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

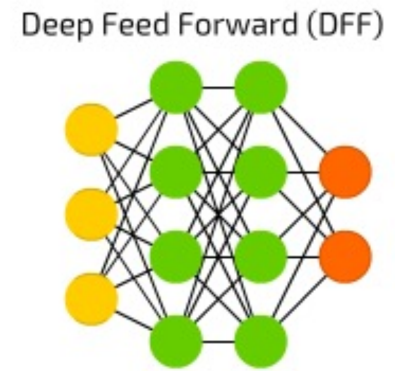
References

- 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

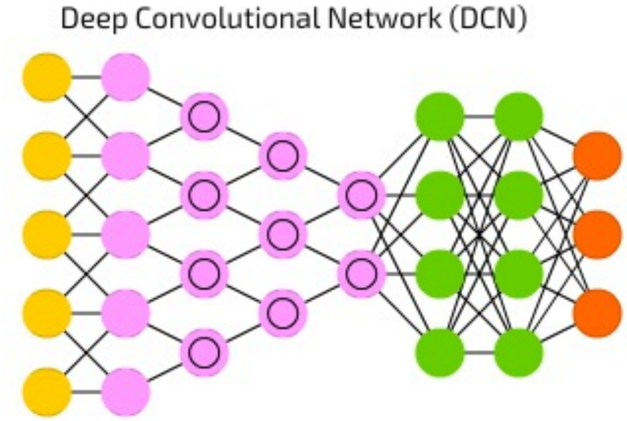
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



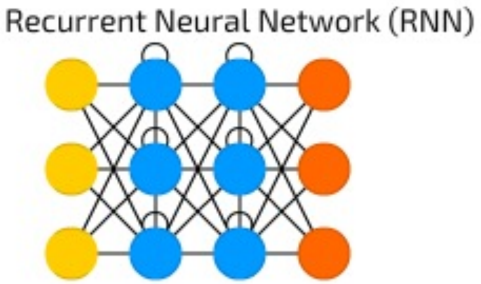
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

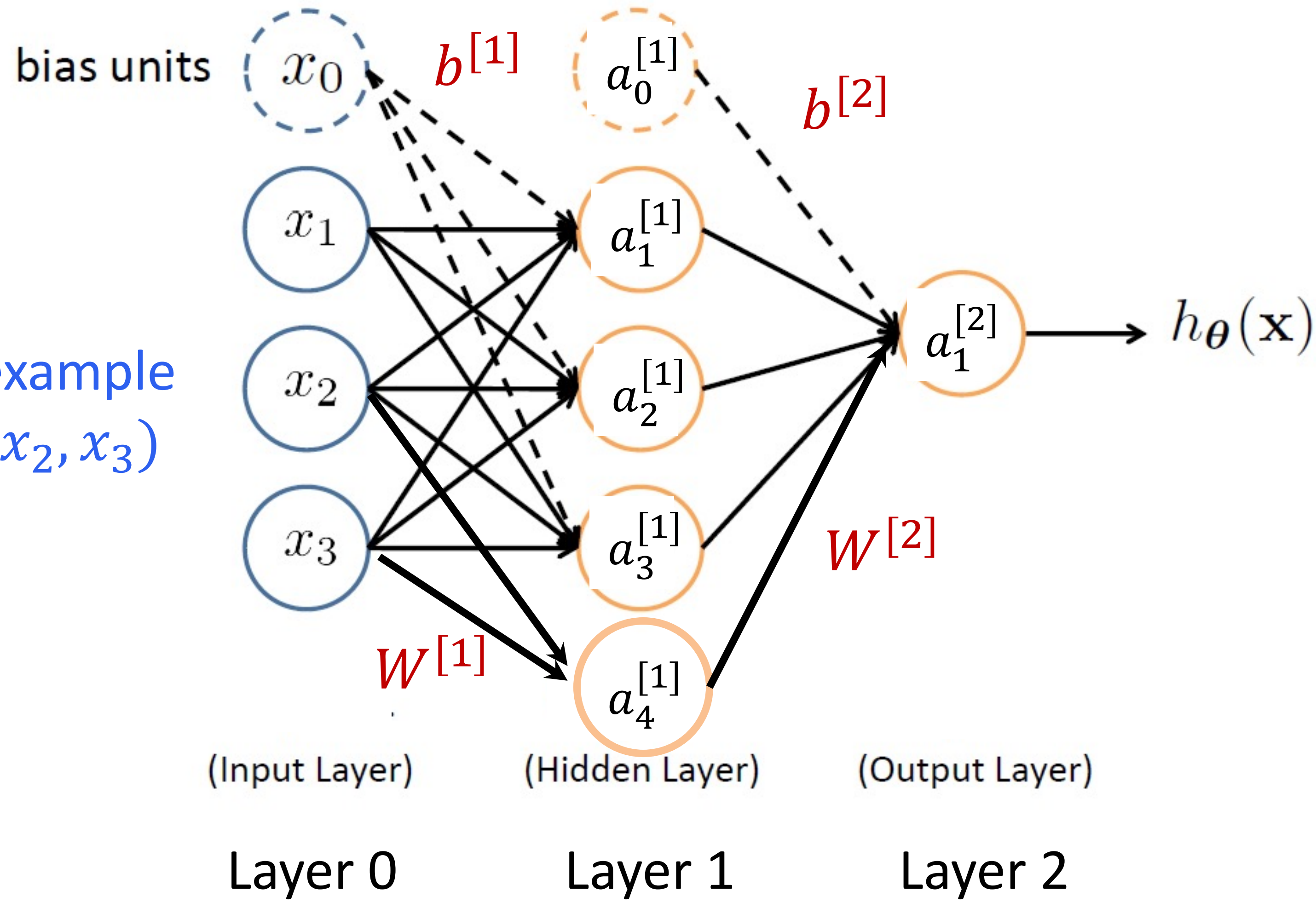


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



Feed-Forward Neural Network



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

No cycles

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

Vectorization

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

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

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

Linear

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

Non-Linear

Vectorization

Output layer

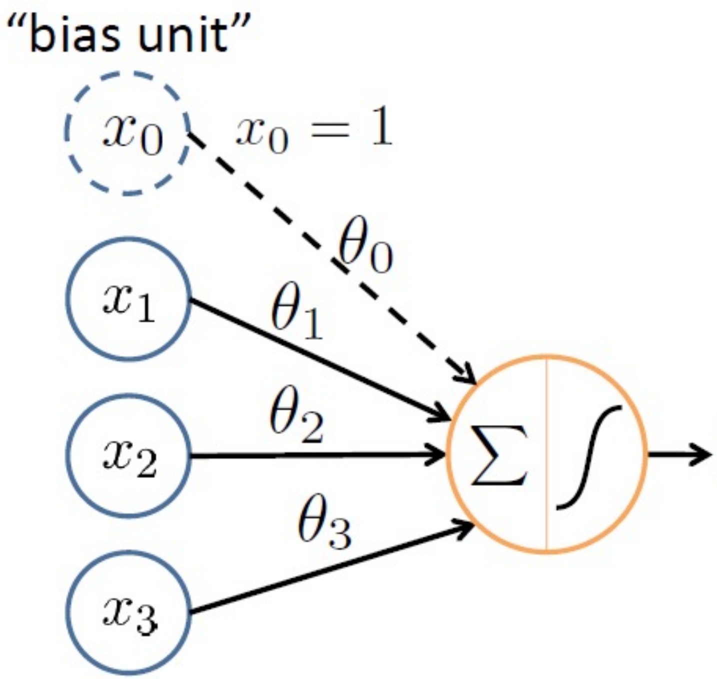
$$z_1^{[2]} = W_1^{[2]^T} a^{[1]} + b_1^{[2]} \quad \text{and} \quad a_1^{[2]} = g(z_1^{[2]})$$

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

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



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

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

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

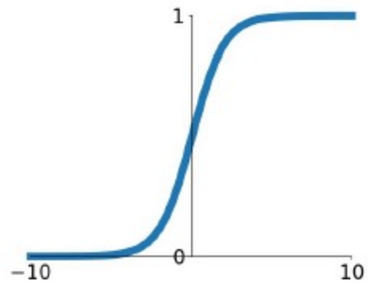
Sigmoid (logistic) activation function: $g(z) = \frac{1}{1 + e^{-z}}$

No hidden layers

Activation Functions

Sigmoid

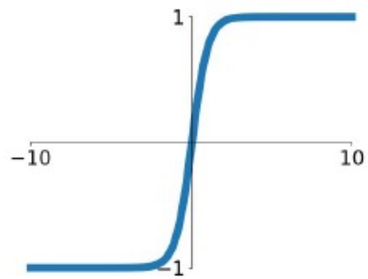
$\sigma(x) = \frac{1}{1+e^{-x}}$



Binary Classification

tanh

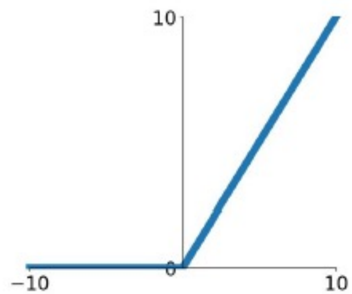
$\tanh(x)$



Regression

ReLU

$\max(0, x)$



Intermediary layers

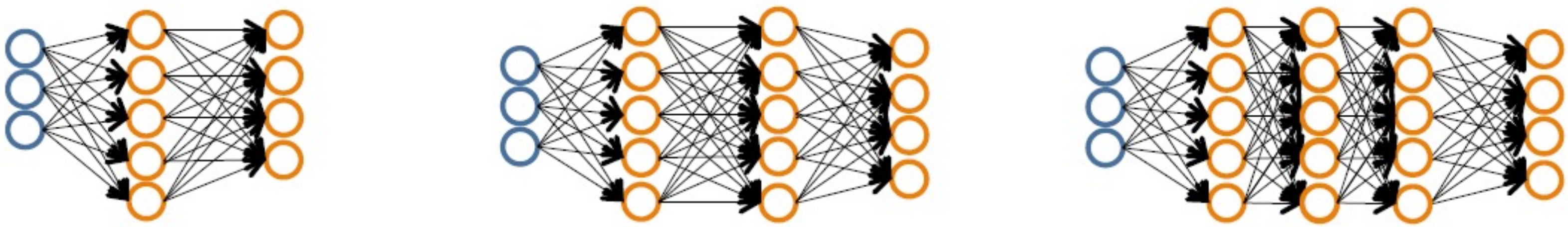
Why Non-Linear Activations?

- Assume g is linear: $\mathrm{g}(z) = Uz$
  - At layer 1: $z^{[1]} = \mathrm{W}^{[1]} x + \mathrm{b}^{[1]}$
  - $a^{[1]} = \mathrm{g}\left(\mathrm{z}^{[1]}\right) = \mathrm{Uz}^{[1]} = \mathrm{UW}^{[1]}\mathrm{x} + \mathrm{Ub}^{[1]}$
- Layer 2:
  - $a^{[2]} = \mathrm{g}\left(\mathrm{z}^{[2]}\right) = \mathrm{Uz}^{[2]} = \mathrm{UW}^{[2]}\mathrm{a}^{[1]} + \mathrm{Ub}^{[2]} =$

  $= \mathrm{UW}^{[2]}\,\mathrm{UW}^{[1]}x + \mathrm{UW}^{[2]}\mathrm{Ub}^{[1]} + \mathrm{Ub}^{[2]}$
- Last layer
  - Output is linear in input!
  - Then NN will only learn linear functions

How to pick architecture?

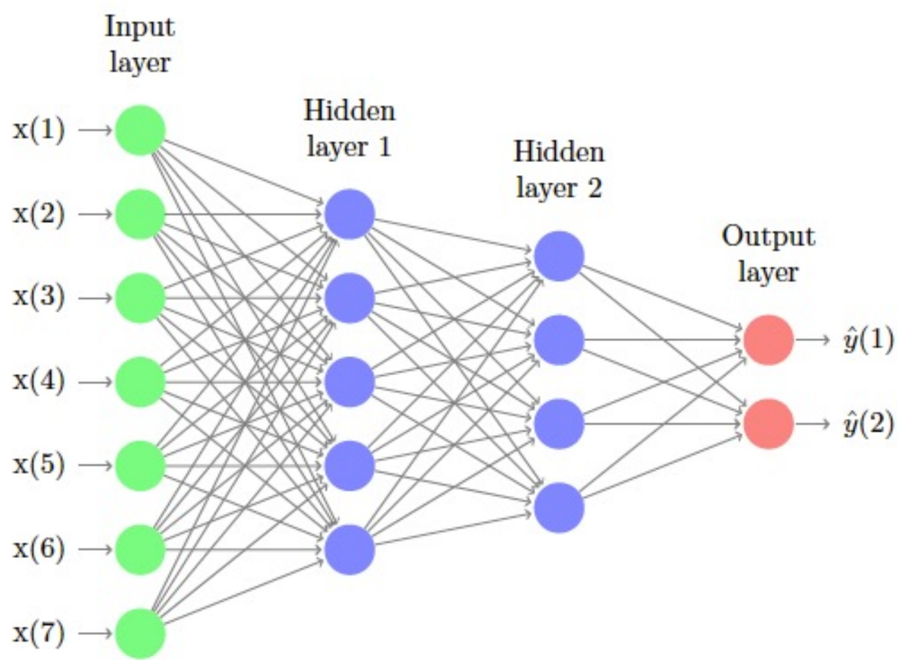
Pick a network architecture (connectivity pattern between nodes)



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

Reasonable default: 1 hidden layer

FFNN Architectures



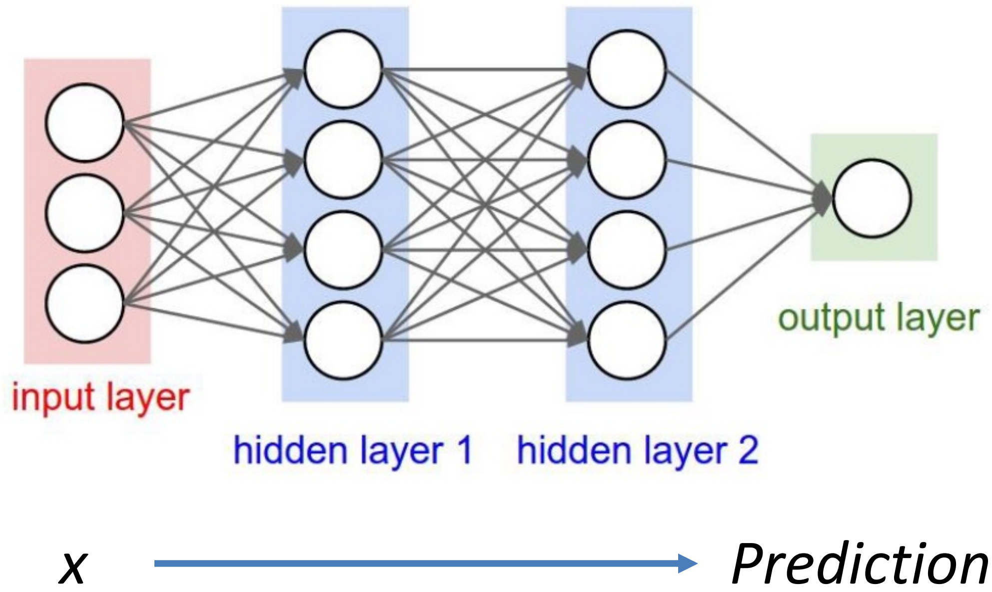
- Input and Output Layers are completely specified by the problem domain
- In the Hidden Layers, number of neurons in Layer i+1 is usually smaller or equal to the number of neurons in Layer i

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.



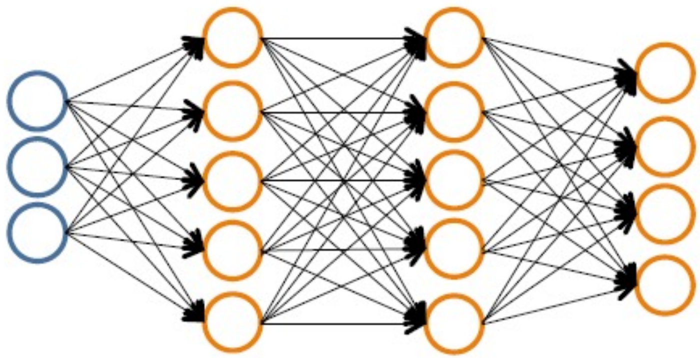
Multi-Class Classsification

Pedestrian

Car

Motorcycle

Truck



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

We want:

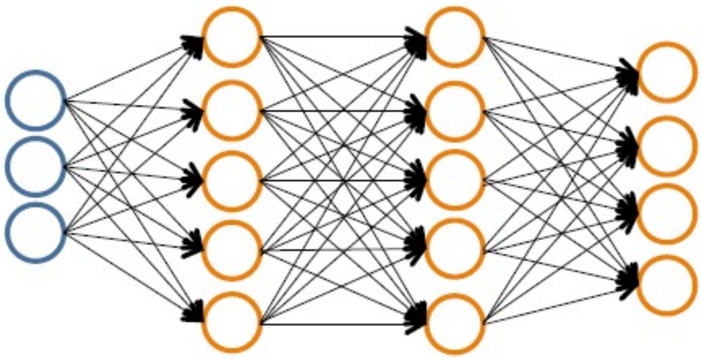
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
when pedestrian

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

<u>Binary classification</u>

y = 0 or 1

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

Sigmoid

<u>Multi-class classification</u> (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

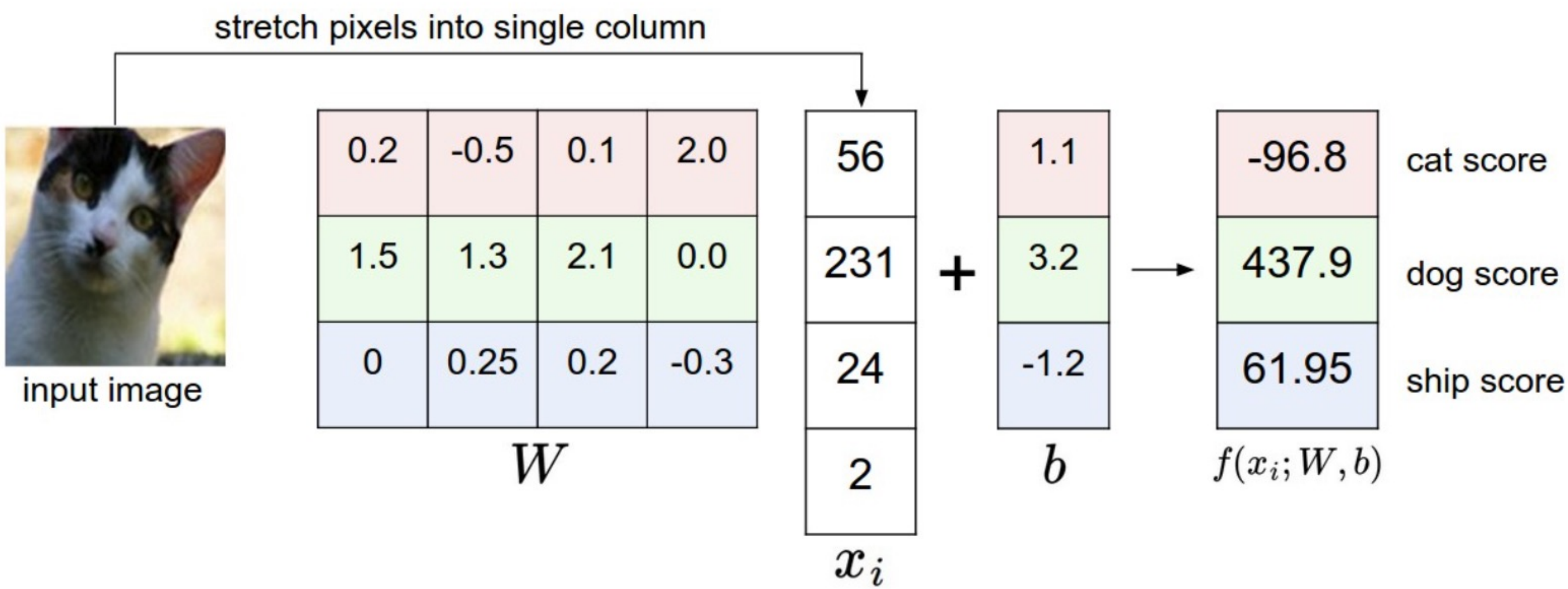
Softmax classifier

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

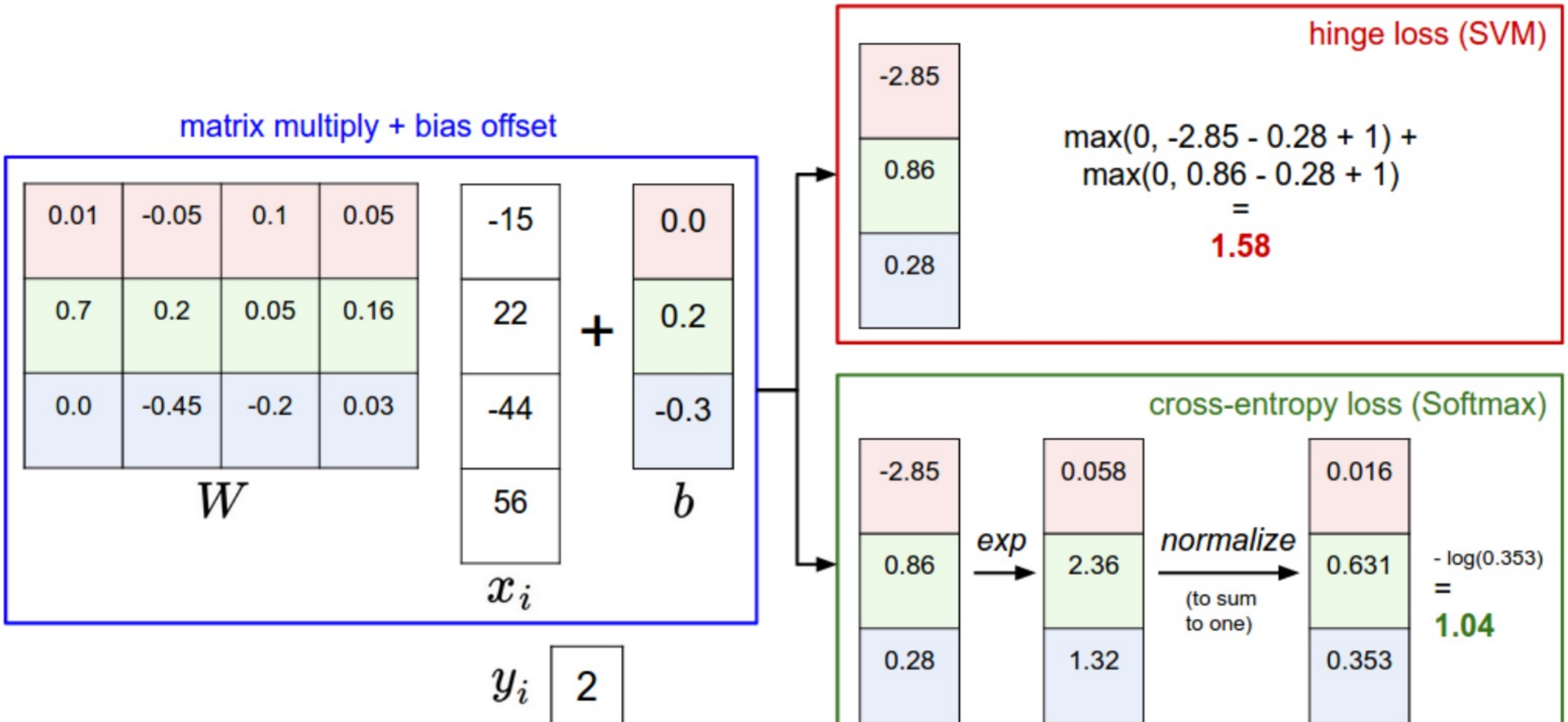
- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

Cross-entropy loss

Softmax Example

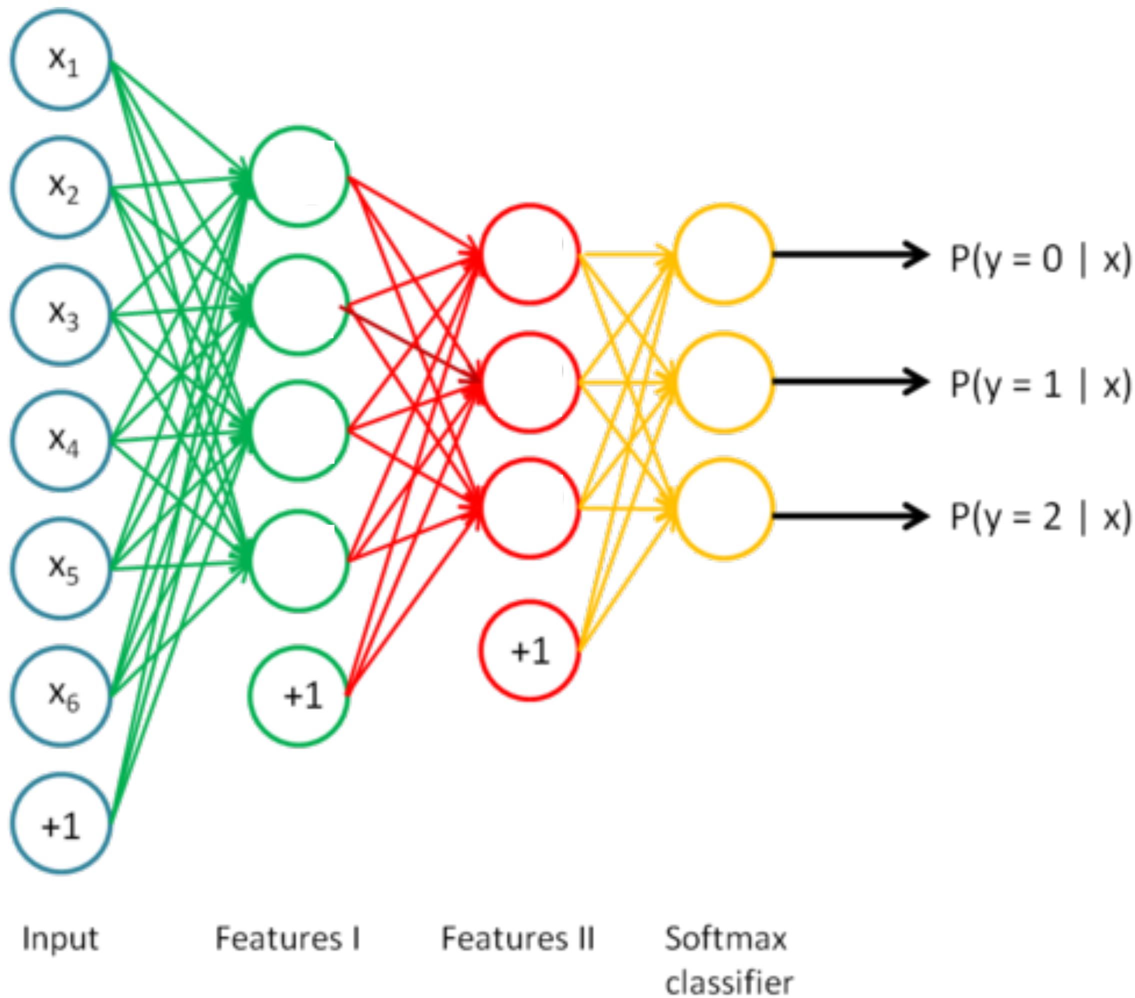


Softmax Example

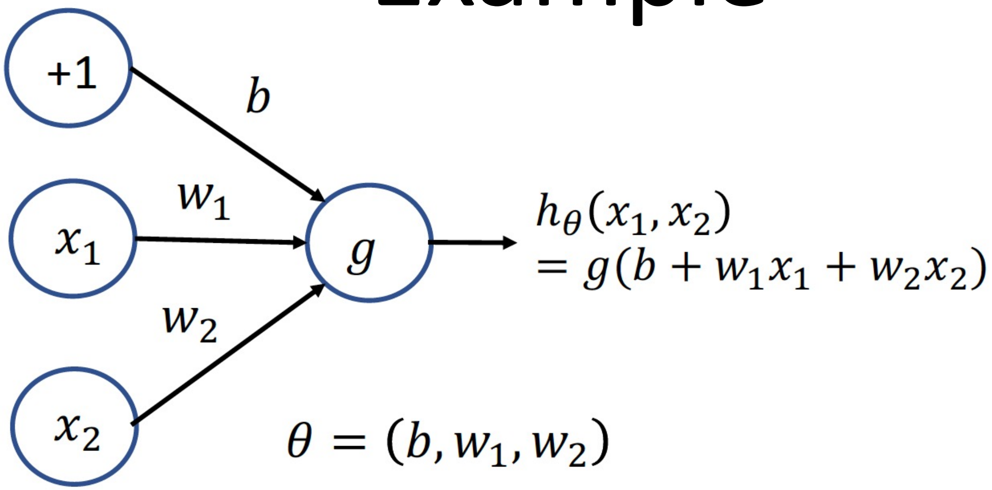


From: https://cs231n.github.io/linear-classify/

Multi-class classification



Example



$\theta = (b, w_1, w_2)$

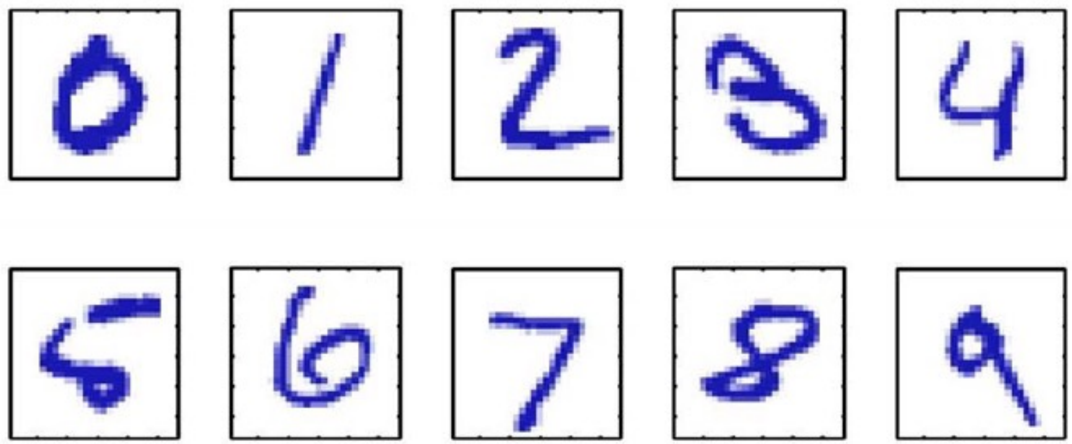
1. Given $b = -10, w_1 = 12, w_2 = 5$
Activation $g(z) = sign(z)$
Compute the output:

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

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,

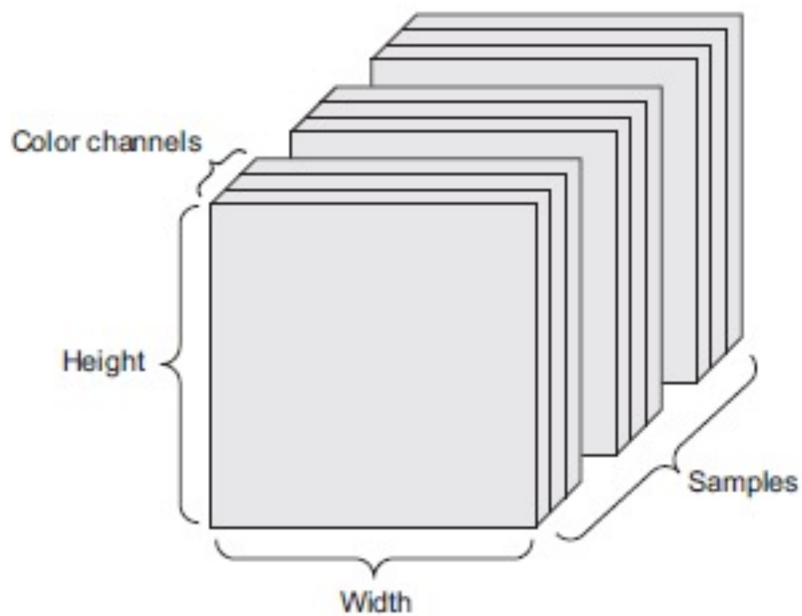
$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

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
```

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

```python
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
```

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

```
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]
```

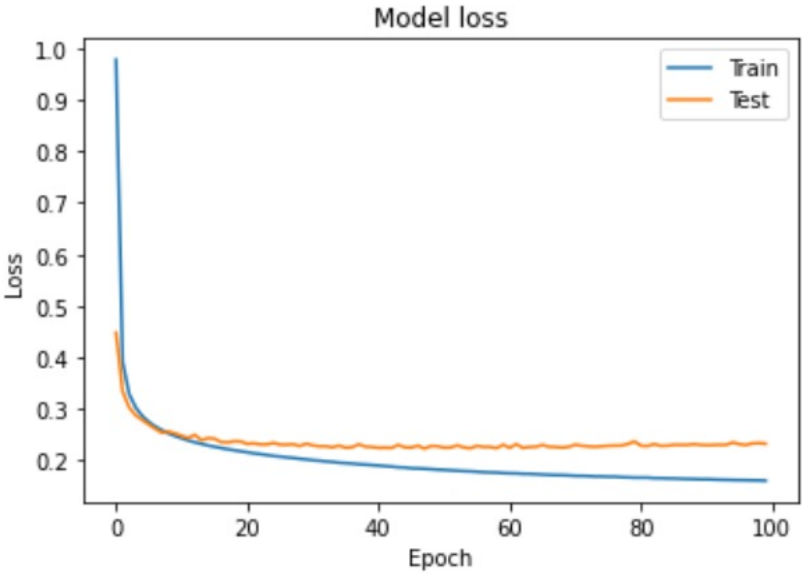
Monitor Loss

```
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)
```



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!