NEURAL NETWORKS

FA590 STATISTICAL LEARNING IN FINANCE

Dr. Zonghao Yang

Stevens Institute of Technology

Previous Lecture

- Understand the fundamentals of ensemble methods, including bagging and boosting
- Learn the mechanics of bagging and random forests
- Gain insight into boosting techniques, including algorithms such as AdaBoost and Gradient Boosting, and comprehend how each model corrects the errors of its predecessors
- Apply ensemble methods to regression and classification tasks

Learning Objectives

- Learn the principles of gradient descent
- Describe the architecture of a neural network, including the roles of input, hidden, and output layers, and understand the properties of different activation functions such as ReLU, sigmoid, and tanh
- Articulate the hyperparameters of a neural network and build intuition on how to tune the hyperparameters based on learning curves

Part I

Gradient Descent

Parameter Estimation

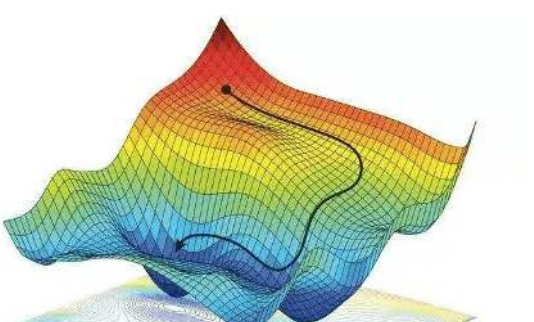
- Objective function $L : \mathbb{R}^d \mapsto \mathbb{R}$ is differentiable
- The objective of solving the optimization problem is

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^d} L(x; \mathbf{w})$$

  where $\mathbf{w}$ is the set of parameters of L
- Common objective function for regression is sum of squared errors
- Common objective function for classification is cross-entropy
- Gradient descent is a general numerical method for solving an optimization problem

Intuition behind Gradient Descent

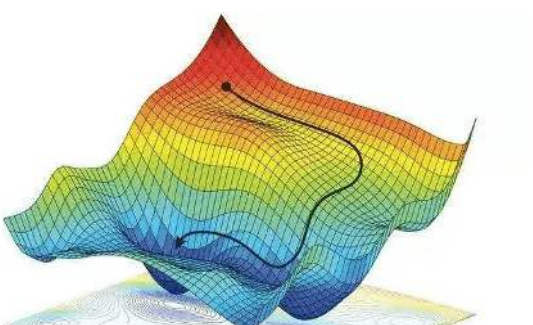
A person is stuck in the mountain, trying to get down as fast as possible. There is heavy fog, so the path down to the valley is not visible. What is the best way to proceed?



Intuition behind Gradient Descent

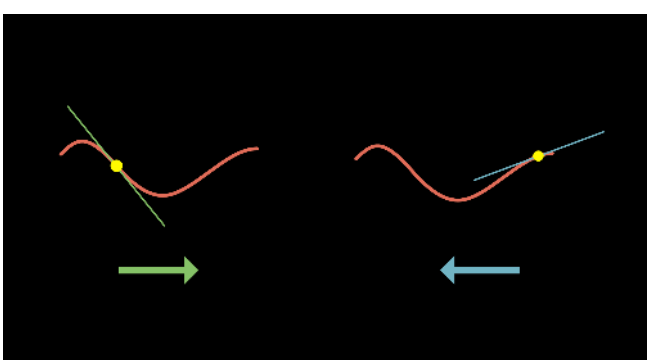
A person is stuck in the mountain, trying to get down as fast as possible. There is heavy fog, so the path down to the valley is not visible. What is the best way to proceed?

Local information is required to find the bottom of the mountain. The hiker can look at the steepness of the hill at their current position, then proceed in the direction with the steepest descent (i.e., downhill).



The Gradient

- Let $L : \mathbb{R}^d \to \mathbb{R}$ be differentiable at $\mathbf{w}_0 \in \mathbb{R}^d$
- The **gradient** of L at the point $\mathbf{w}_0$, denoted $\nabla_{\mathbf{w}} L(x; \mathbf{w}_0)$, is the direction to move in for the fastest increase in $L(\mathbf{w})$, when starting from $\mathbf{w}_0$
- Thus, the opposite direction of the gradient points to the fastest decrease in $L(\mathbf{w})$; see figure below



Gradient Descent

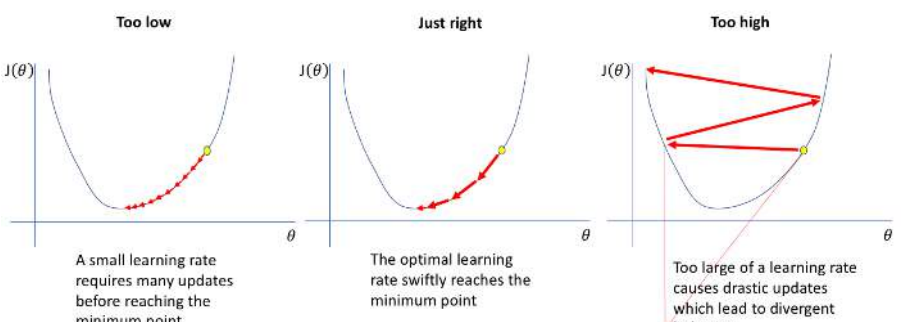
- Initialize $\mathbf{w}_0 = 0$
- For i from 0 to M:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \nabla L(x; \mathbf{w}_i)$$

  where η is the **step size** , also called the **learning rate**
- M is the maximum number of iterations
- Until stopping criterion satisfied
  - When gradient $\|\nabla L(\mathbf{w})\|_2 \leq \epsilon$ for some pre-set ϵ (Recall $\nabla L(\mathbf{w}) = 0$ at minimum)
  - Evaluate the performance on validation data, and stop when not improving

Gradient Descent: Learning Rate

- Step size or learning rate (η) is a hyperparameter
- Small η leads to slow convergence, but large η leads to divergence
- Different step sizes have to be tried



Local and Global Minimum

- Convergence is not guaranteed and one may get stuck in a local minimum
- Many extensions of this algorithm exist while the basic idea stays the same
- **Stochastic gradient descent** Instead of using the entire sample to compute the gradient, SGD uses a small random sample in each iteration

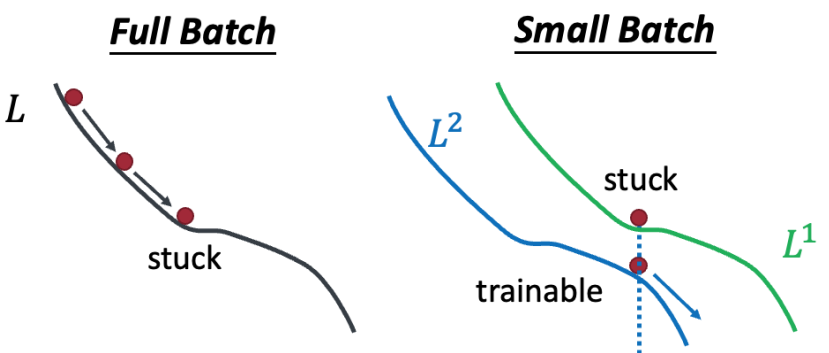
Stochastic Gradient Descent

- SGD is a variant of gradient descent where the gradient is computed using a single training example (or a small batch), rather than the entire dataset

$$\mathbf{w}_{i+1} := \mathbf{w}_i - \eta \cdot \nabla_\theta L(\mathcal{B}; \mathbf{w}_i)$$

- $\mathcal{B}$ — a randomly selected mini-batch of training samples
- Balances efficiency and stability: faster than full-batch, less noisy than single-sample SGD
- Common batch sizes: 32, 64, 128, etc.

Full Batch vs. Small Batch



Part II

A Simple Neural Network

EXAMPLE

- Input: $X \in \mathbb{R}$; Output: $Y \in \mathbb{R}$
- Consider the following functional form:

$$f(x) = w_0 + w_1 h_1(x) + w_2 h_2(x) + w_3 h_3(x),$$

where

$$h_i(x) = \sigma(v_i x + b_i) = \frac{1}{1 + e^{-(v_i x + b_i)}} \text{ for } i = 1, 2, 3,$$

EXAMPLE

- Input: $X \in \mathbb{R}$; Output: $Y \in \mathbb{R}$
- Consider the following functional form:

$$f(x) = w_0 + w_1 h_1(x) + w_2 h_2(x) + w_3 h_3(x),$$

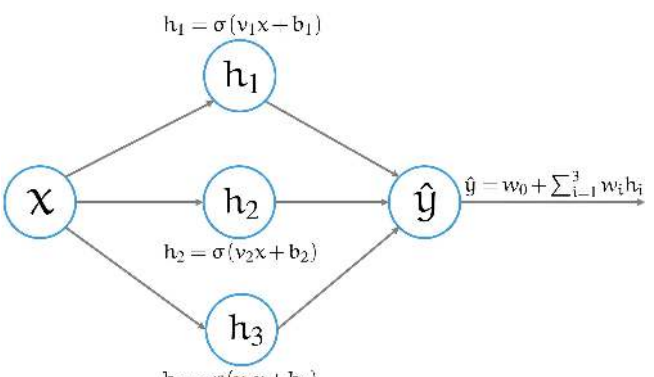
where

$$h_i(x) = \sigma(v_i x + b_i) = \frac{1}{1 + e^{-(v_i x + b_i)}} \text{ for } i = 1, 2, 3,$$

- **This is a neural network!**

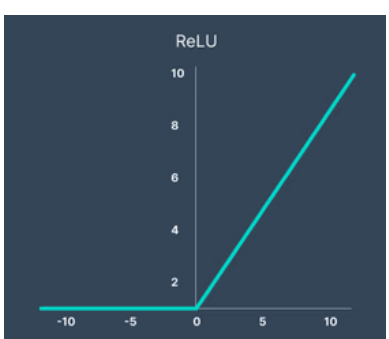
A Simple Neural Network

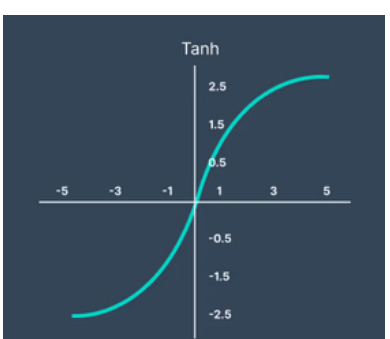
- Neural network with a single 3-neuron hidden layer
- $\sigma(\cdot)$ is a fixed nonlinear **activation function**



Activation Functions

- Rectified linear function (ReLU): $\sigma(x) = \max(0, x)$
  - Fast to calculate, and to calculate its derivatives
- Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
  - The range of the sigmoid function is (0,1), consistent with a probability
- Hyperbolic tangent function: $\sigma(x) = \tanh(x)$



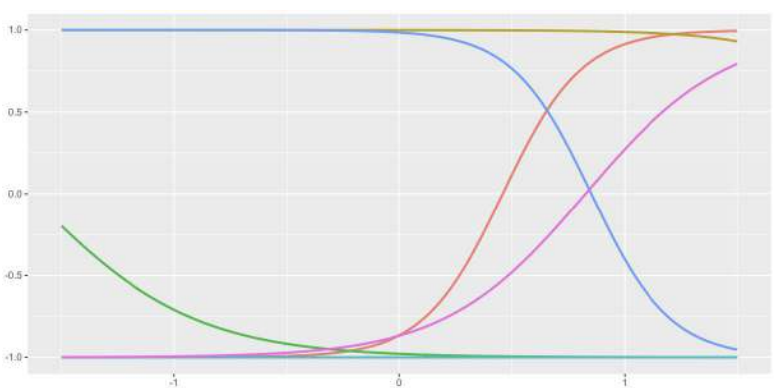


Hidden Layer as Feature

- Parameters in the example neural network:

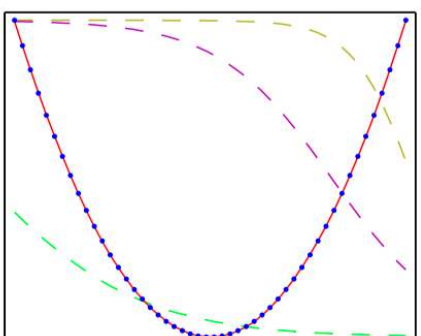
$$b_1, b_2, b_3, v_1, v_2, v_3, w_0, w_1, w_2, w_3 \in \mathbb{R}$$

- Can think of $h_i = h_i(x) = \sigma(v_i x + b_i)$ as a feature of x
  - Learned by fitting the parameters v_i and b_i
- The following figure illustrates some $h_i(x)$'s for $\sigma = \tanh$ and randomly chosen v_i and b_i:



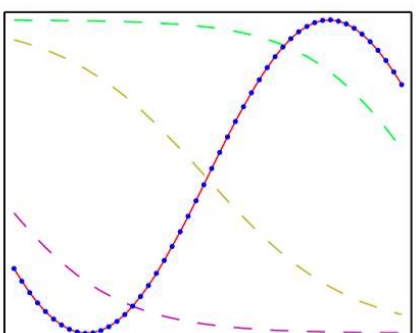
Approximation Ability: $f(x) = x^2$

- 3 hidden units; `tanh` activation functions
- Blue dots are training points
- Dashed lines are hidden unit outputs
- Final output in Red



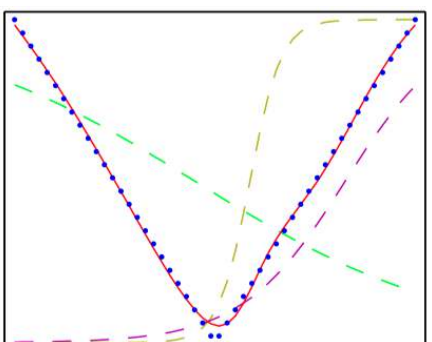
Approximation Ability: $f(x) = \sin(x)$

- 3 hidden units; `logistic` activation function
- Blue dots are training points
- Dashed lines are hidden unit outputs
- Final output in Red



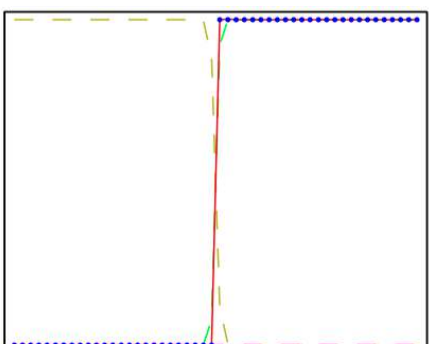
Approximation Ability: $f(x) = |x|$

- 3 hidden units; `logistic` activation functions
- Blue dots are training points
- Dashed lines are hidden unit outputs
- Final output in Red



Approximation Ability: $f(x) = 1(x > 0)$

- 3 hidden units; `logistic` activation function
- Blue dots are training points
- Dashed lines are hidden unit outputs
- Final output in Red



Universal Approximation Theorem

- **Universal Approximation Theorem** proves that a neural network with one hidden layer can uniformly approximate any continuous function on a compact set if the activation function is not a polynomial
- More concretely:
  - Let $\sigma(\cdot)$ be any nonlinear activation function
  - Let $f : K \to \mathbb{R}$ be any continuous function on a compact set $K \subset \mathbb{R}^m$
  - Then $\forall \epsilon > 0$, there exists an integer N (the number of hidden units), and parameters $v_i, b_i \in \mathbb{R}$ and $w_i \in \mathbb{R}^m$ such that the function

$$F(x) = \sum_{i=1}^{N} v_i \sigma(w_i^T x + b_i)$$

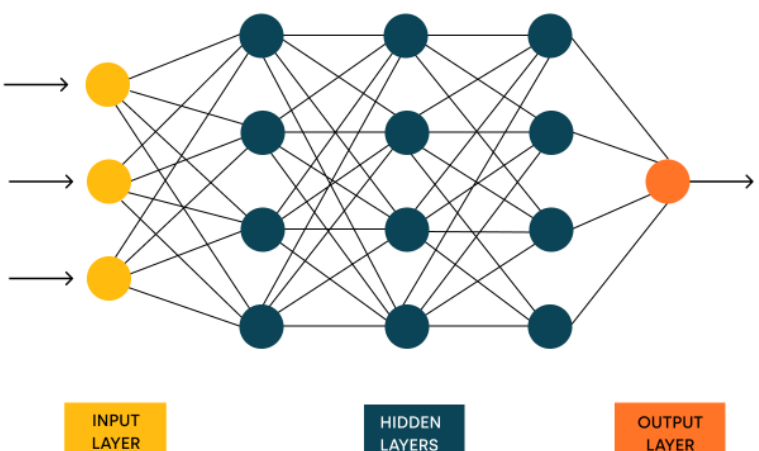
  satisfies $|F(x) - f(x)| < \epsilon$ for all $x \in K$

Part III

Neural Networks

TERMINOLOGIES

- Input layer, hidden layers, output layer
- Neurons: Input neurons, hidden neurons, output neuron
- Activation function: `ReLU`, `Sigmoid`, `tanh`, etc.



Computing Output: Input Layer

- Input neurons take the values of the predictors as input
- The output of an input neuron is the same as the input
- The number of input neurons equals the dimension of the predictors

Computing Output: Hidden Layer

Let's focus on layer $\mathcal{L}(m+1)$; For neuron $j \in \mathcal{L}(m+1)$

- For neuron $j \in \mathcal{L}(m+1)$, calculate the weighted sum of all neurons in the previous layer

$$z_j^{(m+1)} = \sum_{i \in \mathcal{L}(m)} w_{ji} y_i^{(m)} + b_j^{(m+1)}$$

  where $b_j^{(m+1)}$ is a constant bias term specific for neuron j

- Apply the activation function to $z_j^{(m+1)}$, which gives the output of neuron j

$$y_j^{(m+1)} = \sigma(z_j^{(m+1)}) = \sigma\left(\sum_{i \in \mathcal{L}(m)} w_{ji} x_i^{(m)} + b_j^{(m+1)}\right)$$

- The activation functions enable neural networks to capture nonlinear relationships among predictors and outcomes

Computing Output: Output Layer

- The last layer, i.e., $\mathcal{L}(M)$, or output layer, is special since its output represents the final prediction
- Regression task The final layer is just a single neuron with no activation

$$y^{(M)} = z^{(M)}$$

- Classification task with 2 classes The final layer is a single neuron with the sigmoid activation function

$$y^{(M)} = \texttt{sigmoid}\left(z^{(M)}\right) = \frac{1}{1 + e^{-z^{(M)}}}$$

- Classification task with K classes The final layer is composed of K neurons and a softmax function that combines their outputs through normalization

$$y_i^{(M)} = \texttt{softmax}(z_i^{(M)}) = p_i(z_i^{(M)}) \equiv \frac{e^{z_i^{(M)}}}{\sum_{j=1}^{K} e^{z_j^{(M)}}}, \quad i = \{1, \ldots, K\}$$

Multitask Learning

- Suppose $X = \{\text{Images}\}$
- We have two tasks
  - Does the image have a car?
  - Does the image have a pedestrian?
- Assign one output neuron for each task
  - Both neurons can output 1 simultaneously
- Objective function must combine losses from both predictions, e.g., by averaging
- Learning on the same neural network benefits both tasks

Parameter Estimation

- Denote the neural network as $\mathcal{N}(x; \mathbf{w})$, where $\mathbf{w}$ are all the parameters (including weights and biases) and x is the training sample
- Denote the loss function as $\mathcal{L}(\mathbf{w})$
  - For regression, one typically minimizes the sum of squared error

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \mathcal{N}(x_i; \mathbf{w}))^2$$

  - For classification, one typically minimizes the cross-entropy

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log \mathcal{N}_k(x_i; \mathbf{w})$$

- We use gradient descent or its variants to estimate the parameters

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

More Concepts in Neural Network

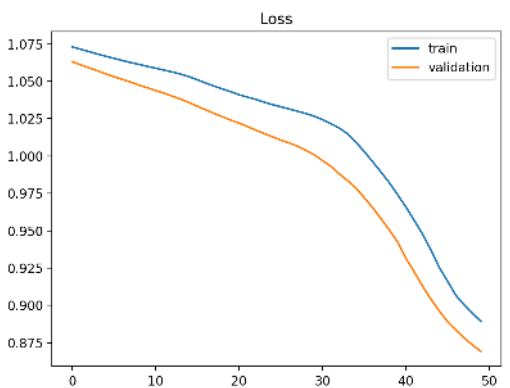
- **Forward Propagation** The process of passing inputs through the network to get an output prediction
- **Backward Propagation** The method of updating weights by calculating and propagating the error (gradient) backward through the network
- **Batches** Small subsets of the training data used to update the model weights during training
  - Typically, we sample the batches at random (without replacement)
- **Epochs** An epoch is one complete pass through the entire training dataset

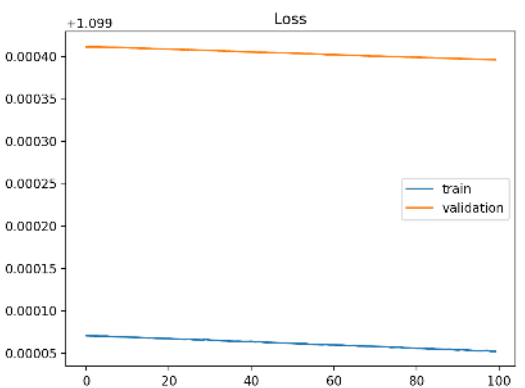
Learning Curve

- **Learning Curve** A learning curve is a plot that shows the number of epochs on the x-axis and the loss values on the y-axis
- Learning curves of model performance on the training and validation datasets can be used to diagnose an underfit, overfit, or well-fit model[1]

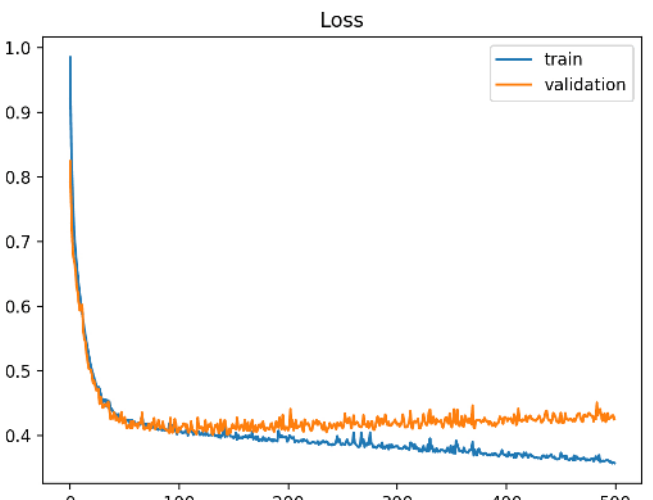
[1] "How to use Learning Curves to Diagnose Machine Learning Model Performance." Jason Brownlee

Learning Curve: Underfit

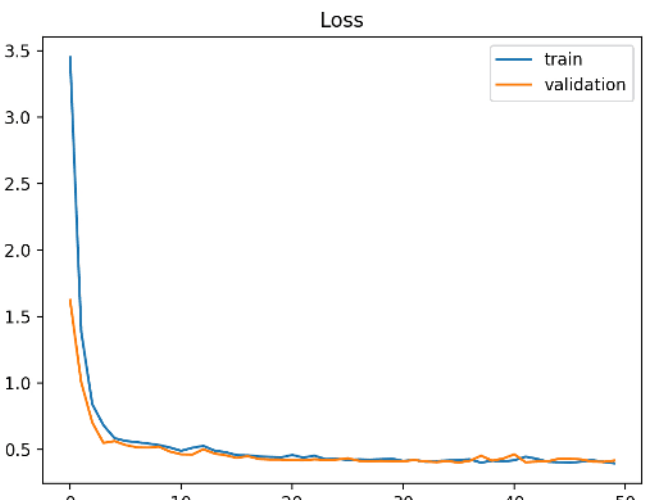




Learning Curve: Overfit



Learning Curve: Well Fit



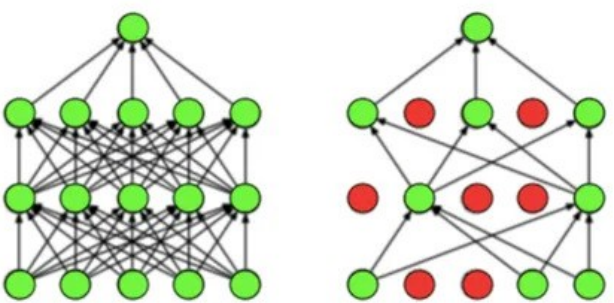
Hyperparameters

- The architecture of the neural network
  - The number of hidden layers
  - The number of hidden neurons in each hidden layer
  - The choice of the activation function
- Training a neural network
  - Batch size
  - Learning rate

Regularization

- **L1- and L2-Regularization** Add a penalty to the loss function to reduce model complexity and prevent overfitting
- **Early Stopping** Stop training when performance on a validation set stops improving to prevent overfitting
- **Dropout** Randomly ignore a fraction of neurons during training to improve generalization

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Dropout in More Details

- Dropout during training
  - In each forward pass during training, dropout randomly disables (or “drops") a specified fraction of neurons in a layer by setting the neurons to zero
  - The remaining active neurons are scaled by a factor (often $\frac{1}{1-\text{dropout rate}}$) to ensure that the output has the correct expected value
- Dropout during inference: When making predictions on new data, dropout is not applied
- Dropout helps the network generalize by preventing it from relying too heavily on specific neurons, reducing the likelihood of overfitting

Part IV

Feature Importance

What Does Neural Network Learn

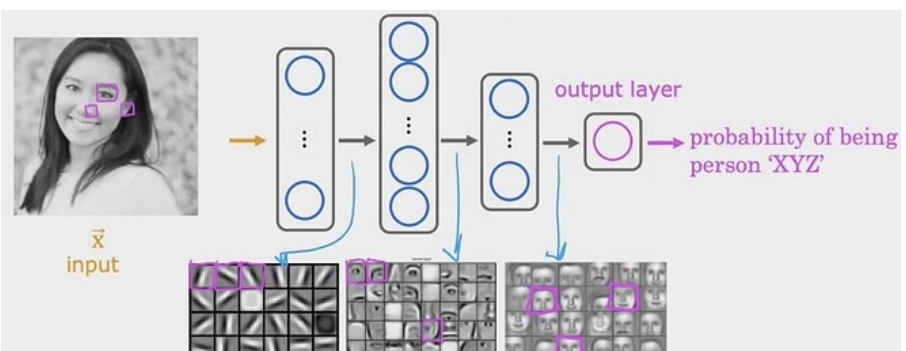


Figure. Example of Face Recognition

Feature Importance

- **Feature Importance** How important is the feature for the prediction?

- Consider a linear regression model

$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_j x_j + \cdots + \beta_p x_p$$

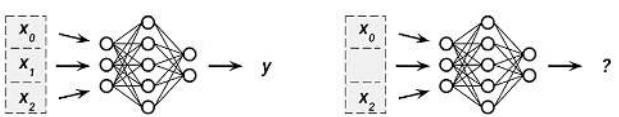
- The contribution of feature j is simply $\beta_j x_j$
- Consider a linear regression model with an interaction term $\beta^* x_j x_i$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_j x_j + \cdots + \beta_p x_p + \beta^* x_j x_i$$

- The contribution of feature j depends on feature i, i.e., $\beta_j x_j + \beta^* x_j x_i$
- For more complex models (e.g., random forests, neural networks), feature importance is even less straightforward

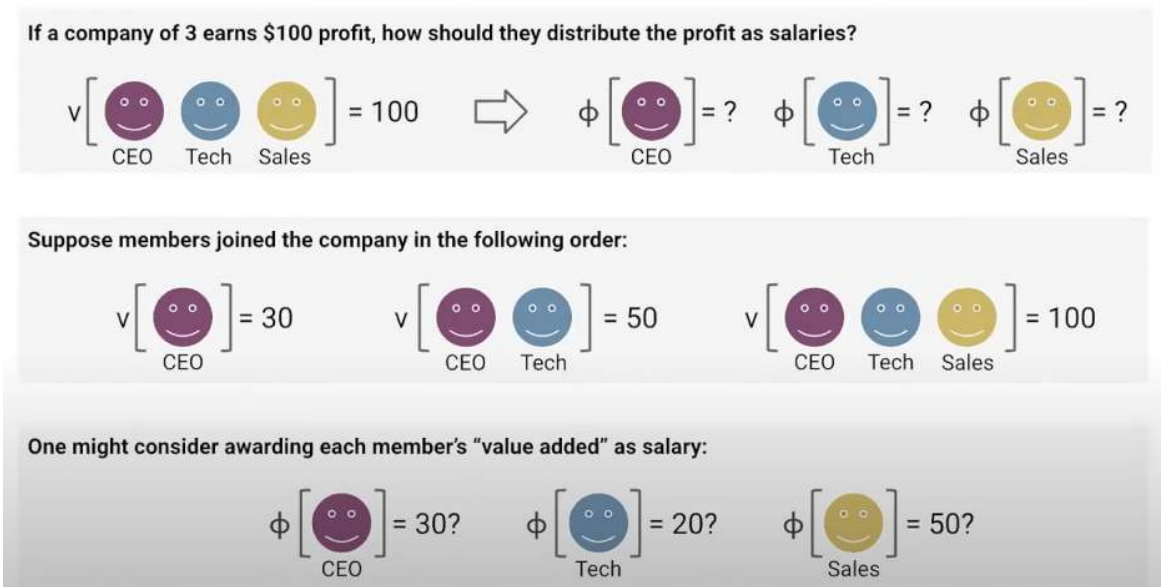
SHAPLEY VALUES

- **Shapley values** attribute a model's prediction to the model's input features
- Shapley values measure the importance of feature x_j by testing how much the model output changes when x_j is withheld from the input



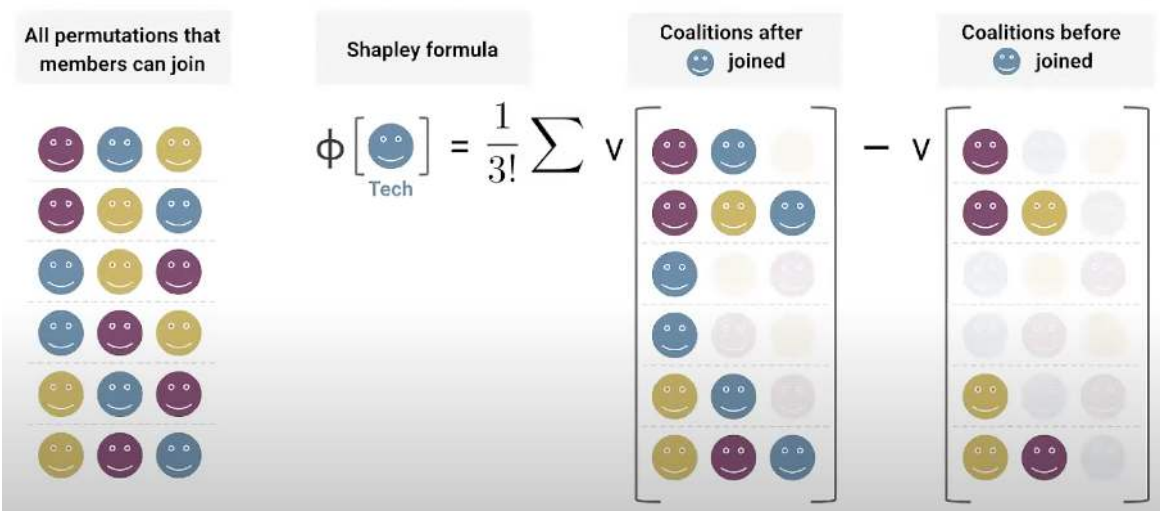
- Shapley values come from cooperative game theory
  - Each feature is treated as a "player" in a game where the prediction is the payout
  - The Shapley value provides a way to fairly distribute the prediction (i.e., the payout) among the features (i.e., the players)

Shapley Values: A Game Theory Example



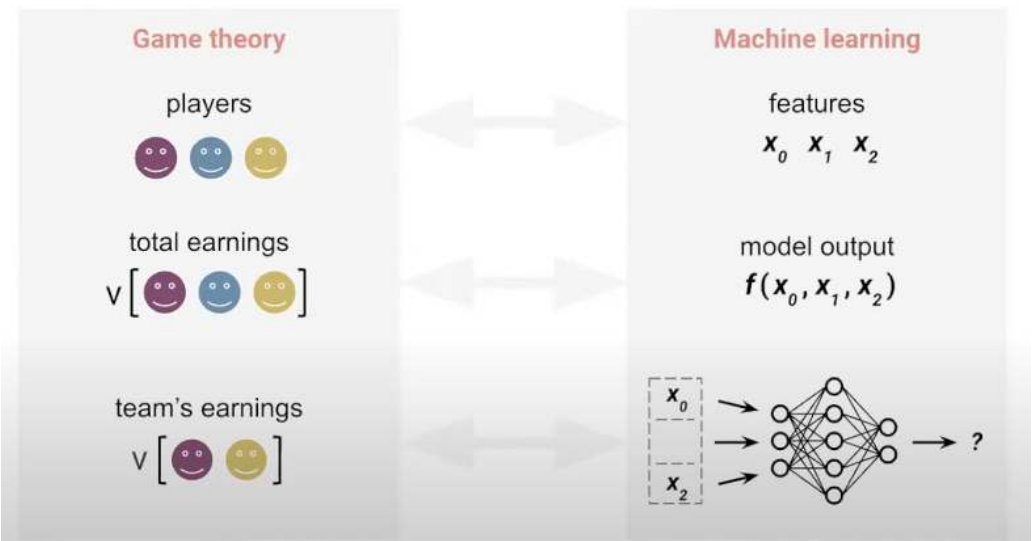
SHAPLEY VALUES: A GAME THEORY EXAMPLE

Shapley values are defined through the average over all permutations



SHAPLEY VALUES: A GAME THEORY EXAMPLE

Adapting Shapley values for feature importance in machine learning



Shapley Values: Calculation

- The Shapley value is the average marginal contribution of a feature across all possible coalitions
- The contribution of each feature is computed by comparing predictions with and without the feature in different feature combinations
- The Shapley value for a feature is defined as

$$\phi_j = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[v(S \cup \{j\}) - v(S) \right],$$

where:

- S is a subset of features
- $v(S)$ is the prediction given only features in S
- F is the full set of features

SHAPLEY VALUES: ESTIMATION

- Since the calculation of Shapley values requires all possible permutations among all features, the computation grows exponentially with the number of features, which is practically intractable
- There are many ways to numerically estimate the Shapley values
- `Python` package `SHAP` provides the numerical estimation of Shapley values

VISUALIZATIONS IN SHAP

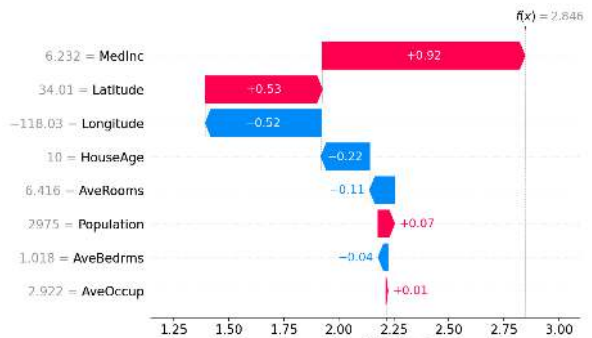


Figure. Waterfall Plot

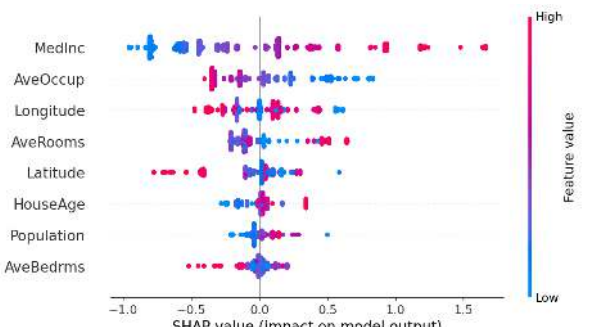


Figure. Summary Plot

Visualizations in SHAP

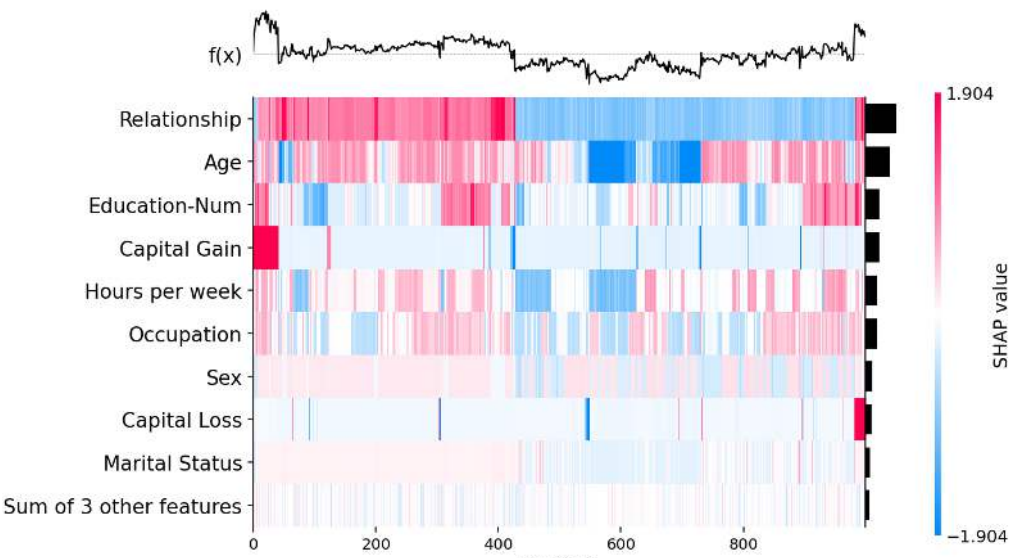
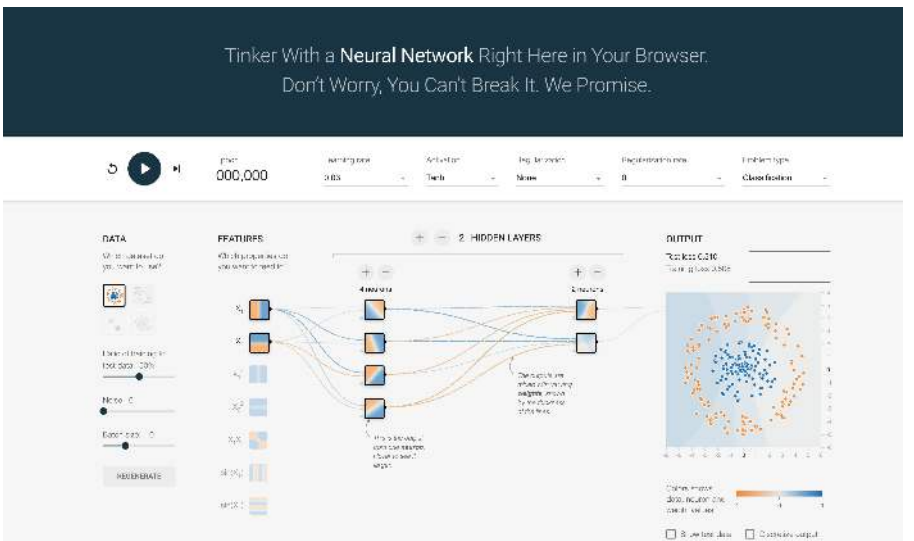


Figure. Heatmap

Explainable Machine Learning

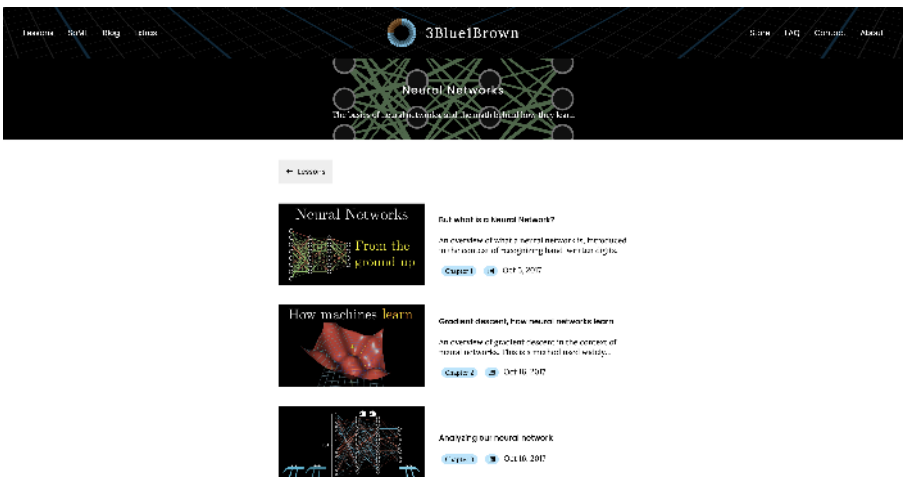
- Shapley values are one of many approaches towards **explainable machine learning**
- Human-understandable explanations of ML predictions are advantageous in several ways
- Consider the case of a denied credit card application, explainability helps
  - instruct the applicant how to improve success rate
  - challenge the decision if unfair treatment is assumed (“algorithmic fairness")
  - increase trustworthiness in the system and AI

Recommended Resource: A Neural Network Playground



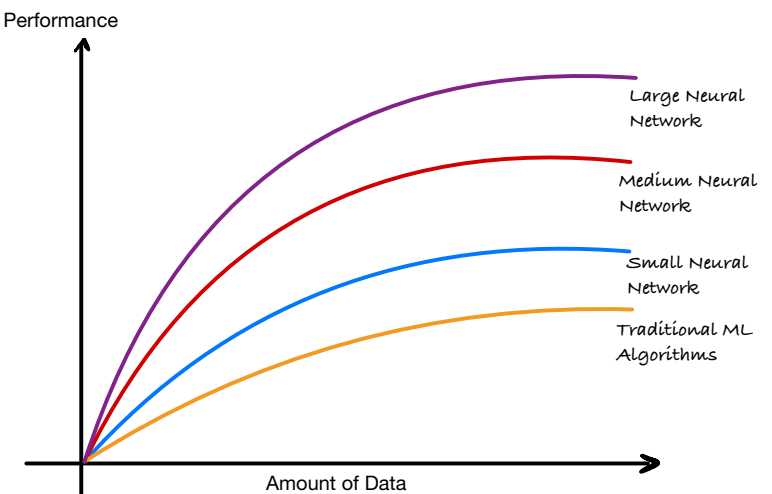
Neural Network Visualization

Recommended Resource: 3Blue1Brown



"The basics of neural networks, and the math behind how they learn."
3Blue1Brown

Neural Networks Benefit from Big Data



Large Language Models

- Large language models are a good example of the increasing capability of neural networks with larger sizes
- Llama is an open-source large language model developed by Meta
  - Llama 1 has 65 billion parameters (released in 2022)
  - Llama 2 has 70 billion parameters (released in 2023)
  - Llama 3 has 405 billion parameters (released in 2024)
- Increasing the size of training data and the model parameters continues to improve the model performance; and we haven't observed the limit of large language models so far

Deep Learning

- Fully-connected neural networks are the building blocks of modern-day deep learning algorithms
  - Many more layers
  - Much more complex network architecture
- Images: Convolutional Neural Networks (CNN)
- Textual data: Recurrent Neural Networks (RNN), Long-Short Term Memory (LSTM), Transformer
- The foundation and application of Large Language Models (LLMs)

- **FA690 Machine Learning in Finance**