Classification & Optimisation

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Overview

In these slides, we'll start by giving some demonstrations of training classification models that: 1) predict a binary outcome, then 2) predict a categorical outcome with > 2 options or levels.

Next, we'll step into the maths of how these classification models make predictions, then go look at the high-level ideas of how to "train" them, then finally look at the maths of this training process.

Imports needed for these demos

```python
import random
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from keras.models import Sequential
from keras.layers import Dense, Input
from keras.callbacks import EarlyStopping

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import confusion_matrix, RocCurveDisplay, PrecisionRecallDisplay
from sklearn import set_config

set_config(transform_output="pandas")
```

UNSW SYDNEY

Lecture Outline

- **Binary Classification**
- Multiclass Classification
- Dense Layers in Matrices
- Optimisation
- Loss and Derivatives

Stroke Prediction Data description

1. `id`: unique identifier
2. `gender`: “Male”, “Female” or “Other”
3. `age`: age of the patient
4. `hypertension`: 0 or 1 if the patient has hypertension
5. `heart_disease`: 0 or 1 if the patient has any heart disease
6. `ever_married`: “No” or “Yes”
7. `work_type`: “children”, “Govt_jov”, “Never_worked”, “Private” or “Self-employed”
8. `Residence_type`: “Rural” or “Urban”
9. `avg_glucose_level`: average glucose level in blood
10. `bmi`: body mass index
11. `smoking_status`: “formerly smoked”, “never smoked”, “smokes” or “Unknown”
12. `stroke`: 0 or 1 if the patient had a stroke

Source: Kaggle, Stroke Prediction Dataset.

UNSW SYDNEY

Load up the (pre-)preprocessed data

```
PROCESSED_DATA_DIR = Path("stroke/processed")

X_train = pd.read_csv(PROCESSED_DATA_DIR / "x_train.csv")
X_val= pd.read_csv(PROCESSED_DATA_DIR / "x_val.csv")
X_test = pd.read_csv(PROCESSED_DATA_DIR / "x_test.csv")
y_train = pd.read_csv(PROCESSED_DATA_DIR / "y_train.csv")
y_val = pd.read_csv(PROCESSED_DATA_DIR / "y_val.csv")
y_test = pd.read_csv(PROCESSED_DATA_DIR / "y_test.csv")

X_train
```

	gender_Female	gender_Male	ever_married_No	ever_married_Yes	Residence_type_Rural	Residence_type_U
0	0.0	1.0	0.0	1.0	1.0	0.0
1	0.0	1.0	1.0	0.0	1.0	0.0
2	0.0	1.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...
3063	1.0	0.0	0.0	1.0	1.0	0.0
3064	1.0	0.0	0.0	1.0	1.0	0.0
3065	0.0	1.0	1.0	0.0	0.0	1.0

3066 rows × 20 columns

UNSW SYDNEY

Target variable

```
1 y_train
```

	stroke
0	0
1	0
2	0
...	...
3063	0
3064	0
3065	0

3066 rows × 1 columns

```
1 classes, counts = np.unique(y_train.values.ravel(), return_counts=True)
2 print("Classes:", classes)
3 print("Counts:", counts)
```

```
Classes: [0 1]
Counts: [2909  157]
```

Setup a binary classification model

```python
def create_model(seed=42):
    random.seed(seed)
    model = Sequential()
    model.add(Input(X_train.shape[1:]))
    model.add(Dense(32, "leaky_relu"))
    model.add(Dense(16, "leaky_relu"))
    model.add(Dense(1, "sigmoid"))
    return model
```

```python
model = create_model()
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	672
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17

Total params: 1,217 (4.75 KB)
Trainable params: 1,217 (4.75 KB)
Non-trainable params: 0 (0.00 B)

UNSW SYDNEY

Fit the model

```
1 model = create_model()
2 model.compile("adam", "binary_crossentropy")
3 model.fit(X_train, y_train, epochs=5, verbose=2)
```

```
Epoch 1/5
96/96 - 0s - 2ms/step - loss: 0.2734
Epoch 2/5
96/96 - 0s - 2ms/step - loss: 0.1753
Epoch 3/5
96/96 - 0s - 2ms/step - loss: 0.1665
Epoch 4/5
96/96 - 0s - 2ms/step - loss: 0.1619
Epoch 5/5
96/96 - 0s - 2ms/step - loss: 0.1595
```

```
<keras.src.callbacks.history.History at 0x127102900>
```

UNSW SYDNEY

Track accuracy as the model trains

```
model = create_model()
model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=5, verbose=2)
```

```
Epoch 1/5
96/96 - 0s - 2ms/step - accuracy: 0.9204 - loss: 0.2711
Epoch 2/5
96/96 - 0s - 2ms/step - accuracy: 0.9488 - loss: 0.1766
Epoch 3/5
96/96 - 0s - 2ms/step - accuracy: 0.9488 - loss: 0.1667
Epoch 4/5
96/96 - 0s - 2ms/step - accuracy: 0.9488 - loss: 0.1623
Epoch 5/5
96/96 - 0s - 2ms/step - accuracy: 0.9488 - loss: 0.1595
```

```
<keras.src.callbacks.history.History at 0x126e8afd0>
```

UNSW SYDNEY

Run a long fit

```
1  model = create_model()
2  model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
3  %time hist = model.fit(X_train, y_train, epochs=500, validation_data=(X_val, y_val), verb
```

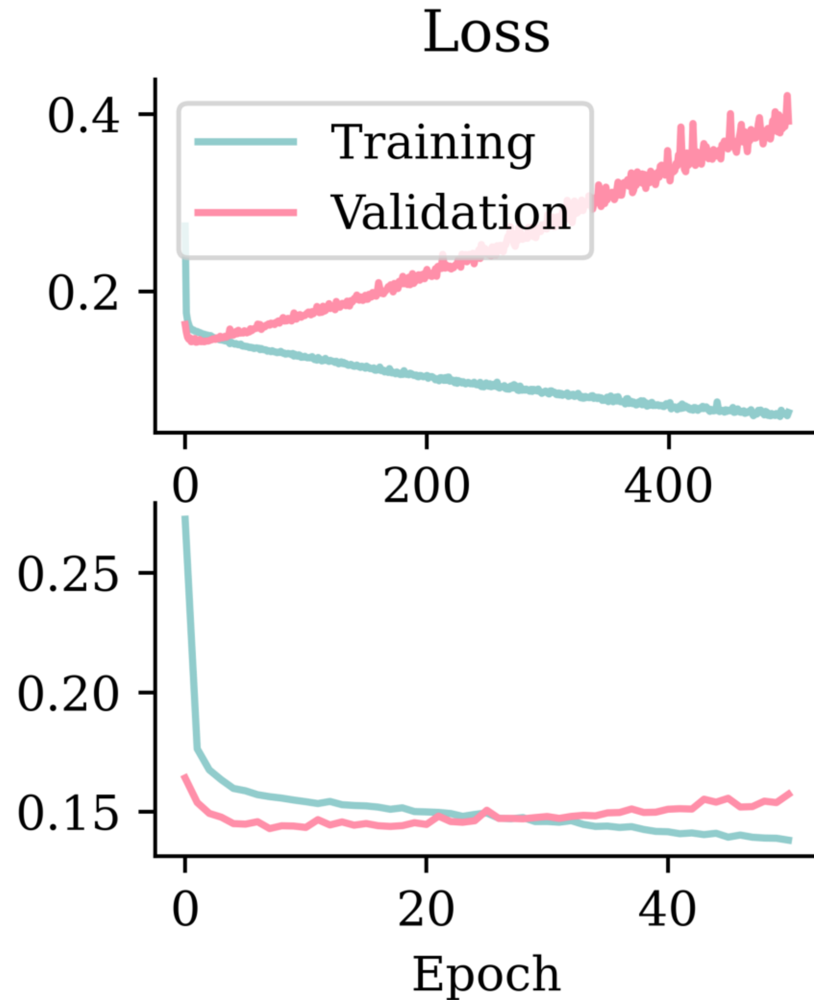
```
CPU times: user 2min 3s, sys: 9.19 s, total: 2min 12s
Wall time: 2min 6s
```

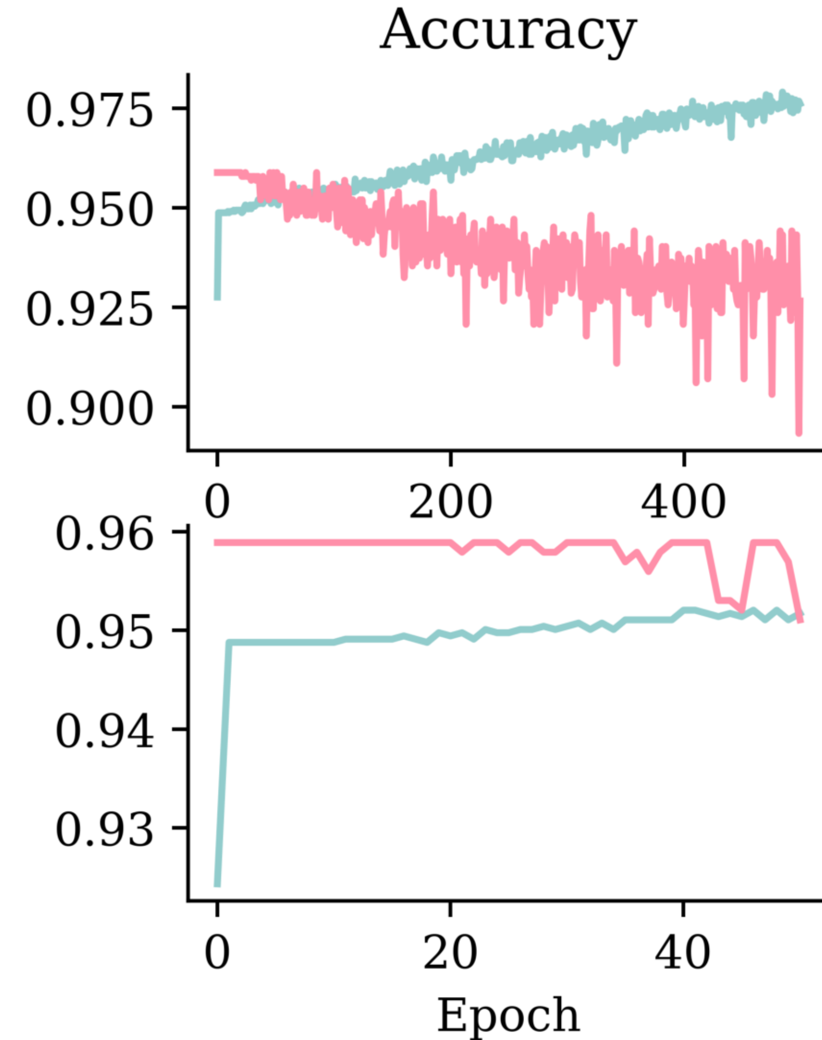
Add early stopping

```
1  model = create_model()
2  model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
3  es = EarlyStopping(restore_best_weights=True, patience=50, monitor="val_accuracy")
4  %time hist_es = model.fit(X_train, y_train, epochs=500, validation_data=(X_val, y_val), c
5  print(f"Stopped after {len(hist_es.history['loss'])} epochs.")
```

```
CPU times: user 12.7 s, sys: 940 ms, total: 13.6 s
Wall time: 13 s
Stopped after 51 epochs.
```

Fitting metrics





UNSW SYDNEY

Add metrics, compile, and fit

```python
model = create_model()

pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
model.compile(optimizer="adam", loss="binary_crossentropy",
    metrics=[pr_auc, "accuracy", "auc"])

es = EarlyStopping(patience=50, restore_best_weights=True,
    monitor="val_pr_auc", verbose=1)
model.fit(X_train, y_train, callbacks=[es], epochs=1_000, verbose=0,
  validation_data=(X_val, y_val));
```

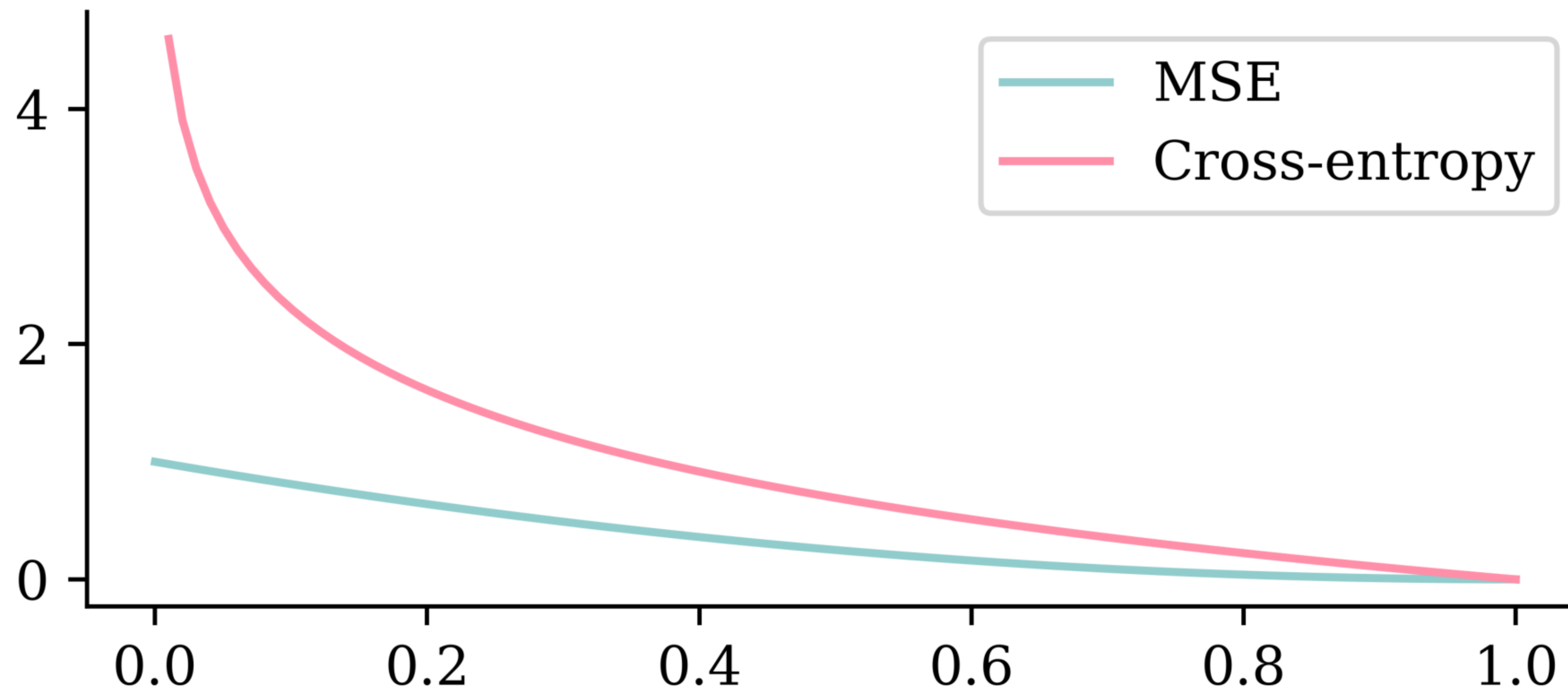
```
Epoch 81: early stopping
Restoring model weights from the end of the best epoch: 31.
```

```python
model.evaluate(X_val, y_val, verbose=0)
```

```
[0.14898666739463806,
 0.12857568264007568,
 0.9569471478462219,
 0.8119411468505859]
```

Why use cross-entropy loss?

```
1 p = np.linspace(0, 1, 100)
2 plt.plot(p, (1 - p) ** 2)
3 plt.plot(p, -np.log(p))
4 plt.legend(["MSE", "Cross-entropy"]);
```



UNSW SYDNEY

Overweight the minority class

```
model = create_model()

pr_auc = keras.metrics.AUC(curve="PR", name="pr_auc")
model.compile(optimizer="adam", loss="binary_crossentropy",
    metrics=[pr_auc, "accuracy", "auc"])

es = EarlyStopping(patience=50, restore_best_weights=True,
    monitor="val_pr_auc", verbose=1)
model.fit(X_train, y_train.to_numpy(), callbacks=[es], epochs=1_000, verbose=0,
  validation_data=(X_val, y_val), class_weight={0: 1, 1: 10});
```

```
Epoch 64: early stopping
Restoring model weights from the end of the best epoch: 14.
```

```
model.evaluate(X_val, y_val, verbose=0)
```

```
[0.3523019552230835,
 0.13380154967308044,
 0.7896282076835632,
 0.8259596824645996]
```

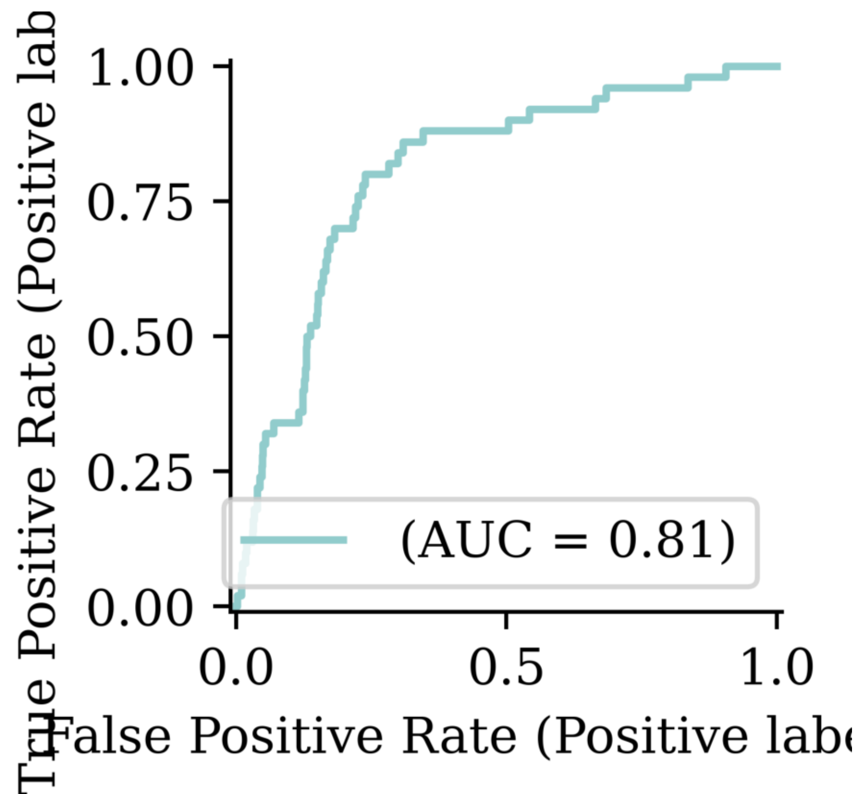
```
model.evaluate(X_test, y_test, verbose=0)
```

```
[0.36996063590049744,
 0.15842117369174957,
 0.7954990267753601,
 0.8060390949249268]
```

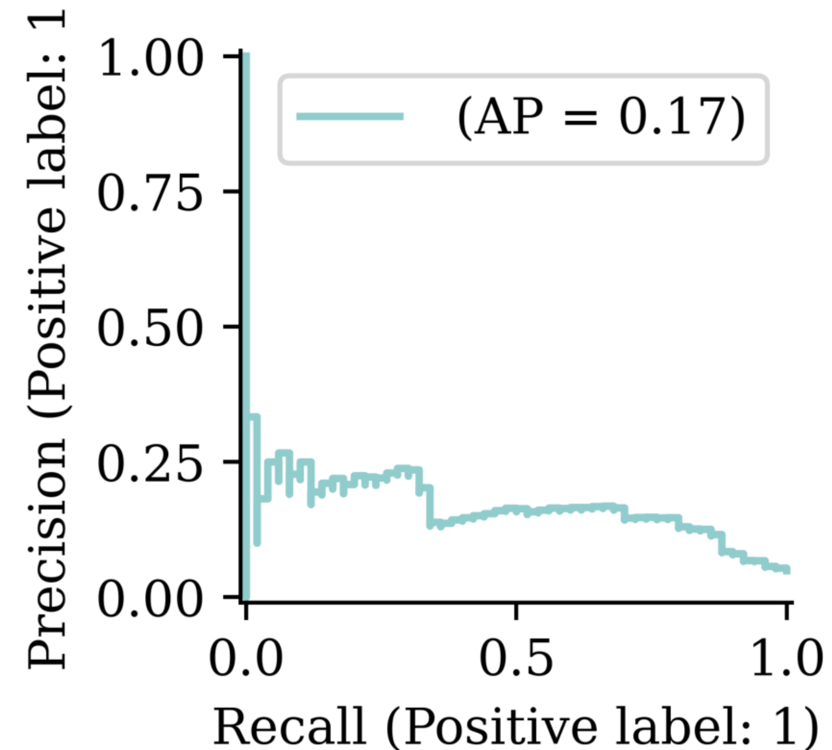
Classification Metrics

```
1  y_pred = model.predict(X_test, verbose=0)
```

```
1  RocCurveDisplay.from_predictions(y_test, y_pred, name="");
```




```
1  PrecisionRecallDisplay.from_predictions(y_test, y_pred, name
```




```
1  y_pred_stroke = y_pred > 0.5
2  confusion_matrix(y_test, y_pred_stroke)
```

```
array([[778, 194],
       [ 15,  35]])
```

```
1  y_pred_stroke = y_pred > 0.3
2  confusion_matrix(y_test, y_pred_stroke)
```

```
array([[647, 325],
       [  7,  43]])
```

Lecture Outline

- Binary Classification
- **Multiclass Classification**
- Dense Layers in Matrices
- Optimisation
- Loss and Derivatives

UNSW SYDNEY

Iris dataset

```python
iris = load_iris()
names = ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth"]
features = pd.DataFrame(iris.data, columns=names)
features
```

	SepalLength	SepalWidth	PetalLength	PetalWidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
…	…	…	…	…
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

UNSW SYDNEY

Target variable

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'],
dtype='<U10')
```

```
iris.target[:8]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0])
```

```
target = iris.target
target = target.reshape(-1, 1)
target[:8]
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0]])
```

```
classes, counts = np.unique(
        target,
        return_counts=True
)
print(classes)
print(counts)
```

```
[0 1 2]
[50 50 50]
```

```
iris.target_names[
  target[[0, 30, 60]]
]
```

```
array([['setosa'],
       ['setosa'],
       ['versicolor']], dtype='<U10')
```

Split the data into train and test

```
1 X_train, X_test, y_train, y_test = train_test_split(features, target, random_state=24)
2 X_train
```

	SepalLength	SepalWidth	PetalLength	PetalWidth
53	5.5	2.3	4.0	1.3
58	6.6	2.9	4.6	1.3
95	5.7	3.0	4.2	1.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
87	6.3	2.3	4.4	1.3
131	7.9	3.8	6.4	2.0

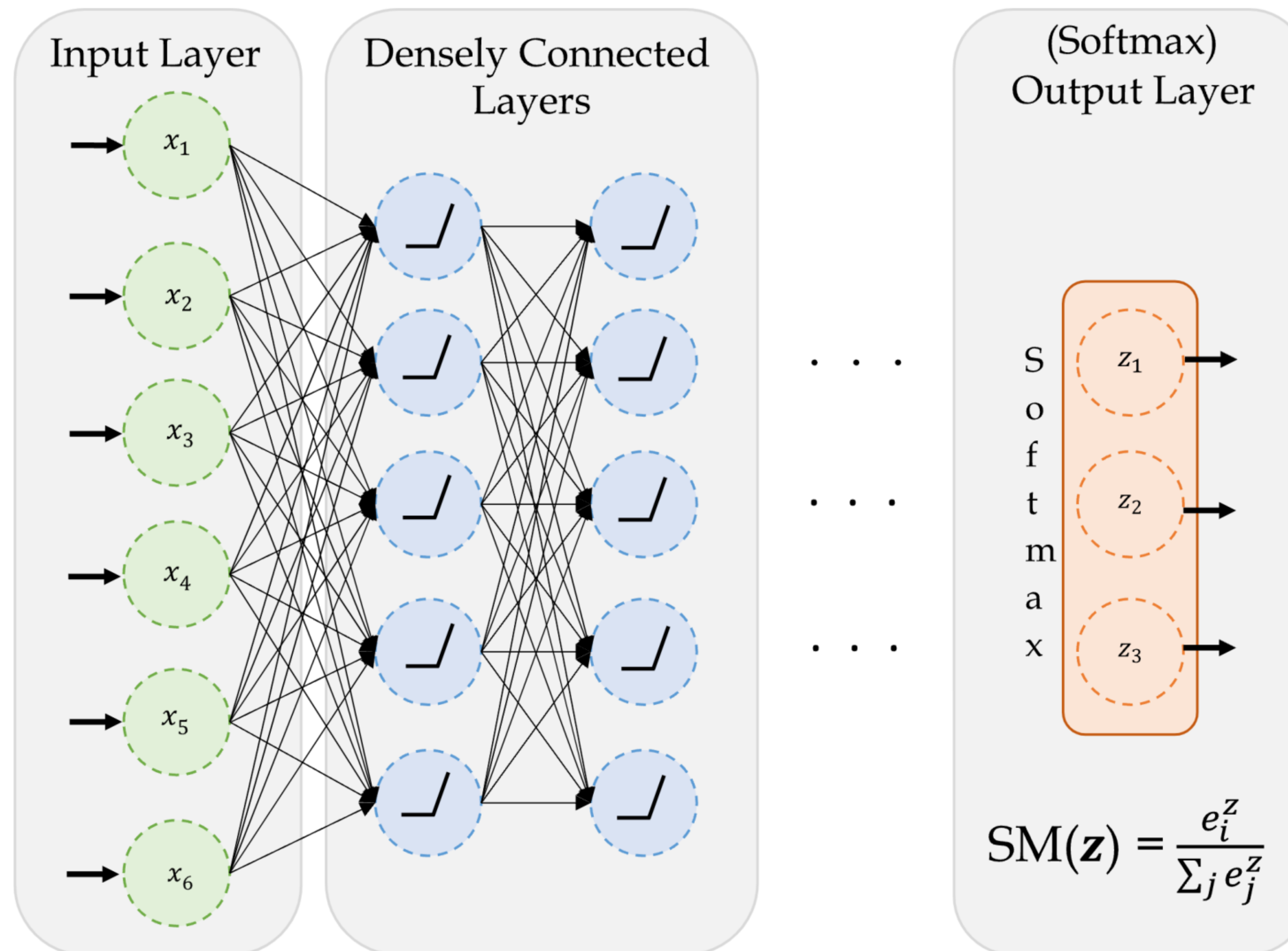
112 rows × 4 columns

```
1 X_test.shape, y_test.shape
```

```
((38, 4), (38, 1))
```

UNSW SYDNEY

A basic classifier network



A basic network for classifying into three categories.

Source: Marcus Lautier (2022).

Create a classifier model

```python
NUM_FEATURES = len(features.columns)
NUM_CATS = len(np.unique(target))

print("Number of features:", NUM_FEATURES)
print("Number of categories:", NUM_CATS)
```

```
Number of features: 4
Number of categories: 3
```

Make a function to return a Keras model:

```python
def build_model(seed=42):
    random.seed(seed)
    return Sequential([
        Dense(30, activation="relu"),
        Dense(NUM_CATS, activation="softmax")
    ])
```

Fit the model

```python
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy")

model.fit(X_train, y_train, epochs=5, verbose=2);
```

```
Epoch 1/5
4/4 - 0s - 2ms/step - loss: 1.3920
Epoch 2/5
4/4 - 0s - 2ms/step - loss: 1.2912
Epoch 3/5
4/4 - 0s - 2ms/step - loss: 1.2196
Epoch 4/5
4/4 - 0s - 2ms/step - loss: 1.1576
Epoch 5/5
4/4 - 0s - 2ms/step - loss: 1.1084
```

UNSW SYDNEY

Track accuracy as the model trains

```python
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=5, verbose=2);
```

```
Epoch 1/5
4/4 - 0s - 2ms/step - accuracy: 0.2857 - loss: 1.3930
Epoch 2/5
4/4 - 0s - 2ms/step - accuracy: 0.2857 - loss: 1.2970
Epoch 3/5
4/4 - 0s - 2ms/step - accuracy: 0.2857 - loss: 1.2203
Epoch 4/5
4/4 - 0s - 2ms/step - accuracy: 0.2946 - loss: 1.1596
Epoch 5/5
4/4 - 0s - 2ms/step - accuracy: 0.3393 - loss: 1.1067
```

UNSW SYDNEY

Run a long fit

```
model = build_model()
model.compile("adam", "sparse_categorical_crossentropy", \
        metrics=["accuracy"])
%time hist = model.fit(X_train, y_train, epochs=500, \
        validation_split=0.25, verbose=False)
```

```
CPU times: user 3.84 s, sys: 466 ms, total: 4.31 s
Wall time: 4 s
```

Evaluation now returns both *loss* and *accuracy*.

```
model.evaluate(X_test, y_test, verbose=False)
```

```
[0.08639740198850632, 0.9736841917037964]
```

Add early stopping

```python
model_es = build_model()
model_es.compile("adam", "sparse_categorical_crossentropy", \
        metrics=["accuracy"])

es = EarlyStopping(restore_best_weights=True, patience=50,
        monitor="val_accuracy")
%time hist_es = model_es.fit(X_train, y_train, epochs=500, \
        validation_split=0.25, callbacks=[es], verbose=False);

print(f"Stopped after {len(hist_es.history['loss'])} epochs.")
```

```
CPU times: user 533 ms, sys: 65.6 ms, total: 599 ms
Wall time: 555 ms
Stopped after 70 epochs.
```

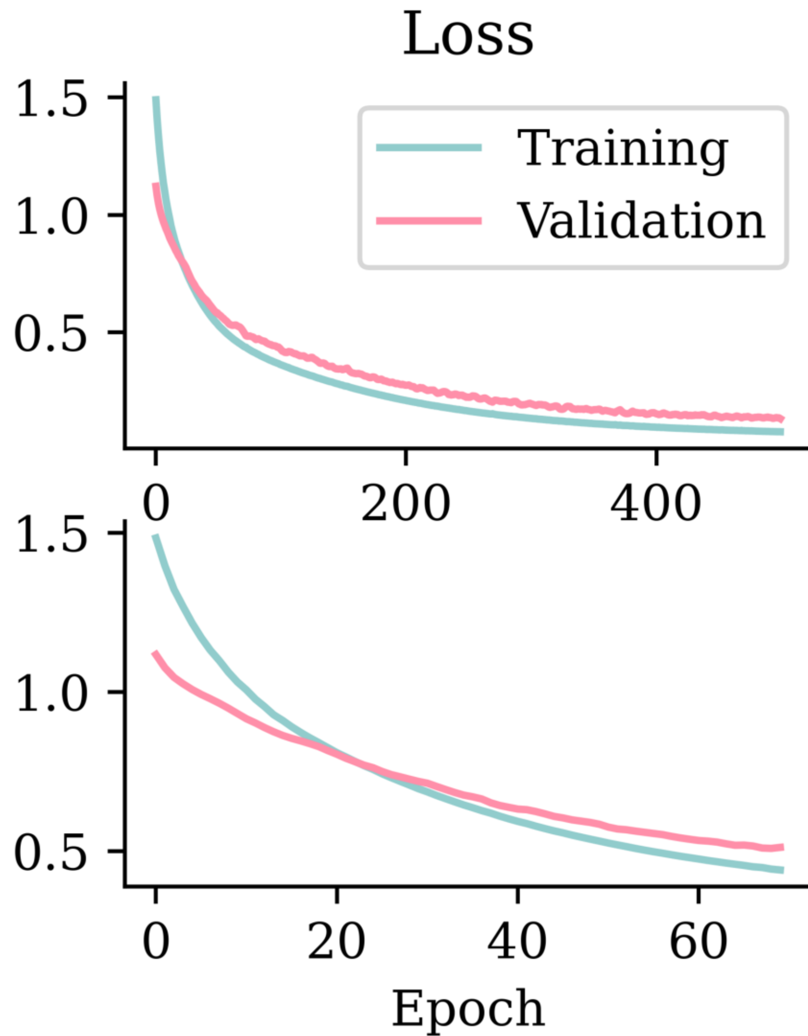
Evaluation on test set:

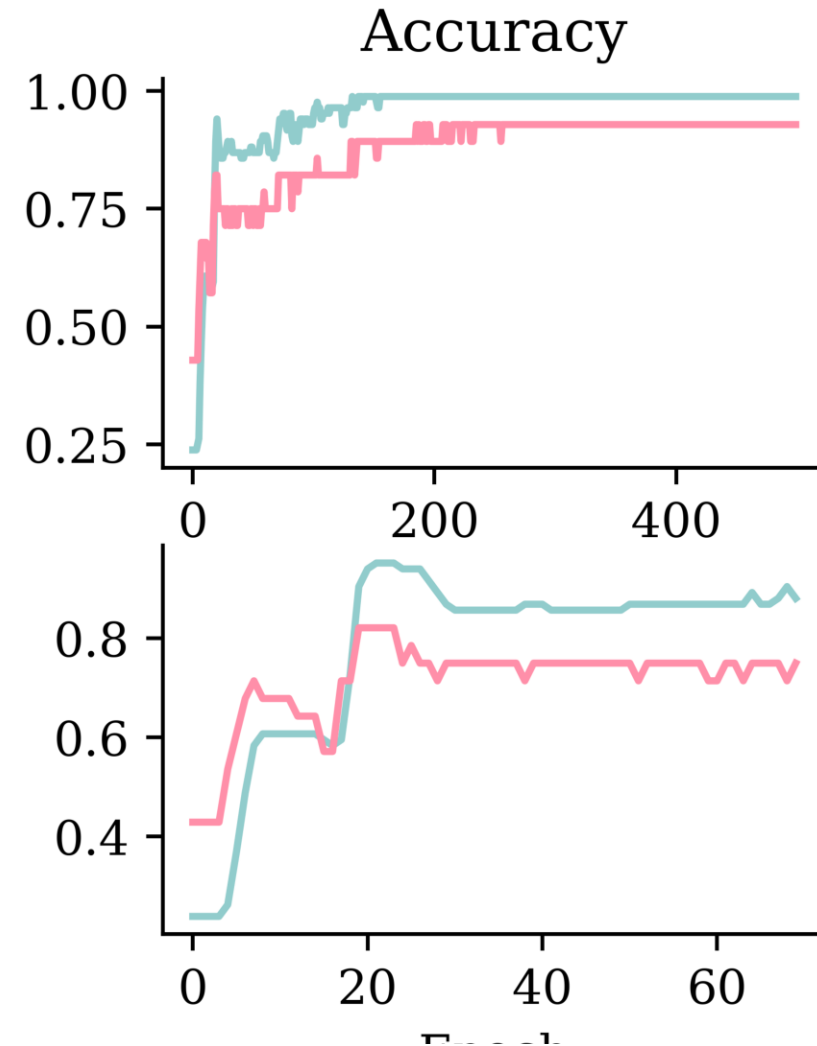
```python
model_es.evaluate(X_test, y_test, verbose=False)
```

```
[0.8077937960624695, 0.9210526347160339]
```

UNSW SYDNEY

Fitting metrics





What is the softmax activation?

It creates a “probability” vector: $\text{Softmax}(\boldsymbol{x})_i = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}}$.

In NumPy:

```
1  out = np.array([5, -1, 6])
2  (np.exp(out) / np.exp(out).sum()).round(3)
```

```
array([0.27, 0.   , 0.73])
```

In Keras:

```
1  out = keras.ops.convert_to_tensor([[5.0, -1.0, 6.0]])
2  keras.ops.round(keras.ops.softmax(out), 3)
```

```
tensor([[0.2690, 0.0010, 0.7310]])
```

UNSW SYDNEY

Prediction using classifiers

```python
y_test[:4]
```

```
array([[2],
       [2],
       [1],
       [1]])
```

```python
y_pred = model.predict(X_test.head(4), verbose=0)
y_pred
```

```
array([[2.02e-06, 7.64e-02, 9.24e-01],
       [1.86e-07, 1.62e-03, 9.98e-01],
       [1.44e-02, 9.76e-01, 1.00e-02],
       [2.80e-03, 8.50e-01, 1.48e-01]], dtype=float32)
```

```python
# Add 'keepdims=True' to get a column vector.
np.argmax(y_pred, axis=1)
```

```
array([2, 2, 1, 1])
```

```python
iris.target_names[np.argmax(y_pred, axis=1)]
```

```
array(['virginica', 'virginica', 'versicolor', 'versicolor'], dtype='<U10')
```

Summary: Classification models in Keras

If the number of classes is c, then:

Target	**Output Layer**	**Loss Function**
Binary ($c = 2$)	1 neuron with `sigmoid` activation	Binary Cross-Entropy
Multi-class ($c > 2$)	c neurons with `softmax` activation	Categorical Cross-Entropy

Summary: Optionally output logits

If the number of classes is c, then:

Target	Output Layer	Loss Function
Binary ($c = 2$)	1 neuron with `linear` activation	Binary Cross-Entropy (`from_logits=True`)
Multi-class ($c > 2$)	c neurons with `linear` activation	Categorical Cross-Entropy (`from_logits=True`)

Summary: Code examples

Binary

```
1 model = Sequential([
2   # Skipping the earlier layers
3   Dense(1, activation="sigmoid")
4 ])
5 model.compile(loss="binary_crossentropy")
```

Multi-class

```
1 model = Sequential([
2   # Skipping the earlier layers
3   Dense(n_classes, activation="softmax")
4 ])
5 model.compile(loss="sparse_categorical_crossentropy")
```

Binary (logits)

```
1 model = Sequential([
2   # Skipping the earlier layers
3   Dense(1, activation="linear")
4 ])
5 loss = BinaryCrossentropy(from_logits=True)
6 model.compile(loss=loss)
```

Multi-class (logits)

```
1 model = Sequential([
2   # Skipping the earlier layers
3   Dense(n_classes, activation="linear")
4 ])
5 loss = SparseCategoricalCrossentropy(from_logits=True)
6 model.compile(loss=loss)
```

Both `BinaryCrossentropy` and `SparseCategoricalCrossentropy` live in `keras.losses`.

Lecture Outline

- Binary Classification
- Multiclass Classification
- **Dense Layers in Matrices**
- Optimisation
- Loss and Derivatives

Logistic regression

Observations: $\mathbf{x}_{i,\bullet} \in \mathbb{R}^2$.

Target: $y_i \in \{0, 1\}$.

Predict: $\hat{y}_i = \mathbb{P}(Y_i = 1)$.

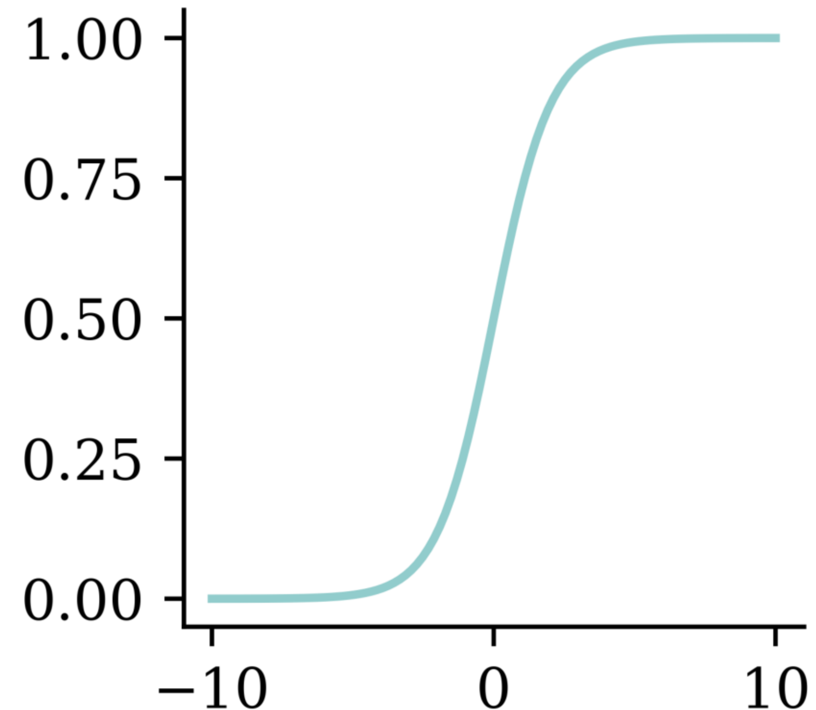
The model

For $\mathbf{x}_{i,\bullet} = (x_{i,1}, x_{i,2})$:

$$z_i = x_{i,1} w_1 + x_{i,2} w_2 + b$$

$$\hat{y}_i = \sigma(z_i) = \frac{1}{1 + \mathrm{e}^{-z_i}}.$$

```python
x = np.linspace(-10, 10, 100)
y = 1/(1 + np.exp(-x))
plt.plot(x, y);
```



Multiple observations

```
1 data = pd.DataFrame({"x_1": [1, 3, 5], "x_2": [2, 4, 6], "y": [0, 1, 1]})
2 data
```

	x_1	x_2	y
0	1	2	0
1	3	4	1
2	5	6	1

Let $w_1 = 1$, $w_2 = 2$ and $b = -10$.

```
1 w_1 = 1; w_2 = 2; b = -10
2 data["x_1"] * w_1 + data["x_2"] * w_2 + b
```

```
0   -5
1    1
2    7
dtype: int64
```

Matrix notation

Have $\mathbf{X} \in \mathbb{R}^{3\times 2}$.

```
1  X_df = data[["x_1", "x_2"]]
2  X = X_df.to_numpy()
3  X
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Let $\mathbf{w} = (w_1, w_2)^\top \in \mathbb{R}^{2\times 1}$.

```
1  w = np.array([[1], [2]])
2  w
```

```
array([[1],
       [2]])
```

$$\mathbf{z} = \mathbf{X}\mathbf{w} + b, \quad \mathbf{a} = \sigma(\mathbf{z})$$

```
1  z = X.dot(w) + b
2  z
```

```
array([[-5],
       [ 1],
       [ 7]])
```

```
1  1 / (1 + np.exp(-z))
```

```
array([[0.01],
       [0.73],
       [1.  ]])
```

UNSW SYDNEY

Using a softmax output

Observations: $\mathbf{x}_{i,\bullet} \in \mathbb{R}^2$. Predict: $\hat{y}_{i,j} = \mathbb{P}(Y_i = j)$.

Target: $\mathbf{y}_{i,\bullet} \in \{(1,0),(0,1)\}$.

The model: For $\mathbf{x}_{i,\bullet} = (x_{i,1}, x_{i,2})$

$$z_{i,1} = x_{i,1} w_{1,1} + x_{i,2} w_{2,1} + b_1 ,$$
$$z_{i,2} = x_{i,1} w_{1,2} + x_{i,2} w_{2,2} + b_2 .$$

$$\hat{y}_{i,1} = \text{Softmax}_1(\mathbf{z}_i) = \frac{\mathrm{e}^{z_{i,1}}}{\mathrm{e}^{z_{i,1}} + \mathrm{e}^{z_{i,2}}},$$
$$\hat{y}_{i,2} = \text{Softmax}_2(\mathbf{z}_i) = \frac{\mathrm{e}^{z_{i,2}}}{\mathrm{e}^{z_{i,1}} + \mathrm{e}^{z_{i,2}}}.$$

UNSW SYDNEY

Multiple observations

```
1 data
```

	x_1	x_2	y_1	y_2
0	1	2	1	0
1	3	4	0	1
2	5	6	0	1

Choose:

$w_{1,1} = 1$, $w_{2,1} = 2$,

$w_{1,2} = 3$, $w_{2,2} = 4$, and

$b_1 = -10$, $b_2 = -20$.

```
1 w_11 = 1; w_21 = 2; b_1 = -10
2 w_12 = 3; w_22 = 4; b_2 = -20
3 data["x_1"] * w_11 + data["x_2"] * w_21 + b_1
```

```
0   -5
1    1
2    7
dtype: int64
```

Matrix notation

Have $\mathbf{X} \in \mathbb{R}^{3\times 2}$.

```
1  X
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

$\mathbf{W} \in \mathbb{R}^{2\times 2}$, $\mathbf{b} \in \mathbb{R}^2$

```
1  W = np.array([[1, 3], [2, 4]])
2  b = np.array([-10, -20])
3  display(W); b
```

```
array([[1, 3],
       [2, 4]])
```

```
array([-10, -20])
```

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{b}, \quad \mathbf{A} = \text{Softmax}(\mathbf{Z}).$$

```
1  Z = X @ W + b
2  Z
```

```
array([[-5, -9],
       [ 1,  5],
       [ 7, 19]])
```

```
1  np.exp(Z) / np.sum(np.exp(Z),
2    axis=1, keepdims=True)
```

```
array([[9.82e-01, 1.80e-02],
       [1.80e-02, 9.82e-01],
       [6.14e-06, 1.00e+00]])
```

UNSW SYDNEY

Lecture Outline

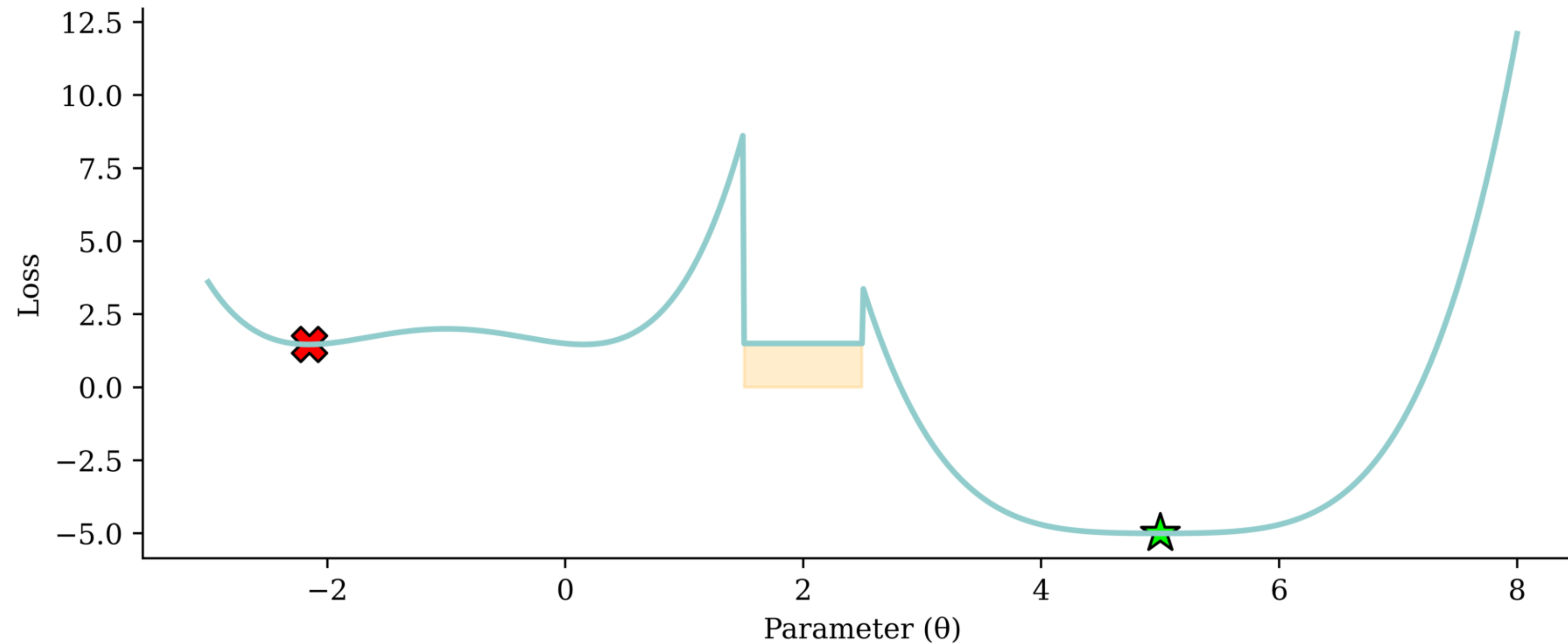
- Binary Classification
- Multiclass Classification
- Dense Layers in Matrices
- **Optimisation**
- Loss and Derivatives

Gradient-based learning

In-class demo

UNSW SYDNEY

Gradient descent pitfalls



UNSW SYDNEY

Go over all the training data

Called *batch gradient descent*.

```
for i in range(num_epochs):
    gradient = evaluate_gradient(loss_function, data, weights)
    weights = weights - learning_rate * gradient
```

UNSW SYDNEY

Pick a random training example

Called *stochastic gradient descent*.

```
for i in range(num_epochs):
    rnd.shuffle(data)
    for example in data:
        gradient = evaluate_gradient(loss_function, example, weights)
        weights = weights - learning_rate * gradient
```

Take a group of training examples

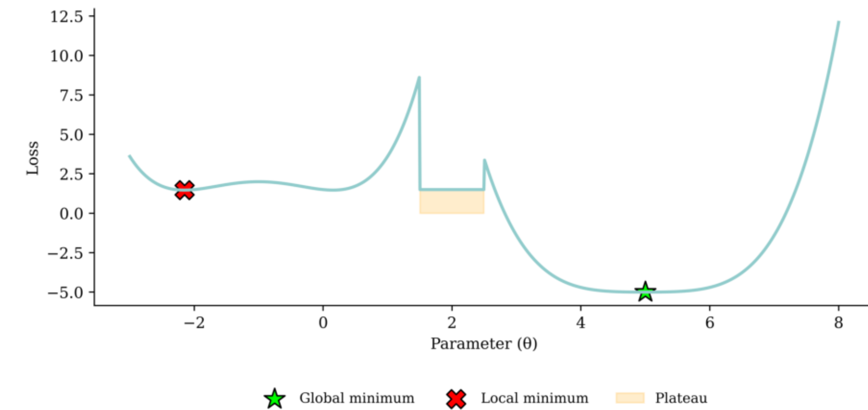
Called *mini-batch gradient descent*.

```
for i in range(num_epochs):
    rnd.shuffle(data)
    for b in range(num_batches):
        batch = data[b * batch_size : (b + 1) * batch_size]
        gradient = evaluate_gradient(loss_function, batch, weights)
        weights = weights - learning_rate * gradient
```

Mini-batch gradient descent

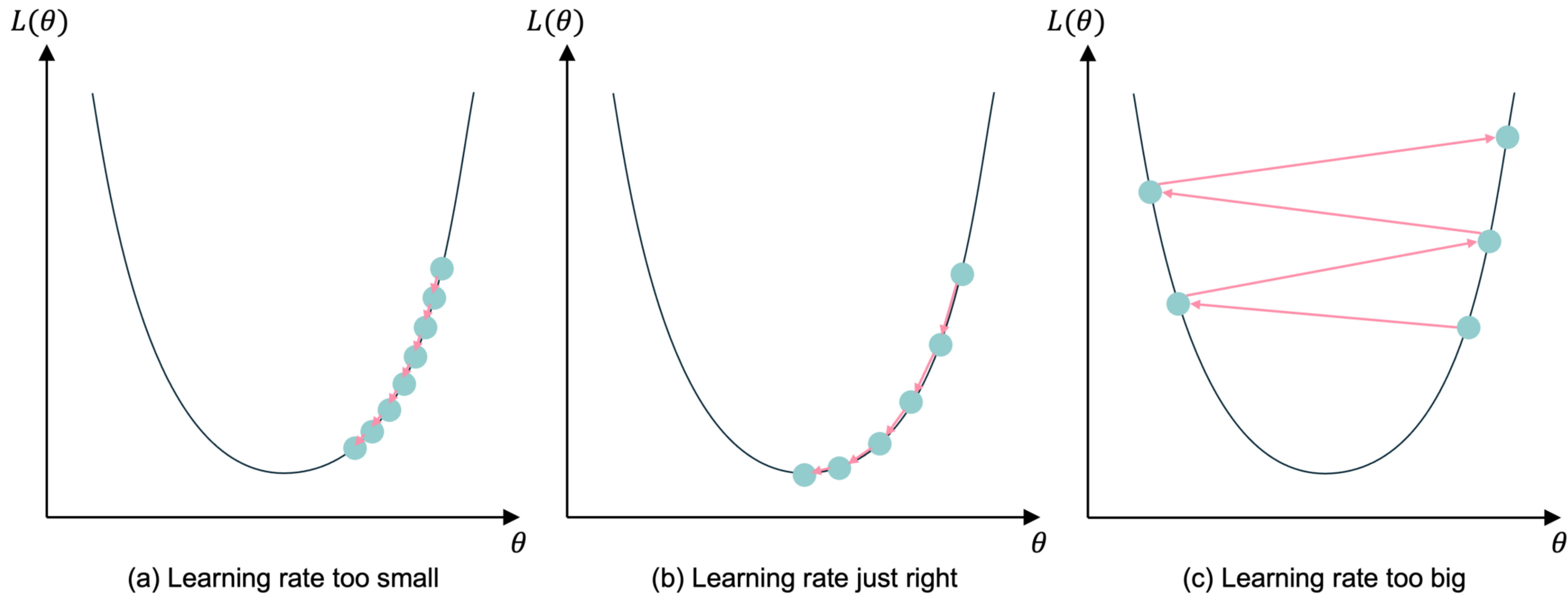
Why?

1. Because we have to (data is too big to shove it all in a single batch)
2. Because it is faster (lots of quick noisy steps *takes longer* than a few slow super accurate steps)
3. The noise helps us jump out of local minima



Noisy gradient means we might jump out of a local minimum.

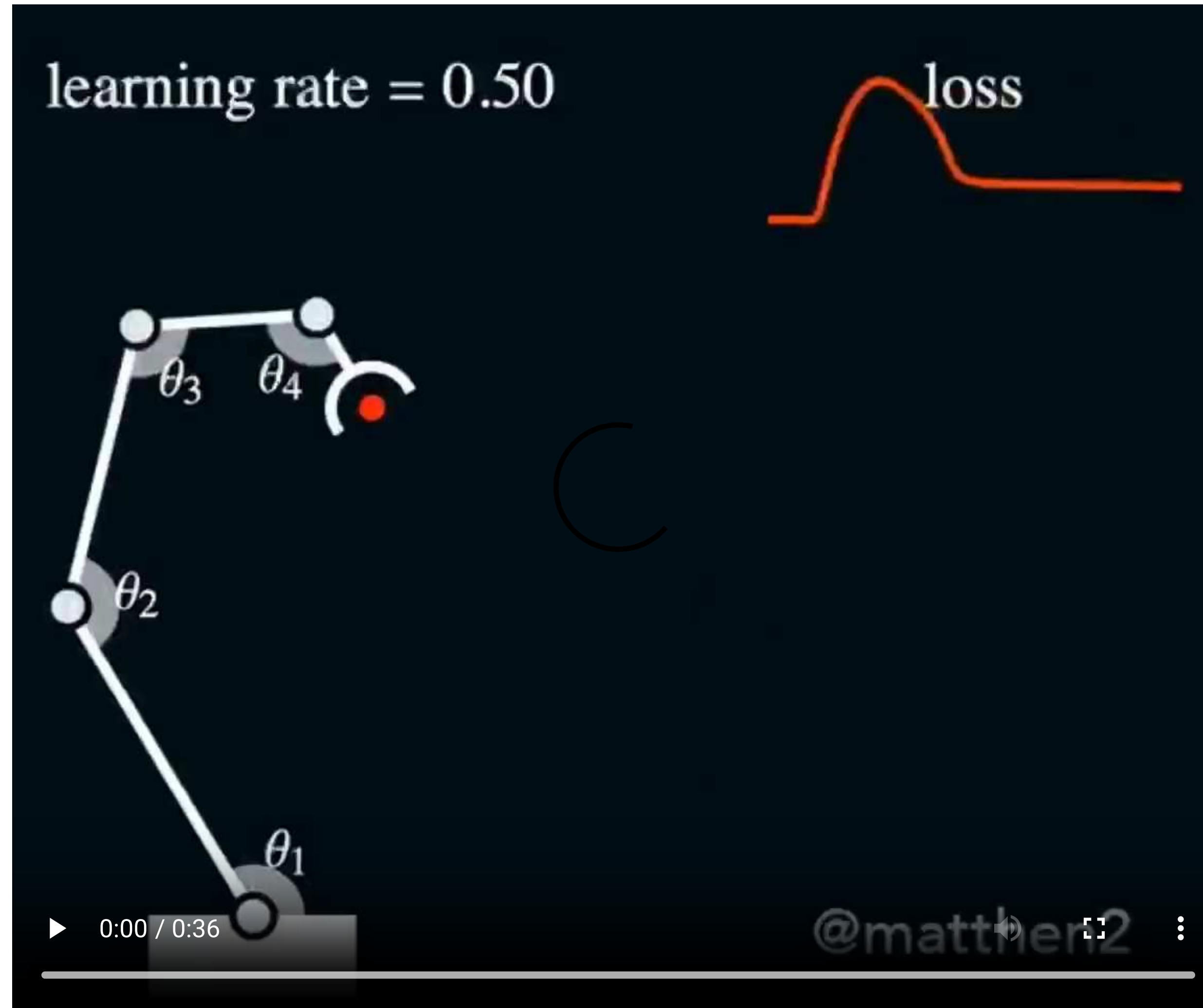
Learning rates



Gradient descent with different learning rates

Source: Melissa Renard (2025), adapted from Jeremy Jordan (2018), *Setting the learning rate of your neural network*.

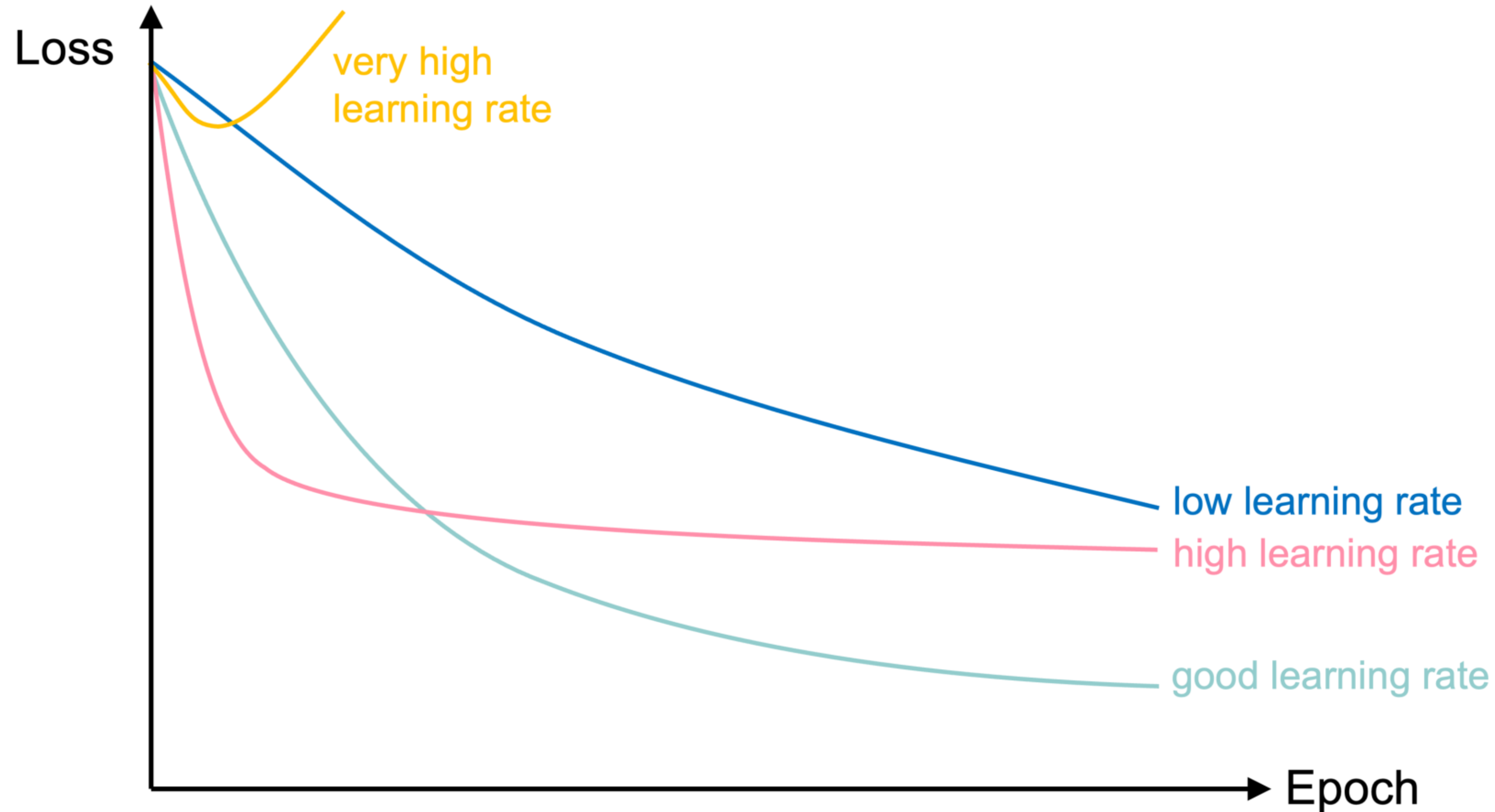
Learning rates #2



Changing the learning rates for a robot arm.

Source: Matt Henderson (2021), X post

Learning rate schedule



Learning curves for various learning rates

In training the learning rate may be tweaked manually.

Source: Melissa Renard (2025), adapted from Deep Learning for Computer Vision – Stanford Spring 2025

UNSW SYDNEY

Lecture Outline

- Binary Classification
- Multiclass Classification
- Dense Layers in Matrices
- Optimisation
- **Loss and Derivatives**

Example: linear regression

$$\hat{y}(x) = wx + b$$

For some observation $\{x_i, y_i\}$, the squared error loss is

$$\text{Loss}_i = (\hat{y}(x_i) - y_i)^2$$

For a batch of the first n observations the MSE loss is

$$\text{Loss}_{1:n} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}(x_i) - y_i)^2$$

UNSW SYDNEY

Derivatives

Since $\hat{y}(x) = wx + b$,

$$\frac{\partial \hat{y}(x)}{\partial w} = x \text{ and } \frac{\partial \hat{y}(x)}{\partial b} = 1.$$

As $\text{Loss}_i = (\hat{y}(x_i) - y_i)^2$, we know

$$\frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} = 2(\hat{y}(x_i) - y_i).$$

UNSW SYDNEY

Chain rule

$$\frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} = 2(\hat{y}(x_i) - y_i), \quad \frac{\partial \hat{y}(x)}{\partial w} = x, \quad \text{and} \quad \frac{\partial \hat{y}(x)}{\partial b} = 1.$$

Putting this together, we have

$$\frac{\partial \text{Loss}_i}{\partial w} = \frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} \times \frac{\partial \hat{y}(x_i)}{\partial w} = 2(\hat{y}(x_i) - y_i)\, x_i$$

and

$$\frac{\partial \text{Loss}_i}{\partial b} = \frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} \times \frac{\partial \hat{y}(x_i)}{\partial b} = 2(\hat{y}(x_i) - y_i).$$

We need non-zero derivatives

This is why can’t use accuracy as the loss function for classification.

Also why we can have the *dead ReLU* problem.

Stochastic gradient descent (SGD)

Start with $\boldsymbol{\theta}_0 = (w, b)^\top = (0, 0)^\top$.

Randomly pick $i = 5$, say $x_i = 5$ and $y_i = 5$.

$$\hat{y}(x_i) = 0 \times 5 + 0 = 0 \Rightarrow \text{Loss}_i = (0 - 5)^2 = 25.$$

The partial derivatives are

$$\frac{\partial \text{Loss}_i}{\partial w} = 2(\hat{y}(x_i) - y_i)\, x_i = 2 \cdot (0 - 5) \cdot 5 = -50, \text{ and}$$
$$\frac{\partial \text{Loss}_i}{\partial b} = 2(0 - 5) = -10.$$

The gradient is $\nabla \text{Loss}_i = (-50, -10)^\top$.

UNSW SYDNEY

SGD, first iteration

Start with $\boldsymbol{\theta}_0 = (w, b)^\top = (0, 0)^\top$.

Randomly pick $i = 5$, say $x_i = 5$ and $y_i = 5$.

The gradient is $\nabla \text{Loss}_i = (-50, -10)^\top$.

Use learning rate $\eta = 0.01$ to update

$$\begin{aligned}\boldsymbol{\theta}_1 &= \boldsymbol{\theta}_0 - \eta \nabla \text{Loss}_i \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 0.01 \begin{pmatrix} -50 \\ -10 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}.\end{aligned}$$

UNSW SYDNEY

SGD, second iteration

Start with $\boldsymbol{\theta}_1 = (w, b)^\top = (0.5, 0.1)^\top$.

Randomly pick $i = 9$, say $x_i = 9$ and $y_i = 17$.

The gradient is $\nabla \text{Loss}_i = (-223.2, -24.8)^\top$.

Use learning rate $\eta = 0.01$ to update

$$\begin{aligned}
\boldsymbol{\theta}_2 &= \boldsymbol{\theta}_1 - \eta \nabla \text{Loss}_i \\
&= \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} - 0.01 \begin{pmatrix} -223.2 \\ -24.8 \end{pmatrix} \\
&= \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} + \begin{pmatrix} 2.232 \\ 0.248 \end{pmatrix} = \begin{pmatrix} 2.732 \\ 0.348 \end{pmatrix}.
\end{aligned}$$

UNSW SYDNEY

Batch gradient descent (BGD)

For the first n observations $\text{Loss}_{1:n} = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}_i$ so

$$\frac{\partial \text{Loss}_{1:n}}{\partial w} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \text{Loss}_i}{\partial w} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \text{Loss}_i}{\hat{y}(x_i)} \frac{\partial \hat{y}(x_i)}{\partial w}$$
$$= \frac{1}{n}\sum_{i=1}^{n} 2(\hat{y}(x_i) - y_i)\, x_i.$$

$$\frac{\partial \text{Loss}_{1:n}}{\partial b} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \text{Loss}_i}{\partial b} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \text{Loss}_i}{\hat{y}(x_i)} \frac{\partial \hat{y}(x_i)}{\partial b}$$
$$= \frac{1}{n}\sum_{i=1}^{n} 2(\hat{y}(x_i) - y_i).$$

UNSW SYDNEY

BGD, first iteration ($\boldsymbol{\theta}_0 = \mathbf{0}$)

	x	y	y_hat	loss	dL/dw	dL/db
0	1	0.99	0	0.98	-1.98	-1.98
1	2	3.00	0	9.02	-12.02	-6.01
2	3	5.01	0	25.15	-30.09	-10.03

So $\nabla \text{Loss}_{1:3}$ is

```python
nabla = np.array([df["dL/dw"].mean(), df["dL/db"].mean()])
nabla
```

```
array([-14.69,  -6.  ])
```

so with $\eta = 0.1$ then $\boldsymbol{\theta}_1$ becomes

```python
theta_1 = theta_0 - 0.1 * nabla
theta_1
```

```
array([1.47, 0.6 ])
```

BGD, second iteration

	x	y	y_hat	loss	dL/dw	dL/db
0	1	0.99	2.07	1.17	2.16	2.16
1	2	3.00	3.54	0.29	2.14	1.07
2	3	5.01	5.01	0.00	-0.04	-0.01

So $\nabla \text{Loss}_{1:3}$ is

```
nabla = np.array([df["dL/dw"].mean(), df["dL/db"].mean()])
nabla
```

```
array([1.42, 1.07])
```

so with $\eta = 0.1$ then $\boldsymbol{\theta}_2$ becomes

```
theta_2 = theta_1 - 0.1 * nabla
theta_2
```

```
array([1.33, 0.49])
```

UNSW SYDNEY

Package Versions

```python
from watermark import watermark
print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch"
```

```
Python implementation: CPython
Python version       : 3.14.5
IPython version      : 9.13.0

keras     : 3.14.1
matplotlib: 3.10.9
numpy     : 2.4.4
pandas    : 3.0.2
seaborn   : 0.13.2
scipy     : 1.17.1
torch     : 2.11.0
```

Recommended viewing

Some very easy-to-follow explanations of these topics, plus catchy tunes:

- Softmax and argmax
- Cross entropy
- Cross entropy derivatives and backpropagation

Glossary

- accuracy
- classification problem
- confusion matrix
- cross-entropy loss
- metrics
- sigmoid activation function
- softmax activation
- batch gradient descent
- batches, batch size
- global minimum, local minimum
- gradient-based learning, hill-climbing
- learning rate, learning rate schedule
- plateau
- stochastic gradient descent
- mini-batch gradient descent

UNSW SYDNEY