Assignment 1. MLPs, CNNs and Backpropagation

Ruben Gerritse
10760326
rgerritse95@gmail.com

Section 1.1 a)

- $\frac{\partial L}{\partial x_i^{(N)}} \text{exp} = \frac{\partial}{\partial x_i^{(N)}} - \sum_i t_i \log x_i^{(N)} = -\frac{t_i}{x_i^{(N)}}$

- $\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} = \frac{\partial}{\partial \tilde{x}_j^{(N)}} \frac{\exp(\tilde{x}_i^{(N)})}{\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)})}$

  $= \mathbb{1}[i=j] \frac{\exp(\tilde{x}_i^{(N)}) \sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}) - \exp(\tilde{x}_i^{(N)})\exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}))^2} + \mathbb{1}[i \neq j] \frac{-\exp(\tilde{x}_i^{(N)})\exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}))^2}$

  $= \mathbb{1}[i=j] \frac{\exp(\tilde{x}_i^{(N)}) \sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)})}{(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}))^2} - \frac{\exp(\tilde{x}_i(N))\exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}))^2} + \mathbb{1}[i \neq j] \frac{-\exp(\tilde{x}_i^{(N)})\exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}))^2}$

  $= \mathbb{1}[i=j] \frac{\exp(\tilde{x}_i^{(N)})}{\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)})} - \frac{\exp(\tilde{x}_i^{(N)})\exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}))^2} + \mathbb{1}[i \neq j] \frac{-\exp(\tilde{x}_i^{(N)})\exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}))^2}$

  $= \mathbb{1}[i=j] x_i^{(N)} - x_i^{(N)} x_j^{(N)} - \mathbb{1}[i \neq j] x_i^{(N)} x_j^{(N)}$

  $= \mathbb{1}[i=j][x_i^{(N)}] - x_i^{(N)} x_j^{(N)}$

  Matrixc form:

  $= Diag(x^{(N)}) - x^{(N)}(x^{(N)})^T$

- $\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} = \mathbb{1}[\tilde{x}^{(l<N)} > 0]$

- $\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = \frac{\partial}{\partial x^{(l-1)}} W x^{(l-1)} + b = W$

- $\frac{\partial \tilde{x}^{(l)}}{\partial W} = W x^{(l-1)} + b = (x^{(l-1)})^T$

- $\frac{\partial \tilde{x}^{(l)}}{\partial b} = W x^{(l-1)} + b = 1$

Section 1.1 b)

- $\frac{\partial L}{\partial \tilde{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}} \circ Diag(x^{(N)}) - x^{(N)}(x^{(N)})^T$

- $\frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} = \frac{\partial L}{\partial x^{(l)}} \circ \mathbb{1}[\tilde{x}^{(l)} > 0]$

- $\frac{\partial L}{\partial x^{(l<N)}} = \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}} = (\frac{\partial L}{\partial \tilde{x}^{(l+1)}} W)^T$

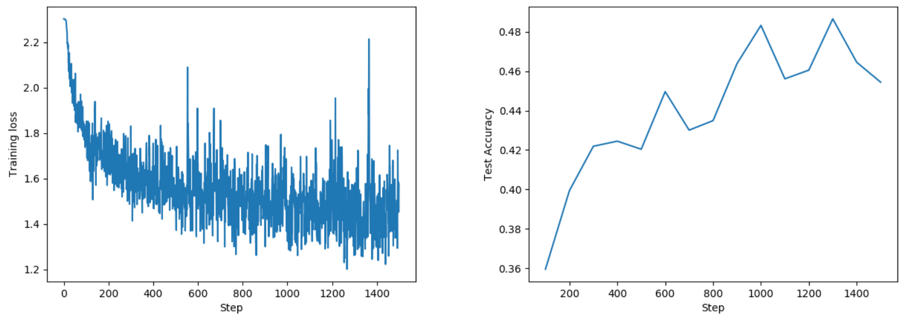
- $\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{l}}{\partial W^{(l)}} = (\frac{\partial L}{\partial \tilde{x}^{(l)}})^T (x^{(l)})^T$

- $\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{l}}{\partial b^{(l)}} = (\frac{\partial L}{\partial \tilde{x}^{(l)}})^T$

Section 1.1 c)

If a batchsize of $B \neq 1$ is used, then the derived backpropagation's equations will change by calculating the mean gradients of the samples in the batch and updated the weights by those mean gradients instead of updating them for the individual gradients per sample.

Section 1.2



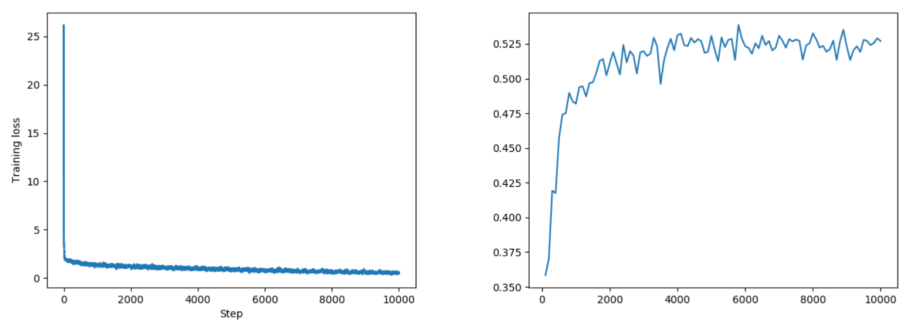
(a) Training loss of the NumpP MLP

(b) Test accuracy of the NumPy MLP

Figure 1: Training loss (left) and test accuracy (right) of the MLP implemented with NumPy

Figures 1 shows the training loss and the test accuracy during training process of the NumPy MLP using the default values of parameters. As shown in the Figure 1a the loss is not decreasing in a smooth manner, instead it is going up an down after each step. This is due to the fact that it is training on batches. The final achieved accuracy is 0.45, however on step 1300 an accuracy of 0.49 was achieved.

Section 2



(a) Training loss of the PyTorch MLP

(b) Test accuracy of the PyTorch MLP

Figure 2: Training loss (left) and test accuracy (right) of the MLP implemented with PyTorch

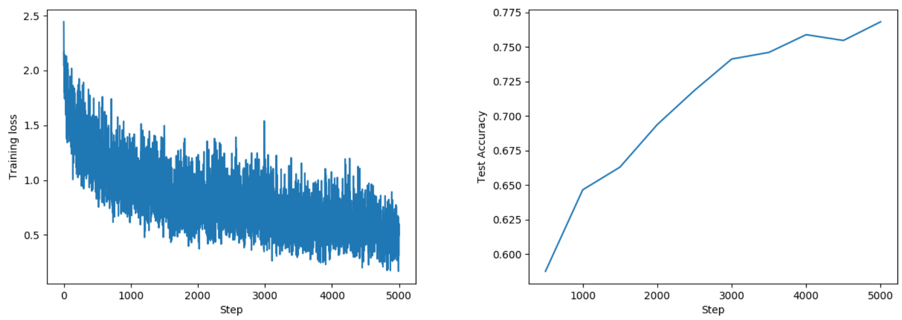
Figure 2 shows the training loss and the test accuracy during training process of the PyTorch MLP. The model consistes of 3 hidden layers each consisting of 700 hidden nodes as the default model was not expressive enough due to model the data. Furthermore, the model was trained for 10000 steps as the loss was still decreasing and the accuracy was still increasing after the default 1500 steps. Also, the learning rate is changed to 1e-3 to be able converge lower. Also the Adam optimizer is used instead of the SGD optimizer as it uses adaptive learning rates and there the model will converge better in local optima. Similar to the NumPy implementation, figure **??** shows that loss is not decreasing in a smooth manner, and also goes up an down after each step. Figure 2b shows the test accuracy each 100 steps. At the end of the training, the model converges around 0.52-0.53, however the highest achieved accuracy is actually 0.54 at step 8900.

Section 3.2 a)

Section 3.2 b)

Section 3.2 c)

Section 4



(a) Training loss of the CNN

(b) Test accuracy of the CNN

Figure 3: Training loss (left) and test accuracy (right) of the CNN

Figure 3 shows the training loss and the test accuracy during training process of the PyTorch CNN. Once again, the loss is not decreasing in a smooth curve as shown in figure 3a. Figure 3b shows the test accuracy each 100 steps. At the end of the training, the model achieved an accuracy of 0.76.