University of Salford MANCHESTER

Artificial Intelligent – Lecture 7

Dr Salem Ameen

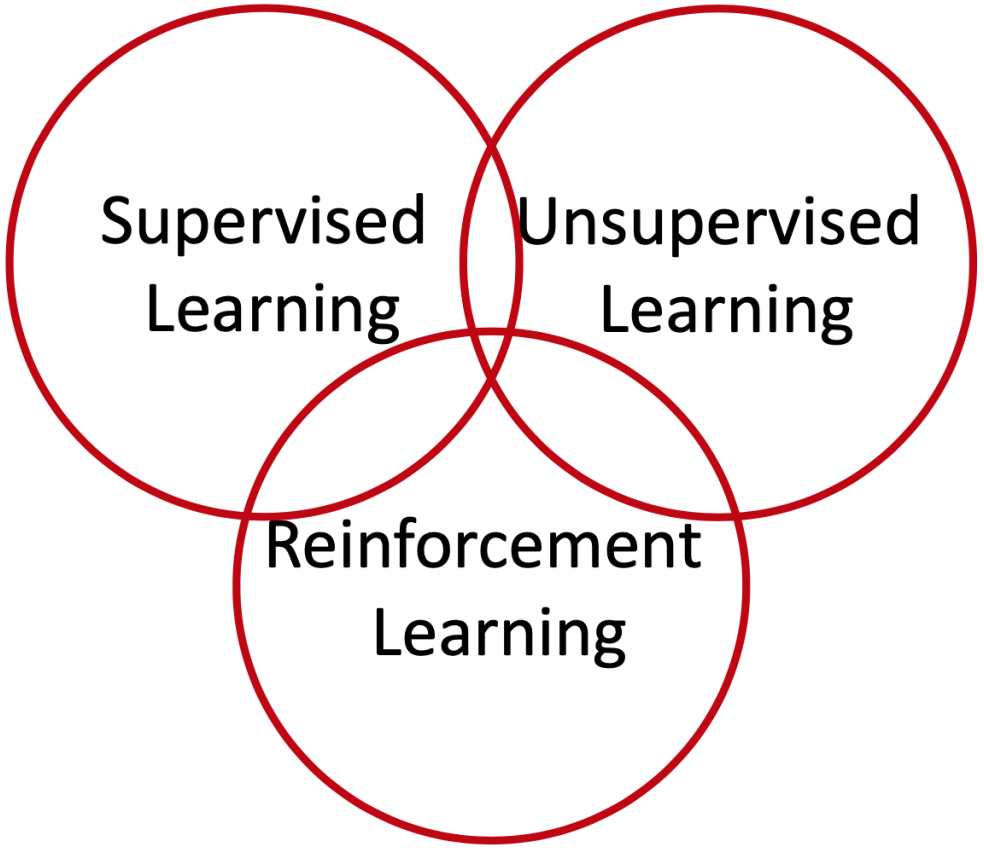
S.A.AMEEN1@SALFORD.AC.UK

Lecture 7

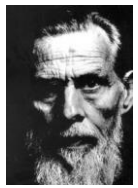
University of Salford MANCHESTER

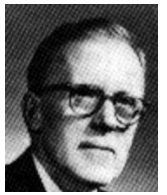
- Machine Learning

Lecture 7 - Machine Learning



Lecture 7 – : Deep Learning
A brief history



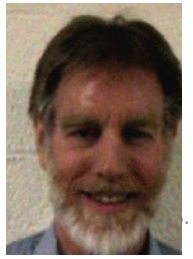


- 1943: Neural networks logical circuits (McCulloch/Pitts)
- 1949: ”Cells that fire together wire together" learning rule (Hebb)

- 1969: Theoretical limitations of neural networks (Minsky/Papert)

- 1974: Backpropagation for training multi-layer networks (Werbos)
- 1986: Popularization of backpropagation (Rumelhardt, Hinton, Williams)





Lecture 7 – : Deep Learning
A brief history

- 1980: Neocognitron, a.k.a. convolutional neural networks (Fukushima)

- 1989: Backpropagation on convolutional neural networks (LeCun)

- 1990: Recurrent neural networks (Elman)

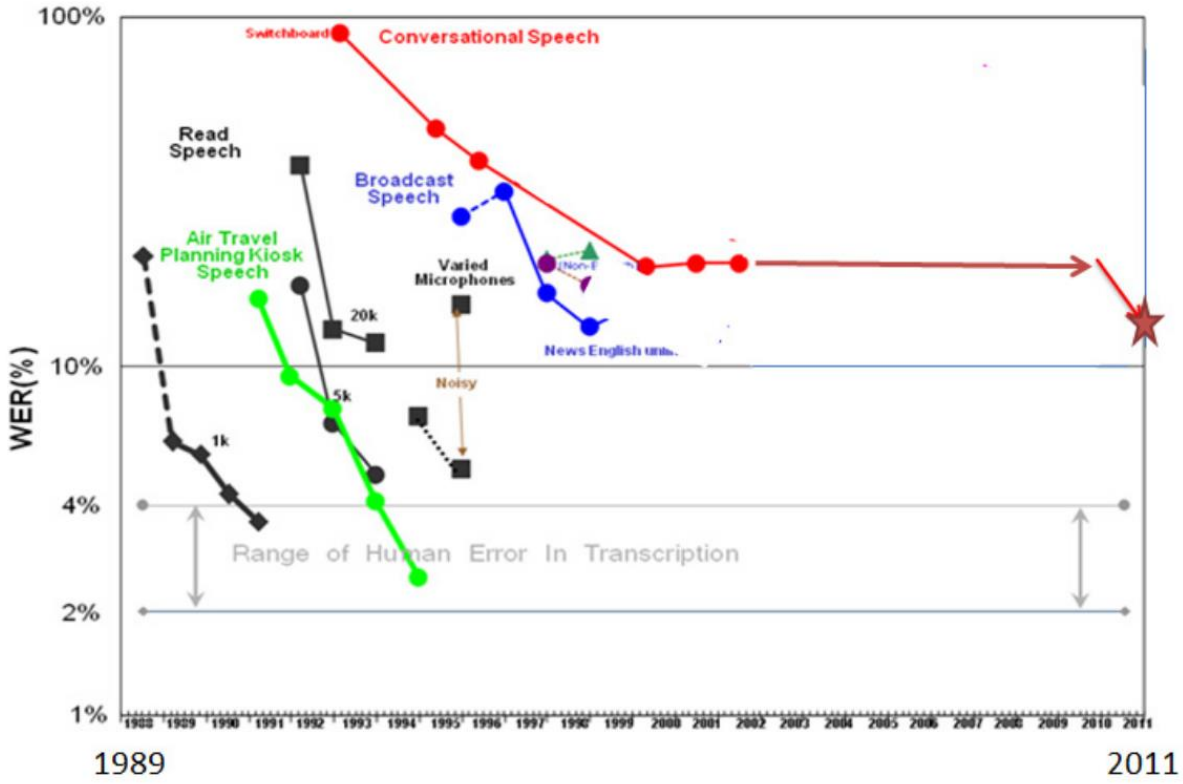
- 1997: Long Short-Term Memory networks (Hochreiter/Schmidhuber)
- 2006: Unsupervised layerwise training of deep networks (Hinton et al.)

Lecture 7 – : Deep Learning
A brief history

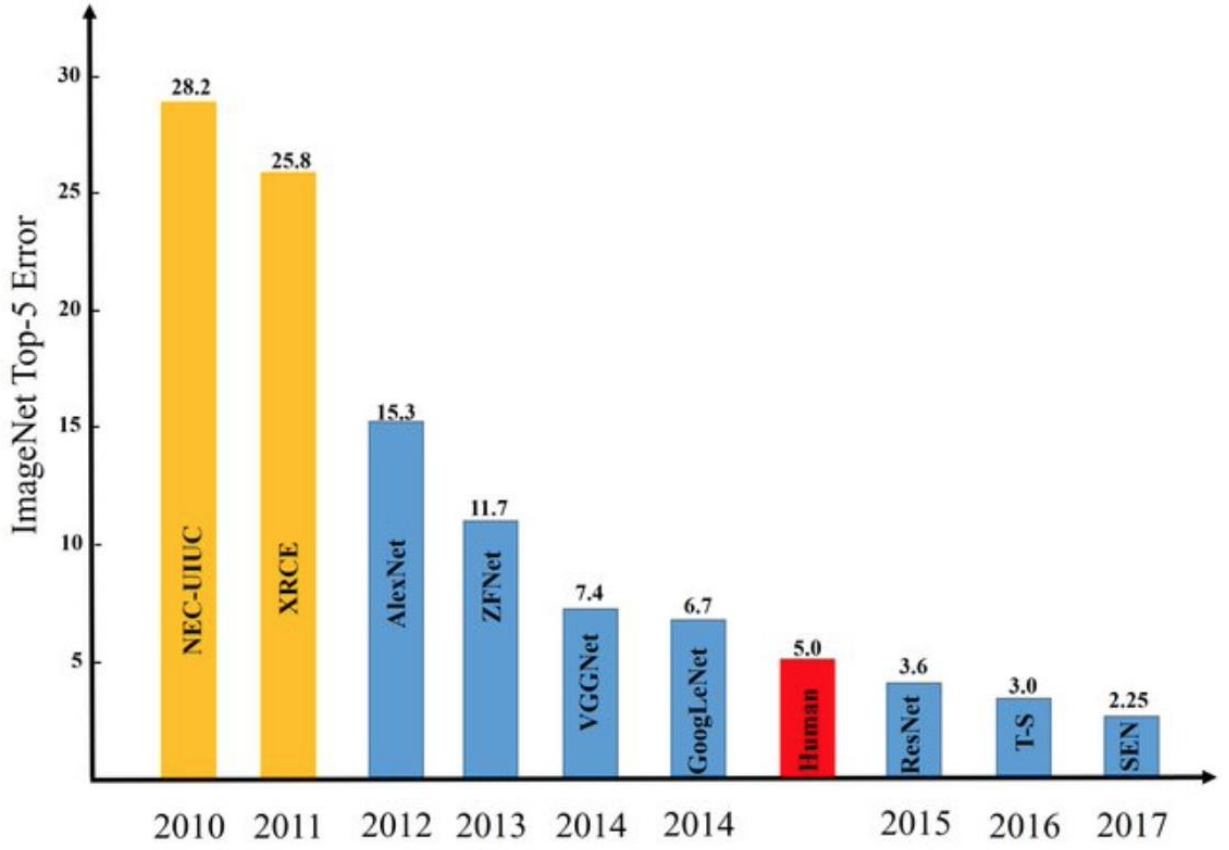
https://www.nytimes.com/2019/03/27/technology/turing-award-ai.html

Lecture 7 – : Deep Learning
A brief history - Google Trends

Lecture 7 – : Deep Learning Speech recognition

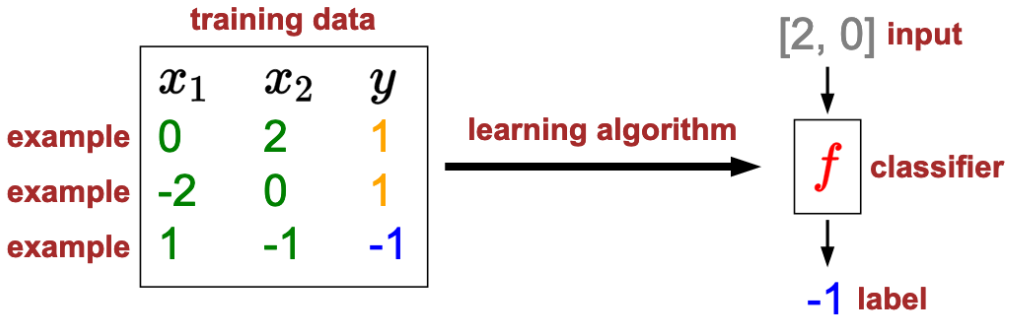


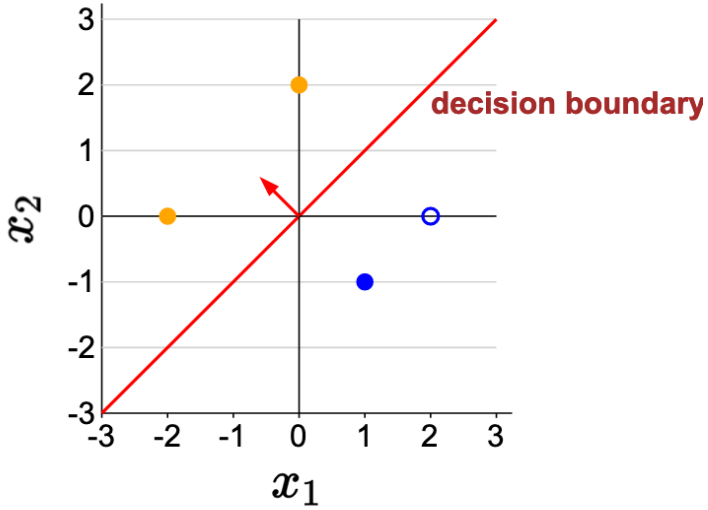
Lecture 7 – : Deep Learning Image Classification



Lecture 7 – : Deep Learning Software – More details later

Lecture 7 – ML
Linear classification framework





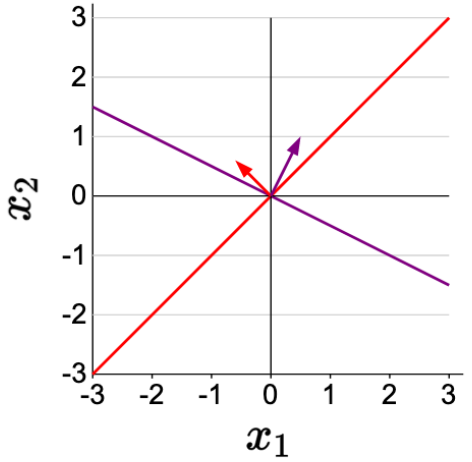
Design decisions:

- Which classifiers are possible? **hypothesis class**
- How good is a classifier? **loss function**
- How do we compute the best classifier? **optimization algorithm**

Lecture 7 – ML
Hypothesis class: which classifiers?

University of Salford MANCHESTER

$$\phi(x) = [x_1, x_2]$$
$$f(x) = \text{sign}([-0.6, 0.6] \cdot \phi(x))$$
$$f(x) = \text{sign}([0.5, 1] \cdot \phi(x))$$



General binary classifier:

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$$

Hypothesis class:

$$\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^2\}$$

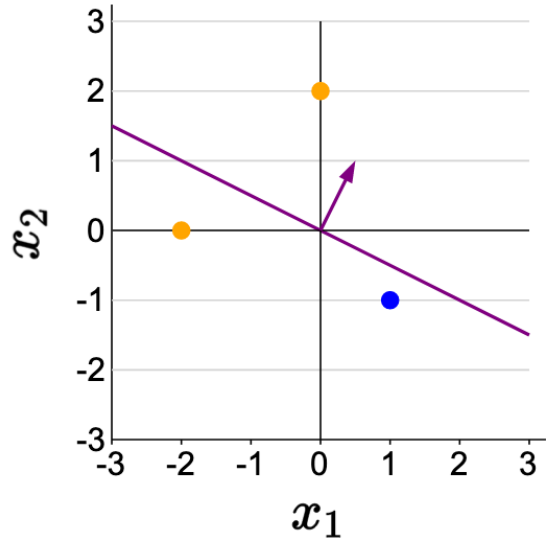
Lecture 7 – ML
Loss function: how good is a classifier?

University of Salford MANCHESTER

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$
$$\mathbf{w} = [0.5, 1]$$
$$\phi(x) = [x_1, x_2]$$

training data $\mathcal{D}_{\text{train}}$

x_1	x_2	y
0	2	1
-2	0	1
1	-1	-1

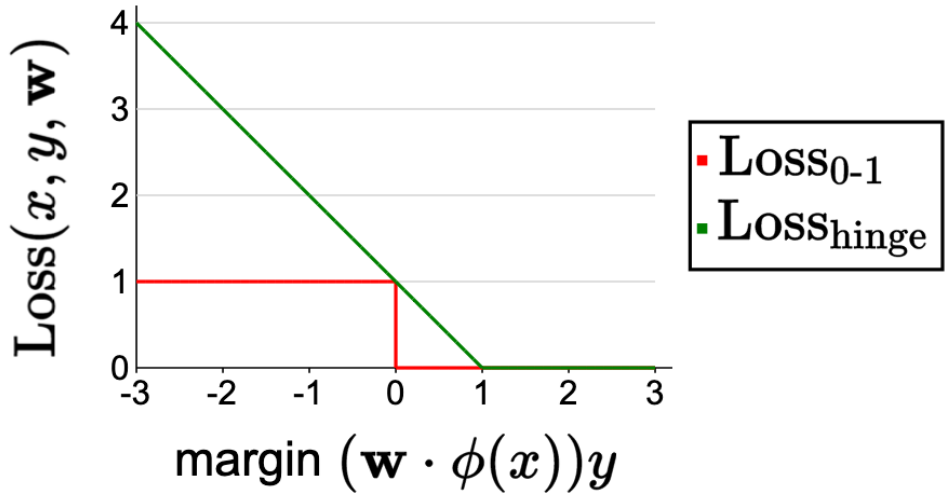


$$\text{Loss}_{\text{0-1}}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$$ zero-one loss

$$\text{Loss}([0, 2], 1, [0.5, 1]) = \mathbf{1}[\text{sign}([0.5, 1] \cdot [0, 2]) \neq 1] = 0$$
$$\text{Loss}([-2, 0], 1, [0.5, 1]) = \mathbf{1}[\text{sign}([0.5, 1] \cdot [-2, 0]) \neq 1] = 1$$
$$\text{Loss}([1, -1], -1, [0.5, 1]) = \mathbf{1}[\text{sign}([0.5, 1] \cdot [1, -1]) \neq -1] = 0$$

$$\text{TrainLoss}([0.5, 1]) = 0.33$$

Lecture 7 – ML
Optimization algorithm: how to compute best?
Hinge loss



$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$

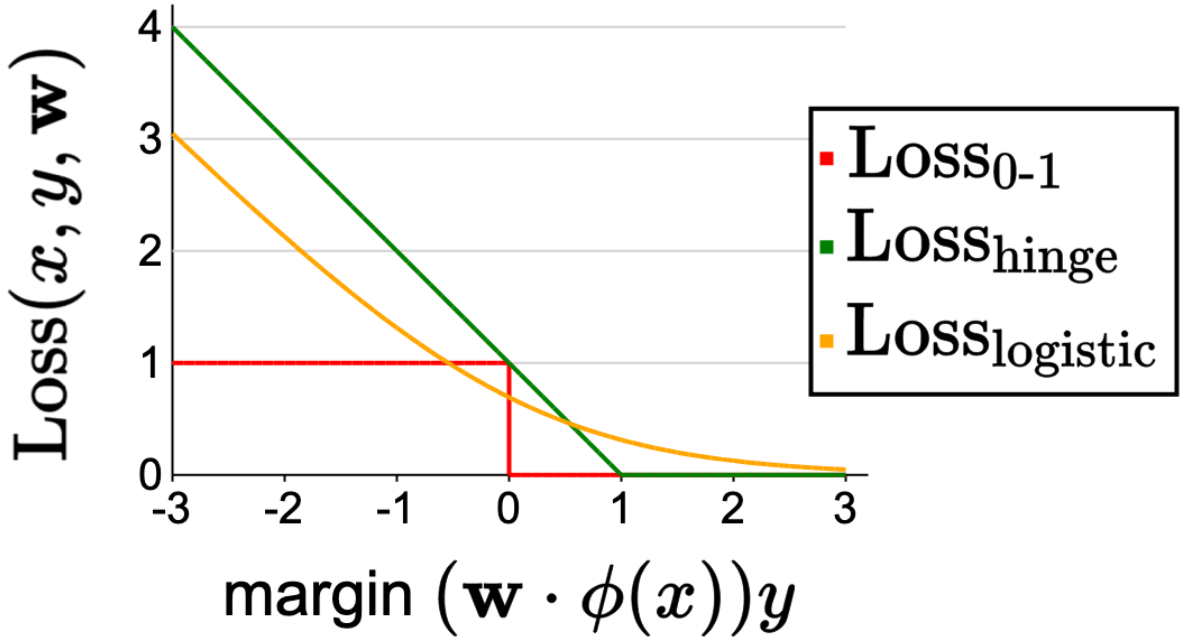
A point has zero loss if it is confidently classified correctly (margin > 1)

Confidently misclassifying a point incurs a linear penalty

Lecture 7 – ML
Optimization algorithm: how to compute best? logistic regression (Softmax)

University of Salford MANCHESTER

$$\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-(\mathbf{w}\cdot\phi(x))y})$$

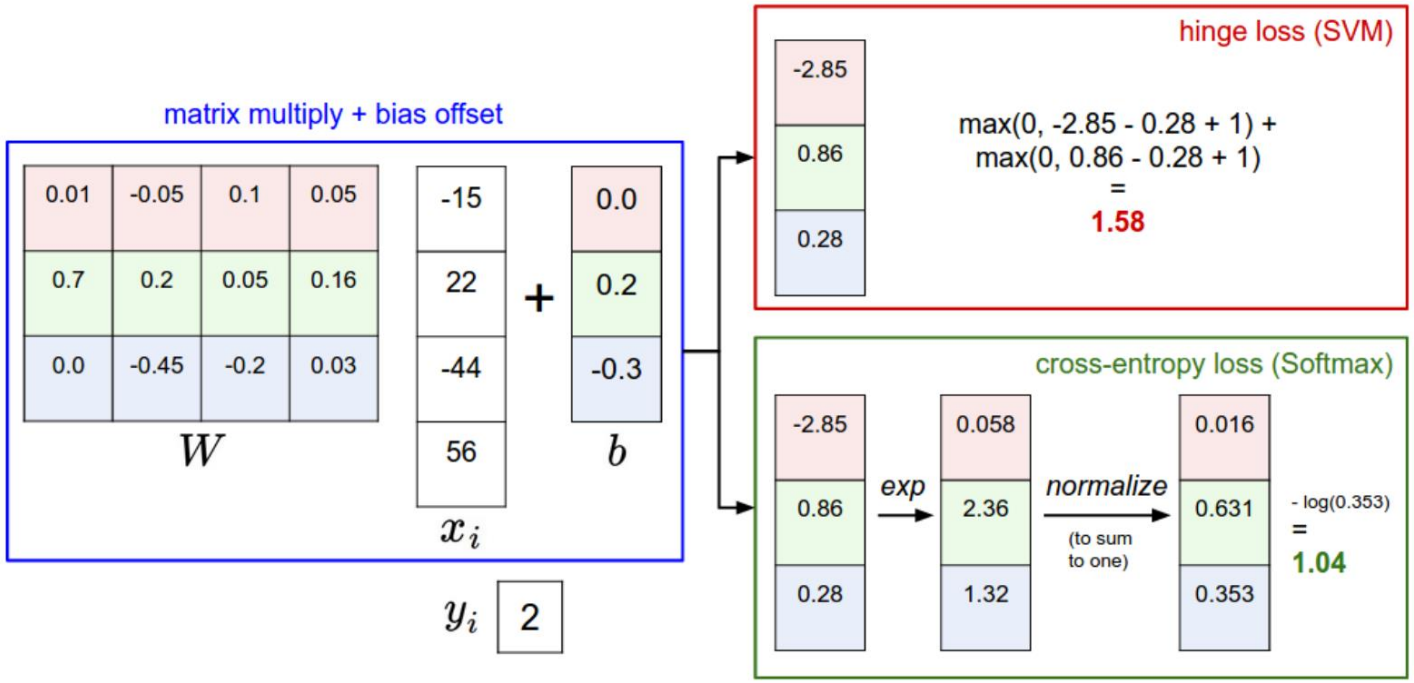


Intuition: Try to increase margin even when it already exceeds 1

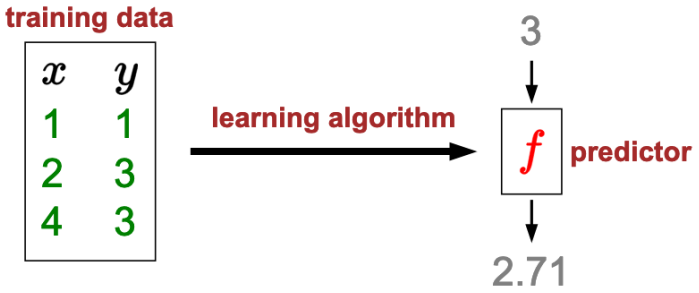
Lecture 7 – ML
Optimization algorithm: how to compute best?
SVM vs. Softmax

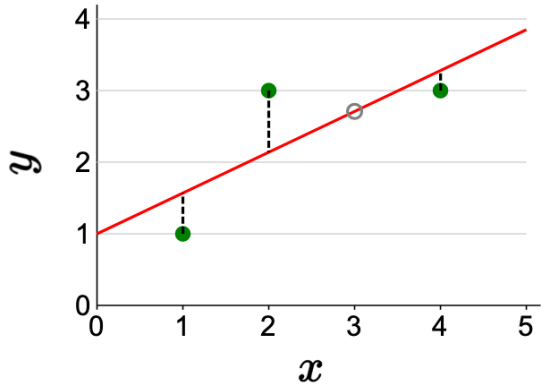
University of Salford MANCHESTER

Correct class (class 2)



Lecture 7 – ML
Optimization algorithm: how to compute best?
Linear regression





Which predictors are possible?
Hypothesis class

Linear functions
$\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)\}, \phi(x) = [1, x]$

How good is a predictor?
Loss function

Squared loss
$\text{Loss}(x, y, \mathbf{w}) = (f_{\mathbf{w}}(x) - y)^2$

How to compute best predictor?
Optimization algorithm

Gradient descent
$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \text{TrainLoss}(\mathbf{w})$

Lecture 7 – ML Optimization algorithm: Loss Function- Code

University of Salford MANCHESTER

import torch

- LSE (Regression)
  - loss_fn = torch.nn.MSELoss()
- LogSoftmax (Classification)
  - loss_fn = torch.nn. LogSoftmax()
- CrossEntropyLoss(Classification)
  - loss_fn = torch.nn. CrossEntropyLoss()
- MultiLabelMarginLoss(Classification)
  - loss_fn = torch.nn. MultiLabelMarginLoss()

import tensorflow as tf

- LSE (Regression)
  - loss_fn = tf.keras. MeanSquaredError()
- LogSoftmax (Classification)
  - loss_fn = tf.nn. log_softmax()
- CrossEntropyLoss(Classification)
  - loss_fn = tf.keras. CategoricalCrossentropy()
- Hinge(Classification)
  - loss_fn = tf.keras. Hinge()

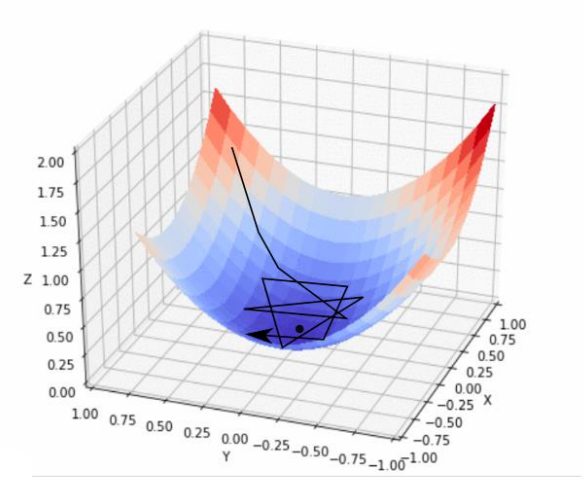
Lecture 7 – ML - Linear Model Optimization algorithm: SGD

University of Salford MANCHESTER

Goal: $\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Definition: gradient

The gradient $\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$ is the direction that increases the training loss the most.



Algorithm: stochastic gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

  For $(x, y) \in \mathcal{D}_{\text{train}}$:

    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$

Lecture 7 – ML - Optimization algorithm: SGD

University of Salford MANCHESTER

Gradient Descent

1. The gradient is evaluated.
2. A small step is made in the direction of the negative gradient, the parameters are updated.

Batch Gradient Descent

- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over all training data set n

  $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{n}\sum_{i=1}^{n} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the direction $\delta\theta = \lambda\nabla_\theta g(\theta)$ where $\lambda \in R^+$ (positive real number) is learning rate or step size
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

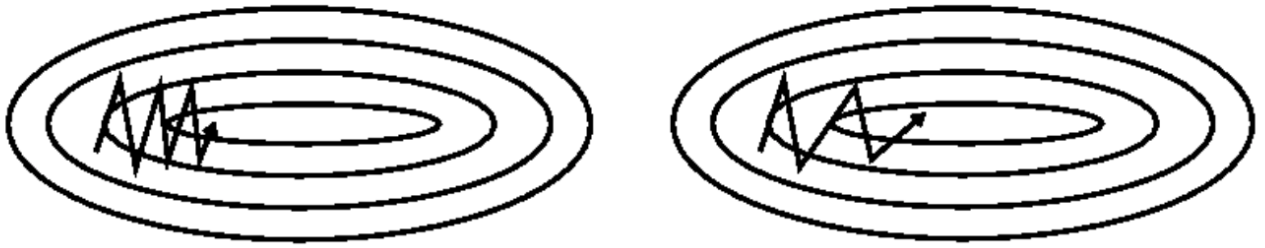
http://usir.salford.ac.uk/id/eprint/45018/1/Thesis_AFTER_viv3.pdf

Lecture 7 – ML - Optimization algorithm:

University of Salford MANCHESTER

Gradient Descent with Momentum

- Sample a minibatch of m examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over m sampling of training data set n $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the update direction $\delta\theta = v$ where $v_{i+1} = \gamma v_i + \lambda \nabla_\theta g(\theta)$ and $\gamma \in R$ (real number and practically set to 0.9) and is called momentum.
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$



(a) GD without momentum

(b) GD with momentum

http://usir.salford.ac.uk/id/eprint/45018/1/Thesis_AFTER_viv3.pdf

Lecture 7 – ML - Optimization algorithm:

University of Salford MANCHESTER

Adagrad

- Sample a minibatch of m examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over m sampling of training data set n $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta)$ where $r_{i+1} = r_i + \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$.
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

http://usir.salford.ac.uk/id/eprint/45018/1/Thesis_AFTER_viv3.pdf

Lecture 7 – ML - Optimization algorithm:

University of Salford MANCHESTER

RMSProp

- Sample a minibatch of m examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over m sampling of training data set n $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta)$ where $r_{i+1} = \rho r_i + (1-\rho)\nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$.
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

http://usir.salford.ac.uk/id/eprint/45018/1/Thesis_AFTER_viv3.pdf

Lecture 7 – ML - Optimization algorithm:

University of Salford MANCHESTER

Adam

- Sample a minibatch of m examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over m sampling of training data set n $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m}$, where δ is a small number (e.g. $1e^{-8}$) where $\hat{m} = \frac{m_g}{1-\beta_1}$ and $\hat{v} = \frac{v_g}{1-\beta_2}$
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

http://usir.salford.ac.uk/id/eprint/45018/1/Thesis_AFTER_viv3.pdf

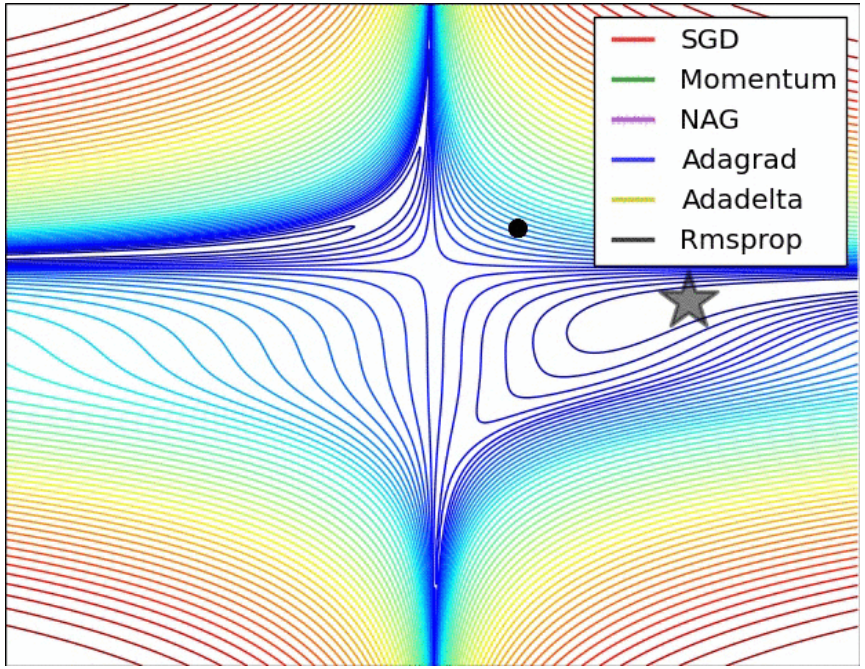
Lecture 7 – ML - Optimization algorithm:

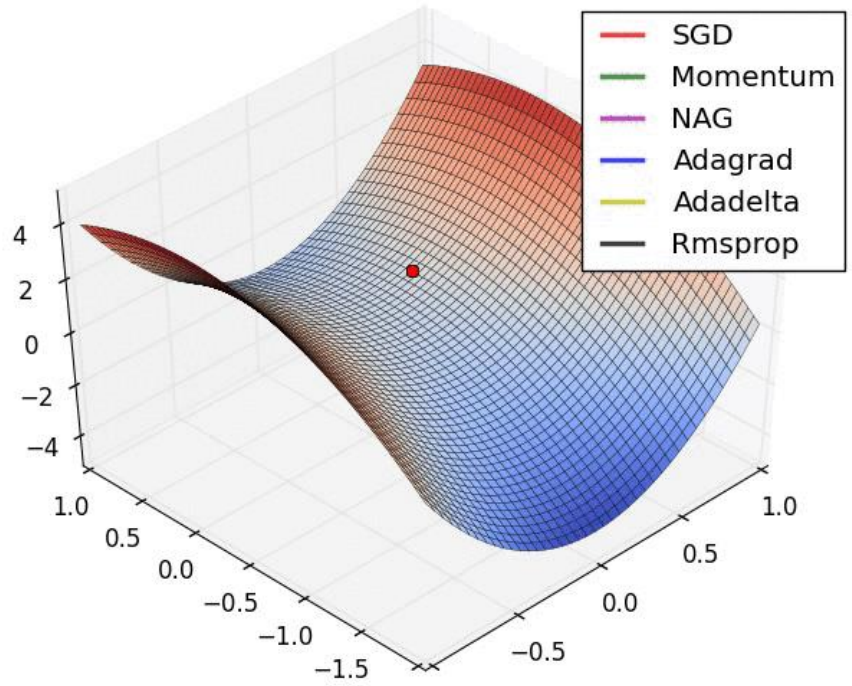
SGD Adagrad

RMSProp

SGD + Momentum

Adam





from sklearn.preprocessing import StandardScaler

Lecture 7 – ML - Optimization algorithm: Code - PyTorch

University of Salford MANCHESTER

- import torch
  - SGD
    - optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
  - SGD + momentum
    - optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)
  - ADAM
    - optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
  - Adagrad
    - optimizer = torch.optim.Adagrad(model.parameters(), lr=learning_rate)
  - RMSprop
    - optimizer = torch.optim.RMSprop(model.parameters(), lr=learning_rate)

Lecture 7 – ML - Optimization algorithm:Code-Tensorflow

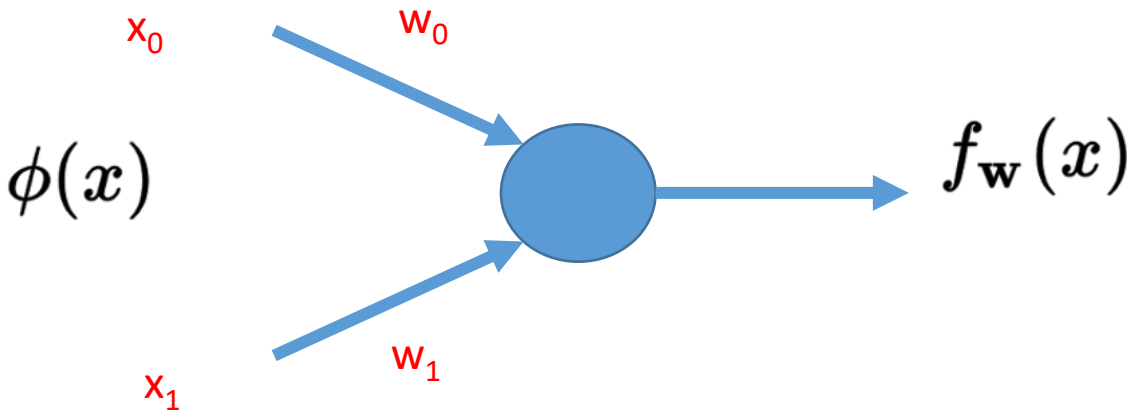
University of Salford MANCHESTER

- import tensorflow as tf
  - SGD
    - optimizer = tf.keras.optimizers.SGD(lr=learning_rate)
  - SGD + momentum
    - optimizer = tf.keras.optimizers.SGD(lr=learning_rate, momentum=0.9)
  - ADAM
    - optimizer = tf.keras.optimizers.Adam(lr=learning_rate)
  - Adagrad
    - optimizer = tf.keras.optimizers.Adagrad(lr=learning_rate)
  - RMSprop
    - optimizer = tf.keras.optimizers.RMSprop(lr=learning_rate)

Lecture 7 – ML - Optimization algorithm: Example

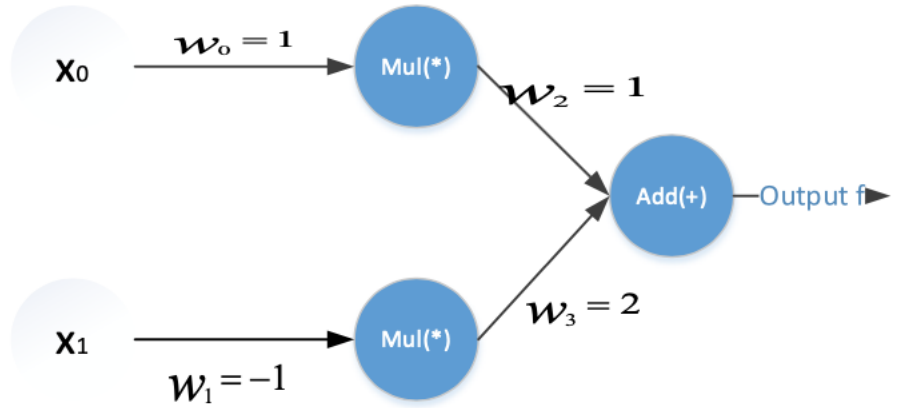
weight vector $\mathbf{w} = [w_1, w_2]$ feature extractor $\phi(x) = [1, x]$ feature vector

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) \text{ score}$$

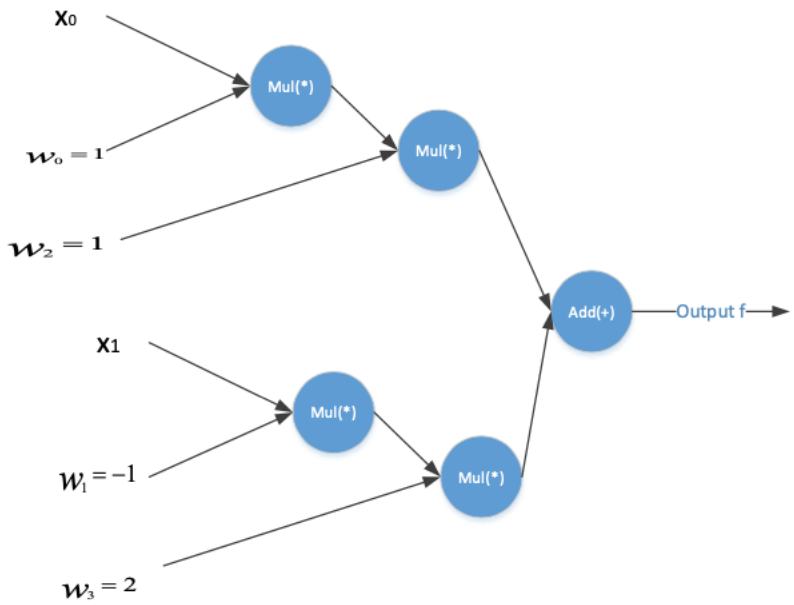


Lecture 7 – ML - Optimization algorithm:

X_0	X_1	y
0	1	1
2	1	4
1	0	2
1	1	1



(a) Graph with 3 nodes, external inputs, and outputs



(b) The graph unfolded to explain the operation

Lecture 7 – ML - Code:

University of Salford MANCHESTER

```
import numpy as np
import torch
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
```

Lecture 7 – ML - Code:


```
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')

inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)

# Define the model
model = nn.Linear(2, 1)
```

Lecture 7 – : Score (Forward Propagation)

University of Salford MANCHESTER

$$f_1 = x_0 w_0$$

X_0	X_1	y
1	1	1

$$f_2 = x_1 w_1$$

$$f_3 = f_1 w_2$$

$$f_4 = f_2 w_3$$

$$f = f_3 + f_4$$

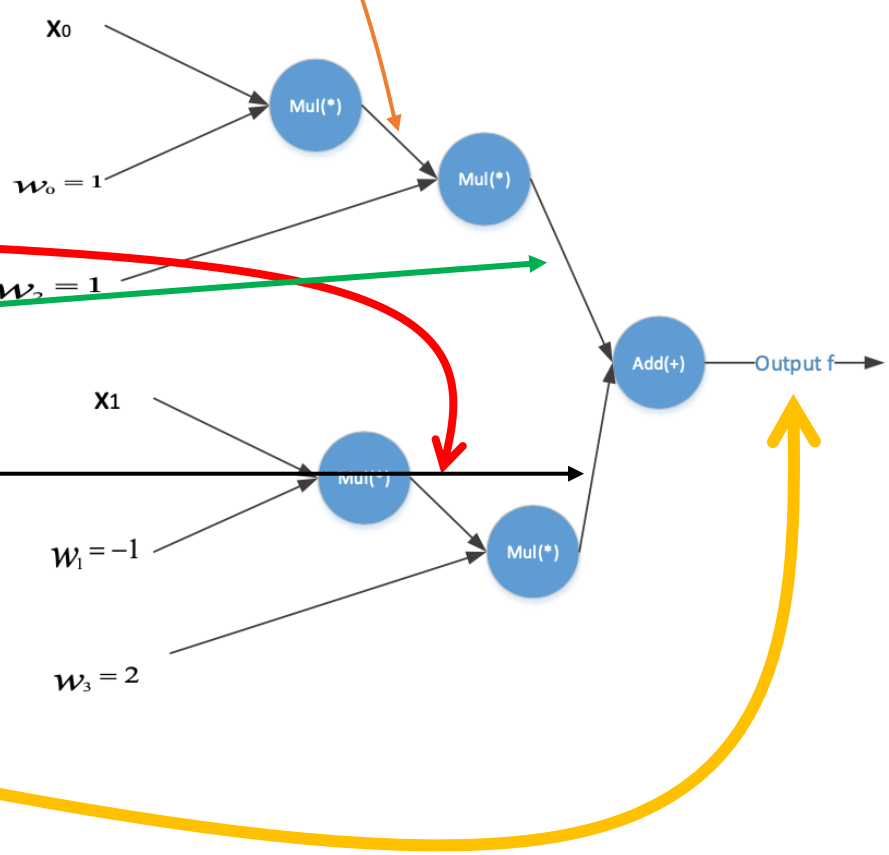
$$f_1 = 1 \times 1 = 1$$

$$f_2 = 1 \times -1 = -1$$

$$f_3 = 1 \times 1 = 1$$

$$f_4 = -1 \times 2 = -2$$

$$f = 1 + (-2) = -1$$



Lecture 7 – ML - Code:

University of Salford MANCHESTER

```
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')

inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)

# Define the model
model = nn.Linear(2, 1)

# Define optimizer
opt = torch.optim.SGD(model.parameters(), lr=1e-5)

# Define loss function
loss = F.mse_loss(model(inputs), targets)
# Start Training
num_epochs = 100
for epoch in range(num_epochs):
    for i in range(len(inputs)):
        # Generate predictions
        pred = model(inputs)
```

Lecture 7 – : Score (Loss Function)

Cost !!!!

University of Salford MANCHESTER

X_0	X_1	y
1	1	1

$$f_1 = x_0 w_0$$

$$f_2 = x_1 w_1$$

$$f_3 = f_1 w_2$$

$$f_4 = f_2 w_3$$

$$f = f_3 + f_4$$

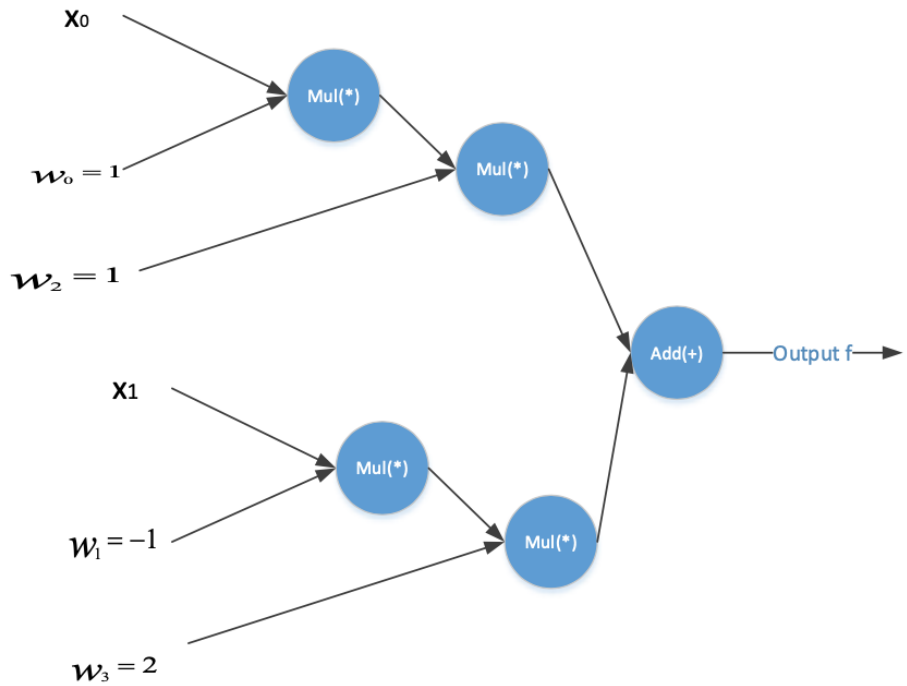
$$f_1 = 1 \times 1 = 1$$

$$f_2 = 1 \times -1 = -1$$

$$f_3 = 1 \times 1 = 1$$

$$f_4 = -1 \times 2 = -2$$

$$f = 1 + (-2) = -1$$



$$L = \frac{1}{2}(y - f)^2 = \frac{1}{2}(1 - (-1))^2 = 2$$

Lecture 7 – ML - Code:


```
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')

inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)

# Define the model
model = nn.Linear(2, 1)

# Define optimizer
opt = torch.optim.SGD(model.parameters(), lr=1e-5)

# Define loss function
loss = F.mse_loss(model(inputs), targets)
# Start Training
num_epochs = 100
for epoch in range(num_epochs):
    for i in range(len(inputs)):
        # Generate predictions
        pred = model(inputs)
        loss = loss_fn(pred, yb)
```

Lecture 7 – : (derivative)

University of Salford MANCHESTER

X_0	X_1	y
1	1	1

$$f_1 = x_0 w_0, \frac{\delta f_1}{dx_0} = w_0, \frac{\delta f_1}{\delta w_0} = x_0$$

$$f_2 = x_1 w_1, \frac{\delta f_2}{dx_1} = w_0, \frac{\delta f_2}{\delta w_1} = x_1$$

$$f_3 = f_1 w_2, \frac{\delta f_3}{\delta f_1} = w_2, \frac{\delta f_3}{\delta w_2} = f_1$$

$$f_4 = f_2 w_3, \frac{\delta f_4}{\delta f_2} = w_3, \frac{\delta f_4}{\delta w_3} = f_2$$

$$f = f_3 + f_4, \frac{\delta f}{\delta f_3} = 1, \frac{\delta f}{\delta f_4} = 1$$

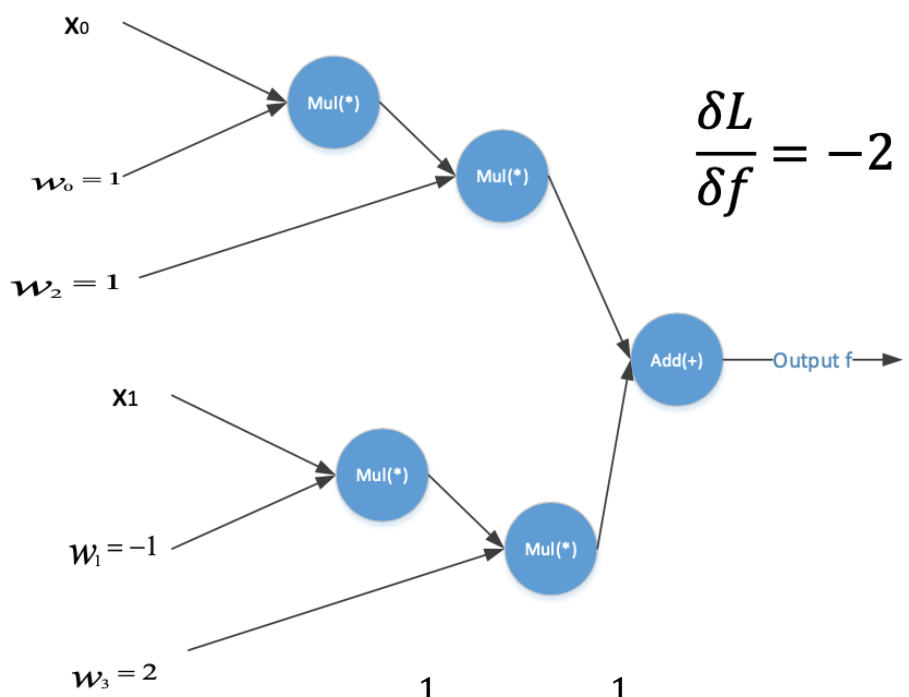
$f_1 = 1\times1 = 1$

$f_2 = 1\times-1 = -1$

$f_3 = 1\times1 = 1$

$f_4 = -1\times2 = -2$

$f = 1 + (-2) = -1$



$$\frac{\delta L}{\delta f} = -2$$

$$L = \frac{1}{2}(y - f)^2 = \frac{1}{2}(1 - (-1))^2 = 2$$

Lecture 7 – : (Chain Rule)(Backward Propagation)

University of Salford MANCHESTER

Computing the Jacobian matrix

$$\frac{\delta L}{\delta f} = -2$$

$$\frac{\delta L}{\delta f_4} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta f_3} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta f_1} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3}\frac{\delta f_3}{\delta f_1} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3}\frac{\delta f_3}{\delta w_2} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta w_0} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3}\frac{\delta f_3}{\delta f_1}\frac{\delta f_1}{\delta w_0} = -2 * 1 * 1 = -2$$

$$\frac{\delta L}{\delta f_2} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4}\frac{\delta f_4}{\delta f_2} = -2 * 2 = -4$$

$$\frac{\delta L}{\delta w_3} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4}\frac{\delta f_4}{\delta w_3} = -2 * (-1) = 2$$

$$\frac{\delta L}{\delta w_1} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4}\frac{\delta f_4}{\delta f_2}\frac{\delta f_2}{\delta w_1} = -2 * 2 * 1 = -4$$

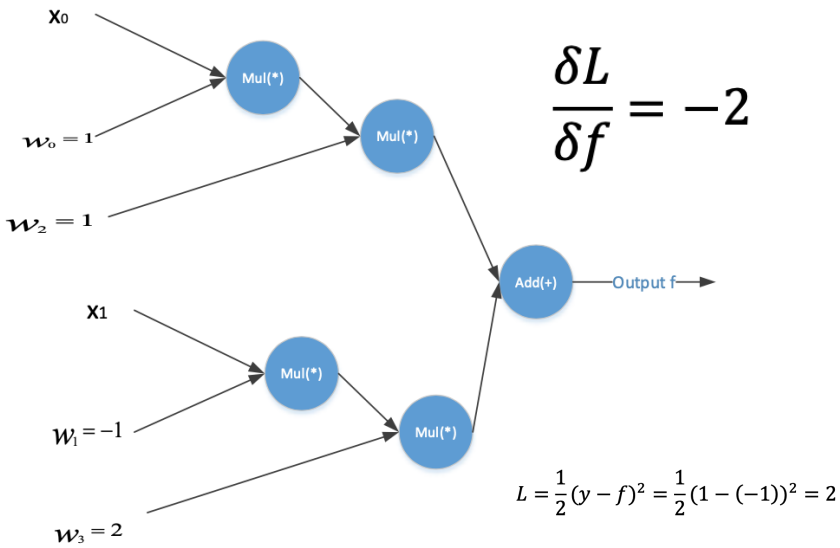
$$f_1 = 1\times 1 = 1$$

$$f_2 = 1\times -1 = -1$$

$$f_3 = 1\times 1 = 1$$

$$f_4 = -1\times 2 = -2$$

$$f = 1 + (-2) = -1$$



$$f_1 = x_0 w_0, \frac{\delta f_1}{dx_0} = w_0, \frac{\delta f_1}{\delta w_0} = x_0$$

$$f_2 = x_1 w_1, \frac{\delta f_2}{dx_1} = w_0, \frac{\delta f_2}{\delta w_1} = x_1$$

$$f_3 = f_1 w_2, \frac{\delta f_3}{\delta f_1} = w_2, \frac{\delta f_3}{\delta w_2} = f_1$$

$$f_4 = f_2 w_3, \frac{\delta f_4}{\delta f_2} = w_3, \frac{\delta f_4}{\delta w_3} = f_2$$

$$f = f_3 + f_4, \frac{\delta f}{\delta f_3} = 1, \frac{\delta f}{\delta f_4} = 1$$

Lecture 7 – ML - Code:


```
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')

inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)

# Define the model
model = nn.Linear(2, 1)

# Define optimizer
opt = torch.optim.SGD(model.parameters(), lr=1e-5)

# Define loss function
loss = F.mse_loss(model(inputs), targets)
# Start Training
num_epochs = 100
for epoch in range(num_epochs):
    for i in range(len(inputs)):
        # Generate predictions
        pred = model(inputs)
        loss = loss_fn(pred, yb)
        # Perform gradient descent
        opt.zero_grad()
        loss.backward()
        opt.step()

print('Training loss: ', loss_fn(model(inputs), targets))
# Prediction
testing = np.array([[2, 0]], dtype='float32')
x_test = torch.from_numpy(testing)
pred_test = model(x_test)
print("Prediction = ", pred_test)
```

Lecture 7 – ML - Code: TensorFlow

University of Salford MANCHESTER

```
import tensorflow as tf
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')


model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(2),
  tf.keras.layers.Dense(1)
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')

model.fit(inputs, targets, epochs=5)
model.predict(np.array([[2,0]]))
```

Lecture 7 – : BP/Optimization algorithm

University of Salford MANCHESTER

□ **Backpropagation** — Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight w is computed using chain rule and is of the following form:

$$\boxed{\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}}$$

As a result, the weight is updated as follows:

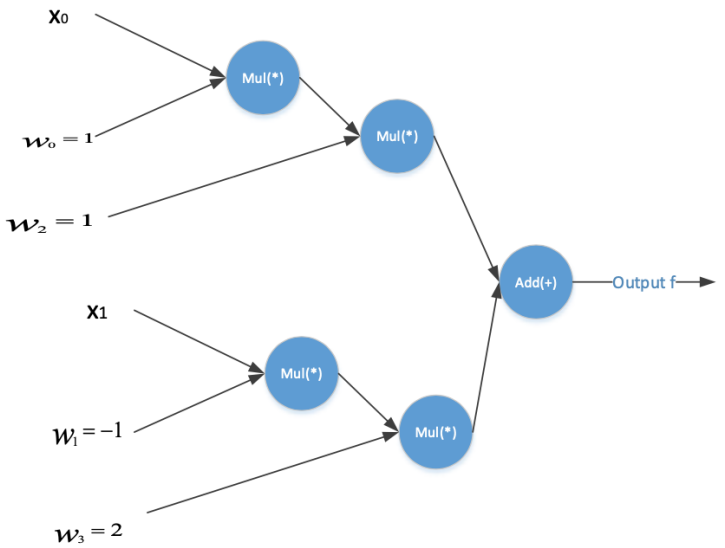
$$\boxed{w \longleftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}}$$

Lecture 7 – : BP/Optimization algorithm

University of Salford MANCHESTER

Computing the Jacobian matrix

$$\nabla_\theta g(\theta) = \nabla_w g(w) = \begin{bmatrix} \frac{\delta L}{\delta w_0} & \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} & \frac{\delta L}{\delta w_3} \end{bmatrix} = \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix}$$

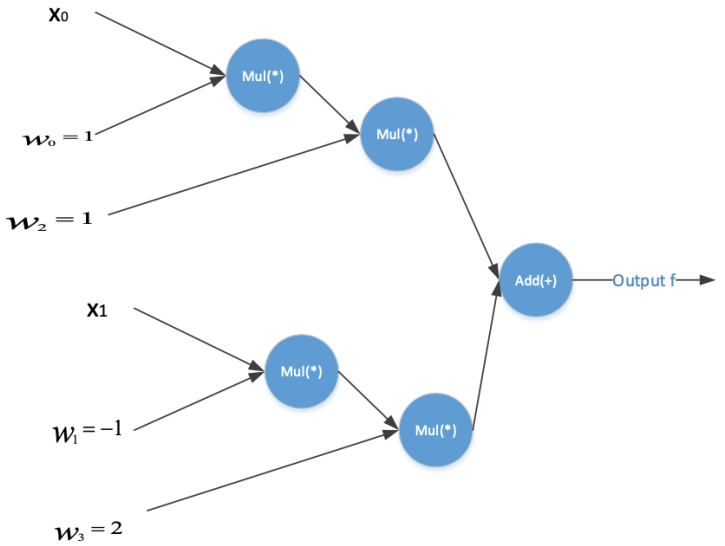


Lecture 7 – : Optimization algorithm

University of Salford MANCHESTER

SGD

$$\delta w_{i+1} = \delta\theta_{i+1} = \lambda \nabla_{w,i}\, g(\theta) = 0.1 * \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} -0.2 & -0.4 \\ -0.2 & 0.2 \end{bmatrix}$$



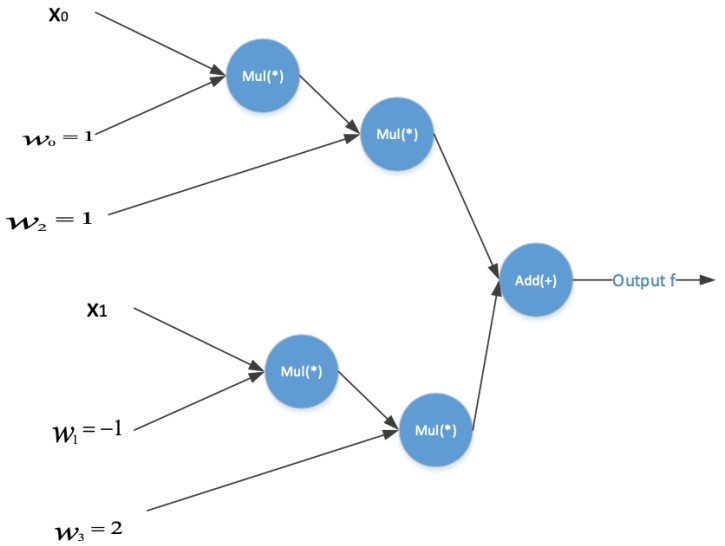
$$w_{i+1} = \theta_{i+1} = \theta_i - \delta\theta_i = w_i - \delta w_i = \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} -0.2 & -0.4 \\ -0.2 & 0.2 \end{bmatrix} = \begin{bmatrix} 1.2 & -0.6 \\ 1.2 & 1.8 \end{bmatrix}$$

Lecture 7 – : Optimization algorithm

University of Salford MANCHESTER

SGD + Momentum

$$Vi{+}1 = \gamma Vi + \lambda \nabla w g(\theta)$$



Lecture 7 – : Optimization algorithm

University of Salford MANCHESTER

Adagrad

The following GD is Adagrad, we compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta)$ where

$r_{i+1} = r_i + \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$ and r initializes at 0 then

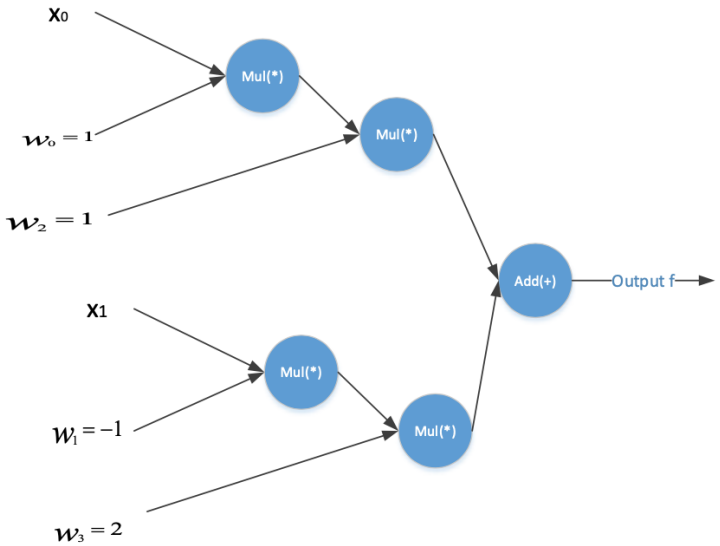
$$r_{i+1} = r_i + \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta) = 0 + \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 16 \\ 4 & 4 \end{bmatrix}$$

Then

$$\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta) = \delta\theta = \frac{0.1}{0.00005+\sqrt{\begin{bmatrix} 4 & 16 \\ 4 & 4 \end{bmatrix}}} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} -0.025 & -0.006 \\ -0.025 & 0.025 \end{bmatrix}$$

Then, perform a parameter update

$$\theta_{i+1} = \theta_i - \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_{\theta,i} g(\theta) = \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} -0.025 & -0.006 \\ -0.025 & 0.025 \end{bmatrix} = \begin{bmatrix} 1.025 & -0.99 \\ 1.025 & 1.98 \end{bmatrix}$$



Lecture 7 – : Optimization algorithm

University of Salford MANCHESTER

RMSProp

The next GD is RMSProp, we compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta)$ where

$r_{i+1} = \rho r_i + (1-\rho)\nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$ and r_i initializes at 0 then

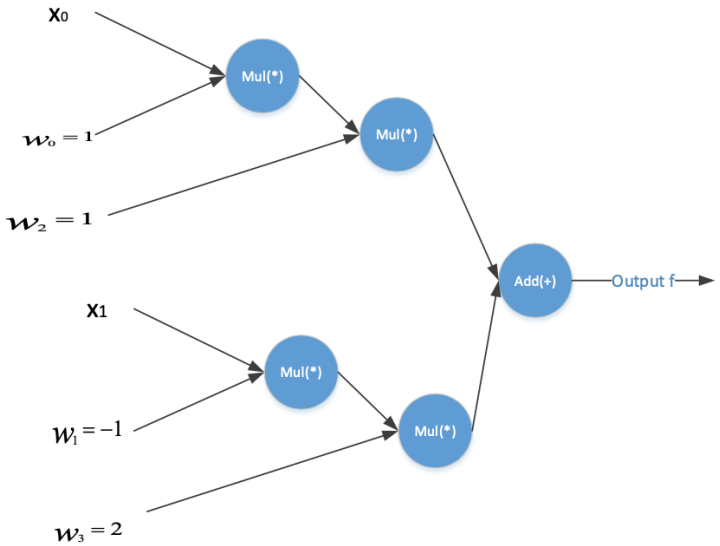
$r_{i+1} = \rho r_i + (1-\rho)\nabla_\theta g(\theta) \odot \nabla_\theta g(\theta) = 0.99 * 0 + (1-0.99) *$

$\begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 0.04 & 0.16 \\ 0.04 & 0.04 \end{bmatrix}$ Then

$$\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta) = \frac{0.1}{0.00005 + \sqrt{\begin{bmatrix} 0.04 & 0.16 \\ 0.04 & 0.04 \end{bmatrix}}} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} -0.249 & -0.063 \\ -0.249 & 0.250 \end{bmatrix}$$

Then, perform a parameter update

$$\theta_{i+1} = \theta_i - \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta) = \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} -0.249 & -0.063 \\ -0.249 & 0.250 \end{bmatrix} = \begin{bmatrix} 1.25 & -0.94 \\ 1.25 & 1.75 \end{bmatrix}$$



Lecture 7 – : Optimization algorithm

University of Salford MANCHESTER

Adam

The last GD is Adam, we compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m}$, where δ is a small number where $\hat{m} = \frac{m_g}{1-\beta_1}$ and $\hat{v} = \frac{v_g}{1-\beta_2}$, $m_{g,i+1} = \beta_1 m_{g,i} + (1-\beta_1)\nabla_\theta g(\theta)$ and $v_{g,i+1} = \beta_2 v_{g,i} + (1-\beta_2)(\nabla_\theta g(\theta))^2$

$$m_{g,i+1} = \beta_1 m_{g,i} + (1-\beta_1)\nabla_\theta g(\theta) = 0.9 * 0 + (1-0.9)\begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix} = \begin{bmatrix}-0.2 & -0.4\\ -0.2 & 0.2\end{bmatrix}$$

$$v_{g,i+1} = \beta_2 v_{g,i} + (1-\beta_2)(\nabla_\theta g(\theta))^2 = 0.999 * 0 + (1-0.999)(\begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix})^2$$

$$= \begin{bmatrix}0.004 & 0.016\\ 0.004 & 0.004\end{bmatrix}$$

$$\hat{m} = \frac{\begin{bmatrix}-0.2 & -0.4\\ -0.2 & 0.2\end{bmatrix}}{1-0.9} = \begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix}$$

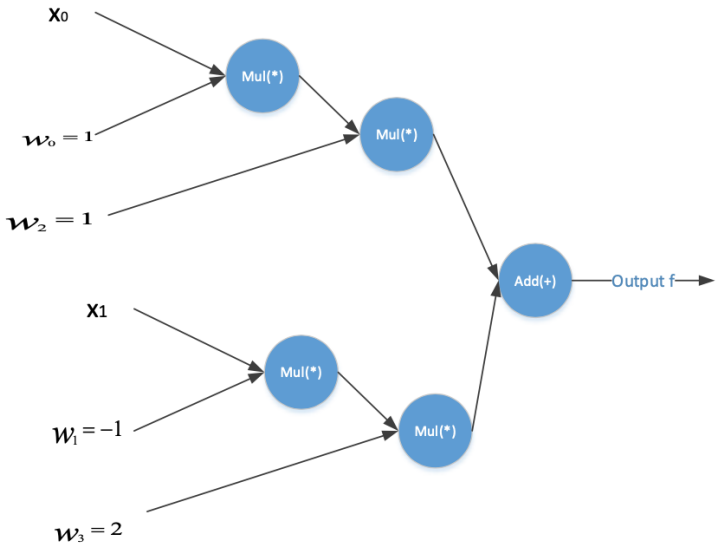
$$\hat{v} = \frac{\begin{bmatrix}0.004 & 0.016\\ 0.004 & 0.004\end{bmatrix}}{1-0.999} = \begin{bmatrix}4 & 0.16\\ 4 & 0.0044\end{bmatrix}$$

Then, computing the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m}$

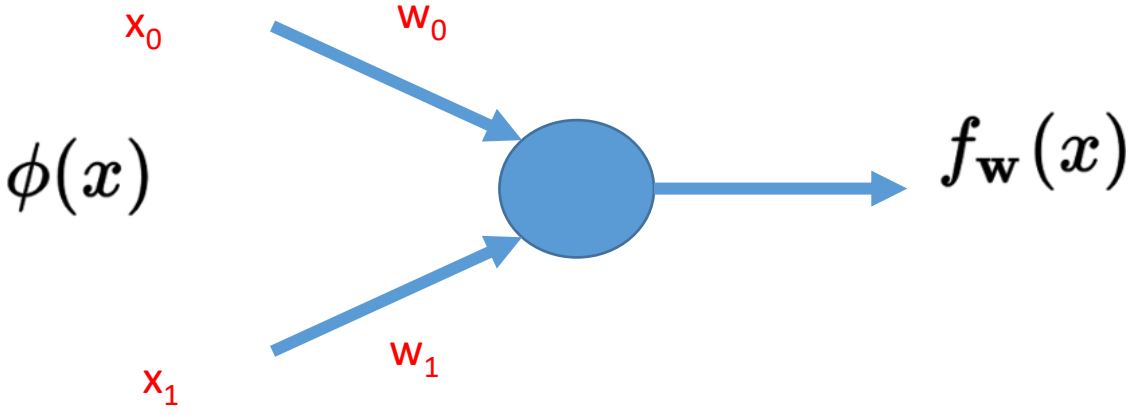
$$\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m} = \frac{0.1}{0.00001 + \sqrt{\begin{bmatrix}4 & 0.16\\ 4 & 0.0044\end{bmatrix}}}\begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix} = \begin{bmatrix}-0.025 & -0.006\\ -0.025 & 0.025\end{bmatrix}$$

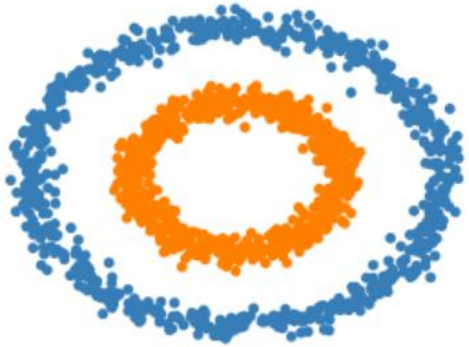
Finally, perform a parameter update

$$\theta_{i+1} = \theta_i - \frac{\lambda}{\delta+\sqrt{r}}\odot\nabla_\theta g(\theta) = \begin{bmatrix}1 & -1\\ 1 & 2\end{bmatrix} - \begin{bmatrix}-0.025 & -0.006\\ -0.025 & 0.025\end{bmatrix} = \begin{bmatrix}1.025 & -0.994\\ 1.025 & 1.975\end{bmatrix}$$

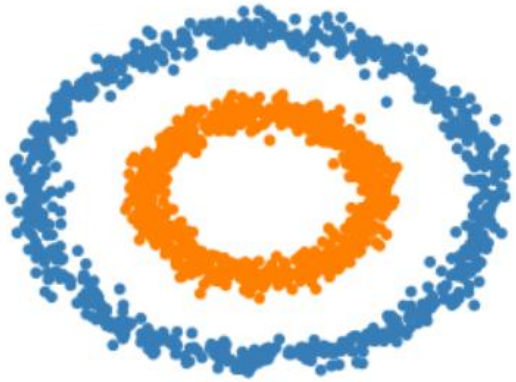


Lecture 7 – : Neural Networks



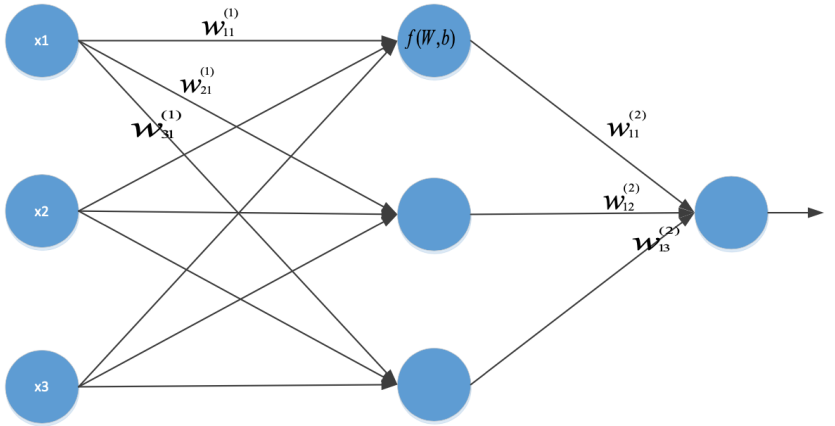


Lecture 7 – :Neural Networks – Linear Transformation



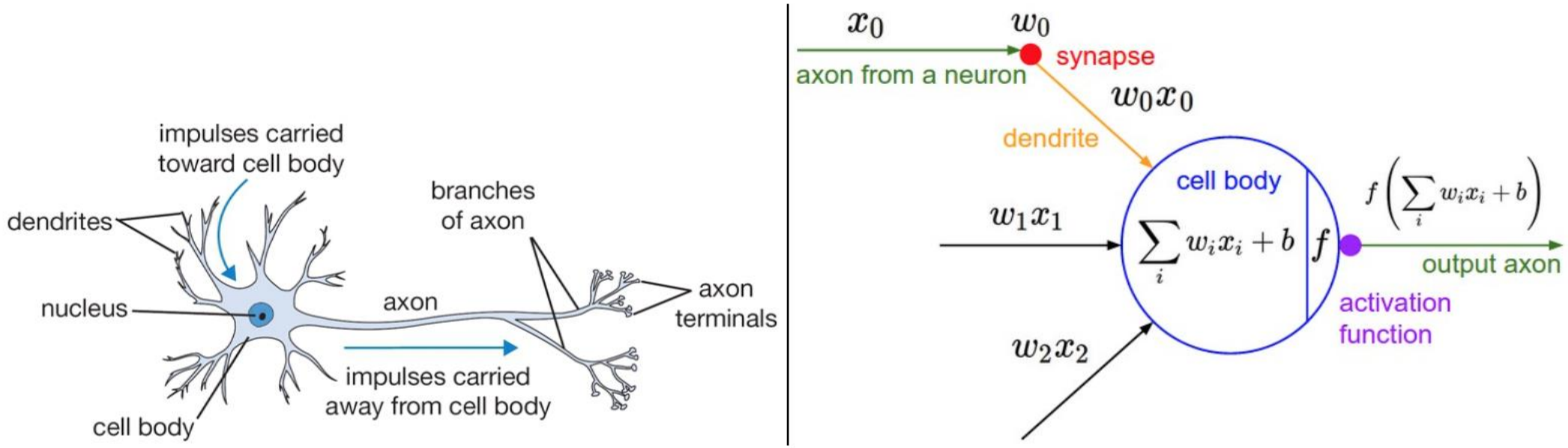
Can we obtain decision boundaries which are circles by using linear classifiers/ Linear Transformation?

Yes
No



Demo

Lecture 7 – :Neural Networks – Neuron / Unit /

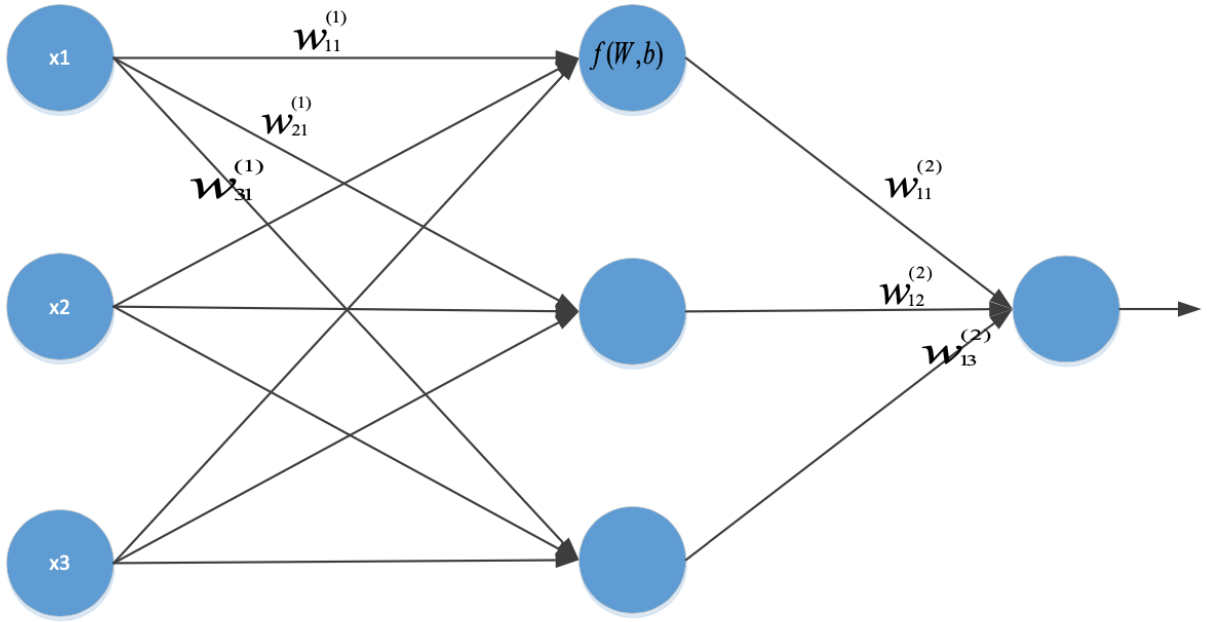


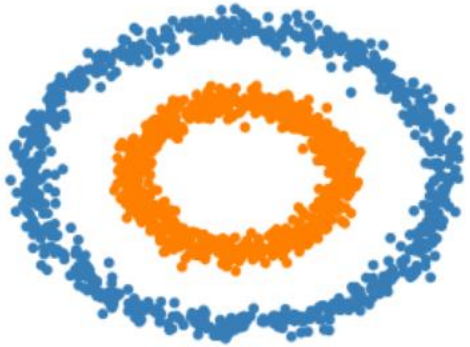
A cartoon drawing of a biological neuron (left) and its mathematical model (right).


Fei-Fei Li


Andrej Karpathy

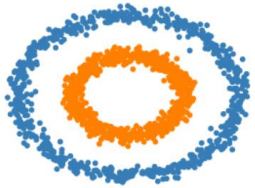
Lecture 7 – :Neural Networks – Activation (Nonlinearity)



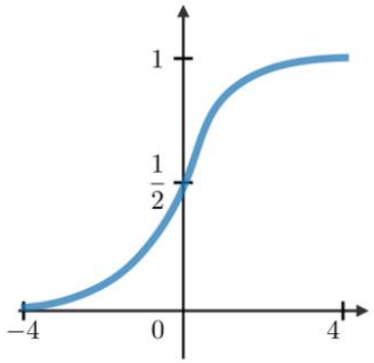
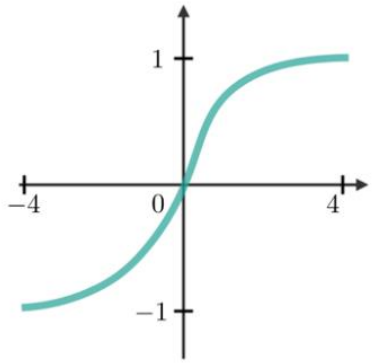


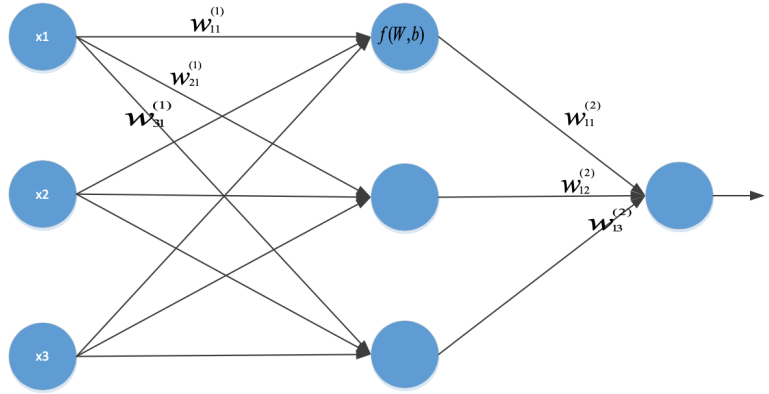
Lecture 7 – :Neural Networks – Activation (Nonlinearity)

University of Salford MANCHESTER

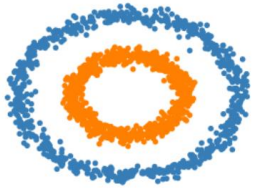


Activation function: Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model.

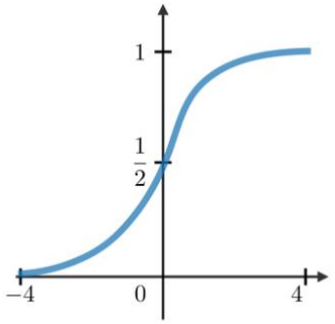
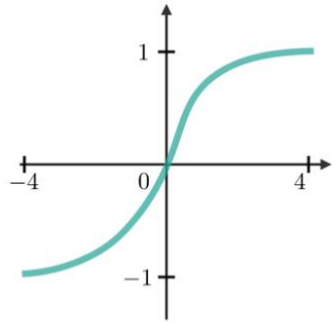
Sigmoid	Tanh
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
	

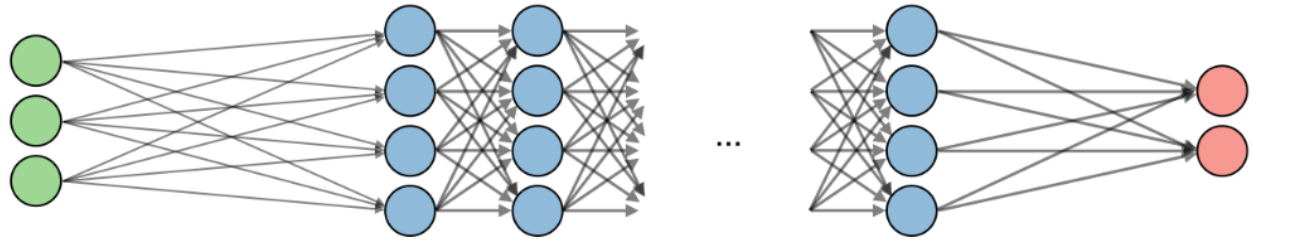


Lecture 7 – :Neural Networks – Activation (Nonlinearity)



Architecture: The vocabulary around neural networks architectures is described in the figure below:

Sigmoid	Tanh
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
	



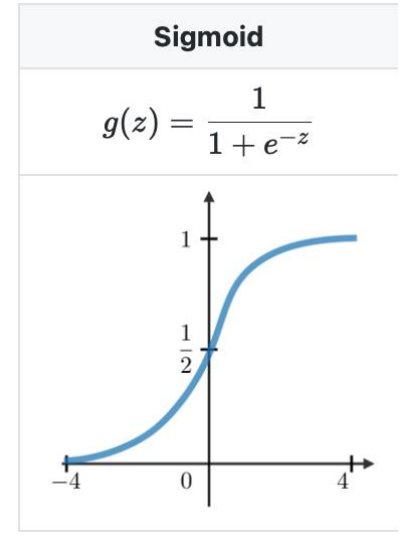
Input layer Hidden layer 1 ... Hidden layer k Output layer

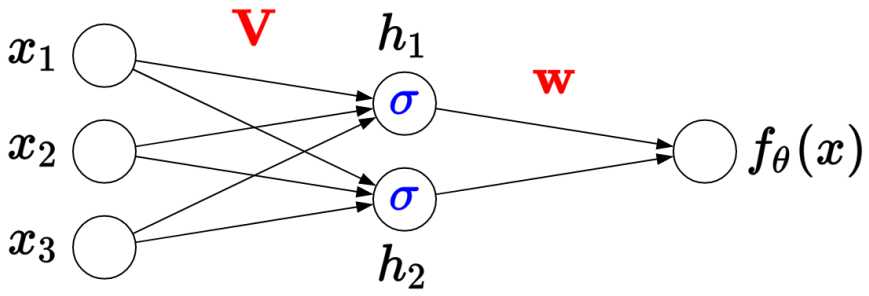
By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]^T} x + b_j^{[i]}$$

Demo

Lecture 7 – :Neural Networks – Activation (Nonlinearity)





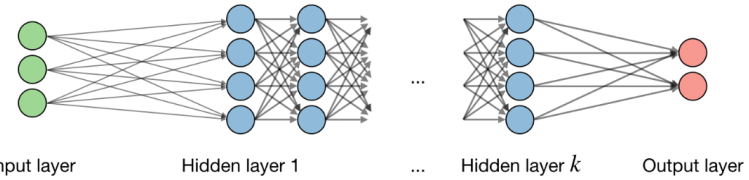
Intermediate hidden units:

$$h_j(x) = \sigma(\mathbf{v}_j \cdot x) \quad \sigma(z) = (1+e^{-z})^{-1}$$

Output:

$$f_\theta(x) = \mathbf{w} \cdot \mathbf{h}(x)$$

Parameters: $\theta = (\mathbf{V}, \mathbf{w})$



By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

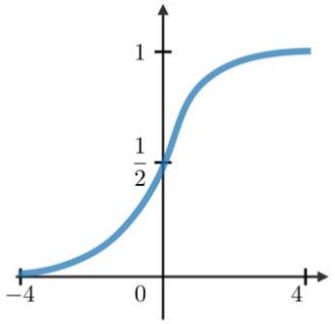
$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

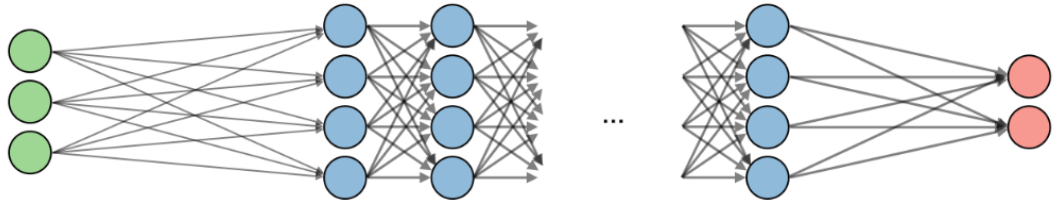
Lecture 7 – :Neural Networks – Activation (Nonlinearity)

Vanishing gradient and Exploding gradient

Sigmoid

$$g(z) = \frac{1}{1+e^{-z}}$$





By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

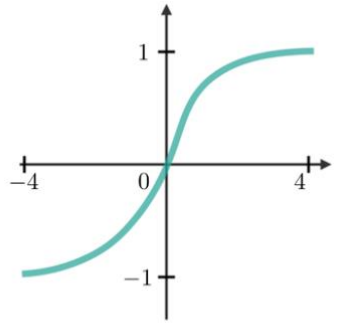
$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

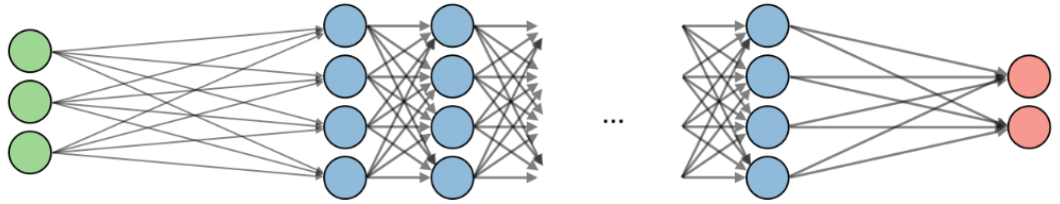
Lecture 7 – :Neural Networks – Activation (Nonlinearity)

Vanishing gradient

Tanh

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

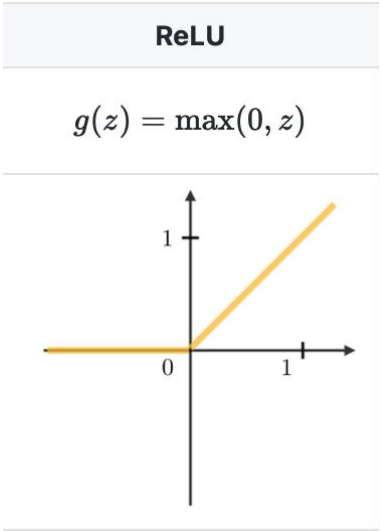


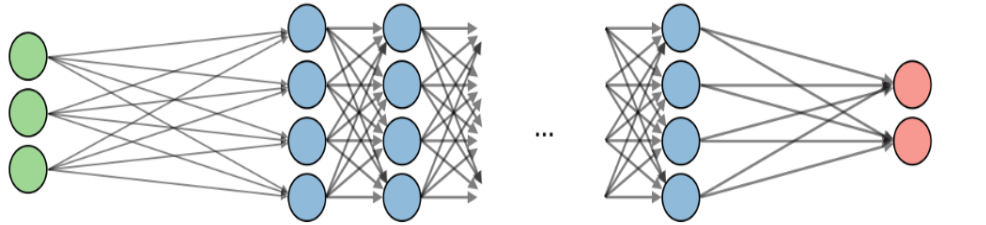


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}$$

Lecture 7 – :Neural Networks – Activation (Nonlinearity)



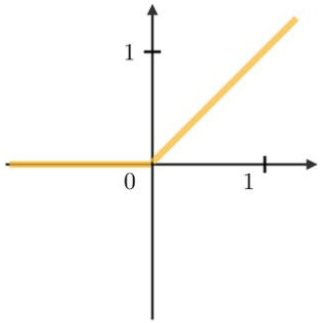
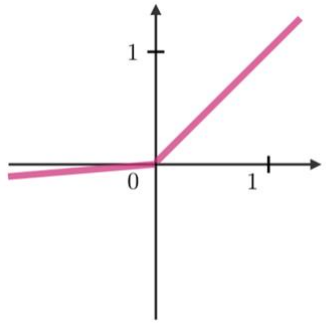


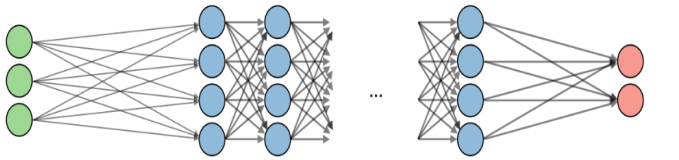
By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

Demo

Lecture 7 – :Neural Networks – Activation (Nonlinearity)

ReLU	Leaky ReLU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
	



By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]^T} x + b_j^{[i]}$$

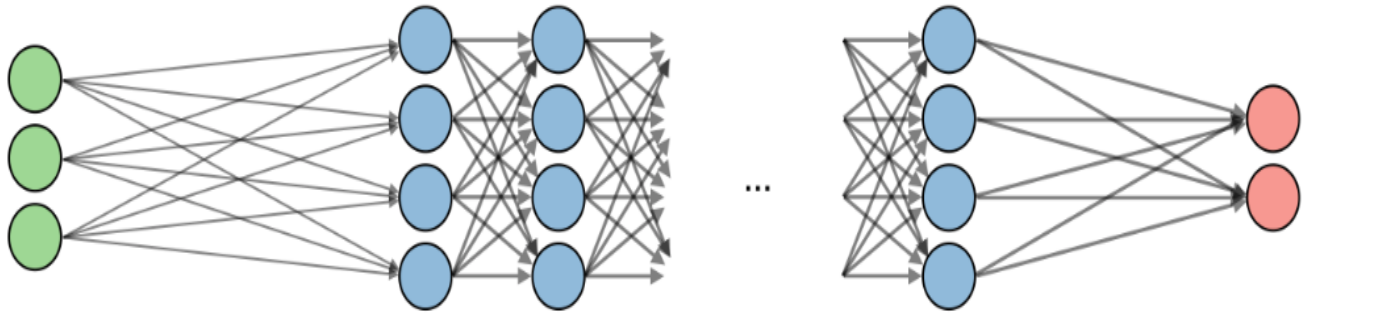
Lecture 7 – :Neural Networks – Activation (Nonlinearity) - Code

University of Salford MANCHESTER

```
import tensorflow as tf
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')


model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(2, activation='relu'),
  tf.keras.layers.Dense(1, activation='softmax')
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')

model.fit(inputs, targets, epochs=5)
model.predict(np.array([[2,0]]))
```

Lecture 7 – : Deep Learning Challenges of Training

Features									y

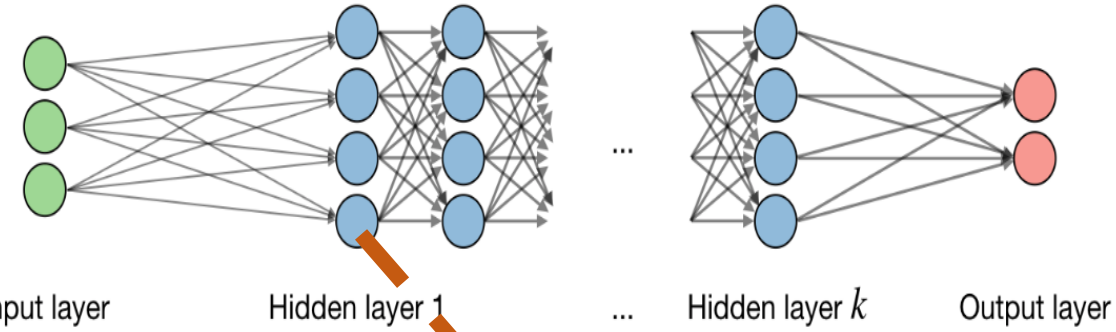


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}$$

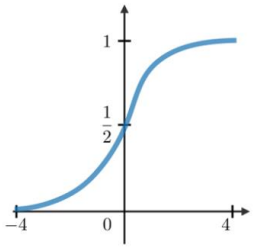
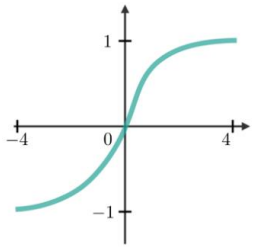
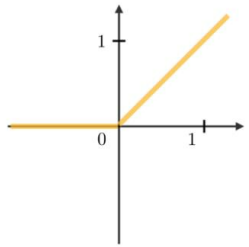
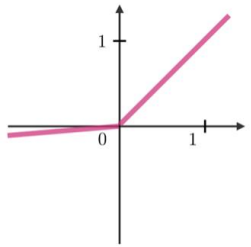
Lecture 2 - Lecture 7 : Artificial Intelligent – Deep Learning

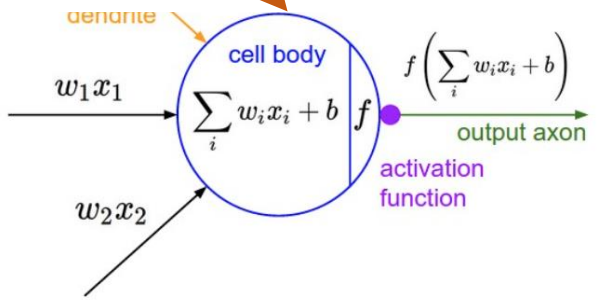
Features									output
x1	x2	x3	x4	x5	x6	x6	x7	x8	y



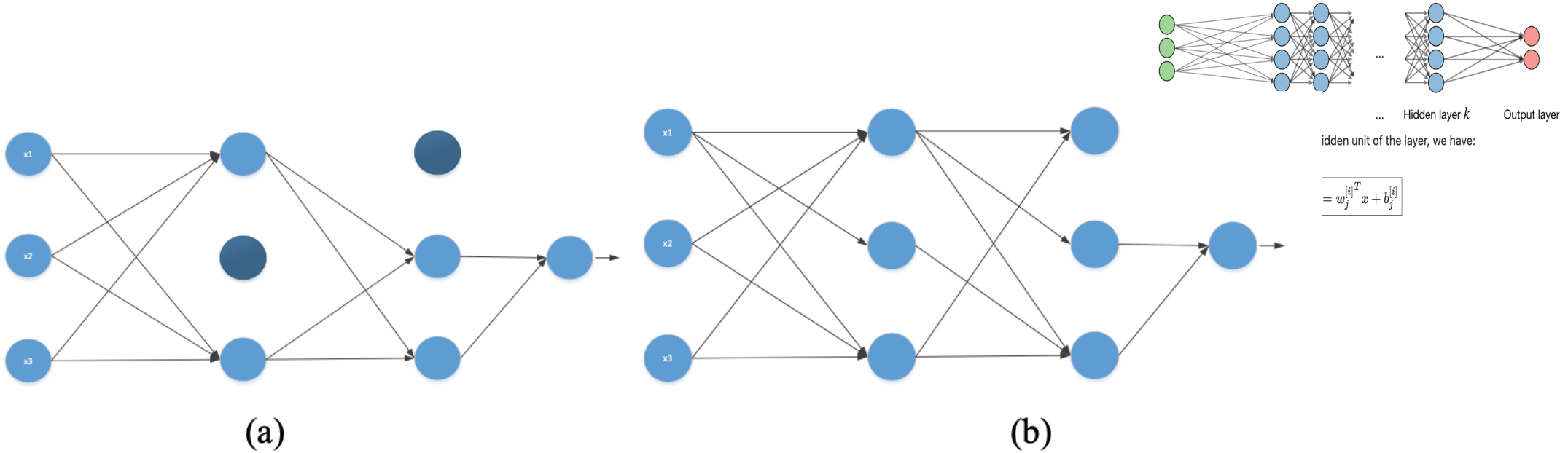
By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			



Lecture 7 Deep Learning Dropout / DropConnection



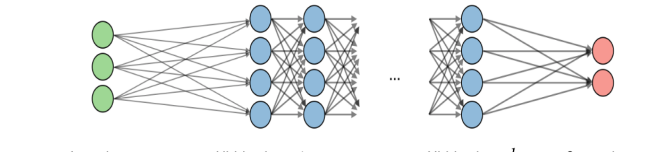
Neural networks (a) after dropout (b) after DropConnection.

Lecture 7 Deep Learning Dropout / DropConnection

University of Salford MANCHESTER

```
import tensorflow as tf
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')


model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(2, activation='relu'),
  tf.keras.layers.Dropout(.2),
  tf.keras.layers.Dense(1, activation='softmax')
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')

model.fit(inputs, targets, epochs=5)
model.predict(np.array([[2,0]]))
```

Lecture 7 Deep Learning
Batch Normalization



g i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]^T} x + b_j^{[i]}$$

1. The first step is to compute the mean and the variance of the mini batch as:

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \text{ and } \sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

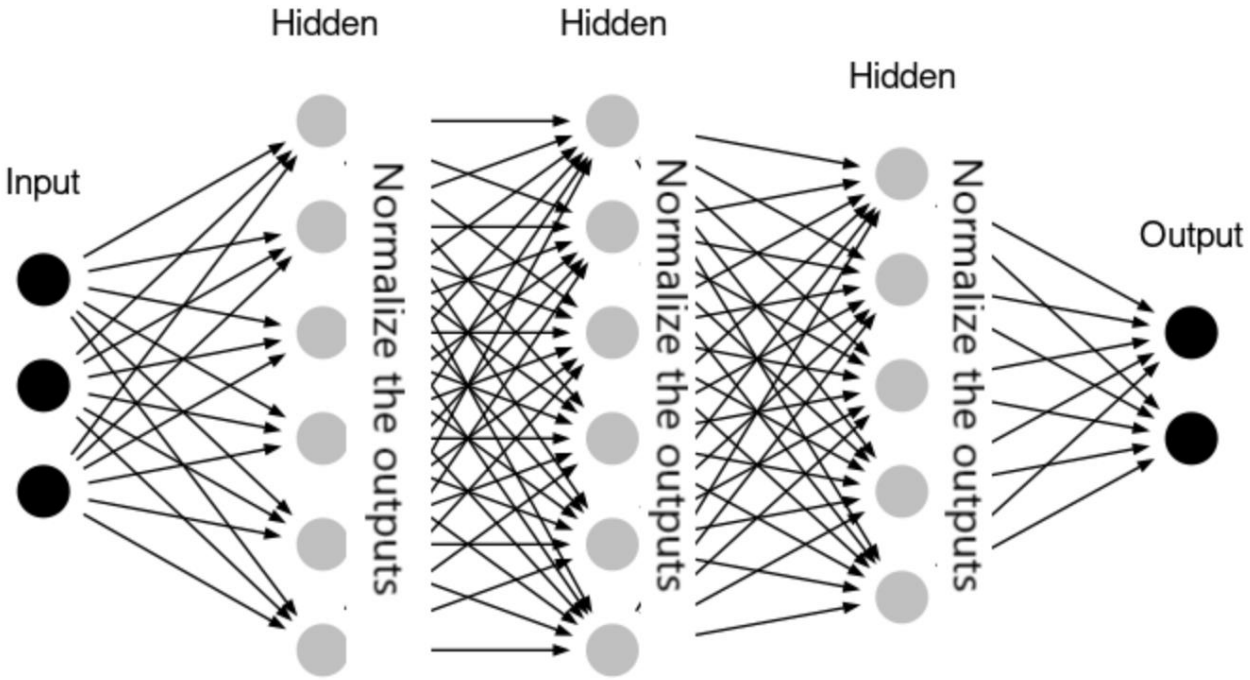
2. The second step involves normalizing the mini batch input as follows:

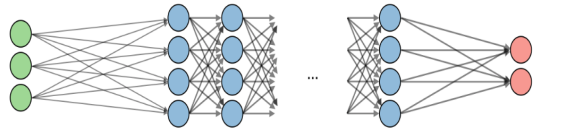
$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

3. Finally, the normalized mini batch is scaled using learnable parameters (γ, β):

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

Lecture 7 Deep Learning Batch Normalization





By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

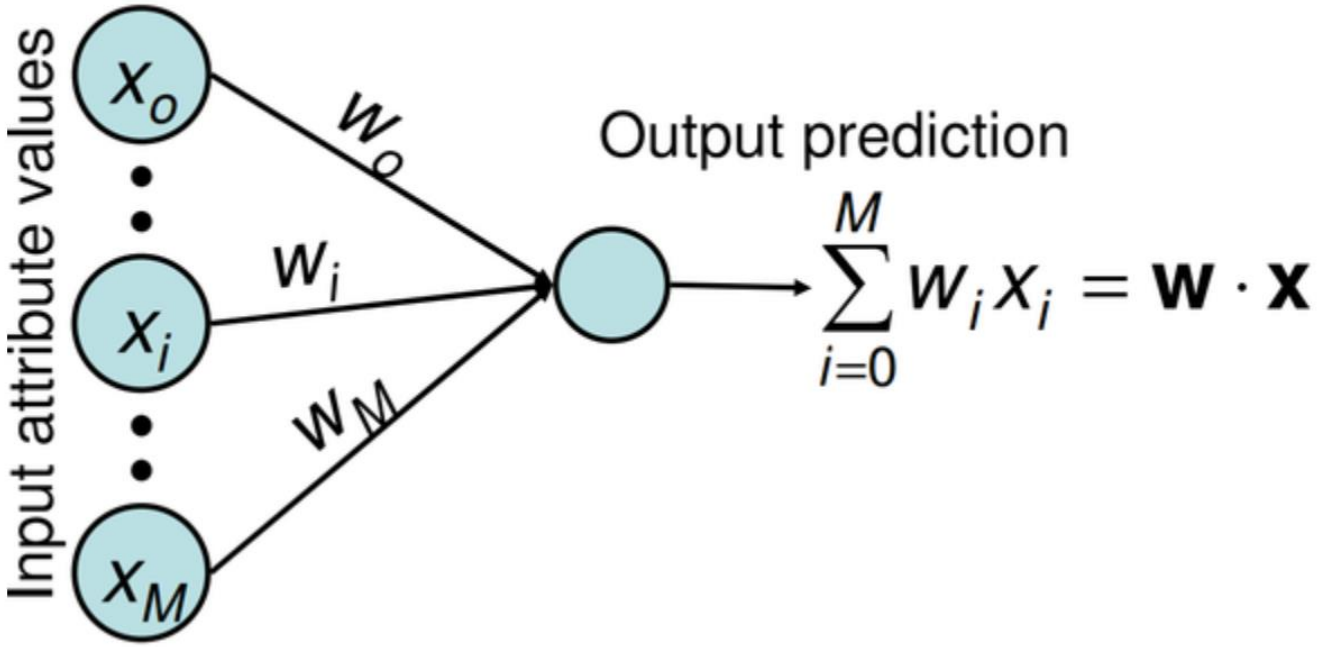
$$z_j^{[i]} = w_j^{[i]^T} x + b_j^{[i]}$$

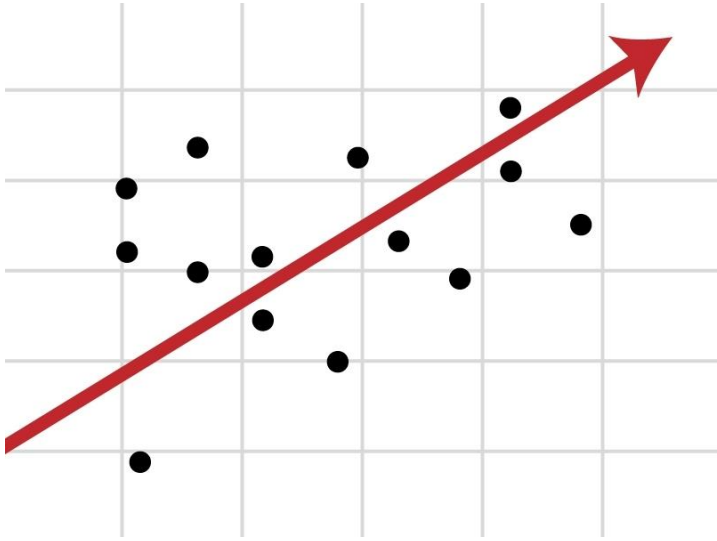
Lecture 7 Deep Learning Batch Normalization

University of Salford MANCHESTER

```
import tensorflow as tf
inputs = np.array([[0, 1],  [2, 1], [1, 0]], dtype='float32')
# Targets (apples, oranges)
targets = np.array([[1], [1], [0]], dtype='float32')


model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(2, activation='relu'),
  tf.keras.layers.Dropout(.2),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Dense(1, activation='softmax')
])

model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')

model.fit(inputs, targets, epochs=5)
model.predict(np.array([[2,0]]))
```

Lecture 7: AI – Deep Learning – Regression - Code



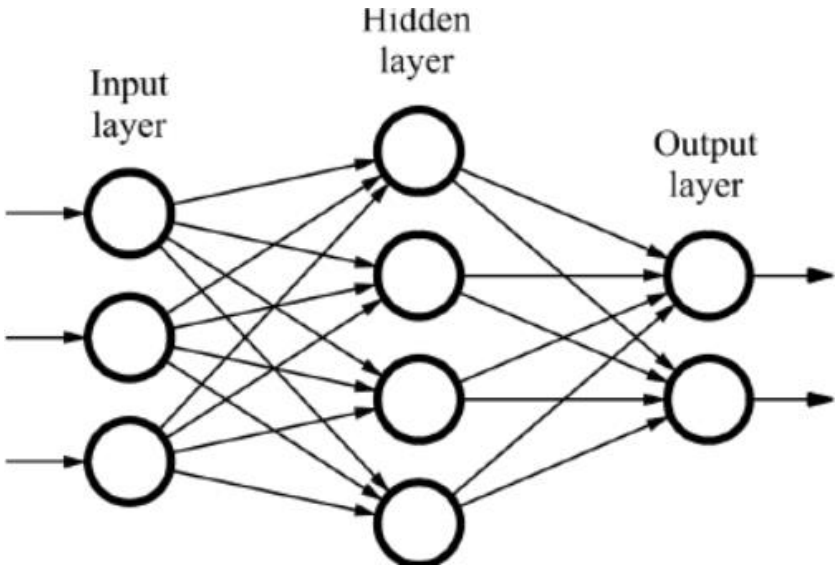


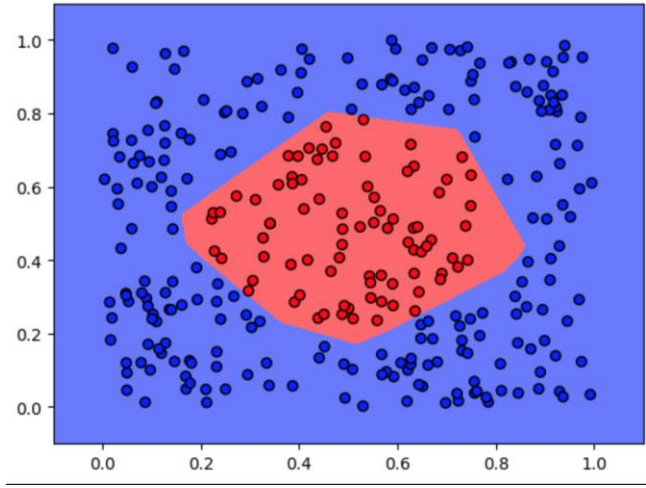
Lecture 7: AI – Deep Learning – Regression - Code

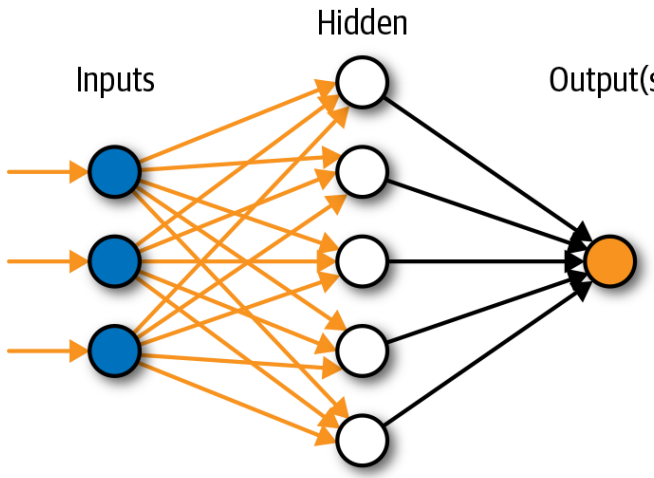
University of Salford MANCHESTER

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pandas as pd

abalone_train = pd.read_csv("abalone_train.csv",names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight","Viscera weight", "Shell weight", "Age"])
abalone_features = abalone_train.copy()
abalone_labels = abalone_features.pop('Age')

abalone_model = tf.keras.Sequential([
 layers.Dense(64, kernel_regularizer=regularizers.l2(0.001)),
 layers.Dropout(0.6),
 layers. BatchNormalization()’
 layers.Dense(1)  ])

abalone_model.compile(loss = tf.losses.MeanSquaredError(), optimizer = tf.optimizers.Adam())
abalone_model.fit(abalone_features, abalone_labels, epochs=10)
```

Lecture 7: AI – Deep Learning - Binary Classification







Lecture 7: AI – Deep Learning - Binary Classification Code

University of Salford MANCHESTER

```
# mlp for binary classification
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
# load the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.csv'
df = read_csv(path, header=None)
df.head(3)
```

	0	1	2	3	4	5	6	7	8	9	...	25	26	27	28	29	30	31	32	33	34
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171	0.41078	-0.46168	0.21266	-0.34090	0.42267	-0.54487	0.18641	-0.45300	g
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468	-0.18401	-0.19040	-0.11593	-0.16626	-0.06288	-0.13738	-0.02447	b
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984	-0.22145	0.43100	-0.17365	0.60436	-0.24180	0.56045	-0.38238	g

3 rows × 35 columns

Lecture 7: AI – Deep Learning - Binary Classification Code

University of Salford MANCHESTER

```
# split into input and output columns
X, y = df.values[:, :-1], df.values[:, -1]
# ensure all data are floating point values
X = X.astype('float32')
# encode strings to integer
le = LabelEncoder().fit(y)
y = le.transform(y)
# split into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# determine the number of input features
n_features = X_train.shape[1]
```

```
(235, 34) (116, 34) (235,) (116,)
```

Lecture 7: AI – Deep Learning - Binary Classification Code

University of Salford MANCHESTER

```
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

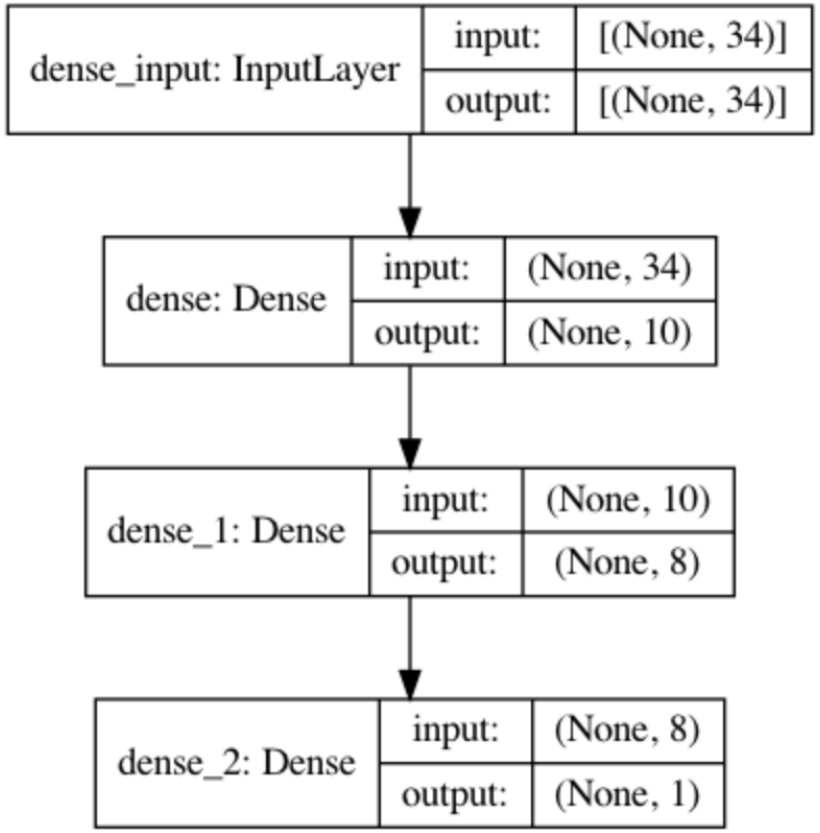
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                350
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 88
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 447
Trainable params: 447
Non-trainable params: 0
_________________________________________________________________
```

Lecture 7: AI – Deep Learning - Binary Classification Code

University of Salford MANCHESTER

```
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
```

```
from tensorflow.keras.utils import plot_model
plot_model(model, 'model.png', show_shapes=True)
```



Lecture 7: AI – Deep Learning - Binary Classification Code

University of Salford MANCHESTER

```
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# fit the model
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)
```

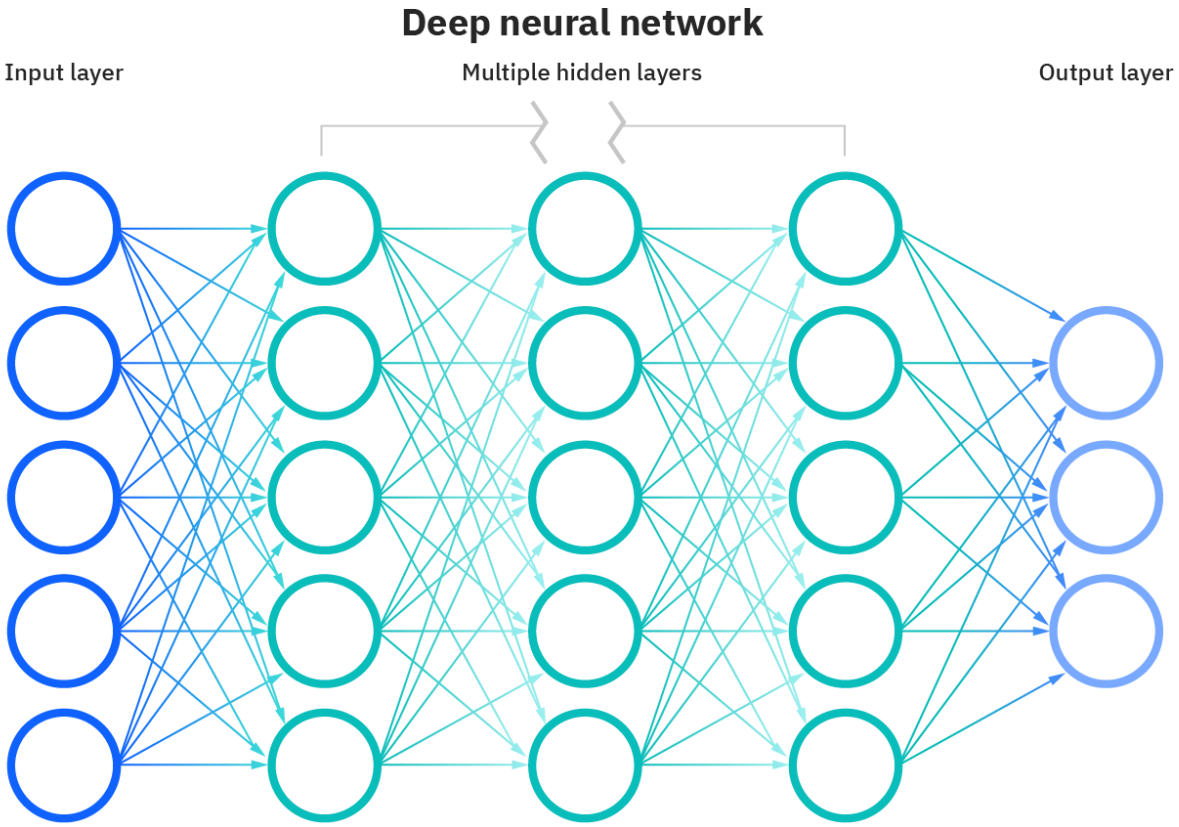
Lecture 7: AI – Deep Learning - Binary Classification Code

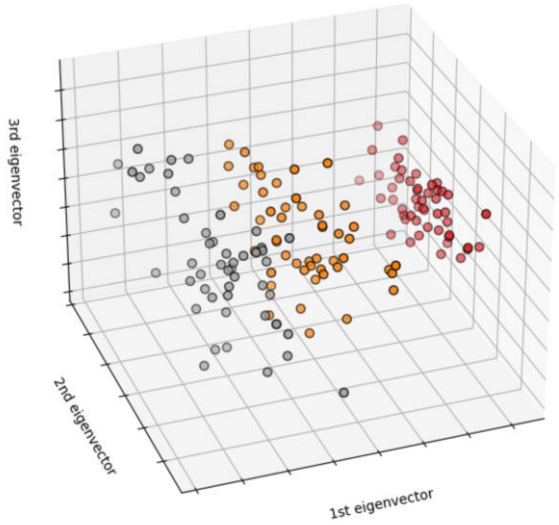
University of Salford MANCHESTER

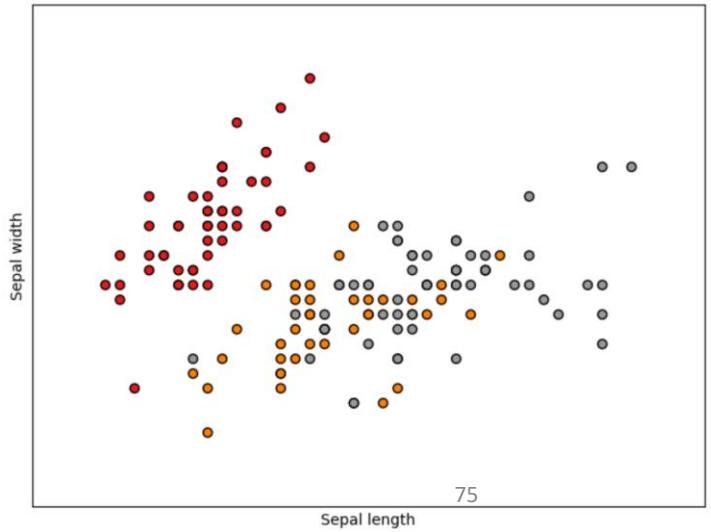
```
# evaluate the model
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
# make a prediction
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.03760,0.85243,-0.17755,0.59755,
       -0.44945,0.60536,-0.38223,0.84356,-0.38542,0.58212,-0.32192,0.56971,-0.29674,0.36946,-0.47357,
       0.56811,-0.51171,0.41078,-0.46168,0.21266,-0.34090,0.42267,-0.54487,0.18641,-0.45300]
yhat = model.predict([row])
yhat[yhat <= 0.5] = 0
yhat[yhat > 0.5] = 1
yhat = int(yhat.item())
print('Predicted: ',  yhat)
print('Predicted: ',  le.inverse_transform([yhat]).item())
```

```
Test Accuracy: 0.922
Predicted:  1
Predicted:  g
```

Lecture 7: AI – Deep Learning - Multi Classification







Lecture 7: AI – Deep Learning - Multi Classification Code

University of Salford MANCHESTER

```
# mlp for multiclass classification
# load the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'
df = read_csv(path, header=None)
# split into input and output columns
X, y = df.values[:, :-1], df.values[:, -1]
# ensure all data are floating point values
X = X.astype('float32')
batch_size = 32
# encode strings to integer
y = LabelEncoder().fit_transform(y)
# split into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# determine the number of input features
n_features = X_train.shape[1]
```

Lecture 7: AI – Deep Learning - Multi Classification Code

University of Salford MANCHESTER

```
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(3, activation='softmax'))
# compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# fit the model

model.fit(X_train, y_train, epochs=150, batch_size=batch_size, verbose=0)
# evaluate the model

loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
# make a prediction
row = [5.1,3.5,1.4,0.2]
yhat = model.predict([row])
print('Predicted: %s (class=%d)' % (yhat, argmax(yhat)))
y_label = argmax(yhat)
print('Predicted: ',  le.inverse_transform([y_label]).item())
```

```
(100, 4) (50, 4) (100,) (50,)
Test Accuracy: 0.900
Predicted: [[0.92005795 0.05173988 0.02820225]] (class=0)
Predicted:  b
```

Lecture 7: AI – Deep Learning - Multi Classification Code - Pipeline

University of Salford MANCHESTER

```
Train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train)).batch(batch_size)
model.fit(Train_dataset)
val_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(batch_size)
loss, acc = model.evaluate(val_dataset)
# make a prediction
row = [5.1,3.5,1.4,0.2]
yhat = model.predict([row])
print('Predicted: %s (class=%d)' % (yhat, argmax(yhat)))
y_label = argmax(yhat)
print('Predicted: ',  le.inverse_transform([y_label]).item())
```

```
4/4 [==============================] - 0s 1ms/step - loss: 0.6117 - accuracy: 0.8900
2/2 [==============================] - 0s 2ms/step - loss: 0.5791 - accuracy: 0.9000
Predicted: [[0.9208795  0.05103501 0.02808554]] (class=0)
Predicted:  b
```

Lecture 7: AI – Deep Learning - Multi Classification Code - TensorBoard

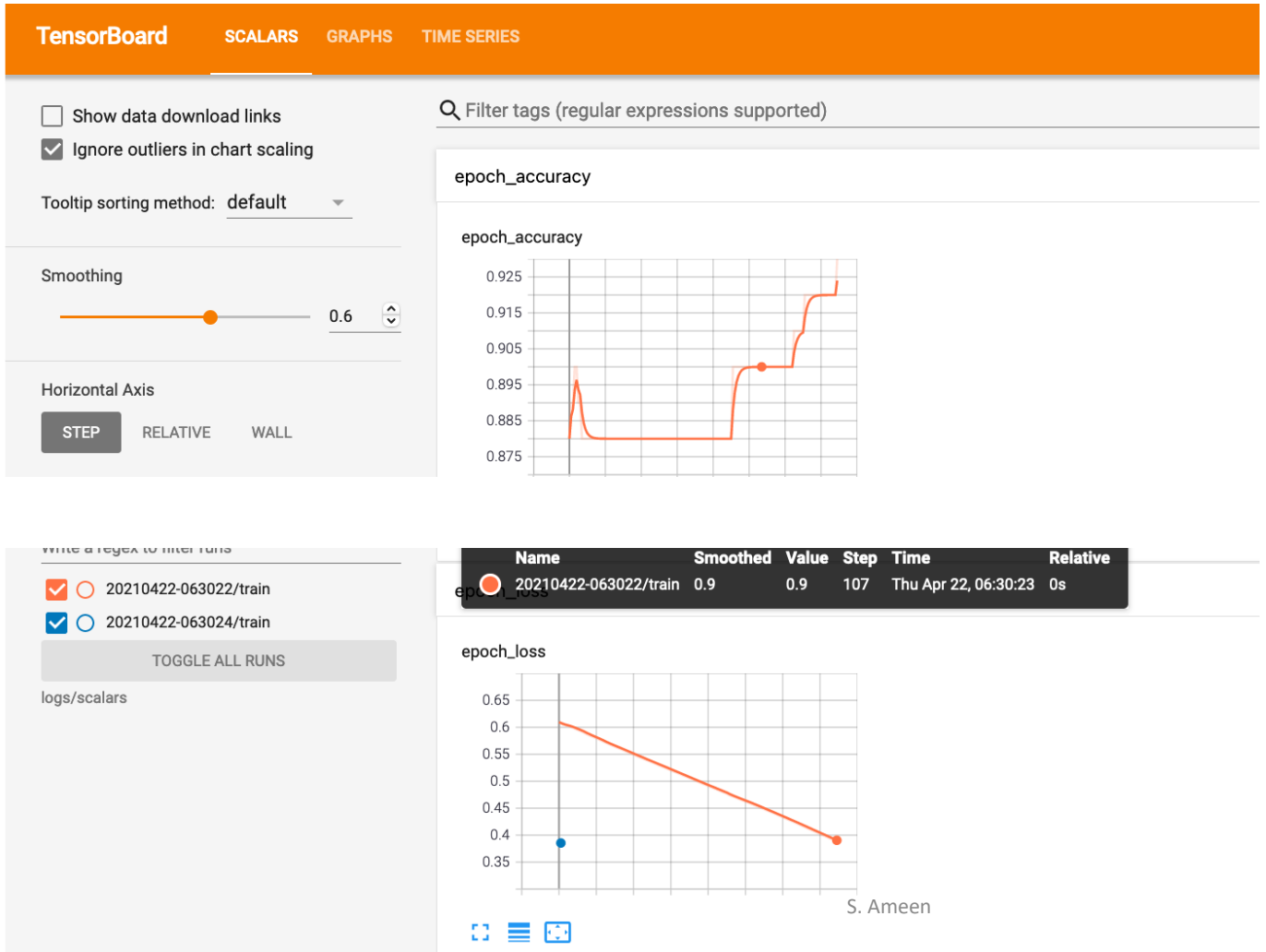
University of Salford MANCHESTER

```
from datetime import datetime
logdir = "logs/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)
```

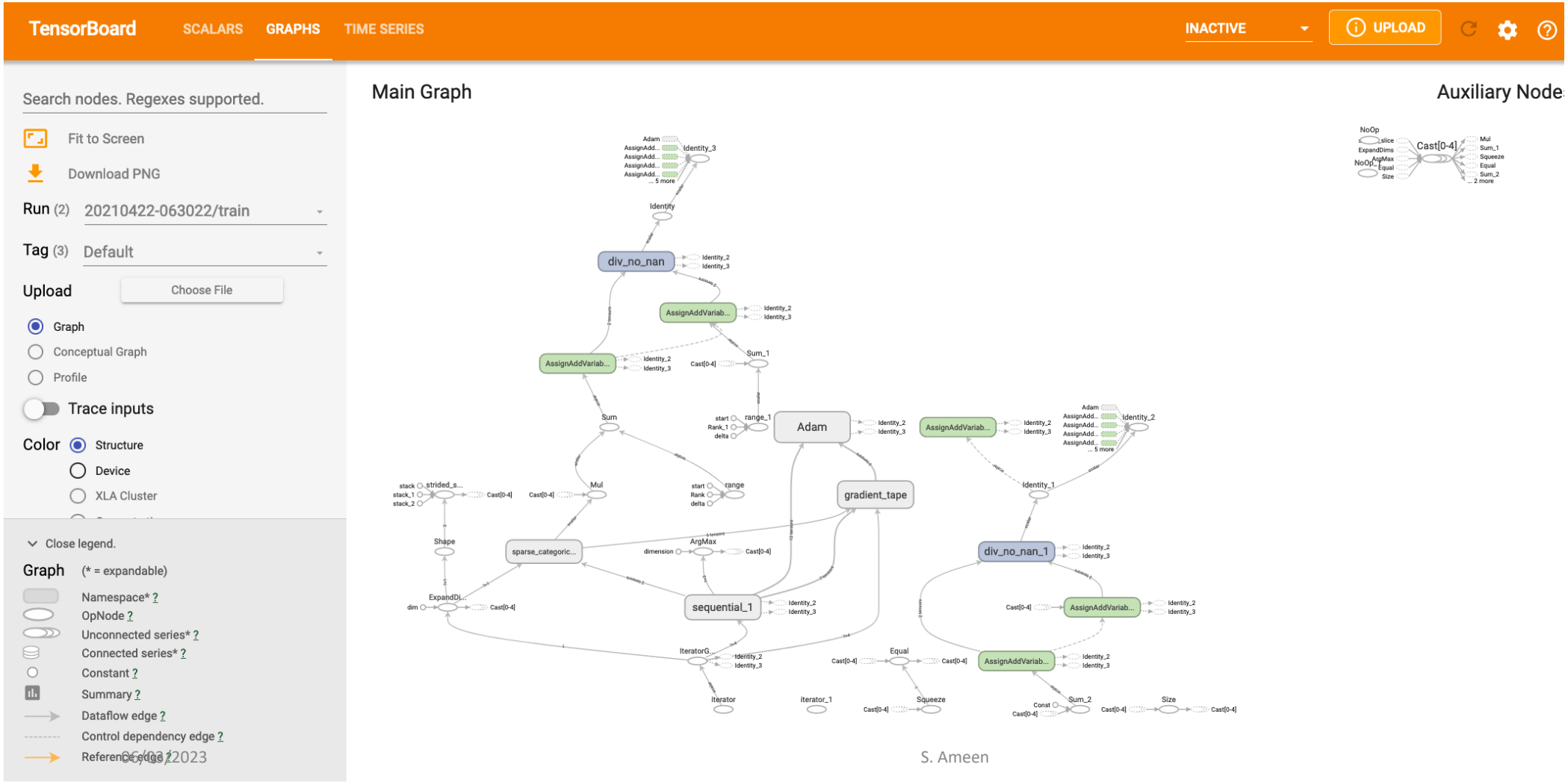
```
model.fit(Train_dataset,epochs=150, batch_size=batch_size, verbose=0, callbacks=[tensorboard_callback])
```

```
%load_ext tensorboard
%tensorboard --logdir logs/scalars
```

Lecture 7: AI – Deep Learning - Multi Classification Code - TensorBoard



Lecture 7: AI – Deep Learning - Multi Classification Code - TensorBoard



Lecture 7: AI – Deep Learning - Multi Classification Code – Save Models

University of Salford MANCHESTER

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath='Models/model_{epoch}',
        save_freq='epoch'),
    tensorboard_callback
]
```

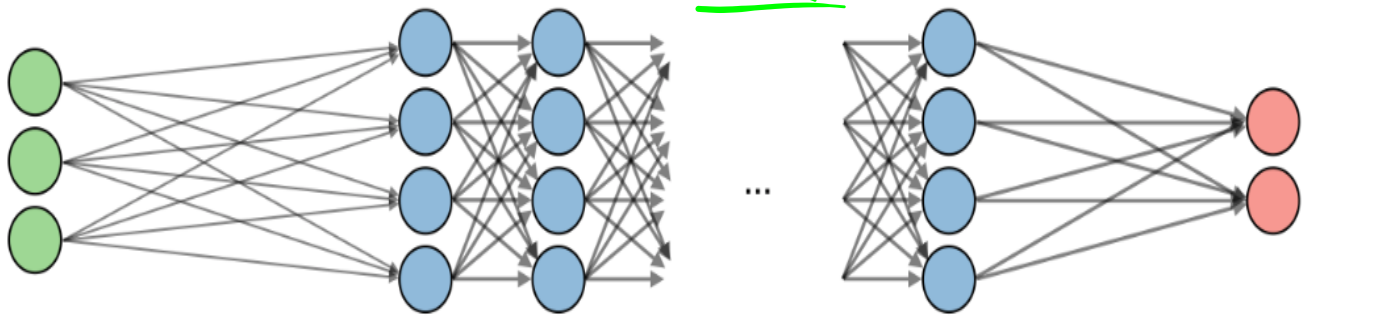
```
model.fit(Train_dataset,epochs=2, batch_size=batch_size, verbose=0, callbacks=callbacks)
```

```
INFO:tensorflow:Assets written to: Models/model_1/assets
INFO:tensorflow:Assets written to: Models/model_2/assets
<tensorflow.python.keras.callbacks.History at 0x7ffbf1c031f0>
```

Lecture 7 Deep Learning
Structured data/ Tabular data

Features									y

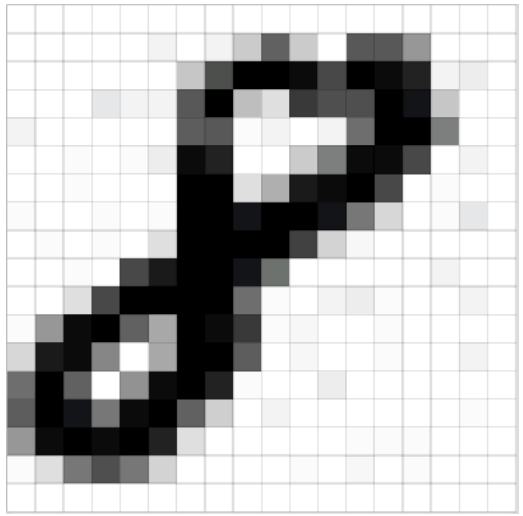
MLP

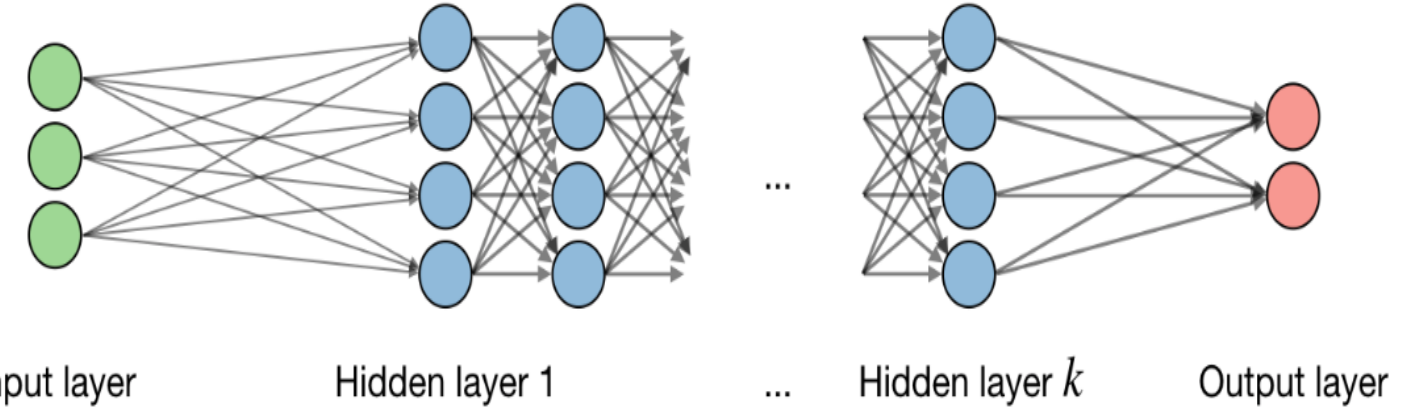


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

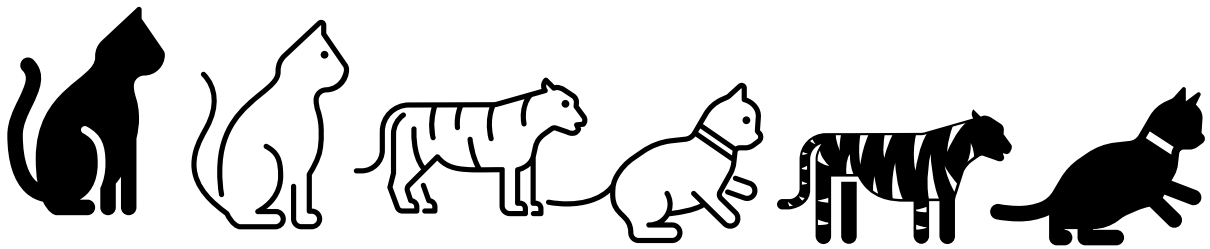
$$z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}$$

Lecture 7 Deep Learning Unstructured data/ Images



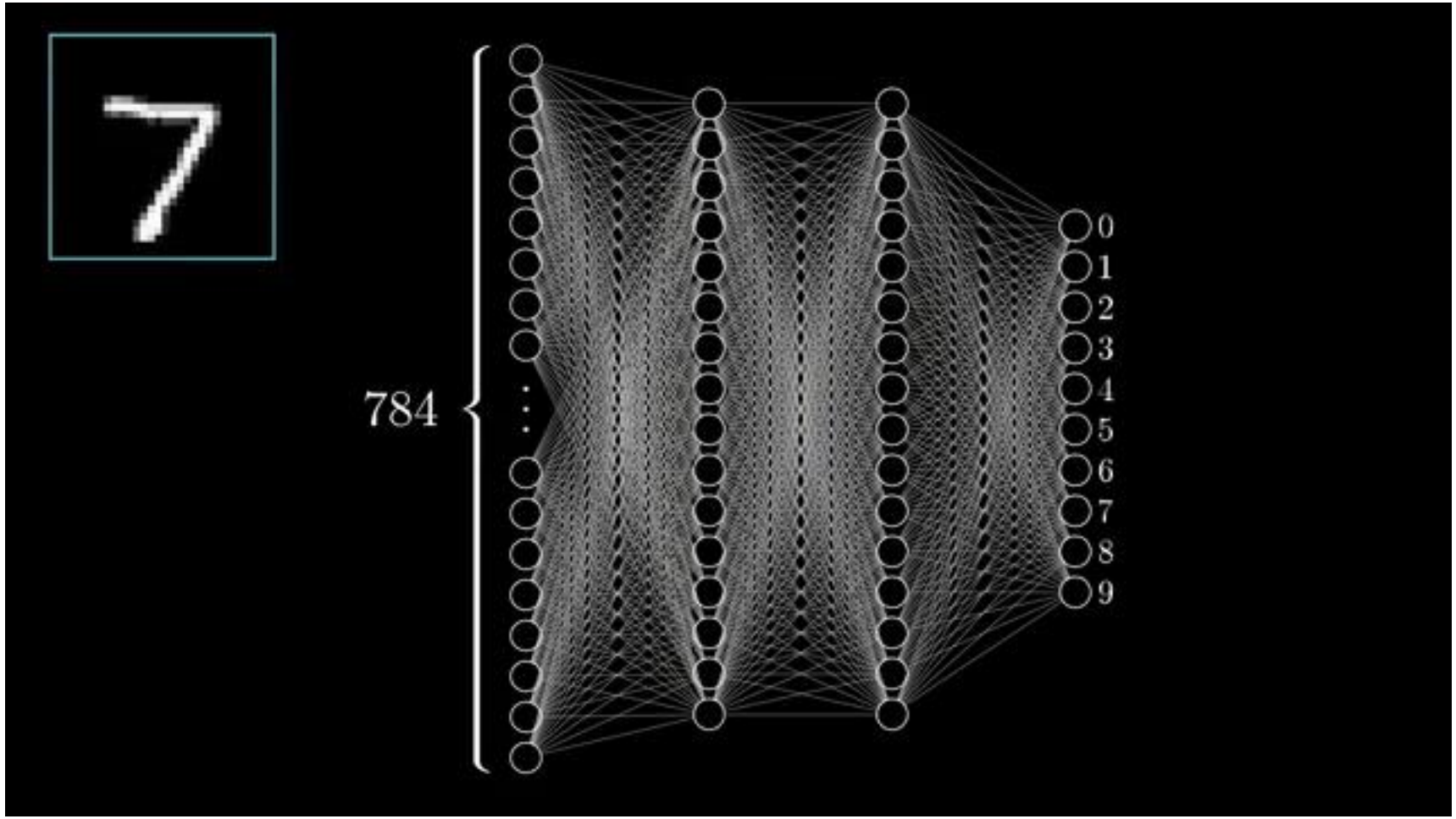


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

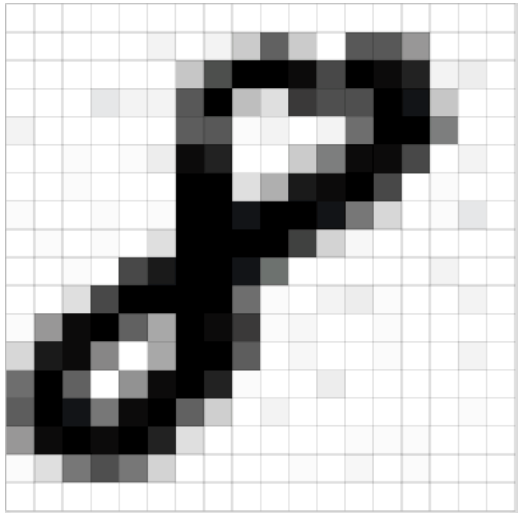


$$z_j^{[i]} = w_j^{[i]^T} x + b_j^{[i]}$$

Lecture 7 Deep Learning Unstructured data/ Images



Lecture 7: Artificial Intelligent – Deep Learning - Coding



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
model = keras.models.Sequential([
  layers.Flatten(input_shape=(28, 28)),
  layers.Dense(128, activation='relu'),
  layers.Dropout(0.2),
  layers.Dense(10)    ])
```

```
model = keras.Sequential()
model.add(layers .Flatten(input_shape=(28, 28))
model.add(layers.Dense(128, activation="relu”))
model.add (layers.Dropout(0.2))
model.add(layers.Dense(10))
```

```
model.compile(optimizer='adam’, loss='sparse_categorical_crossentropy',metrics=['accuracy'])

model.fit(x_train, y_train)
```

Lecture 7 Deep Learning MLP / Code

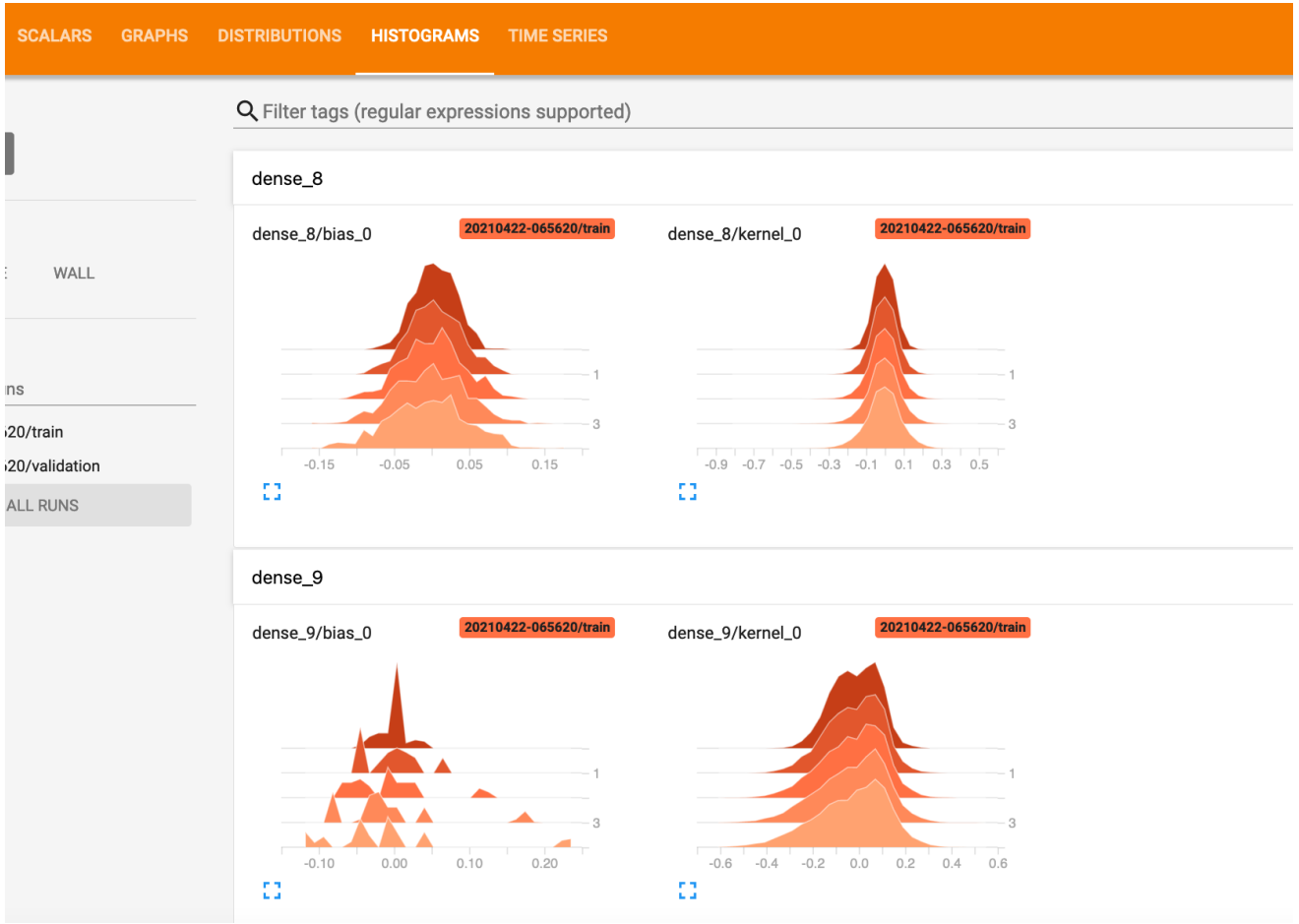
University of Salford MANCHESTER

```
import datetime
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

def create_model():
  return tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
  ])
```

```
model = create_model()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

model.fit(x=x_train,
          y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])
```

```
Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3598 - accuracy: 0.8923 - val_loss: 0.0990 - val_accuracy: 0.9705
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1004 - accuracy: 0.9699 - val_loss: 0.0796 - val_accuracy: 0.9751
Epoch 3/5
```

Lecture 7 Deep Learning MLP / Code



Lecture 7 Deep Learning Hardware and Software

University of Salford MANCHESTER

- Deep learning hardware
  - CPU,
  - GPU
- Deep learning software
  - PyTorch and TensorFlow
  - Static and Dynamic computation graphs

Lecture 7 Deep Learning Hardware and Software

University of Salford MANCHESTER

- Deep learning hardware
  - CPU, GPU

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$385	~540 GFLOPs FP32
GPU (NVIDIA RTX 2080 Ti)	3584	1.6 GHz	11 GB GDDR6	$1199	~13.4 TFLOPs FP32

[Fei-Fei Li]

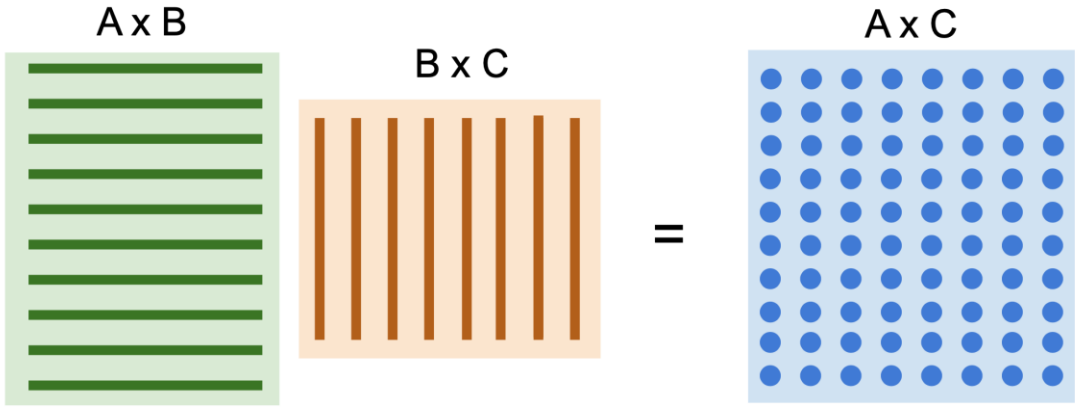
CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

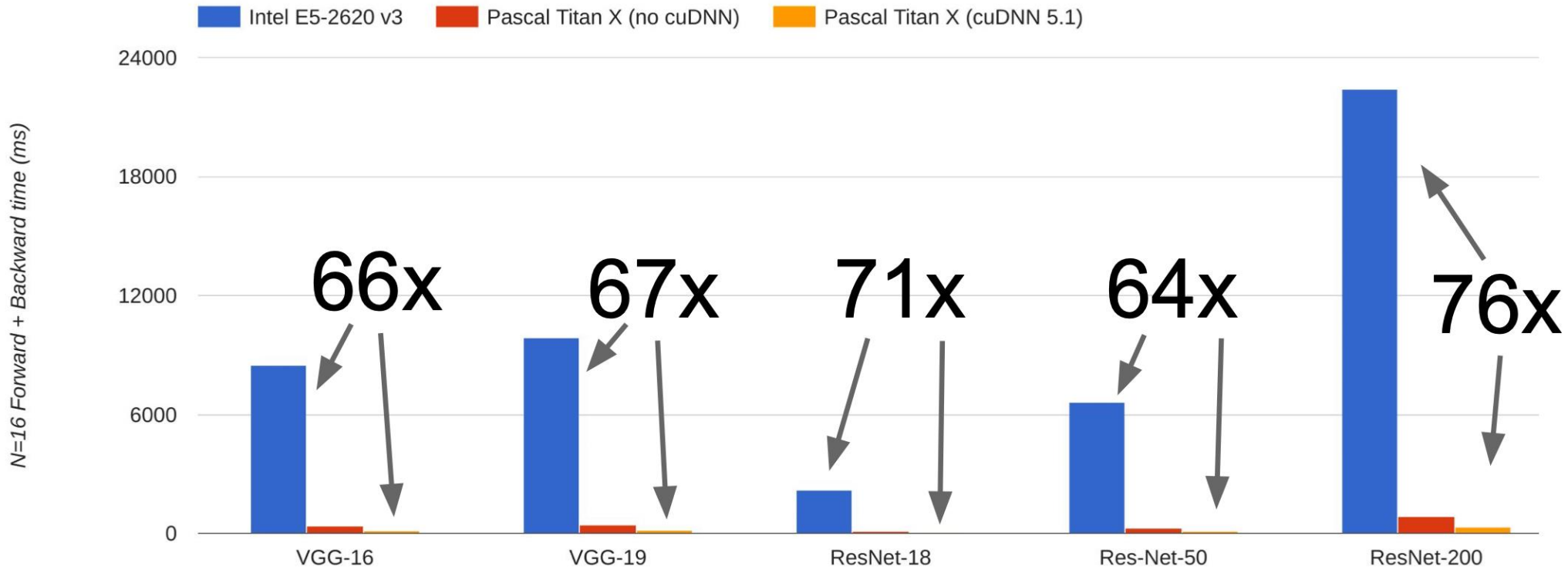
Lecture 7 Deep Learning Hardware and Software

University of Salford MANCHESTER

- Deep learning hardware
  - CPU, GPU , Matrix Multiplication

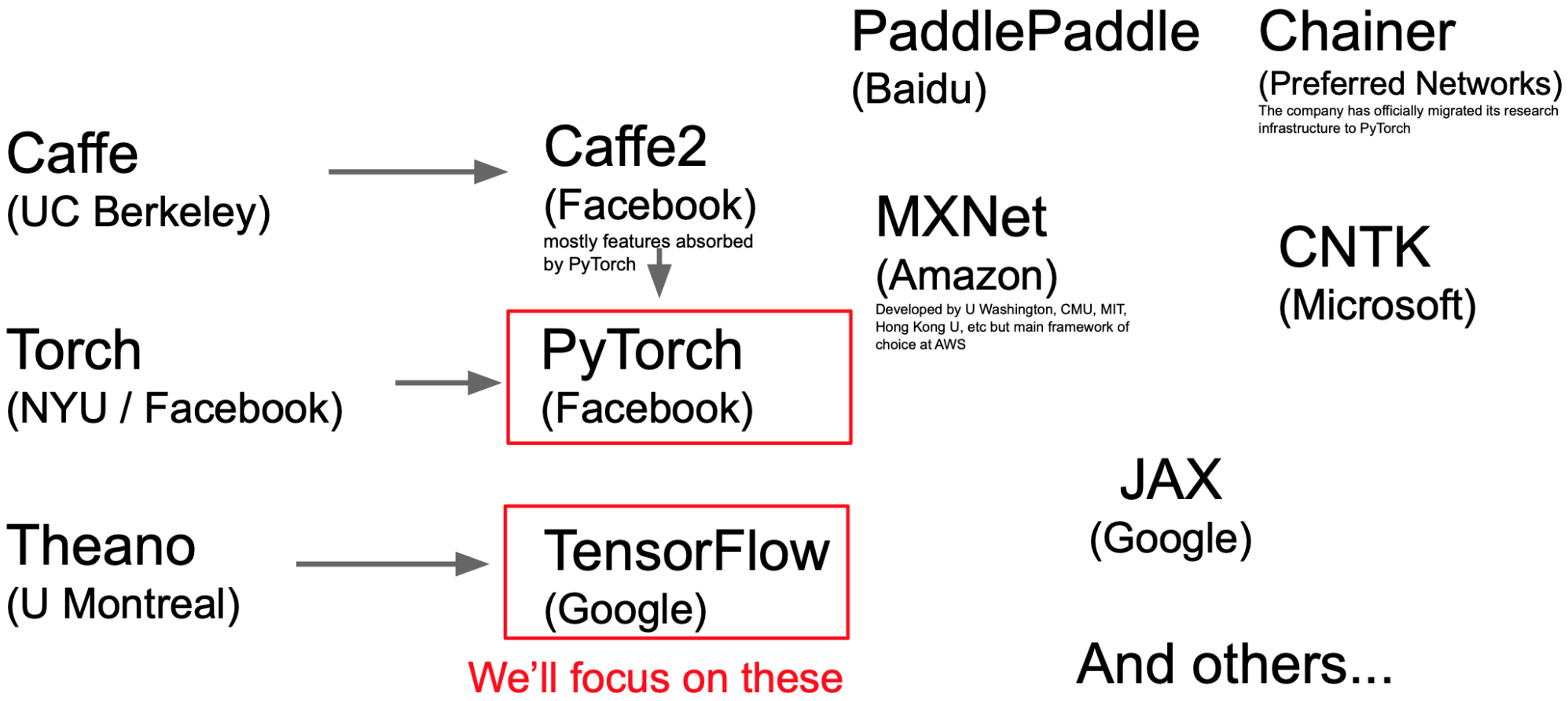


Lecture 7 Deep Learning Hardware and Software



Data from https://github.com/jcjohnson/cnn-benchmarks

Lecture 7 Deep Learning Hardware and Software



Lecture 7 Deep Learning Summary

University of Salford MANCHESTER

Deep Learning
Hardware and Software

Lecture 7: AI – Deep Learning - Next

University of Salford MANCHESTER

CNN

RNN