MVA - Deep Learning In Practice
TP1

Ariane ALIX, Marie HEURTEVENT, Alexandre PACAUD

February 10th, 2020

Exercise 1

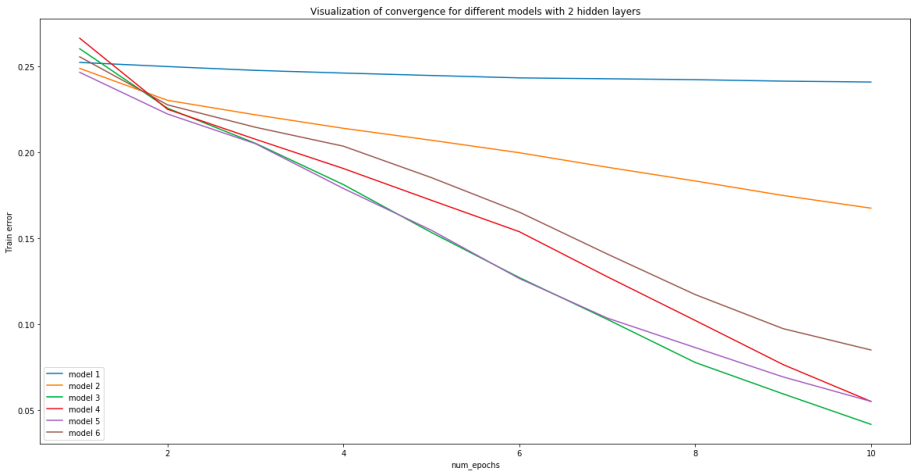
1.1 Question 1: Impact of the architecture of the model

The initial setup (one linear layer) gives an accuracy of 59.40% because it tries to separate the data linearly, which is not possible.

We then tested models with two linear layers, with different functions of activation and different numbers of neurons (which is both the number of outputs of the first layer and the number of inputs of the second layer). We obtained the accuracies shown in Figure 1 of the Appendix.

The best results were obtained with a tanH activation function and around 75 neurons (accuracy of 77.35%).

We then tried to add another hidden linear layer, and tested different activation functions and numbers of neurons in each layer.



The first three models had the same three linear layers, but each had a different activation function :

Activation function	Accuracy
sigmoid [model 1]	59.60 %
tanh [model 2]	78.20 %
relu [model 3]	99.80 %

Since the ReLU function yielded the best results, we then tried different numbers of neurons in each of the three layers, as seen in models 4 - 6.

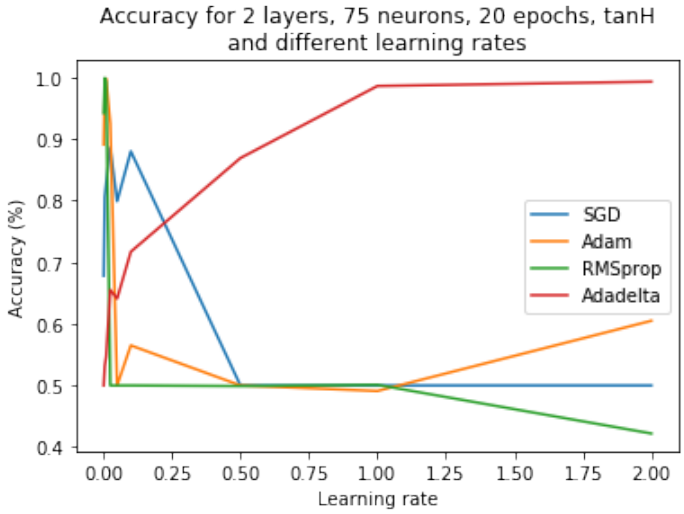
Our best model is therefore made up of three linear layers:

- self.l1 = nn.Linear(2, 100)
- self.l2 = nn.Linear(100, 150)
- self.l3 = nn.Linear(150, 1)

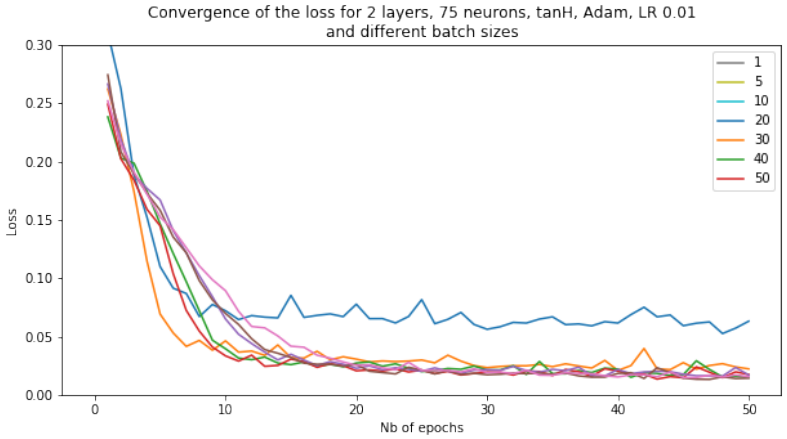
1.2 Question 2: Impact of the optimizer

Globally, we see a similar behaviour for the three models: a small learning rate will need more epochs to converge while a big learning might not converge at all. The optimal learning rate is around the same value for all. Results obtained for the three models are plotted in Figure 2 (see Appendix).

Restarting from the best 2-layer model from the previous question, we tried to improve it by: increasing the number of epochs (which seems a good idea given the steep decrease of the loss even at epoch 10), changing the batch size, the optimizer and the learning rate.



This time, the learning rates cause different behaviours for the different optimizers. The best results were obtained with Adam and a learning rate of 0.01 (99.90%) and RMSprop with a learning rate of 0.005 (99.80%). We tried to improve the Adam version by changing the batch size and increasing the number of epochs.



For 50 epochs and batch size 5, 10 and 20, the model reached 100%. Increasing the number of epochs clearly helps in getting the best results when the learning rate is well chosen. We also notice that some batch sizes seem to cause a convergence to a local minima (see dark blue curve for size 20).

From now on we only look at 2-layer networks. Indeed, the Universal Approximation Theory states that every continuous function an be approximated by a a network of two linear layers. The only limitation is that N (the number of hidden neurons) might be large. In our case however, 75 hidden neurons brings almost-perfect results and is super fast, and a 3-layer network would therefore only complicate the case (plus in our previous tests they needed more than 75 hidden neurons in total).

1.3 Question 3: Impact of the loss function

We changed the MSELoss to the Binary Cross-Entropy one, and retried with the previous 2-layer network (Adam optimizer, 75 hidden neurons). Some graphs obtained by testing this setup with different batch sizes and learning rates are in Figure 3 of the Appendix.

We can notice the following things from the graphs:

- Globally, we converge faster with this new loss (at least 99.5% of accuracy obtained for all batch sizes in less than 10 epochs),
- Generally, increasing the size of the batches makes the computation longer but the convergence faster (hence better results in fewer time in the end),
- The learning rates 0.025 or 0.05 seem more appropriate than 0.01 since they converge faster to a good solution.

1.4 Question 4: Prediction on test set

Reusing the best 2-layer previous model (Adam optimizer with a 0.05 learning rate, 75 hidden neurons, BCEWithLogit Loss and a 40 batch size), we got an accuracy of 100% on the test set in 10 epochs, and the convergence is plotted in Figure 4 of the Appendix.

Exercise 2

1.5 Question 1: Impact of the architecture of the model

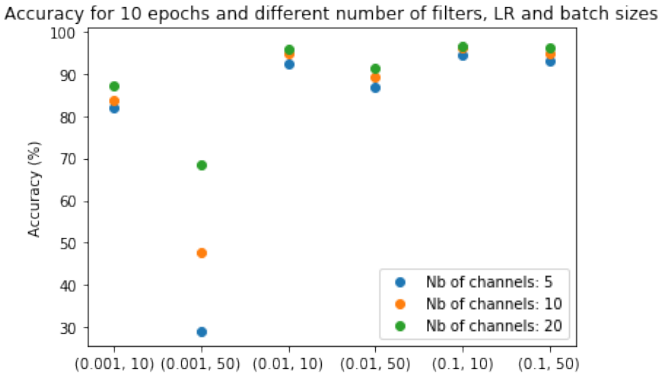
With the initial architecture (one linear layer and a softmax activation function), we got 88.02% of accuracy. We then tried:

- A single convolutionnal layer; to make it work we had to remove the reshaping in the *train* function, flatten the output of the convolutionnal layer and add a linear layer that would trasnform the outpute to the a 10-sized one for the softmax function to work. We tested with different parameters and obtained Figure 5 of the Appendix.
  Here the kernel size and output channels do not seem to have a high impact on the performances so we will stick to a kernel size of 3 and a small number of output channels from now on.

- 2 and 3 convolutionnal layers, with the same number (10) of output channels at each layer. We put a *MaxPool* between each convolutionnal layer to reduce the dimension and look at lower levels of details. We can see the convergences and accuracies for different activation functions (*softmax, sigmoid and tanh*) in the Figure 6 of the Appendix. It is interesting to note that the *softmax* always looks better for the convergence, but the best accuracy (95.09%) was actually reached with *tanh* and three layers. It suggests that the loss function is not appropriate.

1.6 Question 2: Impact of the optimizer

We kept a 3-layers network and tried to test it with different learning rates (keeping the SGD), batch sizes and number of output channels to see the impact on the convergence behaviour. The different convergences are in Figure 7 of the Appendix and a summary of the obtained accuracies is in the following figure:



The best accuracy reached was 96.47%, with 20 output channels, a learning rate of 0.1 and a batch size of 10. We can also notice in the convergences that the higher the number of channels, the faster it converges for a same learning rate/batch size.

Next we tried with 20 epochs and to modify the best previous model (3 layers, 20 channels, batch size 10) by testing different optimizers and learning rates since convergence behaviour might change between optimizers for a same LR. (See Figure 8 of the Appendix) The best results where obtained with SGD and a 0.1 learning rate (97.17%).

1.7 Question 3: Impact of the loss function

We retested the best model : 3 layers, 20 channnels per layer, tanh activation functions, SGD, batch size 10, learning rate 0.1, 20 epochs, and changed the loss to the Cross Entropy one. If we do not chnage anything, we get an exploding gradient error. When clipping the gradient to avoid that, the loss increases after 2 epochs. That issue was solved by switching to a smaller learning rate : we used a 0.025 learning rate with 50 epochs and obtained a 98.21% accuracy.

1.8 Question 4: Prediction on test set

We then tested the best previous model (cf. previous question), and got an accuracy of 98.39% on the previously never seen test dataset.

Exercise 3

1.9 Question 1: Impact of the architecture of the model

We originally tried different architectures with one convolution layer followed by a linear layer. This convolution layer could either be common to both paths (numbers and colors) or separate. The comparison between the two allowed us to see how much weight sharing would possibly be more interesting for the rest of the experiences.

As it turns out, the parameters were easily tuned in a way to allow for a 100% accuracy on the prediction of colors, whether or not the convolution layer was shared. Whether the convolutional layer was common or separate, the model yielded satisfying results (especially considering this was before any more hyper parameter tuning), see figure 9. This made it difficult to completely gather which type of architecture was best. However, because the prediction of numbers relies much more on the actual pattern within the images, as opposed to the uniform color, it made sense to us to keep them separate. Therefore, in the remainder of our experiences, all layers are separate between the two.

We then tried, for one separate convolution layer for the prediction of numbers and the prediction of colors, different values of neurons in the hidden layer, and the different activation layer. The exact optimal number of neurons in the hidden layer depended on the activation layer, but overall, the best activation layer was *Tanh*, see figure 10. This seems consistent with the fact that our model does not have many layers. At most, in our experiences, we will have three layers. For a deeper model, *ReLU* could be a more interesting activation layer, and could possibly yield better results.

1.10 Question 2: Impact of the optimizer

- **Batch size:** As explained in [Kes+16], a batch size is typically between 32 and 512 datapoints. When the batch size exceeds these values, a degradation in performances is witnessed due to its incapability of generalizing. As such, a batch size of 64, which is quite typical of classification problems on standard datasets like MNIST, was used.

- **Optimizer:** The ADAM optimizer is quite classic because it has relatively low memory requirements (though higher than gradient descent and gradient descent with momentum) and it usually works well even with little tuning of hyperparameters. An ADAM optimizer was therefore combined with a scheduler.

- **Learning rate:** The learning rate was initialized at 10^{-3} and decreases up to 10^{-5} using a cosine annealing schedule. This scheduler is a robust enough choice for most experiences.

1.11 Question 3: Impact of the loss function

An MSE loss is very interesting for a regression predicitive problem, but not quite as appropriate for a classification problem.
Instead, a Cross Entropy Loss, which is most commonly used in this type of problem, proved to be very effective and worked very well.
The optimizer and schedulers were optimized with this loss.

1.12 Question 4: Prediction on test set

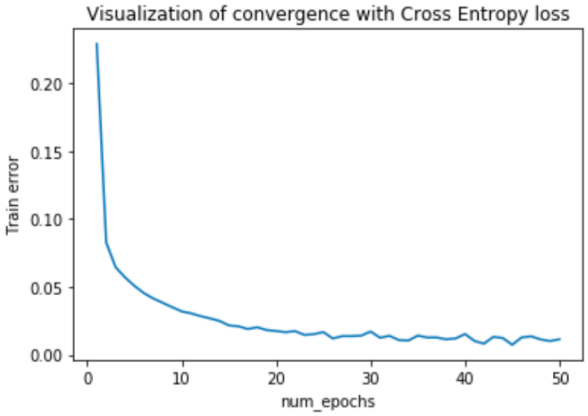
Our final model had three convolution layers, followed by a final linear layer. All four layers are specific to the prediction of numbers and color, and no layer is shared. Each convolution layer is followed by a *MaxPool* layer of stride 2 and a *ReLU* activation function. The three kernel sizes are $k = [5, 3, 3]$.

A *dropout* layer could have been added, but overfitting was clearly not a problem here since the accuracy on the test dataset was actually better than the accuracy on the validation dataset.

This model yielded an accuracy of

- Accuracy of the model for numbers : 99.13%
- Accuracy of the model for colors : 100.00%
- Accuracy of the model for both : 99.13%

on the previously never seen test dataset.



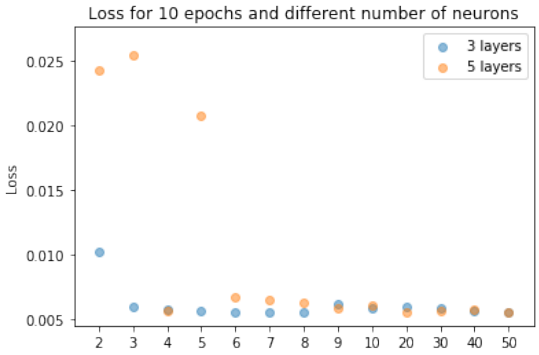
Exercise 4

1.13 Question 1: Impact of the architecture of the model

Before everything, we normalized the dataset. Without that pre-processing we have an exploding gradient due to the high values in the data.

We tested models of 1, 2, 3 and 5 layers with different activation functions (*sigmoid, tanh, ReLU*) and the same number of neurons at each layer: 5. *tanh* gave the minimum loss for all the models, so we will keep it from now on. The 3 layers model gave the best loss (0.005699), but the convergence of the 5-layers one was steeper (see Figure 11 of the Appendix). We will therefore keep trying on 3 and 5 layers model.

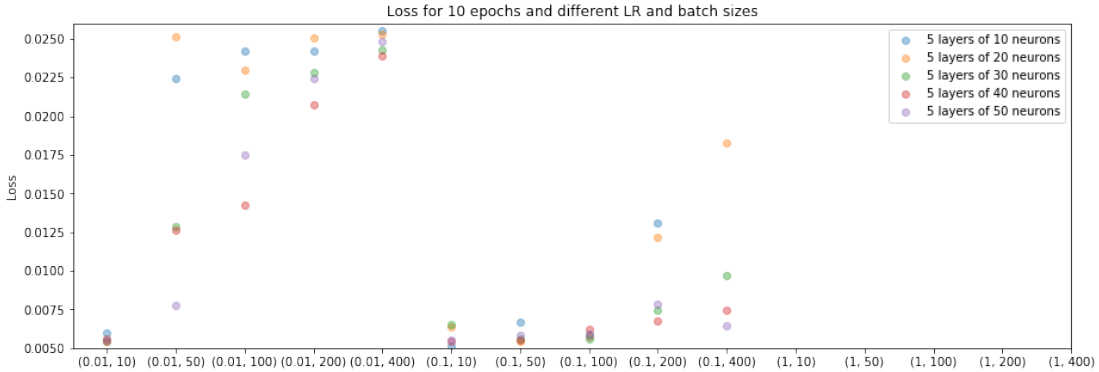
We then tried different number of neurons for the 3 and 5 layers models. We got the losses:



Generally, the higher the number of neurons, the better the loss; but the improvement is not important after 10 neurons. The 5-layers model could reach the best loss: 0.005490 with for 20 neurons at each layer. However, we could reach 0.005518 with a 3 layers model with only 6 neurons at each layer, which has far less parameters than the 5-layers version. In this case we will keep the deeper network since the computation is really fast, but for a different type of dataset we might have preferred the faster and simpler model given by the 3-layers net.

1.14 Question 2: Impact of the optimizer

We tested some 5-layers architectures with different learning rates and atch sizes and got the accuracies summarized below:



We got a *nan* error for learning rates of 1 and above. For all architectures, we see the general (and counter-intuitive) trend that the higher the batch size, the higher the loss.The best lost (0.005119) was obtained for 10 neurons per layer, a batch size of 10 and a learning rate of 0.1.

We then kept the best previous model with 20 epochs and tried to change the learning rate and optimizer (see Figure 12 of the Appendix). The best loss of 0.003820 was obtained with the RMSprop optimizer and a 0.005 learning rate. We notice that a 0.5 learning rate always gives bad performan-1.369851.ces or a *nan* error.

1.15 Question 3: Impact of the loss function

We switched to a Gaussian Loss based on the mean and variance of the prediction instead of the predicted value. To do that we had to modify the last layer of the model, create a custom loss function taking the mean and log-variance as arguments and change the accuracy function. If we do not change the learning rate (0.005), the loss goes back up after a while, therefore we decreased the LR to 0.001 and increased the number of epochs to 100 for it to converge. We obtained a loss of -1.570448.

1.16 Question 4: Prediction on test set

Testing the previous model on the test dataset, we obtained a Gaussian loss of -1.369851, which is in the same range of the loss on the validation set.

Appendix

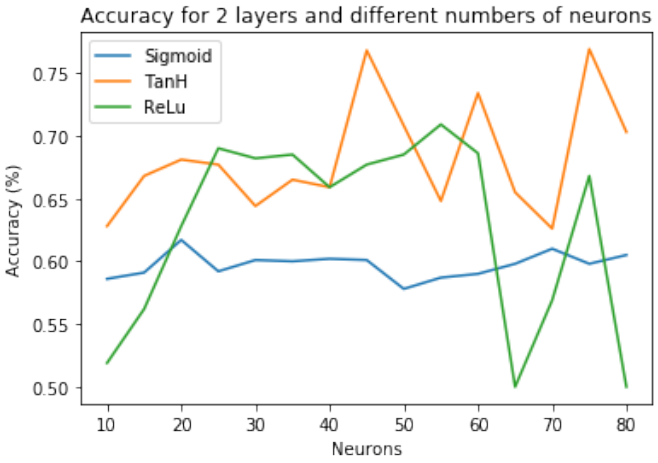


Figure 1

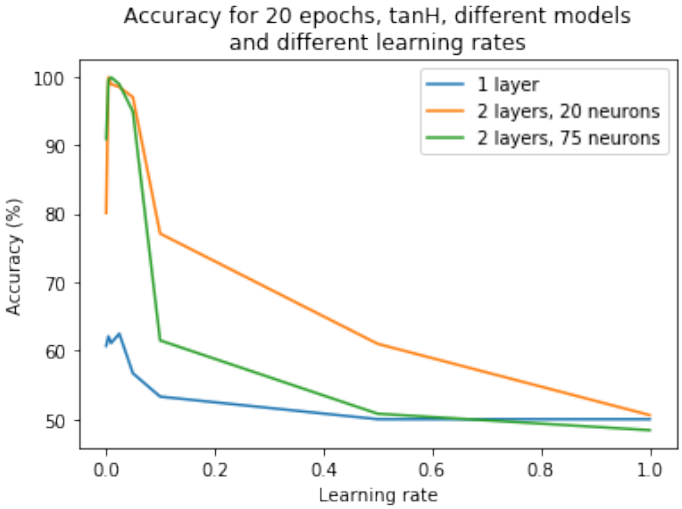


Figure 2

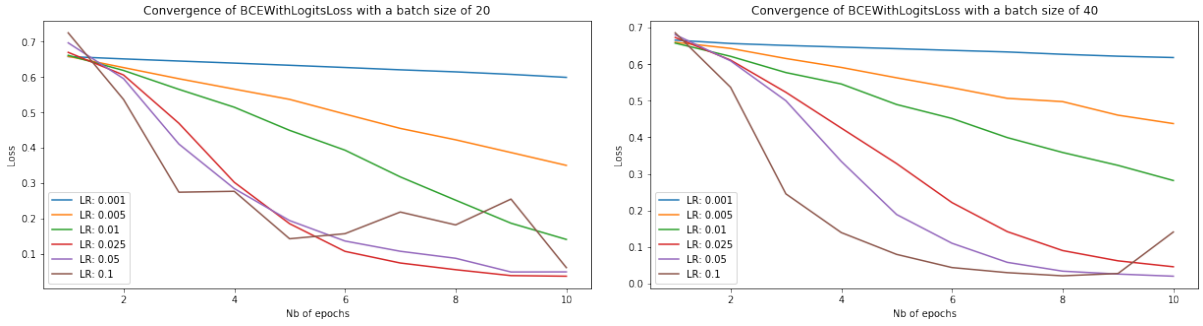


Figure 3

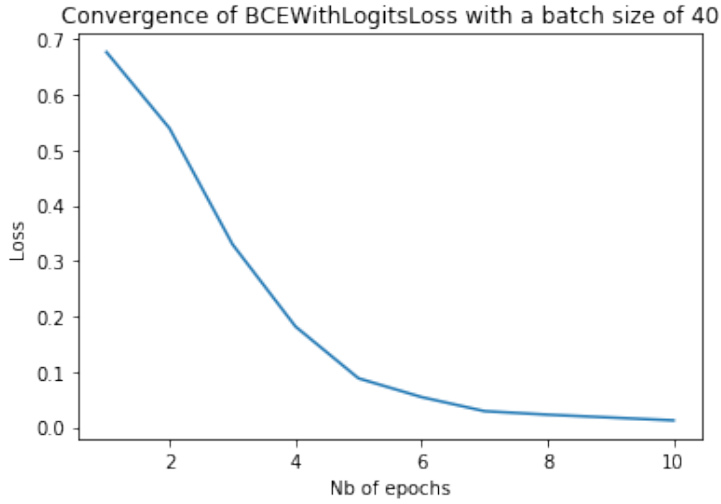


Figure 4

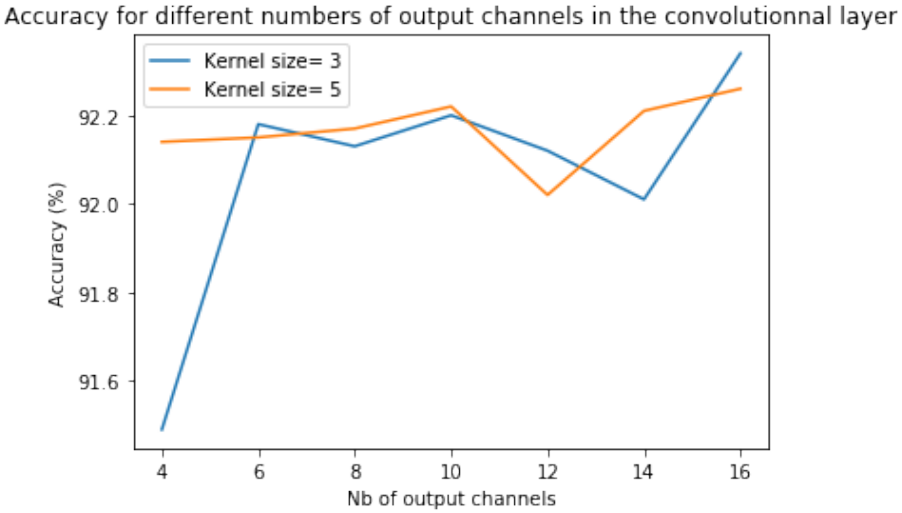


Figure 5

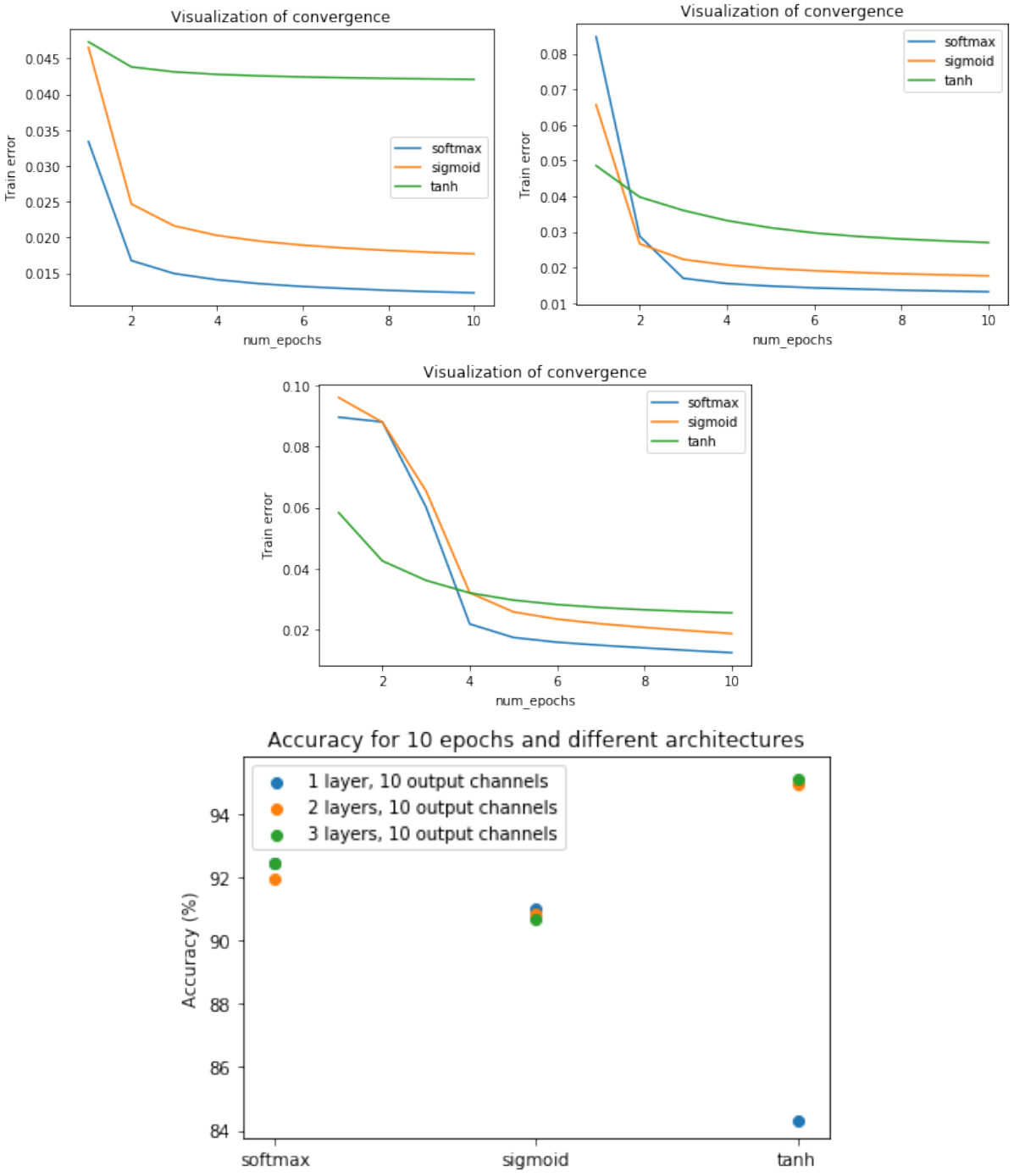


Figure 6: Convergences and accuracy for 1, 2 or 3 layers and different activation functions

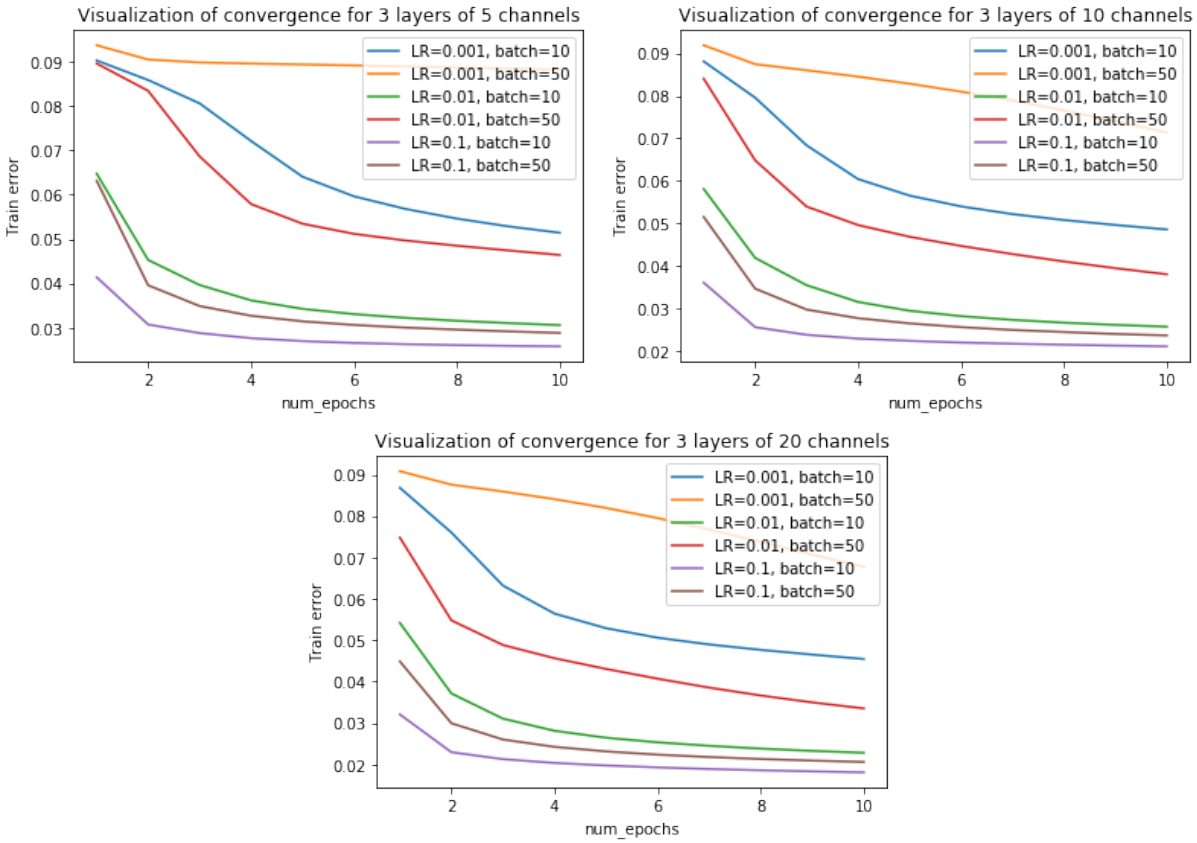


Figure 7: Convergences for different number of channels, batch sizes and learning rates

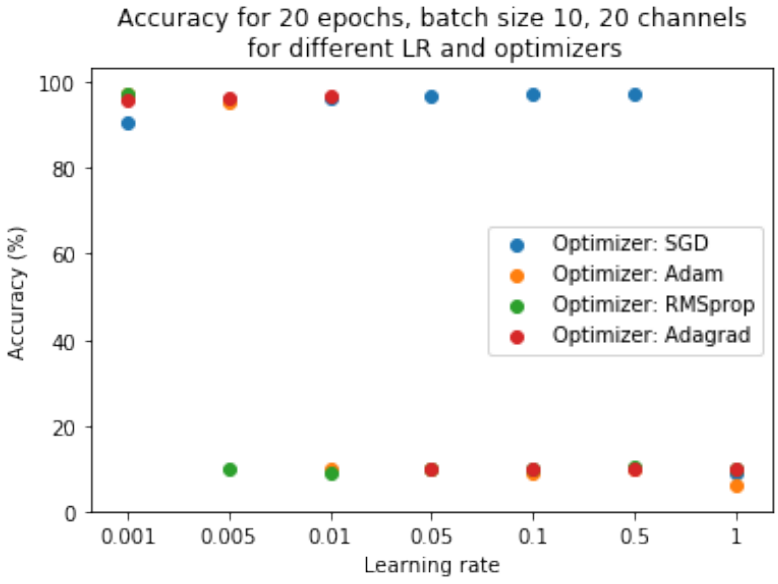


Figure 8: Accuracies for 20 epochs and different learning rates and optimizers

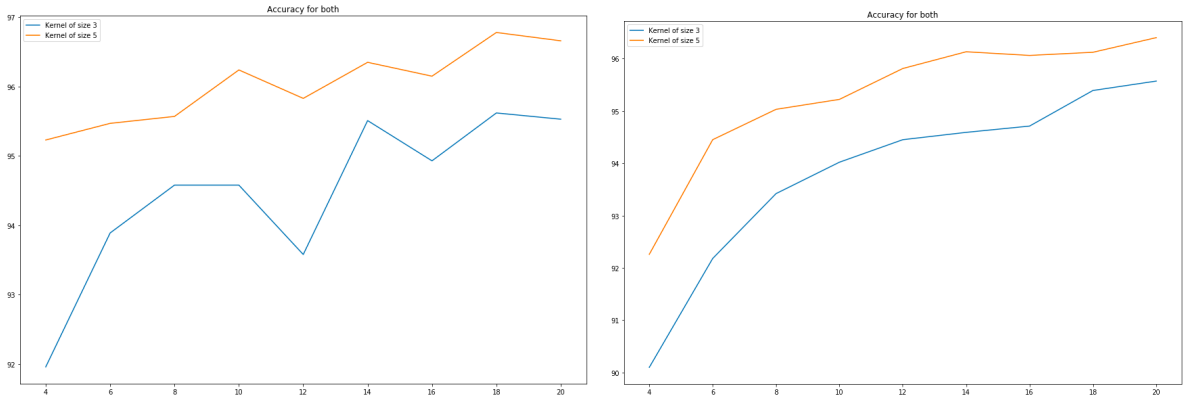


Figure 9: Accuracies for 1 separate *[left]* and 1 common *[right]* convolutional layer and different numbers of neurons

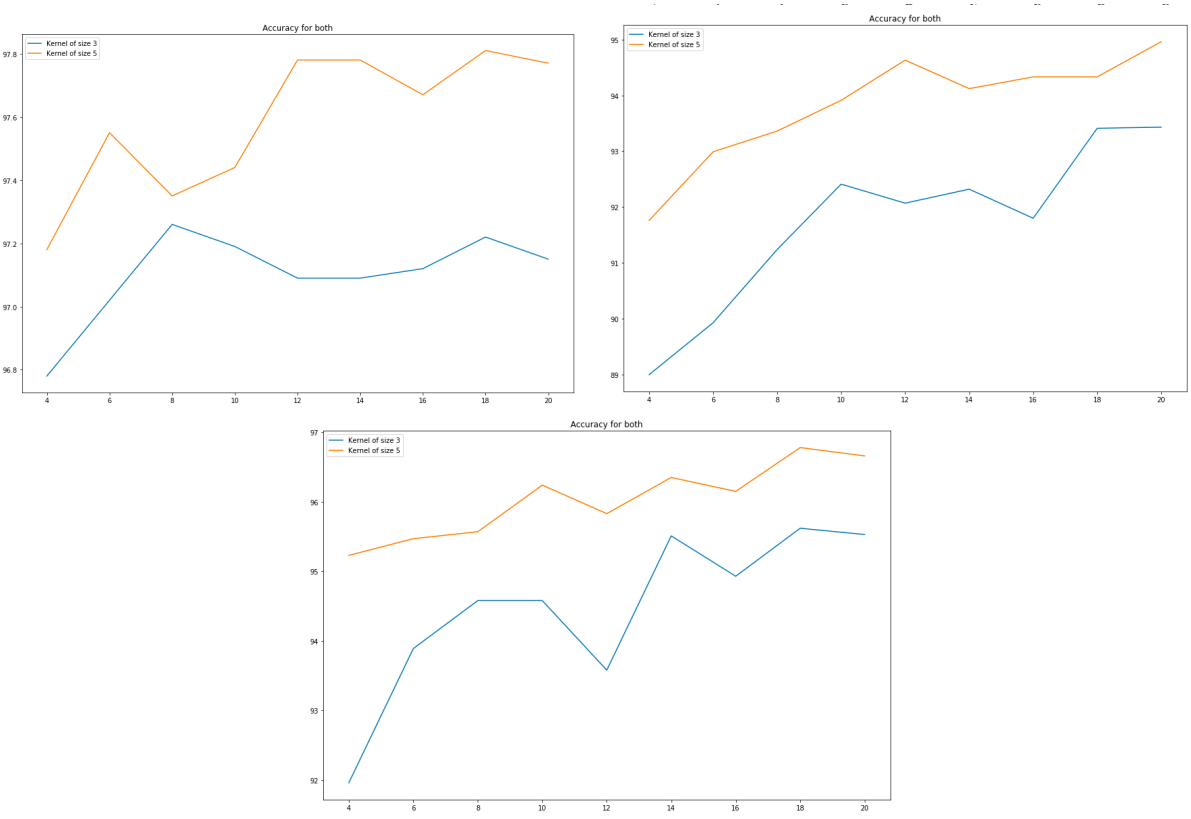


Figure 10: Accuracies for 1 separate convolutional layer and different activation layers: ReLU *[left]*, Sigmoid *[]center]* and Tanh *[right]*

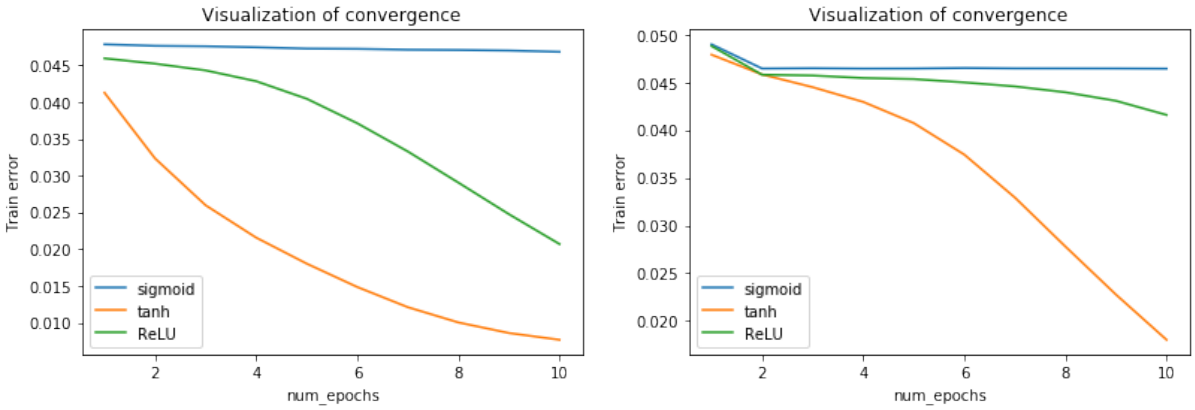


Figure 11: Convergences for a 3-layers (left) and a 5-layers (right) model

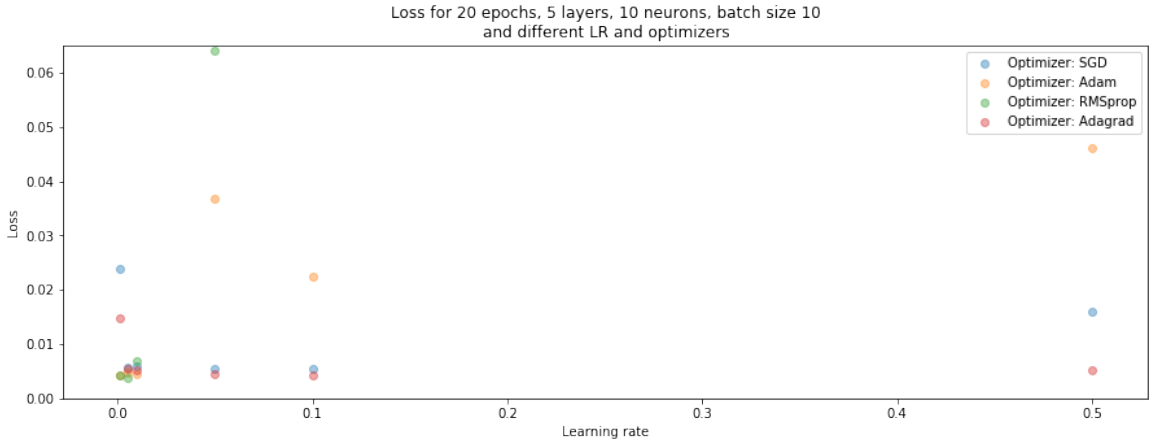


Figure 12: Loss for a 5-layers model for different optimizers and learning rates

References

[Kes+16] Nitish Shirish Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.