Practical 1. MLPs, CNNs and Backpropagation

Hannah Min
University of Amsterdam
Student number 11011580
hannah.min@student.uva.nl

1 MLP backprop and Numpy Implementation

1.1 Question

(a) **Linear Module**

Derivative with respect to $\mathbf{W}$:

$$
\begin{aligned}
[\frac{\partial L}{\partial \mathbf{W}}]_{ij} &= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial W_{ij}} \\
Y_{mn} &= \sum_{p} X_{mp} W^T_{pn} + b_n = \sum_{p} X_{mp} W_{np} + b_n \\
[\frac{\partial L}{\partial \mathbf{W}}]_{ij} &= \sum_{m,n,p} \frac{\partial L}{\partial Y_{mn}} \frac{\partial (X_{mp} W_{np} + b_n)}{\partial W_{ij}} \\
&= \sum_{m,n,p} \frac{\partial L}{\partial Y_{mn}} X_{mp} \delta_{in} \delta_{pj} \\
&= \sum_{m} \frac{\partial L}{\partial Y_{mi}} X_{mj} \\
\frac{\partial L}{\partial \mathbf{W}} &= \mathbf{X}^T \frac{\partial L}{\partial \mathbf{Y}}
\end{aligned}
$$

Derivative with respect to $\mathbf{b}$:

$$
\begin{aligned}
[\frac{\partial L}{\partial \mathbf{b}}]_{i} &= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial b_i} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{ni} \\
&= \sum_{m} \frac{\partial L}{\partial Y_{mi}} \\
\frac{\partial L}{\partial \mathbf{b}} &= \mathbf{1}_{1,m} \frac{\partial L}{\partial \mathbf{Y}}
\end{aligned}
$$

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Derivative with respect to $\mathbf{X}$:

$$
\begin{aligned}
[\frac{\partial L}{\partial \mathbf{X}}]_{ij} &= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n,p} \frac{\partial L}{\partial Y_{mn}} \frac{\partial (X_{mp} W_{np} + b_n)}{\partial X_{ij}} \\
&= \sum_{m,n,p} \frac{\partial L}{\partial Y_{mn}} W_{np} \delta_{mi} \delta_{pj} \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} W_{nj} \\
\frac{\partial L}{\partial \mathbf{X}} &= \frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}
\end{aligned}
$$

(b) **Activation Module**

$$
\begin{aligned}
Y_{mn} &= h(X_{mn}) \\
[\frac{\partial L}{\partial \mathbf{X}}]_{ij} &= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial h(X_{mn})}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{im} \delta_{jn} h'(X_{mn}) \\
&= \frac{\partial L}{\partial Y_{ij}} h'(X_{ij}) \\
\frac{\partial L}{\partial \mathbf{X}} &= \frac{\partial L}{\partial \mathbf{Y}} \circ h'(X_{ij})
\end{aligned}
$$

(c) **Softmax and Loss Modules**

(i) Derivative of softmax:

$$
\begin{aligned}
[\frac{\partial L}{\partial \mathbf{X}}]_{ij} &= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{m,n} \frac{\partial L}{\partial Y_{mn}} \delta_{mi} \frac{\partial Y_{mn}}{\partial X_{ij}} \\
&= \sum_{n} \frac{\partial L}{\partial Y_{in}} \frac{\partial Y_{in}}{\partial X_{ij}}
\end{aligned}
$$

For the case $n = j$ (diagonal)

$$
\begin{aligned}
\frac{\partial Y_{in}}{\partial X_{ij}} &= \frac{\partial (\frac{\exp(X_{in})}{\sum_{n'} \exp(X_{in'})})}{\partial X_{ij}} \\
&= \frac{\exp(X_{in}) \sum_{n'} \exp(X_{in'}) - \exp(X_{in}) \exp(X_{in})}{(\sum_{n'} \exp(X_{in'}))^2} \\
&= \frac{\exp(X_{in})(\sum_{n'} \exp(X_{in'}) - \exp(X_{in}))}{(\sum_{n'} \exp(X_{in'}))^2} \\
&= \frac{\exp(X_{in})}{\sum_{j'} \exp(X_{in'})} \frac{\sum_{n'} \exp(X_{in'}) - \exp(X_{in})}{\sum_{n'} \exp(X_{in'})} \\
&= Y_{in}(1 - Y_{in}) \\
&= Y_{ij}(1 - Y_{ij})
\end{aligned}
$$

For the case $n \neq j$ (off diagonal)

$$\begin{aligned}\frac{\partial Y_{in}}{\partial X_{ij}} &= \frac{\partial(\frac{\exp(X_{in})}{\sum_{n'}\exp(X_{in'})})}{\partial X_{ij}} \\ &= \frac{0 - \exp(X_{in})\exp(X_{ij})}{(\sum_{n'}\exp(X_{in'}))^2} \\ &= -\frac{\exp(X_{ij})}{\sum_{n'}\exp(X_{in'})}\frac{\exp(X_{in})}{\sum_{n'}\exp(X_{in'})} \\ &= -Y_{in}Y_{ij}\end{aligned}$$

$$\begin{aligned}[\frac{\partial L}{\partial \mathbf{X}}]_{ij} &= \sum_n \frac{\partial L}{\partial Y_{in}}\frac{\partial Y_{in}}{\partial X_{ij}} \\ &= \frac{\partial L}{\partial Y_{ij}}(Y_{ij}(\delta_{ij} + Y_{ij}))\end{aligned}$$

(ii) Derivative of cross entropy loss:

$$\begin{aligned}L &= -\frac{1}{S}\sum_{ik} T_{ik}\log(X_{ik}) \\ [\frac{\partial L}{\partial \mathbf{X}}]_{ik} &= -\frac{1}{S}\frac{T_{ik}}{X_{ik}} \\ \frac{\partial L}{\partial \mathbf{X}} &= -\frac{1}{S}\mathbf{T} \oslash \mathbf{X}\end{aligned}$$

$\oslash$ denotes Hadamard division

1.2 Question: Numpy implementation

Test accuracy in last iteration: 0.4856

In the plots, we observe that the training loss and accuracy are noisy due to only considering the training batch. We see that the training and test evaluation stay relatively close to each other, which means that we are not overfitting.

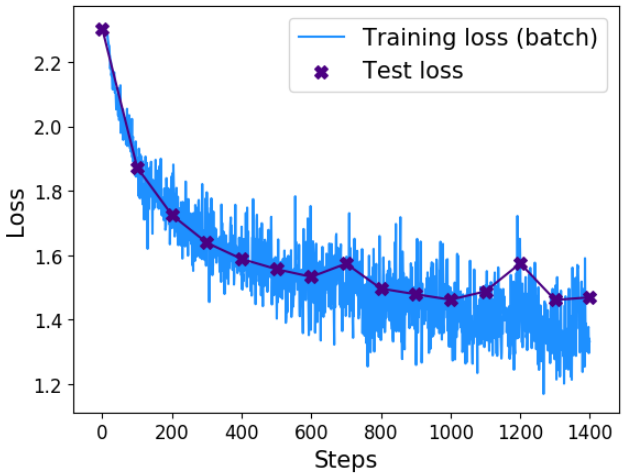

Figure 1: Cross entropy loss curves for MLP implementation in Numpy. Test loss was calculated every 100 steps.

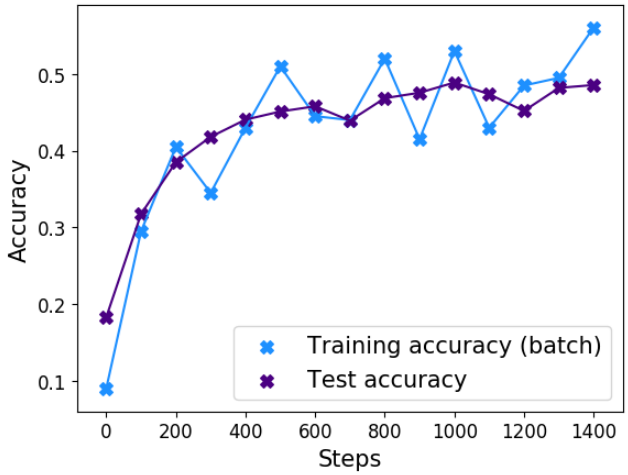

Figure 2: Accuracy curves for MLP implementation in Numpy. The model was evaluated every 100 steps.

2 Pytorch MLP

2.1 Question

My first approach was to use only one hidden layer with Stochastic Gradient Descent (SGD) optimization and play with the batch size and number of units in that hidden layer. With more than 400 units in de hidden layer I observed more overfitting. The following hyperparameter setting led to on time exceeding a test accuracy of 0.52.

Hyperparameter setting:
dnn_hidden_units: 400
max_steps: 1500
$batch\_size$: 1000
$optimizer$: SGD

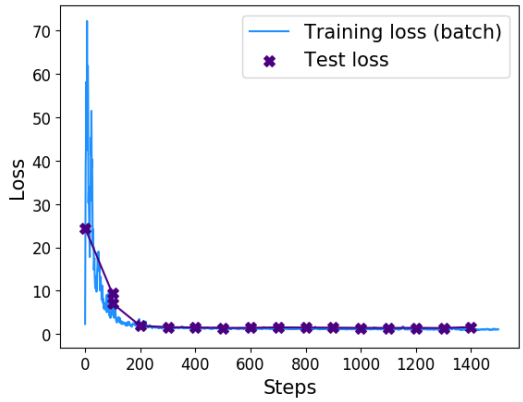

Figure 3: Loss curve with SGD optimizer

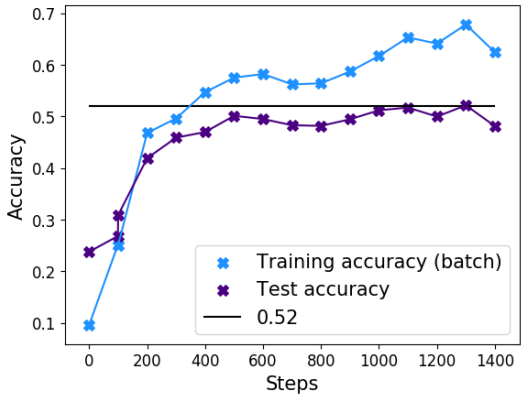

Figure 4: Accuracy curve with SGD optimizer.

My second approach was to try a deeper network using Adam optimizer, as SGD does not work well for deeper architectures. A deeper network has more expressive power, but in the loss and accuracy curves, we see that the network soon starts to overfit, as the training loss and accuracy become much larger than the test loss and accuracy. The following hyperparameter setting led to sometimes exceeding a test accuracy of 0.54.

Hyperparameter setting:
dnn_hidden_units: 200,100,100
max_steps: 2000
$batch\_size$: 800
$optimizer$: Adam

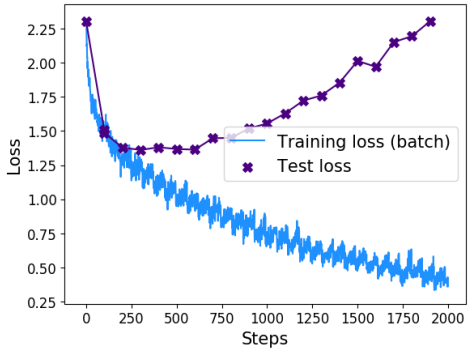

Figure 5: Loss curve with Adam optimizer.

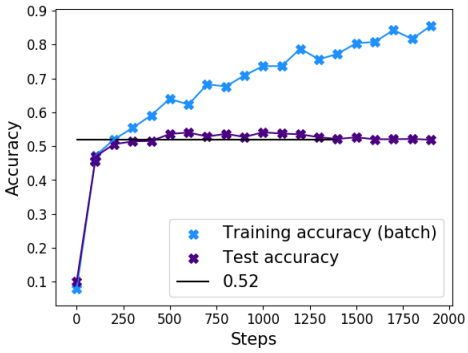

Figure 6: Accuracy curve with Adam optimizer

2.2 Question

Benefits and/or drawbacks of *Tanh* activation function, compared to *ELU*.

- *ELU* can have extremely large activation values for large x, whereas *Tanh* keeps the activations between -1 and 1.
- Both *ELU* and *Tanh* are smooth functions and therefore differentiable everywhere.
- *Tanh* suffers from the vanishing gradient problem, *ELU* does not.

3 Custom Module: Layer Normalization

3.1 Question

Implementation of layer normalization

3.2 Question

(a) Derivative with respect to $\boldsymbol{\gamma}$:

$$[\frac{\partial L}{\partial \boldsymbol{\gamma}}]_i = \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \gamma_i}$$
$$[\frac{\partial L}{\partial \boldsymbol{\gamma}}]_i = \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \hat{X}_{sj} \delta_{ij}$$
$$\frac{\partial L}{\partial \boldsymbol{\gamma}} = \mathbf{1}_{1,s} (\frac{\partial L}{\partial \mathbf{Y}} \circ \hat{\mathbf{X}})$$

Derivative with respect to $\boldsymbol{\beta}$:

$$[\frac{\partial L}{\partial \boldsymbol{\beta}}]_i = \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \beta_i}$$
$$[\frac{\partial L}{\partial \boldsymbol{\beta}}]_i = \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} 1 \cdot \delta_{ij}$$
$$[\frac{\partial L}{\partial \boldsymbol{\beta}}]_i = \sum_{s} \frac{\partial L}{\partial Y_{sj}}$$
$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \mathbf{1}_{1,s} \frac{\partial L}{\partial \mathbf{Y}}$$

Derivative with respect to $\mathbf{X}$:

$$\hat{X}_{si} = \frac{X_{si} - \mu_s}{\sqrt{\sigma_s^2 + \epsilon}}$$
$$Y_{si} = \gamma_i \hat{X}_{si} + \beta_i$$

$$\begin{aligned}
[\frac{\partial L}{\partial \mathbf{X}}]_{ri} &= \sum_{s,j} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial X_{ri}} \\
&= \sum_{s,j,t,k} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \hat{X}_{tk}} \frac{\partial \hat{X}_{tk}}{\partial X_{ri}} \\
&= \sum_{s,j,t,k} \frac{\partial L}{\partial Y_{sj}} \gamma_j \frac{\partial \hat{X}_{tk}}{\partial X_{ri}}
\end{aligned}$$

(b) Layer normalization with *torch.autograd.Function*

(c) Implementation

(d) **Compare Layer Normalization with Batch Normalization:**
Batch normalization normalizes features across the batch, whereas layer normalization normalizes across features (neurons).
What problems do they address?
Both methods try to stabilize and speed up the training process. Layer normalization works especially well for recurrent neural networks.
What are their limitations?
For Batch Normalization, if the batch size is small, the mean and variance are not representative and noisy. This problem is addressed by Layer normalization, as this is independent of the batch size.

4 Pytorch CNN

4.1 Question

(a) Accurcay and loss curves
Test accuracy in last iteration: 0.7926

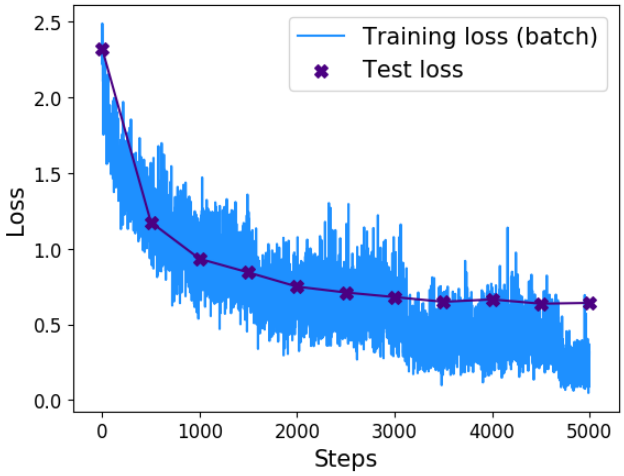

Figure 7: Loss curve for CNN.

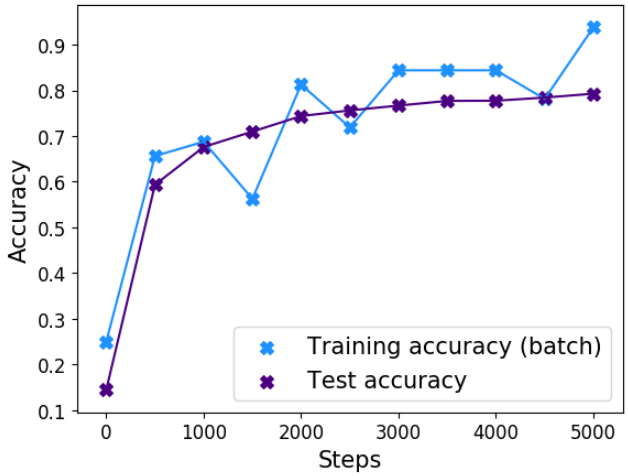

Figure 8: Accuracy curve for CNN.

(b)