Research: Deep Learning Algorithms

HON 401

Don Li (Depts. Computer Science, Mathematics)

Supervised by Prof. Dacian Daescu (Dept. Mathematics)

University Honors College (UHC)

Portland State University

June 12, 2025

Abstract

This talk is given in satisfaction of the requirements of University Honors College (UHC) research course, HON 401. Neural networks are the engine of the current AI boom. Given recent major advancements, we give an introductory overview of deep learning and neural networks, particularly for classification tasks. Moreover, we discuss stochastic gradient descent and backpropagation, i.e., the key algorithms that underlie the training of a neural network, and discuss these algorithms from the standpoint of an optimization problem. We supplement the relevant theory with Python implementations of binary and multi-class classification neural networks.

Portland State UNIVERSITY

Contents

Portland State UNIVERSITY

AI, Machine Learning, & Deep Learning

- **Artificial Intelligence (AI)** – "the science and engineering of making intelligent machines" (McCarthy, 1955)
  - AI programs are those that have capabilities beyond that which they are explicitly programmed to have
- **Machine Learning (ML)** – the subset of AI concerned with learning from/making predictions about a *target variable* from *feature variables* (either *supervised* or *unsupervised*)
- **Deep Learning** – the subset of ML concerned with a specific class of ML models called *neural networks*, inspired by neurons in human/animal brains

Portland State UNIVERSITY

History of ML/DL

- 1950 – Alan Turing developed the Turing Test to test a machine's ability to exhibit intelligent behavior
- 1956 – AI accepted as field at Dartmouth Conference (term AI coined at Dartmouth Conference)
- 1957 – Frank Rosenblatt invents the perceptron algorithm, which was the precursor to modern neural networks
- 1960's-70's – The first "AI Winter" due to doubts regarding viability of AI for machine translation

Portland State UNIVERSITY

History of ML/DL (Continued)

- 1980's – AI regains interest as rule-based expert systems that ran on mainframe computers emerge as major AI advancement
- 1986 – Backpropagation algorithm is introduced, which renews interest in neural networks/deep learning
- Late 1980's-1990's – 2nd AI Winter due to reduced business viability of rule-based expert systems as well as neural networks (backpropagation did not scale well to large datasets)

Portland State UNIVERSITY

History of ML/DL (Continued)

- 1990's-2000's – AI started to find applications in speech recognition, medical diagnosis, robotics, and other areas
- 1997 – In a rematch, IBM's Deep Blue supercomputer defeats then-World Chess Champion Garry Kasparov
- Late 1990's – Google emerges as dominant search engine with its AI PageRank algorithm
- 2006 – Geoffrey Hinton publishes a paper on un-supervised pre-training that allows deeper neural networks to be trained
- 2014 — A team at Stanford creates a computer vision algorithm that can describe photos
- 2016 – DeepMind's AlphaGo, developed by Aja Huang, beats Go master Lee Se-dol

Portland State UNIVERSITY

A Primer on Machine Learning (ML)

- An ML model is concerned with making predictions about a *target variable* (its output) from *feature variables* (its inputs)
- A *supervised* ML model makes predictions with *labeled* data that represent observations the model learns from (inductive inference)
- Building an ML model involves a train/test split (to avoid overfitting)
- Thought experiment – in which city are you more likely to see a car with a WA state license plate: Portland or Eugene?
  - For a hypothetical state license plate prediction model, geographic location would be a relevant feature variable!

Portland State UNIVERSITY

ML Tasks – Regression & Classification

- There are two major tasks in machine learning (ML): **regression** and **classification**
- **Regression** models are concerned with making predictions about a *continuous* target variable (e.g., future share price of a stock, a basketball player's average PPG at end of season)
- **Classification** models are concerned with making predictions about a *discrete/categorical* target variable (e.g., email spam detection, credit card fraudulent transaction detection)
- Regardless of model type, core of ML is mapping relationship between feature variables and target variable in dataset, then using relationship to make predictions

Regression Example – Ames Housing Dataset

- This is a classic dataset in ML/data science used to train regression models
- Suppose we want to be able to predict the sale price of a home in Ames, IA
- We have a dataset with 81 different feature variables (e.g., square footage, number of bedrooms) with labeled target variable (sale price in dollars)
- **Objective** Build an ML model that can accurately predict the sale price of a given home in Ames, IA

```
(base) donli@Dons-MBP Programming Practice  % python regression_example_hon401.py
   Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  ... Misc Feature Misc Val Mo Sold Yr Sold Sale Type Sale Condition SalePrice
0      1  526301100           20        RL         141.0     31770   Pave  ...          NaN        0       5    2010       WD          Normal    215000
1      2  526350040           20        RH          80.0     11622   Pave  ...          NaN        0       6    2010       WD          Normal    105000
2      3  526351010           20        RL          81.0     14267   Pave  ...         Gar2    12500       6    2010       WD          Normal    172000
3      4  526353030           20        RL          93.0     11160   Pave  ...          NaN        0       4    2010       WD          Normal    244000
4      5  527105010           60        RL          74.0     13830   Pave  ...          NaN        0       3    2010       WD          Normal    189900
```

Figure: Ames, IA Housing Dataset

Portland State UNIVERSITY

Classification Example – Startup Acquisition Dataset

- Suppose a Venture Capital (VC) firm wants to be able to predict which startups to invest in based on probability of future acquisition
- We have a dataset with 48 different feature variables (e.g., number of funding rounds, industry/sector type) with labeled target variable ("acquired/closed")
- **Objective** Build an ML model that can accurately predict whether a startup will succeed/fail

```
(base) donli@Dons-MBP Programming Practice  % python classification_example_hon401.py
   Unnamed: 0 state_code   latitude   longitude zip_code       id  ... has_roundB has_roundC has_roundD  avg_participants is_top500    status
0        1005         CA  42.358880  -71.056820    92101   c:6669  ...          0          0          0            1.0000         0  acquired
1         204         CA  37.238916 -121.973718    95032  c:16283  ...          1          1          1            4.7500         1  acquired
2        1001         CA  32.901049 -117.192656    92121  c:65620  ...          0          0          0            4.0000         1  acquired
3         738         CA  37.320309 -122.050040    95014  c:42668  ...          1          1          1            3.3333         1  acquired
4        1002         CA  37.779281 -122.419236    94105  c:65806  ...          0          0          0            1.0000         1    closed

[5 rows x 49 columns]
```

Figure: Startup Acquisition Dataset

Portland State UNIVERSITY

Common ML (Non-DL) Algorithms

What types of ML (non-DL) models could we possibly build to answer these sorts of questions?

- **Regression**
  - Linear Regression, Polynomial Regression, Lasso ($L1$) Regression, Ridge ($L2$) Regression, Elastic Net ($L1 + L2$) Regression
- **Classification**
  - Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machines (SVM's), Naive Bayes, Random Forests

Portland State UNIVERSITY

Introduction to Deep Learning

- Recall that *deep learning* (DL) is a subset of ML in which ML tasks are performed using a *neural network* (NN)
- An NN consists of *nodes* in layers, where output of previous layers are inputs of subsequent layers
- An NN can be thought of as a directed graph with nodes and weighted edges
- An NN must have an input layer and output layer, may also have *hidden layers*

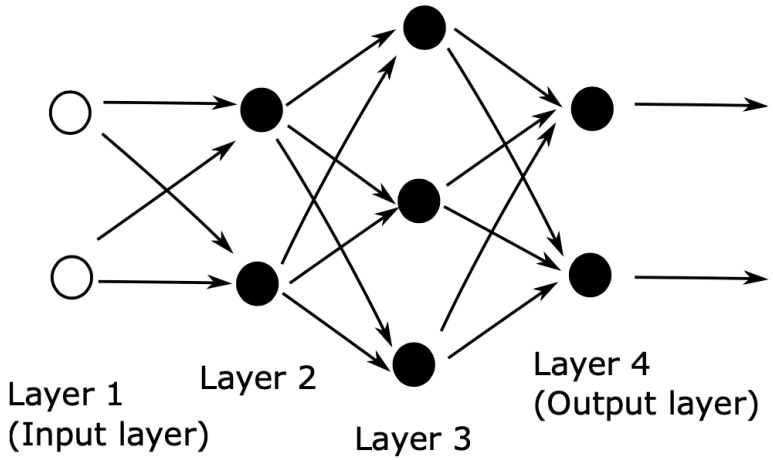


Fig. 2.3 *A network with four layers.*

Portland State UNIVERSITY

Neurons as Units of Cognition

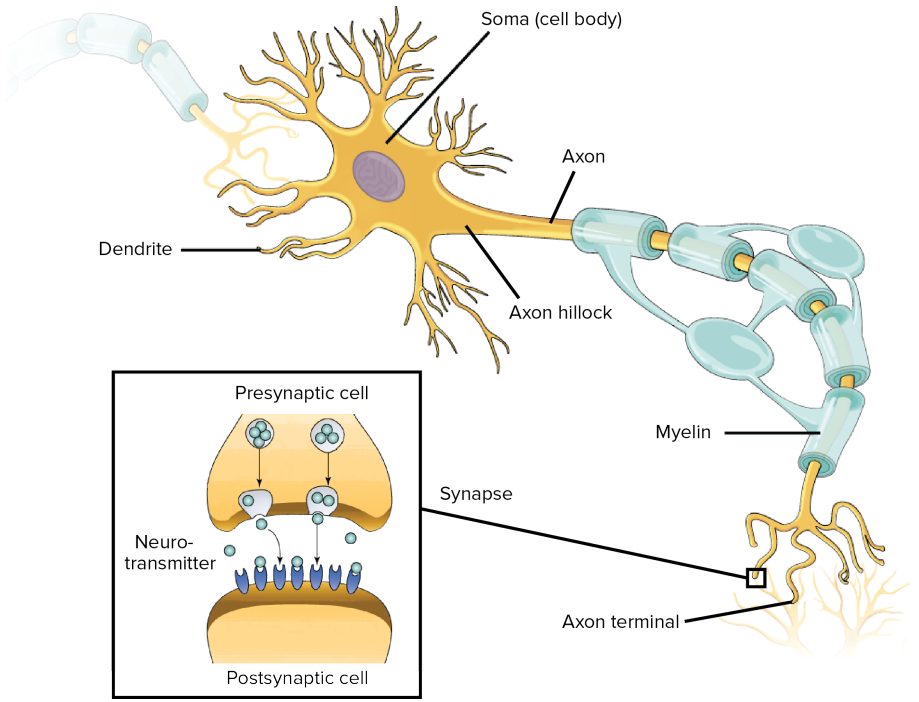


Figure: Anatomy of a Neuron (from Khan Academy)

Portland State UNIVERSITY

Neural Network Basics

- The input layer of the network passes the sum of weighted inputs and a *bias* to the next layer
  - $Z = Wx + b$
- This output Z is then the input of an *activation function*, then the output of the activation function is passed as input to the subsequent layer
- This process is iterated across the layers of the network, called a *forward pass*

Portland State UNIVERSITY

Sigmoid Function as Activation Function

- Note how $Z = Wx + b$ is a linear function
  - But many real-world phenomena (and thus data collected re: such phenomena) are non-linear
- We can still use $Z = Wx + b$ to capture non-linear phenomena via an activation function (i.e., $\sigma(z)$)
- Here, we use the *sigmoid function* $\sigma(x)$ as our activation function

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

Portland State UNIVERSITY

Why Use the Sigmoid Function?

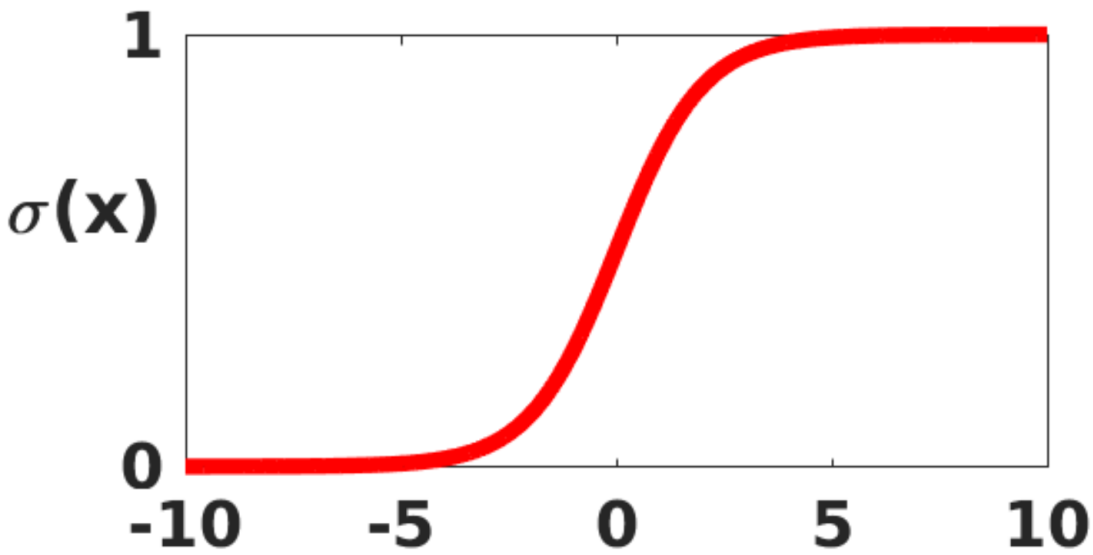


Figure: Graph of $\sigma(x)$

Portland State UNIVERSITY

Why Sigmoid Function?

- It has easy-to-compute derivative (i.e., $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Particularly good for (binary) classification because $\sigma(x) : x \to [0, 1]$
  - Why? Note that $\lim_{x \to -\infty} \sigma(x) = 0$ and $\lim_{x \to \infty} \sigma(x) = 1$
- Smooth, gradual increase of $\sigma(x)$ near 0 preserves nuance of classification around decision boundary
  - Common binary classification decision rule for $\sigma(x)$: if $\sigma(x) \leq 0.5$, then output is 0; otherwise, output is 1

Portland State UNIVERSITY

Rosenblatt's Perceptron

- Simplest possible neural network (i.e., NN with only single neuron)

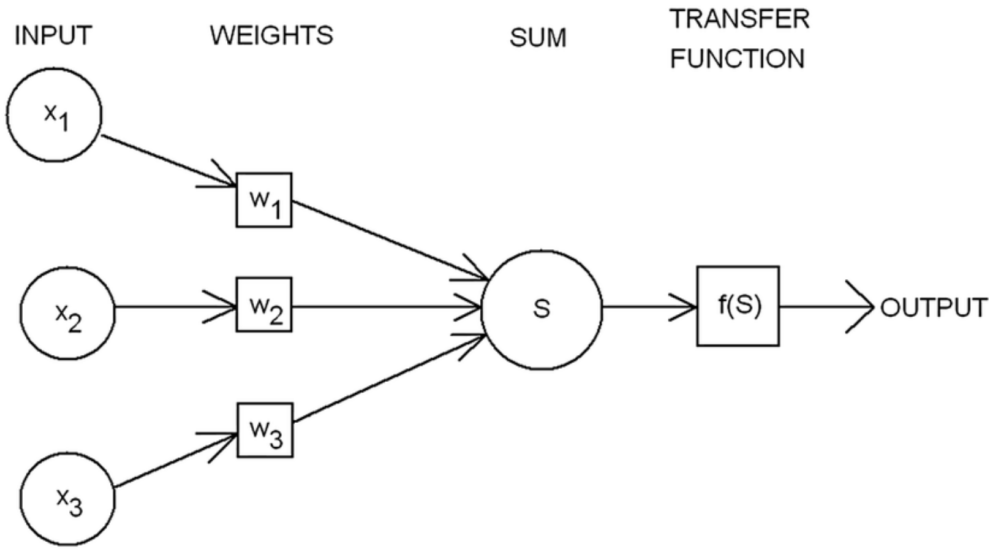


Figure: Rosenblatt's Perceptron (Alves, 2006)

Portland State UNIVERSITY

Rosenblatt's Perceptron

- Recall from the previous diagram that this perceptron model has three inputs: x_1, x_2, x_3
- We also introduce *weights* for each respective input, each of which represent the respective significance of that input on the output (here, W_1, W_2, W_3)
- Output Z is either 0 or 1 as a function of weighted sum of $\Sigma_i w_i x_i$ in relation to some *threshold value* or *bias* b

$$Z = \begin{cases} 0 & \text{if } \Sigma W_i x_i \leq b \\ 1 & \text{if } \Sigma W_i x_i > b \end{cases} \tag{2}$$

Binary Classification Perceptron – A Simple Example

Suppose I want to attend a PSU basketball game, but am not sure if I will go. How can we use the perceptron model to make a decision?
Let 1 encode attending the game and 0 encode not attending the game. I base my decision on these three (binary) variables:

1. Does the game not start late in the evening? (x_1)
2. Will I have at least one friend to join me? (x_2)
3. Do I not have any unfinished assignments due the next day? (x_3)

Portland State UNIVERSITY

Binary Classification Perceptron – A Simple Example

- Let $x_1 = 1$ if the game does not start late in the evening, $x_1 = 0$ if it does
- Let $x_2 = 1$ if I do have at least one friend to join me, $x_2 = 0$ if not
- Let $x_3 = 1$ if I've finished all my coursework for the day, $x_3 = 0$ if not
- I commute to campus and don't want to commute back home too late at night, so I set $W_1 = 2$
- I'd prefer to go with a friend but am fine with going by myself, so I set $W_2 = 1$
- I care about my studies and don't want to cram schoolwork, so I set $W_3 = 3$
- Finally, set threshold/bias $b = 3$

Portland State UNIVERSITY

Binary Classification Perceptron – A Simple Example

- Suppose the game starts late in the evening ($x_1 = 0$), I do have at least one friend to join me ($x_2 = 1$), and my schoolwork is finished ($x_3 = 1$)
- Then
  $\Sigma W_i x_i = W_1 x_1 + W_2 x_2 + W_3 x_3 = (2)(0) + (1)(1) + (3)(1) = 4$
- Recall $b = 3$. Since $\Sigma W_i x_i = 4 > 3$, it follows $Z = 1$ which means that, by perceptron output, I choose to go to the game
- Notice how weights and bias(es) shape the output!

Portland State UNIVERSITY

Macro Algorithm for NN Training

1. **Forward Pass** – Pass inputs $x_1, x_2, \ldots$ into $Z = Wx + b$, then compute $\sigma(z)$, then pass $\sigma(z)$ to subsequent layer
   1. Weights and biases are randomized when initialized
   2. Perform iteratively across layers until reaching output layer to obtain set y_{pred}
2. **Compute Error** – Using specified function and y_{true} (i.e., labeled target variable data), compute *loss function* for each point in training set and *cost function* as average loss function across total training set
3. **Backpropagation** – Using calculus, find magnitude of change of weights and biases that minimizes cost function, then update weights and biases accordingly

Portland State UNIVERSITY

Parameters & Hyperparameters

The central task in training a neural network is to find the weights and biases that minimize the cost function

- A neural network is governed by its *parameters* and *hyperparameters*
- The weights and biases within the network are the parameters
- The network architecture (i.e., number of layers, number of nodes per layer), along with *learning rate* η and *epochs*, are the hyperparameters
  - η is a constant that governs magnitude of update of parameters per iteration
  - Epochs are the number of iterations in training

Portland State UNIVERSITY

Replicating Higham (2019)

Task Reproduce the reduction in cost function seen in Figure 6.1 from Higham (2019)

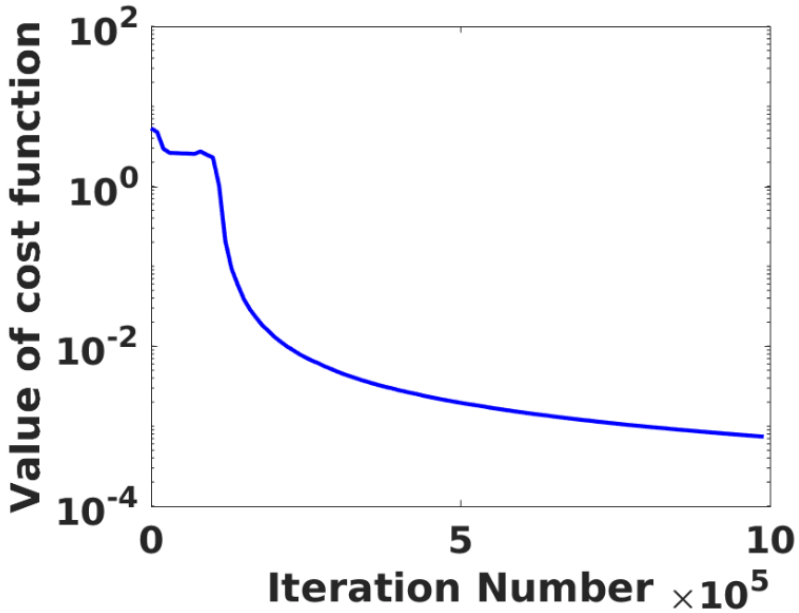


Fig. 6.1 *The vertical axis shows a scaled value of the cost function* (2.6). *The horizontal axis shows the iteration number. Here we used the stochastic gradient method to train a network of the form shown in Figure* 2.3 *on the data in Figure* 2.1. *The resulting classification function is illustrated in Figure* 6.2.

Replicating Higham (2019)

Task Reproduce the binary classification and decision boundary seen in Figure 6.2 from Higham (2019) (circles for Category A, crosses for Category B)

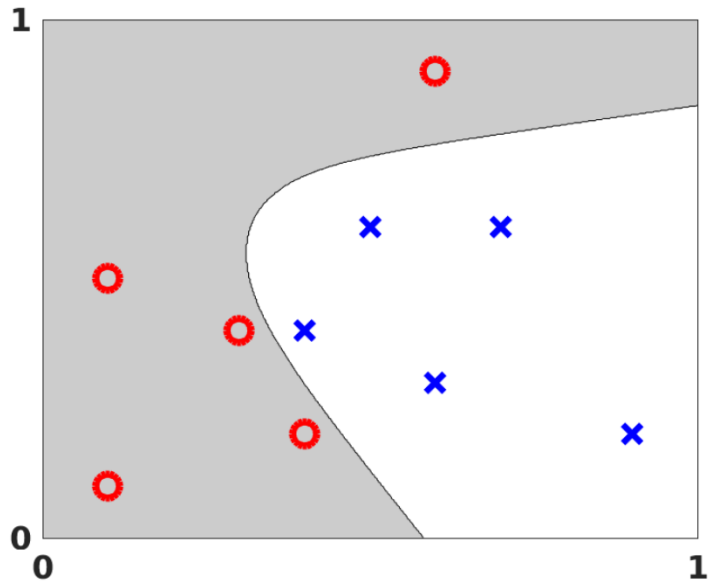


Fig. 6.2 *Visualization of output from an artificial neural network applied to the data in Figure* 2.1. *Here we trained the network using the stochastic gradient method with back propagation—behavior of the cost function is shown in Figure* 6.1. *The same optimization problem was solved with the* `lsqnonlin` *routine from MATLAB in order to produce Figure* 2.4.

Replicating Higham (2019)

This is our input/feature variable data from Higham (2019):
$x_1 = [0.1, 0.3, 0.1, 0.6, 0.4, 0.6, 0.5, 0.9, 0.4, 0.7]$
$x_2 = [0.1, 0.4, 0.5, 0.9, 0.2, 0.3, 0.6, 0.2, 0.4, 0.6]$
This is our output/target variable data from Higham (2019) (i.e., set of labels for supervised ML binary classification):

$$y_{true} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Recall that $y(x_i) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ if x_i is in Category A (red circles) and

$y(x_i) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ if x_i is in Category B (blue crosses)

Portland State UNIVERSITY

Loss Function (MSE)

- For forward pass, recall we have initially randomized parameters
  - Consequently, initial forward pass will not yield high accuracy
- We make the NN more accurate by making it less inaccurate (by minimizing cost function)
- For this project, we use *mean squared error* (MSE) as loss/cost function (here denoted as C)

$$C = \frac{1}{N} \sum_{i=1}^{N} (y_{true} - y_{pred})^2 \tag{3}$$

where N denotes sample size

NN Architecture for Our Project

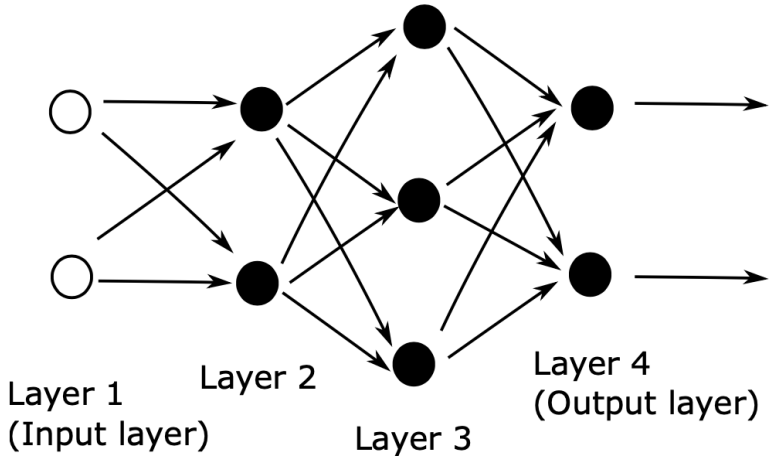


Fig. 2.3 *A network with four layers.*

- 4 layers (one input with two nodes, first hidden with two nodes, second hidden with three nodes, one output with two nodes)

Portland State UNIVERSITY

NN Architecture for Our Project

- $W^{[1]}$ as set of weights into first hidden layer, where $W^{[1]} \in \mathbb{R}^{2\times 2}$
- $b^{[1]}$ as bias vector into first hidden layer, where $b^{[1]}$ is column vector with two elements
- $W^{[2]}$ as set of weights into second hidden layer, where $W^{[2]} \in \mathbb{R}^{3\times 2}$
- $b^{[2]}$ as bias vector into second hidden layer, where $b^{[2]}$ is column vector with three elements
- $W^{[3]}$ as set of weights into output layer, where $W^{[3]} \in \mathbb{R}^{2\times 3}$
- $b^{[3]}$ as bias vector into output layer, where $b^{[3]}$ is column vector with two elements
- 16 total weights, 7 total biases

Portland State UNIVERSITY

Stochastic Gradient

- We can store entire set of parameters (weights and biases) into vector p (here, $p \in \mathbb{R}^{23}$)
- Our task is to minimize cost function $Cost(p)$ (here, $Cost(p) : \mathbb{R}^{23} \to \mathbb{R}$)
- The optimization/minimization algorithm we use to train a neural network is called *stochastic gradient*
  - How do choose perturbation Δp such that new vector (update) $p + \Delta p$ results in lower $Cost(p)$?

Portland State UNIVERSITY

(Non)-Convex Optimization

- Central task in optimization is to either minimize/maximize some objective function
  - For stochastic gradient for training NN's, the cost function is our objective function, and we wish to minimize the cost function
- Such objective functions can be either *convex* or *non-convex*
- Convex functions, by definition, are guaranteed to have a global minimum, whereas non-convex functions are not (the latter may have local minima that are not global minima)

Portland State UNIVERSITY

(Non)-Convex Optimization

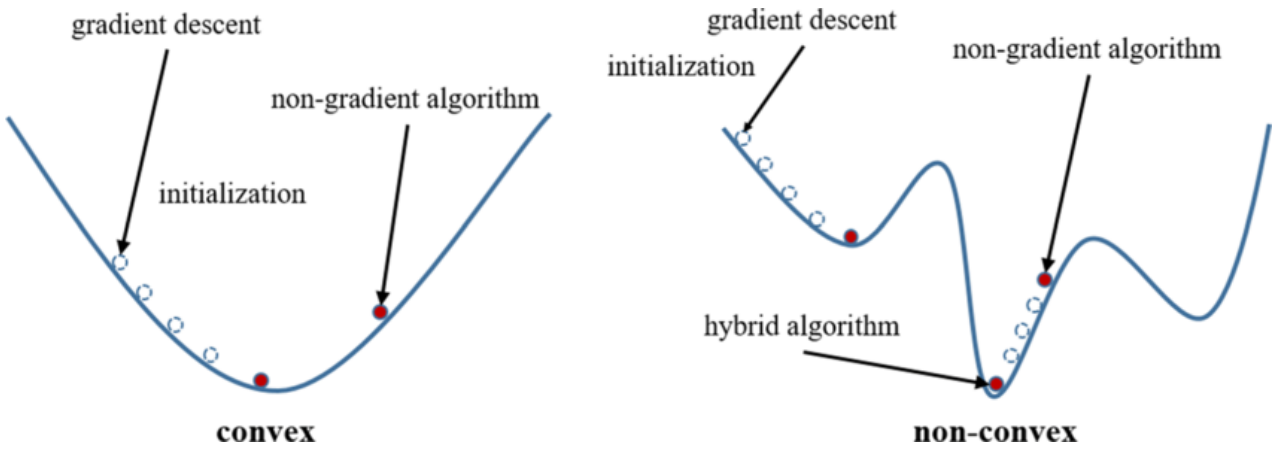


Figure: Convex vs. Non-Convex Cost Functions (Zhang & Xie, 2022)

Portland State UNIVERSITY

Stochastic Gradient Descent Analogy

Portland State UNIVERSITY

Stochastic Gradient

- Our task using stochastic gradient is to find Δp
- By Taylor Series expansion, $Cost(p + \Delta p) \approx Cost(p) + \sum_{r=1}^{s} \frac{\delta Cost(p)}{\delta p_r} \Delta p_r$, where $\frac{\delta Cost(p)}{\delta p_r}$ is partial derivative of cost function with respect to rth parameter
- We do this for each parameter via the *gradient* ∇ such that $\nabla(Cost(p)_r) = \frac{\delta Cost(p)}{\delta p_r}$
- This yields $Cost(p + \Delta p) \approx Cost(p) + \nabla Cost(p)^T \Delta p$

Portland State UNIVERSITY

Stochastic Gradient

- To minimize $Cost(p)$, we need to minimize $\nabla Cost(p)^T \Delta p$
- By Cauchy-Schwarz inequality, this occurs when Δp lies in same direction as $-\nabla Cost(p)$
- This yields our update for p

$$\boxed{p \to p - \eta \nabla Cost(p)}$$

Portland State UNIVERSITY

Simple Stochastic Gradient Method

- In practice, it is not computationally feasible to compute $\nabla Cost(p)$ for all input data, especially when there are many parameters
- One workaround is to perform the *simple stochastic gradient method*, whereby $\nabla Cost(p)$ is computed for a single, randomly chosen input data point for certain number of training iterations (either with or without replacement)
- For simple stochastic gradient, we choose random x_i from input data, then perform update $p \to p - \eta \nabla C_{x_i}(p)$

Portland State UNIVERSITY

Mini-Batch Stochastic Gradient Method

- Another approach to performing stochastic gradient is *mini-batch stochastic gradient*, whereby $\nabla Cost(p)$ is computed for a small subset of the input data
- For mini-batch stochastic gradient, we choose m integers, $k_1, k_2, ..., k_m$ (i.e., select subset), then perform update $p \rightarrow p - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla C_{x^{k_i}}(p)$
- If using mini-batch stochastic gradient, then m itself becomes a hyperparameter

Portland State UNIVERSITY

Backpropagation

- Recall that we need to perform the parameter update $p \to p - \eta \nabla Cost(p)$, which requires computing $\nabla Cost(p)$
  - How do we compute $\nabla Cost(p)$?
- Moreover, recall $\nabla Cost(p) = \langle \frac{\delta p}{\delta r_1}, \frac{\delta p}{\delta r_2}, ... \rangle$
- We compute all of the partial derivatives contained in $\nabla Cost(p)$ via *backpropagation*, i.e., computing the gradient with respect to each parameter via the chain rule from differential calculus

Portland State UNIVERSITY

Backpropagation Formulas

Let L denote the output layer of the NN, and ℓ denote any hidden layer (i.e., $0 \leq \ell \leq L-1$). Then we obtain

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^L - y) \tag{4}$$

$$\delta^{[\ell]} = \sigma'(z^{[\ell]}) \circ (W^{[\ell+1]})^T \delta^{\ell+1} \tag{5}$$

$$\frac{\delta C}{\delta b_j} = \delta_j^\ell \tag{6}$$

$$\frac{\delta C}{\delta w_{jk}^{[\ell]}} = \delta_j^\ell a_k^{[\ell-1]} \tag{7}$$

Portland State UNIVERSITY

Neural Network Training Pseudocode

For counter $= 1$ upto Niter
    Choose an integer k uniformly at random from $\{1, 2, 3, \ldots, N\}$
    $x^{\{k\}}$ is current training data point
    $a^{[1]} = x^{\{k\}}$
    For $l = 2$ upto L
        $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$
        $a^{[l]} = \sigma\left(z^{[l]}\right)$
        $D^{[l]} = \operatorname{diag}\left(\sigma'(z^{[l]})\right)$
    end
    $\delta^{[L]} = D^{[L]}\left(a^{[L]} - y(x^{\{k\}})\right)$
    For $l = L - 1$ downto 2
        $\delta^{[l]} = D^{[l]}(W^{[l+1]})^T \delta^{[l+1]}$
    end
    For $l = L$ downto 2
        $W^{[l]} \to W^{[l]} - \eta\, \delta^{[l]} {a^{[l-1]}}^T$
        $b^{[l]} \to b^{[l]} - \eta\, \delta^{[l]}$
    end
end

Figure: Pseudocode for Stochastic Gradient Descent (Higham, 2019, p. 873)

Python Implementation – Forward Pass

```
# Simple stochastic gradient
# Randomly selected feature variable data point
x_sample = X[i:i+1]
# Corresponding labeled target variable data point
y_sample = y[i:i+1]

# Forward pass
Z1 = np.dot(x_sample, W1) + b1
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2) + b2
A2 = sigmoid(Z2)

Z3 = np.dot(A2, W3) + b3
A3 = sigmoid(Z3)
```

Python Implementation – Compute MSE Loss

```
loss_history = []

# Compute loss
loss = mse(y_sample, A3)
total_loss += loss
```

Python Implementation – Backpropagation

```
dz3 = mse_derivative(y_sample, A3) * sigmoid_derivative(Z3)
dW3 = np.dot(A2.T, dz3)
db3 = dz3

dz2 = np.dot(dz3, W3.T) * sigmoid_derivative(Z2)
dW2 = np.dot(A1.T, dz2)
db2 = dz2

dz1 = np.dot(dz2, W2.T) * sigmoid_derivative(Z1)
dW1 = np.dot(x_sample.T, dz1)
db1 = dz1
```

Portland State UNIVERSITY

Python Implementation – Parameter Update

```
W3 -= learning_rate * dW3
b3 -= learning_rate * db3

W2 -= learning_rate * dW2
b2 -= learning_rate * db2

W1 -= learning_rate * dW1
b1 -= learning_rate * db1
```

Portland State UNIVERSITY

NN Training Visualization

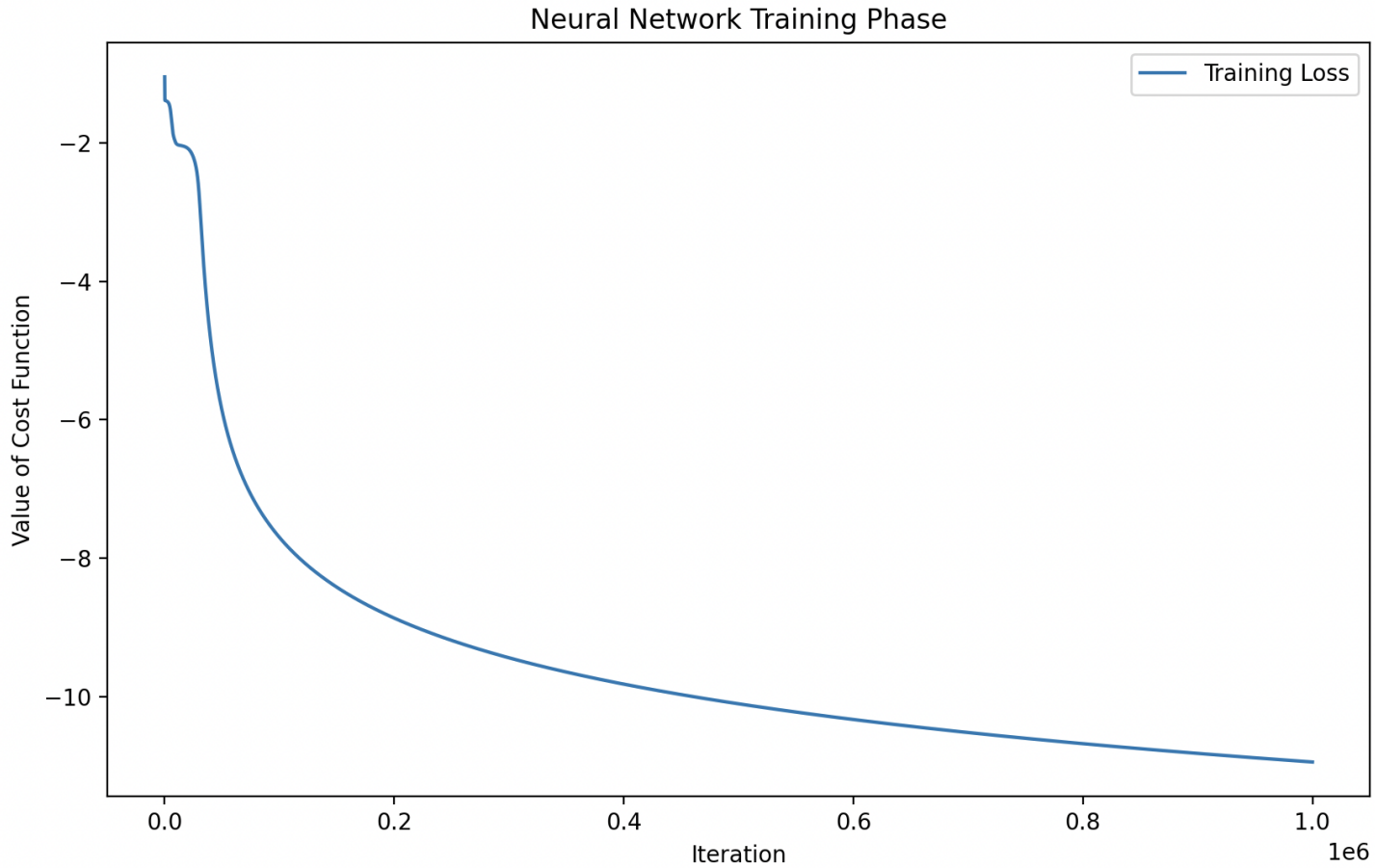


Figure: Successful Replication of Higham (2019), Reduction in Cost Function over 1 Million Iterations

Portland State UNIVERSITY

Support Vector Machine (SVM) for Decision Boundary

- We want to create a decision boundary/surface that visualizes the binary classification for our labeled dataset, as in Figure 6.2 from Higham (2019)
  - How do we achieve this?

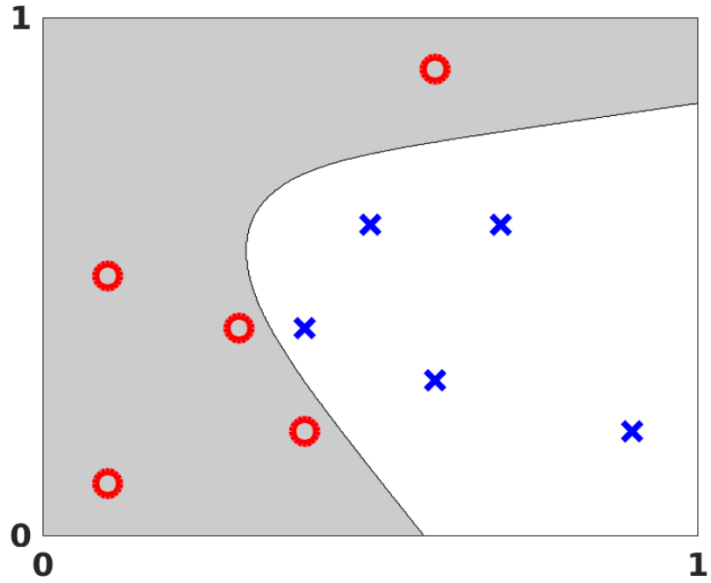


Fig. 6.2 *Visualization of output from an artificial neural network applied to the data in Figure* 2.1. *Here we trained the network using the stochastic gradient method with back propagation—behavior of the cost function is shown in Figure* 6.1. *The same optimization problem was solved with the* `lsqnonlin` *routine from MATLAB in order to produce Figure* 2.4.

SVM for Decision Boundary

- The algorithm I chose to construct the decision boundary/surface for our dataset is a *support vector machine* (SVM)
  - Algorithm for supervised classification
- The objective of SVM is to construct a *hyperplane* that maximizes the distance between the data points from the respective classes (max-margin)
- SVM's achieve this via a *kernel function* k that maps the dataset from original space to higher-dimensional feature space (like change of variables from calculus!)
- I chose polynomial kernel $k(x_i, x_j) = (x_i \cdot x_j + 1)^d$, where d is degree of polynomial (used sklearn)

Portland State UNIVERSITY

Replicated Decision Boundary via SVM

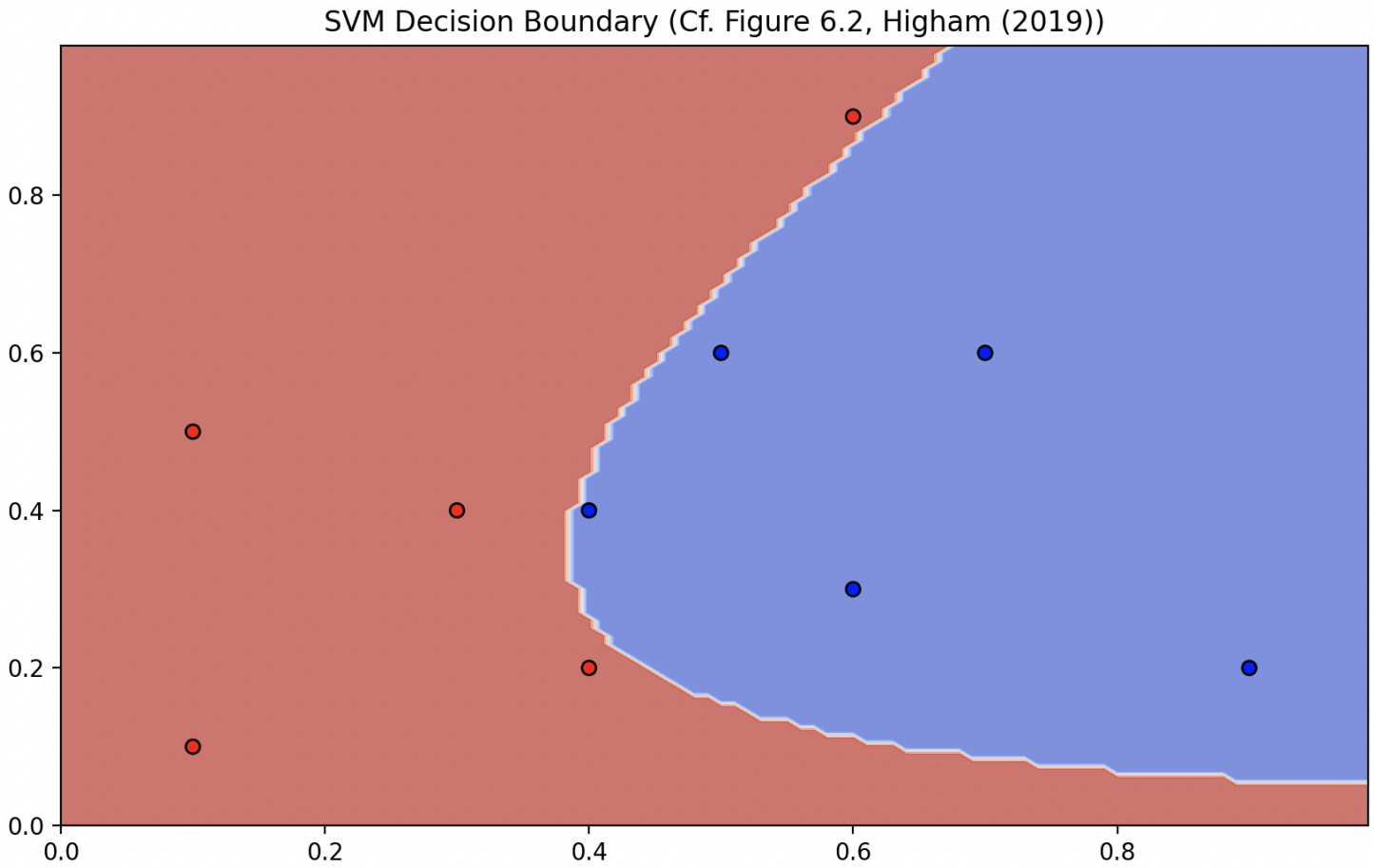


Figure: Decision Boundary for Higham (2019) Classifier via SVM

Short Primer on Convolutional NN's (CNN's)

- One major research/commercial interest for deep learning is apply neural networks for visual data (e.g., autonomous vehicles)
- Recall that number of nodes in input layer of NN corresponds to number of input variables $x_1, x_2, \ldots$
- For a 200×200 RGB image, we would then have to build an NN with $200 \times 200 \times 3 = 120,000$ weights into first hidden layer
  - This is not computationally feasible!
  - Traditional NN's won't scale well for image data
- This is why *convolutional neural networks* (CNN's) are used for visual data

Short Primer on CNN's

- CNN's work by multiplying input matrices by a kernel, filter, or *convolution matrix* that drastically reduce the number of non-zero entries from the original input matrix

$$\begin{bmatrix} 1 & -1 & & & & \\ & 1 & -1 & & & \\ & & 1 & -1 & & \\ & & & 1 & -1 & \\ & & & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{5\times 6}$$

Figure: Example kernel/filter for a CNN

Portland State UNIVERSITY

Example Convolution Matrices

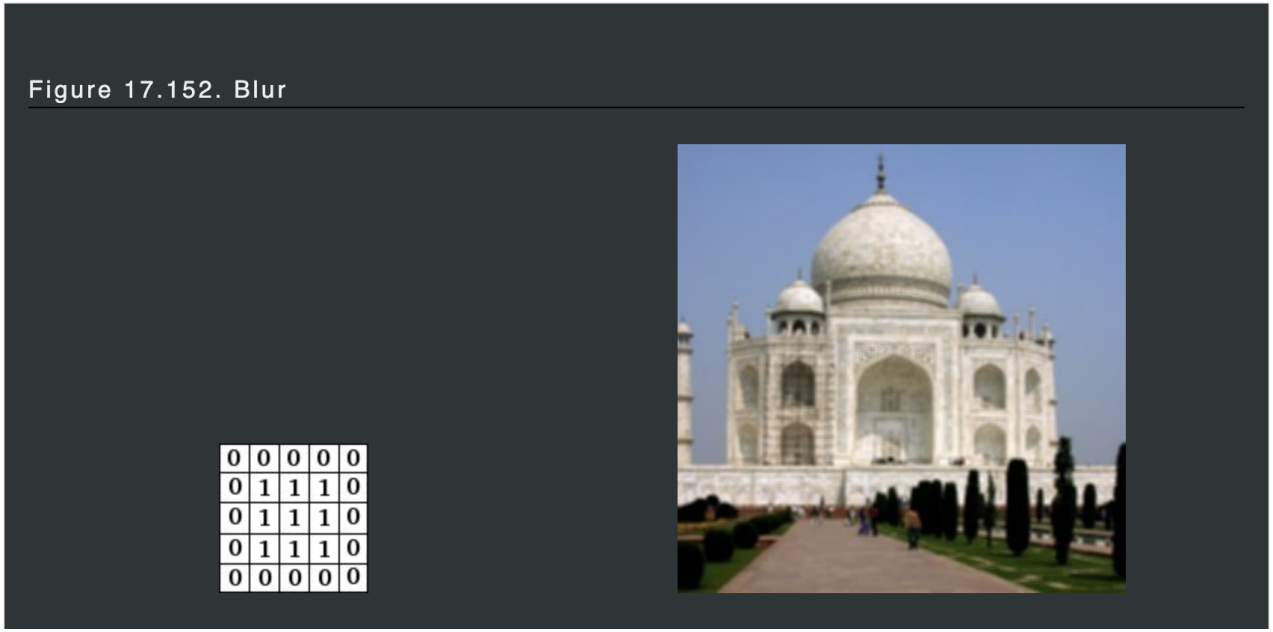


Figure: Kernel Matrix for Blurring Input Image

Portland State UNIVERSITY

Example Convolution Matrices

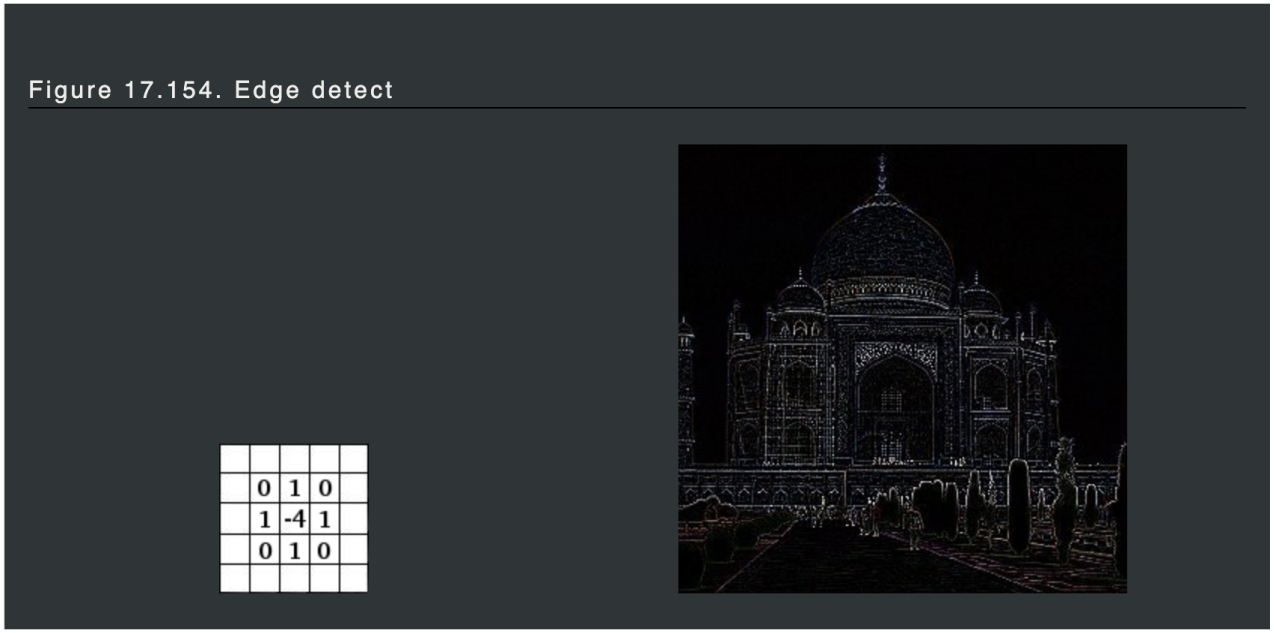
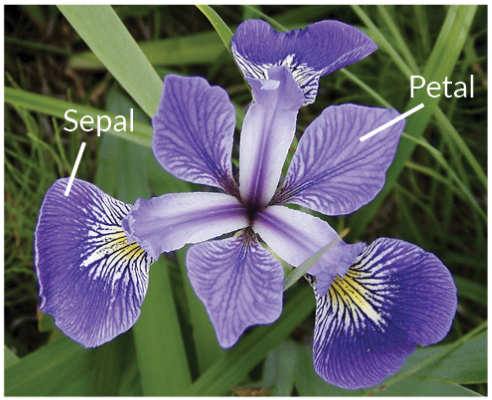


Figure: Kernel Matrix for Edge Detection in Input Image

Portland State UNIVERSITY

Multi-Class Classification – Iris Dataset

- This is another classic dataset in ML/data science, compiled by Ronald Fisher in 1936



Iris Versicolor

Iris Setosa

Iris Virginica

- Feature variables: sepal length, sepal width, petal length, petal width
- Target variable: sub-species of Iris (3, OHE)

Portland State UNIVERSITY

NN Training for Iris Dataset

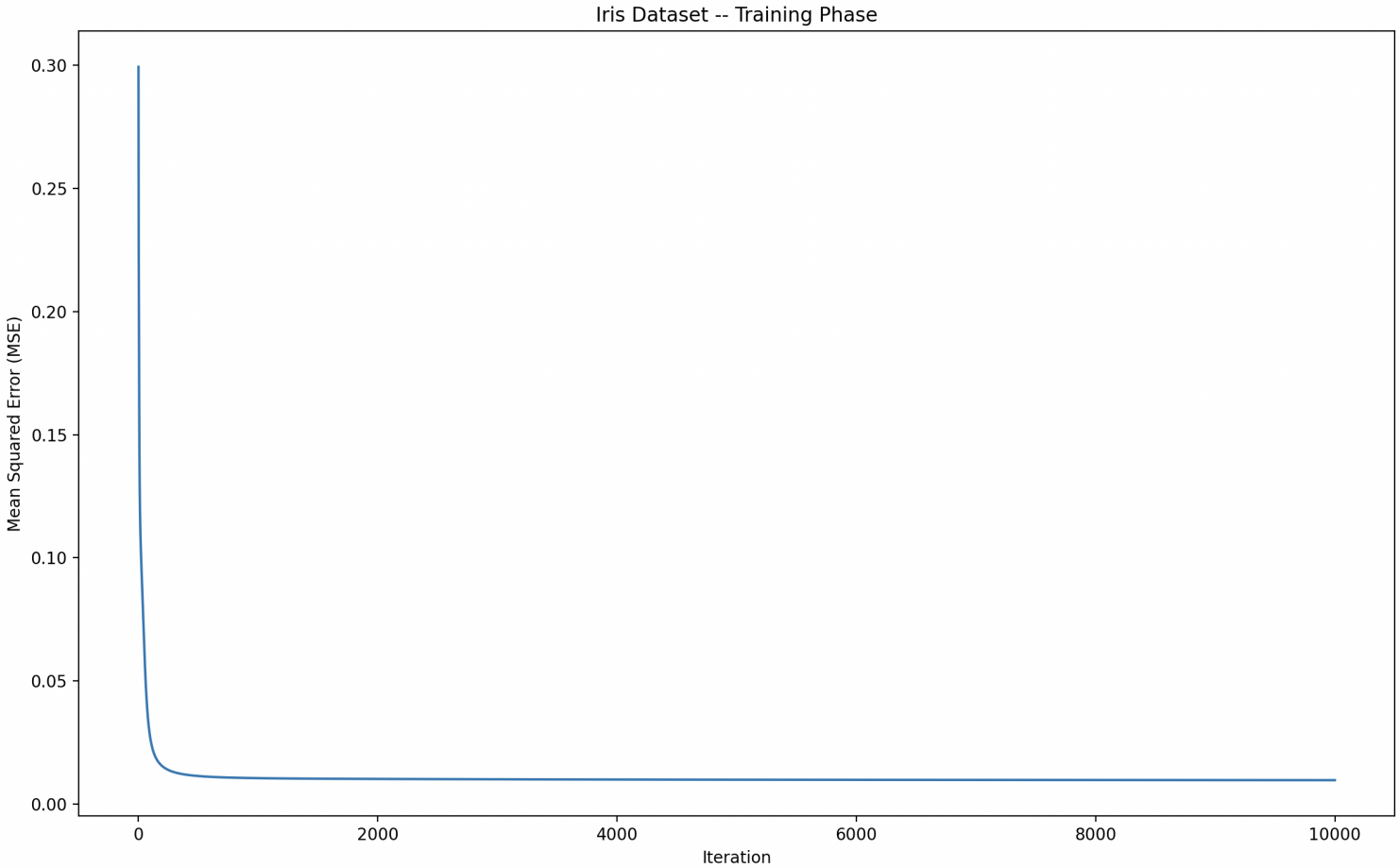


Figure: 3 Layer NN (4 nodes in input, 3 nodes in hidden, 3 nodes in output) trained on Iris Dataset for 10,000 iterations

Portland State UNIVERSITY

Concluding Remarks

- Active Learning
  - How would NN performance change if data point(s) added or removed?
  - How to evaluate relative value of a data point/sample?
  - How can NN's be pre-trained to make such evaluations?
- Different NN Architectures
  - Generative Adversarial Networks (GAN's)
  - Vector embeddings for categorical data (NLP, used by current large language models (LLM's))

Portland State UNIVERSITY