CARNEGIE MELLON UNIVERSITY 10-315
HOMEWORK 5

DUE: Friday, Feb. 21, 2025

https://www.cs.cmu.edu/~10315

INSTRUCTIONS

- **Format:** Use the provided LaTeX template to write your answers in the appropriate locations within the *.tex files and then compile a pdf for submission. We try to mark these areas with STUDENT SOLUTION HERE comments. Make sure that you don't change the size or location of any of the answer boxes and that your answers are within the dedicated regions for each question/part. If you do not follow this format, we may deduct points.

  You may also digitally annotate the pdf. Illegible handwriting will lead to lost points. However, we suggest that try to do at least some of your work directly in LaTeX.

- **How to submit written component:** Submit to Gradescope a pdf with your answers. Again, make sure your answer boxes are aligned with the original pdf template.
- **How to submit programming component:** See Programming combonent for details on how to submit to the Gradescope autograder.
- **Policy:** See the course website for homework policies, including late policy, and academic integrity policies.
- **Proofs and Derivations:** For full credit, you must clearly show all nontrivial steps. Explicit reasons for each step are not required but can be helpful, especially if the step is not obvious.

Name	
Andrew ID	
Hours to complete all components (nearest hour)	

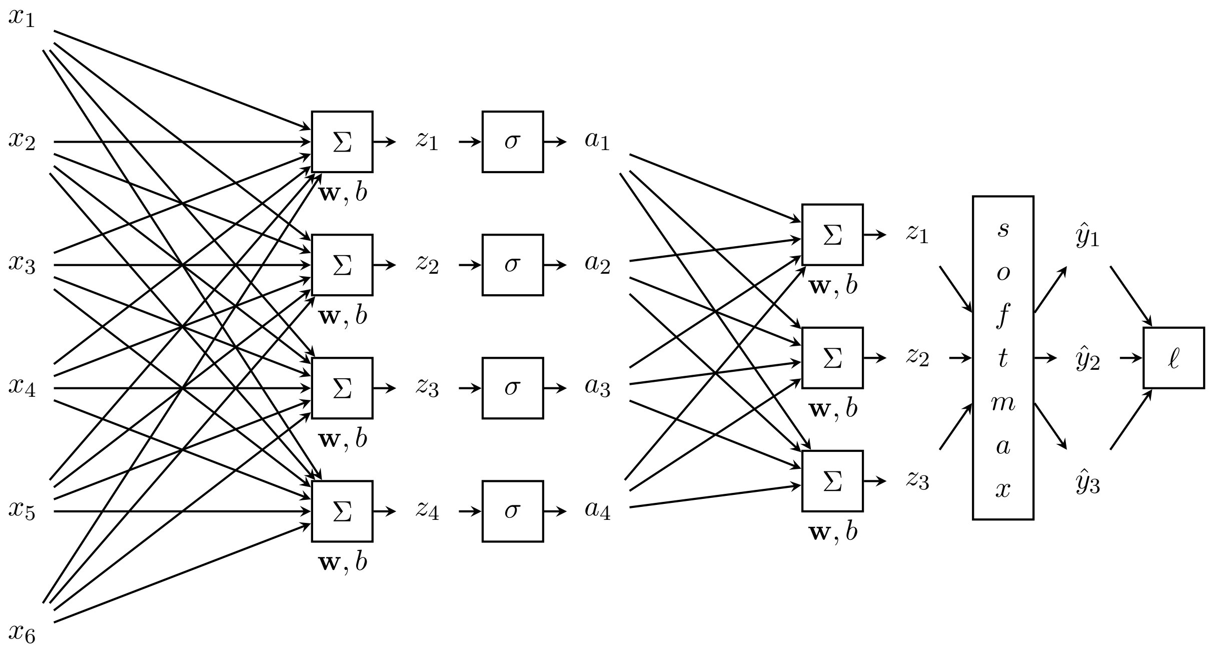
1 Backpropagation in Neural Network layers

In this part, you will derive the necessary backpropagation operations to efficiently implement a simple neural network. You will begin by deriving the formulas for the gradients of each operation in a neural network, including the linear layer, the sigmoid activation layer, and the softmax operator. In part 2, you will use these formulas to derive the gradients for a small neural network.

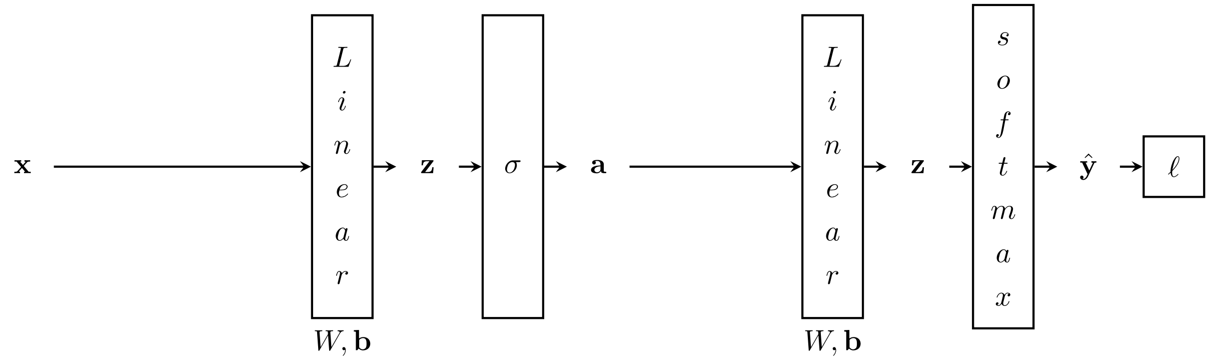
Note: In this assignment, your final answer should **not** contain any summations. You may have summations in the steps leading up to your final answer, but not in the final answer itself. This will be crucial for your implementations in the programming section of the assignment.

Example fully connected classification network

Network diagram (neuron view)



Network diagram (layer view)



Some notes on notation

In this assignment, we'll be using **denominator layout**.

Recall that the denominator layout for the gradient of a scalar y with respect to a N dimensional vector $\mathbf{x}$ is given by:

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_N} \end{bmatrix}$$

where $\mathbf{x}$ is a N dimensional vector.

If we have a scalar y and a $N \times M$ matrix X, the partial derivative of y with respect to X, $\frac{\partial y}{\partial X}$, in denominator format is:

$$\frac{\partial y}{\partial X} = \begin{bmatrix} \frac{\partial y}{\partial X_{1,1}} & \cdots & \frac{\partial y}{\partial X_{1,M}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial X_{N,1}} & \cdots & \frac{\partial y}{\partial X_{N,M}} \end{bmatrix}$$

1. **[15 pts] Backpropagation for a linear layer**

   Recall the formula for a linear layer that maps from M-dimensional space to N-dimensional space:

   $$\mathbf{z} = W\mathbf{x} + \mathbf{b}$$

   where:

   - $\mathbf{x}$ is the M-dimensional input vector (to this layer; not necessarily the input to the full network)
   - $\mathbf{z}$ is the K-dimensional output vector,
   - W is the weight matrix of size $K \times M$, and
   - $\mathbf{b}$ is the bias vector of dimension K.

   When we perform backpropagation on a linear layer, we are given $\frac{\partial J}{\partial \mathbf{z}}$, and we wish to use this to compute $\frac{\partial J}{\partial \mathbf{x}}$, $\frac{\partial J}{\partial W}$, and $\frac{\partial J}{\partial \mathbf{b}}$.

Continued on next page.

(a) Derive $\partial J/\partial \mathbf{x}$ in terms of $\partial J/\partial \mathbf{z}$, $\mathbf{x}$, $\mathbf{z}$, W, and $\mathbf{b}$. Note that you will need some but not all of these variables for your derivation.

Your derivation must include the derivation for a single entry in $\partial J/\partial \mathbf{x}$, i.e., $\partial J/\partial x_j$.

For full credit on this and **all following derivations and proofs**, you must clearly show all nontrivial steps. Explicit reasons for each step are not required but can be helpful, especially if the step is not obvious.

$\partial J/\partial \mathbf{x}$

(b) Derive $\partial J/\partial W$ in terms of $\partial J/\partial \mathbf{z}$, $\mathbf{x}$, $\mathbf{z}$, W, and $\mathbf{b}$. Note that you will need some but not all of these variables for your derivation.

Your derivation must include the derivation for a single entry in $\partial J/\partial W$; i.e. $\partial J/\partial W_{k,j}$.

Reminder that your final answer should **not** contain any summations and should be expressed using only matrix operations.

$\partial J/\partial W$

(c) Derive $\partial J/\partial \mathbf{b}$ in terms of $\partial J/\partial \mathbf{z}$, $\mathbf{x}$, $\mathbf{z}$, W, and $\mathbf{b}$. Note that you will need some but not all of these variables for your derivation.

Your derivation must include the derivation for a single entry in $\partial J/\partial \mathbf{b}$, i.e., $\partial J/\partial b_k$.

$\partial J/\partial \mathbf{b}$

2. **[8 pts] Backpropagation for sigmoid activation**

   Recall the vector version of logistic (sigmoid) activation function $\mathbf{a} = \sigma(\mathbf{z})$, where:

   $$a_k = \frac{1}{1 + e^{-z_k}}$$

   Also, recall the derivative for the scalar logistic function, $d\sigma/dz = \sigma(z)\,(1 - \sigma(z))$.

   Given $\partial J/\partial \mathbf{a}$, derive $\partial J/\partial \mathbf{z}$ in terms of $\partial J/\partial \mathbf{a}$, $\mathbf{a}$, and $\mathbf{z}$. Note that you will need some but not all of these variables for your derivation. You may use $\circ$ to denote element-wise product in your answer (the Hadamard Product).

   Your derivation must include the derivation for a specific entry in $\partial J/\partial \mathbf{z}$, i.e., $\partial J/\partial z_k$.

$\partial J/\partial \mathbf{z}$

3. **[20 pts] Backpropagation for softmax and cross-entropy loss**

Cross-entropy loss is often used when a network ends with a softmax layer. In this question, we'll see that it can be convenient to combine these two into a single layer because the gradient simplifies quite nicely.

Let's start with the softmax function, $\hat{\mathbf{y}} = g_{softmax}(\mathbf{z})$, where

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

where z_k is the input to the softmax for class k, and K is the total number of classes.

Looking at the dimensions of the partial derivative for the output of the softmax function with respect to its input, $\partial \hat{\mathbf{y}} / \partial \mathbf{z}$, we can see that it is a $K \times K$ matrix, where the diagonal elements are $\partial \hat{y}_k / \partial z_k$ and the off-diagonal elements are $\partial \hat{y}_k / \partial z_j$ for all $j \neq k$.

Before we combine the softmax with cross-entropy, derive the off-diagonal and diagonal entries of the softmax partial derivative. Note: You may choose to simplify these expressions here or wait until the following cross-entropy part.

Off-diagonal: $\partial \hat{y}_k / \partial z_j$

Diagonal: $\partial \hat{y}_k / \partial z_k$

Recall the cross-entropy loss:

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

where $\mathbf{y}$ is the true label (as a one-hot vector) and $\hat{y}_k$ is the predicted probability for class k, and K is the total number of classes.

Building on your answers to the previous part, prove that $\partial J / \partial \mathbf{z} = \hat{\mathbf{y}} - \mathbf{y}$.

Your derivation must include the derivation for a specific entry in $\partial J / \partial \mathbf{z}$, i.e., $\partial J / \partial z_j$.

$\partial J / \partial \mathbf{z}$

2 Programming Experiments

This part of the written assignment should be done after you complete the programming.

For all of the following experiments use the following baseline configuration:

- Batch size: 64
- Training data size: 30,000 points
- Validation data size: 5,000 points
- Network: Only one hidden layer (size to be specified below)
- Number of epochs: 20
- Learning rate: (specified below)

1. **[4 pts]** Display the validation data confusion matrix after training the 64-neuron network with learning rate 1.0 for **20 epochs**.

Confusion matrix

Which digit is misclassified the most? (Break any ties arbitrarily.)

For that digit, what digit is it most commonly confused as? (Break any ties arbitrarily.)

What do the diagonal elements represent?

Diagonal elements?

2. **[9 pts] Tune learning rate**

   Train your network with 64 neurons in the hidden layer and try learning rates 10.0, 1.0, and 0.1.

   On the following pages, include plots of your train/val loss and accuracy over epochs for each learning rate. Also include the visualized first layer weights for each learning rate.

   The `plot_training_history` function in `util.py` will be helpful.

   For each learning rate, does it make sense to invest time in continuing to train for more epochs, e.g. epochs 21-40? (Assume we don't change anything before we continue.) Why or why not? Be sure to comment on the results that you are providing on the following pages.

More epochs? Why or why not?: Learning rate 10

○ Yes ○ No

More epochs? Why or why not?: Learning rate 1.0

○ Yes ○ No

More epochs? Why or why not?: Learning rate 0.1

○ Yes ○ No

Loss and Accuracy: Learning rate 10.0

Loss and Accuracy: Learning rate 1.0

Loss and Accuracy: Learning rate 0.1

Weights: Learning rate 10.0

Weights: Learning rate 1.0

Weights: Learning rate 0.1

3. **[6 pts] Experiment with width of hidden layer**

Train your network with learning rate 0.1 and experiment with different numbers of neurons in the hidden layer. (Still just one hidden layer.) Specifically, try 4, 16, and 64 neurons.

On the following pages, include plots of your train/val loss and accuracy over epochs for each network architecture. Also include the visualized first layer weights for each.

Given these results after training for 20 epochs, which architecture performs the best and which performs the worst?

Best: ○ 4 hidden neurons ○ 16 hidden neurons ○ 64 hidden neurons

Worst: ○ 4 hidden neurons ○ 16 hidden neurons ○ 64 hidden neurons

What specifically did you use to determine which architecture is best?

Specific criteria

Comment on how visualizing these first layer weights can help us understand the effect of the number of neurons on performance.

Comment on weight visualization and number of neurons

Loss and Accuracy: 4 neurons

Loss and Accuracy: 16 neurons

Loss and Accuracy: 64 neurons (repeated figure)

Weights: 4 neurons

Weights: 16 neurons

Weights: 64 neurons (repeated figure)

3 Collaboration Policy

After you have completed all other components of this assignment, report your answers to the following collaboration questions.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details including names of people who helped you and the exact nature of help you received.

2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details including names of people you helped and the exact nature of help you offered.

3. Did you find or come across code that implements any part of this assignment? If so, include full details including the source of the code and how you used it in the assignment.