UNIVERSITÉ CÔTE D'AZUR

MSc. Data Science & Artificial Intelligence

Advanced Deep Learning

Dr Alessandro BETTI

Assignment 1 & 2

Author: Joris LIMONIER
joris.limonier@gmail.com
Due: November 4, 2022

Contents

List of Figures

1 Assignment 1

1.1 Exercise 1

The code was completed in the Python file.

1.2 Exercise 2

1.2.1 Question 1

I was not familiar with the PyTorch library, so I had to perform some research in order to know how its methods/classes work. I consulted several sources, including the following, which were useful in order to answer this question:

1. https://discuss.pytorch.org/t/how-sgd-works-in-pytorch/8060/2
2. https://discuss.pytorch.org/t/performing-mini-batch-gradient-descent-or-stochastic-gradient-descent-on-a-mini-batch/21235
3. https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

According to those sources, PyTorch's SGD actually computes a full-batch (vanilla) Gradient Descent, based on the data that is passed to it. It is my understanding that in order to perform actual mini-batch (*i.e.* where 1 < batch size < number of observations), one simply needs to give subsets of the data a each iteration.
In our case, we use the full dataset in `outputs = net(inputs)`, which is why we perform full-batch GD, although we call the `optim.SGD` class.

1.2.2 Question 2

Figures 1, 2 and 3 show a comparison between train and test cross entropy loss. The leftmost diagrams have a linear scale for the vertical axis, while the rightmost diagrams have a logarithmic scale for the vertical axis. Beware that the number of epochs varies significantly between figures. Indeed, more pararmeters in the network implies longer training times per epoch, which in turn means that running for the same time both experiments results in different number of epochs.

We see that in both experiments the train error keeps decreasing, while the test error

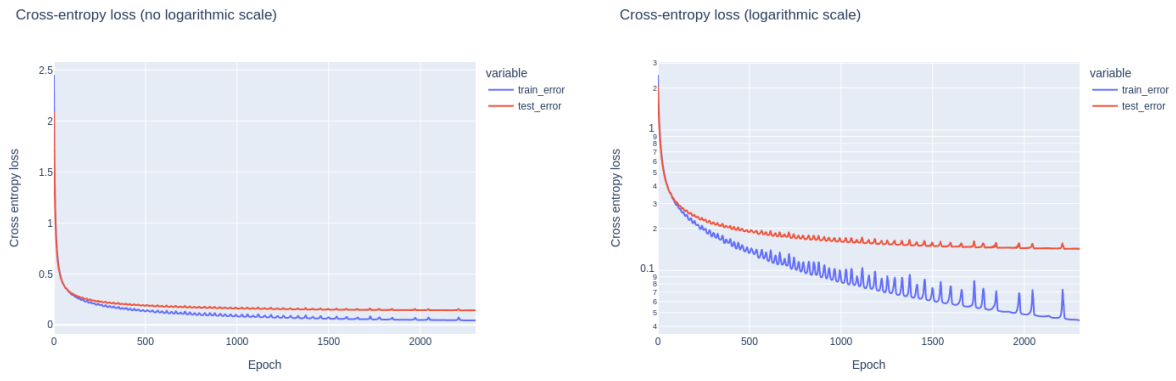

Figure 1: Cross entropy loss, train vs test (300 fully-connected).

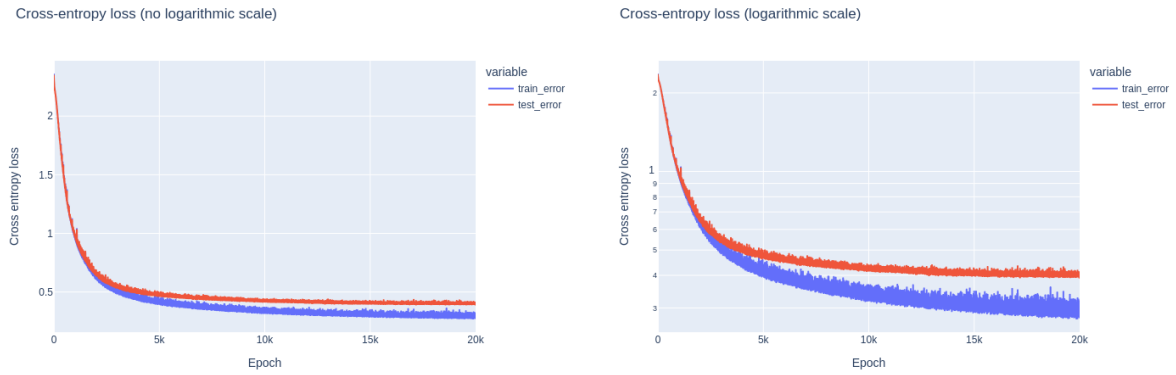

Figure 2: Cross entropy loss, train vs test ($10 - 10$ fully-connected).

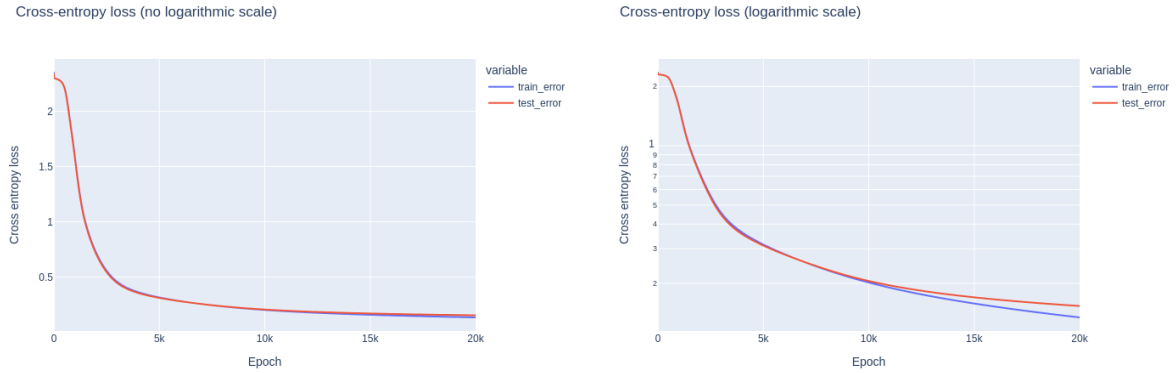

Figure 3: Cross entropy loss, train vs test ($20 - 20$ fully-connected).

stays constant. This means that our network is overfitting and there is no need to continue the experiment further. One way to prevent wasting time training when the network isn't making actual progress is to print the loss, or even to plot the loss curves, every couple iterations. We estimate however that the over-fitting phenomenon isn't too extreme in our case.

1.2.3 Question 3

Let $\beta : \mathbb{N}^{l+1} \to \mathbb{N}$ denote the number of parameters in a fully connected neural network. Then β is given by:

$$\beta(h_0, \ldots, h_l) := \sum_{i=1}^{l} (h_{i-1} * h_i) + h_i \tag{1}$$

Indeed, the multiplication term "$h_{i-1} * h_i$" accounts for the weights by counting the connections from a layer to the next. The last term "h_i" is responsible for counting the biases on each of the hidden layers, as well as the output layer.
We compute β for a few network architectures we used, with $h_0 = 28 * 28 = 784$ in all

cases, since this is the size of the input:

$$\begin{aligned}\beta(784, 10, 10, 10) &= (784 * 10 + 10) + (10 * 10 + 10) + (10 * 10 + 10) \\ &= 7850 + 110 + 110 \\ &= 8070\end{aligned}$$

$$\begin{aligned}\beta(784, 20, 20, 10) &= (784 * 20 + 20) + (20 * 20 + 20) + (20 * 10 + 10) \\ &= 15700 + 420 + 210 \\ &= 16330\end{aligned}$$

$$\begin{aligned}\beta(784, 300, 10) &= (784 * 300 + 300) + (300 * 10 + 10) \\ &= 235500 + 3010 \\ &= 238510\end{aligned}$$

However, we know that not all parameters are created equal. Indeed, adding more layers includes non-linearities in the network. Increasing the number of layers is useful because it "bends" the space in which the data lives, which allows to find more patterns. On the other hand, adding layers causes other issues, namely the vanishing gradient problem: as the gradient goes through multiple activation functions, it shrinks until making close to no update in the weight. The argument for less layers is also strengthened by Cybenko's *Universal approximation theorem* (1989), which proves that a unique hidden layer is sufficient to approximate any *reasonable* function with arbitrary precision. However, it may require the hidden layer to be of exponential size with respect to the desired precision.
The bottom line to this reflexion is as follows: increasing the size of the layers is good because the network can learn more, however there are risks of overfitting. Increasing the number of layers is also an option, but it can cause overfitting and gradients may vanish. Most certainly some trial and error is required, with close baby-sitting of training and validation errors in order to adapt the architecture.

1.2.4 Question 4

1.2.5 Question 5

Using a single output means that the network give us one definitive answer and we have no idea of how certain it is. One hot encoding on the other hand provides probabilities of how sure it is when it makes predictions. This is useful because a network should be penalized equally for different degrees of certainty. Ideally, we want the network to be certain of its prediction and the prediction to be right. If the network is certain of a wrong prediction, we should penalize it a lot. We would prefer a network that makes a wrong prediction but is unsure. This latter network would take less weights adjustments in order to make the prediction right.

1.2.6 Question 6

For this question, we will keep the architecture of a single 300-neurons hidden layer and we will monitor the loss with various learning rates.
Let us call the learning rate η. We already tested the case $\eta = 0.1$ in figure 1.

1.2.7 Question 7

2 Assignment 2