MLP Coursework 2: Learning rules, BatchNorm, and ConvNets

s1679450

Abstract

This paper focus on the classification task on EMNIST dataset. First we set up the baseline multilayer neural network model, discussed the selection of activation function, number of hidden layer, regularization method and dropout rate. After we have setup the baseline model, we started to investigate the influence of different learning rules. Then we evaluated the performance of Batch normalization layer. Finally we discussed the performance of convolutional layer and how different convolutional layer parameters will influence the result.

1. Introduction

Classification tasks are one of the most widely applied field of machine learning. Tasks classifying handwritten letters is quite frequently used to evaluate the performance of machine learning systems. EMNIST(Cohen et al., 2017) is a new dataset for handwritten letters recognition, which provided us the processed picture and its label. We are going to explore how different mechanisms will work on EMNIST dataset. We will train our system on a training set of 100000 examples, evaluate the performance on validation set of 15800 examples and test the model using test set of 15800 examples.

Baseline We will first evaluate simple neural network classifiers and use the tuned result as baseline for following discussions. We are going to first tune some two properties of our neural network model: activation function and depth. In order to suppress overfitting, we are going to explore how regularization and dropout will contribute when eliminating overfitting. Finally we will summarize and set up the baseline model for the following part.

Learning rules For different learning rules, we are going discuss about their difference and carry out experiments to check their actual performance on the EMNIST dataset. Also we will try to interpret and try to explain the result of experiment.

Batch Normalization We will discuss about the concept of batch normalization and how batch normalization can improve the performance of our model. We will carry out the experiment and explain the result.

Convolutional neural networks Finally, we will explain about convolutional network models and discuss about efficient ways to implement a convolutional layer. Also, we will evaluate the performance of our convolutional layer and try to explain how some basic parameters of convolutional layer (like number of filters and size of filters) is going to influence the performance based on experiment result.

2. Baseline systems

- Learning Algorithm: Stochastic gradient descent with 0.1 learning rate.
- Batch size: 100.
- Training epoch number: 50.

The hyper parameters above we won't change during this part. There are few properties we want to investigate in the following part : activation function, hidden layer number, regularization method, dropout rate.

2.1. Activation function selection

Definition In this section, we compared different type of activation function to find out their performance on the EMNIST dataset.

Rectified Linear Units (ReLU) (Nair & Hinton, 2010) is a quite widely used activation function. The advantage of ReLU is that the model would be easier to optimize if its behavior similar to linear (Goodfellow et al., 2016).

$$\mathrm{relu}(x) = \max(0, x)$$

Leaky ReLU (Maas et al., 2013) The problem with ReLU is that ReLU sometimes might not working when the activation is zero. Leaky ReLU is going to fix that problem.

$$\mathrm{lrelu}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases}$$

Exponential Linear Units (ELU) ELU(Clevert et al., 2015) is going to fix the similar problem as LeakyReLU.

$$\mathrm{elu}(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases}$$

Activation function	Accuracy
ELU	0.8270
SELU	0.8189
LRELU	0.8204
RELU	0.8186

Table 1. Accuracy of models with different activation function on test set

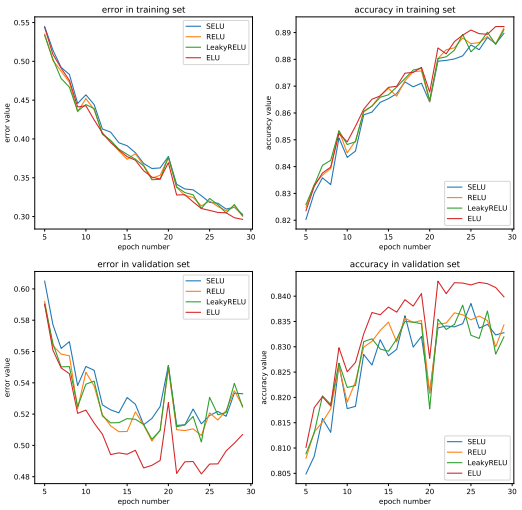

Figure 1. Comparison between models with different kind of activation functions.

Scaled Exponential Linear Units (SELU) SELU (Klambauer et al., 2017) is a variance of ELU, adding an extra tunable parameter λ.

$$\text{selu}(x) = \lambda \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases}$$

Result As Figure 1 shows, ELU have the best overall performance: both in validation set and training set. ELU always has better accuracy and lower cross-entropy error than other activation functions. In test set, ELU also has best accuracy. The reason might because the feature of ELU: compared with ReLU and LReLU, ELU can keep negative values and restrict these values in a certain range $[-\alpha, 0]$.

2.2. Depth of neural networks

Definition The depth of a neural network means the number of hidden layers between input and output layer. We are going to increase the depth of the neural network by adding hidden layers to original model.

Layer number	Accuracy
2	0.8270
3	0.8222
4	0.8193

Table 2. Accuracy of models with different hidden layer number on test set

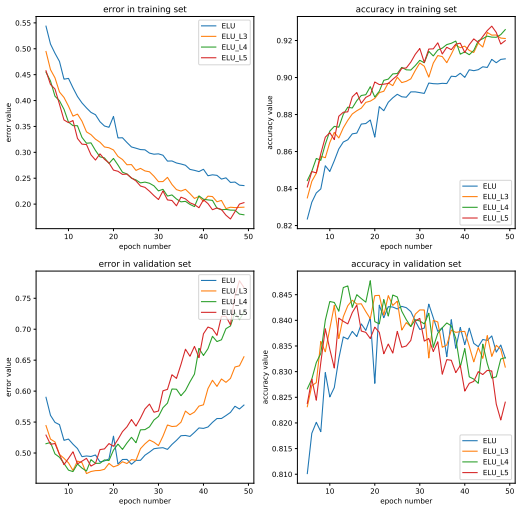

Figure 2. Comparison between models with different number of hidden layers.

Motivation Deeper network model can learn more complex decision boundaries(Goodfellow et al., 2016). Compared with MNIST dataset, the output class number of EMNIST increased from 10 to 48, which might need deeper network to draw the decision boundary.

Result We can see from the result in Figure 2, having deeper layer number can result in better accuracy but higher possibility to overfit. From the Table 2 we can see that the accuracy is getting lower. The reason is at current stage we are not applying any early stopping mechanism or use other methods to stop overfitting. So we finally choose the best overall performance one: the network with 4 hidden layer.

2.3. Regularization

Definition Regularization is basically adding regularization terms to the error function. Based on regularization type, they are penalizing specific features by increasing error. L1 regularization is adding the sum of absolute values of the weights to error. L2 regularization is the sum of the squares of all the weights in the network and then multiplied by $\frac{\lambda}{2n}$. Both of these regularization methods will penalize big weights.

Regularization Method	Accuracy
None	0.8193
L1Penalty(1e-05)	0.8247
L1Penalty(0.001)	0.7296
L2Penalty(0.0001)	0.8322
L2Penalty(0.01)	0.7196

Table 3. Accuracy of models with different regularization on test set

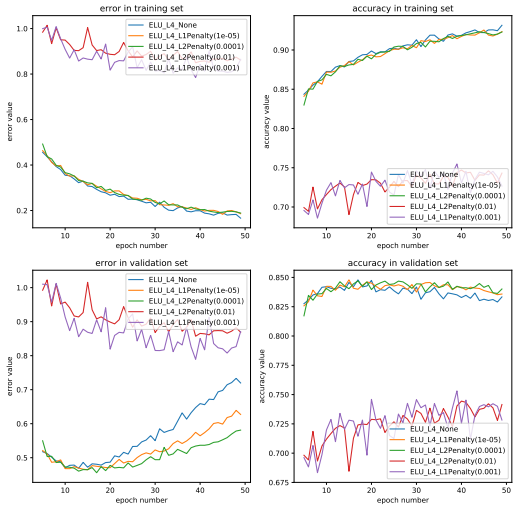

Figure 3. Comparison between models with different regularization methods.

Motivation We have discussed in previous part that deeper network can have higher possibility to overfit. In order to alleviate overfitting, we are going to use regularization.

Result The result is in Figure 3. In the error distribution, we can see that if regularization parameter is not small enough, then it will cause the error to stay in a high level and influence the final result of training. From the curve of error in validation set, we can see that regularisation successfully suppress the increase trend of validation set error. The one with best overall performance is L2 regularization with decay rate at 1e-4 (0.0001).

2.4. Dropout layer

Definition Dropout layer (Srivastava et al., 2014) is a simple layer which will randomly drop some of the unit and their connection to alleviate overfitting.

Motivation As stated in the Dropout paper, "This significantly reduces overfitting and gives major improvements over other regularization methods."(Srivastava et al., 2014).

Dropout percent	Accuracy
0.0	0.8193
0.1	0.8561
0.2	0.8433
0.3	0.8214

Table 4. Accuracy of models with different activation on test set

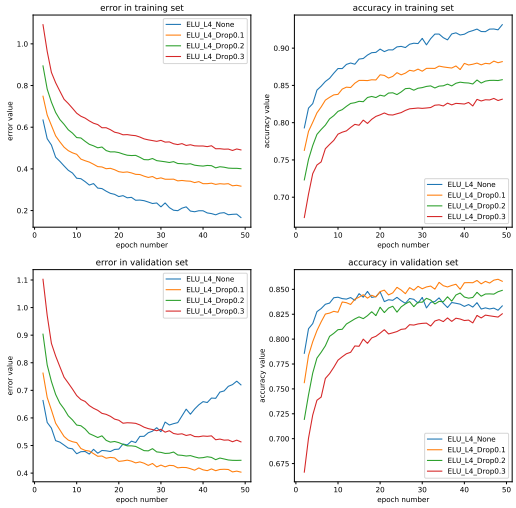

Figure 4. Comparison between models with different dropout rates.

Our baseline model now is suffering from overfitting, so we are going to try Dropout layer for stopping overfitting. But one important thing is higher dropout rate can result in slower training process (i.e. need more epoch to converge).

Result The result is showed in Figure 4 . Compared with the baseline, adding dropout layer can significantly reduce overfitting tread. The models with dropout trains slower in training set, but unlike the one without dropout, the curve never shows a trend to overfit in validation set.

2.5. Final baseline model

After all the discussions above, the final model we choose for baseline is as follows

- ELU activation function
- 4 hidden layers with 100 hidden units per layer
- L2 regularization (decay rate: 0.0001)
- Dropout layer (Drop 0.1)

3. Learning rules

Motivation In general, training a model is a process of finding the local minimum of error. In the past, we use gradient descent, which is simple. People have investigated multiple ways to improve the performance of learning rules.One big problem of the gradient descent rule we used before is that we need to tune the learning rate, which can cost a lot of time and fixed learning rates are not always going to be the optimum. We are going to explore some adaptive learning algorithms to check if they can give us a better result on our dataset.

Algorithm 1 Adam Learning Rule

Input: step size α, size m
Input: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Input: $f(\theta)$: Stochastic objective function with parameters θ
Input: θ: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1 st moment vector)
$v_0 \leftarrow 0$ (Initialize 2 nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converge **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t} + \epsilon$
end while
return θ_t

3.1. Definition

The original learning rule we use is Gradient Descent, which is simply moving the step (learning rate * gradient) in the negative gradient direction. One of the shortcomings of this method is that different weights can have different magnitude of gradients. So choosing a right global learning rate is hard. To solve these issues, we are going to use learning rules with adaptive learning rate.

RMSProp RMSProp (Tieleman & Hinton, 2012) is based on RProp(Riedmiller & Braun, 1992). RProp suggests that we can use only the sign of the gradient and move same step size. This can prevent the issue like some huge gradient suddenly appears. But RProp won't work for mini-batch. Because RProp is equivalent of using a learning rate of $\frac{1}{g_i}$(g_i is the gradient) where g_i can be quite different over different mini batch. Having random learning rate can make converge more difficult (Tieleman & Hinton, 2012). RMSProp can solve the problem by setting learning rate similar to previous mini batches.

$$S_i(0) = 0$$

$$S_i(t) = \beta S_i(t-1) + (1 - \beta) g_i(t)^2$$

Learning rule	Accuracy
Adam	0.8491
RMSProp	0.8508
SGD	0.8485

Table 5. Accuracy of models with different learning rule on test set

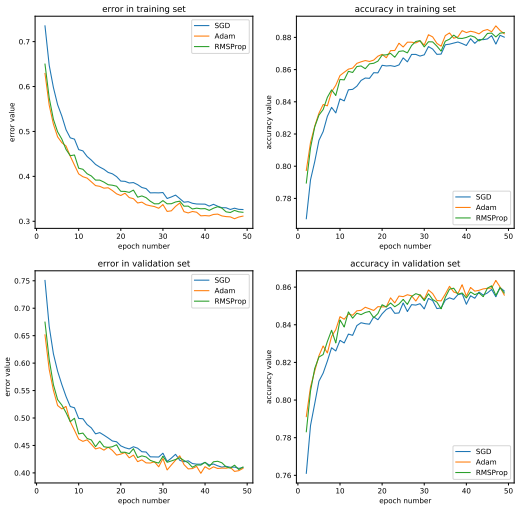

Figure 5. Evaluate Adam, RMSProp and SGD

$$\Delta w_i(t) = \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} g_i(t)$$

The equation above is how RMSProp updates weight.

Adam Adam (Kingma & Ba, 2014) is simply combining RMSProp with momentum. The algorithm of Adam learning rule is described as Algorithm 1.

3.2. Experiment result and analysis

The result is showed as Figure 5. We run the experiment on baseline model and the only variable is setting of learning rule. The learning rule we used is Gradient Descend (learning rate: 0.1), Adam (learning rate 0.001), RMSProp (learning rate 0.001).

This result can show that in our EMNIST dataset, Adam > RMSProp > SGD. Adam and RMSProp takes less epoch to converge. After 50 epochs, all the curves seems to have similar error and accuracy, that might because they reached a similar local optimum. RMSProp is better than Gradient descent, which proved that using only the sign of the gradient is a good trick for improving training process. Adam is slightly better than RMSProp, which might because the advantage of having momentum. Momentum can help better and quicker converge as Adam paper suggested.

4. Batch normalisation

Motivation "Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout." (Ioffe & Szegedy, 2015) Our model suffers a lot from overfitting, so we are going to try batch normalization and test if it can help our model get rid of overfitting.

4.1. Definition

Batch normalization (Ioffe & Szegedy, 2015) is a method to normalize layer inputs. The algorithm is described as Algorithm 2. (Ioffe & Szegedy, 2015)

4.2. Experiment result and analysis

Experiment setting For the experiment, we are going to insert BatchNormlization Layers in front of the non-linearity layers as suggested in the original paper(Ioffe & Szegedy, 2015).

The result of the experiment is shown in Figure 6.

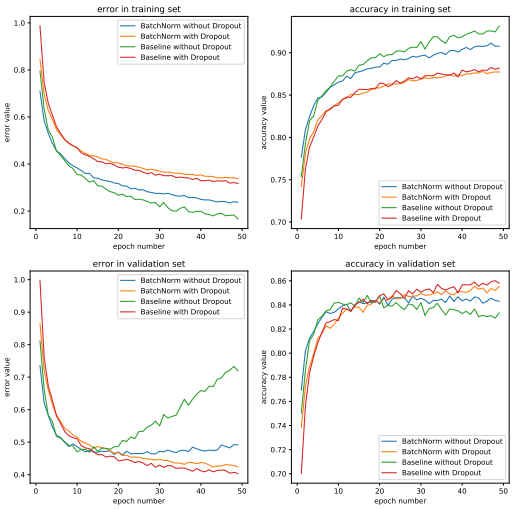

Figure 6. Evaluate batch norm.

Evaluating the feature of reducing overfit Adding batch normalization layer did increases training speed and reduced overfit. From the graph, we can compare the curve of "Baseline without Dropout" and "BatchNorm without Dropout". We can see that "Baseline without Dropout" start to overfit (we can tell from the increasing validation set error and decreasing validation set accuracy) after 30 epochs, while "BatchNorm without Dropout" didn't increase.

The reason of this phenomenon is that regularizers are simply adding terms to error function and penalize some features of the weights that we don't want to see (for example, big weight values). Adding a batch normalization layer can do the job of regularizers in another way: it normalized the value so that there won't be extremely huge values anymore.

Comparing with Dropout So we compared BatchNorm with dropout and without Dropout. We can have a look at Figure 6 and compare the curve of "Baseline with Dropout" and "BatchNorm with Dropout". We can see that initially, the one with BatchNorm improves more quickly, but after 10 epoch, the one with BatchNorm slowed down. This might because having a extra layer can slow down the training process.

5. Convolutional networks

Motivation Convolutional networks are quite widely used in modern machine learning systems, and it shows great performance especially in image recognition. So we are going to apply convolutional networks on our models.

Algorithm 2 Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = x_{1...m}$
Input: Parameters to be learned: β, γ
Output: $y_i = BN_{\gamma,\beta}(x_i)$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x-\mu_B)^2$$

$$\hat{x}_i \leftarrow \frac{x_i-\mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}}+\epsilon}}$$

$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$$

5.1. Convolutional layer

The idea of convolutional layer is using shared filters to detect features and produce feature maps as input to next layer. All the filters can be learned by back propagation when minimizing the error, but we still need to manually set filter size and number. The basic method is to apply filters to small windows of original picture and produce a feature map. Compared with the old method we use, which is simply flatten the picture into independent points, convolution layers can make use of the positional information and detect same feature in different place in the training example.

Implementing a convolution layer: a naive approach The most intuitive way to implement convolution layer is using for-loops. First loop over each example in mini-batch, for each example loop over filters. Then loop over sliding windows of filter size in original picture, and then multiply the windows matrix with filter matrix. This approach is quite easy to understand, but running quite slow in practice.

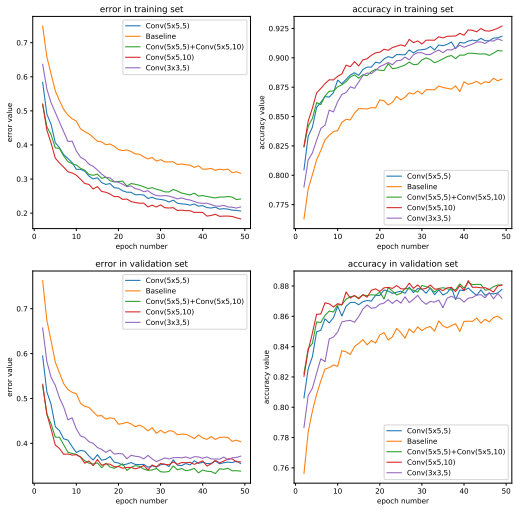

Figure 7. Comparing different form of convolutional neural network.

Implement a convolution layer: matrix manipulation. The for-loop version is quite slow and there is a better way to optimize it. The idea is, instead of use for-loop to move the sliding window and do dot product, we can expand the original matrix and gather all the possible locations that we can apply a filter to. This can be done by using im2col (Kumar Chellapilla, Oct 2006) After we get the expanded matrix, we can do a dot product on it. This optimized approach significantly boosted the performance. The original implementation takes 30 min 27sec for one epoch, while im2col version only needs 120 seconds.

5.2. Maxpooling layer

Maxpooling layer is basically shrinking the original matrix do use the max value of a sub matrix. This can reduce the size of feature map. The point of maxpooling is to shrink image size, and make significant features detected by convolutional layer less sparse.

Implementing Maxpooling layer: naive approach Doing for-loop again is the most simple and direct method. Loop over all examples and loop over all channels, then get the sliding window matrix, replace it with the maximum value inside of it.

Implementing Maxpooling layer: matrix manipulation Similar to convolutional layer, this time we also need to expend the original matrix, but instead of doing dot product, we are going to apply max operation.

5.3. Experiment Result and analysis

The result is in Figure 7, and the test set accuracy is in Table 6 .

Comparing with baseline We added 1 Convolutional Layer with 5 kernel number and 5x5 kernel size followed by an 2x2 2D Maxpooling Layer(curve "Conv(5x5,5)" in Figure 7) in front of the original baseline model. The network structure is as follows.

-> Convoltional Layer (1 channel, 5 kernel, 5x5 feature map)

-> ELU Layer

-> 2D Maxpooling Layer (2 x 2)

-> Baseline model

The improvement is quite significant. The reason I think is using convolutional layers did provided the following hidden layers with extra information which is helpful for classification, because convolutional layer can detect different features making use of the positional information using shared kernel.

Difference between different number of convolutional layers. We compared models with different number of convolutional layers. Figure 7 shows the curve. The curve "Conv(5x5,5)+Conv(5x5,10)" is a model with an extra Convolutional layer and Maxpooling layer. The network structure is as follows. Comparing it with model "Conv(5x5,5)", we can see that the deeper one has better performance. The reason can be that the second convolutional layer can help detect higher dimensional patterns as mentioned in (Lecun et al., 1998). First layer of convolutional network can detect low-level features like straight line, corners, the second convolutional layer can help form higher level features like cross, circle etc., because each window will contains more valid informations. For example, a 5x5 window in second convolutional layer contains extracted information(by maxpooling and kernel) of 14x14 space of original picture. With more higher dimensional features, the work of classification will become more easier, just like what we observed form the result.

-> Convoltional Layer (1 channel, 5 kernel, 5x5 feature map)

-> ELU Layer

-> 2D Maxpooling Layer (2 x 2)

-> Convolutional Layer (5 channel, 10 kernel, 5x5 feature map)

-> ELU Layer

-> 2D Maxpooling Layer (2 x 2)

-> Baseline model

Model	Accuracy
2 Conv layer	0.8680
1 Conv layer (3x3)	0.8671
1 Conv layer (5x5)	0.8671
Baseline	0.8485

Table 6. Accuracy of models with different activation on test set

Different filter size We compared the performance of convolutional networks with different filter size. The curve "Conv(5x5,5)" has 5x5 filter size as mentioned above. The curve "Conv(3x3,5)" has 3x3 filter size in the first convolutional layer. The results turns out to be the one with bigger filter size performs better. I think tuning this parameter should based on specific dataset. As we have discussed above, this convolutional layer is trying to detect low level features of the picture. For EMNIST dataset, 3x3 window is too small to detect even low-level features.

Different filter number We also compared the performance of convolutional layers with different filter number. In the figure, we can compare the curve "Conv(5x5,5)" with "Conv(5x5,10)". We can see the one with 10 kernel has generally better performance. This can because that the more kernels we use, the more different features we can detect.

6. Conclusions

Baseline For the baseline part. We can draw following conclusions from the experiment.

- **Activation function** ELU has the best overall performance than other activation functions.
- **Number of hidden layer** More hidden layer can detect more complex decision boundaries, but we should be careful when selecting different layer number because if the network is too deep, the possibility of overfitting will increase.
- **Regularization method** The parameter of regularization makes a big difference and adding L1 or L2 regularization is a good way to reduce overfit.
- **Dropout rate** Dropout out can significantly surpress overfitting trend, but higher dropout rate means slower converge speed. We need to find a balance between speed and performance.

Learning rule For the learning rule part, we compared the difference between Adam, RMSProp and Stochastic gradient descent. From the experiment, we can see that RMSProp is much better than the original gradient descent. Taking away the influence of gradient value does help when training. Adam has better performance than RMSProp proved that momentum does help training process to go in a right direction.

Batch Normalization We compared baseline networks with and without batch normalization. As a result of our experiment, we can see that adding Batch Normalization is a good way to accelerate training process and reduce overfit.

Convolutional network

- **Filter size** Filter size is relevant to the property of different datasets. In EMNIST evaluated in our experiment, using bigger filter can have better performance, the reason of which might be the bigger filter can include more details and identify more general patterns, but this quite depends on different datasets.
- **Convolutional layer number** More stacked convolutional layers can contribute to better performance, this result might because deeper convolutional layer can learn more high-level features of the picture.
- **Filter number** In our experiment with one convolutional layers, bigger filter number leads to better performance. Filters are used to detect specific feature, so more filters can lead to more low-level features of the original picture. This can provide more information, so the result can reasonably be better.

Future work

- **Visualization** In recent research with convolutional networks, there is a idea to retrieve what feature convolutional layer is actually extracting.
- **More Convolutional Network Experiments** There is still some place for our convolutional model to improve, for example, adding BatchNormalization, tune learning rate, learning algorithm and dropout rate etc.

References

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ArXiv e-prints*, November 2015.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017. URL http://arxiv.org/abs/1702.05373.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-Normalizing Neural Networks. *ArXiv e-prints*, June 2017.

Kumar Chellapilla, Sidd Puri, Patrice Simard. Performance convolutional neural networks for document processing. *Guy Lorette. Tenth International Workshop on Frontiers in Handwriting Recognition*, Oct 2006.

Lecun, Yann, Bottou, LÃl'on, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

Maas, Andrew L., Hannun, Awni Y., and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In Fürnkranz, Johannes and Joachims, Thorsten (eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814. Omnipress, 2010. URL http://www.icml2010.org/papers/432.pdf.

Riedmiller, Martin and Braun, Heinrich. Rprop - a fast adaptive learning algorithm. VII., 12 1992.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Tieleman, T. and Hinton, G. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.