MLP Coursework 2: Learning rules, BatchNorm, and ConvNets

s1546937

Abstract

Activation functions help Artificial Neural Networks to learn and make sense of the non-linearity between the inputs and outputs. Weight initialisation and regularisation techniques are ways to improve the model, which can be further extended using learning rule optimisation techniques and normalisation techniques. Deep neural networks are difficult to train and making a better learning model is to play around with different hyper parameters, optimisation and normalisation techniques, tweaking them and bringing a right fit among them.

This report extends to Convolution Neural Networks., mixing hyper parameters, optimisation and normalisation techniques and building multi-layer convolution neural networks.

1. Introduction

An Artificial Neural Network (ANN) is a computational model which uses Activation Function to learn and make sense of complicated and non-linear functional mapping between the inputs and response parameters. Following to activation function, we will be adding our experiments on different initialisation techniques and regularisers. This helps in forming the baseline for our our system.

With this baseline system, we will start exploring optimisation techniques and normlalisation technique, to understand there use and the hyperparameters required to build a better model. Is batch normalisation helps build a better model and what is the problem of internal covaiate shift (Ioffe & Szegedy, 2015), handled by Batch Normalisation.

Later we will be shifting to Convolution Neural Networks, popularly known as Covnets, and will look at its difference with ANNs. Further expanding in building Covnets and making it deep convolution neural networks.

We will be exploring EMNIST (Extended MNIST) (Cohen et al., 2017) Balanced dataset. The size of the training, validation, and test sets are 100000, 15800 and 15800 respectively, and we will be using this for all further modelling.

2. Baseline systems

In this section, we are comparing different activation functions, initialisation, regularisation techniques and other hyperparameters on EMNIST Balanced dataset. With this we will be building our baseline system.

We will be doing this in three parts: activation function, initialisation technique and regularisation technique. Here is the table 1 of hyperparameters we will be considering in all the three parts.

Hyper Parameters	Value
learning rate	0.1, 0.01
number of epocs	100
number of hidden layers	2
hidden dimension	100
non-linearity	Sigmoid,ReLU, Leaky ReLU, ELU, SELU
batch size	100
weight initialiser	fan-in, fan-out, fan-in-out
regularisation	L1, L2, Dropout
bias initialiser	Constant Initialiser
Inclusion Probability	0.5
Learning Rule	GradientDescentLearningRule

Table 1. Hyper Parameters used in training the model.

Activation Function Here is the comparison of activation function - Sigmoid, ReLU, leakyRelu, ELU and SELU. The model was trained for 100 epocs, with 0.1 as learning rate and Glorot as the weight initialiser.
The Figure 1 compares these activation function on the given hyper parameters, and shows the training and validation accuracy over the given dataset.
As we can see from the figure 1, leaky Relu is the best among all the activation function. Leaky ReLU continuously try to correct the bias shift which is introduced by previous updates of weights. After comparing on the EMNIST dataset using different activation functions, Leaky ReLU seems to be the better choice, as it being Robust and perform better using the 2 hidden layer.In the next section we will be expanding our work Leaky ReLU activation function using different weight initialization schemes and regularisation techniques.

Initialisation Schemes: The weights need to be initialised carefully to break the symmetry between hidden hidden units of the same layer. Here we have extended our work with leaky Relu as the activation function with different initialisation scheme - fan-in, fan-out and fan-in-out. The figure 2, compares different weight initilisers, with leaky Relu activation function on a 2 hidden layer network with learning rate as 0.1. The Figure 2 compares these activation function on the given hyper parameters, and shows the training and validation accuracy over the given dataset.
It is clear from the result obtained from the graph that fan-in performs better than other weight initialiser for the

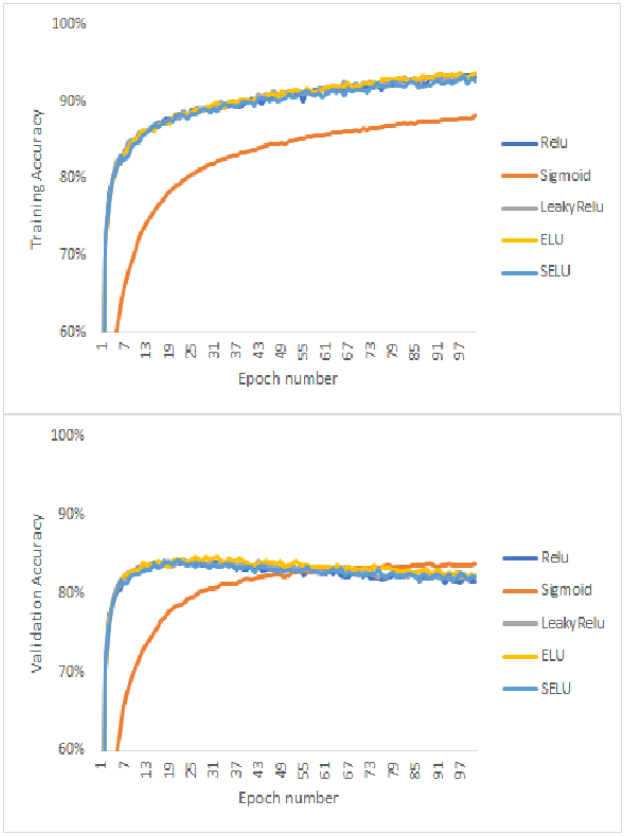

Figure 1. Activation Functions: Training and Validation Accuracy

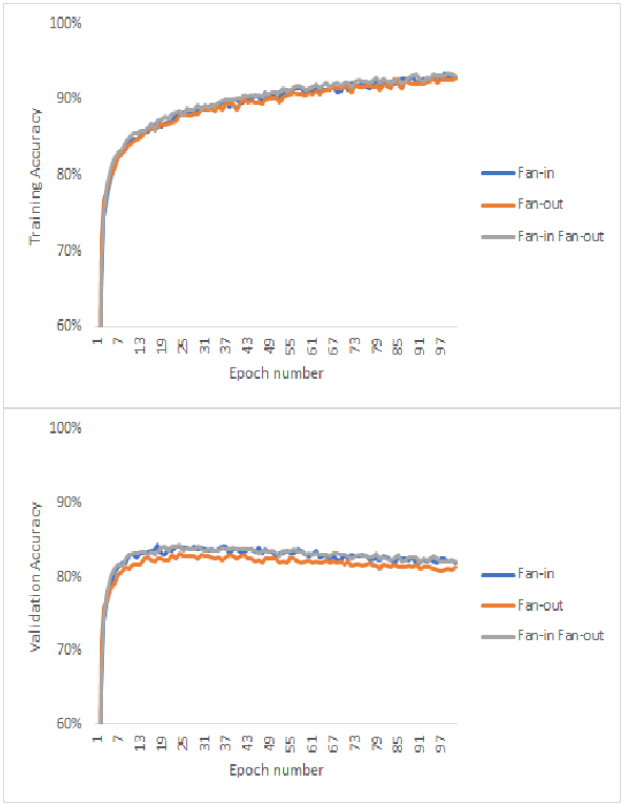

Figure 2. Initialisation Strategy: Training and Validation Accuracy

layer 2 ANN model as fan-in helps in better optimization process and converge to the meaningful solution. So we will be extending leaky-relu and fan-in weight initialiser as our baseline system.

Regularisation Schemes:
Regularization helps avoid overfitting as well as the potential of learning extremely large model parameters. We will be exploring L1, L2 and Dropout regularisors and will be comparing them with our baseline model.
L1 regularisor: L1 regularisor offers some level of sparsity which makes our model more efficient to store and compute and it can also help in checking importance of feature, since the features that are not important can be exactly set to zero.
L2 regularisor: The standard way to avoid overfitting is called L2 regularization. L2 regularisor is a stable solution which gives only one output.
Dropout: Dropout is a widely used regularization technique that is specific to deep learning. It randomly shuts down some neurons in each iteration. The idea behind drop-out is that at each iteration, you train a different model that uses only a subset of your neurons. With dropout, your neurons thus become less sensitive to the activation of one other specific neuron, because that other neuron might be shut down at any time.
We have extended our work with the baseline model using different regularisors. The table 2 and Figure 3, shows the accuracies of L1, L2 and dropout with different hyper parameters.

Regularisor	Validation Accuracy(%)
Dropout Inclusion Probability of 0.3	59.90
Dropout Inclusion Probability of 0.9	85.61
L1 Regularisation Coefficient of 1e-5	82.99
L1 Regularisation Coefficient of 1e-3	82.99
L2 Regularisation Coefficient of 1e-4	82.99
L2 Regularisation Coefficient of 1e-2	82.99
height	

Table 2. Classification accuracies for different regularizors - L1, L2 and Dropout with different coefficient.

As the EMINIST dataset is not complex and already streamlined, the regularization hurts training set performance, compare to the baseline model with Leaky Relu and fan-in weight initialiser. This is because it limits the ability of the network to overfit to the training set.

So following these comparison, we will be using leaky Relu activation function, with fan-in weight initialisor and learning rule as 0.1, without any regularisor as our baseline function. With this baseline function, we will be extending our work to learning rule in next section.

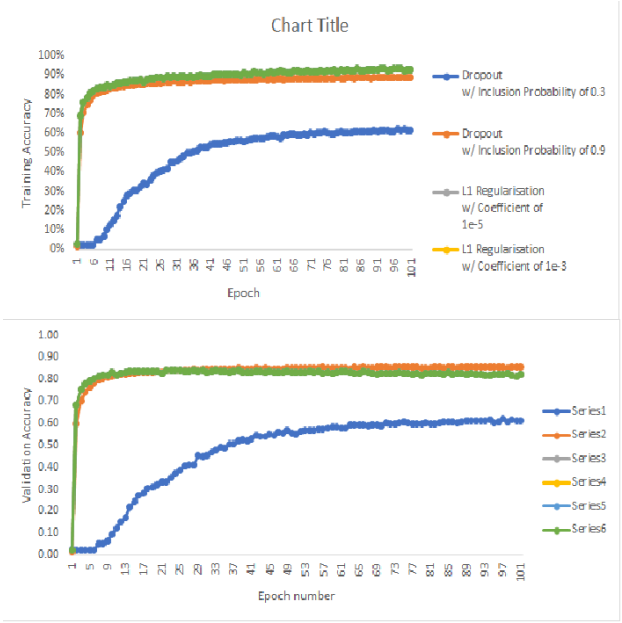

Figure 3. Regularisors- L1, L2, Dropout: Training and Validation Accuracy

3. Learning rules

Learning rule is a method to improve the artificial neural network's performance and usually this rule is performed recursively over the network. It is done by updating the weights and bias levels of a network when a network is simulated in a specific data environment. It can be viewed as solving an optimization problem for a highly non-convex loss function.
Gradient Descent (Lv et al., 2017) is a simple optimization method in machine learning. Gradient-based algorithms are by far the most widely used algorithms for training neural networks, such as basic stochastic gradient descent, Momentum, Root Mean Square Propagation(RMSprop), Adam and others. We will be discussing these and comparing them by running experiments on EMINIST dataset with different hyperparamenters.

- Stochastic Gradient Descent(SGD)
A variant of Gradient Descent is Stochastic Gradient Descent (SGD), which is equivalent to mini-batch gradient descent where each mini-batch has just one example. It computes gradients on just one training example at a time, rather than on the whole training set. When the training set is large, SGD can be faster. But the parameters will "oscillate" toward the minimum rather than converge smoothly.

- Momentum
Momentum helps to accelerate SGD in relevant direction. It dampens oscillations and uses a fraction in the updating. Because mini-batch gradient descent makes a parameter update after seeing just a subset of examples, the direction of the update has some variance, and so the path taken by mini-batch gradient descent will "oscillate" toward convergence. Using momentum can reduce these oscillations. Momentum takes into account the past gradients to smooth out the update.

The momentum update rule is, for $l = 1, ..., L$:

$$\begin{cases} v_{dW^{[l]}} = \beta v_{dW^{[l]}} + (1-\beta) dW^{[l]} \\ W^{[l]} = W^{[l]} - \alpha v_{dW^{[l]}} \end{cases}$$

$$\begin{cases} v_{db^{[l]}} = \beta v_{db^{[l]}} + (1-\beta) db^{[l]} \\ b^{[l]} = b^{[l]} - \alpha v_{db^{[l]}} \end{cases}$$

where L is the number of layers, β is the momentum and α is the learning rate.

- Root Mean Square Propagation (RMSprop)
RMSprop (Tieleman & Hinton, 2012) is an adaptive learning rate method which divides learning rate by an exponential decaying average of squared gradients. The formula is:

$$\begin{aligned} E[g^2]_t &= 0.9E[g^2]_{t-1} + 0.1g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \tag{1}$$

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

- Adam Adam as stated in (Cohen et al., 2017) is one of the most effective optimization algorithms for training neural networks. It combines ideas from RMSProp and Momentum. It computes adaptive learning rates for each parameter. Other than keeping âĂIJpast sum of squared gradientsâĂİ, this also keeps exponentially decaying âĂIJpast gradients".

The update rule is, for $l = 1, ..., L$:

$$\begin{cases} v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1-\beta_1)\frac{\partial \mathcal{J}}{\partial W^{[l]}} \\ v_{dW^{[l]}}^{corrected} = \frac{v_{dW^{[l]}}}{1-(\beta_1)^t} \\ s_{dW^{[l]}} = \beta_2 s_{dW^{[l]}} + (1-\beta_2)(\frac{\partial \mathcal{J}}{\partial W^{[l]}})^2 \\ s_{dW^{[l]}}^{corrected} = \frac{s_{dW^{[l]}}}{1-(\beta_1)^t} \\ W^{[l]} = W^{[l]} - \alpha \frac{v_{dW^{[l]}}^{corrected}}{\sqrt{s_{dW^{[l]}}^{corrected}} + \varepsilon} \end{cases}$$

where: - t counts the number of steps taken of Adam
- L is the number of layers
- β_1 and β_2 are hyperparameters that control the two exponentially weighted averages.
- α is the learning rate
- ε is a very small number to avoid dividing by zero

Here are the experiments performed with SGD, momentum, RMSProp and Adma, on the baseline system of leaky Relu activation function and fan-in weight initialiser with learning rule hyper parameter as 0.1 and 0.01.

SGD performs better compare to momentum, RMSProp and Adams. Momentum usually helps, but given the small

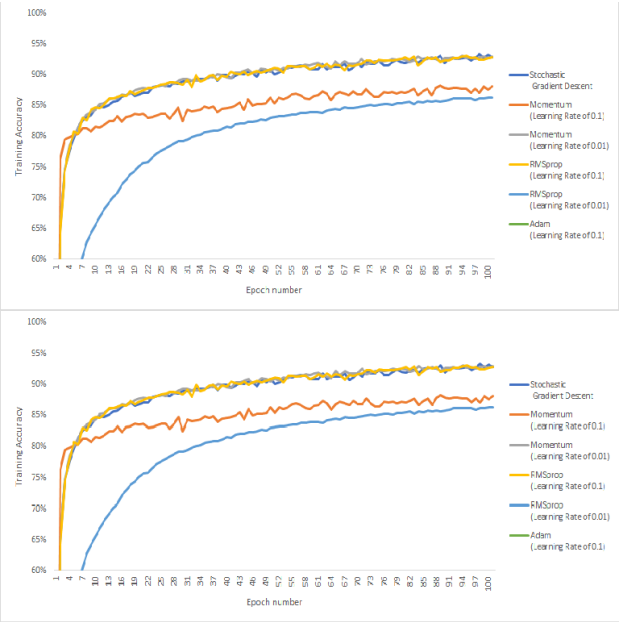

Figure 4. Learning Rules- SGD, Momentum, RmsProp, Adam: Training and Validation Accuracy

Learning Rule	Validation Accuracy(%)
Stochastic Gradient Descent	83.44
Momentum (Learning Rate of 0.1)	81.61
Momentum (Learning Rate of 0.01)	82.99
RMSprop (Learning Rate of 0.1)	82.78
RMSprop (Learning Rate of 0.01)	82.96
Adam (Learning Rate of 0.1)	82.21

height

Table 3. Classification accuracies for different learning rules - SGDs, Momentum, RMSProp, Adam.

learning rate and the simplistic dataset, its impact is almost negligible. Also, the huge oscillations we see in the cost come from the fact that some minibatches are more difficult than others for the optimization algorithm.
Adam on the other hand, generally outperforms mini-batch gradient descent and Momentum. If we run the model for more epochs on this simple dataset, all methods will lead to very good results. However, it has seen that Adam converges a lot faster.
But as EMINIST Balanced dataset is not complex and is already balanced, we see that SGD gives the better results. As stated by (Wilson et al., 2017), the adaptive methods, data and the hyperparameters are the base reasons of Adam not performing, despite Adam algorithm remains incredibly popular.

4. Batch normalisation

Batch Normalisation is the normalisation of the output in each hidden layer. It solves a problem called internal co-variate shift.
Batch normalization potentially helps in two ways - faster learning and higher overall accuracy. The improved method also allows you to use a higher learning rate, potentially providing another boost in speed.

As we know that normalization (shifting inputs to zero-mean and unit variance) is often used as a pre-processing step to make the data comparable across features. As the data flows through a deep network, the weights and parameters adjust those values, sometimes making the data too big or too small again - a problem the refered (Ioffe & Szegedy, 2015) as "internal covariate shift". By normalizing the data in each mini-batch, this problem is largely avoided.

Covariates is just another name for the input feature, often written as X. Co-variate shift means the distribution of the features is different in different parts of the training/test data, breaking the i.i.d assumption used across most of Machine Learning. This problem occurs frequently in medical data (where you have training samples from one age group, but you want to classify something coming from another age group), or finance (due to changing market conditions).

Internal co-variate shift refers to co-variate shift occurring within a neural network, i.e. going from (say) layer 2 to layer 3. This happens because, as the network learns and the weights are updated, the distribution of outputs of a specific layer in the network changes. This forces the higher layers to adapt to that drift, which slows down learning.

Batch normalization helps by making the data flowing between intermediate layers of the network look like whitened data, this means we can use a higher learning rate. Batch normalization has a regularizing effect it also means you can often remove dropout (which is helpful as dropout usually slows down training).
Here is the algorithm showing Batch Normalisation Transform, applied to activation function over mini-batch.

Inputs:
Values of x over a mini-batch: $B = \{x_1....m\}$;
Parameters to be learned: γ, β

Outputs:
$\{y_i = BN_{\gamma,\beta}(x_i)\}$

//mini-batch mean
$\mu_B \longleftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
//mini-batch variance
$\sigma_B^2 \longleftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
//normalise
$\hat{x}_i \longleftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
//scale and shift
$y_i \longleftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

where x_i represents the input feature and the output value in each hidden layer before activations. Like the figure shown below, we use the Batch Norm for the input feature X as well as the output value in each hidden layer. Noticeably, Batch Normalization also used the exponentially weighted averages and bias correction. The γ and β are two hyperparameters, which can be learned by this neural network. The use of γ and β is to keep the mean and variance as 0 and 1 or any fix mean and variance.

In Figure 5, we are comparing Batch Normalisation with our baseline system using learning rate 0.1, 0.2 and dropout regularisor. We can see that regularisor does not perform well here, as the dataset is streamlined and we are not using deep layer networks.

In Figure 6, we are extending our baseline system on deep networks of layer 2,3 and 4 using batch normalisation. We can see that layer 3 network is giving the best result and it is converging. Since the dataset is not complex, so there is no need to make the model deep and use the minimal best layer network, as it will be robust and fast.

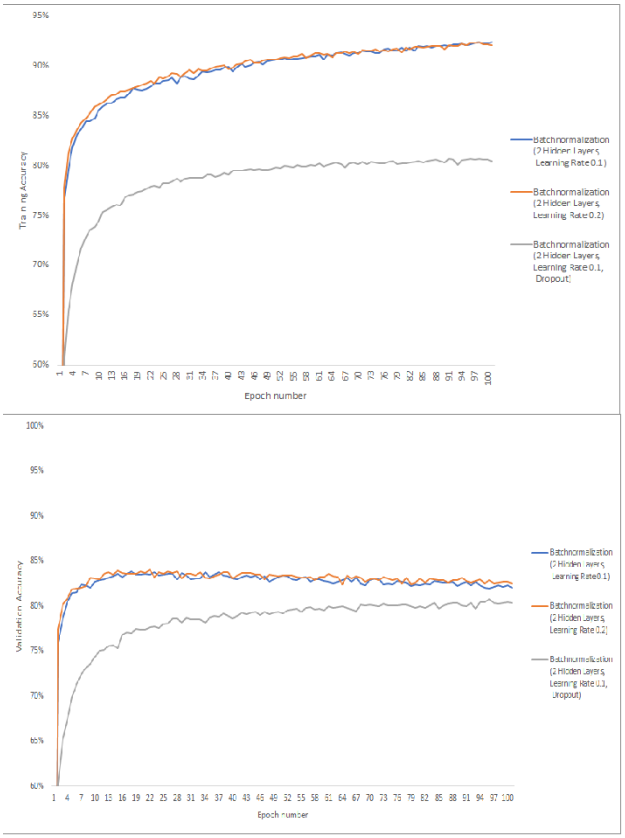

Figure 5. Batch Normalisation- Learning Rate 0.1, 0.2 and dropout: Training and Validation Accuracy

5. Convolutional networks

Convolutional Neural Networks(CNNs), like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. CNNSs operate over volume. Unlike neural networks, where the input is a vector, here the input is a multi-channeled image

Convolutional Layer
In the Convulation layer, we will take many filters and convolve them on the input. Each convolution gives a 2D matrix output. You will then stack these outputs to get a 3D volume.
The formulas relating the output shape of the convolution

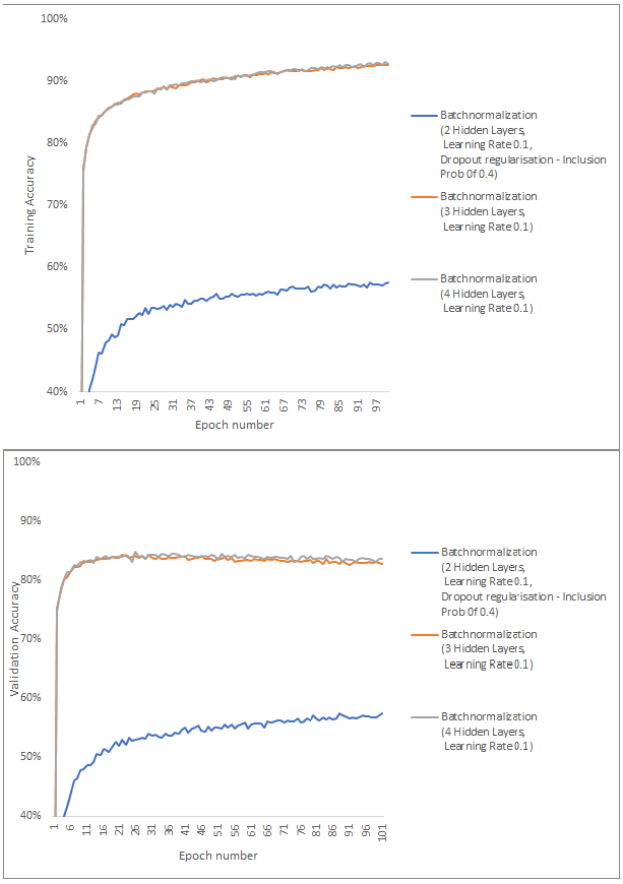

Figure 6. Batch Normalisation- Deep Networks layer 2, 3 and 4: Training and Validation Accuracy

to the input shape is:

$$n_H = \lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

$$n_W = \lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

n_C = number of filters used in the convolution

Pooling Layer

A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling.

The formulas binding the output shape of the pooling to the input shape is:

$$n_H = \lfloor \frac{n_{H_{prev}} - f}{stride} \rfloor + 1$$

$$n_W = \lfloor \frac{n_{W_{prev}} - f}{stride} \rfloor + 1$$

$$n_C = n_{C_{prev}}$$

The convolution neural network experimented for two different models: first one with one convolutional layers, and max-pooling layers, using the EMNIST Balanced data, using convolutional kernels of dimension 5×5 (stride 1) and pooling regions of 2×2 (stride 2, hence non-overlapping). This was trained for 20 epocs, as it started converging.

The second one with two convolutional layers, and maxpooling layers with 5 feature maps in the first convolutional layer, and 10 feature maps in the second convolutional layer, using the EMNIST Balanced data, using convolutional kernels of dimension 5×5 (stride 1) and pooling regions of 2×2 (stride 2, hence non-overlapping). This was trained for 10 epocs and was near to converging.

In the figure 7, accuracies are compared for training and validation set for one and two layer convolution neural network respectively. The validation accuracy with two layer covnet is 84 %, while with one layer covnet is 82 %. Two layer network is giving a better result for less number of epocs.

Mixing Dropout and Batch-normalisation may result in a better covnet. These combination are not tried now and need to be tested in future experiments.

6. Test results

The baseline system of leaky Relu activation function and fan-in weight initialser was tested with the EMINIST Balanced test dataset. It resulted with 81% test accuracy.

We extended this with the two layer convulation neural

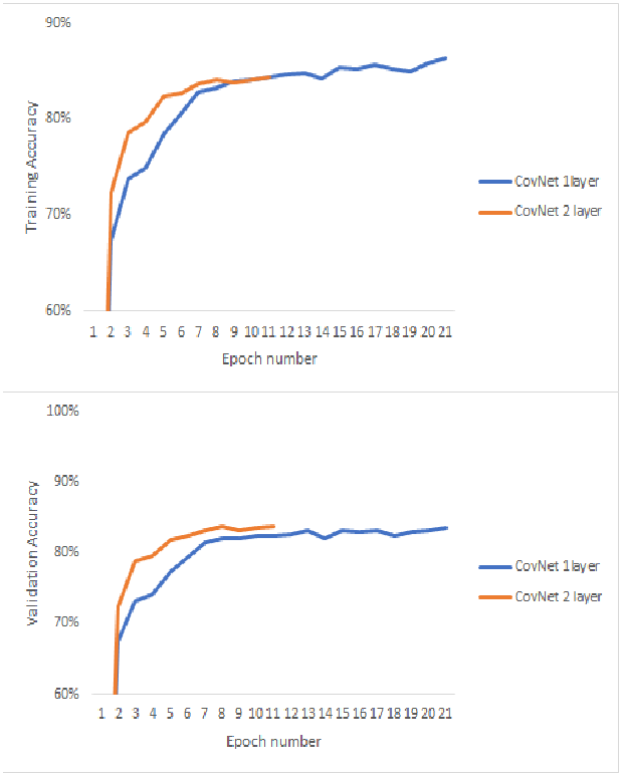

Figure 7. Convulation Neural Network- Networks layer 1 with 20 epocs and layer 2 with 10 epocs: Training and Validation Accuracy

network as used the test data, which resulted in 83% test accuracy.

The convolution neural network was trained using 10 epocs while the baseline system with 100 epocs. So training the convolution neural network to equal baseline may result in increasing the test accuracy.

7. Conclusions

We have discussed about the baseline system which we developed after comparing various activation function, weight initialiser and regularisation techniques in section 2. We finalised leaky Relu and fan-in weight initialiser without any regularisor as our baseline system, having validation acuracy of 84 %.

We then extended our work to optimisation technique, using SGD, Momentum, RMSProp and Adams in Section 3. We further worked towards normalisation techniques using batch normalisation in section 4.

Convolution Neural Networks were explored with different layer in section 5. We got the validation accuracy of around 84.5 % with two layer convolution network, with runnig 10 epocs.

The Convolution Neural Network performed better than the Baseline system, while the convolution Neural Network was only trained for 10 epocs and baseline system was trained for 100 epocs.

In the future experiments, we can extend our experiments with the multi-layer convolution network with Dropout optimiser and Batch Normalisation technique. This paper (Krizhevsky et al., 2017), relates with this and can be a source for extending experiments to different model and hyperparameters combinations using Covnets.

References

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.

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

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL http://doi.acm.org/10.1145/3065386.

Lv, Kaifeng, Jiang, Shunhua, and Li, Jian. Learning gradient descent: Better generalization and longer horizons. *CoRR*, abs/1703.03633, 2017. URL http://arxiv.org/abs/1703.03633.

Tieleman, T. and Hinton, G. E. Coursera: Neural networks for machine learning. 4(2), 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *ArXiv e-prints*, May 2017.