rate, α to denote the momentum coefficient and ∇ to denote the gradient operator. In the sth iteration, assume that the first order Taylor expansion of the loss function (2) informs to update the weight matrix W_i as $W_i^{(s+1)} := W_i^{(s)} - \eta \nabla L(W_i^{(s)})$, the gradient descent update with momentum proposed by Rumelhart et al. [1986] is

$$\Delta W_i^{(s+1)} = \alpha \Delta W_i^{(s)} - \eta \nabla L(W_i^{(s)}) \text{ and } W_i^{(s+1)} = W_i^{(s)} + \Delta W_i^{(s+1)}.$$

To initialise this update, we set $\Delta W_i^{(0)} := 0$. This autoregressive update proposal smooths the wiggliness caused by stochastic gradient descent [Rumelhart et al., 1986], and hence significantly accelerates the rate of convergence.

2.7.3 todo: adam

3 Experiments and results

3.1 Dataset

We illustrate our two NMF algorithms on two real-world face image datasets: `ORL` and `CroppedYaleB` (?). Both `ORL` and `CroppedYale` datasets contain multiple images of distinct subjects with various facial expression, lighting condition, and facial details. Images in ORL are cropped and resized to 92×112 pixels. We further rescale it to 30×37 pixels. Similarly, we reduce the size of images in `CroppedYale` to 42×48 pixels. For each dataset, we flatten the image matrix into a vector and append them together to get a matrix V with shape $d \times n$ where integer d is the number of pixels in one image and integer n is the number of images. In each epoch, we use 90% of data.

3.2 Experiment Setup

We apply two algorithms (NMF and KLNMF) with four categories of noises (no noise, Gaussian noise, Poisson noise and Salt & Pepper noise), which results in eight combinations in each epoch. In each epoch, we randomly select 90% of samples to train NMF algorithms and evaluate three metrics on reconstructed images. The training will terminate when the error reaches the minimum error, or the maximum iteration is reached. The minimum error and maximum iteration are hyperparameters which we learn from iterative experiments. Our code saves the learning errors versus the number of iterations so that we could draw the plot and observe the convergence of the learning process. We increase the number of epochs and calculate the average metrics and confidence intervals.

3.2.1 multiprocessing speed up hyperparameter tuning

3.2.2 Hardware and software

3.2.3 Early stopping decides maximum number of iterations

3.3 Experiments Results

3.3.1 Optimal hyperparameters

Table of results and hyperparameters with optimal and suboptimal coefficients.

Figure 1: Put accuracy vs iteration here with the best possible hyperparameter, and parameters without each of normalisation, dropout, momentum

Table 1: Results and parameters of the best four setups.

Experiment	Test 1	Test 2	Test 3	Test 4	Test 5
Runtime(mins)	2.3	2.7	3.6	17.2	10.8
Accurarcy	89.9%	89.7%	89.8%	90.0%	89.3%
Initialisation	Xavier	Uniform			
$(-1, 1)$	Uniform				
$(-1, 1)$	Xavier	Xavier			
Batch size	1500	1500	1500	1500	1500
Hidden layer nodes	160	150	150	900	160
Activation function	tanh	tanh	tanh	tanh	sigmoid
Weight decay rate	0.0007	0.0007	0.0007	0.007	0.007
Momentum rate	0.9	0.9	0.92	0.9	0.9
Dropout rate	0.95	1.0	1.0	0.5	0.95
Learning rate	0.11	0.05	0.05	0.11	0.11
Early stopping iterations	44	54	66	158	282

Table 2: Results and parameters of different setups for 89.87 results.

Experiment	Test 1	Mini-Batch	Weight decay	Momentum	Drop out 50%
Runtime(minutes)	2.3	0.5	8.6	0.8	7.6
Accurarcy	89.9%	85.5%	89.4%	88.6%	87.5%
Initialisation	Xavier	Xavier	Xavier	Xavier	Xavier
Batch size	1500	1500	1500	1500	60000
Hidden layer nodes	160	160	160	160	160
Activation function	tanh	tanh	tanh	tanh	tanh
Weight decay rate	0.0007	0.0007	0.0007	0	0.007
Momentum rate	0.9	0.9	0	0.9	0.9
Dropout rate	0.95	0.5	0.95	0.95	0.95
Learning rate	0.11	0.11	0.11	0.11	0.11
Early stopping iterations	44	9	198	16	177

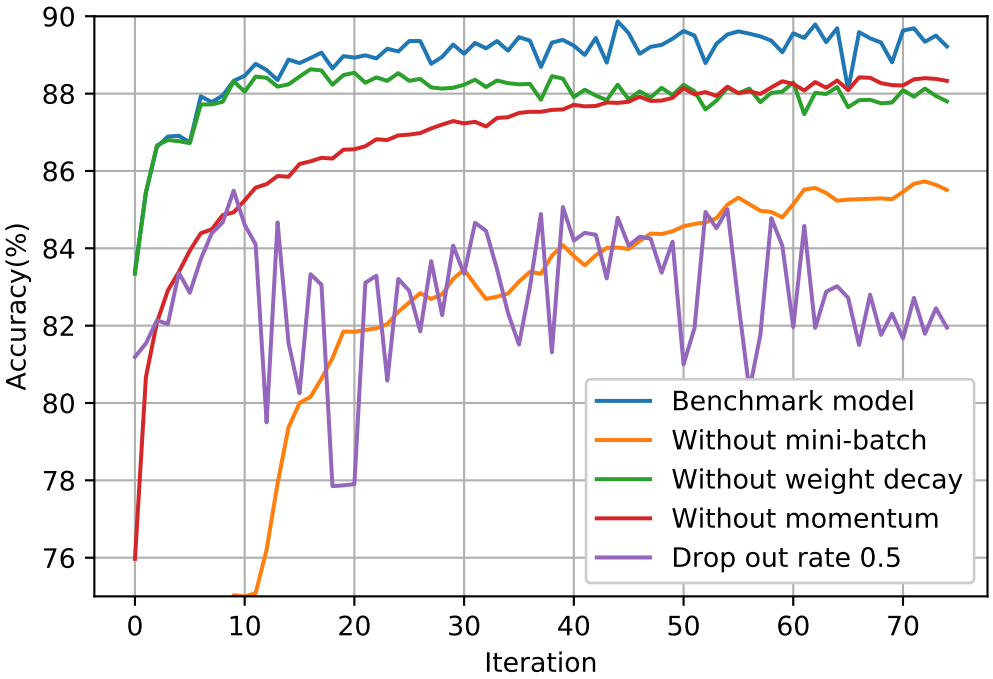

Figure 2: Results and parameters of different setups for 89.87 results.

3.4 Experiments Results

3.4.1 Vectorisation dramatically improves the speed of training

3.4.2 Batch normalisation significantly improves the accuracy

3.4.3 Weight decaying prevents overfitting

3.4.4 dropout does not help here

4 Discussion

4.1 Why relu helpful

4.2 Why batch normalisation helpful

4.3 Weight decaying

4.4 Why drop out is not helpful

4.5 Why early stopping

4.6 Why momentum

5 Conclusion

In conclusion, our numerical simulation supports the theoretical results that NMF based on objective function (**??**) is more robust to Gaussian noise and KLNMF based on objective function (**??**) is more robust to Poisson noise. The two algorithms have a similar performance against Pepper & Salt noise. However, the NMF algorithm reconstructs image better without noise and converges much faster with the multiplicative update rule when comparing with KLNMF. Also, simulation results find both of the multiplicative update rules are sensitive to the initial values of matrices W and H. Section 2 proposes a solution based on parallel programming to solve this problem. However, this solution requires high performance computer so that the algorithm converges in a reasonable amount of time.

Recently, NVIDIA released their Cuda package which parallelises algorithms using GPU. This package improves the speed of parallelisable algorithms, including many NMF algorithms, by a factor of > 100. As a computationally expensive procedure, our suggestion of using multiple initialisations will be more novel if a GPU version based on Cuda could be designed.

References

Build with ai. https://deepai.org/machine-learning-glossary-and-terms/neural-network, 1999. Retrieved: 2019-04-14.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, November 1998. doi:10.1109/5.726791.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

doi:10.1080/00401706.1970.10488634. URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488634.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/ioffe15.html.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2483–2493. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.pdf.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 855–863. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5267-on-the-computational-efficiency-of-training-neural-networks.pdf.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning

representations by back-propagating errors. *Nature*, 323:533–, October 1986. URL http://dx.doi.org/10.1038/323533a0.