Julia Belloni

Question 1

Linear Module

a. Find $\frac{\partial L}{\partial \mathbf{W}}$ analytically

Solution:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$$

$$\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{W}}\right]_{mn} &= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial W_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial \sum_k X_{ik} W^T_{kj} + B_{ij}}{\partial W_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial \sum_k X_{ik} W_{jk} + B_{ij}}{\partial W_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\sum_k \partial W_{jk}}{\partial W_{mn}} X_{ik} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \sum_k X_{ik} \delta_{jm} \delta_{kn} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} X_{in} \delta_{jm} \\
&= \sum_i \frac{\partial L}{\partial Y_{im}} X_{in} \\
&= \sum_i \left[\frac{\partial L}{\partial \mathbf{Y}}\right]_{im} X_{in} \\
&= \sum_i \left[\frac{\partial L}{\partial \mathbf{Y}}\right]^T_{mi} X_{in} \\
&= \left(\frac{\partial L}{\partial \mathbf{Y}}\right)^T \mathbf{X}
\end{aligned}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \left(\frac{\partial L}{\partial \mathbf{Y}}\right)^T \mathbf{X}$$

b.

Find $\frac{\partial L}{\partial \mathbf{b}}$ analytically

Solution:

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{b}}$$

$$\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{b}}\right]_{mn} &= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial b_n} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial \sum_k X_{ik} W^T_{kj} + B_{ij}}{\partial b_n} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial b_j}{\partial b_n} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \delta_{jn} \\
&= \sum_i \frac{\partial L}{\partial Y_{in}} = \mathbf{1} \frac{\partial L}{\partial \mathbf{Y}}
\end{aligned}$$

$$\frac{\partial L}{\partial \mathbf{b}} = \mathbf{1} \frac{\partial L}{\partial \mathbf{Y}}$$

c. Find $\frac{\partial L}{\partial \mathbf{X}}$ analytically

Solution:

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$$

$$\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{mn} &= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial X_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial \sum_k X_{ik} W^T_{kj} + B_{ij}}{\partial X_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial \sum_k X_{ik} W_{jk} + B_{ij}}{\partial X_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\sum_k \partial X_{ik}}{\partial X_{mn}} W_{jk} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \sum_k W_{jk} \delta_{im} \delta_{kn} \\
&= \sum_j \frac{\partial L}{\partial Y_{mj}} W_{jn} \\
&= \sum_j \left[\frac{\partial L}{\partial \mathbf{Y}}\right]_{mj} W_{jn}
\end{aligned}$$

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}$$

Activation Module

d. Find $\frac{\partial L}{\partial \mathbf{X}}$ analytically, using that $Y_{ij} = h(X_{ij})$

Solution:

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$$

$$\begin{aligned}
\left[\frac{\partial L}{\partial \mathbf{X}}\right]_{mn} &= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial X_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial h(X_{ij})}{\partial X_{mn}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} h'(X_{ij}) \delta_{im} \delta_{jn} \\
&= \frac{\partial L}{\partial Y_{mn}} h'(X_{mn}) \\
&= \left[\frac{\partial L}{\partial \mathbf{Y}}\right]_{mn} h'(X_{mn})
\end{aligned}$$

$$\left[\frac{\partial L}{\partial \mathbf{X}}\right] = \left[\frac{\partial L}{\partial \mathbf{Y}}\right] \circ h'(\mathbf{X})$$

e. Find $\alpha\mathbf{M}$ analytically

Solution: Knowing that:

$$\frac{\partial L}{\partial \mathbf{Z}} = \mathbf{Y} \circ \left(\frac{\partial L}{\partial \mathbf{Y}} - \left(\frac{\partial L}{\partial \mathbf{Y}} \circ \mathbf{Y} \right) \mathbf{1}\mathbf{1}^T \right) \tag{1}$$

$$\frac{\partial L}{\partial \mathbf{Y}} = -\frac{1}{S}\frac{\mathbf{T}}{\mathbf{Y}} \tag{2}$$

we can subxtitute equation 2 into equation 1. Getting the following result.

$$\begin{aligned}
\alpha\mathbf{M} = \frac{\partial L}{\partial \mathbf{Z}} &= \mathbf{Y} \circ \left(-\frac{1}{S}\frac{\mathbf{T}}{\mathbf{Y}} - \left(-\frac{1}{S}\frac{\mathbf{T}}{\mathbf{Y}} \circ \mathbf{Y} \right) \mathbf{1}\mathbf{1}^T \right) \\
&= \mathbf{Y} \circ \left(-\frac{1}{S}\frac{\mathbf{T}}{\mathbf{Y}} + \frac{1}{S}\mathbf{T}\ \mathbf{1}\mathbf{1}^T \right) \\
&= \left(-\mathbf{Y} \circ \frac{1}{S}\frac{\mathbf{T}}{\mathbf{Y}} + \frac{\mathbf{Y} \circ \mathbf{T}}{S}\mathbf{1}\mathbf{1}^T \right) \\
&= \left(-\frac{1}{S}\mathbf{T} + \frac{\mathbf{Y} \circ \mathbf{T}}{S}\mathbf{1}\mathbf{1}^T \right) \\
&= \frac{1}{S}\left(-\mathbf{T} + \mathbf{Y} \circ \mathbf{T}\mathbf{1}\mathbf{1}^T \right) \\
&= \frac{1}{S}\left(-\mathbf{T} + \mathbf{Y} \right)
\end{aligned}$$

Therefore, we get that:

$$\alpha\mathbf{M} = \frac{1}{S}\left(-\mathbf{T} + \mathbf{Y} \right)$$

Where $\alpha = \frac{1}{S}$ and $\mathbf{M} = \mathbf{Y} - \mathbf{T}$

Question 2 - 3

c. Provide the achieved test accuracy and training loss curve for the training of the **numpy** and **PyTorch** implementation, for the default values of parameters (one layer, 128 hidden units, 10 epochs, learning rate 0.1, seed 42).

Solution: Numpy implementation results can be seen in Figure 1 (test accuracy = 47.84%), While the Pythorch results are in Figure 2:

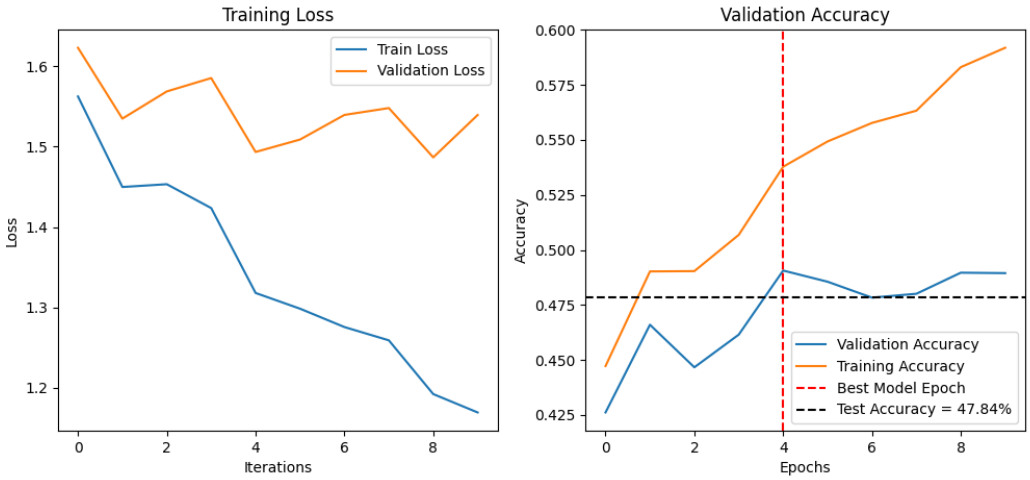


Figure 1: Numpy implementation (test accuracy = 47.84%)

Figure 2: Pytorch implementation results (test accuracy = 49.85%)

Question 4

a. Show that the eigenvalues for the Hessian matrix in a strictly local minimum $\mathbf{x}^*$ are all non-negative. Assume $f : \mathbb{R}^n \to \mathbb{R}$ to be C^2 (twice continuously differentiable).

Solution: Consider $\mathbf{x}^*$ to be a local minimum, then any point $\mathbf{x}^* + \Delta\mathbf{x}$ will result in:

$$f(\mathbf{x}^* + \Delta\mathbf{x}) \geq f(\mathbf{x}^*) \tag{3}$$

Using the second-order Taylor expansion, it is possible to rewrite $f(\mathbf{x}^* + \Delta\mathbf{x})$ as follows:

$$f(\mathbf{x}^* + \Delta\mathbf{x}) \approx f(\mathbf{x}^*) + \Delta\mathbf{x}\nabla f(\mathbf{x}^*) + \frac{1}{2}(\Delta\mathbf{x})^T\nabla^2 f(\mathbf{x}^*)(\Delta\mathbf{x}) \tag{4}$$

$$\tag{5}$$

Knowing that $\mathbf{x}^*$ is a local minimum it holds that: $\nabla f(\mathbf{x}^*) = 0$, leading to:

$$f(\mathbf{x}^* + \Delta\mathbf{x}) \approx f(\mathbf{x}^*) + \frac{1}{2}(\Delta\mathbf{x})^T\nabla^2 f(\mathbf{x}^*)(\Delta\mathbf{x}) \tag{6}$$

$$\tag{7}$$

Combining this result with eq. 3 we get that:

$$f(\mathbf{x}^* + \Delta\mathbf{x}) \approx f(\mathbf{x}^*) + \frac{1}{2}(\Delta\mathbf{x})^T\nabla^2 f(\mathbf{x}^*)(\Delta\mathbf{x}) \geq f(\mathbf{x}^*) \tag{8}$$

$$\frac{1}{2}(\Delta\mathbf{x})^T\nabla^2 f(\mathbf{x}^*)(\Delta\mathbf{x}) \geq 0 \tag{9}$$

$$(\Delta\mathbf{x})^T\nabla^2 f(\mathbf{x}^*)(\Delta\mathbf{x}) \geq 0 \tag{10}$$

This holds if and only if $\nabla^2 f(\mathbf{x}^*)$ is a positive semidefinite matrix. Since the hessian matrix defined by $\nabla^2 f(\mathbf{x}^*)$ is also a square matrix it follows that:

$$\nabla^2 f(\mathbf{x}^*)\mathbf{v} = \lambda\mathbf{v} \tag{11}$$

$$\tag{12}$$

where λ is a scalar. Multiplying each side by $\mathbf{v}^T$:

$$\mathbf{v}^T\nabla^2 f(\mathbf{x}^*)\mathbf{v} = \mathbf{v}^T\lambda\mathbf{v}$$

Since λ is a scalar, it can be brought in front:

$$\mathbf{v}^T\nabla^2 f(\mathbf{x}^*)\mathbf{v} = \lambda\mathbf{v}^T\mathbf{v}$$
$$\mathbf{v}^T\nabla^2 f(\mathbf{x}^*)\mathbf{v} = \lambda \parallel \mathbf{v} \parallel^2$$

Using eq.10 we get that:

$$\mathbf{v}^T\nabla^2 f(\mathbf{x}^*)\mathbf{v} = \lambda \parallel \mathbf{v} \parallel^2 \geq 0 \tag{13}$$

$$\tag{14}$$

Since $\parallel \Delta\mathbf{x} \parallel^2$ is always a positive quantity we get that:

$$\lambda \parallel \Delta\mathbf{x} \parallel^2 \geq 0 \tag{15}$$

$$\lambda \geq 0 \tag{16}$$

This implies that every eigen-value has to be non-negative.

b. If some of the eigenvalues of the Hessian matrix at point p are positive and some are negative, this point would be a saddle point; intuitively, explain why the number of saddle points is exponentially larger than the number of local minima for higher dimensions?

Solution: We can think of the sign of a matrixs eigenvalues as the outcome of coin flips. This can be modeled using a binomial distribution, where the number of positive eigenvalues corresponds to the number of successes k, and the total number of eigenvalues (the matrix rank) corresponds to the number of trials n. Therefore, the probability of a saddle point corresponds to a matrix that has k positive eigenvalues (where $k < n$) is:

$$\frac{n!}{k!(n-k)!}p^k$$

On the other hand, obtaining a local minima is more restrictive as it requires a positive semi-definite matrix, which is equivalent to the case where all the eigenvalues are positive $k = n$.

Therefore, the probability of getting a local minima is equivalent to:

$$\frac{n!}{k!(n-k)!}p^k = p^k \text{ given that } n = k$$

Since, $\frac{n!}{k!(n-k)!} \geq 1$ we can conclude that:

$$\frac{n!}{k!(n-k)!}p^k > p^k$$

When n increases the probability of obtaining a saddle point is $\frac{n!}{k!(n-k)!}$ times more likely, showing that with growing dimensions saddle points become a lot more probable than local minima.

c. By using the update formula of gradient descent around saddle point p, show why saddle points can be harmful to training.

Solution:

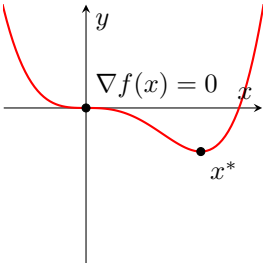
Considering that a gradient descent update is:

$$w^{t+1} = w^t - \alpha \nabla f(x)$$

and that in a saddle point the first order derivative is zero, $\nabla f(x) = 0$, the update would be

$$w^{t+1} = w^t$$

Therefore, the weights would stop getting updated even if the optimal solution has not been reached. This occurs because in both saddle points and local/global minima $\nabla f(x) = 0$ holds. As shown in the picture below the solution would be found in the origin, instead of x^*.



Question 5

a. Adding batch normalization layers causes changes to the backpropagation steps, because we also want to optimize the learnable β_i and γ_i parameters.

Solution:

$$\begin{aligned}
\left[\frac{\partial L}{\partial \gamma}\right]_n &= \sum_i \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial \gamma_n} \\
&= \sum_i \frac{\partial L}{\partial y_i}\frac{\partial(\gamma_i x_i + \beta_i)}{\partial \gamma_n} \\
&= \sum_i \frac{\partial L}{\partial y_i}\frac{\partial \gamma_i x_i}{\partial \gamma_n} \\
&= \sum_i \frac{\partial L}{\partial y_i} x_i \delta_{in} \\
&= \frac{\partial L}{\partial y_n} x_n = \left[\frac{\partial L}{\partial y}\right]_n x_n
\end{aligned}$$

$$\begin{aligned}\left[\frac{\partial L}{\partial \beta}\right]_n &= \sum_i \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial \beta_n} \\ &= \sum_i \frac{\partial L}{\partial y_i}\frac{\partial(\gamma_i x_i + \beta_i)}{\partial \beta_n} \\ &= \sum_i \frac{\partial L}{\partial y_i}\frac{\partial \beta_i}{\partial \beta_n} \\ &= \sum_i \frac{\partial L}{\partial y_i}\delta_{in} \\ &= \frac{\partial L}{\partial y_n} = \left[\frac{\partial L}{\partial y}\right]_n\end{aligned}$$

b. Consider applying batch normalization to a fully connected layer with an input size of 20 and an output size of 40. How many training parameters does this layer have, including batch normalization parameters?

Solution: In total, there are:

- The fully connected layer has $20 \times 40 = 800$ weights and 40 biases, for a total of 840 parameters.
- Batch normalization adds two learnable parameters (β and γ) per output, so $2 \times 40 = 80$ parameters.

Therefore, the total number of training parameters is $840 + 80 = \boxed{920}$

c. During training, batch normalization normalizes inputs using the mean and variance of the current mini-batch. Explain why it would be problematic to normalize inputs the same way during inference (test time), and how batch normalization addresses this problem.

Solution:

There are two main ways of carrying out inference, and depending on the method, batch normalization can be more problematic.

- **inductive**: in this scenario, each datapoint is processed independently by the model. Since the batch size is effectively 1, the sample variance collapses to 0, making normalization based on the current batch impossible. To handle this, batch normalization keeps track of running estimates (moving averages) of the mean and variance during training. At inference time, these global statistics are used to normalize each test example. This is the standard approach used in most applications.
- **transductive**: in this case, it is possible to use the distribution of the test data during inference, allowing the model to compute normalization statistics (mean and variance) from the test data itself, similarly to how it does during training. However, this approach is rarely used in practice.

d. Experimental analysis showed that a high percentage of neurons are dead in networks with ReLU activation functions (you can refer to tutorial 3 for more information). Explain the concept of a dead neuron, when it occurs when using ReLU, and how it harms training.

Solution: For the output of the neuron to be non-zero after ReLU activations, the input vector has to be greater than zero. In some cases, it might happen that all the inputs are negative, effectively leading to a neuron that does not let any activation through. This is called a dead neuron.

e. How does batch normalization prevent neurons from dying?

Solution: Batch normalization ensures that the variance does not change from layer to layer by first enforcing that the batch has zero mean and unit variance and then applying an affine transformation. This is helpful when using ReLU activations because it does not occur that all the inputs to one neuron are negative, effectively tackling the dying neuron issue.

f. Previously, you trained your PyTorch MLP using the default values of parameters (one layer, 128 hidden units, 10 epochs, no batch normalization, learning rate 0.1, seed 42). Now, retrain the model with the same parameters, but this time include batch normalization after the hidden layer. Compare the resulting accuracies with those obtained in the original setting, and motivate why these observations make sense.

Solution: The loss and accuracy curves of a model trained with batch normalization are smoother compared to the model without normalization. Generally, applying batch normalization introduces noise in the training pocedure which makes the variance more stable across the network and the forward propagation more stable. Furthermore, given that the network uses ELU activations, there should be less dead neurons in the later layers of the network, effectively increasing the networks capacity since more parameters can shape the output. Putting these observations together, it is not surprising that the model using batch normalization achieved a slightly higher accuracy (50.62%) compared to the network without (49.85%).

Figure 3: Loss curve and Accuracy for network trained using a PyTorch model with batch normalization.