Assignment 1. MLPs, CNNs and Backpropagation

Maurice Frank
11650656
maurice.frank@posteo.de
Code: github

1 MLP backpropagation

1.1 Analytical derivation of gradients

1.1.a)

$$\begin{aligned}
\frac{\partial L}{\partial x_i^{(N)}} &= -\frac{\partial}{\partial x_i^{(N)}} \sum_i t_i \log x_i^{(N)} \\
&= -t_i \cdot \frac{1}{x_i^{(N)}} \\
&\iff \\
\frac{\partial L}{\partial \boldsymbol{x}^{(N)}} &= -[\cdots \frac{t_i}{x_i^{(N)}} \cdots] \\
&= \boldsymbol{t} \oslash \boldsymbol{x}^{(N)} \\
&\in \mathbb{R}^{d_N}
\end{aligned} \tag{$\frac{\partial \boldsymbol{x}^{(N)}}{\partial L}$}$$

$$\begin{aligned}
\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} &= \frac{\partial}{\partial \tilde{x}_j^{(N)}} \frac{\exp \tilde{x}_i^{(N)}}{\sum_k \exp \tilde{\boldsymbol{x}}_k^{(N)}} \\
&= \frac{\left(\frac{\partial}{\partial \tilde{x}_j^{(N)}} \exp \tilde{x}_i^{(N)}\right) \cdot \sum_k \exp \tilde{x}_k^{(N)} - \exp \tilde{x}_i^{(N)} \cdot \frac{\partial}{\partial \tilde{x}_j^{(N)}} \sum_k \exp \tilde{x}_k^{(N)}}{\left(\sum_k \exp \tilde{x}_k^{(N)}\right)^2} \\
&= \frac{\delta_{ij} \exp \tilde{x}_j^{(N)}}{\sum_k \exp \tilde{x}_k^{(N)}} - \frac{\exp \tilde{x}_i^{(N)} \cdot \exp \tilde{x}_j^{(N)}}{\left(\sum_k \exp \tilde{x}_k^{(N)}\right)^2} \\
&= \mathrm{softmax}(\tilde{x}_j^{(N)}) \cdot (\delta_{ij} - \mathrm{softmax}(\tilde{x}_i^{(N)})) \\
&\Rightarrow \\
\frac{\partial \boldsymbol{x}^{(N)}}{\partial \tilde{\boldsymbol{x}}^{(N)}} &= \begin{bmatrix} & \vdots & \\ \dots & \mathrm{softmax}(\tilde{x}_j^{(N)}) \cdot (\delta_{ij} - \mathrm{softmax}(\tilde{x}_i^{(N)})) & \dots \\ & \vdots & \end{bmatrix} \\
&= \mathrm{diag}(\mathrm{softmax}(\tilde{\boldsymbol{x}}^{(N)})) - \mathrm{softmax}(\tilde{\boldsymbol{x}}^{(N)}) \otimes \mathrm{softmax}(\tilde{\boldsymbol{x}}^{(N)}) \qquad \in \mathbb{R}^{d_N \times d_N}
\end{aligned} \tag{$\frac{\partial \boldsymbol{x}^{(N)}}{\partial \tilde{\boldsymbol{x}}^{(N)}}$}$$

$$\frac{\partial \boldsymbol{x}^{(l<N)}}{\partial \tilde{\boldsymbol{x}}^{(l<N)}} = \frac{\partial}{\partial \tilde{\boldsymbol{x}}^{(l<N)}} \max(0, \tilde{\boldsymbol{x}}^{(l<N)}) \tag{$\frac{\partial \boldsymbol{x}^{(l<N)}}{\partial \tilde{\boldsymbol{x}}^{(l<N)}}$}$$
$$= \operatorname{diag}(\boldsymbol{x}^{(l<N)} \oslash \tilde{\boldsymbol{x}}^{(l<N)})$$
$$\in \mathbb{R}^{d_l \times d_l}$$

$$\frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{x}^{(l-1)}} = \frac{\partial}{\partial \boldsymbol{x}^{(l-1)}} \boldsymbol{W}^{(l)} \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{$\frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{x}^{(l-1)}}$}$$
$$= \boldsymbol{W}^{(l)}$$
$$\in \mathbb{R}^{d_l \times d_{l-1}}$$

$$\frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial}{\partial \boldsymbol{W}^{(l)}} \boldsymbol{W}^{(l)} \boldsymbol{x}^{(l-1)} \tag{$\frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{W}^{(l)}}$}$$
$$= \begin{bmatrix} \vdots \\ \frac{\partial \tilde{\boldsymbol{x}}_i^{(l)}}{\partial \boldsymbol{W}^{(l)}} \\ \vdots \end{bmatrix}$$
$$\in \mathbb{R}^{d_l \times (d_l \times d_{l-1})}$$

with

$$\frac{\partial \tilde{\boldsymbol{x}}_i^{(l)}}{\partial \boldsymbol{W}^{(l)}} = \begin{bmatrix} \vdots \\ \boldsymbol{x}^{(l-1)^T} \\ \vdots \end{bmatrix}$$
$$\in \mathbb{R}^{d_l \times d_{l-1}}$$

$$\frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial}{\partial \boldsymbol{b}^{(l)}} \boldsymbol{b}^{(l)} \tag{$\frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{b}^{(l)}}$}$$
$$= \mathbb{1}^{d_l \times d_l}$$
$$\in \mathbb{R}^{d_l \times d_l}$$

[1]

1.1.b)

$$\frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(N)}} = \frac{\partial L}{\partial \boldsymbol{x}^{(N)}} \frac{\partial \boldsymbol{x}^{(N)}}{\partial \tilde{\boldsymbol{x}}^{(N)}} \tag{$\frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(N)}}$}$$
$$= \frac{\partial L}{\partial \boldsymbol{x}^{(N)}} \cdot \operatorname{diag}(\operatorname{softmax}(\tilde{\boldsymbol{x}}^{(N)})) - \operatorname{softmax}(\tilde{\boldsymbol{x}}^{(N)}) \otimes \operatorname{softmax}(\tilde{\boldsymbol{x}}^{(N)})$$

[1]Note the use of $\oslash$ for element-wise division, the use of δ for the Kronecker-Delta and the use of $\otimes$ for the Outer Product.

$$\begin{aligned}\frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l<N)}} &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}} \frac{\partial \boldsymbol{x}^{(l)}}{\partial \tilde{\boldsymbol{x}}^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}} \cdot \mathrm{diag}(\boldsymbol{x}^{(l)} \oslash \tilde{\boldsymbol{x}}^{(l)}) \\ &= \begin{bmatrix} \ddots & & \\ & \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}} \cdot \frac{x_i^{(l)}}{\tilde{x}_i^{(l)}} & \\ & & \ddots \end{bmatrix}\end{aligned} \tag{$\frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l<N)}}$}$$

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{x}^{(l<N)}} &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l+1)}} \frac{\partial \tilde{\boldsymbol{x}}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l+1)}} \cdot \boldsymbol{W}^{(l+1)}\end{aligned} \tag{$\frac{\partial L}{\partial \boldsymbol{x}^{(l<N)}}$}$$

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}} \frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{W}^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}} \begin{bmatrix} \vdots \\ \frac{\partial \tilde{\boldsymbol{x}}_i^{(l)}}{\partial \boldsymbol{W}^{(l)}} \\ \vdots \end{bmatrix}\end{aligned} \tag{$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}}$}$$

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}} \frac{\partial \tilde{\boldsymbol{x}}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}} \mathbb{1}^{d_l \times d_l} \\ &= \frac{\partial L}{\partial \tilde{\boldsymbol{x}}^{(l)}}\end{aligned} \tag{$\frac{\partial L}{\partial \boldsymbol{b}^{(l)}}$}$$

1.1.c)

If we use a batchsize $B > 1$ we have to perform the same derivations as described above but over a new batch dimension. Meaning all matrices become tensor with the new first axis being of size B. As the batch items do not interfer with each other the results will be equivalent to doing the not batched dervivatives for each item alone. For the actual updates of the weights we gonna have a sum which can be build into the matrix computation.

1.2 NumPy MLP

The performance of the NumPy implementation of the MLP under differnt hyperparameter settings are show in Tabel **??**. The loss and accuracy curves of the best model are shown in Figure **??**. Source code is in files `mlp_numpy.py`, `train_mlp_numpy.py` and `modules.py`.

2 PyTorch MLP

Source code is in files `mlp_pytorch.py` and `train_mlp_pytorch.py`. The performance of the PyTorch MLP model under diffent settings are shown in Table **??**. The loss and accuracy curves during training of the best performing model are shown in Figure **??**. We experiment with different hyperparameter settings. We change the number of hidden neurons and the number of hidden layers.

Layers	LR	Batch size	Test Accuracy	Test Loss
100	2e-3	200	46.86	1.51
10	2e-3	200	39.38	1.68
200	1e-3	200	**50.37**	**1.42**
80,50	1e-5	200	11.17	2.30

Table 1: Results for the NumPy MLP under different parameter settings. Layers shows the sorted list of number of neurons for the hidden layers. LR is learning rate. Reported are the highest accuracy and lowest loss on the test set during training.

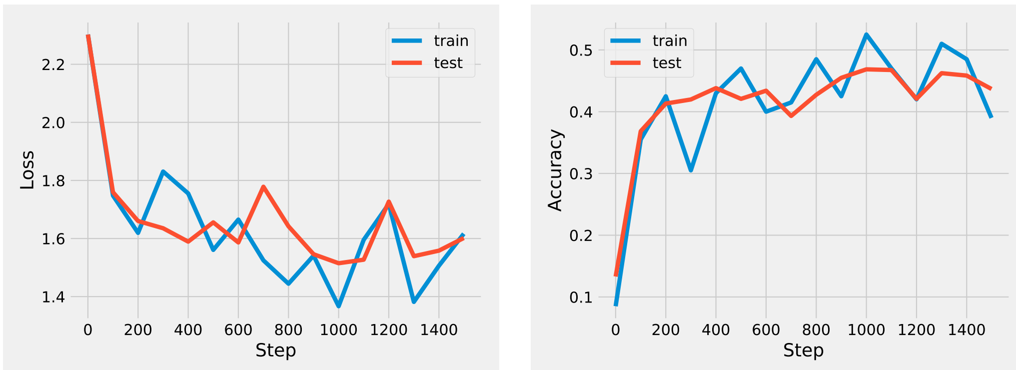

Figure 1: **Left** the loss and **right** the accuracy during training of the NumPy MLP implementation using the default hyperparameters.

We set different learning rates. We use SGD and Adam. SGD is stochastic Gradient descent with variable learning rate but we fix momentum to zero. Alternatively we use the Adam optimizer and add regularization in the form of weight decay with a factor of $1E-2$. The best result is achieved using Adam and three layers of decreasing size. Generally Adam more robustly optimizes under differnt settings. As these results are not averages over multiple runs we see a lot of variability. For example the single layer network with only ten nodes achieves higher accuracy than using three-hundred nodes with SGD.

3 Batch normalization

3.1 Autograd

See the implementation in `custom_batchnorm.py`: `CustomBatchNormAutograd`.

Layers	Optimizer	LR	Test Accuracy	Test Loss
100	SGD	2e-3	45.58	1.61
10	SGD	2e-3	39.52	1.70
200	SGD	1e-3	49.11	1.52
80,50	SGD	1e-5	31.08	2.26
500,300	SGD	1e-3	48.72	1.58
300	SGD	1e-5	33.27	4.25
300	Adam	1e-3	48.46	1.63
500,300	Adam	1e-3	47.66	1.49
600,300,100	Adam	1e-4	**53.55**	**1.43**

Table 2: Results of the PyTorch MLP using differnt settings for the hyperparameters. SGD is stochastic gradient descent withtout momentum.

Figure 2: **Left** the loss and **right** the accuracy during training of the best performing PyTorch MLP implementation. (Single layer 200 neurons)

3.2 Manual gradient

3.2.a)

$$\begin{aligned}\frac{\partial L}{\partial \gamma} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial \gamma} \\ &= \begin{bmatrix} \vdots \\ \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \gamma_j} \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \vdots \\ \sum_s \frac{\partial L}{\partial y_j^s}\hat{x}_j^s \\ \vdots \end{bmatrix} \\ &= \sum_s \frac{\partial L}{\partial y^s} \odot \hat{x}^s \end{aligned} \tag{$\frac{\partial L}{\partial \gamma}$}$$

[2]

$$\begin{aligned}\frac{\partial L}{\partial \beta} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial \beta} \\ &= \begin{bmatrix} \vdots \\ \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \beta_j} \\ \vdots \end{bmatrix} \\ &= \begin{bmatrix} \vdots \\ \sum_s \frac{\partial L}{\partial y_j^s} \\ \vdots \end{bmatrix} \\ &= \sum_s \frac{\partial L}{\partial y^s} \end{aligned} \tag{$\frac{\partial L}{\partial \beta}$}$$

[2]Note the use of $\odot$ for element-wise multiplication.

$$
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial (x-\mu)} + \frac{\partial L}{\partial \mu}\frac{\partial \mu}{\partial x} \tag{$\frac{\partial L}{\partial x}$}
$$

with

$$
\begin{aligned}
\frac{\partial \mu}{\partial x} &= \frac{1}{B}\mathbb{1}^{B\times B} \\
\frac{\partial L}{\partial \mu} &= -\sum_s \left(\frac{\partial L}{\partial (x-\mu)}\right)_s \\
&\implies \\
\frac{\partial L}{\partial x} &= \frac{\partial L}{\partial (x-\mu)} - \sum_s \left(\frac{\partial L}{\partial (x-\mu)}\right)_s \cdot \frac{1}{B}\mathbb{1}^{B\times B}
\end{aligned}
$$

so missing is $\frac{\partial L}{\partial (x-\mu)}$: (We set $\hat{\sigma} = \sqrt{\sigma^2+\epsilon}$ for simplicity.)

$$
\frac{\partial L}{\partial (x-\mu)} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial (x-\mu)} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial {}^1\!/\!\hat{\sigma}}\frac{\partial {}^1\!/\!\hat{\sigma}}{\partial \hat{\sigma}}\frac{\partial \hat{\sigma}}{\partial \sigma}\frac{\partial \sigma}{\partial (x-\mu)}
$$

with

$$
\begin{aligned}
\frac{\partial y}{\partial \hat{x}} &= \gamma \\
\frac{\partial \hat{x}}{\partial (x-\mu)} &= \frac{1}{\hat{\sigma}}
\end{aligned}
$$

and with

$$
\begin{aligned}
\frac{\partial L}{\partial {}^1\!/\!\hat{\sigma}} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial {}^1\!/\!\hat{\sigma}} \\
&= \sum_s \left(\frac{\partial L}{\partial \hat{x}}\right)_s \cdot (x_s - \mu_s) \\
\frac{\partial {}^1\!/\!\hat{\sigma}}{\partial \hat{\sigma}} &= -\frac{1}{\hat{\sigma}^2} \\
\frac{\partial \hat{\sigma}}{\partial \sigma} &= \frac{1}{2\cdot\sqrt{\sigma+\epsilon}} \\
\frac{\partial \sigma}{\partial (x-\mu)} &= 2\cdot(x-\mu)\cdot\frac{1}{B}\mathbb{1}^{B\times B}
\end{aligned}
$$

3.2.b)

See implementation of the manual backward pass in `custom_batchnorm.py`: `CustomBatchNormManualFunction`. We use the functions context to save the five tensors $x-\mu$, σ^2, $\frac{1}{\sqrt{\sigma^2+\epsilon}}$, $\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}$ and γ. In that we make a conscious decisions to trade time performance for more memory usage.

3.2.c)

See the implementation in `custom_batchnorm.py`: `CustomBatchNormManualModule`.

4 PyTorch CNN

See the implementation in `convnet_pytorch.py` and `train_convnet_pytorch.py`. Curves for loss and accuracy during training are shown in Figure **??**. Using the default parameters and the Adam optimizer we reach a maximum of 77.08% accuracy on the test set.

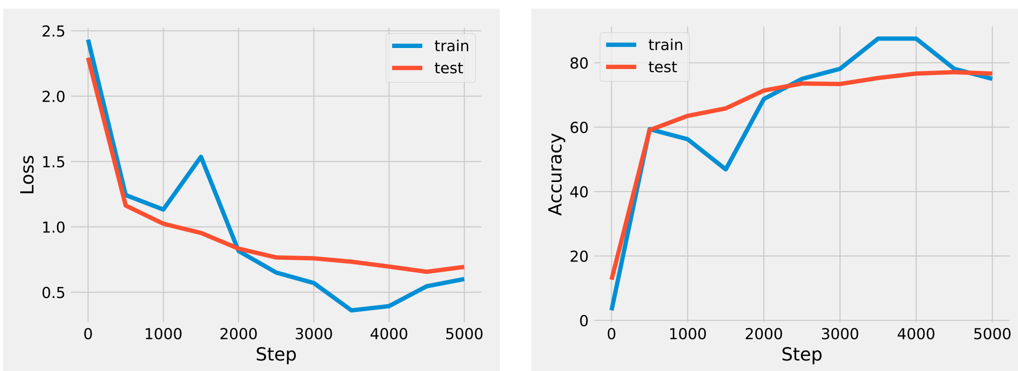

Figure 3: **Left** the loss and **right** the accuracy during training of the PyTorch CNN implementation.