DS 4400

Machine Learning and Data Mining I
Spring 2024

David Liu

Khoury College of Computer Science

Northeastern University

March 19 2024

Announcements

- HW 4 will be released on Thursday and will be due two weeks thereafter.
- Proposal proposal feedback has been provided on Gradescope.
- Expect midterm grades to be online by Wednesday evening and hard copies returned on Friday in class. On Friday, we will review the solutions.
- Will release a “Kaggle” style extra credit (completely optional).

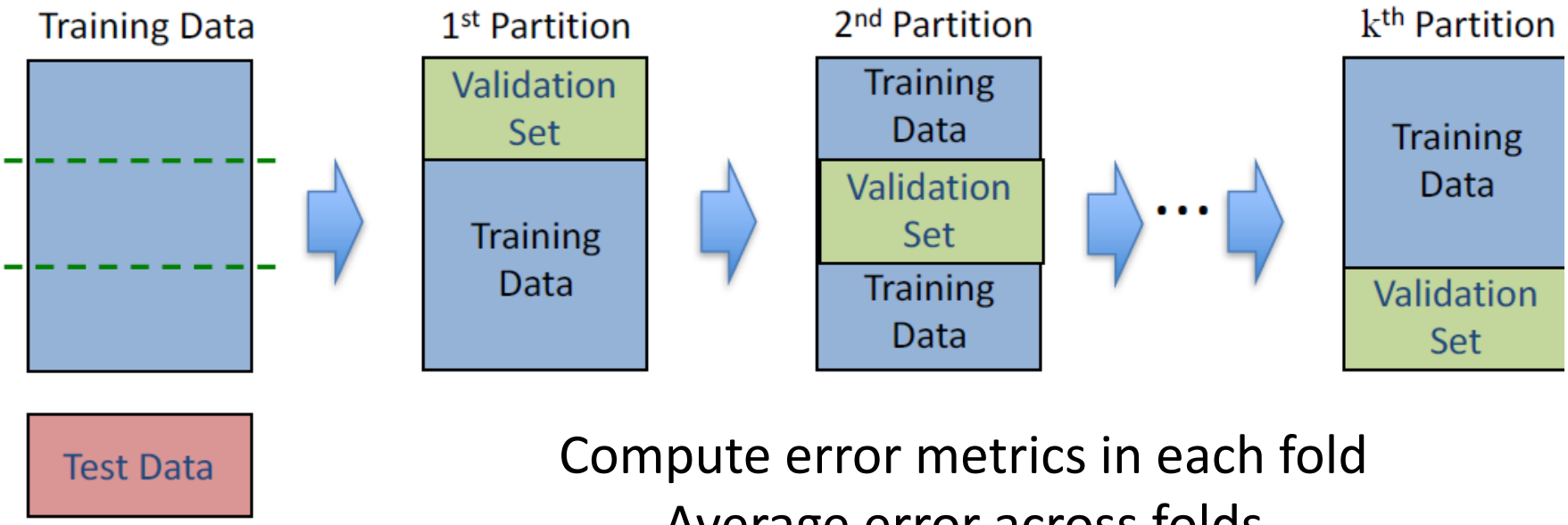
Project Topics

- Uber eats delivery time prediction
- Predicting NBA wins / all star selections (x2)
- Predicting movie box office earnings. (x3)
- Classifying cells based on mRNA data.
- Predicting the US Treasury yield curve
- Predicting NYC Airbnb prices
- Classifying international disasters
- Detecting facial expressions from images.
- Examining the role of race in dating.
- Predicting the stock market
- Detecting car lanes in images.
- Identifying AI generated text
- Predicting diabetes
- Music genre prediction (x2)
- Predicting Amazon review ratings
- Predicting bankruptcy
- Audio CAPCHAS
- Detecting smoking and drinking in patients.
- Recommending songs to go with a book.
- Classifying dog breeds from images
- Predicting song popularity
- Predicting pet breeds
- Predicting road accident severity

Words of caution for projects

- Think about what is a feature and what is a response
- Avoid having one person do the cleaning and another the modeling.
- Dimensionality reduction (PCA, t-SNE) may be relevant for exploration, visualization, and scalability.
- Class imbalance (up-sampling and down sampling)
- Subset the data when n >> d
- Keep in mind that you need ground truth data
- Compare with literature
- Caution with Spotify data

Cross Validation



Compute error metrics in each fold
Average error across folds

1. k-fold CV

- Split training data into k partitions (folds) of equal size
- Pick the optimal value of hyper-parameter according to error metric averaged over all folds

Homework 3 Questions

Outline

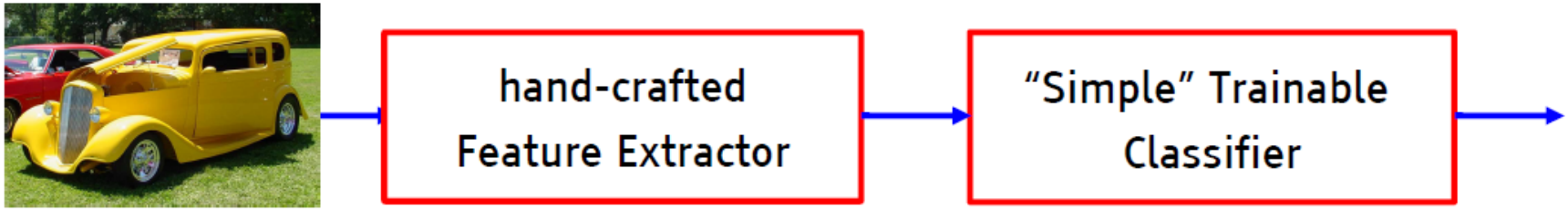
- Deep learning
  - Learning feature representations
  - Perceptron and limitations
- Feed-forward neural networks
  - Activations
  - Forward propagation
  - Vectorization
  - Softmax classifier

References

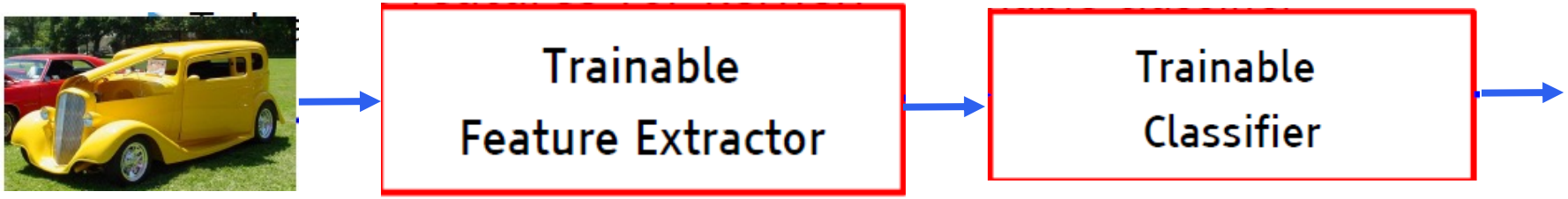
- Deep Learning books
  - https://d2l.ai/ (D2L)
  - https://www.deeplearningbook.org/ (advanced)
- Stanford notes on deep learning
  - http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf

Deep Learning

- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier

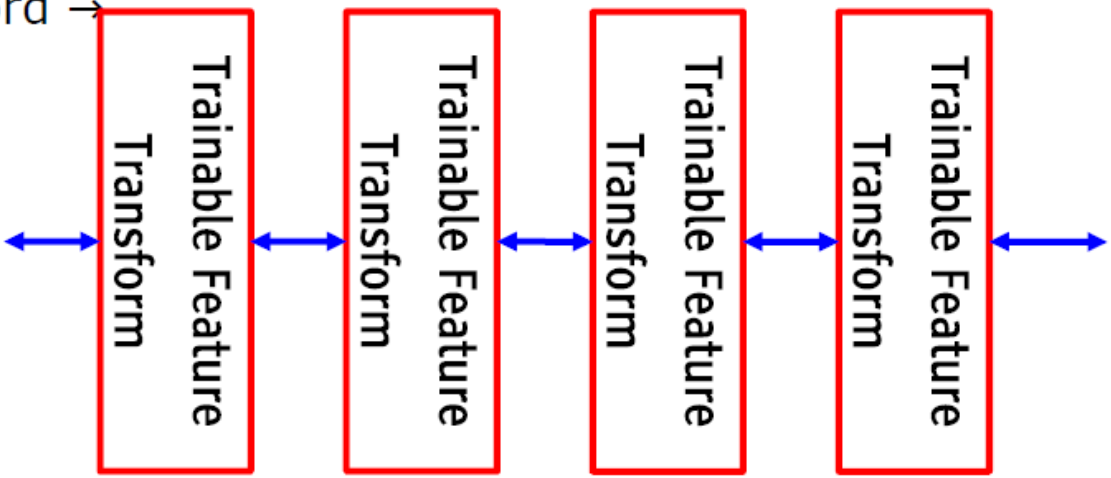


- End-to-end learning / Feature learning / Deep learning

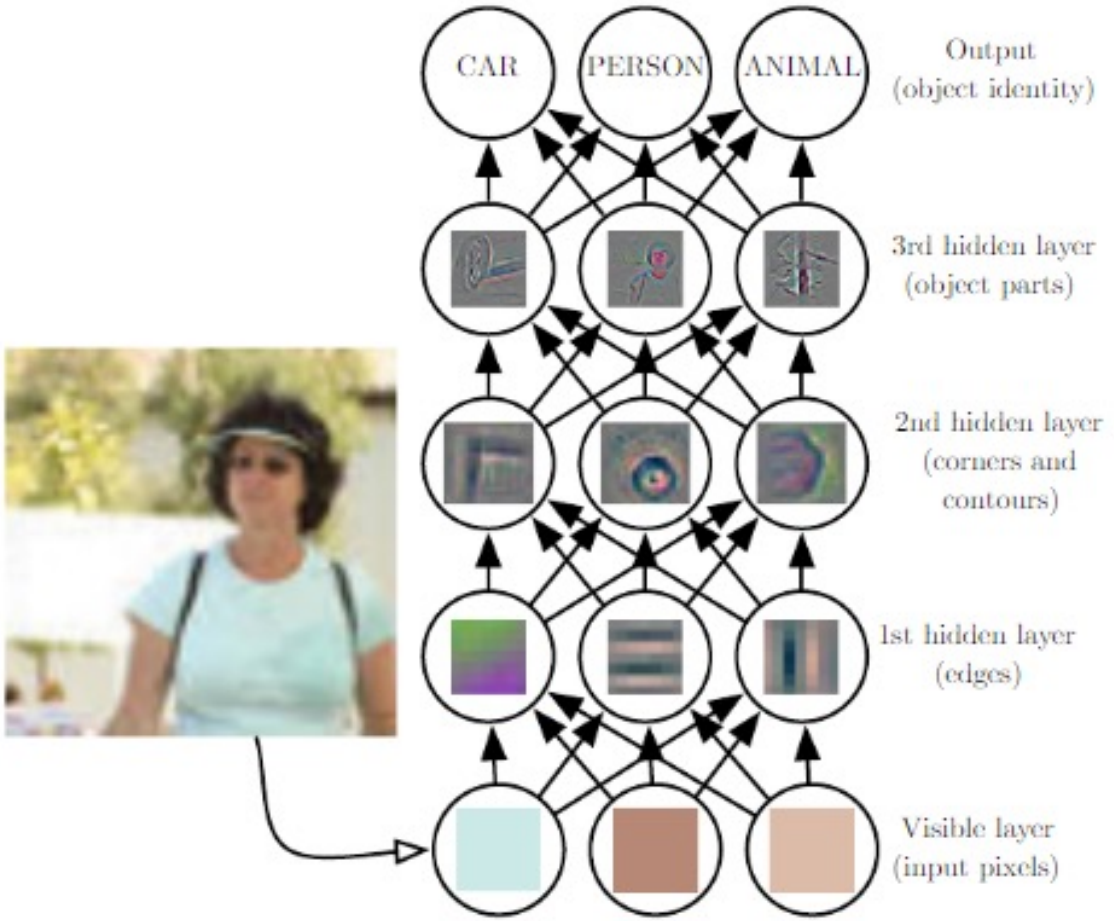


Trainable Feature Hierarchy

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → … → phone → phoneme → word →



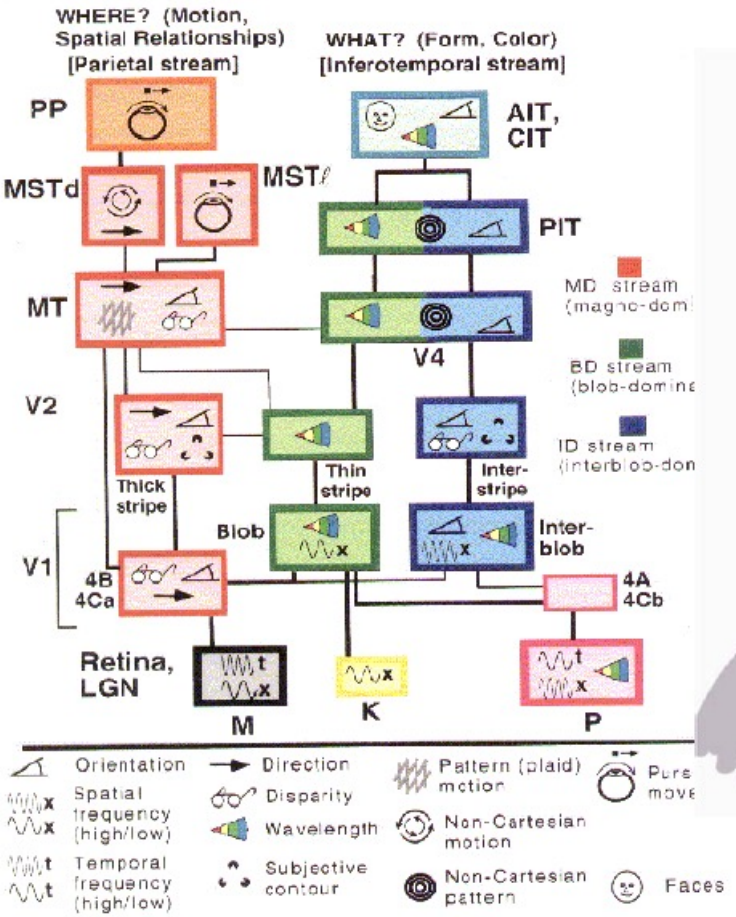
Learning Representations

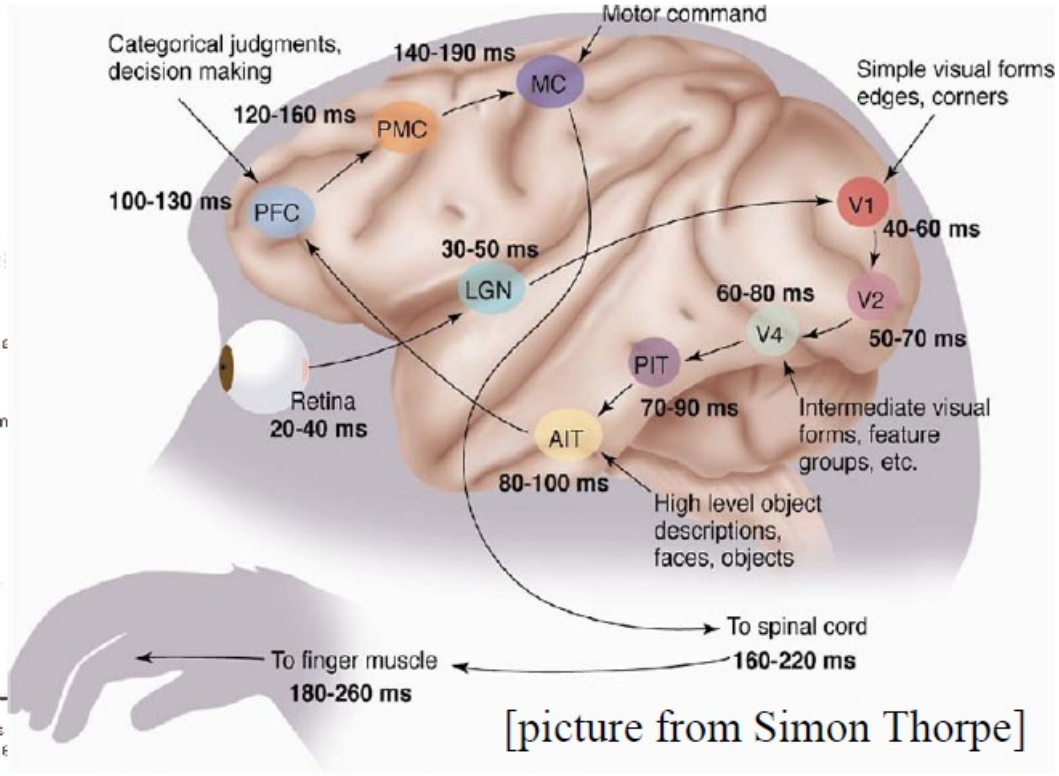


Deep Learning addresses the problem of learning hierarchical representations

The Visual Cortex is Hierarchical

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations





[Gallant & Van Essen]

Cat Visual Cortex

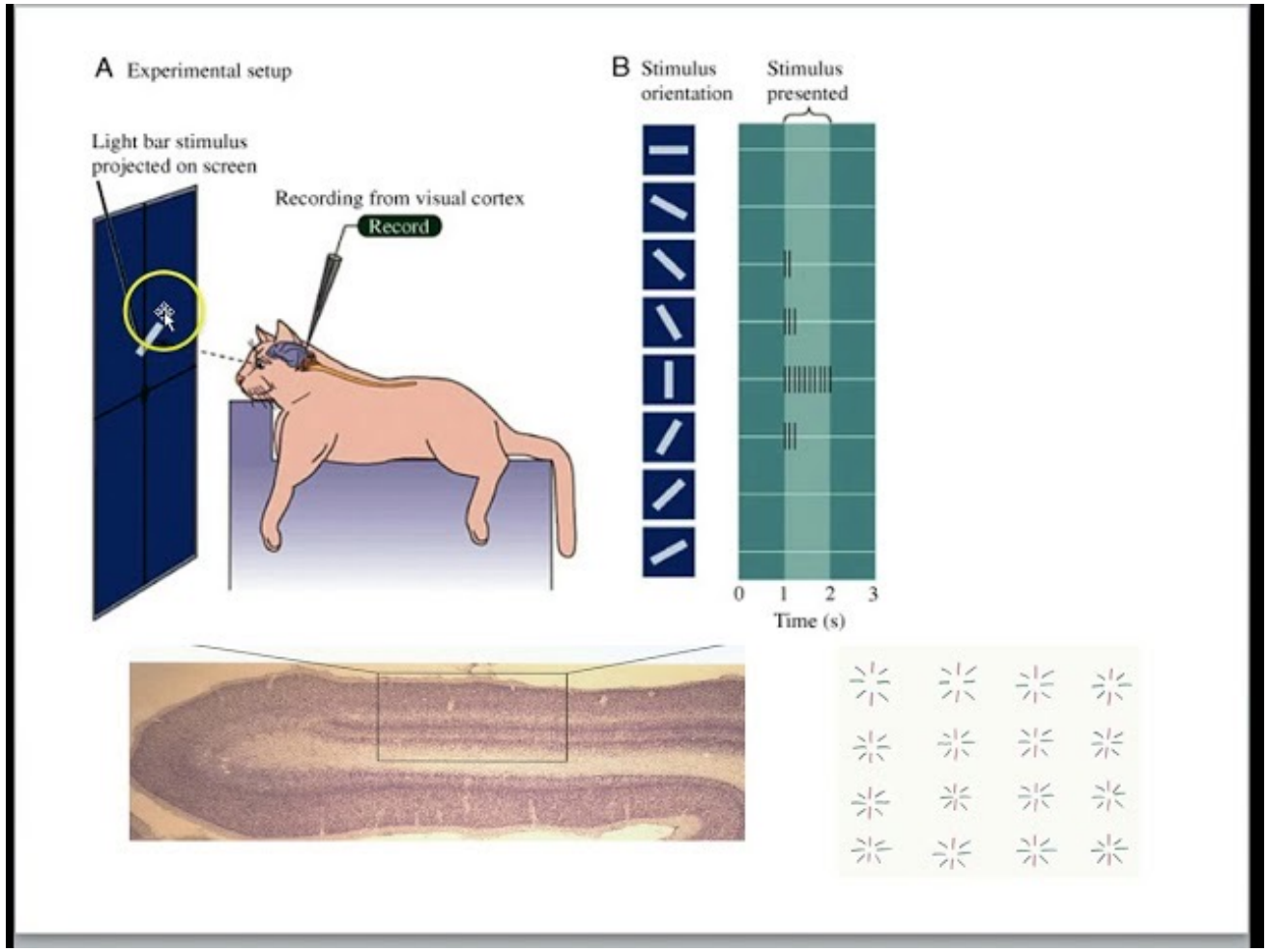
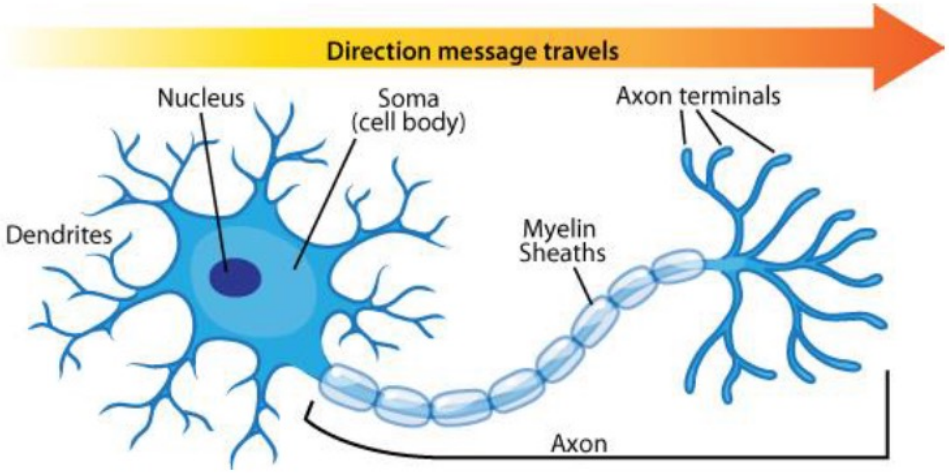


Image credit: https://www.informit.com/articles/article.aspx?p=1431818

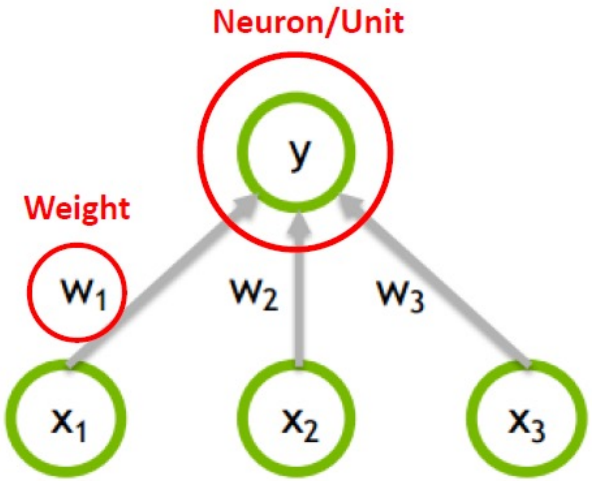
https://www.youtube.com/watch?v=IOHayh06LJ4&pp=ygURY2F0IHZpc3VhbCBjb3J0ZXg%3D

Analogy to Human Brain

Human Brain



Biological Neuron



$$y=F(w_1x_1+w_2x_2+w_3x_3)$$

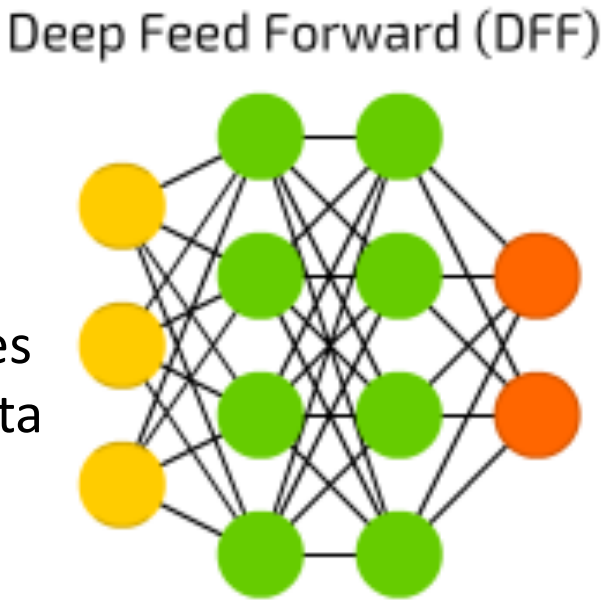
$$F(x)=\max(0,x)$$

Artificial Neuron

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

Neural Networks



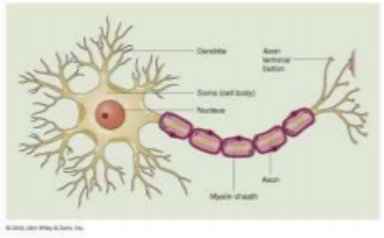
Input nodes
Training data

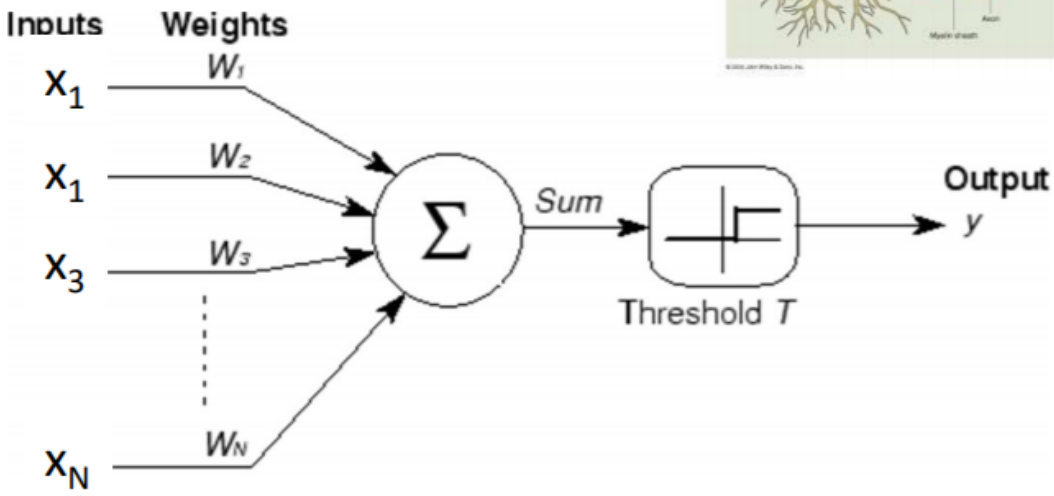
Output nodes

Hidden units
(Learned during training)

- Neural networks: made up from nodes connected by links
- Each link has a weight and activation function
- Feed-forward neural networks
  - Training data is input to left, prediction are output on right

Perceptron





$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

- A threshold unit
  - "Fires" if the weighted sum of inputs exceeds a threshold

The Perceptron

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- The perceptron uses the following update rule each time it receives a new training instance (x_i, y_i)

$$\theta_j \leftarrow \theta_j - \frac{1}{2}(h_\theta(x_i) - y_i)x_{ij}$$

  - If the prediction matches the label, make no change
  - Otherwise, adjust $\boldsymbol{\theta}$

The Perceptron

- The perceptron uses the following update rule each time it receives a new training instance (x_i, y_i)

$$\theta_j \leftarrow \theta_j - \frac{1}{2}\left(h_\theta(x_i) - y_i\right)x_{ij}$$

The Perceptron

- The perceptron uses the following update rule each time it receives a new training instance (x_i, y_i)

$$\theta_j \leftarrow \theta_j - \frac{1}{2}\,(h_\theta(x_i) - y_i)x_{ij}$$

either 2 or -2

- Re-write as $\theta_j \leftarrow \theta_j + y_i x_{ij}$ (only upon misclassification)

Perceptron Rule: If x_i is misclassified, do

$$\theta \leftarrow \theta + y_i\, x_i$$

Online Perceptron

Let $\theta \leftarrow$[0,0,…,0]
Repeat:
    Receive training example (x_i, y_i)
    If $y_i \theta^T x_i \leq 0$ // prediction is incorrect
        $\theta \leftarrow \theta + y_i \, x_i$
Until stopping condition

Online learning – the learning mode where the model update is performed each time a single observation is received

Batch learning – the learning mode where the model update is performed after observing the entire training set

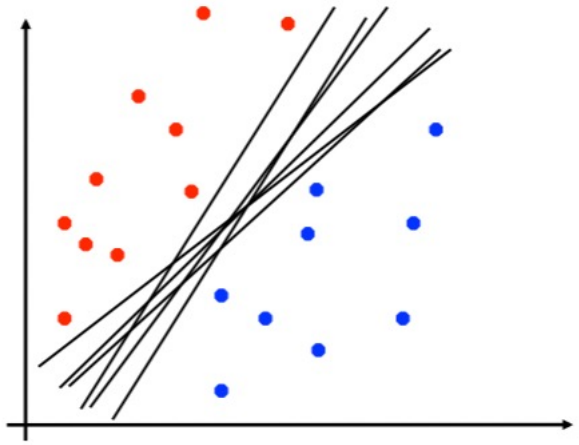
Batch Perceptron

Let $\theta \leftarrow [0,0,\ldots,0]$
Repeat:
$\Delta = [0,0,\ldots,0]$
For $i = 1$ to N // Consider all training examples
If $y_i \theta^T x_i \leq 0$ // Prediction is incorrect
$\Delta \leftarrow \Delta + y_i \, x_i.$ // Accumulate all errors
$\theta \leftarrow \theta + \frac{\Delta}{N}$ // Parameter update rule
Until stopping condition

- Guaranteed to find separating hyperplane if data is linearly separable

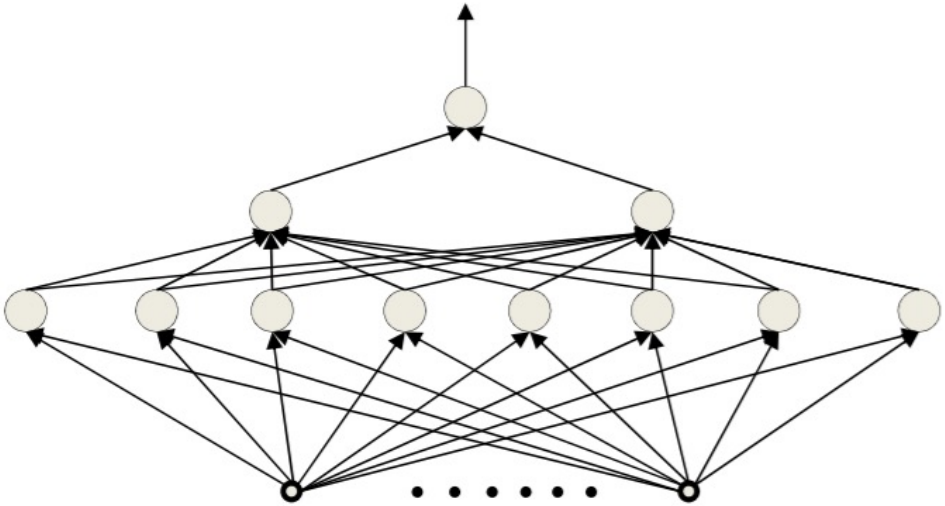
Perceptron Limitations

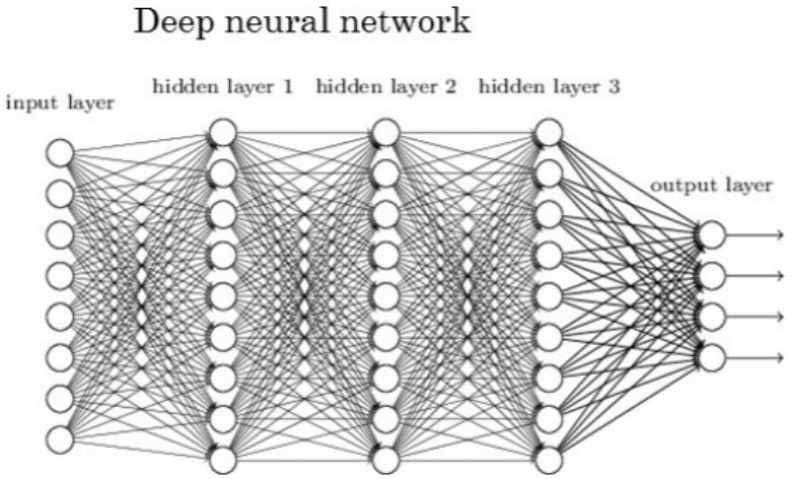
- Is dependent on starting point
- It could take many steps for convergence
- Perceptron can overfit
  - Move the decision boundary for every example
- It is a linear model



Which of this is optimal?

Multi-Layer Perceptron



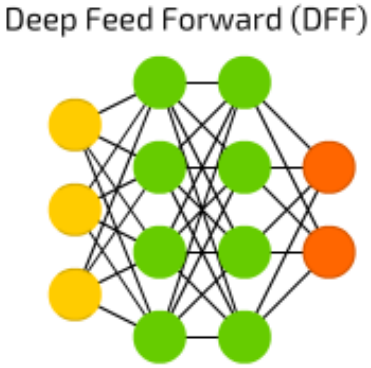


- A network of perceptrons
  - Generally “layered”

Neural Network Architectures

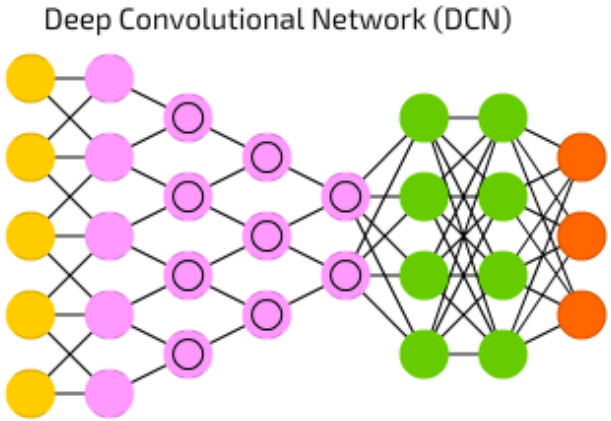
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



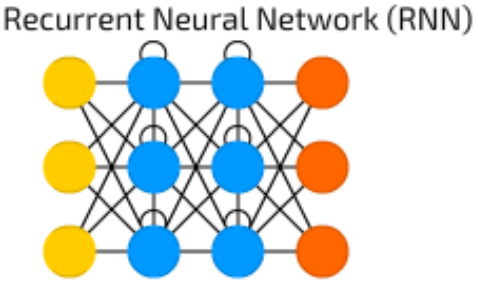
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

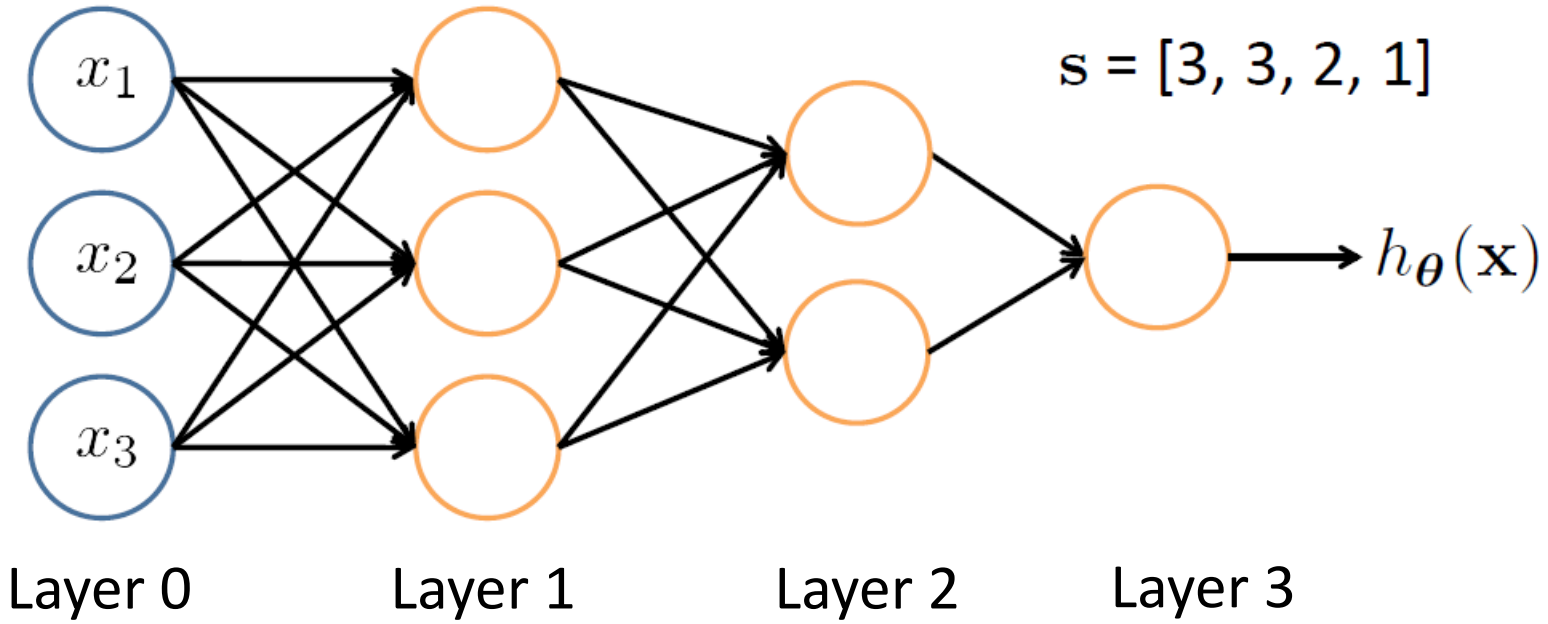


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



Feed-Forward Networks



L denotes the number of layers

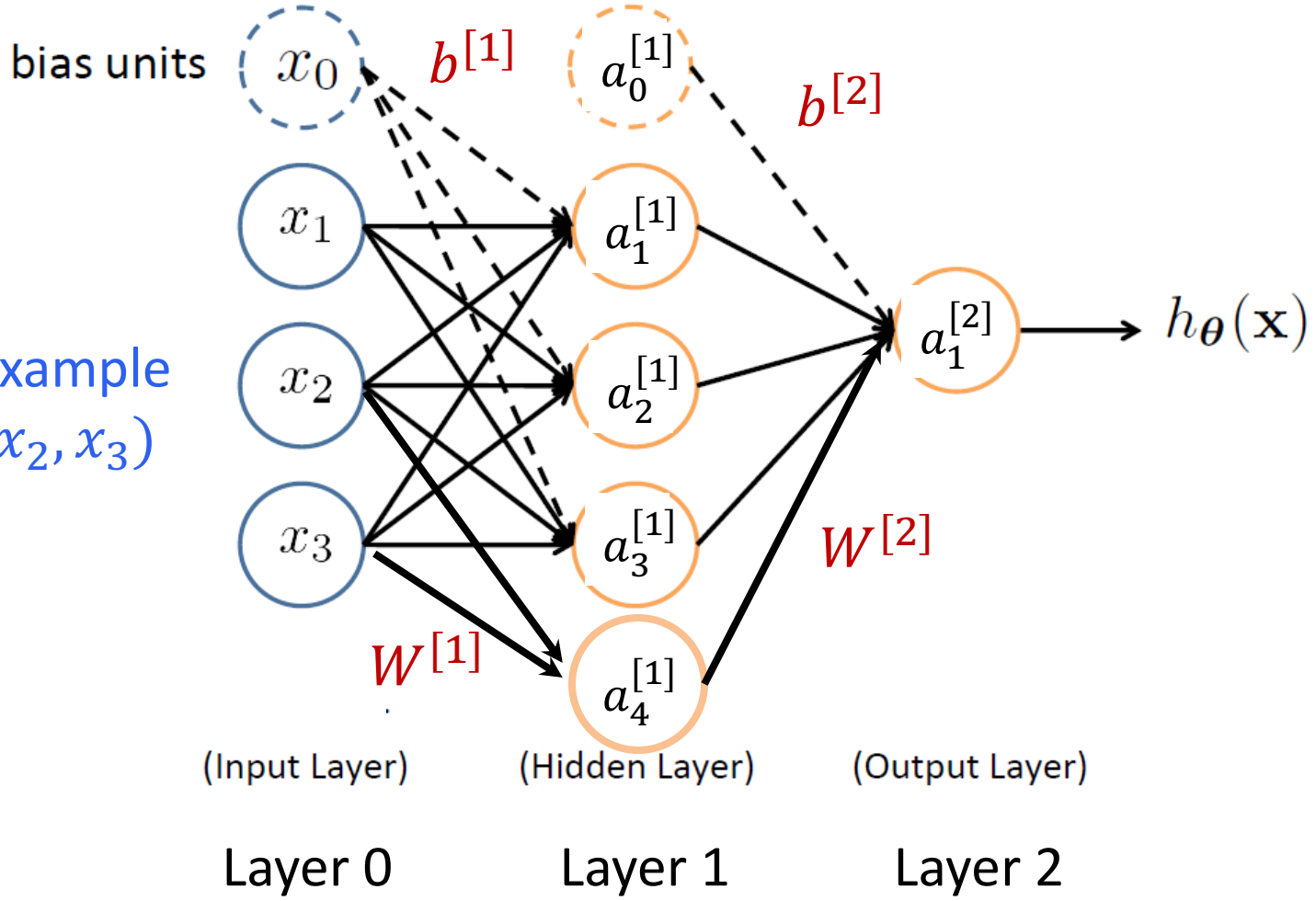
$\mathbf{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1}=K$ (# classes)

Feed-Forward NN

- Hyper-parameters
  - Number of layers
  - Architecture (how layers are connected)
  - Number of hidden units per layer
  - Number of units in output layer
  - Activation functions
- Other
  - Initialization
  - Regularization

Feed-Forward Neural Network



Training example
$x = (x_1, x_2, x_3)$

No cycles

$\theta = (b^{[1]}, W^{[1]}, b^{[2]}, W^{[2]})$

Vectorization

$$z_1^{[1]} = W_1^{[1]}\ x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]})$$
$$\vdots \qquad\qquad\qquad \vdots \qquad\qquad \vdots$$
$$z_4^{[1]} = W_4^{[1]}\ x + b_4^{[1]} \quad \text{and} \quad a_4^{[1]} = g(z_4^{[1]})$$

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — & W_1^{[1]} & — \\ — & W_2^{[1]} & — \\ & \vdots & \\ — & W_4^{[1]} & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Linear

$$a^{[1]} = g(z^{[1]})$$

Non-Linear

Vectorization

Output layer

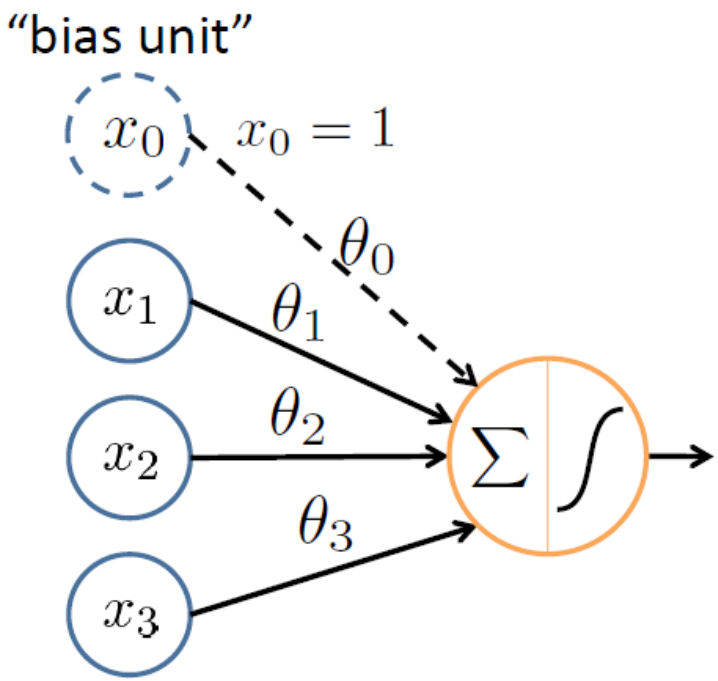
$$z_1^{[2]} = W_1^{[2]T} a^{[1]} + b_1^{[2]} \quad \text{and} \quad a_1^{[2]} = g(z_1^{[2]})$$

$$\underbrace{z^{[2]}}_{1\times1} = \underbrace{W^{[2]}}_{1\times4}\underbrace{a^{[1]}}_{4\times1} + \underbrace{b^{[2]}}_{1\times1} \quad \text{and} \quad \underbrace{a^{[2]}}_{1\times1} = g(\underbrace{z^{[2]}}_{1\times1})$$

Hidden Units

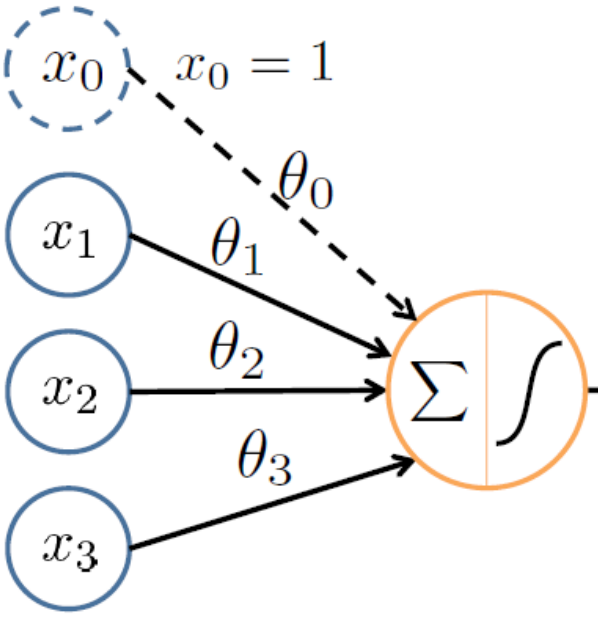
- Layer 1
  - First hidden unit:
    - Linear: $z_1^{[1]} = \mathrm{W}_1^{[1]\,\mathrm{T}} x + \mathrm{b}_1^{[1]}$
    - Non-linear: $a_1^{[1]} = \mathrm{g}(\mathrm{z}_1^{[1]})$
  - …
  - Fourth hidden unit:
    - Linear: $z_4^{[1]} = \mathrm{W}_4^{[1]\,\mathrm{T}} x + \mathrm{b}_4^{[1]}$
    - Non-linear: $a_4^{[1]} = \mathrm{g}(\mathrm{z}_4^{[1]})$
- Terminology
  - $a_i^{[j]}$ - Activation of unit i in layer j
  - g - Activation function
  - $W^{[j]}$ - Weight matrix controlling mapping from layer j-1 to j
  - $b^{[j]}$ - Bias vector from layer j-1 to j

Logistic Unit: A simple NN



$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Logistic Unit: A simple NN



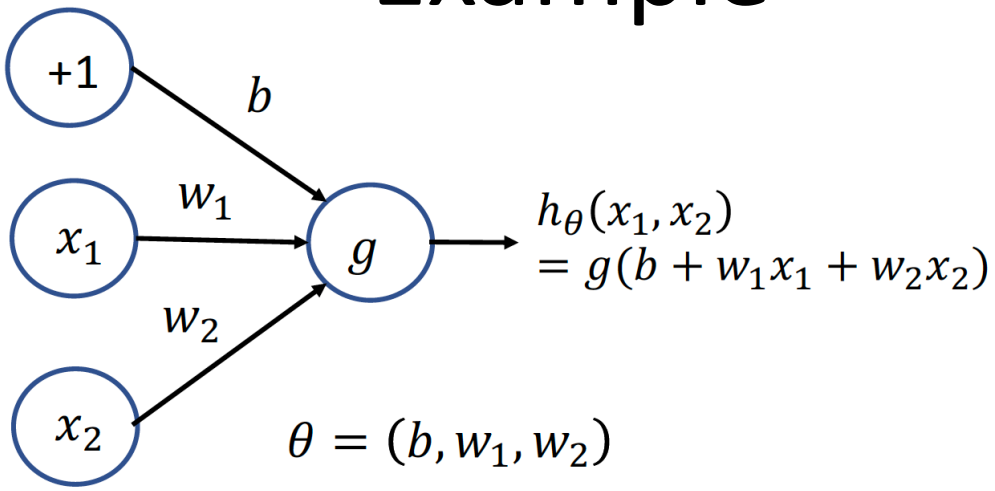
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = g(w^T x + b)$$
$$= \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

No hidden layers

Example



1. Given $b = -10, w_1 = 12, w_2 = 5$
   Activation $g(z) = sign(z)$
Compute the output:

x_1	x_2	$h(x_1, x_2)$
0	0	
0	1	
1	0	
1	1	

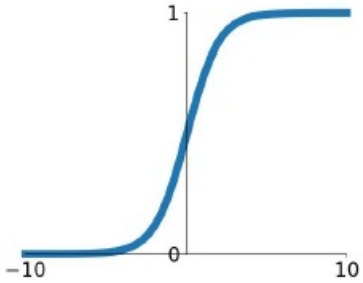
2. Find out the weights b, w_1, w_2 and activation function to get the following output:

x_1	x_2	$h(x_1, x_2)$
0	0	1
0	1	1
1	0	1
1	1	0

Non-Linear Activation Functions

Sigmoid

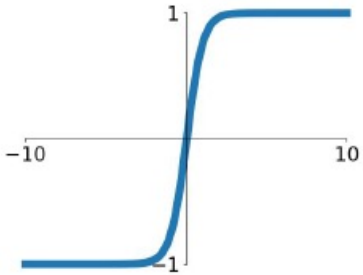
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Binary Classification

tanh

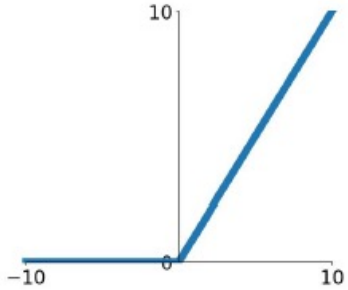
$$\tanh(x)$$



Regression

ReLU

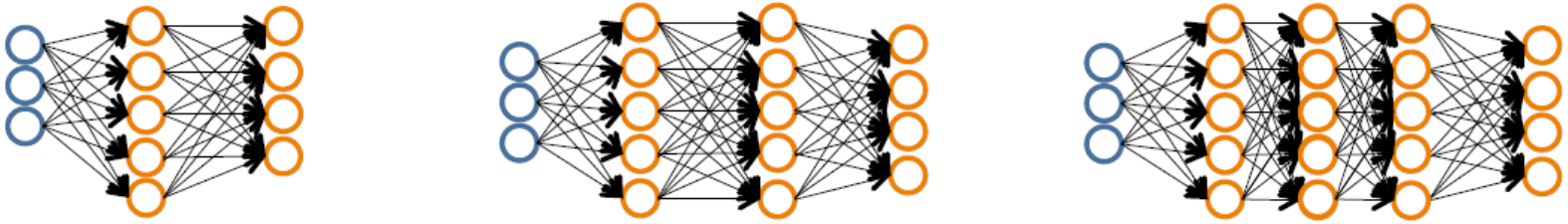
$$\max(0, x)$$



Intermediary layers

How to pick architecture?

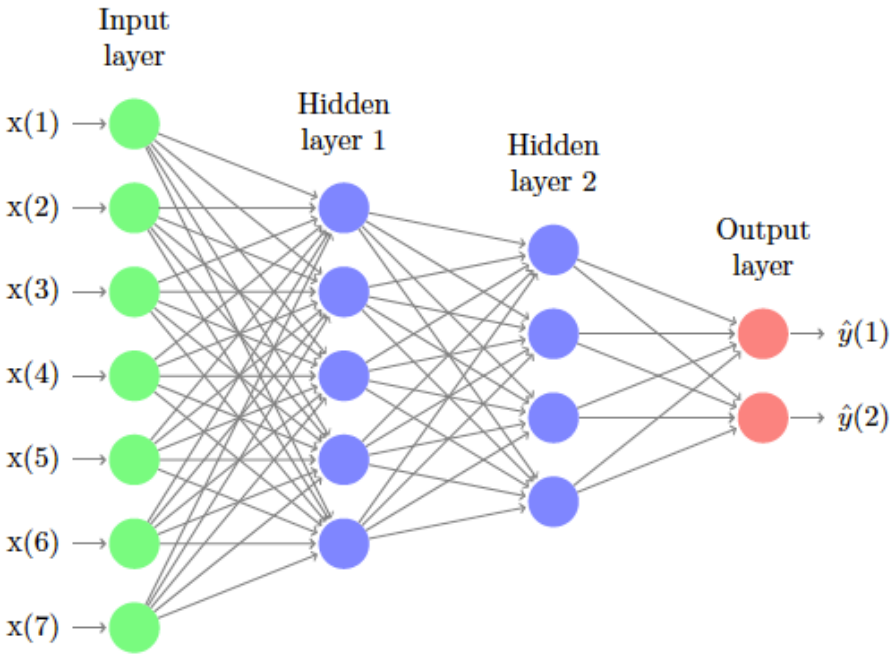
Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

FFNN Architectures



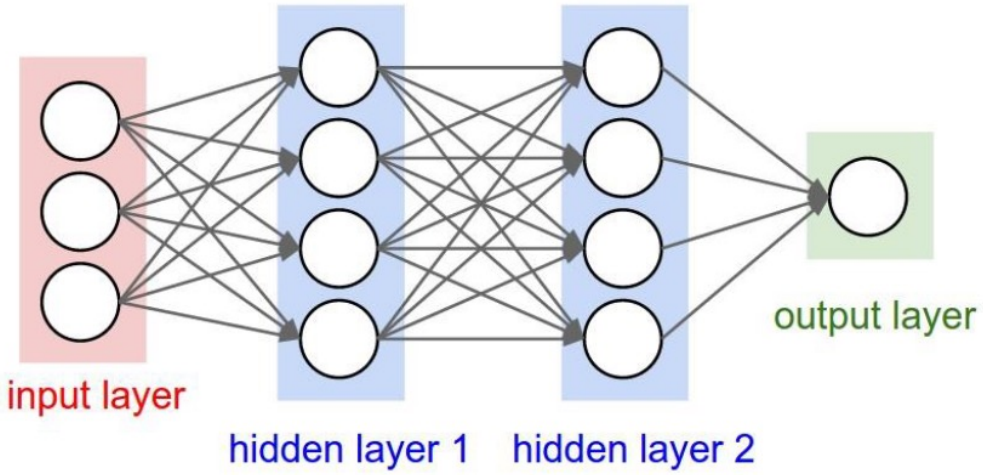
- Input and Output Layers are completely specified by the problem domain
- In the Hidden Layers, number of neurons in Layer i+1 is usually smaller or equal to the number of neurons in Layer i

Training Neural Networks

- Input training dataset D
  - Number of features: d
  - Labels from K classes
- First layer has d+1 units (one per feature and bias)
- Output layer has K units
- Training procedure determines parameters that optimize loss function
  - Backpropagation
  - Learn optimal $W^{[i]}, b^{[i]}$ at layer *i*
- Evaluation of a point done by forward propagation

Forward Propagation

- The input neurons first receive the data features of the object. After processing the data, they send their output to the first hidden layer.
- The hidden layer processes this output and sends the results to the next hidden layer.
- This continues until the data reaches the final output layer, where the output value determines the object's classification.
- This entire process is known as Forward Propagation, or Forward prop.



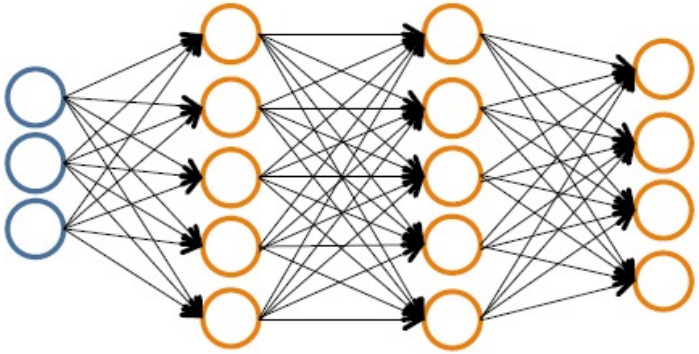
Multi-Class Classsification

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

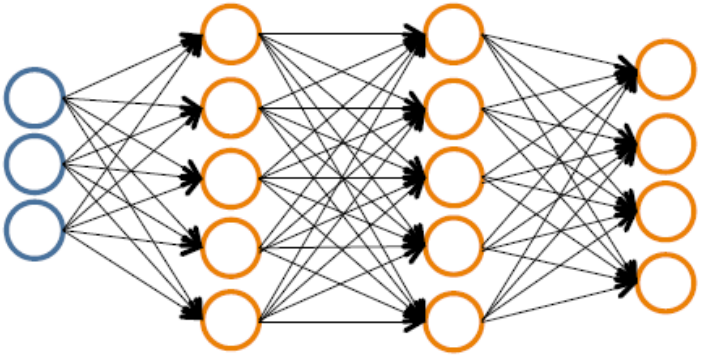
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Sigmoid

Multi-class classification (K classes)

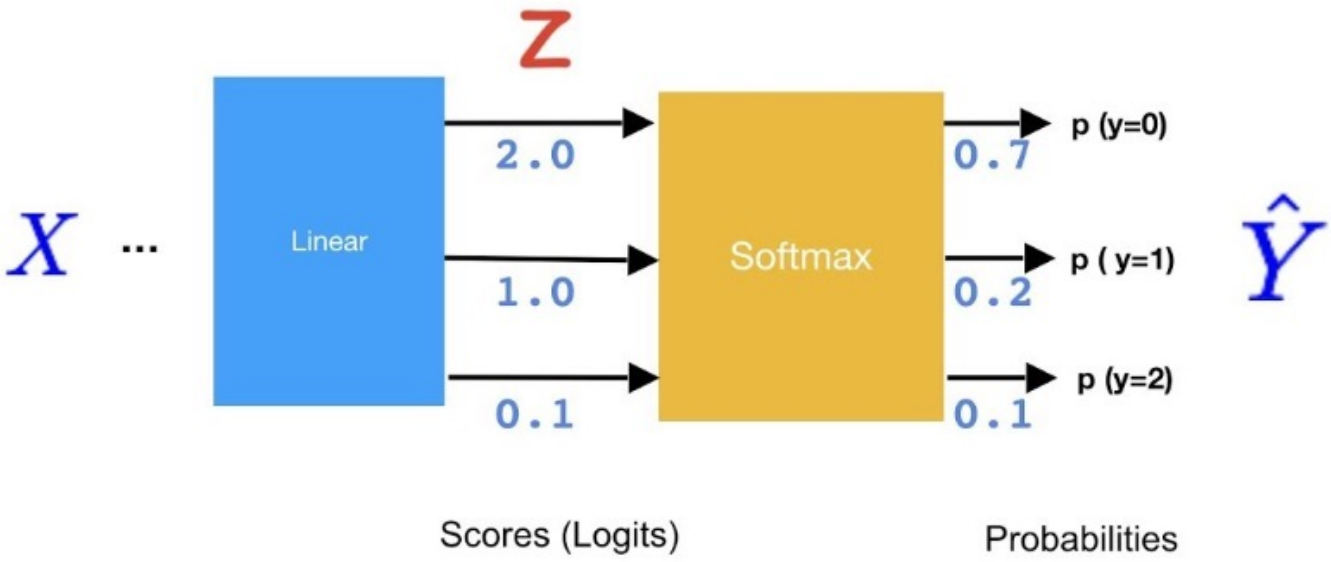
$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

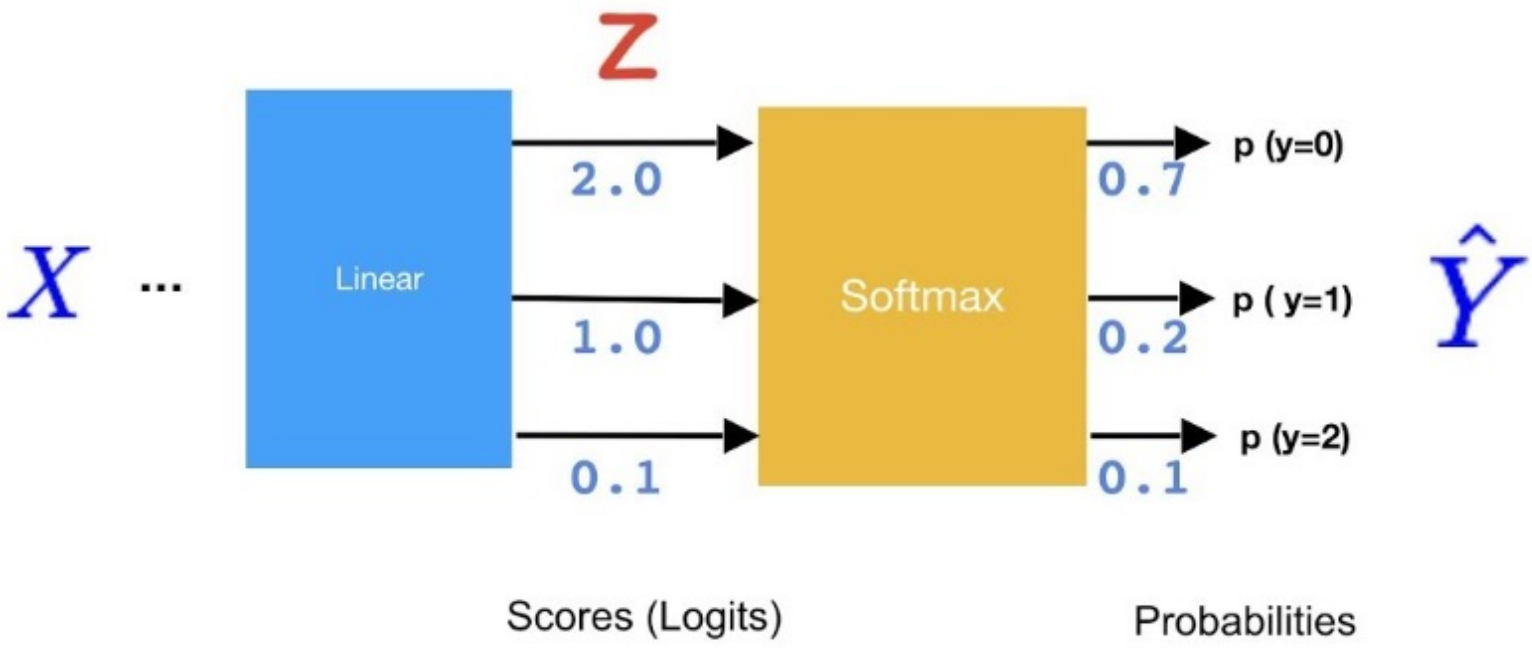
K output units $(s_{L-1} = K)$

Softmax

Softmax classifier



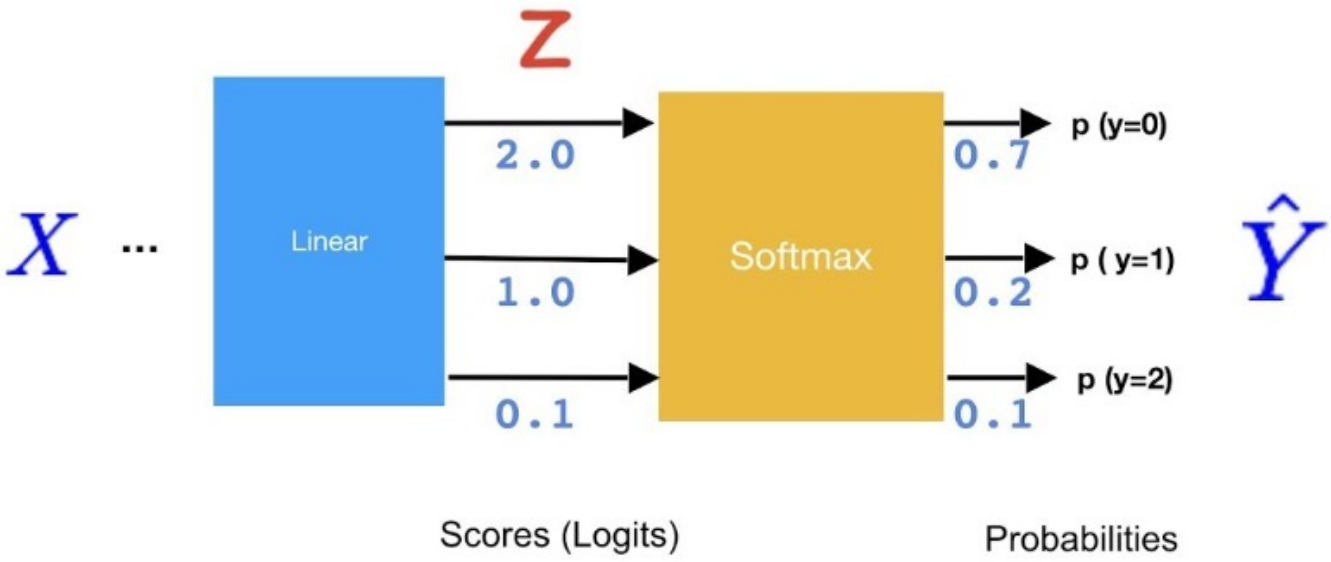
Softmax classifier



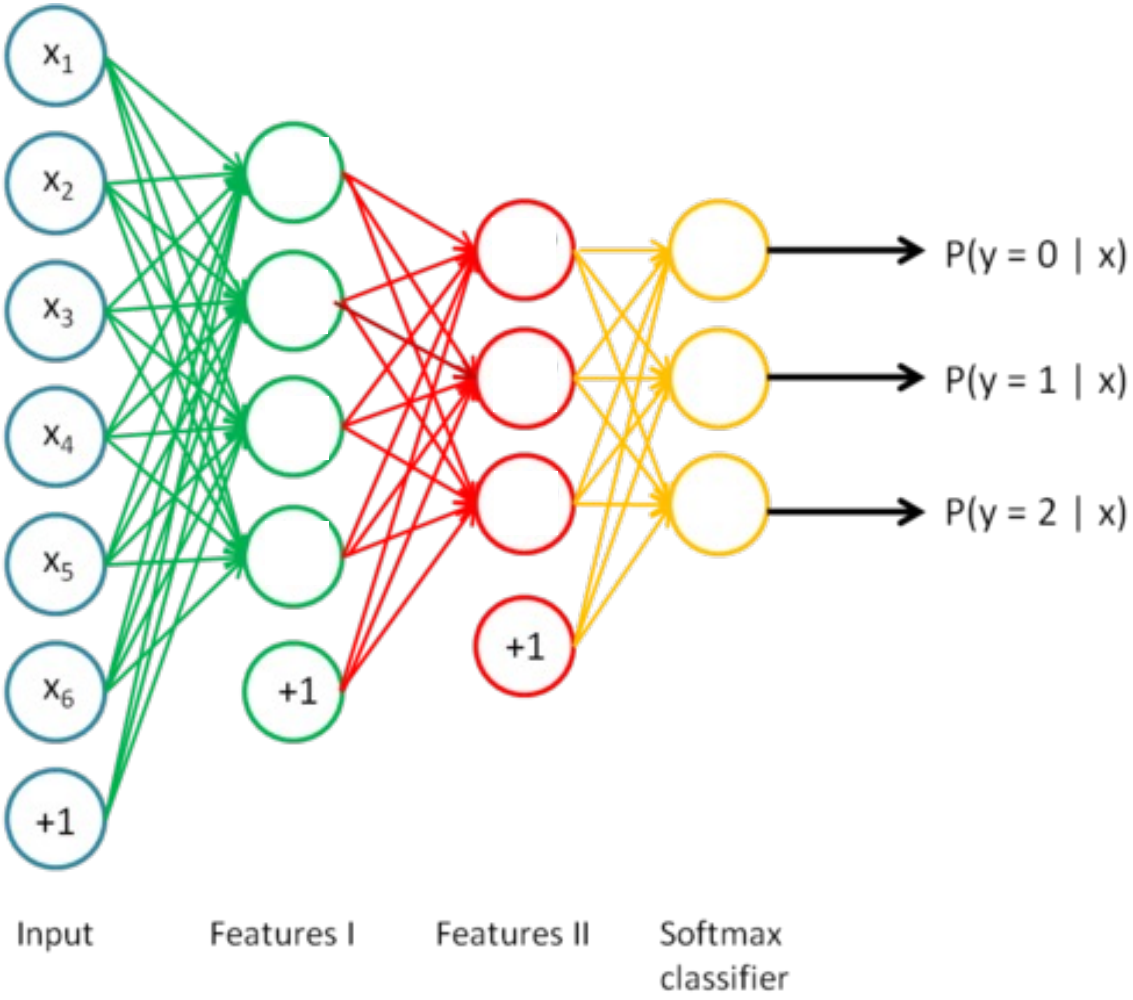
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

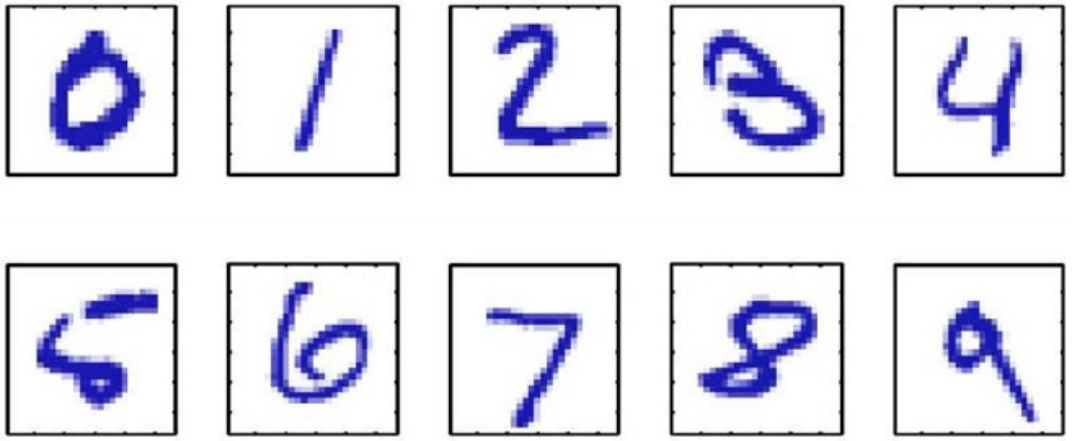
Cross-entropy loss



Multi-class classification



MNIST: Handwritten digit recognition



Images are 28 x 28 pixels

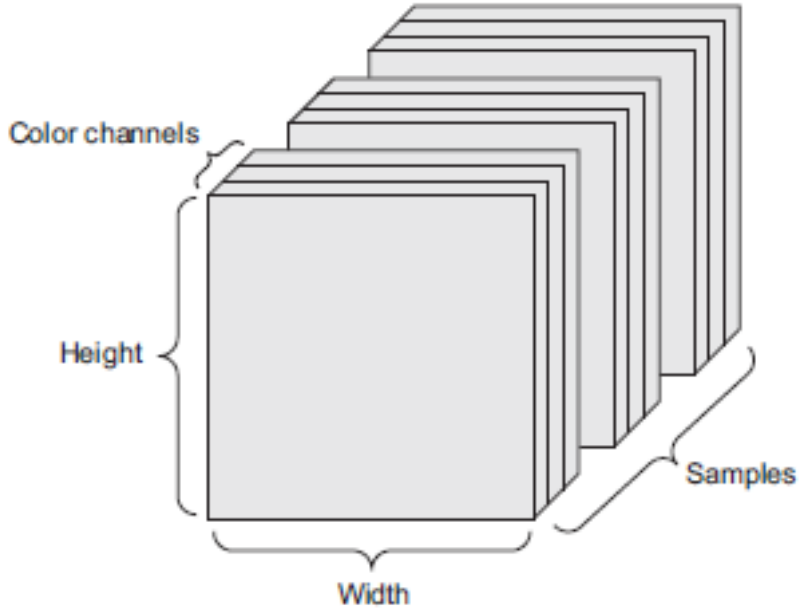
Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$
Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Predict the digit
Multi-class classifier

Image Representation

- Image is 3D “tensor”: height, width, color channel (RGB)
- Black-and-white images are 2D matrices: height, width
  - Each value is pixel intensity



Lab – Feed Forward NN

```
import time
import numpy as np
#!pip install tensorflow
#!pip install keras

from keras.utils import np_utils
import keras.callbacks as cb
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from keras.datasets import mnist

import matplotlib
import matplotlib.pyplot as plt
```

```
def load_data():
    print("Loading data")
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')

    # Normalize
    X_train /= 255
    X_test /= 255

    y_train = np_utils.to_categorical(y_train, 10)
    y_test = np_utils.to_categorical(y_test, 10)

    X_train = np.reshape(X_train, (60000, 784))
    X_test = np.reshape(X_test, (10000, 784))

    print("Data loaded")
    return [X_train, X_test, y_train, y_test]
```

Lab – Feed Forward NN

```
import time
import numpy as np
#!pip install tensorflow
#!pip install keras

from keras.utils import np_utils
import keras.callbacks as cb
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from keras.datasets import mnist

import matplotlib
import matplotlib.pyplot as plt
```

Import modules

```
def load_data():
    print("Loading data")
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')

    # Normalize
    X_train /= 255
    X_test /= 255

    y_train = np_utils.to_categorical(y_train, 10)
    y_test = np_utils.to_categorical(y_test, 10)

    X_train = np.reshape(X_train, (60000, 784))
    X_test = np.reshape(X_test, (10000, 784))

    print("Data loaded")
    return [X_train, X_test, y_train, y_test]
```

Load MNIST data
Processing

Vector
representation

Neural Network Architecture

```
def init_model1():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(10, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished "+format(time.time() - start_time))
    return model
```

Feed-Forward Neural Network Architecture

- 1 Hidden Layer (“Dense” or Fully Connected)
- 10 neurons
- Output layer uses softmax activation

Neural Network Architecture

```
def init_model():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(10, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

10 hidden units
ReLU activation

Output Layer
Softmax activation

Loss function

Optimizer

Feed-Forward Neural Network Architecture

- 1 Hidden Layer (“Dense” or Fully Connected)
- 10 neurons
- Output layer uses softmax activation

Number of Parameters

Number of Parameters

```
model1.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 10)                7850
_________________________________________________________________
activation_16 (Activation)   (None, 10)                0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                110
_________________________________________________________________
activation_17 (Activation)   (None, 10)                0
=================================================================
Total params: 7,960
Trainable params: 7,960
Non-trainable params: 0
_________________________________________________________________
```

Train and evaluate

```
def run_network(data=None, model=None, epochs=20, batch=256):
    try:
        start_time = time.time()
        if data is None:
            X_train, X_test, y_train, y_test = load_data()
        else:
            X_train, X_test, y_train, y_test = data

        print("Training model")
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch,
                  validation_data=(X_test, y_test), verbose=2)

        print("Training duration:"+format(time.time() - start_time))
        score = model.evaluate(X_test, y_test, batch_size=16)

        print("\nNetwork's test loss and accuracy:"+format(score))
        return history
    except KeyboardInterrupt:
        print("KeyboardInterrupt")
        return history
```

Training/testing results

```
Compiling Model
Model finished 0.04014420509338379
Loading data
Data loaded
Training model
Epoch 1/10
235/235 - 1s - loss: 0.9142 - accuracy: 0.7501 - val_loss: 0.4398 - val_accuracy: 0.8833
Epoch 2/10
235/235 - 0s - loss: 0.3856 - accuracy: 0.8959 - val_loss: 0.3392 - val_accuracy: 0.9050
Epoch 3/10
235/235 - 0s - loss: 0.3245 - accuracy: 0.9093 - val_loss: 0.3043 - val_accuracy: 0.9141
Epoch 4/10
235/235 - 0s - loss: 0.2992 - accuracy: 0.9165 - val_loss: 0.2890 - val_accuracy: 0.9178
Epoch 5/10
235/235 - 0s - loss: 0.2853 - accuracy: 0.9202 - val_loss: 0.2797 - val_accuracy: 0.9214
Epoch 6/10
235/235 - 0s - loss: 0.2755 - accuracy: 0.9234 - val_loss: 0.2735 - val_accuracy: 0.9217
Epoch 7/10
235/235 - 0s - loss: 0.2690 - accuracy: 0.9251 - val_loss: 0.2689 - val_accuracy: 0.9252
Epoch 8/10
235/235 - 0s - loss: 0.2634 - accuracy: 0.9263 - val_loss: 0.2658 - val_accuracy: 0.9271
Epoch 9/10
235/235 - 0s - loss: 0.2590 - accuracy: 0.9276 - val_loss: 0.2666 - val_accuracy: 0.9257
Epoch 10/10
235/235 - 0s - loss: 0.2554 - accuracy: 0.9284 - val_loss: 0.2616 - val_accuracy: 0.9284
Training duration:3.1347107887268066
625/625 [==============================] - 1s 707us/step - loss: 0.2616 - accuracy: 0.9284

Network's test loss and accuracy:[0.2615792751312256, 0.9283999800682068]
```

Training/testing results

```
Epoch 98/100
235/235 - 0s - loss: 0.1611 - accuracy: 0.9552 - val_loss: 0.2329 - val_accuracy: 0.9411
Epoch 99/100
235/235 - 0s - loss: 0.1609 - accuracy: 0.9550 - val_loss: 0.2334 - val_accuracy: 0.9392
Epoch 100/100
235/235 - 0s - loss: 0.1603 - accuracy: 0.9550 - val_loss: 0.2323 - val_accuracy: 0.9401
Training duration:22.163272857666016
625/625 [==============================] - 1s 711us/step - loss: 0.2323 - accuracy: 0.9401

Network's test loss and accuracy:[0.23233847320079803, 0.9401000142097473]
```

Changing Number of Neurons

```
def init_model2():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(500, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished "+format(time.time() - start_time))
    return model
```

```
Epoch 98/100
235/235 - 1s - loss: 1.1261e-08 - accuracy: 1.0000 - val_loss: 0.1432 - val_accuracy: 0.9835
Epoch 99/100
235/235 - 1s - loss: 1.1088e-08 - accuracy: 1.0000 - val_loss: 0.1432 - val_accuracy: 0.9834
Epoch 100/100
235/235 - 1s - loss: 1.0918e-08 - accuracy: 1.0000 - val_loss: 0.1435 - val_accuracy: 0.9835
Training duration:82.20909094810486
625/625 [==============================] - 1s 932us/step - loss: 0.1435 - accuracy: 0.9835

Network's test loss and accuracy:[0.1434623748064041, 0.9835000038146973]
```

Number of Parameters

Number of Parameters

```
model2.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 500)               392500
_________________________________________________________________
activation_22 (Activation)   (None, 500)               0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                5010
_________________________________________________________________
activation_23 (Activation)   (None, 10)                0
=================================================================
Total params: 397,510
Trainable params: 397,510
Non-trainable params: 0
_________________________________________________________________
```

Two Layers

```
def init_model4():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(500, input_dim=784))
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    model.add(Dense(300))
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

```
Epoch 98/100
235/235 - 1s - loss: 0.0164 - accuracy: 0.9961 - val_loss: 0.1677 - val_accuracy: 0.9844
Epoch 99/100
235/235 - 1s - loss: 0.0146 - accuracy: 0.9966 - val_loss: 0.1701 - val_accuracy: 0.9841
Epoch 100/100
235/235 - 1s - loss: 0.0134 - accuracy: 0.9968 - val_loss: 0.1666 - val_accuracy: 0.9849
Training duration:131.49207520484924
625/625 [==============================] - 1s 1ms/step - loss: 0.1666 - accuracy: 0.9849

Network's test loss and accuracy:[0.16656503081321716, 0.9848999977111816]
```

Number of Parameters

Model Parameters

```
model4.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 500)               392500
_________________________________________________________________
activation_26 (Activation)   (None, 500)               0
_________________________________________________________________
dropout_9 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_27 (Dense)             (None, 300)               150300
_________________________________________________________________
activation_27 (Activation)   (None, 300)               0
_________________________________________________________________
dropout_10 (Dropout)         (None, 300)               0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                3010
_________________________________________________________________
activation_28 (Activation)   (None, 10)                0
=================================================================
Total params: 545,810
Trainable params: 545,810
Non-trainable params: 0
_________________________________________________________________
```

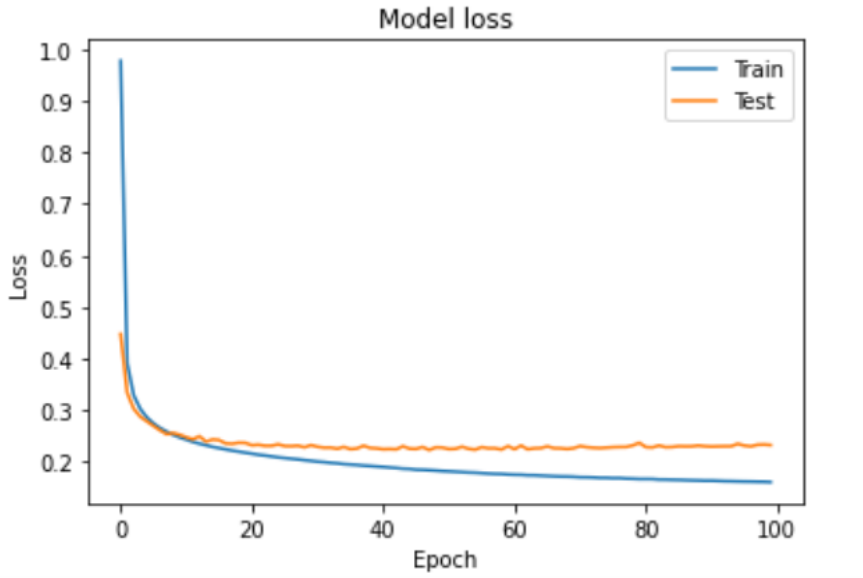
Monitor Loss

```
def plot_losses(hist):
    plt.plot(hist.history['loss'])
    plt.plot(hist.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper right')
    plt.show()
```

```
model1 = init_model1()

history1 = run_network(model = model1, epochs=100)

plot_losses(history1)
```



Loss

```
model2 = init_model2()

history2 = run_network(model = model2, epochs=100)

plot_losses(history2)
```

```
model4 = init_model4()

history4 = run_network(model = model4, epochs=100)

plot_losses(history4)
```