DS 4400

Machine Learning and Data Mining I
Spring 2021

Alina Oprea
Associate Professor
Khoury College of Computer Science
Northeastern University

March 23 2021

Announcements

- HW 4 is due on Friday, March 26
- Project milestone due on March 31
  - Template in Gradescope
- Final exam on Tuesday, April 6
  - Review on Thursday, April 1
- Last homework on ethics
  - After ethics class (April 8)
  - Group assignment

Outline

- Feed Forward Neural Networks
  - Forward Propagation
  - Hyper-parameters
  - Activations
- Multi-class classification
  - The softmax classifier
- Examples
- Keras tutorial

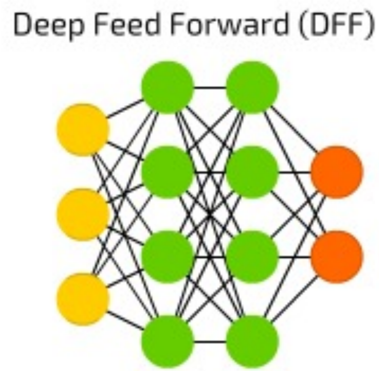
References

- 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

Neural Network Architectures

Feed-Forward Networks

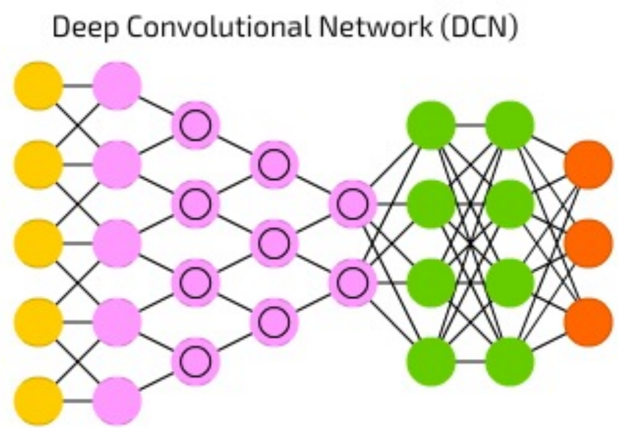
- Neurons from each layer connect to neurons from next layer



MLP

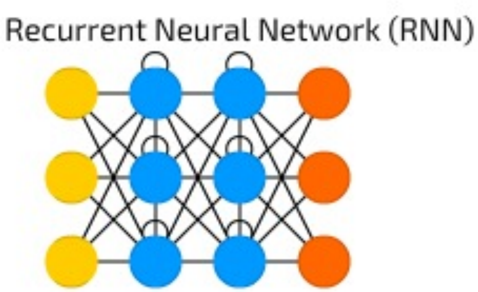
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations



Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



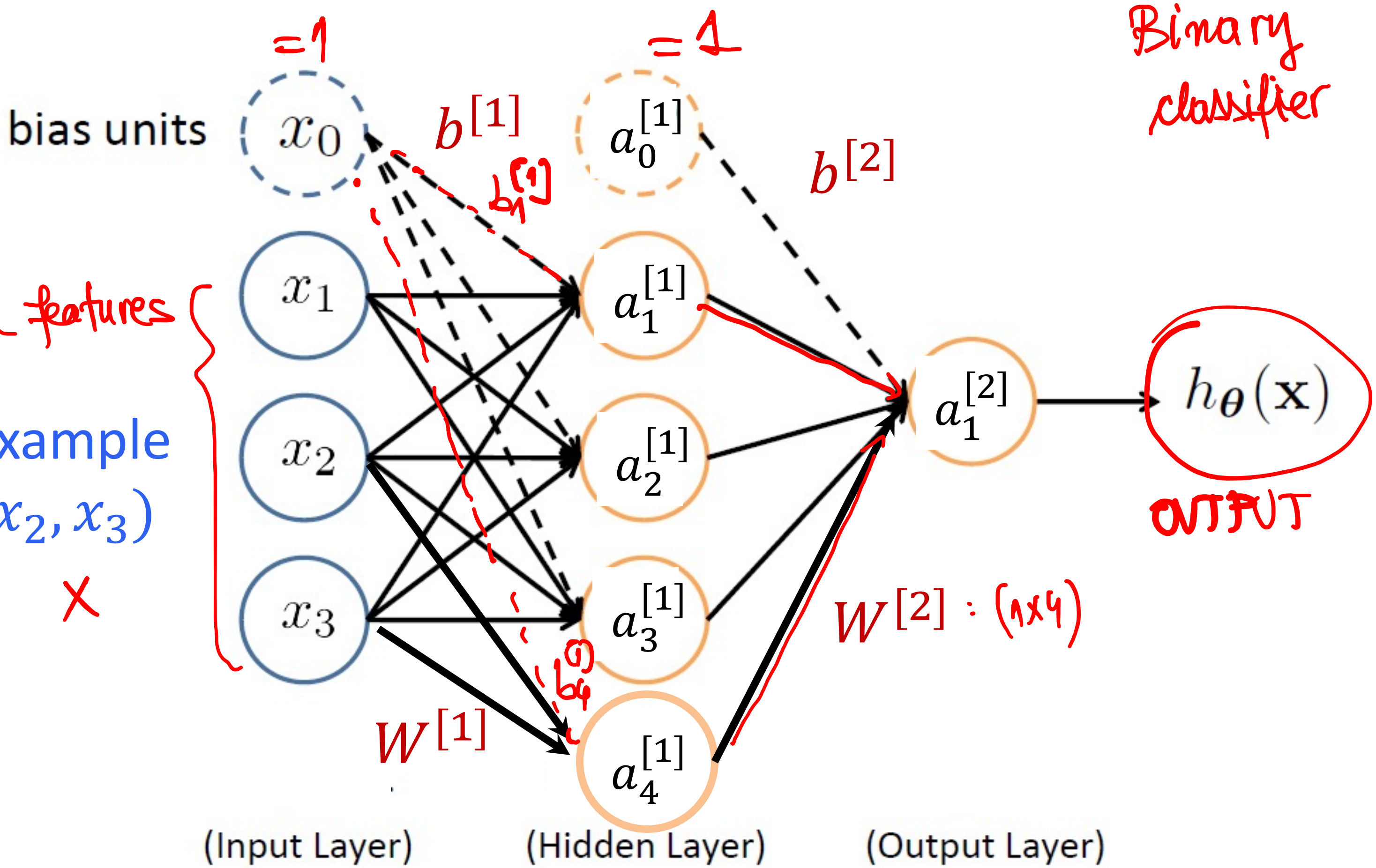
Feed-Forward Neural Network

Binary classifier

d features

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

X



Layer 0 Layer 1 Layer 2

No cycles

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

vector matrix

Hyperparameters

- Architecture
    - # Layers (hidden)
    - Activation function
    - # neurons per layer
- Hyper-param for GD
    - Learning rate; stopping condition
- Regularization

Vectorization: First Layer

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

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix} \quad W^{[1]} = \begin{bmatrix} w_1^{[1]} \\ w_2^{[1]} \\ \vdots \\ w_4^{[1]} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}$$

size: (4×1) size: (4×3) (3×1) (4×1)

$\downarrow$ # neurons Layer 1 $\downarrow$ d

$$a^{[1]} = g(z^{[1]}) = \begin{bmatrix} g(z_1^{[1]}) \\ \vdots \\ g(z_4^{[1]}) \end{bmatrix}$$ NON-LINEAR ACTIVATION

size (4×1)

Vectorization: Second Layer

$$z^{[2]} = \underbrace{W^{[2]}}_{(1\times4)} \cdot \underbrace{a^{[1]}}_{(4\times1)} + \underbrace{b^{[2]}}_{(1\times1)}$$ LINEAR

(1×1)

$\vdots$

$$z^{[j]} = W^{[j]} \cdot a^{[j-1]} + b^{[j]}$$

$$a^{[2]} = g(z^{[2]})$$ ACTIVATION

OUTPUT : $h_\theta(x) = a^{[2]}$

Terminology

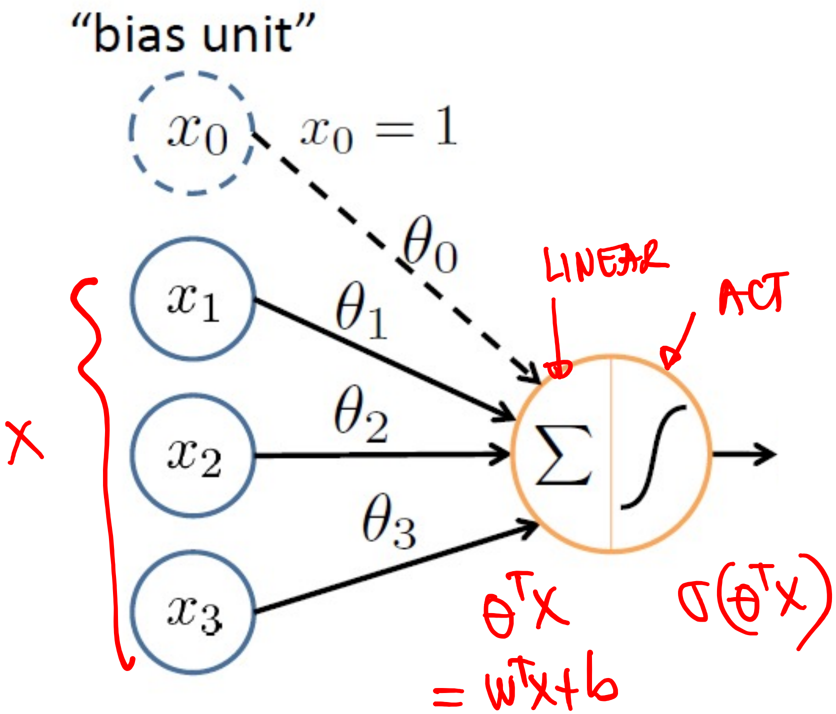
g = act. function

$a_i^{[j]}$ = act. of unit i at layer j

$W^{[j]}$ = weight matrix layer j

$b^{[j]}$ = bias vector layer j

Logistic Unit: A simple NN



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

$\theta_0 = b$, $\quad \underbrace{\theta_1, \theta_2, \theta_3}_{w}$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

$$\text{sigmoid}(\theta^T x)$$

$$\|$$

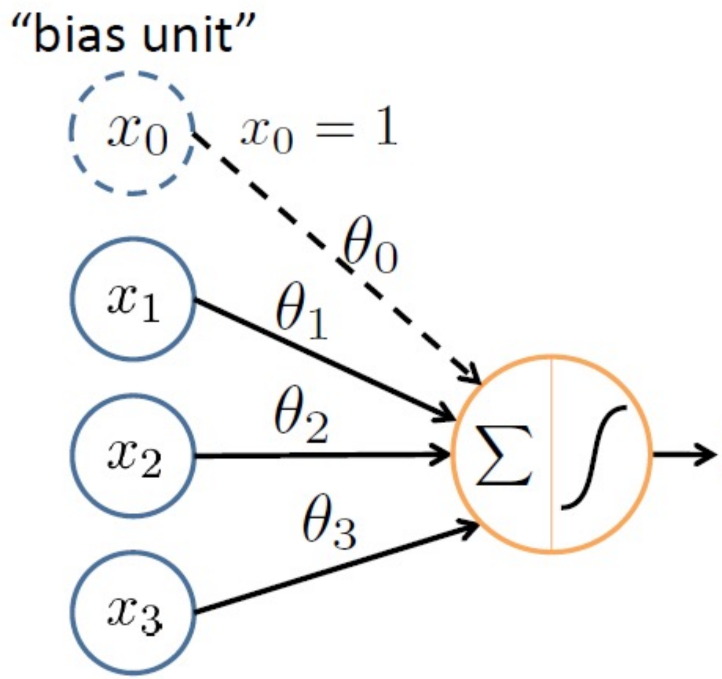
$$w^T x + b$$

$g = \text{sigmoid}$

$W^{[1]} = [\theta_1 \ \theta_2 \ \theta_3]$

$b^{[1]} = \theta_0.$

Logistic Unit: A simple NN



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

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

$$= \frac{1}{1 + e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}}}$$

PERCEPTRON $\Rightarrow$ sign $(\theta^T x) \Rightarrow$ act. func is sign

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

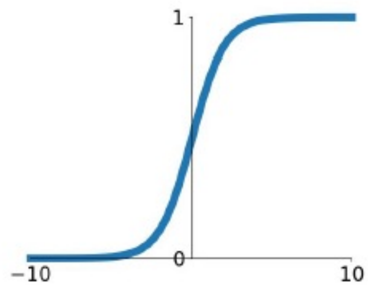
No hidden layers

Activation Functions

last layer

- **Sigmoid**

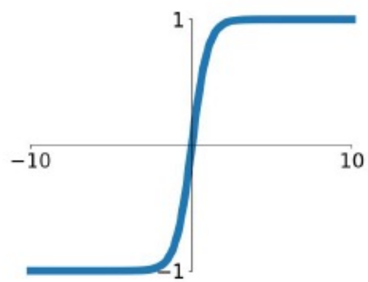
$\sigma(x) = \frac{1}{1+e^{-x}}$



Binary Classification

tanh

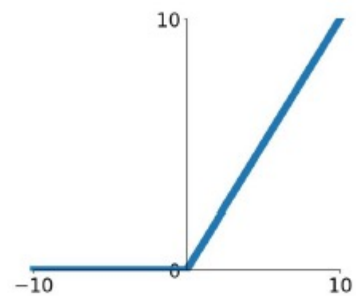
$\tanh(x)$



Regression

- **ReLU**

$\max(0, x)$



Intermediary layers

- softmax

Multi-class classification

Why Non-Linear Activations?

- Assume g is linear: $g(z) = Uz$; U matrix.

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

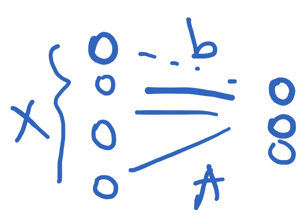
$$a^{[1]} = g(z^{[1]}) = Uz^{[1]} = UW^{[1]}x + Ub^{[1]}$$ LINEAR

Layer 2: $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$$a^{[2]} = g(z^{[2]}) = Uz^{[2]} = UW^{[2]}a^{[1]} + Ub^{[2]}$$

$$= \underbrace{UW^{[2]}UW^{[1]}}_{A}x + \underbrace{UW^{[2]}\cdot Ub^{[1]} + Ub^{[2]}}_{b}$$

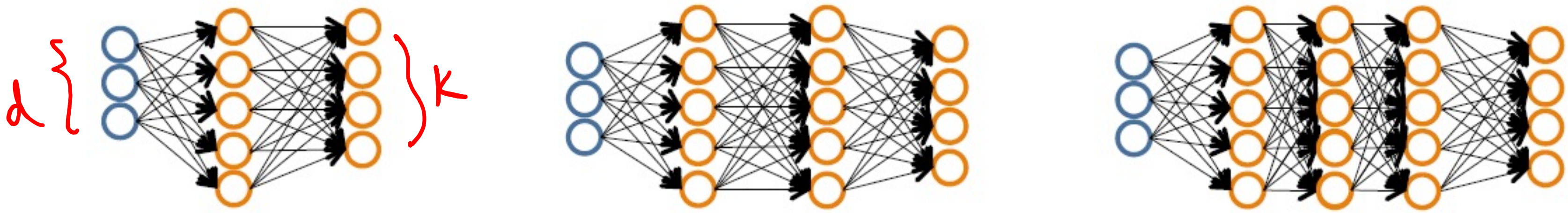
$$= Ax + b$$ LINEAR FUNCTION of INPUT X

Similar to 

; No hidden layers

How to pick architecture?

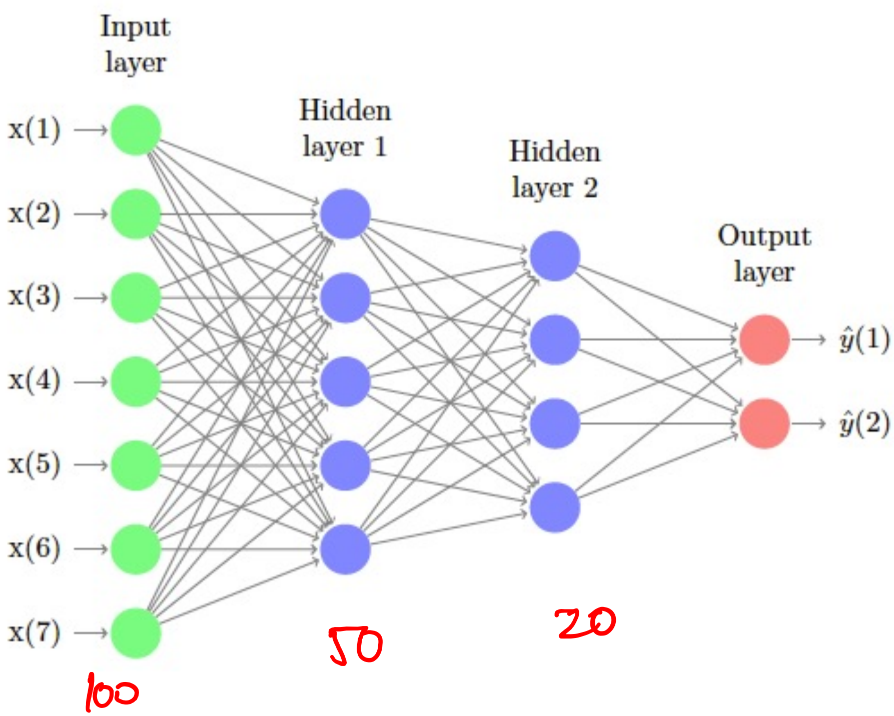
Pick a network architecture (connectivity pattern between nodes)



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

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 : training
  - Learn optimal $W^{[i]}, b^{[i]}$ at layer *i*

testing

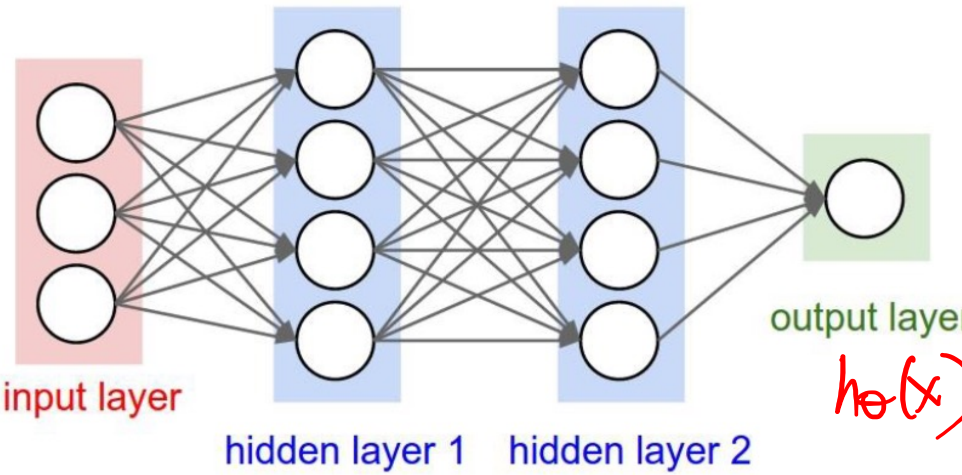
- Evaluation of a point done by forward propagation

Forward Propagation

Fix $w^{[j]}, b^{[j]}$

- 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.

x



$h_\theta(x)$

Testing

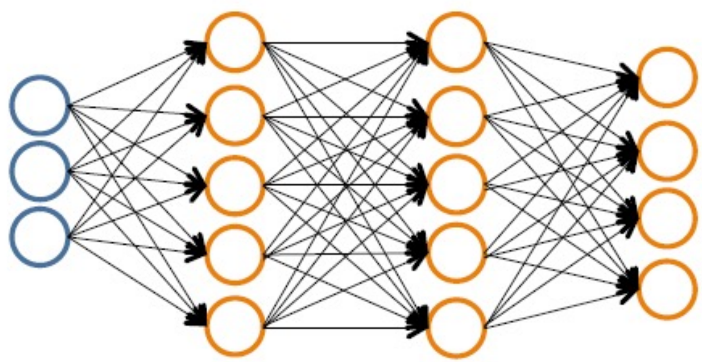
Multi-Class Classsification

Pedestrian

Car

Motorcycle

Truck



4

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

We want: y_i labels.

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

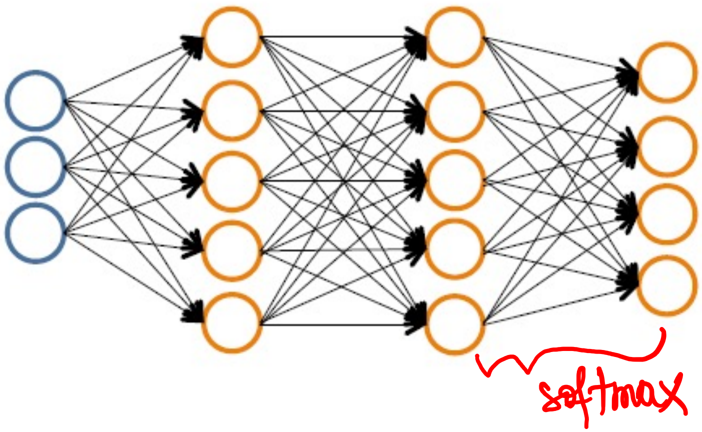
$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car

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

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

one-hot encoding for label

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)

<u>Binary classification</u>

y = 0 or 1

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

classes 0 | 1

Sigmoid

Prob of class 1

<u>Multi-class classification</u> (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{array}{cccc} \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, & \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, & \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, & \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \\ \text{pedestrian} & \text{car} & \text{motorcycle} & \text{truck} \end{array}$

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

$\begin{pmatrix} 0.1 \\ 0 \\ 0.8 \\ 0.1 \end{pmatrix}$

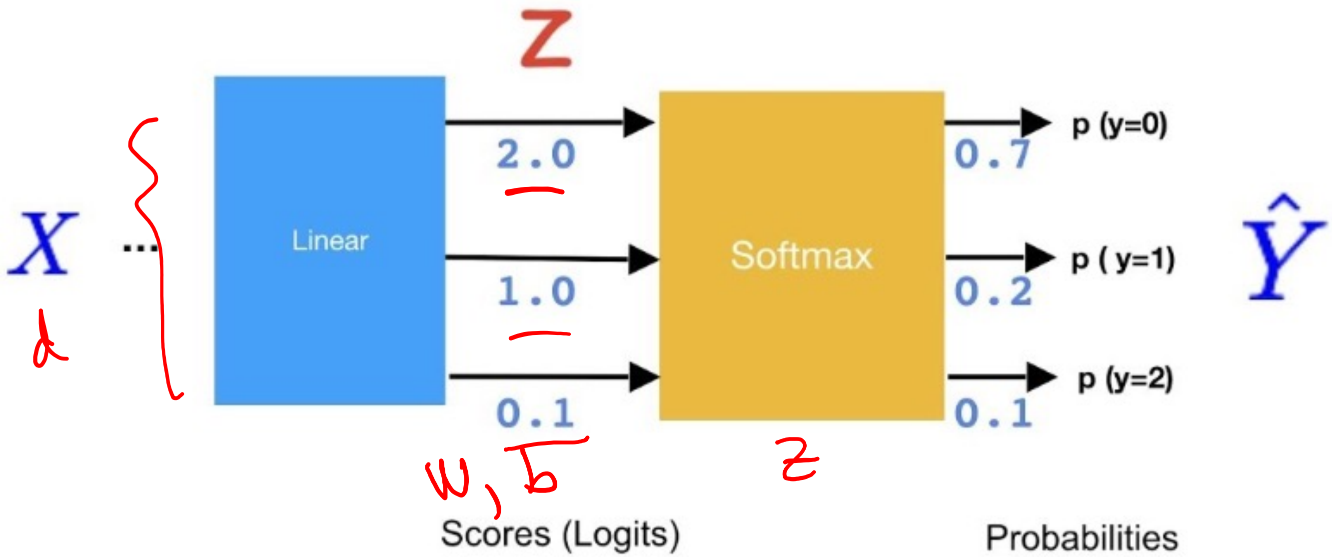
Softmax

Vector of prob. for each class

Output class of max prob.

Softmax classifier

LINEAR.



$z = Wx + b$ logits

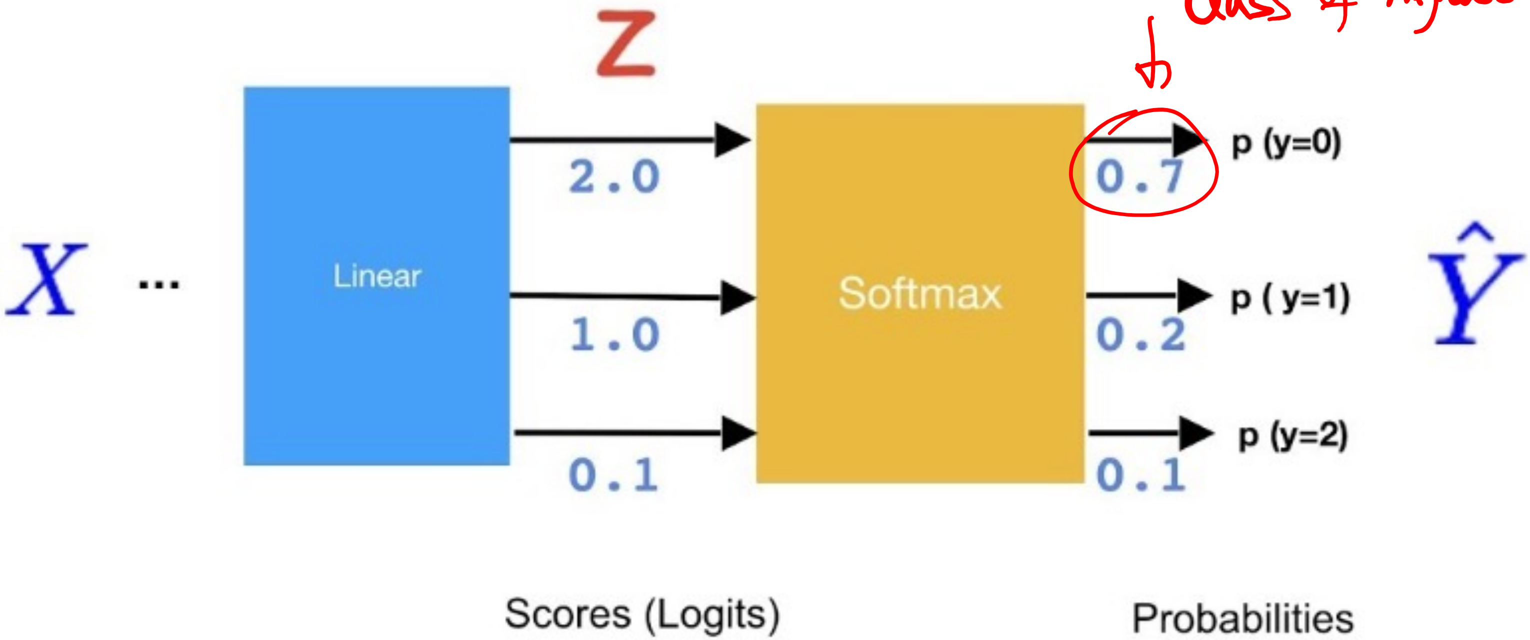
$$\begin{bmatrix} z_1 \\ \vdots \\ z_K \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix}$$

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

softmax is an act. function

Softmax classifier

class of highest prob



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

Softmax function

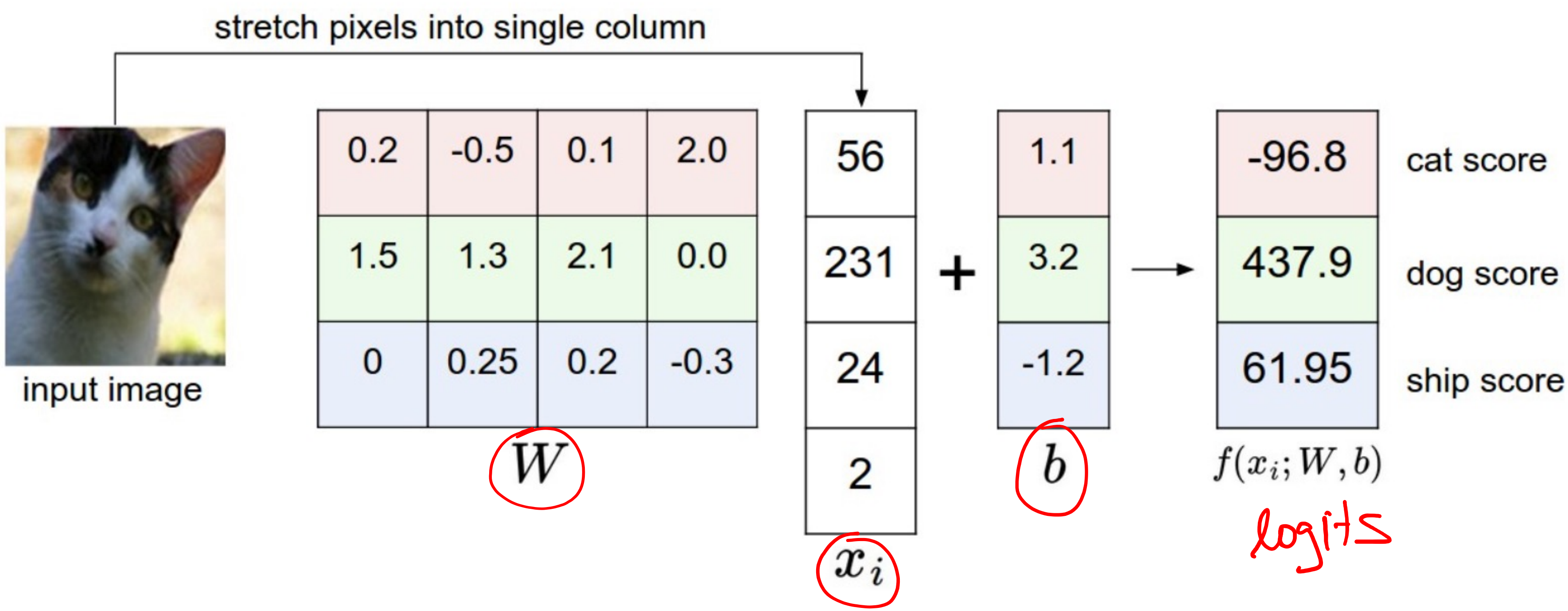
Generalizes the sigmoid

Cross-entropy loss

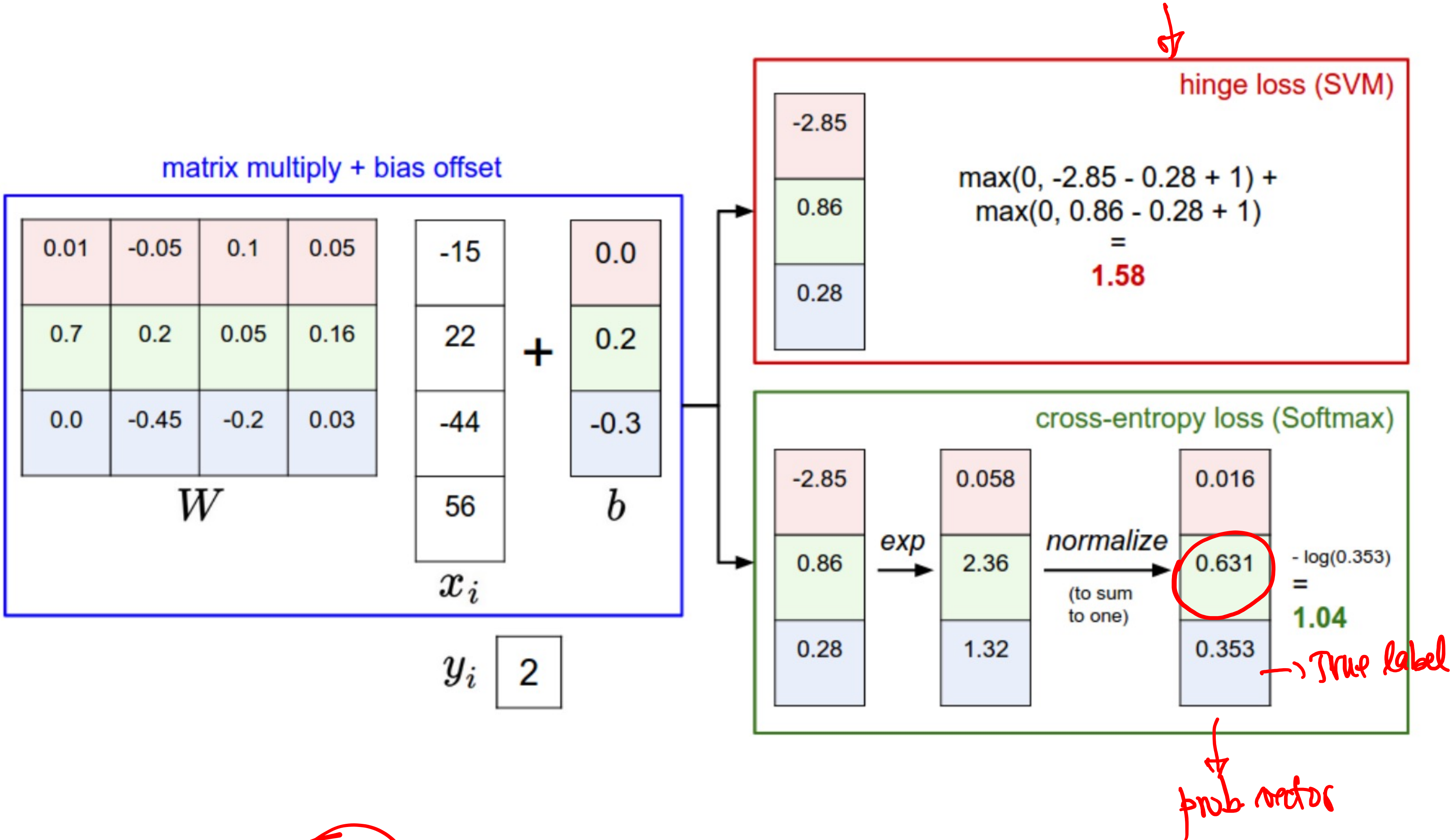
$$L_j = -\log\left(\frac{y_j}{\sum_{k=1}^{K} y_k}\right)$$

y_j = true label of training example

Softmax Example

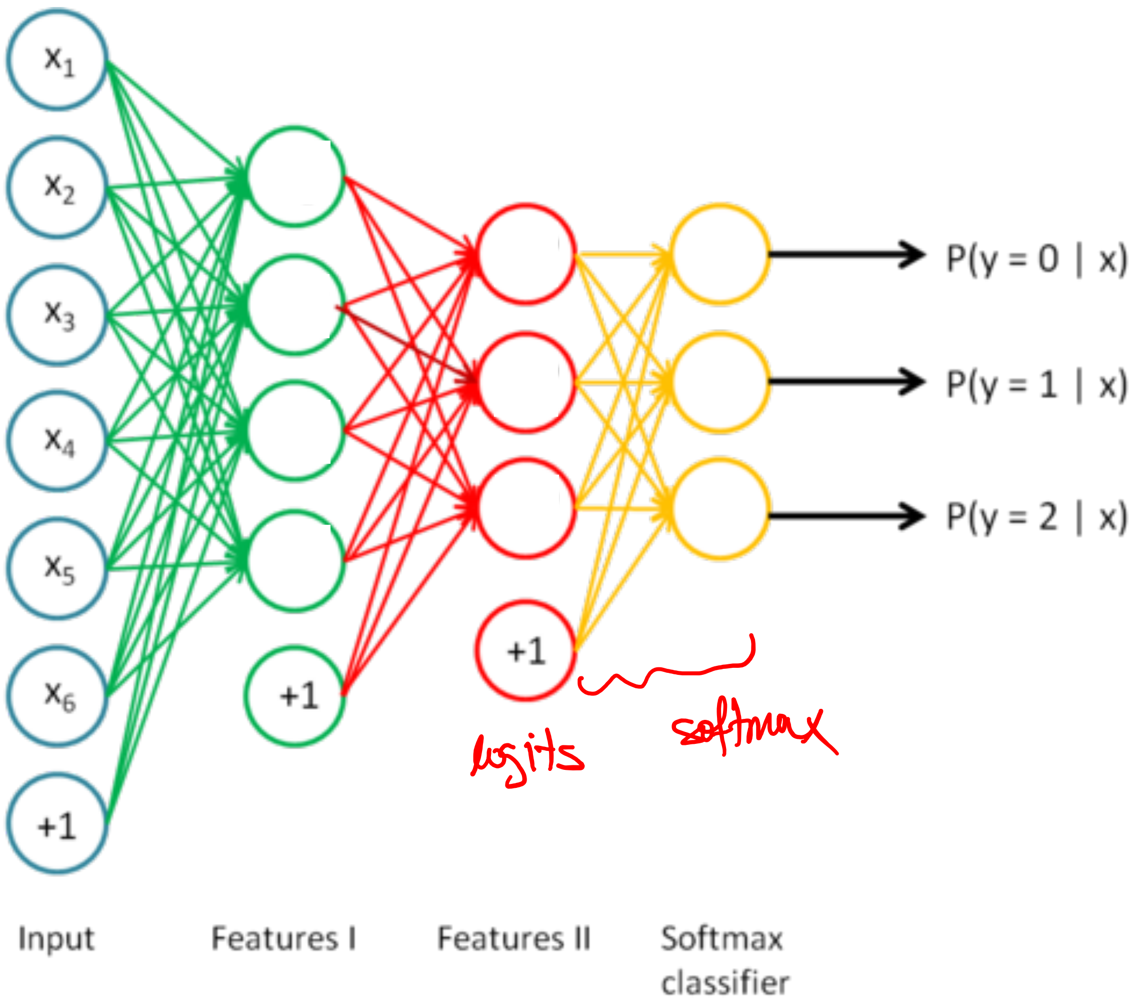


Softmax Example

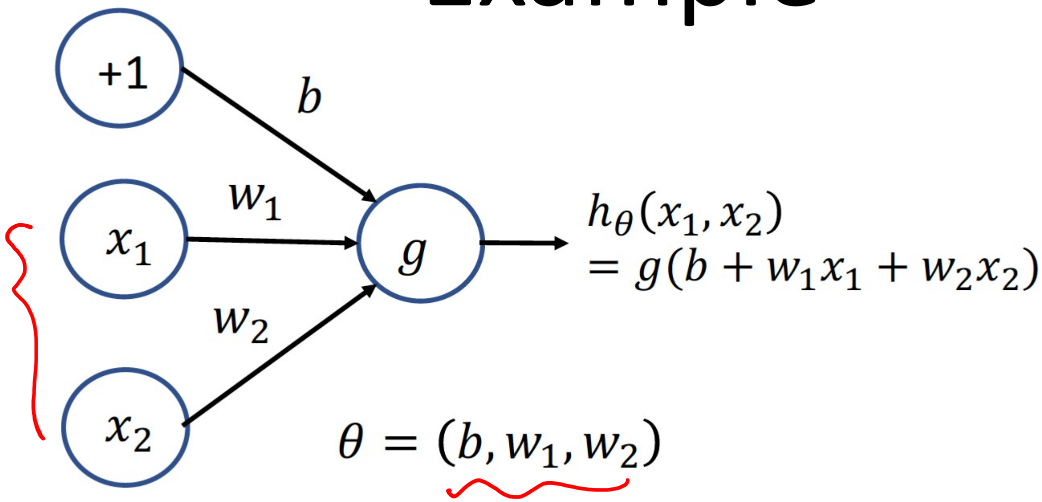


From: https://cs231n.github.io/linear-classify/

Multi-class classification



Example



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

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{h(x_1, x_2)}$
0	0	-1
0	1	-1
1	0	1
1	1	1

sign(-10)
sign(-10+5)
sign(-10+12) = 1
sign(-10+12+5) = 1

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

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{h(x_1, x_2)}$
0	0	1
0	1	1
1	0	1
1	1	0

N AND

1

$b = +12$

$w_1 = -8$

$w_2 = -4$

$$g(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

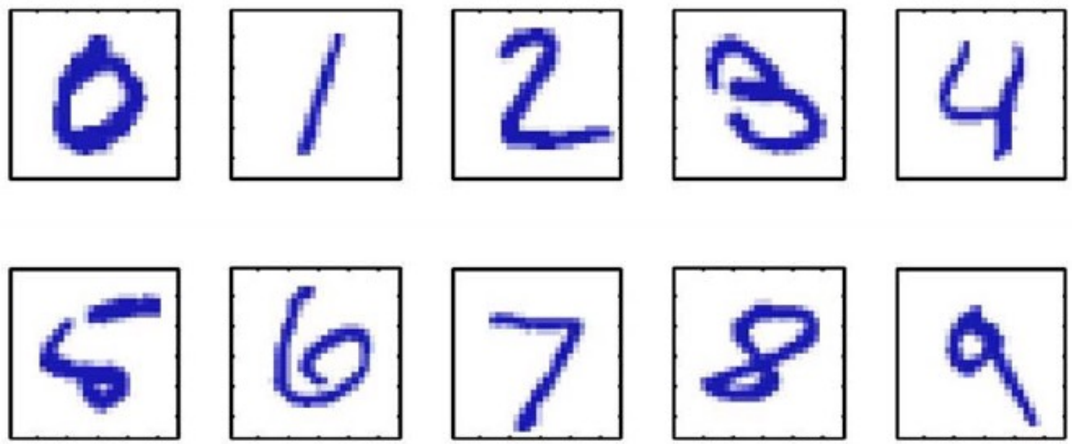
2

$b = 1$

$w_1 = -0.5$

$w_2 = -0.5$

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,

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

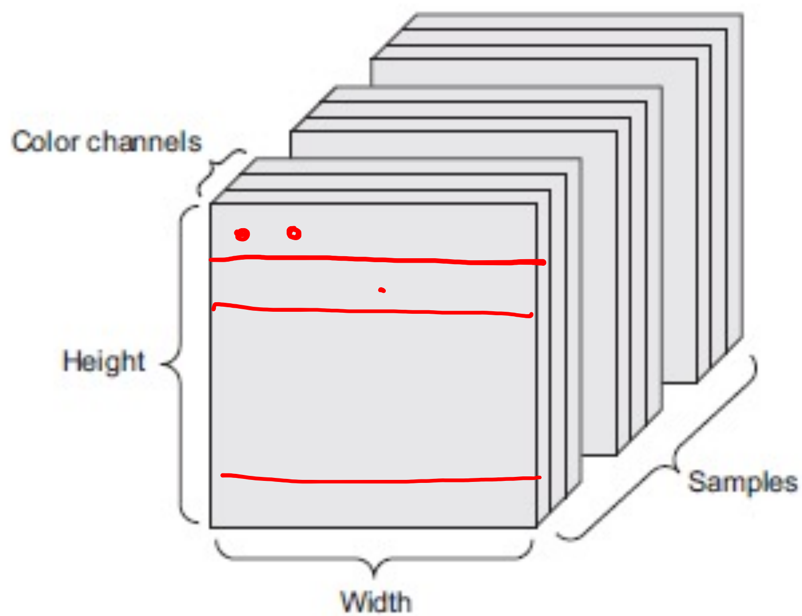
Predict the digit

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

– Pixel $\in [0, 255]$



Row1

Row2

Row 28

28×28
$= 784$

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]
```

$\in [0,1]$

→ VECTOR

Neural Network Architecture

FF-NN INPUT

L1

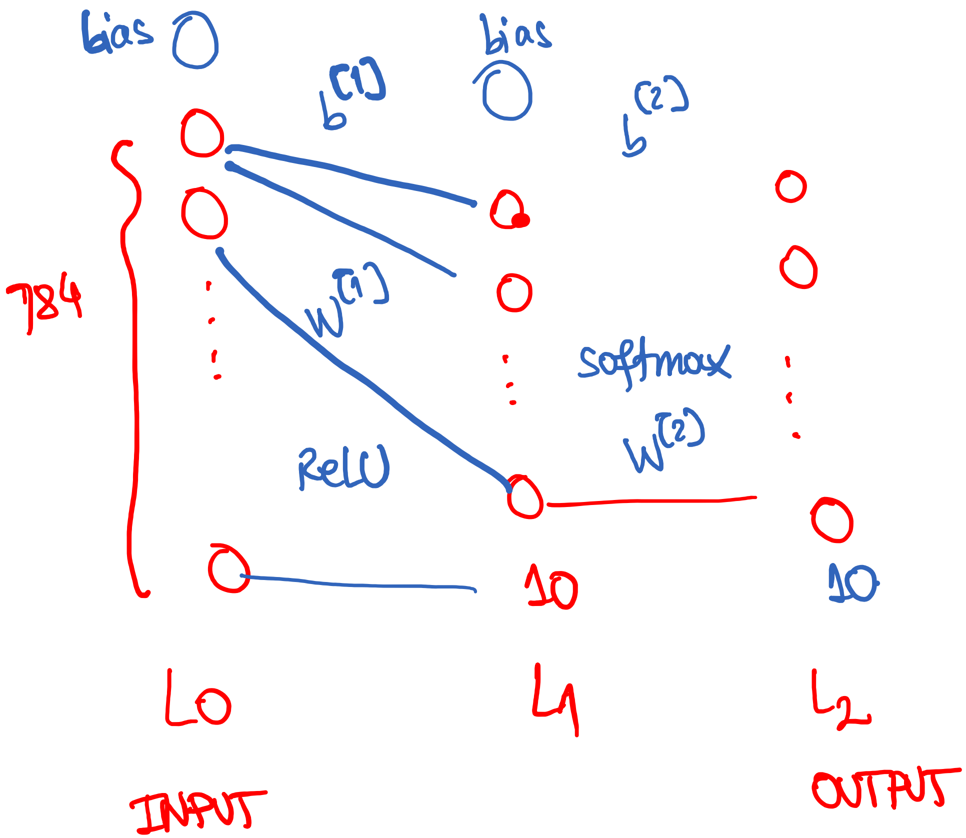
OUTPUT L2

```
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

Number of Parameters



L0 → L1

$$\left.\begin{aligned} W^{[1]} &: 10 \times 784 = 7840 \\ b^{[1]} &: 10 \end{aligned}\right\} 7850$$

L1 → L2

$$\left.\begin{aligned} W^{[2]} &: 10 \times 10 = 100 \\ b^{[2]} &: 10 \end{aligned}\right\} 110$$

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
```

CV.

Training/testing results

Train

Val.

```
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]
```

sec.

Loss

Acc

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 sec.
625/625 [==============================] - 1s 711us/step - loss: 0.2323 - accuracy: 0.9401

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

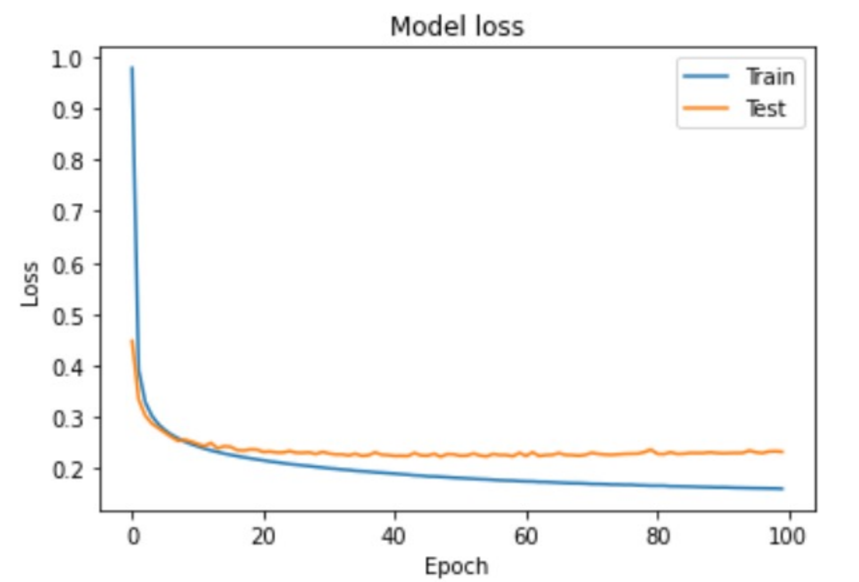
Monitor Loss

```python
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()
```

Train

Val.

```python
model1 = init_model1()

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

plot_losses(history1)
```



Review

- Feed-Forward Neural Networks are the common neural networks architectures
  – Fully connected networks are called Multi-Layer Perceptron
- Input, output, and hidden layers
  – Linear matrix operations followed by non-linear activations at every layer
- Activations:
  – ReLU, tanh, etc., for hidden layers
  – Sigmoid (binary classification) and softmax (for multi-class classification) at last layer
- Forward propagation: process of evaluating input through the network

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
  - Yann LeCun
- Thanks!