MLP Coursework 2: Learning rules, BatchNorm, and ConvNets

s1448320

Abstract

In such rapidly developing areas as neural networks and image classification, even a slight gain in performance seems a step forward towards the magic accuracy of 100%. This experiment is a continuation of our research regarding neural networks. This time we will consider Balanced MNIST dataset, which along with digits, also includes hand-written letters making the classification more complex. We sought additional improvement in performance by introducing our own implementation of RMSProp and Adam learning rules as well as batch normalisation. These techniques, along with other methods such as regularisation and dropout allowed us to construct our final architecture of a deep neural network (DNN). This model was then compared with our implementation of a completely new type of network, namely convolutional neural network (CNN) designed especially to take images as their inputs for which we also investigated influence of depth (number of convolutional layers) on its performance. Indeed, despite a long time to complete the training, our experiments confirmed recent studies showing CNNs superiority over DNNs when applied to image classification tasks such as this one (Simard et al., 2003).

1. Introduction

Balanced EMNIST dataset is one of the most widely applicable datasets among all different variations of NIST database. This is because, it is intended to provide the most fair and consistent classification task derived from the NIST Database since it provides more classes than MNIST but prevents the classifiers from confusion between very similar upper and lower-case letters (Cohen et al., 2017).

This dataset originally consists of 115,800 training and 15,800 testing images of hand-written digits and letters collected from American Census Bureau employees and American high-school students. Compared to MNIST which had been used in our previous experiment, it also contains 26 upper and lower-case letters. This was simplified further to avoid classification errors resulting from misclassification between uppercase and lowercase for some letters such as ’O’ or ’C’.

Each image in the dataset is 28 pixels high and 28 pixels wide. This means that a single image can be represented by 28 by 28 matrix, containing 784 entries (each entry represents a single pixel). This number also corresponds to the input dimension of the investigated neural networks. The same applies to the output dimension which corresponds to 47 possible classes (0-9 digits, 26 lower case and 26 upper-case letters with C, I, J, K, L, M, O, P, S, U, V, W, X, Y and Z being considered as case-insensitive).

For the purpose of this experiment, training set has been divided further into training and validation sets containing 100,000 and 15,800 images respectively. This split is vital since it will allow us to monitor the network’s performance and check if the data has been overfitted using a separate set of images which had not been seen while training the network (validation set).

Accuracy, as well as the error rate, on the validation set is monitored during the training process and those measures will be used to assess the performance our networks to choose the most optimal hyperparameters.

In this experiment, we will first construct a baseline deep neural network considering various architectures (section 2). These involve different network depths (number of hidden layers), values of hyperparameters (learning rate, hidden units) as well as using various optimisation techniques such as regularisation (L1, L2) and dropout which extend number of the hyperparameters even further. All these different options will allow us to choose the best architecture for our baseline network to compare it with other models developed in the later stages of the experiment.

As a next step, we will investigate our implementation of different learning rules (RMSProp and Adam). To achieve this, we will focus on the theory behind each of the rules and their influence on training with respect to test/validation accuracy (section 3). Performance of each of the investigated rules will then be compared with the baseline system of the same architecture using Stochastic Gradient Descent (SGD).

Furthermore, we will determine the impact of normalising each batch by introducing our implementation of batch normalisation (section 4).

All the above procedures will allow us to determine the best architecture for our deep neural network which will then be assessed using testing set (i.e. a set that has never been seen before by the network).

The next part of this experiment will introduce us to convolutional neural networks which gained a huge popularity in common image classification tasks like ours, often outperforming other kinds of deep neural networks (Krizhevsky et al., 2012). This is mainly due to their design, considering regions of images for different patterns (i.e. features). In this section (section 5) we will design, train an asses the networks consisting of up to two convolutional and max-pooling layers.

As a final step of this experiment (section 6), we will assess the performance the most optimal architecture for the convolutional neural network using test set. This will allow us to compare our convolutional neural network with the deep neural network designed previously. Since the main focus of this experiment is to assess and understand the designs of these two kinds of networks, we will try to carefully explain all the reasons behind the differences in their performance.

The final section of this paper are conclusions regarding all the findings drawn throughout the experiment (section 7).

2. Baseline systems

The first aspect of the baseline architecture to consider was its depth. As indicated in most recent papers and our previous experiments, the depth of the most popular networks vary significantly with AlexNet (Krizhevsky et al., 2012) of 7 layers to over 1000 for ResNet (He et al., 2015) with deeper networks often outperforming shallower architectures. More layers (i.e. nurons and connections between them) result in more parameters to fit the data better. Deeper networks are computationally more expensive, however this constrain has been recently reduced due to introduc-

Learning rate	Max accuracy	Final accuracy
0.005	8.358×10^{-1}	8.358×10^{-1}
0.010	8.417×10^{-1}	8.373×10^{-1}
0.050	**8.434**$\times10^{-1}$	**8.194**$\times10^{-1}$
0.100	8.413×10^{-1}	8.250×10^{-1}
0.200	8.372×10^{-1}	8.165×10^{-1}
0.400	8.294×10^{-1}	8.244×10^{-1}

Table 1. Maximum and final accuracy after 100 epochs of the baseline model on the validation set for different values of learning rate.

L2 penalty	Max accuracy	Final accuracy
None	8.434×10^{-1}	8.194×10^{-1}
1.0×10^{-6}	8.439×10^{-1}	8.211×10^{-1}
1.0×10^{-5}	8.430×10^{-1}	8.240×10^{-1}
1.0×10^{-4}	8.475×10^{-1}	8.253×10^{-1}
1.0$\times10^{-3}$	**8.551**$\times10^{-1}$	**8.522**$\times10^{-1}$
1.0×10^{-2}	7.817×10^{-1}	7.571×10^{-1}

Table 2. Maximum and final accuracy after 100 epochs of the baseline model on the validation set for different values of L2 penalty parameter.

Inclusion probability	Max accuracy	Final accuracy
0.5	5.296×10^{-1}	5.278×10^{-1}
0.7	7.726×10^{-1}	7.620×10^{-1}
0.9	8.458×10^{-1}	8.449×10^{-1}
None (1.0)	**8.551**$\times10^{-1}$	**8.522**$\times10^{-1}$

Table 3. Maximum and final accuracy after 100 epochs of the baseline model on the validation set for different values of dropout inclusion parameter combined with L2 regularisation.

ing more advanced hardware (GPUs) and more efficient library implementations of the most used functions (CUDA) (Jia et al., 2014).

After investigating different network depths (3-8), we have decided to use 4 hidden layers with 100 connections per layer. This set-up recorded one of the highest accuracies on the validation set of 8.410×10^{-1}, not suffering from overfitting as much as the other deeper models.

We also considered increasing the number of units per hidden layer since our task has been extended to letter classification. However, having more connections between layers would significantly lengthen the learning process. Hence, due to our limited resources and time constrain, we decided to keep the number of connections between hidden layers as before (100 per hidden layer).

Another hyperparameter to consider was learning rate which can be considered as a 'step size' over the loss function which controls the speed in which network is learning (finding the minimum of the loss function). In all of the investigated values for learning rate, the best performance in case of the maximum accuracy on the validation set has been recorded for the value of 0.05 (Table 1.) and this value was decided as optimal.

As a next step, we tried to improve network's performance even further and reduce overfitting. One way of limiting overfitting would be to to decrease the flexibility of the model by limiting its depth which would reduce the number of free parameter. However, this would also affect the maximum accuracy achieved by our baseline network. That is why we considered to introduce regularisation. Regularisation can be viewed as a way of compromising between finding small weights and minimising the original cost function (Girosi et al., 1995). It reduces overfitting by adding a complexity penalty to the loss function.

We have considered two types of regularisation, namely L1 and L2. L1 regularisation makes units to use only a sparse subset of their most important inputs, making them less affected by any 'noisy' inputs. L2 on the other hand, makes the network to use all of its inputs a little rather that some of its inputs a lot which seems to be much more efficient approach than L1 according to many recent studies (Ng, 2004). This has also been confirmed by our initial tests (data not included). Therefore, we have decided to further investigate our model using L2 regularisation.

Regularisation introduces a new hyperparameter (β) which indicates scaling penalty term. Our tests recorded the best performance (increase of 1.39% in accuracy) for L2 parameter of 1.0×10^{-3} (Table 2.). Regularisation with 1.0×10^{-3} yield lower accuracy on the training set than smaller values for this parameter (8.882×10^{-1} compare with the average of 9.250×10^{-1} for smaller values). However, due to reduced overfitting, the accuracy recorded on the validation set was higher than for any other smaller values (Table 2.). Therefore, the value of 1.0×10^{-3} for L2 penalty was decided to be used in the next step of our baseline model optimisation. The last aspect considered while constructing our baseline model was dropout. Dropout modifies the network itself instead of modifying the cost function as L1 and L2 regularisation. It keeps only some part of the hidden units in each minibatch as active (determined by the parameter p indicating inclusion probability for each unit with 1 meaning that every unit is included) with the rest being set to zero. After updating the weights, different part of hidden layers is chosen for the next minibatch and this continues. This procedure makes the network behave as an average of multiple networks (with smaller number of hidden units each).

According to (Srivastava et al., 2014) dropout seems to yield better performance on the MNIST classification task when combined with L2 (classification error of 1.25 when combined compared with 1.62 for L2) and this was the main motivation to introduce this regularisation technique. However, according to our experiments (Table 3.), introduction of the dropout layer did not improve the performance yielding drop of almost 1% on the validation accuracy for parameter of 0.9.

This lack of improvement might be caused by the amount of noise in the gradients introduced by dropout compared with stochastic gradient descent which makes gradients to cancel each other. As (Srivastava et al., 2014) suggests, increasing learning rate by the factor of 10-100 (compared with the standard value) would make this less significant. The study also suggests to introduce dropout of different scale for input (higher probability) and hidden layers (lower probability). Indeed, increasing the learning rate by the factor of 20 and introducing dropout of 0.9 for the input and 0.7 for hidden layers improved the accuracy on the validation set up to 8.492×10^{-1}. However, this level of accuracy is still much lower than that of the model using just L2 regularisation, therefore we decided not to use dropout for our baseline model. Therefore, our final version of baseline model consists of 4 hidden layers with 100 connections between them, L2 regularisation value of 1.0×10^{-3}, learning rate of 0.05 and stochastic gradient descent learning rule. We also decided to leave the size of each minibatch unchanged since (value of 100) this setting yield satisfactory results and made the learning process take acceptable amount of time per epoch (5 seconds).

We also decided to use ReLu activation function, since this source of introducing non-linearity to the network seems to be the most popular choice in the recent literature (Krizhevsky et al., 2012).

3. Learning rules

Up to this point, our baseline model had been using stochastic gradient descent (SGD) with a fixed learning rate. In this section we will present and discuss the results of the experiments comparing networks performance using new learning rules on the Balanced MNIST task. These are RMSProp and Adam, which change the value of learning rate over time using different paradigms (per-parameter change as opposed to global change as for SGD).

3.0.1. Stochastic Gradient Descent

Stochastic gradient descent was the only learning rule investigated so far in our baseline model. The highest accuracy recorded for that model was 8.551 $\times 10^{-1}$.

$$\Delta w_i = -\eta g_i(t) \tag{1}$$

As we can see from its equation (Equation 1.), SGD updates the parameter vector by simply multiplying the gradient by the negative value of learning rate (η). The negative sign is used because we are trying to minimise the loss function. This results in non-negative progress on the loss function when evaluated on the full dataset for a low enough learning rate.

It is worth noting that the learning rate is kept constant for the whole training process. However, after some number of epochs it might turn up (Coates et al., 2011) that the steps taken along the loss function are too small (or big) resulting in inefficient learning (i.e. parameter vector bounces around chaotically or gets stack in a local minimum for too big and too small value of learning rate respectively).

3.0.2. RMSProp

RMSProp introduces an interesting concept of per-parameter adaptive learning rate. However, before discussing this learning rule in more detail, we need to introduce AdaGrad which is a predecessor of RMSProp.

$$\begin{aligned} S_i(0) &= 0 \\ S_i(t) &= S_i(t-1) + g_i(t)^2 \\ \Delta w_i(t) &= \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} g_i(t) \end{aligned} \tag{2}$$

From AdaGrad's equation (Equation 2.), we can see that the update step for parameter w_i is normalised by the square root of the sum squared gradients for that parameter. This results in larger updates for less frequent and smaller updates for more frequent parameters, which effects in separate, normalised update for each weight. Studies suggest that Adagrad greatly improves the robustness of SGD (Coates et al., 2011) due to the learning rate adjustment.

Adagrad has the natural effect of decreasing the learning rate. However, this decrease might be too drastic for some effective learning rates which can negatively affect the learning process (Duchi et al., 2011).

$$\begin{aligned} S_i(t) &= \beta S_i(t-1) + (1-\beta) g_i(t)^2 \\ \Delta w_i(t) &= \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} g_i(t) \end{aligned} \tag{3}$$

RMSProp is one of the attempts to resolve Adagrad's radically diminishing learning rates by replacing the sum by a moving average for S (Equation 3.). It keeps running average of its recent gradient magnitudes and divides the next gradient by this average so that loosely gradient values are normalised making learning to work much better (Ruder, 2016).

Learning rule (η)	Max accuracy	Final accuracy
SGD (0.05)	**8.551**$\times 10^{-1}$	**8.522**$\times 10^{-1}$
RMSProp (0.05)	2.405$\times 10^{-2}$	2.038$\times 10^{-2}$
RMSProp (0.001)	**8.585**$\times 10^{-1}$	**8.461**$\times 10^{-1}$
RMSProp (0.0001)	8.444$\times 10^{-1}$	8.420$\times 10^{-1}$
Adam (0.05)	2.405$\times 10^{-1}$	1.962$\times 10^{-1}$
Adam (0.001)	**8.590**$\times 10^{-1}$	**8.576**$\times 10^{-1}$
Adam (0.0001)	8.454$\times 10^{-1}$	8.453$\times 10^{-1}$

Table 4. Maximum and final accuracy after 100 epochs of the baseline model with SGD, RMSProp and Adam on the validation set for different values of learning rate.

RMSProp introduces new parameter β indicating learning rate decay which has been estimated to be the most optimal for the value of 0.9 (Ruder, 2016). RMSProp makes learning rate to change for each parameter, after every epoch. This proved our baseline learning rate to be very inefficient achieving very low maximum accuracy on the validation set of 2.405$\times 10^{-2}$ due to the parameter vector chaotically bouncing along the loss function. This is why we decided to investigate smaller learning rates.

Lowering learning rate by a factor of 50 (to 0.001) yield significantly higher accuracy on the validation set. This time we are not overshooting and instead start with taking smaller steps along the loss function in order to adjust the magnitude of the step as the training progresses.This resulted in slightly higher maximum accuracy on the validation set. However, the difference is not significant and it might suggest to try slightly higher values than 0.001 of this hyperparameter.

Moreover, the process of learning seems slightly more unstable for RMSProp compared with SGD which was indicated by the accuracy on the validation set varying much more compared with the one for SGD. This might be due to RMSProp constantly adjusting learning rate, trying to move along the loss function slightly more rather than being stack around local minimum like SGD.

3.0.3. Adam

The last investigated learning rule which computes adaptive learning rates for each parameter is Adam. Like RMSProp, it keeps exponentially decaying average of past squared gradients S_i, additionally storing exponentially decaying average of past gradients M_i, having similar effect as momentum (Equation 4.). This is why it is often refereed to as 'RMSProp with momentum'.

$$\begin{aligned} M_i(t) &= \alpha M_i(t-1) + (1-\alpha) g_i(t) \\ S_i(t) &= \beta S_i(t-1) + (1-\beta) g_i(t)^2 \\ \Delta w_i(t) &= \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} M_i(t) \end{aligned} \tag{4}$$

Newly introduced parameters α and β control the decay rates of the moving averages with values estimated to 0.9 and 0.999 respectively.

As suggested by various studies (Ruder, 2016), Adam tends to outperform previously investigated learning rules due to its robustness to noise and adding the momentum parameter which results in more kinetic energy (analogous to physics) allowing to explore more optimal regions of the loss function. Indeed, Adam's advantage has been confirmed by our research results. Using Adam increased the accuracy on the validation set up to 8.590$\times 10^{-1}$.

Similarly to RMSProp, the learning rate needed to be significantly decreased for Adam to yield anywhere close to comparable accuracy with our baseline model (SGD).

Despite the not significant increase in performance, our results seem to confirm that introducing the momentum improves learning. However, due to the time constrains we were not able to perform any further search for a more optimal learning rate.

Moreover, recorded results seem to confirm recent studies (Ruder, 2016) suggesting that Adam tends to slightly outperform RMSProp towards the end of optimization as gradients become sparser making Adam one of the default methods for deep learning optimisation (Karpathy, 2015).

Summing up, after detailed investigation of RMSProp and Adam, both of the rules tend to yield better results than SGD with Adam recording the highest maximum accuracy on the validation set which made us include this learning rule to improve on our baseline model and achieve the accuracy on the validation set to up to 8.590×10^{-1}.

4. Batch normalisation

In this section we will introduce the concept of batch normalisation and test our implementation of `BatchNormalizationLayer` in our previous models for SGD, RMSProp and Adam.

Batch normalisation is a very popular technique helping with network initialisation by normalising activations of each layer which has been recently introduced (2015) by Ioffe and Szegedy from Google Research Lab (Ioffe & Szegedy, 2015).

As the network gets deeper, the learning rate needs to be decreased and the learning process takes longer time to complete. We experienced that in our previous experiment (coureswork 1). This is because as the parameters of the previous layers change, the distribution of each layer's inputs also has to change (Ioffe & Szegedy, 2015). This makes the choice of the network parameters more complicated and very dependent of its architecture.

Ioffe and Szegedy came with a solution to ease the initialisation process by explicitly forcing the activations across the whole network to take a unit Gaussian distribution throughout the training process.

As we can see from its equation (Equation 5.), batch normalisation introduces two new parameters of γ and β which are responsible for scaling and shifting normalised activations respectively. μ and σ^2 are simply mean and variance of each hidden unit activation across the minibatch.

$$\begin{aligned} u_i &= \mathbf{w}_i\mathbf{x} \\ \hat{u}_i &= \frac{u_i - \mu_i}{\sqrt{\sigma^2 + \epsilon}} \\ z_i = \gamma_i\hat{u}_i + \beta_i &= BatchNorm(u_i) \end{aligned} \tag{5}$$

Batch normalisation is often referred to as pre-processing step since we shift inputs to zero-mean and unit variance for each minibatch (Equation 5.) which makes the data comparable across features. An important thing to notice is that we are doing it for each layer and for each minibatch, not just at the beginning of the training which makes the data to 'adjust' after each layer.

According to Ioffe and Szegedy (Ioffe & Szegedy, 2015), batch normalisation allows us to use higher learning rates which makes learning much quicker and yielding better accuracy on some of the most popular datasets for classification tasks such as ImageNet (4.9% top-5 validation error). It can also be applied as a regulariser and eliminate a need for dropout.

Moreover, due to the normalisation, the network is more robust to gradients for outlier activations by reducing them which otherwise might have affected learned activations. This is why batch normalisation allows us to use higher values of learning rates. As recommended, we have applied `BatchNormalizationLayer` immediately after each fully connected layer. However, our experiments did not support the claims stated in the research (Ioffe & Szegedy, 2015). We have experienced a decrease for each of the networks' performance with the biggest drop recorded for Adam (8.284×10^{-1} compared with 8.590×10^{-1} when no batch normalisation has been applied).

Learning rule (η)	Max accuracy	Final accuracy
SGD (0.05)	8.363×10^{-1}	8.327×10^{-1}
RMSProp (0.001)	8.342×10^{-1}	8.342×10^{-1}
Adam (0.001)	8.284×10^{-1}	8.244×10^{-1}

Table 5. Maximum and final accuracy after 100 epochs of the baseline model with SGD, RMSProp and Adam on the validation set after applying batch normalisation.

This lack of improvement might be due to a very low depth of our networks' architecture (only 4 hidden layers). We were not able to assess this claim due to the limited computational resources. However, batch normalisation is usually considered in much deeper architectures such as Inception-ResNet (Szegedy et al., 2016) where it yields significant improvement. This might be an interesting aspect to consider in our future research.

After careful investigation of the batch normalisation and its disappointing lack of improvement our network performance, we decided not to include it in our final model for deep neural network.

5. Convolutional networks

In this section, we are going to introduce concept of a new kind of neural network, namely convolutional neural network which gained a huge popularity in image classification tasks. This type of network was able to decrease the error rate down to 0.4% on the MNIST classification task compared with 1.6% for an ordinary neural network of the same depth (Simard et al., 2003) . We will also describe its implementation and present the results of our experiments.

Our main goal is to investigate how accuracy of convolutional neural network (CNN) improves with its depth (i.e. number of convolutional and maxpooling layers) and how it compares with deep neural network (DNN) architectures introduced in the previous sections of this experiment.

5.0.1. Background

CNNs, like standard neural networks, consist of neurons that have learnable weights and biases with all other details being the same as before. However, CNNs give us advantage of considering spacial structure of the input by assuming that it is an image. Previously, each image was treated as a vector with adjacent pixels not being very related to each other. CNNs allow us to view images as 2D structures as they are to look for patterns (features) in the image by looking at patches of pixels at a time.

The first step in CNN architecture involves so called convolutional layer. This step is used to extract features from the input image. These are achieved due to convolution operation which preserves the spatial relationship between pixels by learning image features using small squares of input data (kernels). For the purpose of this assignment, we decided to choose 5×5 kernel with stride 1 which seems to be the most popular choice in recent literature (He et al., 2016) along with 3×3 variations (Simonyan & Zisserman, 2014b). Kernels are also known as 'feature detectors' due to the way they allow us to detect patterns in the image. They simply slide over the image (size of the step determined by stride) taking the dot product and sum over each 'chunk' of an image to produce a feature map.

Each entry in a feature map is able to extract the same feature by sharing weights across the kernels. Moreover, different types of kernels detect different features from an image (i.e. curves or edges) resulting in different feature maps. Specific kernels are learned during the training process with their number being specified beforehand. Our default architecture involves 5 feature maps for a single layer model and 5 and 10 feature maps for the first and second layer respectively in our 2 layers model.

It is worth to mention that the more features are extracted from an image (i.e. number of feature maps), the better our network becomes at recognising patterns in unseen images. However, more feature maps increases the amount of computation required to complete the training.

Since convolution is a linear operation (element-wise matrix multiplication and addition) and most real-world task involve non-linear problems, an activation function needs to be introduced. The details of different activation functions has been discussed extensively in our previous research. We decided to use ReLu unit which seems to be a very popular choice according to the most recent studies (Srivastava & Masci, 2013) and yielded good results in our previous experiments.

The next step in CNN training is dimensionality reduction using pooling layer. In this step, we are trying to reduce the dimensionality of each feature map with a purpose to keep the most important information about each feature. Without this reduction, we would have to deal with rapidly increasing number of inputs to each layer resulting in increased time needed to complete the classification process.

For the sake of this experiment we decided to implement Max-Pooling layer which takes the maximum entry within each pooling region of size 2×2 and stride 2 which results in no overlap between the considered regions. This operation greatly decreases the dimensionality of each of the feature maps (i.e. halving both dimensions in this case), preserving all the relevant information at the same time. Moreover, pooling operation reduces the number of parameters and computations in the network, which greatly helps in controlling the overfitting.

Despite other popular operations such as averaging or summing, taking the maximum of each pooling region seems to yield the best results (Nagi et al., 2011).

This whole operation allows us to achieve scale equivariant representation of the original image, enabling the network to detect/classify object no matter the location. This makes CNN so powerful and often superior compared to ordinary DNNs (Simonyan & Zisserman, 2014a).

Additional advantage of pooling is that it makes the network more invariant to small variations in the inputted images (i.e. transformations or distortions). This is because a small change in the input image will not affect the output of the pooling layer greatly since a pooling region is considered as opposed to each individual pixel.

It is important to notice that we can have arbitrary many convolutional layers followed by ReLu before applying pooling layer. This also applies to pooling layers when the mentioned sequence is used multiple times after each other. This makes the network grow in depth an allow it to detect more complicated features of an image. Taking example of face detection, first layer might be used for edge detection, next one might extract simple shapes with the third one detecting more advanced features such as facial characteristics (Lee et al., 2009).

Due to lack of computational resources, the deepest architecture of CNN used in this experiment consisted of only 2 convolutional/maxpooling layers. In fact the most advanced and accurate CNNs tend to rapidly extend their depth due to more advanced hardware (GPUs) and implementation techniques (faster libraries, parallel computing). The best example is Deep Residual Network (ResNet) being up to 1202 layers deep and achieving 7.93% error rate on CIFAR-10 image recognition dataset (He et al., 2016).

The final stage in CNN architecture is introduction of fully-connected layer (`AffineLayer`) known from the previously investigated neural networks. The output from convolutional/pooling layers blocks consists of multiple feature maps of an input image. Fully-connected layer combines those features allowing final image classification.

Including multiple fully-connected layers is also a common technique used in recent implementations with LeNet-5 used for MNIST task being a good example (Bengio & Lecun, 1997). Its architecture consisted of two fully connected layers at the end.

5.0.2. Implementation

Our initial implementation of `ConvolutionalLayer` involved a naive approach using multiple for-loops (up to 6 in `bprop` method) and processing each entry in the local receptive field at a time. This proved to be extremely inefficient due to a long time to complete a single epoch (up to an hour).

This is due to huge number of convolution operations performed at each pass. As opposed to a standard `AffineLayer`, kernel needs to scan an image at multiple positions during each pass. This makes the whole process more computationally expensive.

This made us realise how crucial efficient implementation is for training more advanced models such as CNNs. Therefore, we decided to take advantage of available Python libraries and array broadcasting.

Our more efficient version involved `scipy.signal` function `convolve` to perform convolution in the forward propagation stage. We used cross-correlation instead of convolution which involves 'flipping' the kernel due to consistency with our original version of the implementation. However, those two methods (cross-correlation and convolution) are equivalent in this case.

We also considered using `fftconvolve` function which uses fast Fourier transform method, therefore being much more efficient for bigger matrices (up to 100 times faster for $N > 500$). However, we noticed it to be less accurate with respect to floating point arithmetics, and having troubles with passing the tests provided without converting the results to integers. Therefore we decided to keep using standard `convolve` function for cross-correlation to pass the tests and use `fftconvolve` to run our experiments since the difference in accuracy is negligible.

Due to the time constraints, we did not manage to apply the same kind of improvement for back-propagation. However, we managed to drastically reduce the number of for-loops using broadcasting in Python. Broadcasting provides a means of vectorising array operations so that looping occurs in C instead of Python which makes the implementation more efficient. We applied exact same approach in our implementation of `MaxPoolingLayer`.

As mentioned before, there are many techniques available to speed up the calculations and make specific implementations more efficient. Those include more efficient library implementation such as previously mentioned `fftconvolve` or different optimising compilers such as Numba which uses the LLVM compiler infrastructure to compile Python to machine code. This results in some functions running up to 200 times faster than standard Python implementation. It also supports compilation of Python to run on

Model (learning rate)	Max accuracy	Final accuracy
1-Layer CNN (0.001)	**8.413**$\times 10^{-1}$	**8.335**$\times 10^{-1}$
1-Layer CNN (0.01)	7.952×10^{-1}	7.911×10^{-1}
2-Layers CNN (0.001)	**8.600**$\times 10^{-1}$	**8.549**$\times 10^{-1}$
2-Layers CNN (0.01)	8.438×10^{-1}	8.300×10^{-1}

Table 6. Maximum and final accuracy after 50 epochs of CNN model of different depths on the validation set for different learning rates.

GPU hardware which recently s widely used for deep learning.

Other alternative include widely used TenserFlow introduced by Google (TenserFlow) or less known GPU Coder in MATLAB (available from R2017b version) which generates optimised CUDA code from MATLAB code (MATLAB).

It is also common to take advantage of parallel computing and train a network on a computation cluster. Popular solutions include AWS with each node containing up to 16 GPUs (AWS, 2017).

5.0.3. Results

As our first investigated CNN consisted of a reshape layer used to make the inputs take appropriate 4D shape (batch size, number of channels, height and width of each image). Next, a single convolutional layer with 5×5 kernel and stride of 1 and 5 feature maps was added, followed by ReLu activation function and maxpooling layer with non-overlapping pooling regions of 2×2 (stride 2). Finally, another reshape layer was used to flatten the dimensions of the outputs from the pooling step to be able to pass it the final fully-connected layer to allow the classification.

In order to investigate the effect of depth on CNN's accuracy, we also introduced a deeper architecture consisting of two sets of convolutional and maxpooling layers. The overall structure and the dimensions of kernels and pooling regions were kept exactly the same as in the shallower implementation in order to make them both comparable. The only difference was the number of feature maps for each of the convolutional layers - 5 and 10 feature maps in the first and second convolutional layers respectively.

As the aim of this experiment is to compare the performance of CNNs with respect DNNs, we decided to keep all other network settings the same as in our best performing DNN. Hence, we used minibatch of size 100, ReLu non-linear unit and Adam learning rule since it proved to yield the best performance in our previous experiments. Initially, we set the learning rule 0.001 with a plan to adjust it further by performing multiple runs for different values. However, due to much longer time needed to complete each epoch (100 slower than DNN), we were only able to investigate two different values of this parameter for each of the CNNs after just 50 epochs. Nonetheless, we could still observe the effect of different learning rates on our CNN architectures (Figure 1.).

As expected, the highest accuracy was recorded for deeper CNN (2 convolutional layers) and learning rate of 0.001. This architecture achieved 8.600×10^{-1} accuracy on the validation set compared with 8.413×10^{-1} for the same model with just 1 layer (Table 6.). This was due to deeper network's ability to extract more features from each image using higher number of feature maps in its second layer to classify an image.

We would expect that adding more convolutional layers would improve the accuracy even further, however we were not able to prove this claim due to limited computational resources.

It also seems that shallower network tends to overfit the data more

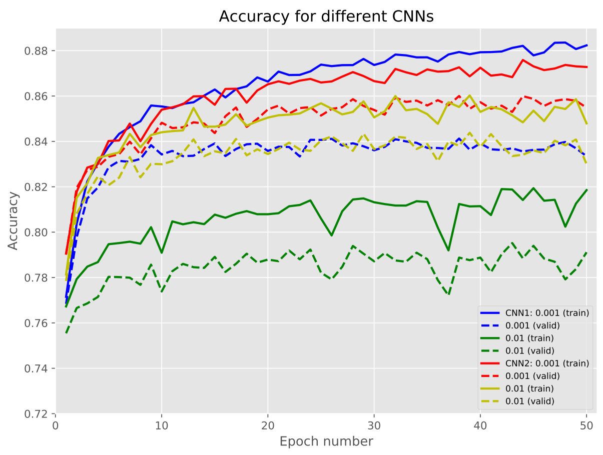

Figure 1. Accuracy on training and validation set for the investigated CNN implementations.

than its deeper version and performed better with respect to accuracy on the training set. This could be explained by the fact that the shallower network extracts smaller number of less advanced features from each image in the minibatch making it more vulnerable to noisy data and taking more assumptions about its structure (Simonyan & Zisserman, 2014b). Due to much higher number of convolutional operations, deeper CNN took on average 50% longer to complete each epoch. However, the improvement on validation accuracy was noticeable and since this measure along with testing time are more crucial aspects, our CNN containing 2 convoloutional layers was decided to be the most optimal.

Higher values of learning rate did not improve accuracies of any of the networks with the shallower implementation being more affected due the the lower number of extracted features from each image.

We also tried to apply batch normalisation to improve the performance of the CNN. However, due to difference in the network structure, batch normalisation algorithm works slightly different in case of CNNs. Here, the parameters of γ and β need to be collected per feature map, not per activation as it was done in our implementation used for DNN (Johannes, 2017). Hence, this resulted in drop in accuracy down to 7.548×10^{-1} for the deeper CNN when the batch normalisation has been applied.

Summing up, the best performing architecture of the tested CNNs was the one consisting of 2 convolutional layers, learning rate of 0.001 and no bath normalisation. This model will be used in the next section for comparison with our best performing DNN, this time on the testing set.

6. Test results

In this final section we will directly compare the performance of our two best architectures for DNN and CNN.

Since specific network architectures had been described in previous sections (sections 4 and 5), we will briefly remind the reader about their structures.

The best performing DNN was the model consisting of 4 hidden layers and L2 regularisation of 1.0×10^{-3}. In comparison, the chosen CNN contains 2 convolutional layers with 5 and 10 features in first and second layer respectively.

Both models are using the same learning rate of 0.001, Adam learning rule, minibatch size of 100 and ReLu activation function

Model	Training	Validation	Testing
DNN	**8.787**$\times10^{-1}$	8.549$\times10^{-1}$	8.395$\times10^{-1}$
CNN	8.751$\times10^{-1}$	**8.559**$\times10^{-1}$	**8.444**$\times10^{-1}$

Table 7. Accuracies on training, validation and testing sets after 100 epochs for two final implementations of different networks.

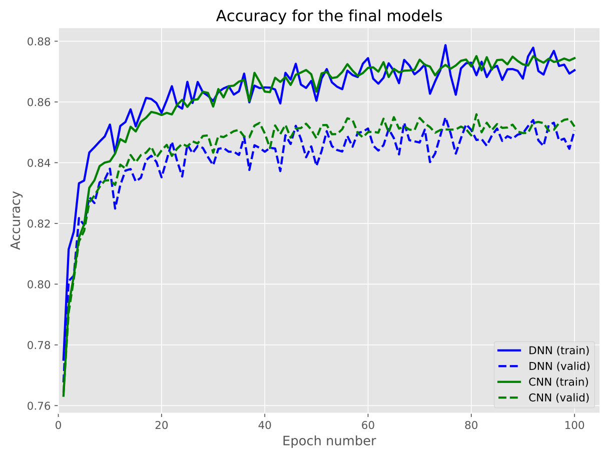

Figure 2. Accuracy on training and validation set for the best performing architectures of DNN and CNN.

which makes them easily comparable due a relatively big similarity of the models. Looking at the final reaults (Table 7.) we can notice that our implementation of CNN outperformed previously constructed DNN with respect to the accuracy on the testing set (8.444$\times10^{-1}$ compared to 8.395$\times10^{-1}$).

This might not seem like a big improvement considering the amount of time needed for the CNN to complete the training. However, we should remember that 2-layer CNN is still a very shallow implementation compared to recently used CNN models such as LeNet being 19-layers deep (Simonyan & Zisserman, 2014a).

We also did not have opportunity to apply additional optimisation techniques such as dropout, batch normalisation (adjusted to the CNN case) etc. The fact that a basic implementation of a CNN managed to outperform carefully optimised DNN with respect to testing data (unseen images) shows the power of convolutional networks and the proves why it is so popular in image recognition/classification tasks like ours.

The advantage of CNN comes from its convolution step where, instead of flattening the image into a vector, we are considering patches (receptive fields) of an image at a time. This, with shared weights across the receptive fields allows each convolutional layer to extract relevant features from each image in the minibatch (feature maps).

We are sure that adding more convolutional layers will improve our network performance even further. This could be achieved by more efficient implementations of `ConvolutionalLayer` and `MaxPoolingLayer` mentioned in the previous section (section 5). Another argument in favour of using CNNs in tasks like MNIST is that despite testing deeper versions of DNNs (up to 8 hidden layers), we did not manage to improve its performance much further, risking the progress of learning due to quickly increasing number of free parameters in the network.

Moreover, the accuracy plot (Figure 2.) for DNN seems to vary significantly when compared to CNN which indicates learning process being less stable for the DNN. This might be due to the value of regularisation (L2) being set too low or the learning rate being set too high (Johannes, 2017). This makes us to think that CNN are slightly easier to initialise when compared with DNNs, which seems to be backed by recent studies (Simonyan & Zisserman, 2014b).

DNN also tends to overfit the data slightly more than CNN achieving higher accuracy on the training set but lower on the validation. This might be due to the fact that DNNs do not consider image spacial structure. Each of its hidden units look at all the units in the layer below, making pixels that are spatially separate to be treated the same way as pixels that are adjacent. This may result in false assumptions (patterns) about the image resulting in overfitting.

All the above results make us conclude that CNNs seem to perform much better in image classification tasks like ours (Balanced MNIST) yielding better performance and being reliable (stable learning). This should not be treated as a surprise since CNNs had been around for quite a long time (Lecun et al., 1998) and had been specially designed for inputs being images.

7. Conclusions

Balanced MNIST task classification proved to be more complex than our previous experiments on a simplified MNIST dataset containing only digits. This time we were not able to exceed accuracy of 90% on the validation set. Nonetheless, we explored many commonly used optimisation techniques to construct our baseline deep neural network with the most efficient being L2 regularisation which increased the maximum validation accuracy by over 1% and reduced overfitting. Other popular techniques such as dropout did not yield any improvement despite learning rate adjustment and dropout policy suggested by Hinton (Srivastava et al., 2014).

Our research also considered implementation of RMSProp and Adam learning rules which proved to have positive influence on network performance by introducing per-parameter change learning rule adjustment with Adam being slightly more influential due to momentum term inclusion. The finding were consistent with previous studies claiming Adam's superiority over SGD (Ruder, 2016). Our future experiments could further be extended by including other learning rules such as AdaMax or Nadam which also gain popularity in recent architectures (Dozat, 2015).

We also explored batch normalisation commonly used to ease parameters initialisation and reduce overfitting (Ioffe & Szegedy, 2015). However, due to our relatively shallow architecture consisting of just 4 hidden layers we were not able to record any improvement in classification. However, due to its great popularity within the field we are motivated to explore this concept further by introducing deeper architectures in our future studies.

Moreover, we explored convolutional neural networks of different depths and number of feature maps which proved to be especially applicable to image recognition tasks like ours (Simonyan & Zisserman, 2014a). Experiments on CNNs were the most valuable part of this research since they showed us how including spatial structure of an image yields better results than previously studied DNNs. Therefore, we can state that CNNs are more powerful than ordinary neural networks in image classification tasks.

In our future research we would like to explore CNNs further by experimenting with deeper architectures using more advanced tools such as TenserFlow allowing us to speed up the computations significantly. Moreover, we would like to compare them with yet another type of recently popular Residual Neural Networks (He et al., 2016) used for image recognition.

References

AWS, 2017. URL https://aws.amazon.com/marketplace/pp/B01LZMLK1K?qid=1475196062961&sr=0-1&ref_=srh_res_product_title.

Bengio, Y and Lecun, Yann. Convolutional networks for images, speech, and time-series. 11 1997.

Coates, A., Carpenter, B., Wang, T., Wu, D. J., and Ng, A. Y. Text detection and character recognition in scene images with unsupervised feature learning. In *2011 International Conference on Document Analysis and Recognition*, pp. 440–445, Sept 2011. doi: 10.1109/ICDAR.2011.95.

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.

Dozat, Timothy. Incorporating nesterov momentum into adam. 2015.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021068.

Girosi, Federico, Jones, Michael, and Poggio, Tomaso. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995. doi: 10.1162/neco.1995.7.2.219. URL https://doi.org/10.1162/neco.1995.7.2.219.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016. doi: 10.1109/CVPR.2016.90.

He, Kaiming, Zhang, Xiangyu, and Ren, Shaoqing. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

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.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, and Karayev. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pp. 675–678, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3063-3. doi: 10.1145/2647868.2654889. URL http://doi.acm.org/10.1145/2647868.2654889.

Johannes, Kuhn, 2017. URL https://wiki.tum.de/display/lfdv/Batch+Normalization.

Karpathy, Andrej, 2015. URL http://cs231n.github.io/neural-networks-3/.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Lee, Honglak, Grosse, Roger, Ranganath, Rajesh, and Ng, Andrew Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 609–616, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553453. URL http://doi.acm.org/10.1145/1553374.1553453.

MATLAB. URL https://uk.mathworks.com/products/gpu-coder/features.html.

Nagi, J., Ducatelle, F., Caro, G. A. Di, and CireÅ§an, D. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pp. 342–347, 2011. doi: 10.1109/ICSIPA.2011.6144164.

Ng, Andrew Y. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pp. 78–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015435. URL http://doi.acm.org/10.1145/1015330.1015435.

Ruder, Sebastian. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

Simard, Patrice Y., Steinkraus, Dave, and Platt, John. Best practices for convolutional neural networks applied to visual document analysis. Institute of Electrical and Electronics Engineers, Inc., 2003. URL https://www.microsoft.com/en-us/research/publication/best-practices-for-convolutional-neural-networks-applied-to-visual-document-

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014a.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014b. URL http://arxiv.org/abs/1409.1556.

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.

Srivastava, Rupesh K and Masci, Jonathan. Compete to compute. In *Advances in Neural Information Processing Systems 26*, pp. 2310–2318. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5059-compete-to-compute.pdf.

Szegedy, Christian, Ioffe, Sergey, and Vanhoucke, Vincent. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. URL http://arxiv.org/abs/1602.07261.

TenserFlow. URL https://www.tensorflow.org/.