Data Science and Scientific Computing Track Core Course

Segment 2 Project

Marwan Elkrewi, Sreyam Sengupta

19-05-2021

Institute of Science and Technology, Klosterneuburg, Austria

1 Part 1

1.1 Introduction

If a loss function is convex, it has one unique minimum. Then optimization is relatively straightforward, and if a good algorithm is given enough iterations, it can reach this minimum upto a certain tolerance. However, in real-life applications loss functions can be far more complicated. In our example we use the MNIST database of handwritten letters [1]. The task is to train a network to classify the digits into ten classes (the digits 0-9 respectively).

The objective in this part was to understand mode connectivity [2], which is a very interesting phenomena. It suggests that simple paths on which the loss is almost constant connect the optima discovered by gradient based optimization.

1.2 Training the network

In this part we use a 2-layer fully-connected network. Each digit is stored in a 28×28 px image, which is flattened into a 1-D vector. The first layer of the network has shape $28^2 \times 500$, while the second is 500×10, because there are ten classes of output. We used Rectified Linear Unit for activation, and the loss function we have used is the cross-entropy loss. The training set has a 100 images, while the test set has 1000.

We trained two models for 14 epochs, each 600 iterations long, with a common learning rate $\alpha = 0.0001$. Both converged to a final accuracy of $\sim 98\%$. We used these two models as endpoints of our path. The idea of mode connectivity is that minima in such high-dimensional loss landscapes are not isolated, but rather connected by (possibly curved) paths over which the value of loss changes very little.

1.3 Path construction

In this section we tested the ideas discussed above. The two models we previously trained define the endpoints of our paths. We chose two possible ways to construct paths - a simple linear interpolation between the end points and a 2-segment path. If θ_1 and θ_2 are the parameters that define the endpoints, then the linear interpolation is:

$$\pi_{\text{linear}}(t) = (1-t)\theta_1 + t\theta_2, \quad t \in [0,1] \ . \tag{1}$$

For the 2-segment method, we need to train another model as a midpoint with parameters θ:

$$\pi(t) = \begin{cases} 2((0.5-t)\theta_1 + t\theta), & t \in [0, 0.5] \\ 2((1-t)\theta + (t-0.5)\theta_2), & t \in [0.5, 1] \ . \end{cases} \tag{2}$$

All models trained in this part achieved an accuracy of $\sim 98\%$. We compare the test losses of both the above functions for 50 timesteps in $[0, 1]$. The results are plotted in figure (1).

1.4 Bonus

We compared a third path: the Bezier curve. This is a quadratic interpolation defined as:

$$\pi(t) = (1-t^2)\theta_1 + t^2\theta_2 + 2t(1-t)\theta, \quad t \in [0,1] \tag{3}$$

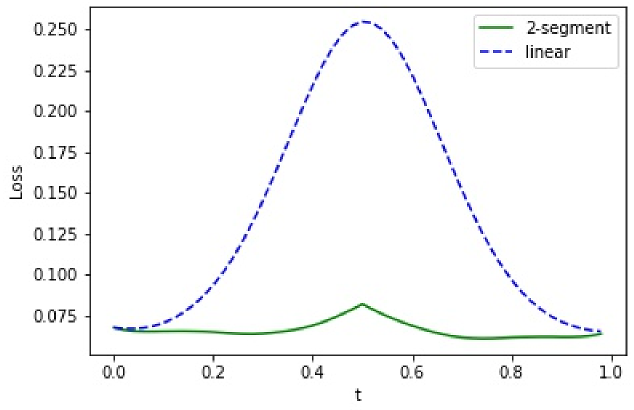

Figure 1: (Part 1) The value of loss along the path for the two methods. We clearly see the 2-segment method finds a better path, i.e. one over which the loss is much less.

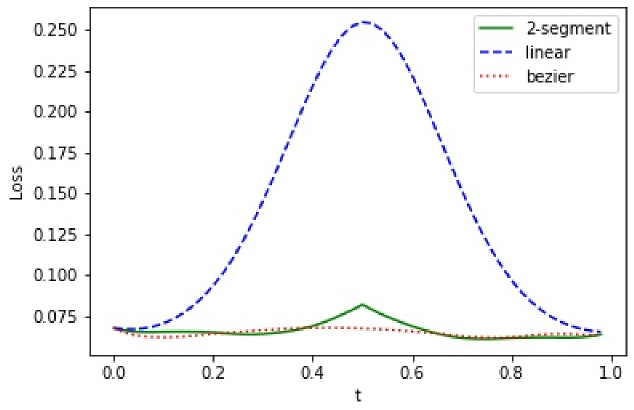

Figure 2: (Part 1) Same as above, but also including the Bezier path now. We see that the quadratic Bezier does slightly better than the piecewise linear 2-segment method, and both are much superior to the simple linear path.

and the model to define θ achieved accuracy $> 98\%$. We compared the Bezier method to the above two methods; results are plotted in figure (2).

2 Part 2

2.1 Introduction

In this part we learned about dropout stability. When the number of neurons N in our network is sufficiently large, dropping half the connections in the network (and then rescaling the remaining outputs) causes very small change in the loss function. We used a 2-layer network as in the previous part. Our network looks like this:

$$y_{\theta_N}(x) = \frac{1}{N}\sum_{i=1}^{N} a_i \sigma(w_i^T x + b_i), \tag{4}$$

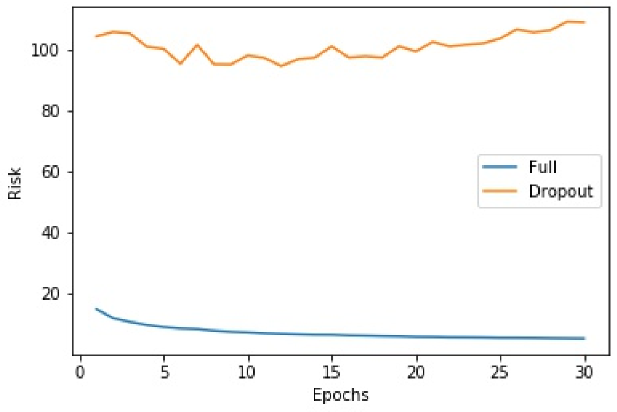

Figure 3: (Part 2) The test loss (also known as population risk) as a function of the epoch, both for the full $N = 50$ as well as for the dropout $N_{1/2} = 25$ networks. We see that the dropout does not catch up to the full, at least within our timeframe.

where w_i and b_i are the weights and biases of the first layer respectively, x is our datavector, σ is the ReLU function, and a_i are the weights of the second layer.

First, we trained two models with different values of N (50 and 2000 in our case) on the same MNIST dataset. The point was to see that dropout stability is only achieved for N sufficiently large. The parameter values a_i and w_i were drawn from the distributions given for our group which were:

$$a \sim 0.5 \cdot \mathcal{U}[0.5, 1.5] + 0.5 \cdot \mathcal{U}[-1.5, -0.5] \tag{5}$$

$$w \sim \mathcal{N}[0, 1.2] \tag{6}$$

2.2 Dropout stability

First, we train the $N = 50$ network on the dataset. Using 30 epochs and $\alpha = 0.001$, we achieved accuracy $\sim 95\%$ for the full network. However, the dropout managed only $\sim 55\%$. Two indicators of network performance are loss and error (defined as one minus the accuracy); for the $N = 50$ network, these are plotted in figures (3) and (4) respectively. We see that overall, the dropout performs much poorly compared to the full network.

Now we used $N = 2000$ and repeated the above procedure. Unfortunately, we did not rescale the number of epochs or α due to time constraints, and used the values above. Even then, the full network achieved an accuracy $\sim 98\%$. The dropout performed noticeably better than the previous case, achieving accuracy $\sim 96\%$. The loss/risk and error as a function of epoch are plotted in figures (5) and (6). Here, the performance of the dropout (while less than the full) is much better than the earlier case with fewer neurons.

2.3 Path construction

Here, as before, we defined a piecewise linear path between two optima for the $N = 2000$ case. One model we have already, from the above section. We trained another model also for $N = 2000$ and all other parameters same, this time achieving accuracy $\sim 98\%$. The parameters of these two models define our endpoints θ and θ'.

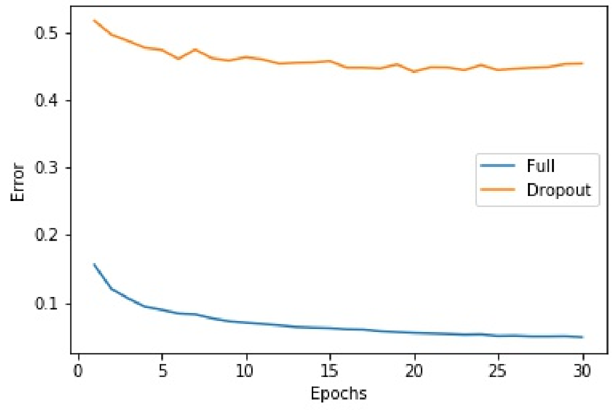

Figure 4: (Part 2) The error as a function of epoch, again for both full and dropout cases. Here again, we see the dropout performs much worse compared to the full network.

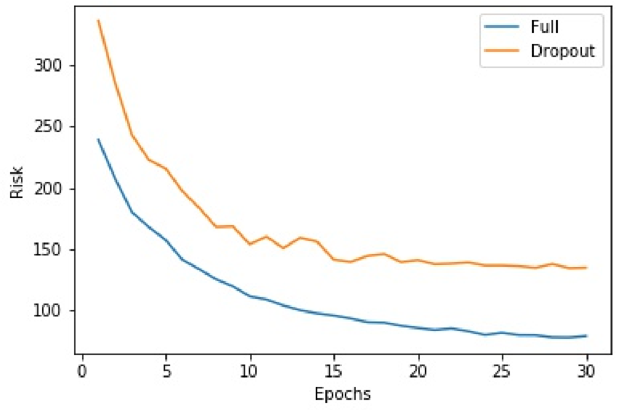

Figure 5: (Part 2) The test loss/risk as a function of the epoch, for both full $N = 2000$ and dropout $N_{1/2} = 1000$ cases. Here we see dropout performs much better than $N = 50$ case, which we expect.

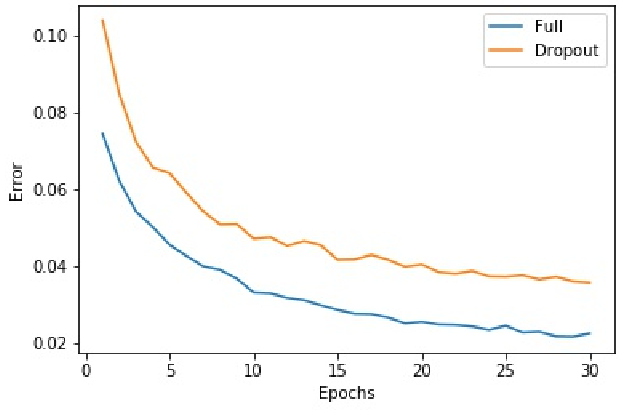

Figure 6: (Part 2) The error as a function of epoch, for both full and dropout cases. Like the risk, the error of the dropout is also not too far from that of the full network.

Using these endpoints, we define a 7-segment piecewise linear path between θ and θ' (details in notebook). To compare, we also did a simple linear interpolation as in the last part. The results are plotted in figure (7). The result was unexpected - we expected the segmented path to perform better, as in the previous part of the project. However, it actually performs worse than the linear path.

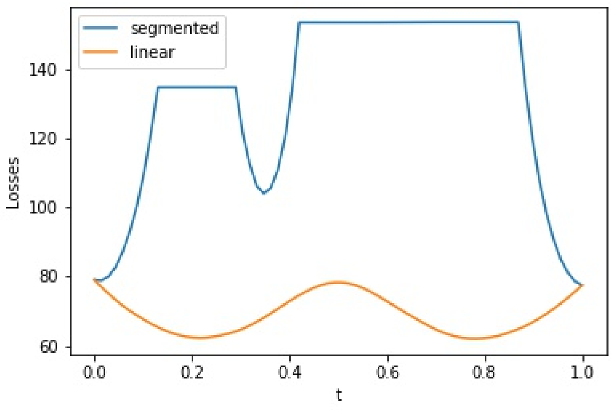

Figure 7: (Part 2) The losses along our paths, both for the segmented or piecewise linear path, as well as for the simple linear path. Somewhat surprisingly, we see the naïve linear path performs better.

2.4 Issues

- Our N values were 50 and 2000 respectively. We knew that for the larger network, we would have to scale down the learning rate α by a factor of 40 ($= 2000/50$), while scaling up the number of epochs by the same number. It was not clear to us the dependence of optimization time on α, but each epoch takes roughly the same amount of time, and we simply did not have enough time to run $30 \times 40 = 1200$ epochs.
- The segmented path was expected to perform better than the linear, but it actually performed worse, which we cannot explain.

References

[1] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[2] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.