# CS5242 Neural Networks and Deep Learning

Semester 1, AY2022-23
Homework 3

Agrawal, Shubhankar
A0248330L

September 2022

## 1 Question 1

### 1.1 Learning Rate Analysis

The Figure 1 shows how the loss varies across epochs for a given learning process. As we progress through epochs, our weights are updated using a gradient descent process which aims to minimize loss.

- At each step the gradient descent updates the weight using the learning rate as a controlled hyper-parameter.
- Thus the larger the learning rate, the larger our weight updates are.
- At each weight update step, the loss is calculated along a gradient descent curve, the value of which is used to calculate the magnitude and direction of the update.
- As the loss aims to move along the gradient descent curve over epoch iterations to a minimum, a large learning rate would overshoot the optimum lowest loss value and lead to weight updates that would increase the loss value at each epoch.
- The loss would keep increasing due to the large learning rate and thus generate such a curve.

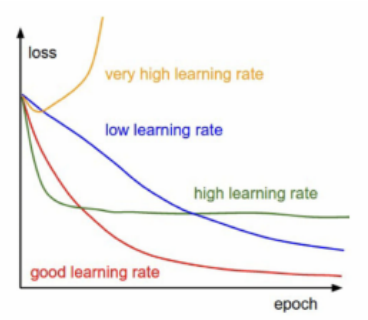



Figure 1: Loss vs Epoch

### 1.2 Learning Rate Schedule Analysis

The Figure 2 shows the loss value across epochs as we use a learning rate scheduler along with our gradient descent.

- A scheduler such as the one showed is often used with learning rates on gradient descent to optimize the learning process.
- The image shows the scheduler reducing the learning rate at two points around the 15th and 30th iteration as we can see a sudden drop in the error curve.
- This scheduler could be a step based scheduler which reduces learning rate on every few epochs. Another possibility is of being a plateau based scheduler which reduces learning rate when the error value starts to plateau.
- As the curve plateaus before these points, it indicates that the gradient descent was able to adjust weights close to the optimum value, but kept oscillating due to the learning rate. The reduction of learning rate thus helped move it closer to the optimum point.

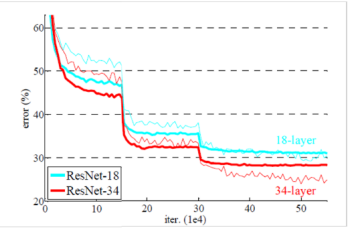
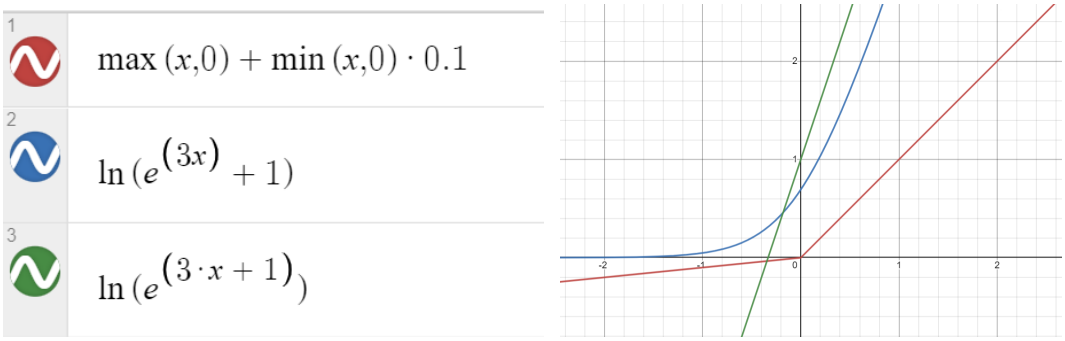



Figure 2: Loss vs Iteration

- By keeping the learning rate constant throughout the whole process, we might always overshoot the minimum (if the learning rate is too large) or progress too slowly (if the learning rate is too small).
- Thus, we would want to start off with a larger learning rate and reduce our learning rate as we progress through iterations and move closer towards the best/optimum loss for the gradient process.
- This would help in taking larger steps during the initial steps of the process and later take smaller steps to obtain the closest value corresponding to the best loss.
- We can thus have a more efficient learning process for the neural network by not overshooting and converging towards the best values for weights.

## 2 Question 2

Three activation functions are presented and plotted in the following Figures 3a and 3b. The graph contains the line of the 3 functions across the positive and negative ranges to get an idea of how the curves appear. The colors indicated in the legend correspond to the colors of the graph lines.




(a) Formulae (b) Graphs

Figure 3: Activation Functions

Each of the activation function formulae are summarized in Table 1 below.

| No. | Formula | Simplified | Suitable | Gradient Vanishing | Comments |
|---|---|---|---|---|---|
| 1) | $max(x,0) + min(x,0) * 0.1$ | $max(0.1x, x)$ | Yes (Hidden layers) | No | Leaky ReLU |
| 2) | $ln(e^{3x} + 1)$ | | Yes | Yes | Similar to Softplus |
| 3) | $ln(e^{3x+1})$ | $3x + 1$ | No | No | Linear |

Table 1: Activation Functions Summary

- As per the results from the activation functions above, we can see that not all are suitable or prone to gradient vanishing problems.

1. The first activation function, which is a version of the Leaky ReLU is a suitable piecewise activation function for the hidden layers of a Neural network. It should not be used in the output layer as it is unbounded. It does not suffer from vanishing gradient as the derivative does not converge to 0.
2. The second activation function is a suitable function since it is continuous and differentiable at all points. However, it is also susceptible to the vanishing gradient problem as the gradient (slope) tends to 0 as the value of x becomes small on the negative side.
3. The third activation function simplified to a linear $3x + 1$ is not a suitable activation function as the derivative is a constant (3) not related to the value of x and does not have a factor for x. It is equivalent to a normal weight multiplication. It does not suffer from the vanishing gradient problem as the derivative remains constant.

# 3 Question 3

The computational graph in Figure 4 is shown here. The equation of the computation is given by $c = a^2 * (a + b)$
If the edges from the power function and the addition function towards the multiplication are labelled as $v_1$ and $v_2$ respectively, then we calculate our derivatives with the following formulae. We are also given the values of inputs $a = -1$ and $b = 4$.

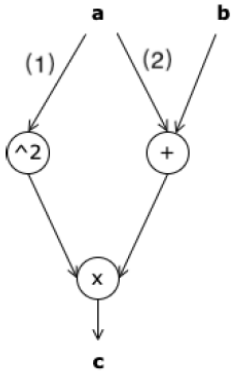



Figure 4: Computation

$\frac{\partial c}{\partial v_1} = v_2 = a + b$
$\frac{\partial c}{\partial v_2} = v1 = a^2$
$\frac{\partial v_1}{\partial a} = \frac{\partial a^2}{\partial a} = 2a$
$\frac{\partial v_2}{\partial a} = \frac{\partial a+b}{\partial a} = 1$
$\frac{\partial v_2}{\partial b} = \frac{\partial a+b}{\partial b} = 1$

Calculating the gradient of c with respect to a:

$\frac{dc}{da} = \frac{d(a^2*(a+b))}{da} = \frac{d(a^2)}{da} * (a + b) + a^2 * \frac{d(a+b)}{da}$

$\frac{dc}{da} = 2a * (a + b) + a^2 = 3a^2 + 2ab = 3(-1)^2 + 2(-1)(4) = -5$

The derivatives at points (1) and (2) would be as follows:

$\frac{\partial c}{\partial a(1)} = \frac{\partial c}{\partial v_1}\frac{\partial v_1}{\partial a} = (a + b) * 2a = (-1 + 4) * 2 = -6$
$\frac{\partial c}{\partial a(2)} = \frac{\partial c}{\partial v_2}\frac{\partial v_2}{\partial a} = a^2 * 1 = 1$

# 4 Question 4

We develop an MLP for a binary classification problem with input x belongs to $R^{100\times1}$. There are 2 hidden layers and one output layer. No bias is added and the ReLU activation function is used. The output layer uses logistic regression.

## 4.1 Hidden Layer 1: 100 units, Hidden Layer 2: 20 units

- From the input to hidden layer 1 there are $100 \times 100 = 10000$ weights
- From hidden layer 1 to hidden layer 2 there are $100 \times 20 = 2000$ weights
- From hidden layer 2 to the output (logistic regression) there are 20 weights
- The total number of parameters is $10000 + 2000 + 20 = 12020$

## 4.2 Hidden Layer 1: 20 units, Hidden Layer 2: 100 units

- From the input to hidden layer 1 there are $100 \times 20 = 2000$ weights
- From hidden layer 1 to hidden layer 2 there are $20 \times 100 = 2000$ weights
- From hidden layer 2 to the output (logistic regression) there are 100 weights
- The total number of parameters is $2000 + 2000 + 100 = 4100$

# 5 Question 5

Figure 5 shows a function to be minimized over gradient descent where learning rate $a = 0.3$ and $h > a$. We take that the first descent is represented by a right triangle with sides of length $(1, 1, \sqrt{2})$ and the bump is represented by an isoceles triangle with sides $(\sqrt{2}h, \sqrt{2}h, 2h)$ where 2h is the base and h is the height. We assume beyond the bump at h, the equation of the line changes again to and increases slope (closer to 0). We are looking to calculate the minimum value after 1000 steps. We are also given $L = 1$

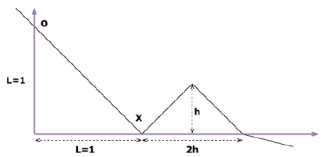



Figure 5: Function to minimize

## 5.1 Lowest value of f

In order to calculate the lowest value of loss function (f), we get the equation for the piece wise loss function over the weights.

$$f(w) = \begin{cases} L - w & 0 \leq w \leq L \\ w - L & L \leq w \leq L + h \\ 2h + L - w & L + h \leq w \leq L + 2h \\ h + \frac{L}{2} - \frac{w}{2} & L + 2h \leq w \end{cases}$$

The gradients of the piece wise function would be as follows.

$$f^{'}(w) = \begin{cases} -1 & 0 \leq w \leq L \\ 1 & L \leq w \leq L + h \\ -1 & L + h \leq w \leq L + 2h \\ -\frac{1}{2} & L + 2h \leq w \end{cases}$$

We know that the gradient descent weight update is given by $w_{t+1} = w_t - \alpha * f^{'}(w)$ where $\alpha = 0.3$
Therefore at each step, our point 'o' would move along the x axis by a value of $0.3 * f^{'}(w)$
Table 2 shows values for the first few steps. At step 0, we start from position o.

| Step | Weight | Loss | New Gradient | Update |
|---|---|---|---|---|
| 0 | 0 | 1 | -1 | 0.3 |
| 1 | 0.3 | 0.7 | -1 | 0.3 |
| 2 | 0.6 | 0.4 | -1 | 0.3 |
| 3 | 0.9 | 0.1 | -1 | 0.3 |
| 4 | 1.2 | 0.2 | 1 | -0.3 |
| 5 | 0.9 | 0.1 | -1 | 0.3 |
| | | ... | | |
| 1000 | 1.2 | 0.2 | 1 | -0.3 |

Table 2: Gradient Descent Step Values

Thus it appears that after the first 3 iterations, the weights would constantly oscillate with values $0.9, 1.2$ around the lowest loss. While at the $1000^{th}$ step, the weight would be 1.2 with loss 0.2, the lowest loss is reached at the oscillation when the weight is at 0.9 with loss 0.1 (the minimum loss).

## 5.2 ADAM Optimizer (Bonus)

The parameters of $L = 1$ and $a = 0.001$ are maintained along with the ADAM parameters of $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 0$

The ADAM optimizer is programmed and tested for values of h in the range $(a, 4a) = (0.001, 0.004)$ with a step size of $0.1a = 0.0001$. The experiment is run for 5000 iterations using the same gradient function as defined above. In order to escape the local minimum at x, it needs to cross the peak at $w = L + h$.

The results of the experiment are summarized in Table 3

| h | Local Minimum Peak | Time Step Cross ($w > L + h$) | Final Weight (Step 5000) |
|---|---|---|---|
| 0.0010 | 1.0010 | 1002 | 4.3517 |
| 0.0011 | 1.0011 | 1002 | 4.3515 |
| 0.0012 | 1.0012 | 1002 | 4.3515 |
| 0.0013 | 1.0013 | 1002 | 4.3515 |
| 0.0014 | 1.0014 | 1002 | 4.3514 |
| 0.0015 | 1.0015 | 1003 | 4.3492 |
| 0.0016 | 1.0016 | 1003 | 4.3492 |
| 0.0017 | 1.0017 | 1003 | 4.3492 |
| 0.0018 | 1.0018 | 1003 | 4.3491 |
| 0.0019 | 1.0019 | 1004 | 4.3469 |
| 0.0020 | 1.0020 | 1004 | 4.3469 |
| 0.0021 | 1.0021 | 1004 | 4.3468 |
| 0.0022 | 1.0022 | 1005 | 4.3445 |
| 0.0023 | 1.0023 | 1005 | 4.3445 |
| 0.0024 | 1.0024 | 1006 | 4.3422 |
| 0.0025 | 1.0025 | - | 0.9998 |
| | | ... | |
| 0.0025 | 1.0025 | - | 0.9998 |

Table 3: Gradient Descent Step Values

After our experiments, we can see that the local minimum is escaped up till $h = 0.0024 = 2.4a$ for the small value of a. With a larger value of a, we might see a different maximum value of h for which a local minimum peak can be escaped. We can see that for $h \leq 0.0024$, the local minimum is escaped (peak crossed) after 1000 iterations and then the gradient descent continues along the direction towards the next local minimum. If the number of iterations were to be increased, the weights would continue increasing further as the loss keeps decreasing once we cross the local minimum. However, for values $h \geq 0.0024$ the final gradient stays around 0.999 which is our initial local minimum and cannot escape the peak.

The code for the following is attached as a Jupyter notebook along with the submission.