Advanced Learning Algorithms

Week 1: Neural Networks

Learning Objectives:

- Get familiar with the diagram and components of a neural network
- Understand the concept of a "layer" in a neural network
- Understand how neural networks learn new features.
- Understand how activations are calculated at each layer.
- Learn how a neural network can perform classification on an image.
- Use a framework, TensorFlow, to build a neural network for classification of an image.
- Learn how data goes into and out of a neural network layer in TensorFlow
- Build a neural network in regular Python code (from scratch) to make predictions.
- (Optional): Learn how neural networks use parallel processing (vectorization) to make computations faster.

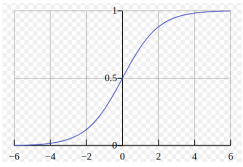
Key Concepts:

Neuron: Basic computational unit in a neural network that takes input, processes it through an activation function, and passes the output to the next layer.

Activation Function: Applies a non-linear transformation to produce the neuron's output.

Common Activation Functions:

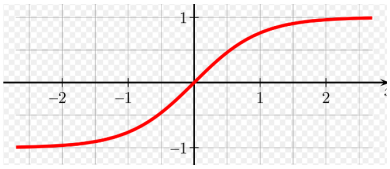
1. **Sigmoid Function**



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

Output range: (0, 1)

2. **Hyperbolic Tangent (Tanh)**



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

Output range: (-1, 1)

3. **Rectified Linear Unit (ReLU)**

ReLU(x)=max(0,x)
Output range: [0, ∞)

4. **Leaky ReLU**

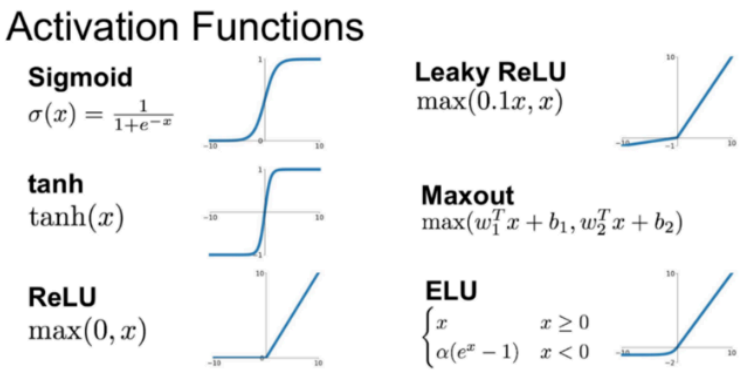
Formula: Leaky ReLU(x)=max(αx,x) where α is a small constant.
Output range: (-∞, ∞)
Parametric ReLU (PReLU) is where α is a learnable parameter.

5. **Exponential Linear Unit (ELU)**

Formula: ELU(x)=x if x>0, else α(e^x - 1)
Output range: (-α, ∞)

6. **Swish**

Formula: Swish(x)=x·σ(x)
Output range: (-∞, ∞)



Different Activation Functions and their Graphs

A **layer** in a neural network consists of a collection of neurons (Input, Hidden, Output).

Forward Propagation: Process of computing outputs from inputs through network layers.

$$a^{[l]} = g(z^{[l]}), z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$$

TensorFlow: Framework for building and training neural networks.

Tensorflow is a machine learning package developed by Google. In 2019, Google integrated Keras into Tensorflow and released Tensorflow 2.0. Keras is a framework developed independently by François Chollet that creates a simple, layer-centric interface to Tensorflow.

Inference: Making predictions with a trained model.

Epochs: Complete passes through the training dataset.

Batches: Subsets of training data for weight updates.

Artificial general intelligence (AGI) is a type of artificial intelligence (AI) that matches or surpasses human capabilities across a wide range of cognitive tasks.

Week 2: Neural network training

Learning Objectives:

- Train a neural network on data using TensorFlow
- Understand the difference between various activation functions (sigmoid, ReLU, and linear)
- Understand which activation functions to use for which type of layer
- Understand why we need non-linear activation functions
- Understand multiclass classification
- Calculate the softmax activation for implementing multiclass classification
- Use the categorical cross entropy loss function for multiclass classification
- Use the recommended method for implementing multiclass classification in code
- (Optional): Explain the difference between multi-label and multiclass classification

In **multiclass classification**, each instance belongs to one and only one class out of a set of multiple classes.
Example: Classifying images of animals into categories such as cats, dogs, and horses. Each image belongs to exactly one category.

In **multi-label classification**, each instance can belong to multiple classes simultaneously.
Example: Classifying news articles into categories such as sports, politics, and technology. A single article can belong to more than one category.

Adam stands for Adaptive Moment Estimation. Key features: Adaptive Learning Rates, Momentum and Bias Correction.

Binary cross-entropy is a crucial loss function for binary classification problems,

The binary cross-entropy loss for a single instance can be defined as:

$$\text{BCE}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

Softmax Function

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}}$$

SparseCategorialCrossentropy: expects the target to be an integer corresponding to the index. For example, if there are 10 potential target values, y would be between 0 and 9.

CategoricalCrossEntropy: Expects the target value of an example to be one-hot encoded where the value at the target index is 1 while the other N-1 entries are zero. An example with 10 potential target values, where the target is 2 would be [0,0,1,0,0,0,0,0,0,0].

Code:

```
tf.random.set_seed(1234) # for consistent results
model = Sequential(
    [
        tf.keras.Input(shape=(400,)),
        Dense(25, activation="relu"),
        Dense(15, activation="relu"),
        Dense(10, activation="linear")
    ], name = "my_model"
)
```

Week 3: Advice for applying machine learning

Learning Objectives:

- Evaluate and then modify your learning algorithm or data to improve your model's performance
- Evaluate your learning algorithm using cross validation and test datasets.
- Diagnose bias and variance in your learning algorithm
- Use regularization to adjust bias and variance in your learning algorithm
- Identify a baseline level of performance for your learning algorithm
- Understand how bias and variance apply to neural networks
- Learn about the iterative loop of Machine Learning Development that's used to update and improve a machine learning model
- Learn to use error analysis to identify the types of errors that a learning algorithm is making
- Learn how to add more training data to improve your model, including data augmentation and data synthesis
- Use transfer learning to improve your model's performance.
- Learn to include fairness and ethics in your machine learning model development
- Measure precision and recall to work with skewed (imbalanced) datasets

A **diagnostic** is a test that you run to gain insight into what is/isn't working with a learning algorithm, to gain guidance into improving its performance.

Bias: Error due to overly simplistic assumptions in the learning algorithm. High bias can cause the model to miss relevant relations (underfitting).

Variance: Error due to too much complexity in the learning algorithm. High variance can cause the model to model the random noise in the training data (overfitting).

To fix a high bias problem, you can:

- try adding polynomial features
- try getting additional features
- try decreasing the regularization parameter

To fix a high variance problem, you can:

- try increasing the regularization parameter
- try smaller sets of features
- get more training examples

Before you can diagnose a model for high bias or high variance, it is usually helpful to first have an idea of what level of error you can reasonably get to. As mentioned in class, you can use any of the following to set a **baseline level of performance**:

- human level performance

- competing algorithm's performance
- guess based on experience

Precision: The ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall (Sensitivity): The ratio of correctly predicted positive observations to the all observations in actual class.

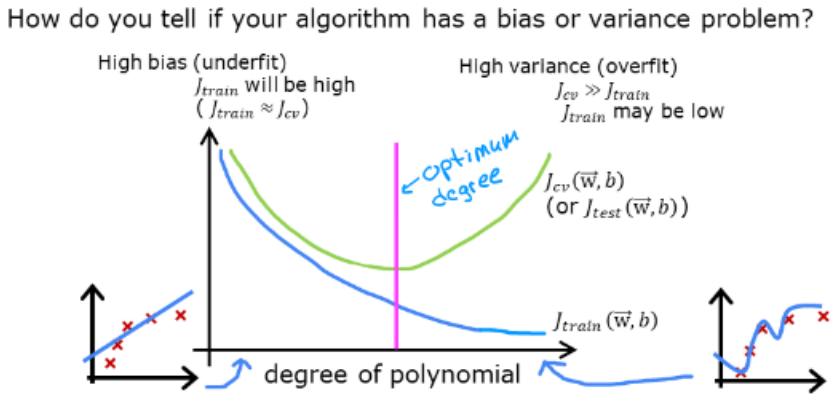
$$\text{Recall} = \frac{TP}{TP+FN}$$

F1-Score: The harmonic mean of precision and recall.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

ROC-AUC: The Area Under the Receiver Operating Characteristic Curve, which plots True Positive Rate against False Positive Rate at various threshold settings.

Precision-Recall AUC: Area Under the Precision-Recall Curve, useful when dealing with imbalanced datasets.



The **L2 regularizer**, also known as **L2 regularization** or **Ridge regularization**, is a technique used to prevent overfitting in machine learning models by penalizing large weights

$$\text{L2 Regularization Term} = \frac{\lambda}{2} \sum_{i=1}^{n} w_i^2$$

$$\text{Loss} = \text{Original Loss} + \frac{\lambda}{2} \sum_{i=1}^{n} w_i^2$$

Week 4: Decision trees

Learning Objectives:

- See what a decision tree looks like and how it can be used to make predictions
- Learn how a decision tree learns from training data
- Learn the "impurity" metric "entropy" and how it's used when building a decision tree
- Learn how to use multiple trees, "tree ensembles" such as random forests and boosted trees
- Learn when to use decision trees or neural networks

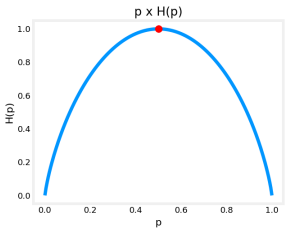
A **decision tree** is a supervised learning algorithm used for classification and regression tasks. It splits the data into subsets based on the value of input features, aiming to create a tree-like model of decisions.

Purity is a measure of how homogenous the samples in a node are with respect to the target variable. A node is considered "pure" if all samples in that node belong to a single class.

Measuring Purity:

1. **Entropy**: Measures the amount of uncertainty or impurity in the node. It is based on the concept of entropy from information theory.

$$\text{Entropy}(p_1, p_2, \ldots, p_k) = -\sum_{i=1}^{k} p_i \log_2(p_i)$$



$$H(p_1) = -p_1 \log_2(p_1) - (1 - p_1)\log_2(1 - p_1)$$

2. **Gini Impurity**: It is the probability of misclassifying a randomly chosen element in a set.

$$\text{Gini}(p_1, p_2, \ldots, p_k) = 1 - \sum_{i=1}^{k} p_i^2$$

3. **Misclassification Rate**: The proportion of samples in the node that do not belong to the majority class.

$$\text{Misclassification Rate} = 1 - \text{Max}(p_1, p_2, \ldots, p_k)$$

Information Gain is defined as the reduction in entropy (uncertainty) after a dataset is split on a particular feature

$$\text{Information Gain} = H(p_1^{\text{node}}) - \left(w^{\text{left}} H\left(p_1^{\text{left}}\right) + w^{\text{right}} H\left(p_1^{\text{right}}\right)\right),$$

One-hot encoding is a technique for converting categorical data into a binary matrix where each category is represented as a unique binary vector.

Regression Trees are decision trees used for predicting continuous values rather than categories. They work by recursively splitting the data based on feature values to minimize the variance (or other criteria) within each subset.

Tree Ensembles combine multiple decision trees to improve predictive performance and robustness. The two main types are Random Forests and Boosted Trees.

Random Forests are an ensemble learning method that creates a collection of decision trees and combines their predictions to improve accuracy and control overfitting. Reduces overfitting compared to a single decision tree.

How It Works:

- Bootstrap Sampling: Generate multiple bootstrap samples (random samples with replacement) from the training data.
- Tree Building: Train a decision tree on each bootstrap sample. Each tree is built using a random subset of features at each split (feature bagging).
- Aggregation: For regression, predictions are averaged; for classification, the majority vote is taken.

Boosted Trees (XGBoost): Sequentially builds trees to correct errors of previous trees using gradient boosting and regularization to improve performance.

How It Works

- Sequential Learning: Trees are added one at a time, with each new tree focusing on the residual errors of the combined ensemble of previous trees.
- Gradient Boosting: Uses gradient descent to minimize a loss function by adjusting the model based on the errors of the previous trees.
- Regularization: Includes regularization terms to prevent overfitting and improve model generalization.

GridSearchCV is a Scikit-Learn tool for hyperparameter tuning. It exhaustively searches over a specified parameter grid to find the best combination of hyperparameters for a model using cross-validation.