Assignment 3

Numerical Optimization WS2024/25

Nhat Minh Hoang, (Group: A3 77)
Nadina Kapidzic, (Group: A3 77)

January 13, 2025

1. Size of Matrices and Bias Terms

- Input layer to hidden layer: Let N_H be the number of neurons in the hidden layer.
  $\mathbf{W}^{(0)}$ is the weight matrix between input layer and hidden layer.
  Since the input layer is of size: $x^s \in \mathbb{R}^4$, and the hidden layer is of size: $\mathbb{R}^{N_H}$, size of the weight matrix is then: $\mathbf{W}^{(0)} \in \mathbb{R}^{N_H \times 4}$.
  $\mathbf{b}^{(0)}$ is the bias terms of hidden layer, so it has the size of: $\mathbf{b}^{(0)} \in \mathbb{R}^{N_H \times 1}$.

- Hidden layer to output layer: Similarly, $\mathbf{W}^{(1)}$ is the weight matrix between hidden layer and output layer. Since the output layer is representing the 3 different classes of penguin, it is of size: $\mathbb{R}^3$. So, the size of the weight matrix is then: $\mathbf{W}^{(1)} \in \mathbb{R}^{3 \times N_H}$.
  $\mathbf{b}^{(1)}$ is the bias terms of output layer, so it has the size of: $\mathbf{b}^{(1)} \in \mathbb{R}^{3 \times 1}$.

The total number of learnable parameters is:

$$\begin{aligned}\text{Total Parameters} &= \text{Weights and Biases of hidden layer} + \text{Weights and Biases of output layer}\\ &= (4N_H + N_H) + (3N_H + 3)\\ &= 8N_H + 3\end{aligned}$$

2. Derivative of Total Loss

Network Architecture:

- Total number of training samples: m = 280.
- Number of neurons in each layer:
  - Input layer: $n_0 = 4$ neurons.
  - Hidden layer: $n_1 = N_h$ neurons
  - Output layer: $n_2 = 3$ neurons
- Matrix of pre-activated neurons in each layer for **1 sample:**
  - Hidden layer: z_1 with shape $(n_1 \times 1) = (N_h \times 1)$
  - Output layer: z_2 with shape $(n_2 \times 1) = (3 \times 1)$
- Matrix of activated neurons in each layer for **1 samples**:
  - Input layer: a_0 (or x) with shape $(n_0 \times 1) = (4 \times 1)$
  - Hidden layer: a_1 with shape $(n_1 \times 1) = (N_h \times 1)$
  - Output layer: a_2 with shape $(n_2 \times 1) = (3 \times 1)$

- Matrix of weights for **1 sample**:
  - Hidden layer: w_1 with shape $(n_1 \times n_0) = (N_h \times 4)$
  - Output layer: w_2 with shape $(n_2 \times n_1) = (3 \times N_h)$
- Matrix of bias for **1 samples**:
  - Hidden layer: b_1 with shape $(n_1 \times 1) = (N_h \times 1)$
  - Output layer: b_2 with shape $(n_2 \times 1) = (3 \times 1)$
- The correct classification of **1 samples**:
  - y with shape $(n_2 \times 1) = (3 \times 1)$

The Loss function of 1 sample (with output $\tilde{y}^s$) with 1 correct classification y^s of that sample:

$$Loss(\tilde{y}^s, y^s) = -\sum_{i=1}^{n_2} y_i^s \ln(\tilde{y}_i^s)$$

And the total Loss for all 280 samples (aka Cost function) is simply the average of all Loss:

$$\text{Cost} = \frac{1}{280}\sum_{i=1}^{280} Loss(\tilde{y}^i, y^i)$$

So we calculate the cost of the whole dataset in 1 iteration, reduce the cost using gradients, and repeat until the Cost is at minimum possible.

Gradients for Output Layer

1. Derivative of the Loss with respect to the output's weight w_2:
The Loss function is given by:

$$Loss(\tilde{y}^s, y^s) = -\sum_{i=1}^{n_2} y_i^s \ln(\tilde{y}_i^s)$$

Since y^s is the correct classification of the s-th sample, it is a constant, so the only variable is $\tilde{y}^s$, which is a_2. So the Loss function is depended on a_2.
But the function for a_2 is given by:

$$a_2 = \text{Softmax}(z_2)$$

So a_2 is depended on z_2. But z_2 is given by:

$$z_2 = w_2.a_1 + b_2$$

So z_2 is depended on w_2 (as well as a_1 and b_2).
Using the chain rule for derivative, the derivative of the Loss function w.r.t w_2 is then given by:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{w}_2} = \frac{\partial \text{Loss}}{\partial \boldsymbol{a}_2}.\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2}.\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_2}$$

Let's first look at:

$$\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_2} = \frac{\partial}{\partial \boldsymbol{w}_2}(w_2.a_1 + b_2) = a_1$$

Now let's look at:

$$\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2}$$

This one is more tricky, because this uses the Softmax function, which is applied individually for each neuron of the Output layer, so we can's calculate the derivative of the whole matrix like $(\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_2})$ above. Instead, we have to consider derivative of each element of the matrix.

As mentioned above, the shape of z_2 and a_2 is both (3×1), so this derivative is a matrix of derivative of shape (3×3): (let a_2 be A and z_2 be Z for now for easier notation):

$$\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2} = \frac{\partial \boldsymbol{A}}{\partial \boldsymbol{Z}} = \begin{bmatrix} \frac{\partial A_1}{\partial Z_1} & \frac{\partial A_1}{\partial Z_2} & \frac{\partial A_1}{\partial Z_3} \\ \frac{\partial A_2}{\partial Z_1} & \frac{\partial A_2}{\partial Z_2} & \frac{\partial A_2}{\partial Z_3} \\ \frac{\partial A_3}{\partial Z_1} & \frac{\partial A_3}{\partial Z_2} & \frac{\partial A_3}{\partial Z_3} \end{bmatrix}$$

(For example: A_1 means the first activated neuron of the Output layer, Z_2 means the second pre-activated neuron of the Output layer)
The formula for the derivative of A_i and Z_j is as follow: $(0 \leq i, j \leq 3)$

$$\frac{\partial \boldsymbol{A}_i}{\partial \boldsymbol{Z}_j} = \frac{\partial}{\partial \boldsymbol{Z}_j}(\text{Softmax}(z_i)) = \frac{\partial}{\partial \boldsymbol{Z}_j}(\frac{e^{z_i}}{\sum_{k=1}^{n_2} e^{z_k}})$$

Note that for this function, only Z_j is treated as variable, everything else will be constant. This means Z_i is also a constant, unless $(i = j)$. So we have 2 cases:
Case 1: i $\neq$ j

$$\frac{\partial \boldsymbol{A}_i}{\partial \boldsymbol{Z}_j} = -\boldsymbol{A}_i.\boldsymbol{A}_j$$

Case 2: i = j

$$\frac{\partial \boldsymbol{A}_i}{\partial \boldsymbol{Z}_j} = \boldsymbol{A}_i.(1 - \boldsymbol{A}_i)$$

Let's now look at:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{a}_2}.\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2} = \frac{\partial \text{Loss}}{\partial \boldsymbol{z}_2} = \frac{\partial \text{Loss}}{\partial \boldsymbol{Z}}$$

Again, Softmax of a_2 makes the derivative of Loss function also tricky, because we can't calculate the derivative of the whole matrix at once, instead we have to look at each individual derivative, hence we using term Z again.
Let's take a look at derivative of the Loss function w.r.t 1 neuron Z_i:

$$\begin{aligned} \frac{\partial \text{Loss}}{\partial \boldsymbol{Z}_i} &= \frac{\partial}{\partial \boldsymbol{Z}_i}(-\sum_{k=1}^{n_2} y_k \ln(A_k)) \\ &= \frac{\partial}{\partial \boldsymbol{Z}_i} - (y_i \ln(A_i) + \sum_{k \neq i}^{n_2} y_k \ln(A_k)) \\ &= \frac{-y_i}{A_i}.\frac{\partial A_i}{\partial \boldsymbol{Z}_i} - \sum_{k \neq i}^{n_2} \frac{y_k}{A_k}.\frac{\partial A_k}{\partial Z_i} \end{aligned}$$

The $\frac{\partial A_i}{\partial \boldsymbol{Z}_i}$ is case 2, and the $\frac{\partial A_k}{\partial \boldsymbol{Z}_i}$ is case 1. So we have:

$$\begin{aligned} \frac{\partial \text{Loss}}{\partial \boldsymbol{Z}_i} &= \frac{-y_i}{A_i}.(\boldsymbol{A}_i.(1 - \boldsymbol{A}_i)) - \sum_{k \neq i}^{n_2} \frac{y_k}{A_k}.(-\boldsymbol{A}_k.\boldsymbol{A}_i) \\ &= A_i - y_i \end{aligned}$$

So the derivative of the Loss function w.r.t the whole Z is then:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{Z}} = A - y$$

And remember that Z and A are actually z_2 and a_2:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{z_2}} = a_2 - y$$

Finally, the derivative of the Loss function w.r.t output's weight w_2 is given as:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{w}_2} = \frac{\partial \text{Loss}}{\partial \boldsymbol{a}_2} . \frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2} . \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_2} = \frac{\partial \text{Loss}}{\partial \boldsymbol{z}_2} . \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_2} = (a_2 - y).a_1$$

This is however only for 1 sample. The total Loss (Cost function) of all 280 samples with respect to w_2 would be:

$$\frac{\partial \text{Cost}}{\partial \boldsymbol{w}_2} = \frac{1}{280}(\boldsymbol{A}_2 - \boldsymbol{Y}) \cdot \boldsymbol{A}_1^T$$

Where:

- A_2 is a (3×280) matrix of softmax outputs for all observations. (Note: I am using capital A again here, but it is different to the capital A above during calculation)
- Y is a (3×280) matrix of one-hot encoded true labels.
- A_1^T is an $(280 \times N_h)$ matrix (transposed) of hidden layer activations for all observations.
- The result $(A_2 - Y) \cdot A_1^T$ is of shape $(3 \times N_h)$.

2. Derivative of the Loss function with respect to the output's bias b_2: As mentioned earlier, the Loss function is depended on a_2, which is depended on z_2, which is depended on b_2. Again, using the chain rule for derivatives, we get:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{b}_2} = \frac{\partial \text{Loss}}{\partial \boldsymbol{a}_2} . \frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2} . \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{b}_2}$$

We already calculated the first 2 terms so let's not go through that mess again and just reuse it:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{a}_2} . \frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2} = a_2 - y = dz_2$$

And for the 3rd term:

$$\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{b}_2} = \frac{\partial}{\partial \boldsymbol{b}_2}(w_2.a_1 + b_2) = 1$$

So together, the derivative of the Loss function with respect to b_2 is then:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{b}_2} = dz_2.1 = dz_2$$

And for the Cost function, note that $(dz_2 = a_2 - y)$ so dz_2 has shape of (3×1). To sum this loss across all 280 samples (DZ_2), we have to perform row-summation on the (3×280) matrix DZ_2:

$$\frac{\partial \text{Cost}}{\partial \boldsymbol{b}_2} = \frac{1}{280}\text{row-sum}(DZ_2)$$

The result of this Cost function has shape (3×1).

Gradients for Hidden Layer

1. Derivative of the Loss function w.r.t hidden layer's weight w_1:
Earlier we mentioned that z_2 is depended on a_1, but a_1 is given as follow:

$$a_1 = \text{SiLU}(z_1)$$

So a_1 is depended on z_1. And z_1 is given as follow:

$$z_1 = w_1.a_0 + b_1$$

So z_1 is depended on w_1 (as well as a_0 and b_1).
Using the chain rule for derivative, the derivative of the Loss function w.r.t w_1 is then given by:

$$\frac{\partial \text{Loss}}{\partial \boldsymbol{w}_1} = \frac{\partial \text{Loss}}{\partial \boldsymbol{a}_2} . \frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2} . \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_1} . \frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1} . \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{w}_1} = dz_2 . \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_1} . \frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1} . \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{w}_1}$$

Let's go through each one:

$$\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_1} = \frac{\partial}{\partial \boldsymbol{a}_1}(w_2.a_1 + b_2) = w_2$$
$$\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1} = \frac{\partial}{\partial \boldsymbol{z}_1}(\mathrm{SiLU}(z_1))$$
$$\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{w}_1} = \frac{\partial}{\partial \boldsymbol{w}_1}(w_1.a_0 + b_1) = a_0 = x$$

Let's now look closer at the second one, which is the derivative of the SiLU function w.r.t z_1. Here, SiLU has the same problem as Softmax function above, where it is applied to each element individually, so we have to consider the derivative of each element of z_1

$$\frac{\partial}{\partial \boldsymbol{z}_{1i}}(\mathrm{SiLU}(z_{1i})) = \sigma(z_{1i}) + z_{1i}\sigma(z_{1i})(1-\sigma(z_{1i})) = \sigma(z_{1i})(1 + z_{1i}(1-\sigma(z_{1i})))$$

When we apply this for all i-element of z_1, we can store the result in a Jacobian matrix. Since the SiLU activation is applied element-wise, the Jacobian of $SiLU(z_1)$ with respect to z_1 is a **diagonal matrix** because each element z_{1i} in the vector only affects the corresponding output element a_{1i}. Therefore:

$$\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1} = \mathrm{diag}(\mathrm{SiLU}'(\boldsymbol{z}_1))$$

Substituting these terms back into the chain rule (with some matrix transposed and the order adjusted so the matrix multiplication is valid):

$$\frac{\partial \mathrm{Loss}}{\partial \boldsymbol{w}_1} = \boldsymbol{w}_2^T \cdot dz_2 \cdot \mathrm{diag}(\mathrm{SiLU}'(\boldsymbol{z}_1)) \cdot \boldsymbol{x}^T.$$

And thus the Cost function would then be:

$$\frac{\partial \mathrm{Cost}}{\partial \boldsymbol{W}_1} = \frac{1}{280} \cdot \boldsymbol{W}_2^T \cdot DZ_2 \cdot \mathrm{SiLU}'(\boldsymbol{Z}_1) \cdot \boldsymbol{X}.$$

2. Derivative of the Loss function with respect to the output's bias b_2: As mentioned earlier, z_1 is also depended on b_1. Again, using the chain rule for derivatives, we get:

$$\frac{\partial \mathrm{Loss}}{\partial \boldsymbol{w}_1} = \frac{\partial \mathrm{Loss}}{\partial \boldsymbol{a}_2}.\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2}.\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_1}.\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1}.\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{b}_1} = dz_2.\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_1}.\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1}.\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{b}_1}$$

We already calculated everything except for the last term:

$$dz_2.\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_1}.\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1} = \boldsymbol{w}_2^T \cdot dz_2 \cdot \mathrm{diag}(\mathrm{SiLU}'(\boldsymbol{z}_1)) = dz_1$$

And for the last term:

$$\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{b}_1} = \frac{\partial}{\partial \boldsymbol{b}_1}(w_1.a_0 + b_1) = 1$$

So together, the derivative of the Loss function with respect to b_1 is then:

$$\frac{\partial \mathrm{Loss}}{\partial \boldsymbol{b}_2} = dz_1.1 = dz_1$$

And for the Cost function:

$$\frac{\partial \mathrm{Cost}}{\partial \boldsymbol{b}_1} = \frac{1}{280}\text{row-sum}(DZ_1)$$

The result of this Cost function has shape $(N_h \times 1)$.

Summary of Gradients

$$\frac{\partial \text{Cost}}{\partial \boldsymbol{w}_2} = \frac{1}{280} DZ_2 \cdot \boldsymbol{A}_1^T$$
$$\frac{\partial \text{Cost}}{\partial \boldsymbol{b}_2} = \frac{1}{280} \text{row-sum}(DZ_2)$$
$$\frac{\partial \text{Cost}}{\partial \boldsymbol{W}_1} = \frac{1}{280} \cdot DZ_1 \cdot \boldsymbol{X}.$$
$$\frac{\partial \text{Cost}}{\partial \boldsymbol{b}_1} = \frac{1}{280} \text{row-sum}(DZ_1)$$

Where:

$$DZ_2 = \boldsymbol{A}_2 - \boldsymbol{Y}$$
$$DZ_1 = \boldsymbol{W}_2^T \cdot DZ_2 \cdot \text{SiLU}'(\boldsymbol{Z}_1)$$

And:

$$\text{SiLU}'(z_{1i}) = \sigma(z_{1i})(1 + z_{1i}(1 - \sigma(z_{1i})))$$

Gradient Methods Discussion

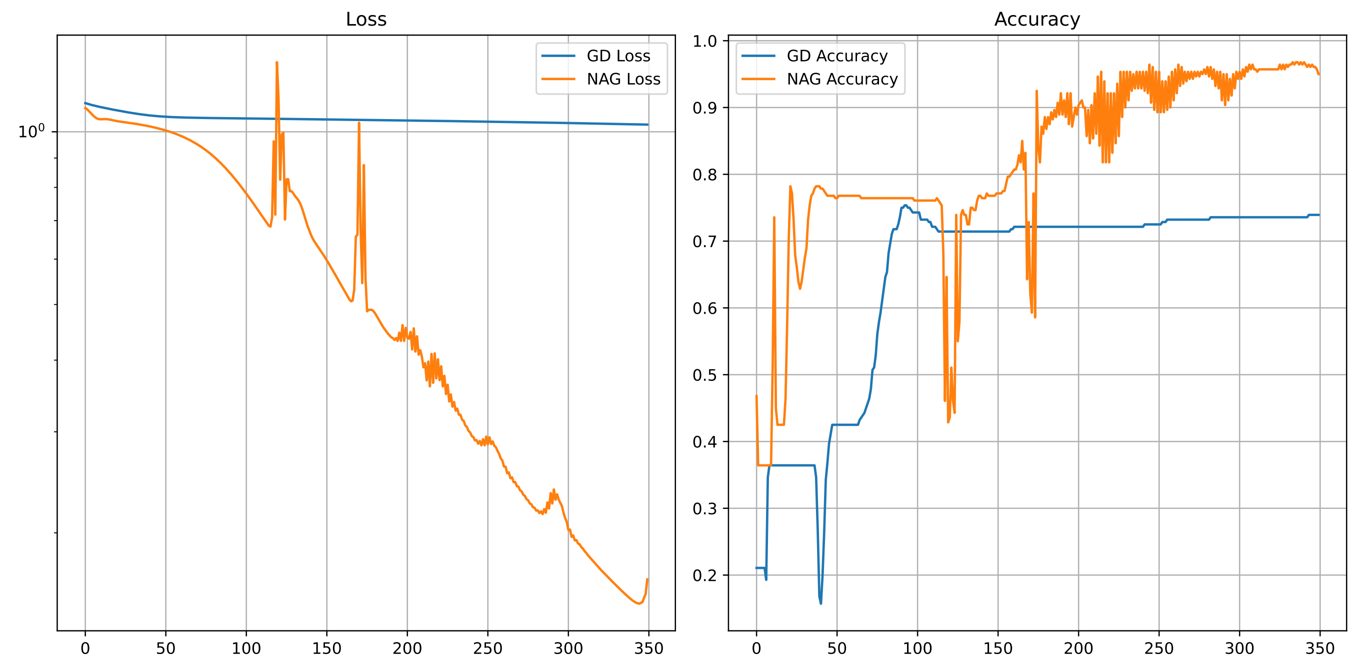

Figure 1: Plots of Loss and Accuracy

Let's begin with the loss curves. The loss measures how well our output matches the expected output. A good model would have a decreasing loss as training progresses, which would indicate improved predictions over time.

If we observe the loss curve for the Gradient Descent Method, it appears that loss is generally decreasing at a steady but slow rate. On the other hand if we observe the Nesterov Accelerated Gradient we can see that it is converging much faster, because of it's look ahead step. NAG shows faster convergence than GD.

As for the accuracy curves, they measure how often the predicted class matches the true class. In the case of GD the accuracy has a steady increase with a final plateau. NAG, as expected, reaches higher accuracy more quickly than standard GD. THe accuracy of GD is lower than NAG which could indicate that GD is not exploring the parameter space as efficiently. The learning rate in our case is 0.01 with a momentum of 0.9. The final accuracy for NAD is cca. 95% and for GD is cca. 74%.