

Name:

NetID:

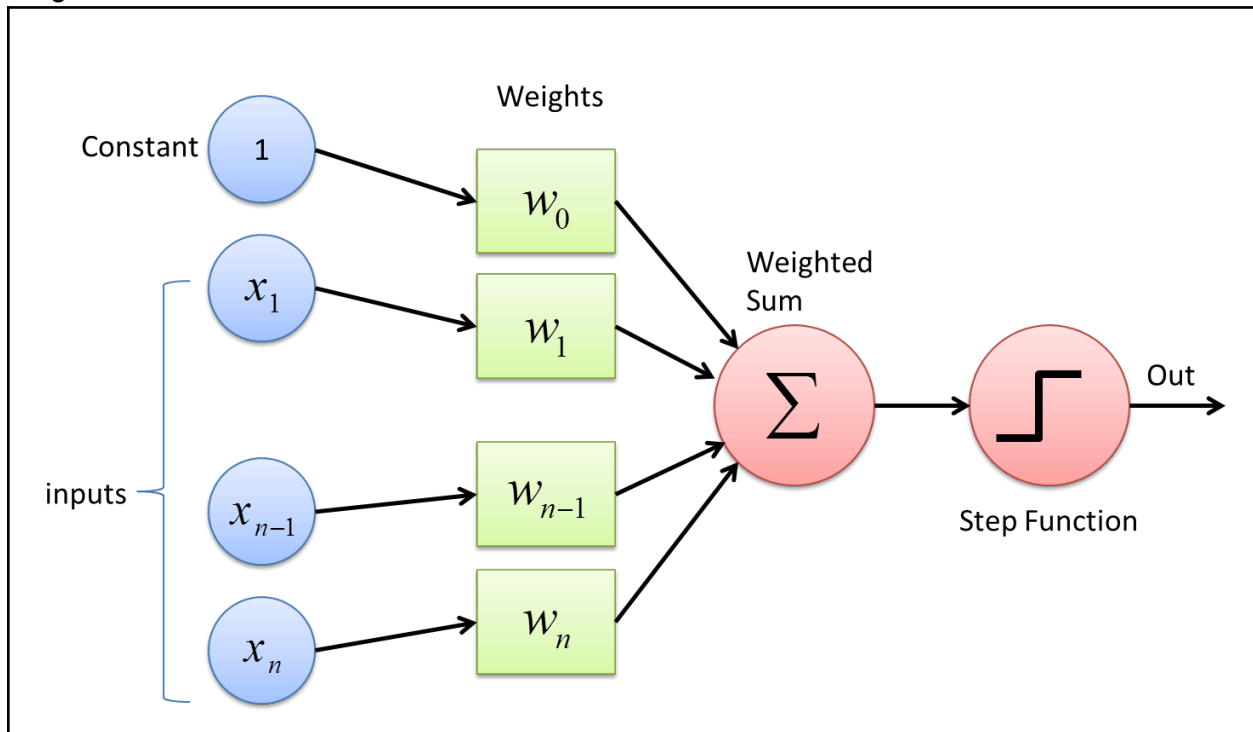
**Reminder:** Please turn this practice packet in on either canvas or in-person at the exam for 5 points of credit on the exam (completion). This packet contains problems covering content from 02/26 to 03/31 (aka from the lecture titled “Perceptrons” through the lecture titled “Gradient Descent and Backpropagation”).

**Note:** Some questions have “Tom’s Thoughts” parts to them in parentheses. These “Tom’s Thoughts” are not necessary to answer for credit on the packet, but are intended to prompt your thinking on a deeper level about some of the concepts covered in this unit. They are things that Tom has seen in classes that follow this one that will be useful for you to remember and know!

**Note:** Questions that have been directly lifted from previous exams (and thus are incredibly indicative of what you may expect) have been marked with their semester and year

### Perceptrons (02/26)

1.) Please draw a diagram of a perceptron. Make sure to include inputs, weights, a bias term, a weighted sum, and an activation function.



2.) Given below is a Perceptron. Using the Step function between -1 and 1 as the activation function, classify the three points given below. Then update the Perceptron weights for each misclassified sample and record the new weights. (Fall 2024)

$$\omega = (1.2, 0.7, 0.2, 0.5), \eta = 0.2$$

Samples	$X_1$	$X_2$	$X_3$	Y
$S_1$	2	3	-1	1
$S_2$	1	1	1	1
$S_3$	-2	-4	3	-1

Classify:

$$S_1: 1.2(1) + 0.7(2) + 0.2(3) + 0.5(-1) = 2.7 \rightarrow \text{step}(2.7) = 1 \quad \hat{y} = 1 \quad y = 1 \quad \checkmark$$

$$S_2: 1.2(1) + 0.7(1) + 0.2(1) + 0.5(1) = 2.6 \rightarrow \text{step}(2.6) = 1 \quad \hat{y} = 1 \quad y = 1 \quad \checkmark$$

$$S_3: 1.2(1) + 0.7(-2) + 0.2(-4) + 0.5(3) = 0.5 \rightarrow \text{step}(0.5) = 1 \quad \hat{y} = 1 \quad y = -1 \quad \times$$

Weight Updates:

$$w' = w + \eta \times y$$

$$w'_0 = 1.2 + 0.2(1)(-1) = 1.0$$

$$w'_1 = 0.7 + 0.2(-2)(-1) = 1.1$$

$$w'_2 = 0.2 + 0.2(-4)(-1) = 1.0$$

$$w'_3 = 0.5 + 0.2(3)(-1) = -0.1$$

Final Weights:

$$\omega = (\omega_0: 1.0, \omega_1: 1.1, \omega_2: 1.0, \omega_3: -0.1)$$

3.) What is the perceptron learning rule? Give a brief “english” label for each component (I.E. “old/previous weight”)

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

learning rate
target value
perceptron output
input value

$w_i = w_i + \alpha y_i x_i$ 
Feature  
2 weight

$$w \leftarrow w + \eta \cdot (y - \hat{y}) \cdot x$$

Lots of different ones, w is the old weight, nu/alpha is the learning rate, yhat is the output prediction, y is the ground truth, and x is the input feature

4.) If you replace the step activation function in a perceptron with a sigmoid function, what kind of model does it become? What could we use instead of the Perceptron learning rule?

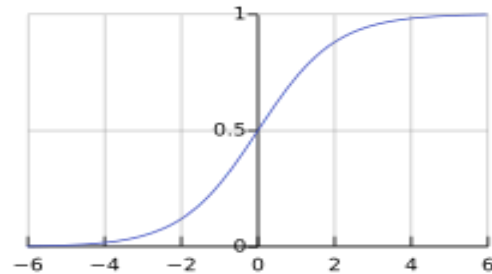
If you replace the step activation function in a perceptron with a sigmoid function, the model transforms from a binary classifier into a type of logistic regression model.

The key difference is that the sigmoid function is differentiable, which allows for the use of gradient-based optimization methods, such as gradient descent, to train the model. The sigmoid also allows you to interpret the classification value as a probability instead of a binary 1 or -1/0

## Logistic Regression (03/03)

1.) What is a sigmoid function? Write down the equation and graph for the sigmoid function. Explain the role of the sigmoid function in logistic regression (and in binary classification in general). (Tom's Thoughts: What properties of the sigmoid function make it useful for classification?)

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



The sigmoid function is an activation function commonly used in binary classification tasks, such as logistic regression. It maps any input (usually a weighted sum of inputs) to a value between 0 and 1 (which can be interpreted as a probability). The sigmoid function is particularly useful for problems where the goal is to model the likelihood of a particular class.

In logistic regression, the goal is to model the probability that a given input belongs to the positive class. The sigmoid function helps transform the linear combination of the input features into a value between 0 and 1, which can be interpreted as the probability of an input belonging to the positive class.

Tom's Thoughts: The sigmoid is useful for classification tasks because:

1. It transforms the output of a linear model into a probability between 0 and 1.
2. It provides a smooth gradient for optimization during training.
3. It makes the model interpretable by having the output represent the probability of class membership

2.) Why is Binary Cross-Entropy Loss commonly used with Logistic Regression? What does it measure? (Tom's Thoughts: why use a log in the loss function?)

In logistic regression, the most commonly used loss function is the **binary cross-entropy loss** (also known as log loss). This loss function is designed to measure the difference between the predicted probabilities (the output of the sigmoid function) and the actual binary labels (0 or 1).

$$-(y \log(p) + (1 - y) \log(1 - p))$$

Equation of Log loss function. **y**-Actual output, **p**-probability predicted by the logistic regression

Tom's Thoughts: You use a logarithm because it makes it so the model is heavily penalized when it makes confident but incorrect predictions, and is penalized very little when making confident correct predictions. The log also ensures a convex function, which is easier to optimize using gradient descent than a non-convex function (won't get caught in a bunch of local minimas).

More info on why log loss:

<https://www.geeksforgeeks.org/ml-cost-function-in-logistic-regression/>

## Support Vector Machines (SVMs) (03/05)

1.) What makes an SVM different from a Perceptron with respect to the decision boundary? (Fall 2024)

Professor Bill Answer: Perceptron's find "a" line while SVMs find "the" line. Additionally due to the kernel trick SVMs can find nonlinear decision boundaries

Tom Answer: The objective of a Support Vector Machine (SVM) is to find the optimal hyperplane that best separates data points of different classes in a feature space. In simple terms, SVMs aim to classify data by drawing a hyperplane that maximizes the margin between the closest points of different classes.

Vocab note: A hyper plane is a decision boundary that separates data points in a higher-dimensional space. 2D = line, 3D = plane, n dimensions > 3D then hyperplane

2.) What are support vectors, and why are they important in SVM? (Fall 2024)

Support vectors are the data points that lie closest to the decision boundary (hyperplane) in a SVM. These points are critical because they determine the position and orientation of the optimal hyperplane.

3.) Explain the difference between a **hard-margin SVM** and a **soft-margin SVM**. When might you want to use each?

A hard-margin SVM assumes that the data is perfectly linearly separable and does not allow any misclassification.

When to use:

- The data is linearly separable

A soft-margin SVM allows some misclassification to make the model more robust to noise and overlapping data. Instead of strictly separating classes, it minimizes the number of misclassifications while still maximizing the margin.

When to use:

- When the data is not linearly separable
- When there is noise in the dataset
- When avoiding overfitting is important.

4.) How does SVM handle non-linearly separable data? What role does the kernel function play in this?

When the data is not linearly separable, meaning a straight line (or hyperplane) cannot perfectly classify the data in its current form, a standard (hard margin) SVM will fail. To address this, SVMs use the Kernel Trick, which maps the data into a higher-dimensional space where it becomes linearly separable.

But why the kernel trick?

- Computing the data transformations (mapping the data to higher dimensions) can be very expensive, especially if the new space is high-dimensional.
- Instead of explicitly computing the transformed data, the kernel trick allows SVMs to directly compute the dot product in the higher-dimensional space using a kernel function. This allows SVM to operate as if the data is in the higher-dimensional space, without ever actually transforming it.
- In this transformed higher dimensional space, the SVM can find a linear decision boundary, which corresponds to a non-linear boundary in the original space.

5.) What are the three kernels we talked about in class used in SVMs? (Tom's Thoughts: What is an advantage and disadvantage of each kernel?)

Linear Kernel:

- Fast and simple, but fails if the decision boundary is non linear

Polynomial Kernel:

- Captures polynomial decision boundaries, aka can model more complex relationships in the data, but is computationally expensive and can be prone to overfitting with very high dimensional polynomials

Radial Basis Function (RBF) Kernel:

- Can model very complex, non-linear decision boundaries
- Requires careful tuning for  $\gamma$  (gamma) and C (margin-misclassification tradeoff)

### **K-Nearest Neighbors (KNN) (03/05)**

1.) Why would we call KNN "lazy"?

There is no fitting process with a KNN, we simply compare distances between points at prediction/inference time

2.) What's a strategy we could use to break ties in voting with a KNN? (Tom's Thoughts: What are "distance weighted knns")

One strategy is just to make sure we have an odd number of comparisons. Another is distance-weighted KNNs which use the distance from the inferred point as a weight on the "vote" the neighbor makes

### **Clustering (03/17)**

1.) What is the goal of unsupervised learning? What is the goal of clustering? How does clustering differ from classification?

The goal of unsupervised learning is to find patterns, structures, or relationships in data (typically by comparing data points to each other).

The goal of clustering is to group similar data points into clusters so that data points within a cluster are more similar to each other than to those in other clusters.

Clustering and classification deal with grouping/categorizing data, there are some key differences:

- Clustering: Unsupervised method, group data based on their features
- Classification: Supervised method, assign labels to / predict labels of data points based on learned features

2.) What are the four steps of k-means clustering we saw in class? What assumptions does it make? What are some pros and cons of kmeans?

K-Means clustering is an unsupervised learning algorithm used to group data into K clusters based on similarity. The algorithm aims to minimize intra-cluster variance by iteratively updating cluster centroids.

Steps:

1. Choose the number of clusters (K)
2. Initialize Cluster Centroids
  - a. Randomly select K datapoints as the initial cluster centroids
3. Assign data points to clusters
  - a. Compute the distance from a datapoint to the centroid of each cluster, add the datapoint to the nearest cluster (do this for all points)
4. Recompute the new centroids of the clusters by taking the average of each sample in the cluster

Assumptions:

- Assumes there is K clusters (aka K is a hyperparameter)
- Assumes spherical/globular clusters

Pros:

- Always gives a solution
- Easy to implement
- Scales well for large datasets

Cons:

- Sensitive to initialization (different random points picked to start can yield different results)
- Requires prespecified K
- Bad with non-globular clusters

3.) What is hierarchical clustering and what is an advantage has over k means?

Hierarchical clustering is a method of clustering that seeks to build a tree of clusters. It does this by either merging clusters or splitting them based on a distance metric.

One of the key advantages of hierarchical clustering is that it does not require you to specify the number of clusters (K) in advance, unlike K-Means. The number of clusters can be determined after the tree (dendrogram) is built by cutting the tree at a desired level.

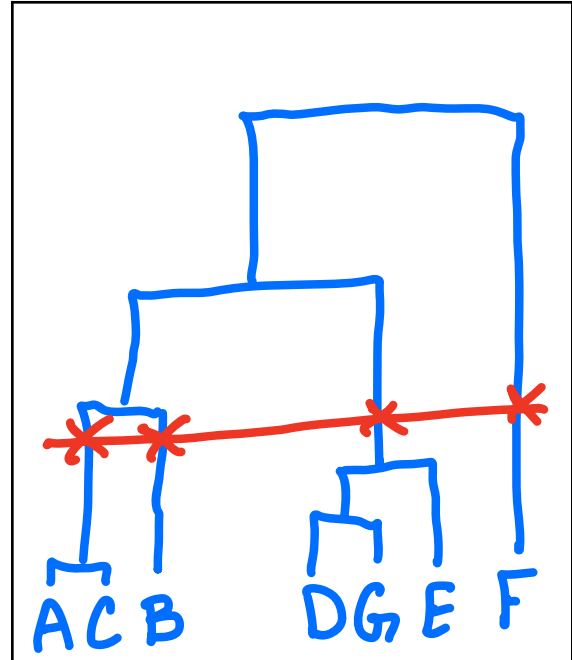
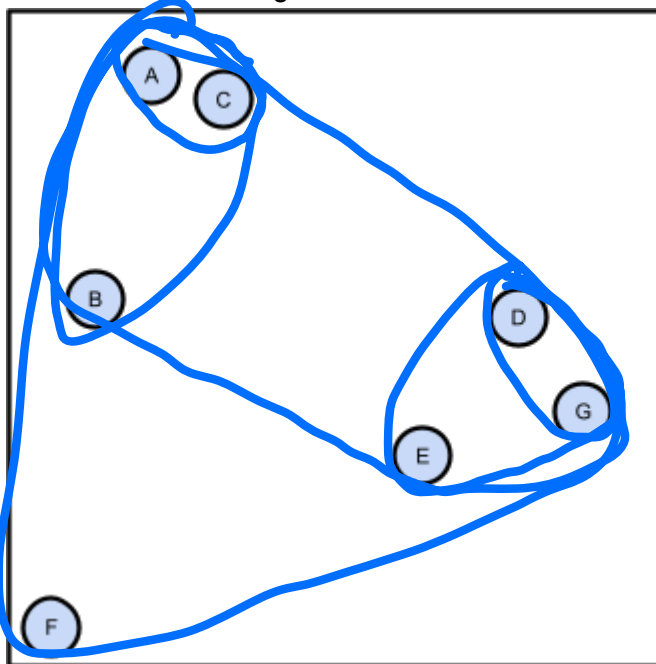
Dendrograms provide a clear visual representation of how clusters are formed and merged step-by-step. The tree structure allows us to easily see the hierarchical relationship between clusters.

Pros of hierarchical clustering:

- Number of clusters not a hyperparameter
- Deterministic



4.) Given the data points, draw the dendrogram that would be created using agglomerative hierarchical clustering and then draw a line on the dendrogram to create 4 clusters. (Fall 2024)



5.) Explain the difference between agglomerative and divisive hierarchical clustering.

Hierarchical clustering builds a hierarchy of clusters using either a bottom-up (agglomerative) or top-down (divisive) approach.

Agglomerative clustering starts with each data point as its own cluster, then iteratively merge the closest clusters based on a distance metric (e.g., Euclidean distance). It continues until only one cluster remains or a stopping condition is met (aka a preset number of clusters).

Divisive clustering starts with all clustering as a single cluster and recursively splits the largest cluster into 2 until each data point is its own cluster.

6.) Provide two real-world applications where clustering could be used and explain why.

Application 1: Customer segmentation

- Businesses use clustering to segment customers based on their purchasing behavior, demographics, or activity

Application 2: Credit Card Fraud Detection

- Uses clustering to identify unusual transactions as potential fraud

## Dimensionality Reduction (03/19)

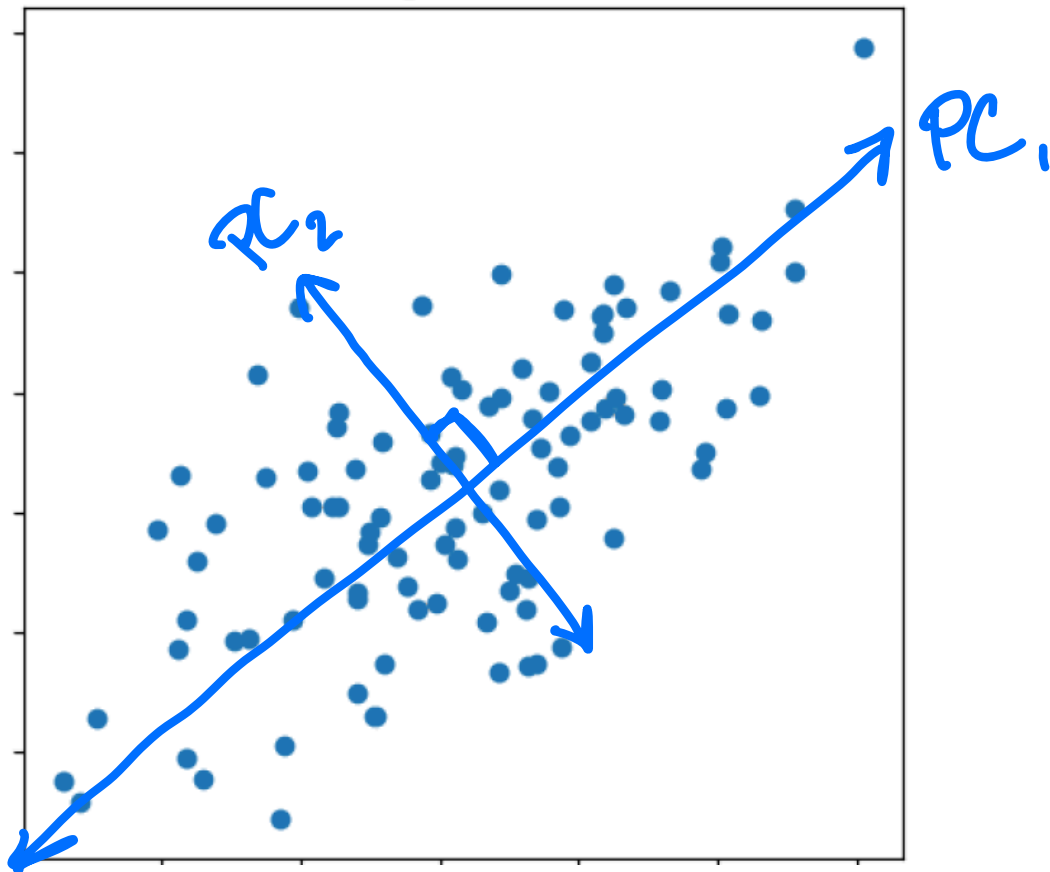
1.) What is dimensionality reduction, and when may we use it?

Dimensionality reduction is the process of reducing the number of features describing data in a dataset while preserving as much important information as possible. This is done by transforming the data from a higher-dimensional to a lower-dimensional space.

Dimensionality reduction can be done through feature selection (choosing a subset of already existing relevant features) or feature extraction (creating or learning new features from existing ones).

Dimensionality reduction is important because it can mitigate the curse of dimensionality, reduce computational cost, improve model performance, and enhance data visualizations.

2.) Draw (approximately) the two principal components on the plot and label them (Fall 2024)



3.) Why are the principal components always orthogonal to each other

If they weren't orthogonal to each other we'd have diagonals in the covariance matrix that were non-zero thus indicating informational overlap in the PCs and defeating the entire purpose

4.) Explain the curse of dimensionality and how it affects machine learning models. (Fall 2024)

The curse of dimensionality refers to the challenges that arise as the number of dimensions (features) in a dataset increases. When the data has too many dimensions, models struggle to learn effectively due to increased data sparsity, higher computational costs, and overfitting risks.

5.) What are the roles of eigenvalues and eigenvectors in PCA (Fall 2024)

The eigenvalues represent how much variance is described by / contained in each principal component and the corresponding eigenvector represents the direction in which the principal component points. In other words, the eigenvector describe the transformed coordinate system

6.) If I have a dataset with 5 input features and then transform the 5 features into 5 principal components, did I lose any information in my dataset? Why or why not? If I only select and use the top 2 principal components did I lose any information? Why or why not?

If you have a dataset with 5 inputs features, and transform the 5 features into 5 principle components, and then use all 5 principle components, you have not lost any information in your dataset. PCA only rotates the data to a new coordinate system but does not remove any dimensions when you keep all 5 principal components. In other words, the transformation is lossless because all variance from the original dataset is contained within the 5 principle components.

If you only select 2 of the principle components found, then you have likely lost information. The top 2 principal components capture only a portion (likely a majority) of the total variance in the dataset. Lower-ranked principal components (PC3, PC4, PC5) that are discarded contain less, but still some information (variance).

7.) What does it mean to standardize your data? (Tom's Thoughts: Why do you need to standardize your data before performing PCA?)

Standardizing data means scaling the features so that they have Zero mean (Centering the data around 0) and Unit variance (Each feature has a standard deviation of 1).

Tom's Thoughts: PCA is based on variance, which is sensitive to scale differences. If features are not standardized features with larger numerical ranges will dominate the principal components, making the results biased. In other words, without standardization the direction of maximum variance (and the majority of variance in general) might be skewed towards features with larger scales rather than towards meaningful structural variance in the data.

8.) What is input space? What is feature space? (Tom's Thoughts: Why is having/learning a good feature space important?)

Input space refers to the raw, original data representation before any transformations are applied. It consists of the original features collected from a dataset.

Feature space refers to a transformed representation of the data where meaningful patterns can be more easily identified. It consists of extracted or engineered features that enhance learning.

Tom's Thoughts: A good feature space is important because it can make different classes easier to separate, reduce dimensionality, make learning more efficient, and make models generalize better.

## Practicum - Evaluation Metrics (03/24)

1.) What are Type 1 and Type 2 errors?

Type I Errors: false positive

Type II Errors: false negative

2.) Given the following experiments, which of the metrics do you think would be most useful for measuring task performance. Select only one. (Multiple Choice) (Fall 2024)

a.) An imbalanced multiclass classification task

- Precision
- Recall
- F1 Score
- MSE

b.) Deciding whether to give someone a loan

- Precision
- MSE
- Accuracy
- Silhouette Score

c.) A regression task

- Precision

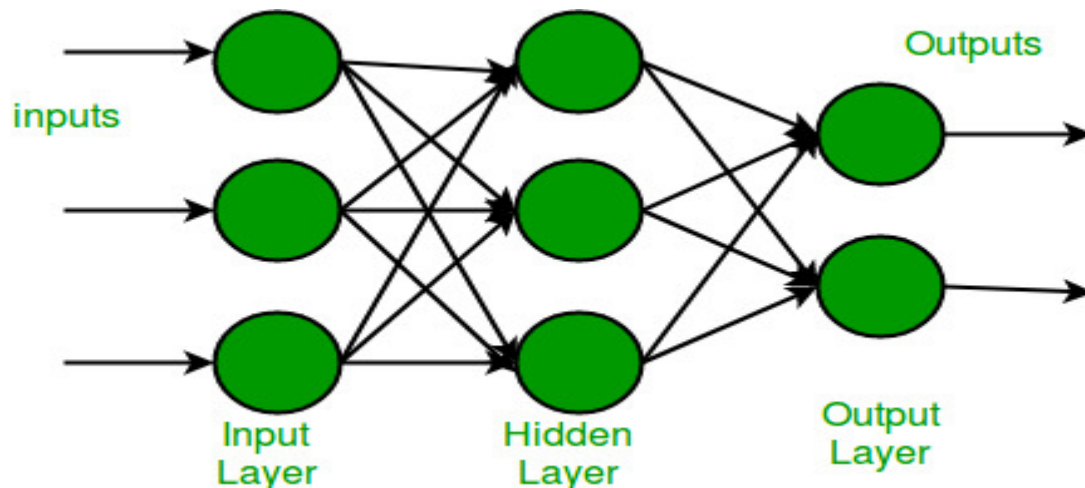
- F1 Score
- MSE
- Laplacian Difference

3.) Match the following metrics to their equation and definition: Accuracy, Precision, Recall, F1 Score

Accuracy	$(TP + TN) / (TP + TN + FP + FN)$	Accuracy	"How close a given set of measurements are to their actual values"
Precision	$TP / (TP + FP)$	Precision	"How close measurements are to each other"
Recall	$TP / (TP + FN)$	Recall	"Proportion of actual positives that were classified correctly"
F1	$2 * (Precision * Recall) / (Precision + Recall)$	F1	"The harmonic mean of precision and recall"

## Feed Forward Neural Networks (FFNs) (MLPs) (03/26)

1.) Explain the structure of a multilayer perceptron (MLP). What are the roles of the input layer, hidden layers, and output layer?



### Input Layer:

- Used to feed data into the network.
- Each neuron corresponds to one feature. (aka the number of neurons = the number of input features).
- No activation function is applied here—data is simply passed to the next layer.

### Hidden Layer(s):

- Extracts features and learns complex patterns.
- Consists of multiple neurons connected to previous and next layers. Each neuron applies a weighted sum of inputs, followed by an activation function.
- multiple hidden layers = deep learning

### Output Layer:

- Produces the final prediction
- Number of neurons = number of output classes.
- Uses a task specific activation function (sigmoid for binary classification, softmax for multi-class classification, linear or no activation function for regression))

2.) What are activation functions, and why are they necessary in an MLP? (Fall 2024) (Tom's Thoughts: Compare and contrast common activation functions, specifically: **ReLU**, **sigmoid**, and **tanh**.)

An activation function is a mathematical function applied to the end of a neuron in a neural network. They introduce non-linearity into the model and allow neural networks to learn complex, non-linear relationships in data.

**ReLU**: (equation, graph, graph of derivative)

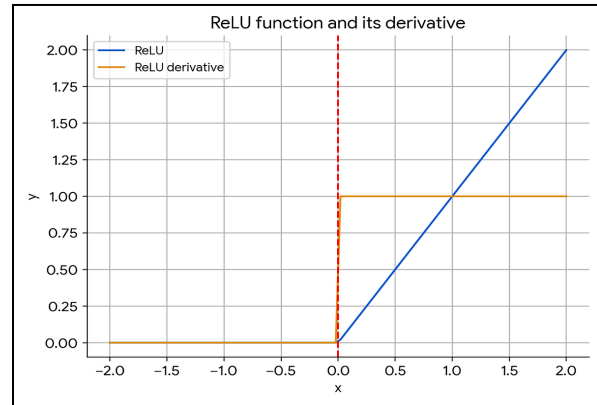
$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

Pros:

- fast to compute, reduces vanishing gradient problem risk

Cons:

- Dying ReLU problem, exploding activations & gradients



**Sigmoid**: (equation, graph, graph of derivative)

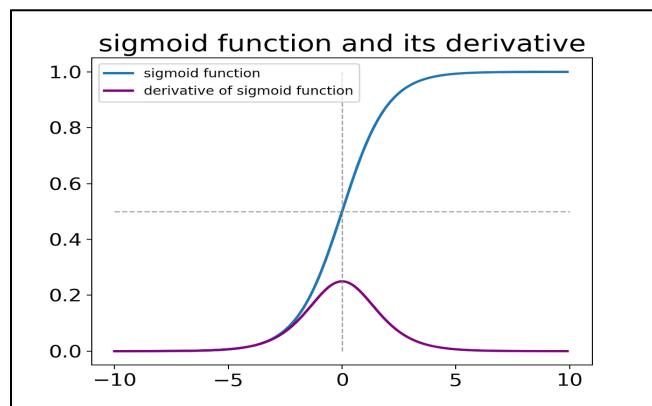
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Pros:

- Smooth and differentiable, good for shallow networks

Cons:

- Vanishing gradient problem, computationally expensive compared to ReLU



**Tanh**: (equation, graph, graph of derivative)

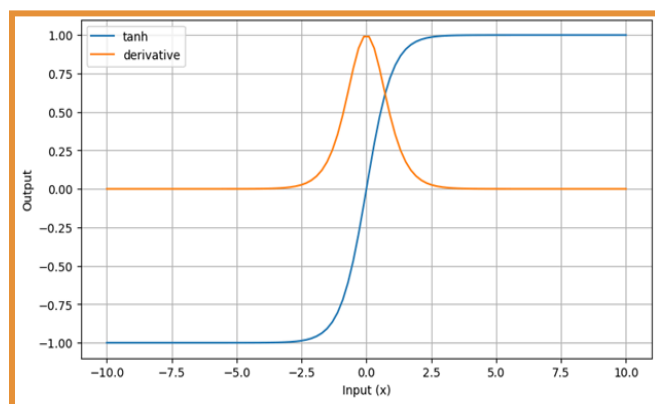
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Pros:

- Zero-centered output (between -1 and 1),

Cons:

- Vanishing gradient problem, computationally expensive





3.) Describe the forward propagation process in an MLP. How is the output of a neural network calculated from the input?

Forward propagation is the process by which a feed forward neural network computes its output given an input. It involves passing data through the network layer by layer. You can denote this process as nested functions.

For each layer, the output from the previous layer is multiplied by the weights of the current layer (added with the bias). Then the result is used as the input to the activation function of the current layer (and the output of the activation function is used as the input to the next layer). This process of information flow is repeated through the layers until reaching the output.

4.) What is the role of a loss function in training an MLP? What are common loss functions used in classification and regression tasks? (Fall 2024)

A loss function in an MLP quantifies how far the model's predictions are from the actual target values. During training, the loss function guides the optimization process by providing a measure of error, which the model seeks to minimize through backpropagation and gradient descent.

For classification, cross entropy loss is common. For regression tasks, mean squared error (MSE) and mean absolute error (MAE) are common.

### **Gradient Descent and Backprop (03/31)**

1.) What is the difference between backpropagation and gradient descent? How are they related? (Fall 2024)

Backpropagation is an algorithm used to compute the gradient of the loss function with respect to the model's weights. It uses the chain rule to propagate errors backward through the network layer by layer. Backpropagation itself does not update the weights, it only calculates the gradients needed for weight updates.

Gradient descent is an optimization algorithm used to minimize the loss function by updating the model's weights in the direction of the negative gradient. It uses the gradients computed by backpropagation to adjust weights in order to reduce the loss iteratively.

**TLDR:** Backpropagation provides the gradients, and gradient descent uses these gradients to update the weights in order to optimize the neural network.

## 2.) Describe the role of the chain rule in backpropagation. Why is it necessary?

Since neural networks consist of multiple layers, the relationship between inputs and outputs is composed of nested functions. The chain rule helps us decompose the derivative of this composition into smaller, manageable parts.

Once the gradient of the loss with respect to the output is known, gradients for earlier layers can be computed sequentially using the chain rule. This means you can calculate the gradients for all the weights in the network in one wave or “propagation” backwards through the network.

## 3.) What are vanishing and exploding gradients, and how do they affect backpropagation / neural network training? (Fall 2024)

When training deep neural networks, the gradients of the loss function with respect to the weights are computed layer by layer using the chain rule. However, in very deep networks, these gradients can become extremely small (vanishing gradients) or excessively large (exploding gradients), leading to training difficulties.

### Vanishing gradients

- As backpropagation moves from the output layer to the input layer, the gradients of earlier layers can become very small.
- This is often caused by activation functions like the sigmoid or tanh, which squash values into a small range, leading to small derivatives.
- This means that the updates to the weights in these layers become negligible, slowing down learning or even stopping it entirely (think about what happens if you multiply 4 or 5 numbers together that are all less than 1...)

### Exploding gradients

- As backpropagation moves from the output layer to the input layer, the gradients grow super (even exponentially) large when there are large values in the weights or activations
- Happens when weights are too large, and the repeated multiplication of large values during backpropagation amplifies the gradients
- This leads to unstable weight updates and the loss function oscillating sporadically

4.) What is the difference between batch gradient descent, stochastic gradient descent (SGD), and mini-batch gradient descent?

Batch gradient descent:

- Uses the entire training dataset to compute the gradient of the loss function and update the model weights. For each update, the gradient is computed based on the sum (or average) of the gradients from every training example.
- The weights are updated after processing the entire dataset (aka one update per full pass through the training set)

Stochastic gradient descent:

- Uses only one training example at a time to compute the gradient and update the weights
- The weights are updated after every single training example (aka updates happen very frequently)

Mini-batch gradient descent:

- Uses a small subset (mini-batch) of the training data (typically between 32 and 512 samples) to compute the gradient and update the weights
- The weights are updated after processing the mini-batch (rather than after each individual example or after the entire dataset)

5.) How does the learning rate affect the performance of gradient descent? What are potential issues with too high or too low learning rates? (Fall 2024)

The learning rate controls the size of the steps taken in the direction of the negative gradient during optimization. It directly affects how quickly the algorithm converges to the minimum of the loss function and the stability of the training process.

A higher learning rate makes larger updates to the model's weights, which can lead to faster progress toward the loss functions minimum in the early stages of training. However, if the learning rate is too high, the weight update might overshoot this minimum. Instead of moving to and settling at the optimal point, the weight updates might oscillate around the minimum.

A smaller learning rate means smaller steps, which can make convergence slow (aka while a smaller learning rate can result in more precise updates, it can also prolong the training process significantly). If the learning rate is too small, the model might get stuck in local minima or saddle points.

6.) Explain the role of momentum in gradient descent. How does it help optimize convergence?

Momentum is a technique used to improve the convergence of gradient descent by smoothing out the weight updates and helping the optimization process move faster and more efficiently (especially when dealing with noisy gradients).

In standard gradient descent, each update is based solely on the current gradient of the loss function. Momentum adds a "memory" to the optimization process by incorporating a fraction of the previous weight update in the current update. This helps the model's weights to maintain (or at least more likely maintain) movement in the right direction, even when the gradient fluctuates.

### Neural Network Application Questions

Given the network diagram, answer the following questions. (Similar to Fall 2024)

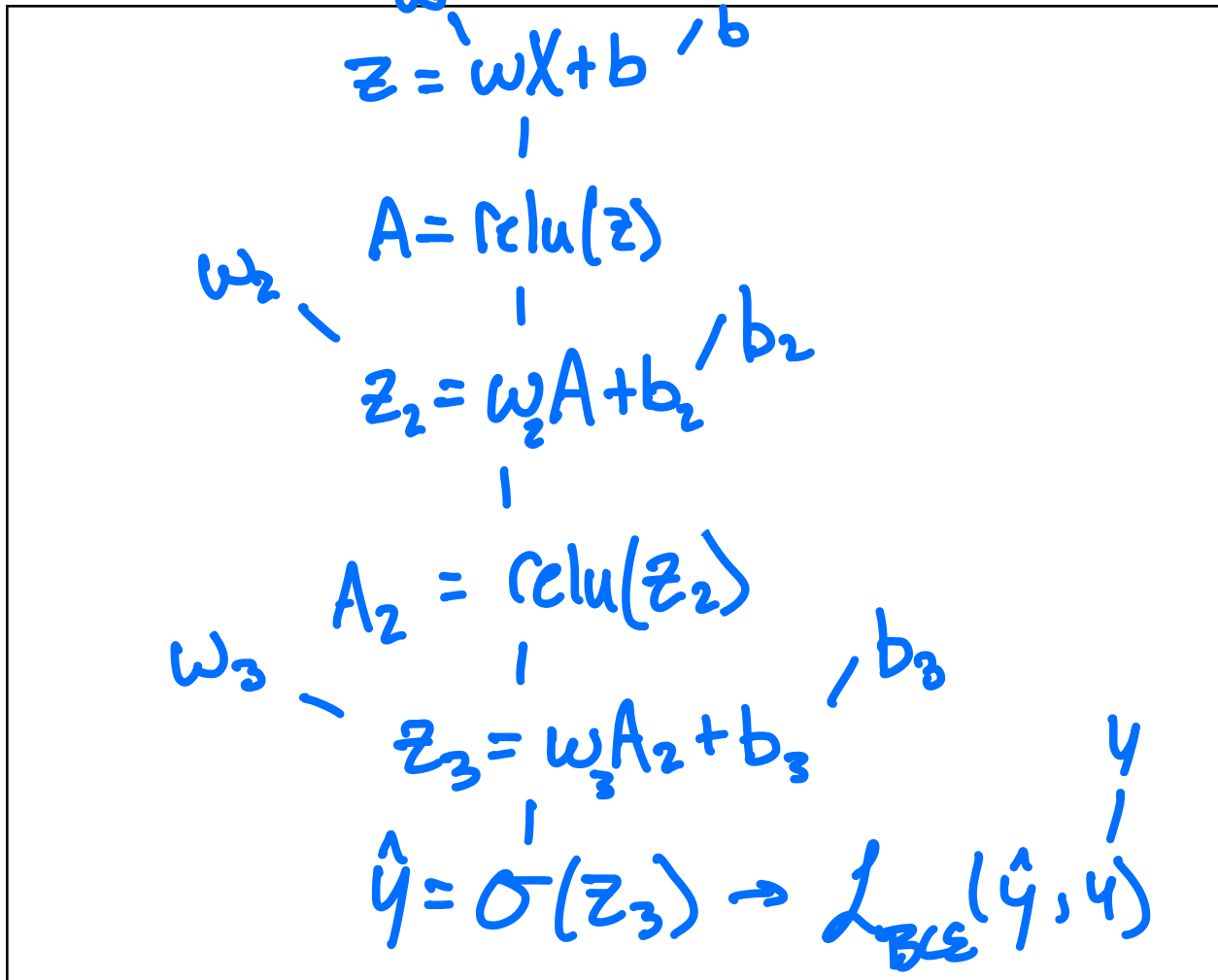
```
import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleNetwork(nn.Module):
    def __init__(self):
        super(SimpleNetwork, self).__init__()
        self.fc1 = nn.Linear(2, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x

model = SimpleNetwork()
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

1.) Draw the computation graph for the network (either Left to Right or Top to Bottom is fine)  
 (Note: Please refer to the final page for a lookup table of functions and their derivatives)



2.) Write the output of the neural network ( $y_{\text{hat}}$ ) as a single statement of nested functions.  
 (Note: Please refer to the final page for a lookup table of functions and their derivatives)

$$\hat{y} = \sigma(w_3(\text{relu}(w_2(\text{relu}(w_1 x + b_1) + b_2)) + b_3))$$

3.) Write the chain of derivatives performed to calculate the gradient of the bias in 1st the hidden layer.


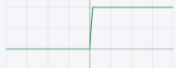


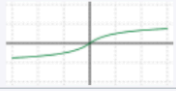
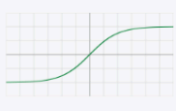




$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial A_2} \frac{\partial A_2}{\partial z_2} \frac{\partial z_2}{\partial A} \frac{\partial A}{\partial z} \frac{\partial z}{\partial b}$$

4.) Give the equation you'd use to calculate the gradients. (**Note:** Please refer to the final page for a lookup table of functions and their derivatives)

$$\frac{\partial L}{\partial b} = (\hat{y} - y) (w_3) \begin{pmatrix} z_2 > 0 : 1 \\ z_2 \leq 0 : 0 \end{pmatrix} (w_2) \begin{pmatrix} z > 0 : 1 \\ z \leq 0 : 0 \end{pmatrix} (1)$$

5.) What type of task is this network probably attempting to solve? How do you know?

Binary classification of 2D point  
 Loss is BCE + Sigmoid and input  
 dim is 2

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$
ElliotSig Softsign		$f(x) = \frac{x}{1 +  x }$	$f'(x) = \frac{1}{(1 +  x )^2}$
Square Nonlinearity (SQNL)		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$	$f'(x) = 1 \mp \frac{x}{2}$
Rectified linear unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Bipolar rectified linear unit (BRELU)		$f(x_i) = \begin{cases} ReLU(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$f'(x_i) = \begin{cases} ReLU'(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU'(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric rectified linear unit (PReLU)		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

Name	Equation	Derivative
Mean Squared Error (MSE)	$\mathcal{L} = \frac{1}{n} \sum (y - \hat{y})^2$	$\frac{\partial \mathcal{L}}{\partial y} = -2(y - \hat{y})$
Mean Absolute Error (MAE)	$\mathcal{L} = \frac{1}{n} \sum  y - \hat{y} $	$\frac{\partial \mathcal{L}}{\partial y} = -\text{sign}(y - \hat{y})$
Huber Loss	$L = 0.5(y - \hat{y})^2$ if $ y - \hat{y}  \leq \delta$ ; else $\delta( y - \hat{y}  - 0.5\delta)$	Piecewise gradient
Binary Cross Entropy (BCE)	$\mathcal{L} = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$	$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$
BCE with Sigmoid	$\mathcal{L} = -[y \log(\sigma(z)) + (1 - y) \log(1 - \sigma(z))]$	$\frac{\partial \mathcal{L}}{\partial z} = \sigma(z) - y$
Categorical Cross Entropy	$\mathcal{L} = -\sum_i y_i \log(\hat{y}_i)$	$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{\hat{y}_i - y_i}{\hat{y}_i}$
KL Divergence	$\mathcal{L} = \sum y \log\left(\frac{y}{\hat{y}}\right)$	$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}}$
Poisson Loss	$\mathcal{L} = \hat{y} - y \log(\hat{y})$	$\frac{\partial \mathcal{L}}{\partial \hat{y}} = 1 - \frac{y}{\hat{y}}$
Hinge Loss	$\mathcal{L} = \max(0, 1 - y\hat{y})$	$\partial \mathcal{L} / \partial \hat{y} = -y$ if $y\hat{y} < 1$ ; else 0
Squared Hinge Loss	$\mathcal{L} = \max(0, 1 - y\hat{y})^2$	$\partial \mathcal{L} / \partial \hat{y} = -2y(1 - y\hat{y})$ if $y\hat{y} < 1$ ; else 0