CS 148a Problem Set 1

Question 1

(a)

$$\begin{aligned} L &= w_3 s \\ \implies \frac{\partial L}{\partial w_3} &= s \\ &= 3.3 \end{aligned}$$

(b)

$$\begin{aligned} \frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial s}\frac{\partial s}{\partial w_1} \\ &= w_3(1) = w_3 \\ &= 5.7 \end{aligned}$$

By symmetry, we also get

$$\frac{\partial L}{\partial w_2} = w_3 = 5.7$$

.

(c)

$$\begin{aligned} w_1 &= w_1 - \eta\frac{\partial L}{\partial w_1} \\ &= 1 - 0.01(5.7) \\ &= 0.943 \\ w_2 &= w_2 - \eta\frac{\partial L}{\partial w_2} \\ &= 2.3 - 0.01(5.7) \\ &= 2.243 \\ w_3 &= w_3 - \eta\frac{\partial L}{\partial w_3} \\ &= 5.7 - 0.01(3.3) \\ &= 5.667 \end{aligned}$$

(d)

$$\begin{aligned} L &= (w_1 + w_2) \cdot w_3 \\ &= (0.943 + 2.243) \cdot 5.667 \\ &= 18.055 \end{aligned}$$

The loss decreased. This is expected, since the gradient descent update we performed moves each parameter in the direction opposite to the gradient of the loss with respect to that parameter, which locally decreases the loss.

Question 2

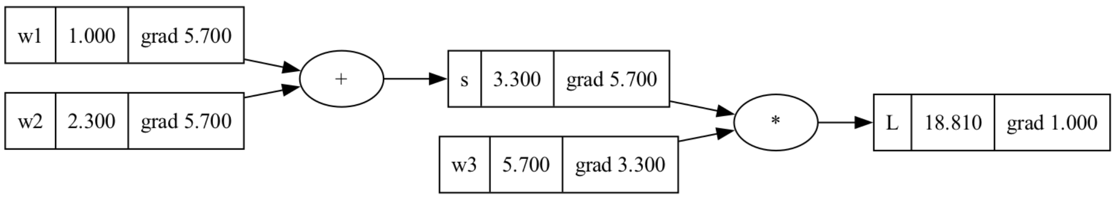


Figure 1: Network diagram before backprop

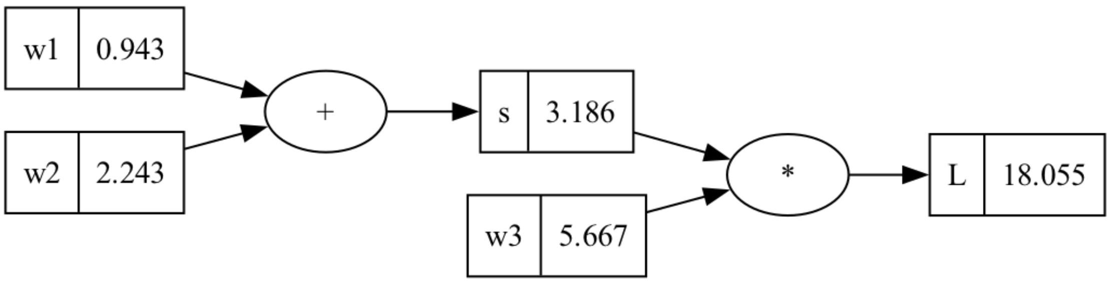


Figure 2: Network diagram after one backpropagation

Question 3

I trained a three-layer network for 10 epochs with the AdamW optimizer and a learning rate of 0.001 on cross-entropy loss, and achieved over 85% validation accuracy.

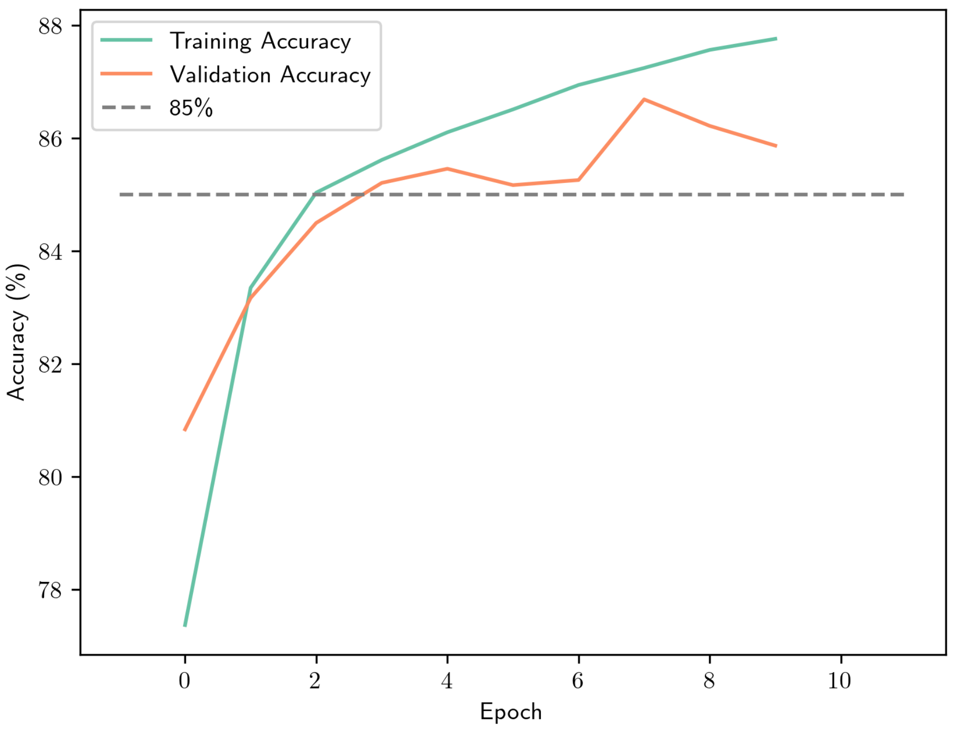


Figure 3: Accuracy over epochs

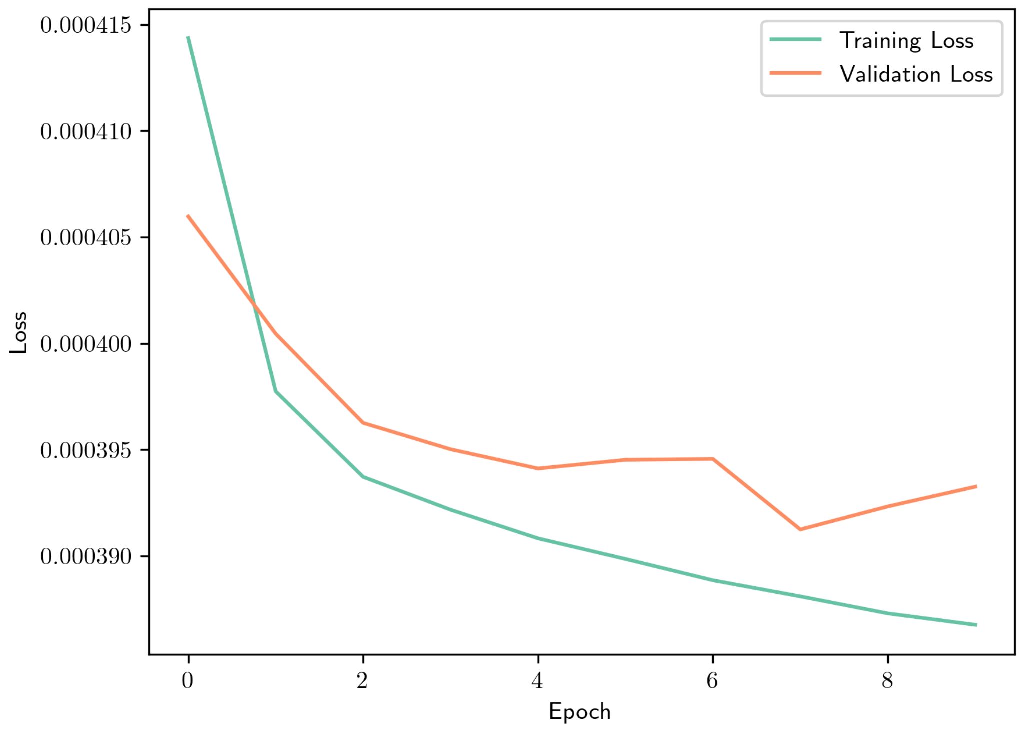


Figure 4: Loss over epochs

Question 4

1. *Proof.*

$$\begin{aligned}\tanh(x)+1 &= \frac{e^x-e^{-x}}{e^x+e^{-x}}+1\\ &= \frac{e^x-e^{-x}+e^x+e^{-x}}{e^x+e^{-x}}\\ &= \frac{2e^x}{e^x+e^{-x}}\\ &= 2\frac{1}{1+e^{-2x}}\end{aligned}$$

But

$$2\sigma(2x)=2\frac{1}{1+e^{-(2x)}}$$

So $\tanh(x)+1=2\frac{1}{1+e^{-2x}}=2\sigma(2x)$. □

2. *Proof.* In the previous part, we showed that tanh and σ were equivalent up to affine transformations. These can be expressed using the weights and biases of the following layer in a multi-layer perceptron.

We show specifically how this equivalency holds in both possible cases.

- Expressing a sigmoid neuron with tanh: from the previous proof,

$$\sigma(z)=\frac{1}{2}\left[\tanh\left(\frac{z}{2}\right)+1\right].$$

So, if the input to the neuron is $\mathbf{w}^T\mathbf{x}+b$, then

$$\sigma(\mathbf{w}^T\mathbf{x}+b)=\frac{1}{2}\left[\tanh\left(\frac{\mathbf{w}^T\mathbf{x}+b}{2}\right)+1\right]$$

The RHS can be expressed with a tanh neuron with weights $\mathbf{w}'=\frac{1}{2}\mathbf{w}^T$ on the inputs and a bias $b'=\frac{b}{2}$. This value can then be transformed in the following layer of the multi-layer perceptron with weight 0.5 associated with it every time it is used as an input to a node, and an additional bias of 0.5.

- Expressing a tanh neuron with sigmoid: from the previous proof,

$$\tanh(z)=2\sigma(2z)-1$$

So if the input is $\mathbf{w}^T\mathbf{x}+b$, then

$$\tanh(\mathbf{w}^T\mathbf{x}+b)=2\sigma(2\mathbf{w}^T+2b)-1$$

This can be expressed with a σ neuron with weights $\mathbf{w}'=2\mathbf{w}$ and a bias $b'=2b$. In the next layer, the weights associated with this value would be set to 2, followed by a bias of -1 to create equivalency.

Therefore, any network using one activation can be converted layer-by-layer into a network using the other activation by tuning the weights and biases.

Note that this requires there to be at least two layers in the network – a single-neuron model would not be sufficient. □

3. *Proof.* Let the dataset be $\mathcal{D}=(x_i,\mathbf{y}_i)_{i=1}^N$, with $\mathbf{y}_{i_j}=1 \iff x_i\in C_j$ (i.e., the labels are one-hot encoded). Let $|C|$ be the number of classes.

Let the output of the network for $\mathbf{x}_i$ be $\hat{\mathbf{y}}_i$, a vector in which the j-th value corresponds to the probability of $\mathbf{x}$ belonging to class j.

Define the model likelihood $P(\hat{\mathbf{y}}=\mathbf{y}_i|\mathbf{x},\theta)$ as the probability that the output of the model $\hat{\mathbf{y}}$, using parameters θ, is correct on input x. Then, assuming independence of predictions, the likelihood of the model on the data will be,

$$\prod_{i=1}^{N}P(\hat{\mathbf{y}}_i=\mathbf{y}_i|\mathbf{x}_i)$$

So the log-likelihood will be:

$$\begin{aligned}\mathcal{L}_{\log}(\theta) &= \sum_{i=1}^{N} \log p(\hat{\mathbf{y}}_i = \mathbf{y}_i | \mathbf{x}_i, \theta) \\ &= \sum_{i=1}^{N} \mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i)\end{aligned}$$

e dot product with the one-hot encoded true vector to zero out every entry except for the one at the index of the correct label)

$$= \sum_{i=1}^{N} \sum_{j=1}^{|C|} y_{i_j} \cdot \log(\hat{y}_{i_j})$$

So

$$\arg\max_{\mathbf{w}} \sum_{i=1}^{N} \sum_{j=1}^{|C|} \mathbf{y}_{i_j} \cdot \log(\hat{\mathbf{y}}_{i_j}) = \arg\min_{\mathbf{w}} \left[-\sum_{i=1}^{N} \sum_{j=1}^{|C|} y_{i_j} \cdot \log(\hat{y}_{i_j}) \right]$$

But cross-entropy loss is defined as

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N} \sum_{k=1}^{|C|} y_{i_k} \cdot \log(\hat{y}_{i_k})$$

.

So maximizing log likelihood is equivalent to minimizing cross-entropy loss.

□

4. *Proof.* In Adam, we normalize the gradient by the square root of its variance to ensure constant scale for all parameter updates, such that

$$w_{t+1,i} = w_{t,i} - \eta \frac{\bar{s}_{t+1,i}}{\sqrt{\bar{r}_{t+1,i} + \delta}}$$

where

$$g_{t,i} = \frac{\partial \mathcal{L}(w_{t,i})}{\partial w_i}$$

$$s_{t+1,i} = \beta_1 s_{t,i} + (1-\beta_1) g_t \qquad \bar{s}_{t+1,i} = \frac{s_{t+1,i}}{1-\beta_1^{t+1}} \tag{1}$$

$$r_{t+1,i} = \beta_2 r_{t,i} + (1-\beta_2) g_{t,i}^2 \qquad \bar{r}_{t+1,i} = \frac{r_{t+1,i}}{1-\beta_2^{t+1}} \tag{2}$$

We require that, in expectation, the factor $s_{t+1,i}$ is equal to the mean of the gradient for that parameter $\mu = \mathbb{E}[g_i]$.

$$\begin{aligned} s_{t+1,i} &= (1-\beta_1) \sum_{k=1}^{t} \beta_1^{t-k} g_{k,i} && \text{(unrolling the recurrence)} \\ \implies \mathbb{E}[s_{t+1,i}] &= (1-\beta_1) \sum_{k=1}^{t} \beta_1^{t-k} \mathbb{E}[g_k] \\ &= (1-\beta_1)\mu \sum_{k=1}^{t} \beta_1^{t-k} && \text{(letting } \mu = \mathbb{E}[g_k]) \\ &= \mu(1-\beta_1)(1)\frac{1-\beta_1 tn}{1-\beta_1} && \text{(sum of geometric series)} \\ &= \mu(1-\beta_1^t) \end{aligned}$$

So to correct this, we use

$$\bar{s}_{t+1,i} = \frac{s_{t+1,i}}{1-\beta_1^t}$$

.

Similarly, we require $r_{t+1,i}$ in expectation to be equal to the variance $\mathbb{E}[g_i^2]$.

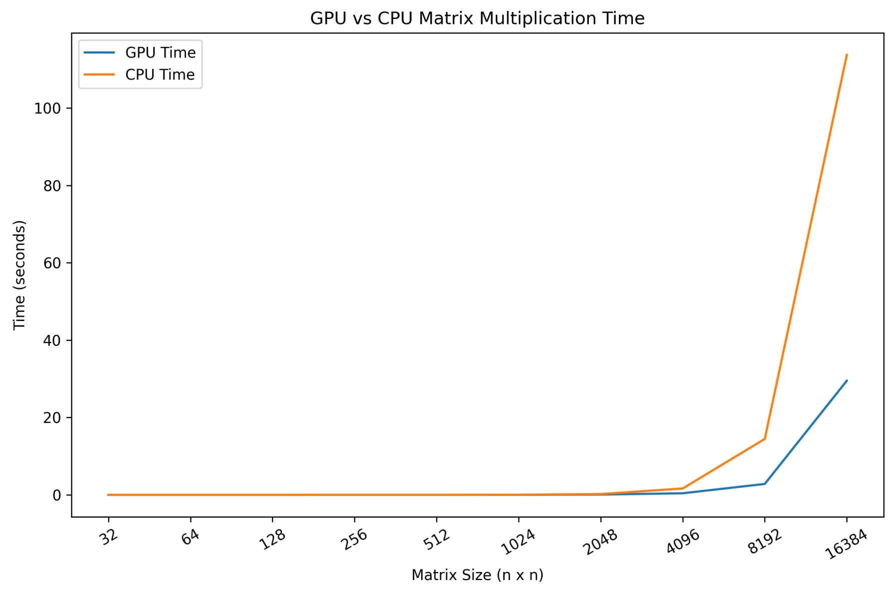
$$r_{t+1,i} = (1-\beta_2) \sum_{k=1}^{t} \beta_2^{t-k} g_{t,i}^2$$

$$\begin{aligned}
\implies \mathbb{E}[r_{t+1,i}] &= (1-\beta_2)\sum_{k=1}^{t}\beta_2^{t-k}\mathbb{E}[g_{t,i}^2] \\
&= \mathbb{E}[g_{t,i}^2](1-\beta_2)(1)\frac{1-\beta_2^t}{1-\beta_2} \\
&= (1-\beta_2^t)\mathbb{E}[g_{t,i}^2]
\end{aligned}$$

So we need to apply the same correction $\square$

Question 5

1. `doublify` takes a 0.001714 seconds on the kernel and 0.007752 seconds on the CPU for $N = 2^{12}$
2. See the figure below:



The GPU scales much better than the CPU, on a log scale, we see sharp changes in the slope of the CPU, which scales with a faster exponential coefficient with respect to the matrix size than the GPU does.

3.
   - Each thread (one kernel operation) only calculates one element in the resulting matrix C, despite loading a row and column from A and B which are reused for other multiplications. It might be more efficient to compute the results over spatially adjacent regions of A and B.
   - Each thread currently only outputs one element, but we could reduce the number of required threads by computing multiple values of C in the same kernel and thus reuse cached data as well as reduce the number of required kernels.

Question 6

1. We show that adding layers to a neural network without non-linear activation functions does not increase its expressive power.

   Let $\mathbf{z}_1 = W_1\mathbf{z}_0 + \mathbf{b}_1$.

   Then $\mathbf{z}_2 = W_2\mathbf{z}_1 + \mathbf{b_2} = W_2(W_1\mathbf{z}_0 + \mathbf{b_1}) + \mathbf{b_2} = W_2W_1\mathbf{z}_0 + (W_2\mathbf{b_1} + \mathbf{b_2})$. But if we define $W' = W_2W_1$ and $\mathbf{b'} = W_2\mathbf{b_1} + \mathbf{b_2}$, then we can express the value of the output of this second linear layer z_2 as a single linear transformation $W'\mathbf{z} + '$. So adding additional layers does not increase expressive power, since each additional layer only represents an affine transformation.

   By induction, we can show that any D-layer network without nonlinearities can be collapsed down to one linear layer.

2. Suppose $x \in \mathbb{R}^10$, $W_1 \in \mathbb{R}^{2\times10}$, $W_2 \in \mathbb{R}^{10\times2}$.

   Then $W' = W_2W_1 \in \mathbb{R}^{10\times10}$, but $\text{rank}W' \leq \text{rank}(W_1)$ since $\text{im}W' = \{W_2\mathbf{y} \mid \mathbf{y} \in \text{im}W_1\}$.

   Similarly, $\text{rank}W' \leq \text{rank}W_2$ since $\text{im}W' = \{W_2(W_1\mathbf{x}) \mid \mathbf{x} \in \mathbb{R}^{10}\} \subseteq \text{im}(W_2)$.

   This implies $\text{rank}W' \leq \min(\text{rank}(W_1), \text{rank}(W_2)) = 2$.

   So applying $W_2W_1\mathbf{x}$ maps from $\mathbb{R}^{10} \to \mathbb{R}^{10}$ but collapses down x to a 2-dimensional latent subspace, while a single layer $W_l \in \mathbb{R}^{10\times10}$ might have had more expressive power.

3. For AND, let $\mathbf{w} = [1, 1]$ and $b = -1.5$. For OR, let $\mathbf{w} = [1, 1]$ and $b = -0.5$. In both cases make the output $\text{sign}(\mathbf{w}^T\mathbf{x} + b)$.

Input $\mathbf{x}$	AND layer output	OR layer output
$(0, 0)$	$\text{sign}(-1.5) = -1$	$\text{sign}(-0.5) = -1$
$(0, 1)$ or $(1, 0)$	$\text{sign}(-0.5) = -1$	$\text{sign}(0.5) = 1$
$(1, 1)$	$\text{sign}(0.5) = 1$	$\text{sign}(1.5) = 1$

   XOR maps $(0, 1)$ and $(1, 0)$ to 1, and $(0, 0)$ as well as $(1, 1)$ to 0. In $\mathbb{R}^2$, these lie on a square, with positive examples on a line connecting one set of opposite corners of the square, and the negative examples on another line connecting the other two corners of the square.

   A single layer perceptron computes $\mathbf{w}^T\mathbf{x}+b$, which is a hyperplane in D-dimensional space; in 2-dimensional space, a single layer defines a line. For the perceptron to correctly classify XOR, all positive examples must lie on one side of this line and all negative examples on the other side of the line. However, no such line exists. Any line separating $(0, 0)$ from $(0, 1)$ and $(1, 0)$ must also separate $(0, 0)$ from $(1, 1)$, but $(0, 0)$ and $(1, 1)$ belong to the same class.

4. In the base case, with $D = 1$,

   $$y = \sum_{i=1}^{W_1} a_i \text{RELU}(\mathbf{w}^T x + b)$$

   Each RELU component corresponds to 2 pieces of a piecewise linear function, such that we can approximate a function f if $\kappa(f) \leq 2W_1$. This is exactly the required relationship for $D = 1$.

   Now, the inductive step: assume that $2^D \prod_{d=1}^{D} W_d$ pieces are sufficient to represent f.

   Then, if we add $D + 1$-th layer, the final output y will be

   $$y = \sum_{i=1}^{W_{D+1}} \alpha_i \text{RELU}(\mathbf{w}^T f(x) + b)$$

   $\mathbf{w}^T f(x) + b$ requires the same number of pieces as f due it only being an affine translation on f. The RELU gives us at most twice as many piecewise functions. So we need $2W_{D+1} \times 2^D \prod_{i=1}^{D} W_d = \times 2^{D+1} \prod_{i=1}^{D+1} W_d$ to represent the function, which is sufficient to show by induction that the required relationship holds.