Deep Learning assignment 1

Jier Nzuanzu
10223258
jier.nzuanzu@student.uva.nl

1 MLP backprop and Numpy implementation

1.1 Analytical derivation of gradients

- a

$$\begin{aligned}
\frac{\partial L}{\partial x^N} &= \left(\frac{\partial L}{\partial x^N}\right)_i = \left(\frac{\partial L}{\partial x_i^N}\right) \\
&\Rightarrow \frac{-t_i}{x_i^N} \\
&\equiv -\left[\frac{t_0}{x_0^N} \cdots \frac{t_{dN}}{x_{dN}^N}\right] \\
\frac{\partial x^N}{\partial \tilde{x}^N} &= \left(\frac{\partial x^N}{\partial \tilde{x}^N}\right)_{ij} = \frac{\partial x_i^N}{\partial \tilde{x}_j^N} \\
&:= \frac{\partial}{\partial \tilde{x}_j^N} \frac{\exp \tilde{x}_i^N}{\sum_k^{dN} \exp(\tilde{x}^N)_k} \\
\Rightarrow i = j & \\
&\equiv \frac{\sum_k^{dN} \exp(\tilde{x}^N)_k \exp \tilde{x}_i^N - \exp \tilde{x}_i^N \exp \tilde{x}_j^N}{\left(\sum_k^{dN} \exp(\tilde{x}^N)_k\right)^2} \\
&\equiv \frac{\exp \tilde{x}_i^N}{\sum_k^{dN} \exp(\tilde{x}^N)_k} \left(1 - \frac{\exp \tilde{x}_j^N}{\sum_k^{dN} \exp(\tilde{x}^N)_k}\right) \\
&\equiv x_i^N (1 - x_i^N) \\
\Rightarrow i \neq j & \\
&\equiv \frac{\sum_k^{dN} \exp(\tilde{x}^N)_k * 0 - \exp \tilde{x}_i^N \exp \tilde{x}_j^N}{\left(\sum_k^{dN} \exp(\tilde{x}^N)_k\right)^2} \\
&\equiv \frac{-\exp \tilde{x}_i^N \exp \tilde{x}_j^N}{\left(\sum_k^{dN} \exp(\tilde{x}^N)_k\right)^2} \\
&\equiv -x_i^N x_j^N
\end{aligned}$$

Combining both cases we obtain

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

$$\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} = \begin{cases} 1_{[\tilde{x}_i > 0]} + \mathrm{a} * 1_{[\tilde{x}_i \leq 0]} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}.$$

Preprint. Work in progress.

$$\frac{\partial \tilde{x}^l}{\partial x^{(l-1)}} = \frac{\partial}{\partial x^{(l-1)}}\left(W^l \cdot x^{(l-1)} + b^l\right) = W^l$$
$$\frac{\partial \tilde{x}^l}{\partial W^l} = \left(\frac{\partial \tilde{x}^l}{\partial W^l}\right)_{ijk} = \frac{\partial \tilde{x}_i^l}{\partial W_{jk}^{(l)}}\left(W_{jk}^l \cdot x_i^{(l-1)} + b^l\right) = \left(x_i^{(l-1)}\right)_{jk}^T$$
$$\frac{\partial \tilde{x}^l}{\partial b^l} = \frac{\partial}{\partial b^{(l)}}\left(W^l \cdot x^{(l-1)} + b^l\right) = \mathcal{I}$$

- b

$$\frac{\partial L}{\partial \tilde{x}^N} = \frac{\partial L}{\partial x^N}\frac{\partial x^N}{\partial \tilde{x}^N}$$
$$\equiv \frac{\partial L}{\partial x^N} \odot X^N - X^N \cdot (X^N)^T \cdot \frac{\partial L}{\partial x^N}, \qquad \odot, \text{ element-wise multiplication}$$

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^l}\frac{\partial x^l}{\partial \tilde{x}^N}$$
$$\equiv \frac{\partial L}{\partial x^l}\begin{cases} 1_{[\tilde{x}_i>0]} + \text{a} * 1_{[\tilde{x}_i \leq 0]} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

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

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

$$\frac{\partial L}{\partial b^l} = \frac{\partial L}{\partial \tilde{x}^l}\frac{\partial \tilde{x}^l}{\partial b^l}$$
$$\equiv \frac{\partial L}{\partial \tilde{x}^l}\mathcal{I}$$
$$= \frac{\partial L}{\partial \tilde{x}^l}$$

- c

When $B = 1$ every datapoint is evaluated in the forward and backward propgagation. Evaluating at $B \neg 1$ means that input comes per batch and this leads to an extra dimension from the input space in the forward. Every batch is stacked in the column space and the value are evaluated per row during backpropagation, which at the input should be broadcasted again as the dimension of the input. This is what changes.

1.2 NumPy implementation

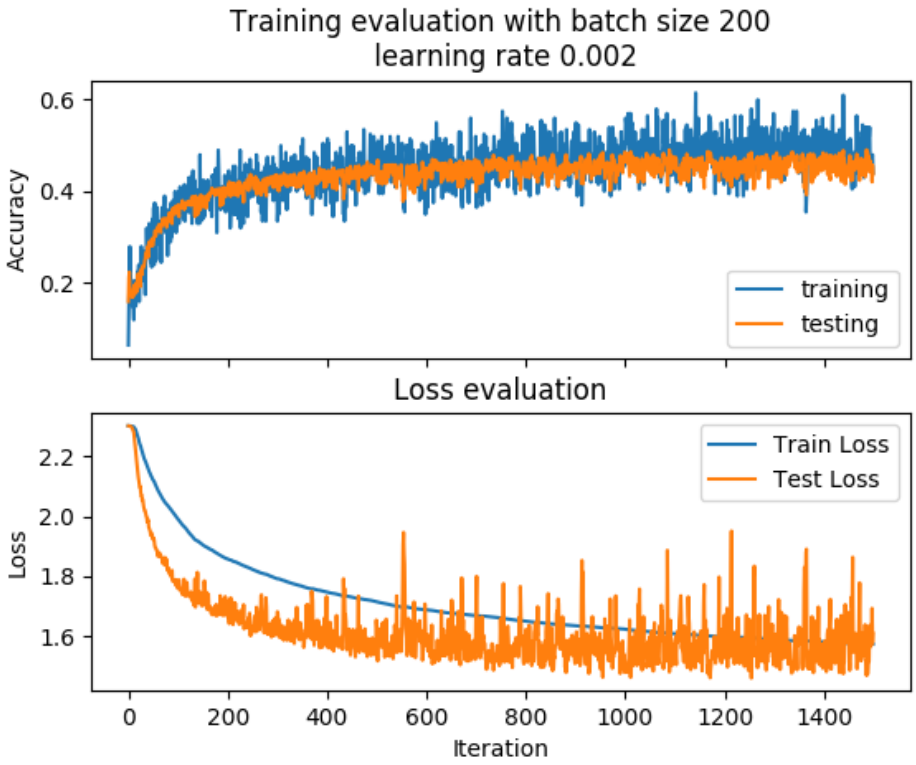

Figure 1: Implementation of deep neural network with Numpy with highest test accuracy of 0.46 for 1500 iteration, evaluated at each iteration

We see in Figure 1.2 that during training the train set keeps the high accuracy while the test set does not, conversely we see a slow convergence of the loss on the train set rather than the test set. This evaluation on each iteration show much variance on the test loss than its train counterpart. A bigger evaluation step might solve the variance for the loss and accuracy evaluation.

2 PyTorch MLP

2.1 Grid Search MLP PyTorch

Table 1: Grid search with wider hidden layers, using LeakyRelu activation function and neg slope of 0.02

Accuracy	Hidden	Learning rate	Max step	Batch size	Optimizer
0.526	500,800,800,500,100	0.002	1500	300	Adam
0.537	500,800,800,500,100	0.002	1500	300	Adamax
0.539	500,800,800,500,100	0.002	1500	300	Adagrad

Table 2: Grid search with deeper hidden layers, using LeakyRelu activation function and neg slope of 0.02

Accuracy	Hidden	Learning rate	Max step	Batch size	Optimizer
0.454	300,300,400,700,500,700,400,300,300	0.002	1500	300	Adam
0.539	300,300,400,700,500,700,400,300,300	0.002	1500	300	Adamax
0.527	500,800,800,500,100	0.002	1500	300	Adagrad

Table 3: Grid search with deeper hidden layers, using LeakyRelu activation function and neg slope of 0.02 and smaller batch size

Accuracy	Hidden	Learning rate	Max step	Batch size	Optimizer
0.264	300,300,400,700,500,700,400,300,300	0.002	1500	150	Adam
0.508	300,300,400,700,500,700,400,300,300	0.002	1500	150	Adamax
0.525	500,800,800,500,100	0.002	1500	150	Adagrad

As we can see from Table 1 wider hidden layers reproduce a overall better result during test on the three different opitmizer, while Table 2 does not. This surprisingly was not expected, the only variable that had been modified were the structure of the hidden layers. Morover, having a lower batch size, see Table 3 has not yield better test results over any optimizer, it scored less than the *NUMPY* implementation. However, more could have been done with respect to the activation functions variant of a **LEAKYRELU** as for example: **PRELU, ELU, SELU, SELU6**.

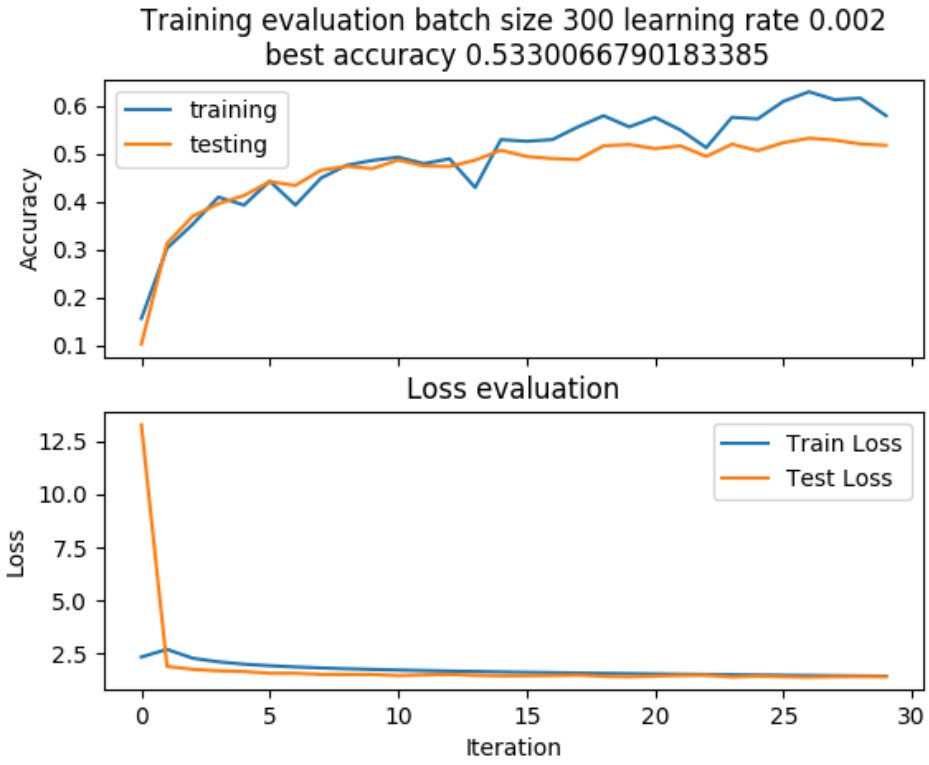

Figure 2: Implementation of deep neural network with PyTorch with highest test accuracy of 0.53 using **ADAM** optimizer, evaluated every 50-th iteration for 1500 iterations

We remark from Figure 2.1 that with a wider network used from our result of the gridsearch we achieve an high test accuracy, however the loss on the train nor the test converge further than $1, 13$. The stagnating loss, may be due the choice of the optimizer that might lead to a slow convergence.

3 Custom Module: Batch Normalisation

3.1 Automatic differentiation

$$
\begin{aligned}
\mu_i &= \frac{1}{B}\sum_{s=1}^{B} x_i^s \\
\sigma_i^2 &= \frac{1}{B}\sum_{s=1}^{B} (x_i^s - \mu_i) \\
\hat{x}_i^s &= \frac{x_i^s - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}, \qquad \epsilon \ll 1 \\
y_i^s &= \gamma_i \hat{x}_i^s + \beta_i
\end{aligned}
$$

3.2 Manual Implementation of backward pass

- a

$$
\begin{aligned}
\frac{\partial L}{\partial \gamma} = \left(\frac{\partial L}{\partial \gamma}\right)_j &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \frac{\partial \partial y_i^s}{\partial \gamma_j} \\
&= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} 1_{[i=j]} \hat{x}_j^s \\
&= \sum_s \frac{\partial L}{\partial y^s} \odot \hat{x}^s
\end{aligned}
$$

$$
\begin{aligned}
\left(\frac{\partial L}{\partial \beta}\right)_j &= \frac{\partial L}{\partial y_i^s} \frac{\partial y_i^s}{\partial \beta_j} \\
&= \sum_s \sum_i \frac{\partial L}{\partial y_i^s} 1_{[i=j]} \\
&= \sum_s \frac{\partial L}{\partial y^s}
\end{aligned}
$$

$$
\begin{aligned}
&\frac{\partial}{\partial x_j}\mu_j = \frac{1}{B} \\
&\frac{\partial}{\partial x_j}\sigma_j^2 = \frac{1}{B}\sum_{s=1}^{B} 2\left(x_j^s - \mu_j\right)\left(\frac{\partial}{\partial x_j}x_j^s - \frac{\partial}{\partial x_j}\mu_j\right) \\
&\Rightarrow \frac{2}{B}\sum_{s=1}^{B}\left(x_j^s - \mu_j\right)\left(1_{[s=r]} - \frac{1}{B}\right) \\
&\Rightarrow \frac{2}{B}\sum_{s=1}^{B}\overbrace{\frac{1}{B}\left(x_j^s - \mu_j\right)}^{=0} + (x_r^j - \mu_j)
\end{aligned}
$$

Combining the above in the below chain rule we will continue to differentiate

$$
\begin{aligned}
\left(\frac{\partial L}{\partial x}\right)_j^r &= \sum_s \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \hat{x}_j^s}\frac{\partial \hat{x}_j^s}{\partial x_r^j} \\
&\Rightarrow \frac{\partial}{\partial x_j}\hat{x}_j^s = \frac{\frac{\partial}{\partial x_j^r}x_j^s - \frac{\partial}{\partial x_j^r}\mu_j}{\frac{\partial}{\partial x_j^r}\sqrt{\sigma_j^2+\epsilon}} \\
&\equiv \frac{\sqrt{\sigma_j^2+\epsilon}\left(1_{[r=s]} - \frac{1}{B}\right) - (x_j^s-\mu_j)\frac{1}{2}(\sigma_j^2+\epsilon)^{-\frac{1}{2}}(\frac{2}{B}(x_j^r-\mu_j))}{\left(\sigma_j^2+\epsilon\right)} \\
&\equiv \frac{\sqrt{\sigma^2+\epsilon}1_{[r=s]}}{\sigma^2+\epsilon} - \frac{\frac{\sqrt{\sigma^2+\epsilon}}{B}}{\sigma^2+\epsilon} - \frac{\frac{(x^s-\mu)(\frac{2}{B}(x^r-\mu))}{2\sqrt{\sigma^2+\epsilon}}}{\sigma^2+\epsilon} \\
&\equiv \frac{1_{r=s}}{\sqrt{\sigma^2+\epsilon}} - \frac{1}{\sqrt{\sigma^2+\epsilon}}\frac{1}{B} - \frac{(x^s-\mu)(\frac{2}{B}(x^r-\mu))}{2\sqrt{\sigma^2+\epsilon}}\frac{1}{\sigma^2+\epsilon} \\
&\equiv \frac{1}{\sqrt{\sigma^2+\epsilon}}\left(1_{[r=s]} - \frac{1}{B} - \frac{1}{B}\frac{(x^s-\mu)(x^r-\mu)}{\sigma^2+\epsilon}\right) \\
&\equiv \frac{1}{\sqrt{\sigma^2+\epsilon}}\left(1_{[r=s]} - \frac{1}{B}\left(1 + \frac{(x^s-\mu)(x^r-\mu)}{\sigma^2+\epsilon}\right)\right)
\end{aligned}
$$

Combining the above chainrule we will obtain the following

$$
\begin{aligned}
\left(\frac{\partial L}{\partial x}\right)_j^r &= \frac{1}{\sqrt{\sigma^2+\epsilon}}\left(\gamma \odot \frac{\partial L}{\partial y^r} - \gamma * \sum_s \frac{\partial L}{\partial y^s}\frac{1}{B}(1 + \frac{(x^s-\mu)(x^r-\mu)}{\sigma^2+\epsilon})\right) \\
&\equiv \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}\frac{1}{B}\left(B * \frac{\partial L}{\partial y^r} - \sum_s \frac{\partial L}{\partial y^s}(1 + \frac{(x^s-\mu)(x^r-\mu)}{\sigma^2+\epsilon})\right) \\
&\equiv \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}\frac{1}{B}\left(B * \frac{\partial L}{\partial y^r} - \sum_s \frac{\partial L}{\partial y^s} + \sum_s \frac{\partial L}{\partial y^s}\frac{(x^s-\mu)(x^r-\mu)}{\sigma^2+\epsilon}\right)
\end{aligned}
$$

- b
  See code

- c
  See code

4 PyTorch CNN

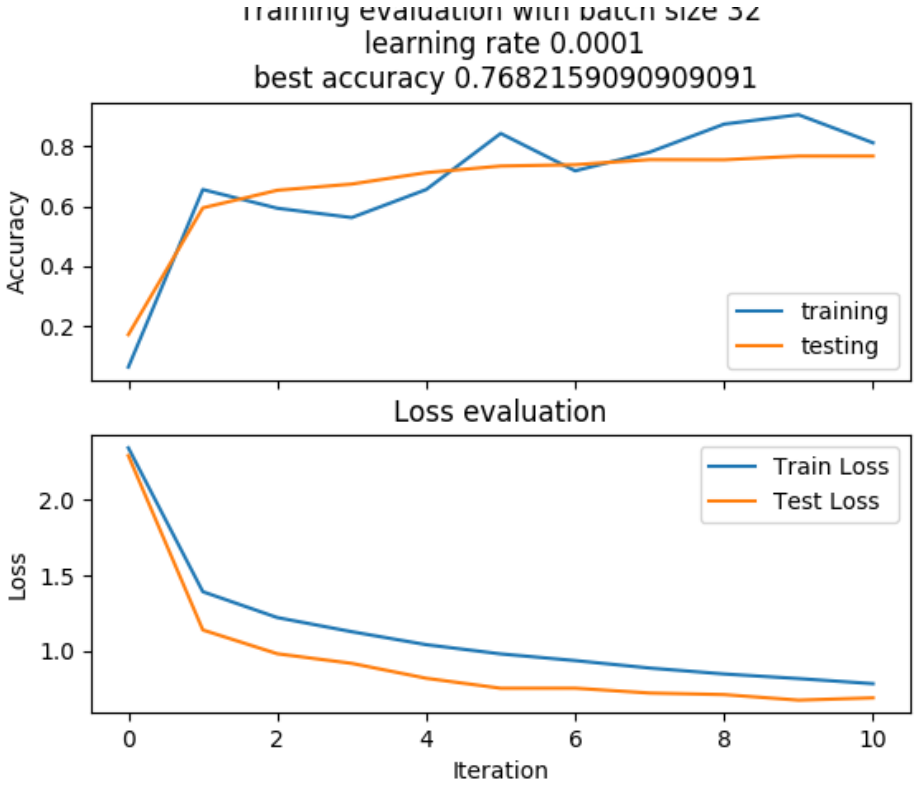

Figure 3: Implementation of Convolutional deep neural network with PyTorch with highest test accuracy of 0.768 using **ADAM** optimizer on 32 batch sizes, evaluated every 500-th iteration for 5500 iterations

Remarkable from Figure 4 compared to Figure 2.1 or 1.2 that this implementation surpasses previous implementation on the test accuracy. Moreover we can see, however the fluctuation during training, the test accuracy increases without any big changes. This is also visible from the loss of the test set, where we see no abrupt changes.