MLP Coursework 2: Learning rules, BatchNorm, and ConvNets

s1740192

Abstract

Deep learning has shifted the paradigm for building machine learning systems. Instead of careful feature engineering, the practitioner focuses on network design and selection. It is well known that network topology has a significant impact on the performance of recognition systems, but aspects such as activation functions, number of layers, learning rules and regularization methods should also be taken into account. The purpose of this study is fourfold. First, we conduct experiments on the task of handwritten digit and letter recognition using fully-connected networks, exploring the interplay between different activation functions and number of layers, initialization schemes and regularization. We expand this analysis to different learning rules; and, subsequently, to batch normalization. Finally, we investigate additional topologies by considering convolutional networks. We find that while network topology is important, we should not downplay the value of fine-tuned networks.

1. Introduction

Machine learning algorithms have become ubiquitous, powering not only web search engines, but also many consumer products (LeCun et al., 2015). Traditional machine learning systems require extensive domain knowledge and careful engineering as to find suitable features that can be fed to the next learning module (Goodfellow et al., 2016). Deep Learning has revolutionized this process by allowing the machine learning researcher and practitioner to focus instead on network design and selection. Deep neural networks are comprised of simple nodes that are often arranged in a particular topology such that it reflects the prior belief of the modeler on how the data should be perceived. For instance, in fully-connected feedforward neural networks, each node, after receiving information from a previous layer, propagates it forward to every node in the following layer; whereas, in a convolutional neural network, nodes are only connected to local patches of the previous layer. The latter is known to be a more faithful characterization of the visual cortex and therefore is able to better identify spatial structure. At each successive layer, deep learning methods are able to automatically learn more abstract representations of the input data and, while the topology of the network is certainly a crucial factor, there are other design aspects that the modeler should also take into account.

Expanding on previous work (s1740192, 2017), this study focuses on the integrated task of handwritten digit and letter recognition. In particular, we are interested in measuring the impact that not only different activation functions have on learning such task, but also the interplay between these and learning rules, dropout (Srivastava et al., 2014), batch normalization (Ioffe & Szegedy, 2015) and convolutional neural networks. For this purpose, in Section 2, we first test fully-connected networks where, in addition to different activations, we explore several architectures and relate these findings to our previous work. Subsequently, we expand this analysis by considering the use of dropout. In Section 3, we present advanced gradient-based stochastic optimizers and assess their performance by comparing with the standard Stochastic Gradient Descent (SGD) algorithm. Building on these previous experiments, in Section 4, we explain the concept of batch normalization and investigate its impact; whereas, in Section 5, we turn our attention to topologies beyond fully-connected feedforward networks, i.e. we briefly explain the idea behind convolutional neural networks and analyze their performance.

All results are assessed with respect to accuracy and, when relevant, we also consider the cross entropy loss. Unlike our previous work, where we used the Modified National Institute of Standards and Technology (MNIST) dataset, we now examine the Extended MNIST (EMNIST) Balanced dataset (Cohen et al., 2017). In this dataset, each sample is a 28×28 pixel grayscale image of a handwritten digit or letter. Despite the total number of labels being 62 (10 digits, 26 lower case letters and 26 upper case letters), we only consider a reduced set of 47 labels by merging the upper- and lower-case labels for the following letters: C, I, J, K, L, M, O, P, S, U, V, W, X, Y and Z. The size of our training, validation and test sets is $100,000$, $15,800$ and $15,800$ respectively. In all the previously mentioned sections, the reported performances correspond to the first two parts. In Section 6, we however report the results on the test set using a fully-connected deep neural network and a convolutional network that have shown good accuracy and generalization capacity on the validation set.

2. Baseline systems

Before proceeding with this analysis, we note that all networks are trained via backpropagation using Stochastic Gradient Descent (SGD) with a learning rate $\alpha = 0.006$ (based on our previous work) and a batch size of 100 for a total of 100 epochs. We consider architectures that range from 2 to 10 hidden layers, using 100 units per hidden layer. We

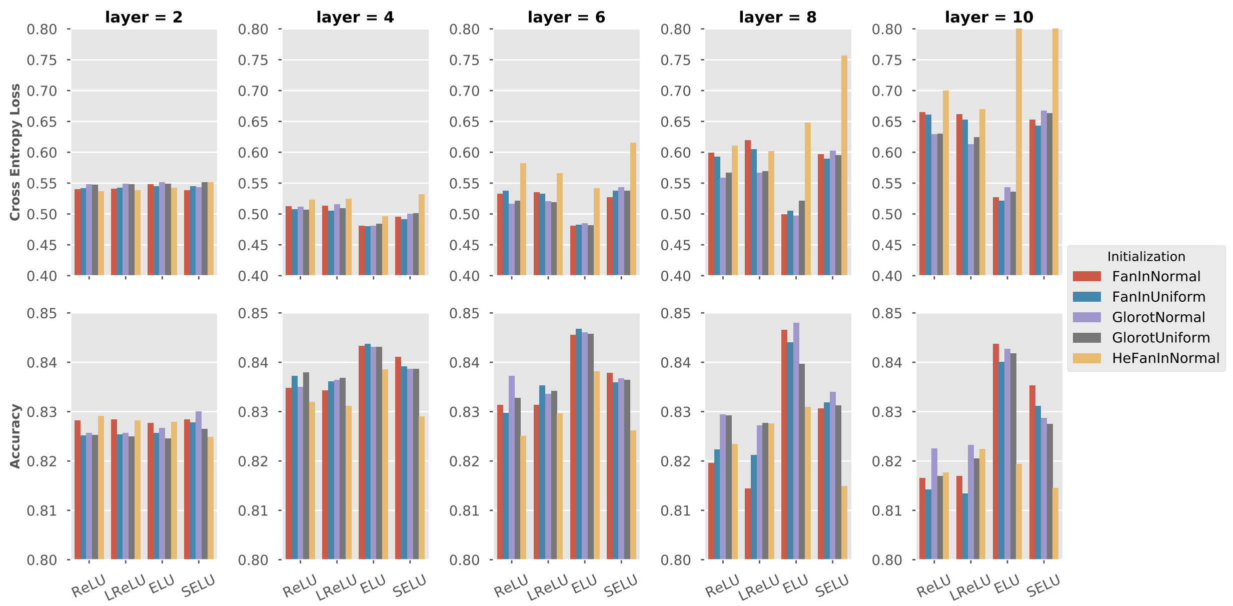

Figure 1. Cross entropy loss and accuracy on the validation set of non-regularized fully-connected networks with different activation functions, initialization schemes and number of hidden layers. Values observed after training for 100 epochs.

also test different activation functions: ReLU (Glorot et al., 2011), LReLU (Maas et al., 2013), ELU (Clevert et al., 2015) and SELU (Klambauer et al., 2017). Additionally, we consider several initialization schemes: FanInNormal, FanInUniform, GlorotNormal, GlorotUniform (Glorot & Bengio, 2010) and HeFanInNormal (He et al., 2015). All these schemes draw samples from distributions with zero mean, but different variances according to the number of incoming and outgoing connections for a particular layer. Due to space constraints we are unable to provide a more detailed specification. Instead, we refer the reader to our previous work (s1740192, 2017).

A summary of our results can be found in Figure 1. For legibility reasons, we do not show the performance curves over time. Instead, we analyze a bar chart that shows the validation error and accuracy after training for 100 epochs. Note that that one limitation of this plot is that we are unable to show which configuration showed the fastest convergence (without overfitting), but since we are not performing early stopping this is not as problematic. If we were to perform early stopping, perhaps a better criterion would be to compare the points at which the cross entropy loss is minimum. Indeed, unlike the previous assignment, we are now able to use early stopping, but, for simplicity reasons, we choose not to adopt any such rule. For instance, using stochastic regularization, the training process can become significantly noisy. In these circumstances, adopting a greedy approach is likely to lead to suboptimal results – a difficulty that is exacerbated for small datasets (see (Prechelt, 1998) for an early discussion, and e.g. (Mahsereci et al., 2017) for more recent developments). That said, if we were to adopt a particular early stopping rule, it would be wise to first compare different approaches.

In terms of depth, it can be seen that classifiers with 2 hidden layers are unable to learn effectively because, when compared to architectures of 4 hidden layers, the error is higher and the accuracy is noticeably lower – a clear sign of underfitting. Networks with 8 or more hidden layers appear to be overfitting (for ReLU and LReLU, it starts when this number increases beyond 4). Consequently, a good compromise between generalizability, accuracy and model complexity can be obtained if we choose networks with 6 hidden layers. Throughout this study, we will however also test networks with 10 hidden layers, so that we are able to hypothesize on the effects of regularization and optimizers on networks of increasing depth.

Regarding initialization strategies, a more delicate analysis is required. Although not shown in the figure, initializing the weights with an increased variance, as is done in HeFanInNormal, allows the network to learn faster during the first epochs. Since we are not applying any regularization, it also begins to overfit sooner, and naturally this effect becomes more severe for increasingly complex models. As a result, its generalization capacity degrades over time (more so than others) and, at epoch 100, it is considerably worse than the rest. For instance, we can observe differences that can go up to roughly 1%, 1.5% and 2% in accuracy for networks of depth 6, 8 and 10 respectively. This effect is not as pronounced for ReLU partly because HeFanInNormal has been specifically designed for such activations. Therefore, as the figure suggests, HeFanInNormal without regularization can lead to suboptimal behavior, but this does not mean that it cannot possibly be an effective scheme when coupled with such techniques. In what follows, we test a widely used regularization method (dropout), but, due to time constraints, we are unable to test it with different initialization schemes. Instead of HeFanInNormal, we decide to choose an initialization scheme that is more stable. As the performance curves would have shown, it is difficult to state confidently which initialization strategy (if any) is likely to lead to the best performance. We have mentioned the importance of sensitivity analysis and this

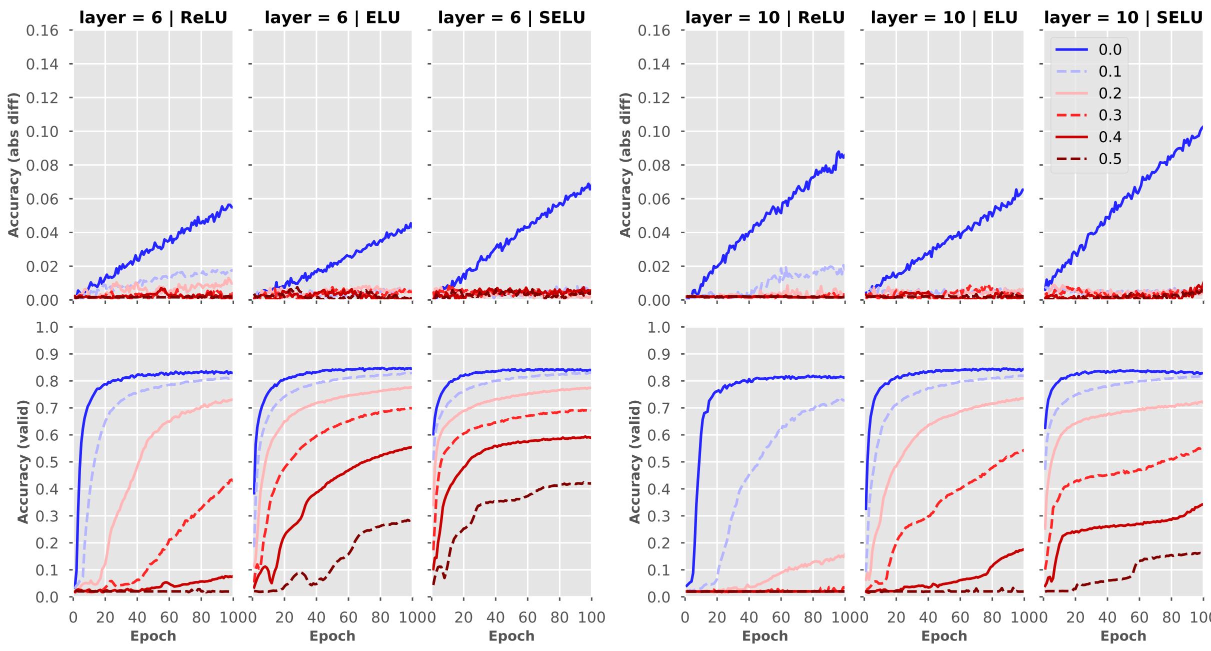

Figure 2. Absolute difference in accuracy between training and validation (*top*) and accuracy on the validation set (*bottom*) of regularized fully-connected networks with 100 hidden units. Different levels of dropout, activation functions and number of hidden layers are tested.

continues to hold true for this work. Due to the volume of tests we need to carry out for the following sections and the additional tests that this would have entailed, we reluctantly decide not to follow our own advice and pursue a more pragmatic approach. Based on the results, we observe that, for ELU and SELU networks of 6 hidden layers, FanInNormal reaches the lowest error, while being among the top in terms of accuracy. For this reason, and in addition to being the method recommended by (Klambauer et al., 2017), we only consider FanInNormal in all subsequent experiments.

While all the activation functions exhibit similar error and accuracy values for networks with 2 hidden layers (approximately 0.55 and 82.5%), their differences become more apparent with increasing depth. A noteworthy exception are ReLU and LReLU networks, whose behavior remain similar throughout – a fact that we also have observed in our previous work. In particular, the difference in accuracy between ELU and ReLU (and LReLU) is about 0.75%, 1%, 2% and 2.5% for networks of depth 4, 6, 8 and 10 respectively. Similarly, the difference in accuracy between ELU and SELU is approximately 0.5%, 1%, 1.5%, 1.25%. The former set of results are reassuring because we reported similar behavior in our previous work; the latter, however, are to a certain degree contradictory – we observed that SELU was less susceptible to overfitting, but for this task, this seems to no longer be the case. This leads us to believe that we are either in the presence of spurious events or this relationship is task-dependent. In order to assess this matter further, we need to perform additional tests.

We now investigate the effect of (stochastic) regularization. In particular, we consider dropout, a method that tries to solve the problem of overfitting from a different perspective than that of L1/L2 regularization. Summarily, the idea is to randomly disable units during training, effectively learning an ensemble of models. For a more detailed description, we refer the reader to (Srivastava et al., 2014). We begin our set of experiments by considering networks whose number of hidden units is fixed (100) and we test fractions that range from 0 (no dropout) to 0.5 with steps of 0.1. From Figure 2, we observe that as the fraction of units dropped increases, the overall accuracy decreases. This is not surprising because by dropping with a given probability r, we are training networks whose effective size is $n_l \cdot (1 - r)$, where n_l denotes the number of units at layer l. In addition, since we are also applying dropout to visible units, the process is equivalent to training an ensemble of classifiers where each chooses different inputs. When the number of epochs is 100 and $r > 0.1$, the fact that we are training ensembles cannot outweigh the reduction in effective size and as a result the accuracy becomes worse. Another unsurprising aspect is that we are now able to train significantly more robust models, as is shown by the curves corresponding to the absolute difference in accuracy between training and validation. In order to effectively apply dropout, we therefore need to increase our learning rate (each unit is being updated less frequently), increase the number of units (adjust effective size) and increase the number of epochs (ensemble training is more complex). Before proceeding, a brief note on activation functions. We observe that SELU networks are able to learn faster when the effective size of the network is too small, but they do not offer any advantage over ELU networks otherwise. The accuracy values for the pairs ($r = 0.0$, $r = 0.1$) are (82.8%, 81.1%), (84.6%, 83.0%), (83.8%, 83.0%) for networks with 6 hidden layers using ReLU, ELU and SELU respectively. These values are not better for networks with 10 hidden layers, but the learning curves are informative – e.g. the differences between ReLU, ELU and SELU become more pronounced.

In the next set of experiments, we choose $r = 0.5$, increase the number of hidden units to 200 and the num-

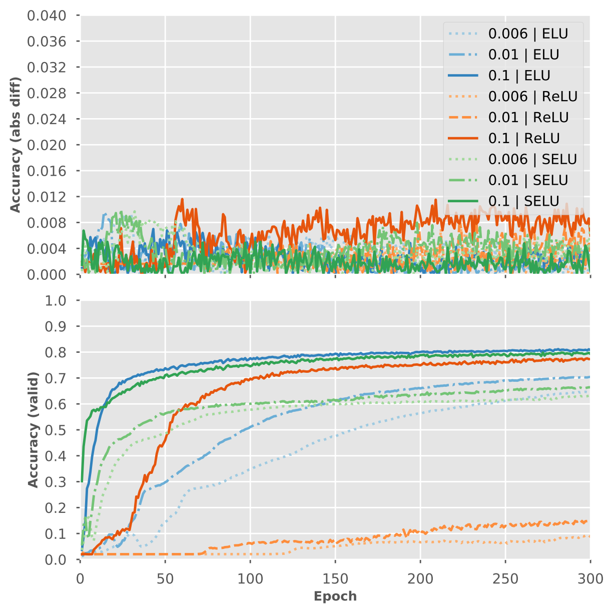

Figure 3. Absolute difference in accuracy between training and validation (*top*) and accuracy on validation (*bottom*) of regularized fully-connected networks with 6 hidden layers, 200 hidden units and $r = 0.5$. Different learning rates for SGD are tested.

ber of epochs to 300. We test different learning rates: $\alpha \in \{0.006, 0.01, 0.1\}$. From Figure 3, we observe that increasing the learning rate is indeed crucial, even more so in ReLU networks. For these learning rates, 0.1 is clearly the ideal value and, in this case, ELU shows the best performance (81.1%) compared to SELU (79.3%) or ReLU (77.3%). We note however that it would have been interesting to test higher rates, but due to time constraints we were unable to do so. It also becomes more clear now that SELU is able to learn faster during the first epochs and that it appears to be a general statement. This is perhaps unsurprising because compared to ELU there are multiplicative factors greater than 1, which can be equivalent to training with a higher learning rate. However, given enough training time, it appears to come at a cost in terms of accuracy and we believe (Klambauer et al., 2017) do not test this effect thoroughly. A higher "effective" learning rate can indeed lead to worse solutions, but we are not yet certain whether this is the case. Fundamentally, additional tests are required.

We now evaluate ELU and SELU networks with 200, 400, 600 and 1000 hidden units which is equivalent to effective sizes of 100, 200, 300 and 500 ($r = 0.5$). We also increase the number of epochs to 500 to reflect the fact that we are effectively training more complex ensembles. Note that training a network of 1000 hidden units for 500 epochs takes us two full days and, for this reason, we were unable to train larger networks. From Figure 4, we observe that the most tangible boost in performance occurs when we increase the number of hidden units from 200 to 400 (roughly from 81% to 85%), but successive increases are reflected in more accurate systems. Despite training these networks for 500 epochs, we are still able to maintain a reasonable generalization capacity. Indeed, if we compare these absolute differences in accuracy with that of a non-

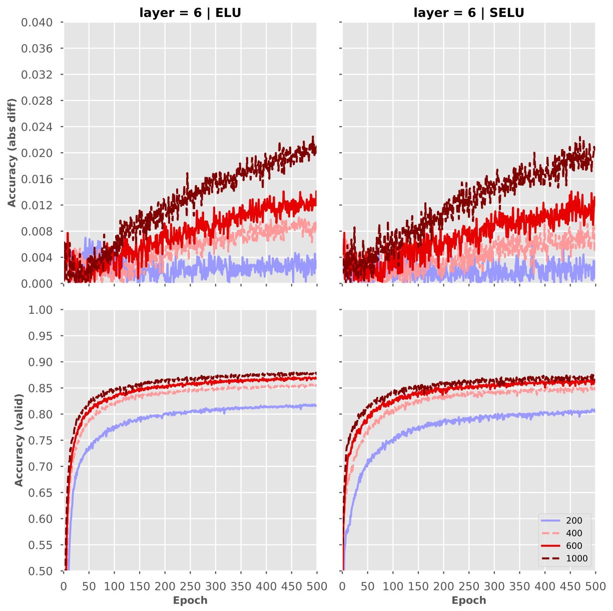

Figure 4. Absolute difference in accuracy between training and validation (*top*) and validation accuracy (*bottom*) of regularized fully-connected networks with a learning rate of 0.1 (SGD) and $r = 0.5$. Different number of hidden units are tested.

regularized network, we see a two- to threefold increase for the latter. In addition, SELU is able to achieve 86.7% on the validation set, whereas for ELU we observe a value of 87.7%.

3. Learning rules

We have noted the need for higher learning rates (SGD) such that regularized networks using dropout can learn efficiently. In this section, we analyze more advanced gradient-based stochastic optimizers. In particular, we investigate if there are more efficient ways to learn without necessarily increasing the learning rate as this can also lead to poor solutions. In SGD, each parameter vector $\boldsymbol{\theta}$ is updated according to the following rule:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \cdot \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}), \tag{1}$$

where $\boldsymbol{\theta}_t$ denotes a parameter vector at time t, α is the learning rate and $\nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$ corresponds to the gradient of the cost function at time t w.r.t. $\boldsymbol{\theta}$. RMSProp (Tieleman & Hinton, 2012) expands on this update by scaling the gradients according to an exponential moving average (EMA) of its magnitude. This scaling has a positive impact on learning because it allows separate adaptive rates: the learning rate is set manually, but it is multiplied by a local factor that is estimated based on the EMA of the square of the gradients and can vary substantially for different parameters. By taking the EMA, it also includes information from adjacent minibatches, resulting in a more stable rule. The update corresponds to

$$\boldsymbol{v}_t = \beta_2 \cdot \boldsymbol{v}_{t-1} + (1-\beta_2) \cdot (\nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}))^2, \tag{2}$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \cdot \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}) / (\sqrt{\boldsymbol{v}_t} + \epsilon), \tag{3}$$

where β_2 is the weight of the EMA (0.9 by default), $\boldsymbol{v}_t$ is the estimate of the magnitude of the uncentered (raw)

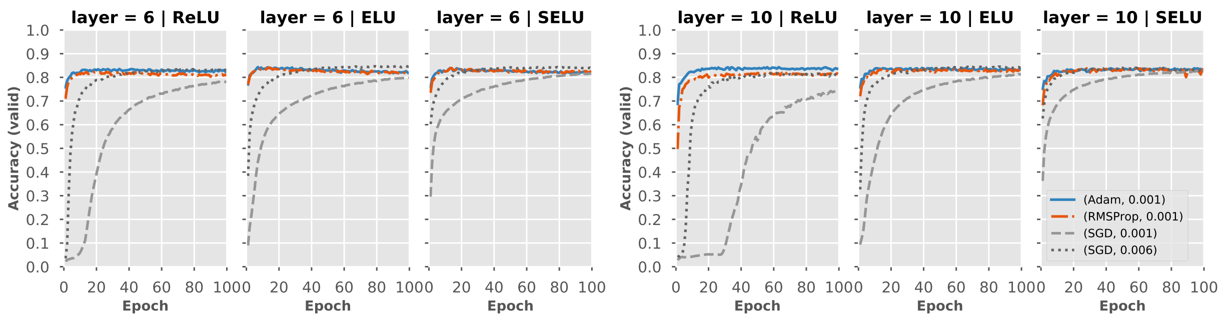

Figure 5. Accuracy (validation) of non-regularized fully-connected networks with 100 hidden units. Different learning rules are tested: Adam ($\beta_1 = 0.9, \beta_2 = 0.999$), RMSProp ($\beta_2 = 0.999$) and SGD. The learning rates are 0.001 (Adam, RMSProp, SGD) and 0.006 (SGD).

second moment of the gradient, often initialized at zero, and ϵ is a small positive scalar to prevent division by zero (e.g. 10^{-8}). Note that the we have included ϵ for numerical stability reasons. The operators ^{2} and $/$ denote scalar or elementwise square and division, depending on the context.

More recently, another optimizer has been proposed by (Kingma & Ba, 2014). The authors argue that rules using EMAs, including RMSProp, suffer from biases due to zero initialization – this leads to biased estimators that are particularly problematic during the first epochs and when the weights of EMAs are close to one. In order to solve this, the authors then propose bias-corrected estimates. The resulting algorithm (Adam) is as follows:

$$\boldsymbol{m}_t = \beta_1 \cdot \boldsymbol{m}_{t-1} + (1-\beta_1) \cdot \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}), \tag{4}$$

$$\hat{\boldsymbol{m}}_t = \boldsymbol{m}_t / (1-\beta_1^t), \tag{5}$$

$$\boldsymbol{v}_t = \beta_2 \cdot \boldsymbol{v}_{t-1} + (1-\beta_2) \cdot (\nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}))^2, \tag{6}$$

$$\hat{\boldsymbol{v}}_t = \boldsymbol{v}_t / (1-\beta_2^t), \tag{7}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \cdot \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon), \tag{8}$$

where β_1 is the weight of the EMA estimating the expected value of the gradient over the last minibatches (also known as momentum) and $\hat{\boldsymbol{m}}_t$ and $\hat{\boldsymbol{v}}_t$ are the bias-corrected estimates. Both $\boldsymbol{m}$ and $\boldsymbol{v}$ are initialized at zero. Note that momentum is not an unique feature of Adam: there are many alternatives with such option, including SGD and RMSProp with momentum. The authors recommend $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

In the remainder of this section, we evaluate these learning rules. Since we are interested in testing these for different hyperparameters (α, β_1, β_2) we have not run these sets of experiments on the best network we have found so far as it would have taken more computing power than we currently have. Instead, we explore the same configurations of non-regularized networks. It would have been interesting however to also take into account dropout since their authors claim that Adam performs significantly better than SGD in cases where the gradients are sparse. This would for instance occur when dropping out visible units with high r (as done before). In addition, we consider $\alpha \in \{0.001, 0.006\}$, $\beta_1 \in \{0.9, 0.8, 0.7\}$, $\beta_2 \in \{0.999, 0.95, 0.9, 0.8\}$. Due to space constrains and for legibility reasons, we only show the best configurations of β_1 and β_2, which turned out to be the recommended by (Kingma & Ba, 2014) – this includes β_2 in RMSProp. A learning rate of 0.006 also revealed to be too high for both Adam and RMSProp which is not surprising because, since we are scaling the gradient by its magnitude, updates that would otherwise be small, become bigger (in addition to momentum for Adam). Therefore, we expect the optimal learning rate to be lower than that of SGD. From Figure 5, we observe that Adam is particularly effective for ReLU networks and that, despite α being lower, both Adam and RMSProp allow networks to learn faster than SGD with $\alpha = 0.006$. At epoch 100, the accuracy of systems using Adam and RMSProp can be lower than that of SGD, but this is due to overfitting (mitigated by e.g. dropout). We believe a more representative measure of the overall learning process is the mean of the accuracy values discounted by the epoch as Table 1 shows. According to this measure, Adam emerges as the best gradient-based stochastic optimizer and, for this reason, we use this algorithm in all subsequent experiments (unless stated otherwise).

4. Batch normalization

So far, we investigated different learning rules as to accelerate the learning process which, as seen in Section 2, is of particular importance when training robust networks. We now examine batch normalization (BN) which, according to (Ioffe & Szegedy, 2015), should help us in this regard. The authors begin by acknowledging the presence of an effect known as covariate shift. Summarily, this phenomenon occurs when the distribution of the features presented to a model changes. In such circumstance, a model can only learn effectively if, in addition to adjusting its parameters due to the inherent learning process, is also able to track the resulting changes in distribution of the features. It then becomes clear that, if we are able to mitigate these changes, the training becomes less difficult. In feedforward neural networks, since information is propagated forward, parameter updates of previous layers affect the inputs of layers above – this is known as interval covariate shift. BN seeks to mitigate this problem by normalizing these inputs. In particular, let $\mathcal{B} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$ denote a given minibatch of size m, where each input $\boldsymbol{x}_i \in \mathbb{R}^d$ is a d-dimensional vector to be standardized. At test time, the BN transform applied to $\boldsymbol{x}_i$ corresponds to

$$\mathrm{BN}(\boldsymbol{x}_i; \boldsymbol{\beta}, \boldsymbol{\gamma}) = \boldsymbol{\beta} + \boldsymbol{\gamma} \odot \frac{\boldsymbol{x}_i - \hat{\mathbb{E}}[\boldsymbol{x}]}{\sqrt{\widehat{\mathrm{Var}}[\boldsymbol{x}] + \epsilon}}, \tag{9}$$

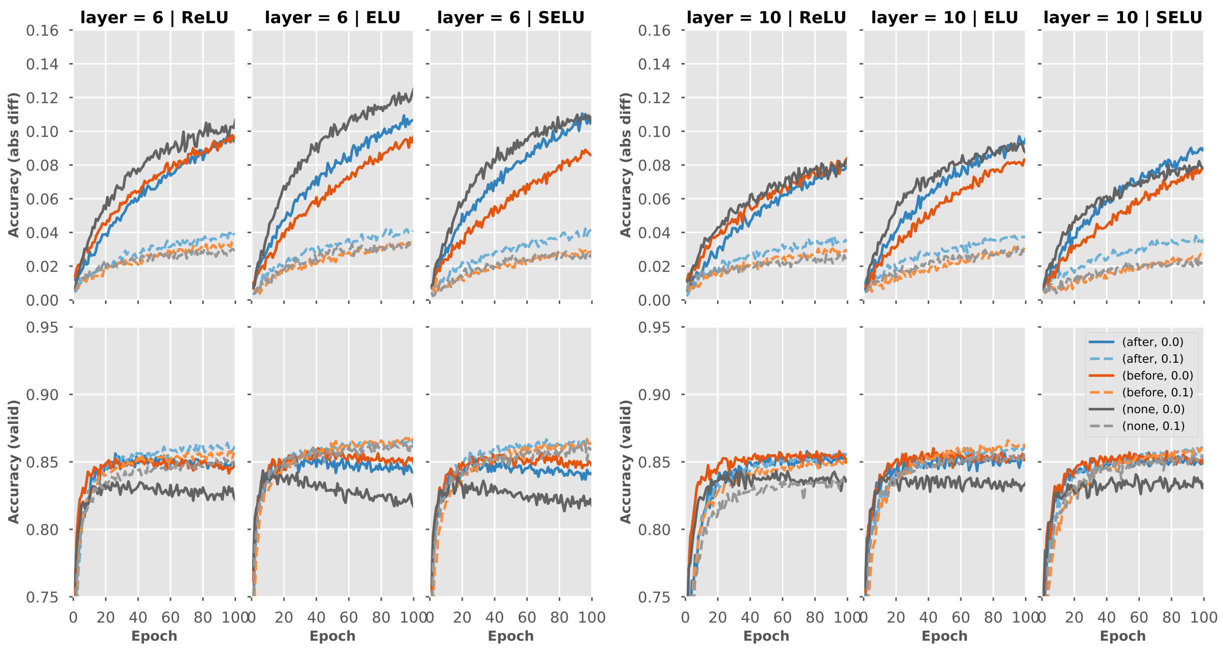

Figure 6. Absolute difference in accuracy between training and validation (*top*) and accuracy on validation (*bottom*) of fully-connected networks with 100 hidden units. Different configurations of BN (after, before, none) and dropout $(0.0, 0.1)$ are tested.

where $\odot$ denotes the elementwise product, ϵ is a small positive scalar (e.g. 10^{-5}) and $\hat{\mathbb{E}}[\boldsymbol{x}]$, $\widehat{\text{Var}}[\boldsymbol{x}]$ are unbiased estimators of the first two central moments of the inputs $\boldsymbol{x}$. These are estimated during training, often using EMAs with weight 0.99, and initialized at zeros and ones respectively (Chollet et al., 2015). This transform is parameterized by $\boldsymbol{\beta} \in \mathbb{R}^d$ and $\boldsymbol{\gamma} \in \mathbb{R}^d$, corresponding to a location shift (mean) and scale (standard deviation) and are learned during training. Additionally, at training time, the two central moments are estimated using maximum likelihood based on the samples from the current minibatch. Due to space constraints, we are unable to provide a more detailed description of the training process (e.g. gradients) – we refer the reader to (Ioffe & Szegedy, 2015).

In order to evaluate BN, we consider networks with 100 hidden units per hidden layer (6 and 10) and dropout probability $r \in \{0.0, 0.1\}$ (dropout is applied after BN). We test one approach where BN is applied *before* activations and another where BN is used *after* activations. The latter is motivated by the findings of (Mishkin, 2016). We compare these with similar configurations without BN. These results are shown in Figure 6 and Table 2. Surprisingly, we observe that, as opposed to what (Klambauer et al., 2017) suggest, combining BN with SELU leads to better performance. Again, we believe the authors do not test this thoroughly and $\alpha = 10^{-5}$ on the MNIST task seems particularly low for SGD (implicit use), which, as noted in Section 2, biases the results. In addition, (Ioffe & Szegedy, 2015) state that BN enables higher learning rates, not lower. Another interesting observation is that our results seem to validate the findings of (Mishkin, 2016) in that it is preferable to use BN after ReLU, but this does not necessarily apply to ELU/SELU. In fact, Table 2 shows that it is inconclusive. On the other hand, it seems clear that we should simultaneously use BN and dropout. The highest validation accuracy (86.7%) is observed for a network that, in addition to dropout, uses BN before ELU. Consequently, the next logical step is to test the model with $r = 0.5$ and 1000 hidden units, as presented in Section 2. We noted that the previous model takes us two full days to train. In this network, the training time roughly doubles due to the additional computational cost of BN and Adam (as opposed to SGD). Moreover, an unforeseen problem with the source code of BN led us to repeat all the experiments presented in this section and, as a result, we were unable to test this network.

5. Convolutional networks

Unlike the previous sections, where we explored fully-connected (standard) networks, we now investigate a different topology that is more suitable for image recognition and, in general, spatial structure detection. Convolutional neural networks (CNN) extend standard networks with two types of layers that can also be stacked to create deeper models (LeCun et al., 2015). In convolutional layers, the hidden units are arranged according to feature maps and, in each map, these units are only connected to local patches of the previous layer, known as local receptive fields. In turn, each receptive field has a particular set of weights (kernel/filter). The dimension of the feature map is thus determined by the shapes of the input and the kernel, in addition to the stride of the receptive field. Moreover, in each feature map, units are constrained to extract the same feature and, as such, the weights are shared across the receptive fields. This is particularly important as it allows the network not only to detect motifs (e.g. edges) irrespective of their locations, but also reduces the number of parameters to learn. Hence, compared to a standard model without regularization, CNNs are less prone to overfitting. Given this constraint, the filtering operation can also be seen as a (discrete) convolution.

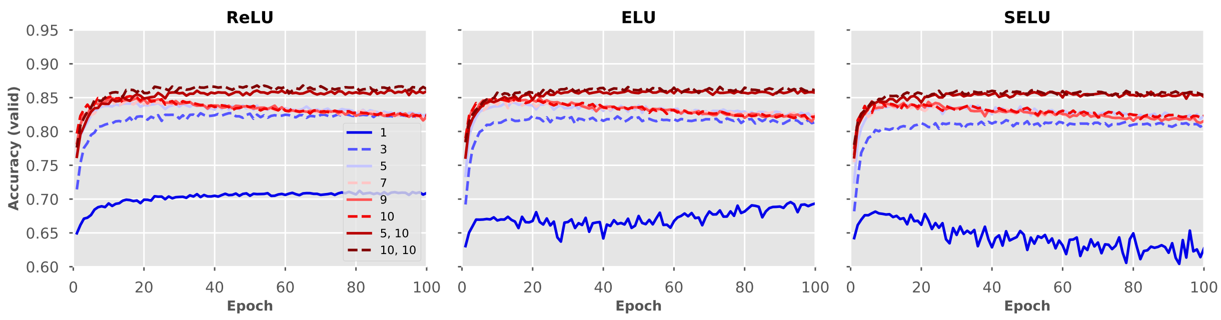

Figure 7. Accuracy (validation) of convolutional networks with different number of feature maps (one and two convolutional layers). Best accuracy values for CNNs with [10, 10] feature maps: ReLU (86.8%, epoch 65), ELU (86.6%, epoch 73), SELU (86.2%, epoch 52).

In terms of implementation, we can take into account the existence of optimized matrix multiplication routines and implement the convolution operation as such. Indeed, we followed this approach by arranging each receptive field into a column and we tried to generate views of the objects whenever possible as to reduce the computational overhead. In addition to the convolutional layer, another important layer is maxpooling, where we combine adjacent features by taking its maximum. Consequently, not only it enables the detection of more course-grained motifs in the subsequent layers, but also allows CNNs to become more robust to translations and distortions of the input. Since this layer only performs downsampling, it is completely specified by the shape of the pooling regions and the stride.

We now consider CNNs on the EMNIST task. We evaluate networks of one and two convolutional layers, each followed by a non-overlapping maxpooling layer with 2x2 regions. We use convolutional kernels of dimension 5x5 and stride 1. Each CNN has a final fully-connected hidden layer. We test different number of feature maps: $\{1, 3, 5, 7, 9, 10\}$ for CNNs with one convolutional layer and $\{[5, 10], [10, 10]\}$ for CNNs with two. From Figure 7, we observe that we are able to achieve better performance as the number of filter maps increases, which is naturally explained by the ability to detect an increasing number of different motifs (shapes) in image patches. CNNs with 1 and 3 feature maps lack the required flexibility to correctly classify the images. On the other hand, CNNs with more than 5 begin overfitting after 10 to 40 epochs. For CNNs with one convolutional layer, the maximum accuracy is observed for ReLU networks with 10 feature maps (85.2%, epoch 11). It is surprising to note that the performance of ReLU on CNNs seems to be marginally better than ELU (84.9%, epoch 9). SELU seems to perform the worst (84.4%, epoch 8). CNNs with [10, 10] feature maps are able to achieve the best results for all activations. An interesting, but not surprising observation is that these do not show signs of significant overfitting. In particular, the fully-connected hidden layer is remarkably smaller due to the successive application of convolution (without padding) and maxpooling. Indeed, the latter are more expressive (more nonlinearities) than the former and there is almost a sevenfold decrease in the number of parameters. We presume that more layers and smaller kernels are capable of boosting performance.

6. Test results

In this section, we analyze the performance of our best models on the test set. The choice is based on validation accuracy and generalization capacity. We test the ELU network with 1000 hidden units (6 hidden layers) and dropout $r = 0.5$ using SGD (see Sections 2 and 4 for details) after being trained for 500 epochs. Regarding CNN, we retrain the ReLU network with [10, 10] feature maps (same initialization) and and perform early stopping based on the previous validation results (maximum accuracy), i.e. epoch 65. The fully-connected network achieves 86.9% on the test set (+2.9% on train, +0.8% valid.), whereas for the CNN we observe 85.7% (+3.5% on train, +1.1% valid.). The fully-connected network is more accurate and generalizes better, but takes longer to train and has almost 470 times as many parameters as the CNN. We expect the CNN to be able to achieve better results after additional fine-tuning.

7. Conclusions

In this study, we analyzed the interplay between activation functions and architectures of different size, depth and topology. We also considered dropout, several learning rules and batch normalization. Some experiments in (Klambauer et al., 2017) deserve further analysis: we have consistently achieved better results using ELU in fully-connected networks and batch normalization seems to improve the performance of all networks, including those using SELU. Dropout seems an effective method if we aim to build more robust systems. However, the training procedure is costlier and, as a result, advanced learning rules become even more relevant. Adam achieves good results, but additional tests are required. For instance, it would be informative to compare Adam to SGD and RMSProp with momentum. We also tested convolutional networks and ReLU achieved marginally better results than ELU. It is unclear if this statement can be generalized and, as such, deserves further investigation. Our fully-connected network with dropout performed better than our best convolutional network, which shows that fine-tuning is an important process and should not be neglected. We expect convolutional networks to achieve better results and it might be worthwhile to also test smaller kernels, different strides and zero padding. Additional interesting approaches are proposed by (Springenberg et al., 2014) and (Huang et al., 2016).

Layer	6			10		
Activation	ReLU	ELU	SELU	ReLU	ELU	SELU
Adam	**4.18**	**4.24**	**4.20**	**4.10**	**4.20**	**4.17**
RMSProp	4.07	4.22	4.16	3.73	4.12	4.05
SGD	2.66	3.50	3.86	2.11	3.43	3.93

Table 1. Mean validation accuracy discounted by epoch (division) and multiplied by a factor of 100. These results correspond to fully-connected networks using Adam ($\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$), RMSProp ($\alpha = 0.001, \beta_2 = 0.999$) and SGD ($\alpha = 0.006$). Note the difference between Adam and RMSProp/SGD in ReLU networks compared to ELU/SELU. Momentum seems particularly helpful because ReLU has an inactive state whose gradient is zero. As previously noted, compared to SELU, ELU appears to be slightly better.

Layer	6						10					
Batchnorm	None		Before		After		None		Before		After	
Dropout	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1
ReLU	82.2%	85.1%	84.7%	85.6%	84.7%	**86.2%**	83.7%	83.5%	85.2%	85.2%	85.3%	**85.6%**
ELU	81.7%	86.2%	84.9%	**86.7%**	84.2%	86.3%	83.5%	85.2%	85.5%	**86.0%**	85.2%	**86.0%**
SELU	81.8%	86.0%	84.7%	**86.4%**	84.0%	**86.4%**	83.4%	85.5%	85.2%	**86.0%**	84.9%	85.8%

Table 2. Validation accuracy after training fully-connected neural networks with different configurations of batch normalization and dropout for 100 epochs. It appears that batch normalization is more effective when applied after ReLU, but the best approach for ELU and SELU networks is unclear. Dropout with $r = 0.1$ is particularly helpful because all networks without regularization show signs of overfitting.

References

Chollet, François et al. Keras. https://github.com/fchollet/keras, 2015.

Clevert, Djork-Arné, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv:1511.07289*, 2015.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. EMNIST: an extension of MNIST to handwritten letters. *arXiv:1702.05373*, 2017.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *PMLR*, volume 9, pp. 249–256, 2010.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier neural networks. In *PMLR*, volume 15, pp. 315–323, 2011.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv: 1502.01852*, 2015.

Huang, Gao, Liu, Zhuang, Weinberger, Kilian Q, and van der Maaten, Laurens. Densely connected convolutional networks. *arXiv:1608.06993*, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *PMLR*, volume 37, pp. 448–456, 2015.

Kingma, Diederik P. and Ba, Jimmy. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*, 2014.

Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-Normalizing Neural Networks. *arXiv: 1706.02515*, 2017.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521:436–444, 5 2015. doi: 10.1038/nature14539.

Maas, Andrew L., Hannun, Awni Y., and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, 2013.

Mahsereci, Maren, Balles, Lukas, Lassner, Christoph, and Hennig, Philipp. Early Stopping without a Validation Set. *arXiv:1703.09580*, 2017.

Mishkin, Dmytro. ducha-aiki: caffenet-benchmark - Batch Normalization. https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md, 2016.

Prechelt, Lutz. Early stopping-but when? *Neural Networks: Tricks of the trade*, pp. 553–553, 1998.

s1740192. MLP Coursework 1: Activation Functions. 2017.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. *arXiv:1412.6806*, 2014.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. In *JMLR*, volume 15, pp. 1929–1958, 2014.

Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.