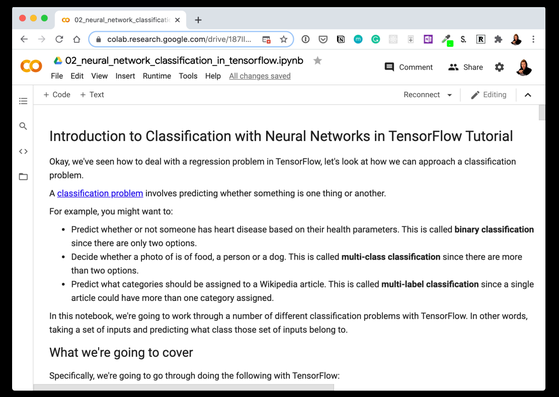
# Neural Network Classification with

TensorFlow

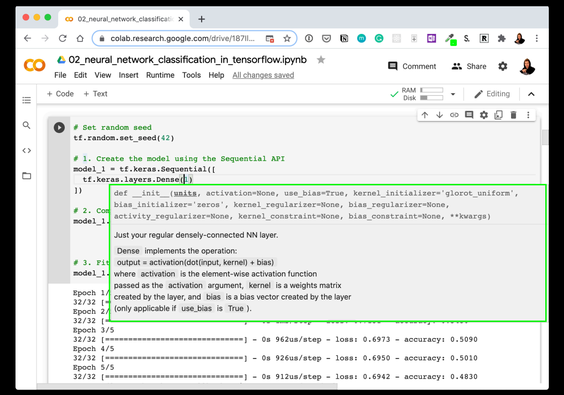
# Where can you get help?

- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask (don't forget the Discord chat!)

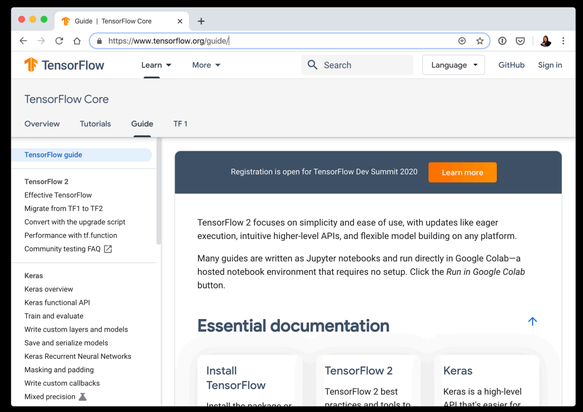
(yes, including the "dumb" questions)




"If in doubt, run the code"







“What is a classification problem?”

# Example classification problems

"Is this email spam or not spam?"

To: daniel@mrdbourke.com
Hey Daniel,

This deep learning course is incredible!
I can't wait to use what I've learned!

Not spam

To: daniel@mrdbourke.com
Hay daniel...

C0ongratu1ations! U win $1139239230

Spam

**Binary classification**

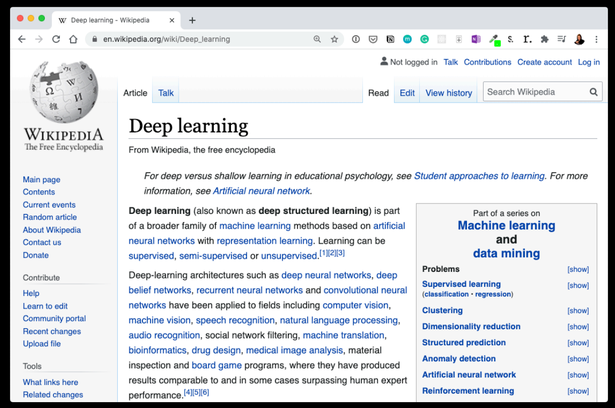
(one thing or another)

"Is this a photo of sushi, steak or pizza?"

**Multiclass classification**

(more than one thing or another)

"What tags should this article have?"




Machine learning
Representation learning
Artificial intelligence

(multiple label options per sample)

**Multilabel classification**

# What we're going to cover

(broadly)

- Architecture of a neural network classification model
- Input shapes and output shapes of a classification model (features and labels)
- Creating custom data to view and fit
- Steps in modelling
  - Creating a model, compiling a model, fitting a model, evaluating a model
- Different classification evaluation methods
- Saving and loading models

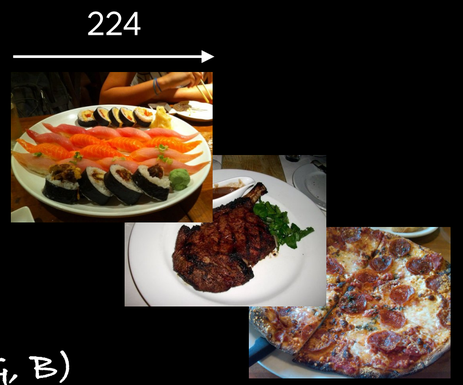
(we'll be cooking up lots of code!)

**How:**

# Classification inputs and outputs

W = 224
H = 224
C = 3

(c = colour channels, R, G, B)




Sushi
Steak
Pizza

**Actual output**

```
[[0.31, 0.62, 0.44…],
[0.92, 0.03, 0.27…],
[0.25, 0.78, 0.07…],
…,
```

(normalized pixel values)

**Numerical encoding**

Inputs

Machine Learning Algorithm

(often already exists, if not, you can build one)

Outputs

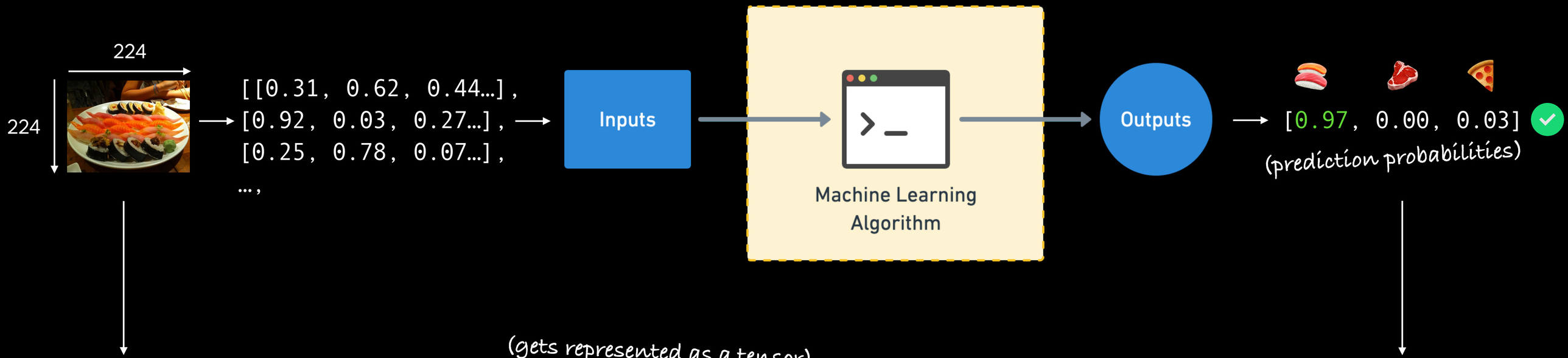
```
[[0.97, 0.00, 0.03],
[0.81, 0.14, 0.05],
[0.03, 0.07, 0.90],
…,
```

**Predicted output**

(comes from looking at lots of these)

# Input and output shapes

(for an image classification example)




(gets represented as a tensor)

`[batch_size, width, height, colour_channels]`

```
Shape = [None, 224, 224, 3]
        or
Shape = [32, 224, 224, 3]
```

(32 is a very common batch size)

These will vary depending on the problem you're working on.

```
Shape = [3]
```

(typical)

# Architecture of a classification model

(we're going to be building lots of these)

| Hyperparameter | Binary Classification | Multiclass classification |
|---|---|---|
| Input layer shape | Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction) | Same as binary classification |
| Hidden layer(s) | Problem specific, minimum = 1, maximum = unlimited | Same as binary classification |
| Neurons per hidden layer | Problem specific, generally 10 to 100 | Same as binary classification |
| Output layer shape | 1 (one class or the other) | 1 per class (e.g. 3 for food, person or dog photo) |
| Hidden activation | Usually ReLU (rectified linear unit) | Same as binary classification |
| Output activation | Sigmoid | Softmax |
| Loss function | Cross entropy ( `tf.keras.losses.BinaryCrossentropy` in TensorFlow) | Cross entropy ( `tf.keras.losses.CategoricalCrossentropy` in TensorFlow) |
| Optimizer | SGD (stochastic gradient descent), Adam | Same as binary classification |

**Source:** Adapted from page 295 of Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book by Aurélien Géron

→

```python
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.Input(shape=(224, 224, 3)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

→

Sushi 🍣
Steak 🥩
Pizza 🍕

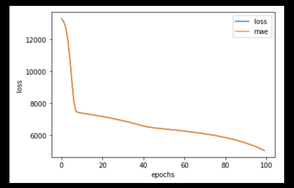
# Let’s code!

# The machine learning explorer's motto

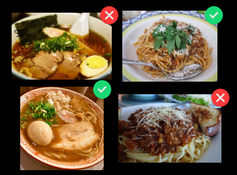
"Visualize, visualize, visualize"

**Data**

**Model**




**Training**



**Predictions**

It's a good idea to visualize these as often as possible.

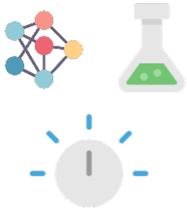
# Steps in modelling with TensorFlow

1. Get data ready (turn into tensors)


2. Build or pick a pretrained model (to suit your problem)

3. Fit the model to the data and make a prediction

4. Evaluate the model



5. Improve through experimentation

6. Save and reload your trained model

```python
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.Input(shape=(224, 224, 3)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

(typical)

# Architecture of a classification model

(we're going to be building lots of these)

| Hyperparameter | Binary Classification | Multiclass classification |
|---|---|---|
| Input layer shape | Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction) | Same as binary classification |
| Hidden layer(s) | Problem specific, minimum = 1, maximum = unlimited | Same as binary classification |
| Neurons per hidden layer | Problem specific, generally 10 to 100 | Same as binary classification |
| Output layer shape | 1 (one class or the other) | 1 per class (e.g. 3 for food, person or dog photo) |
| Hidden activation | Usually ReLU (rectified linear unit) | Same as binary classification |
| Output activation | Sigmoid | Softmax |
| Loss function | Cross entropy ( `tf.keras.losses.BinaryCrossentropy` in TensorFlow) | Cross entropy ( `tf.keras.losses.CategoricalCrossentropy` in TensorFlow) |
| Optimizer | SGD (stochastic gradient descent), Adam | Same as binary classification |

**Source:** Adapted from page 295 of Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book by Aurélien Géron

→

```python
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.Input(shape=(224, 224, 3)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

→

Sushi 🍣
Steak 🥩
Pizza 🍕

# Steps in modelling with TensorFlow

```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.Input(shape=(224, 224, 3)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

1. Construct or import a pretrained model relevant to your problem
2. Compile the model (prepare it to be used with data)
    - Loss — how wrong your model's predictions are compared to the truth labels (you want to minimise this).
    - Optimizer — how your model should update its internal patterns to better its predictions.
    - Metrics — human interpretable values for how well your model is doing.
3. Fit the model to the training data so it can discover patterns
    - Epochs — how many times the model will go through all of the training examples.
4. Evaluate the model on the test data (how reliable are our model's predictions?)

# Improving a model

(from a model's perspective)

```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(4, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_subset, y_train_subset, epochs=5)
```

Smaller model



```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_full, y_train_full, epochs=100)
```

Larger model

## Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate
- Fitting on more data
- Fitting for longer

(because you can alter each of these, they're hyperparameters)

# Improving a model

(from a model's perspective)

```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(4, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_subset, y_train_subset, epochs=5)
```

Smaller model



```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_full, y_train_full, epochs=100)
```

Larger model

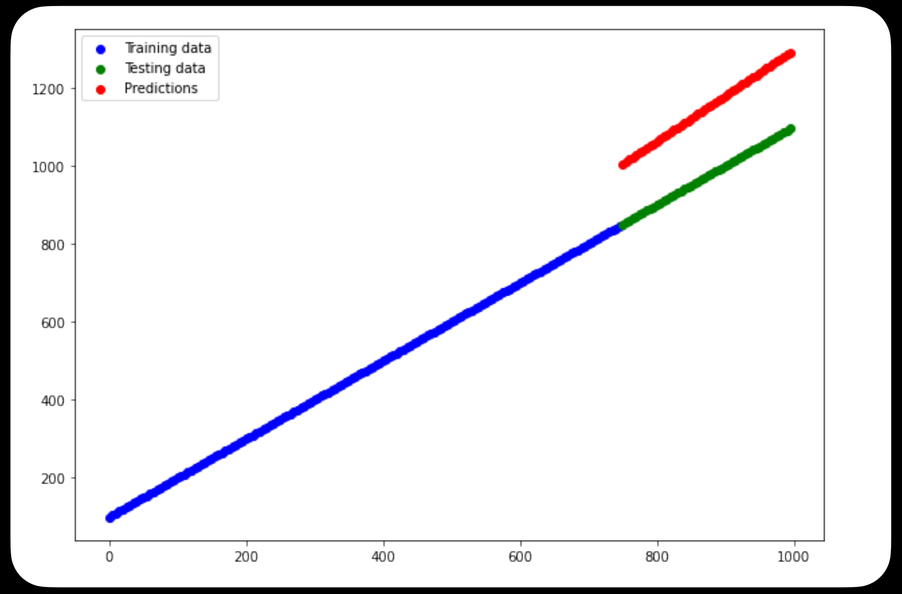
## Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate
- Fitting on more data
- Fitting for longer

(because you can alter each of these, they're hyperparameters)

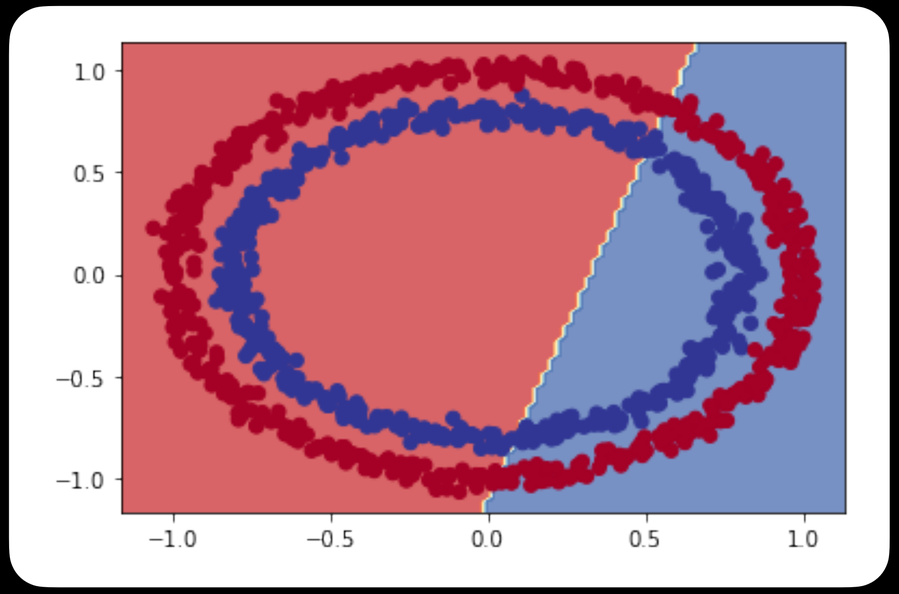
# The missing piece: Non-linearity

🤔 “What could you draw if you had an unlimited amount of straight (linear) and non-straight (non-linear) lines?”




**Linear data**

(possible to model with straight lines)



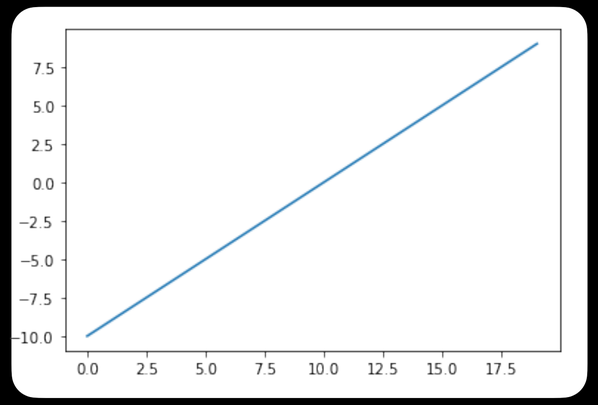

**Non-linear data**

(not possible to model with straight lines)

# The missing piece: Non-linearity

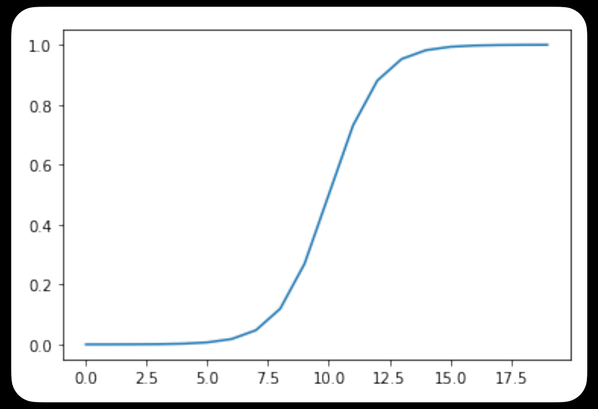
```
A = tf.cast(tf.range(-10, 10), tf.float32)
```

```
tf.keras.activations.linear(A)
```




**Linear activation**

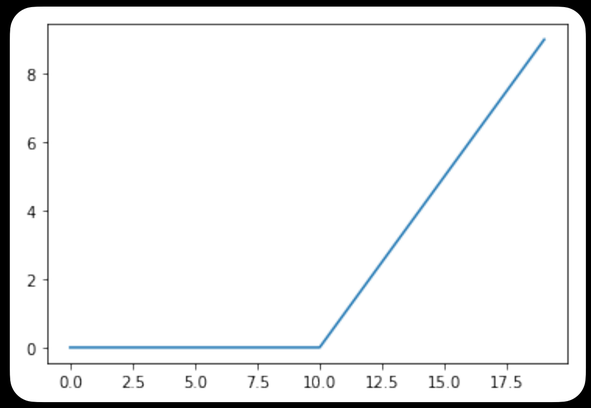
```
tf.keras.activations.sigmoid(A)
```




**Sigmoid activation**

*(non-linear)*

```
tf.keras.activations.relu(A)
```




**ReLU activation**

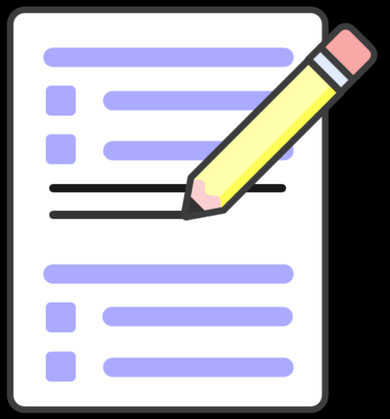
*(non-linear)*

# Three datasets

(possibly the most important concept in machine learning...)

**Course materials (training set)**

**Practice exam (validation set)**



**Final exam (test set)**

**Generalization**

The ability for a machine learning model to perform well on data it hasn't seen before.

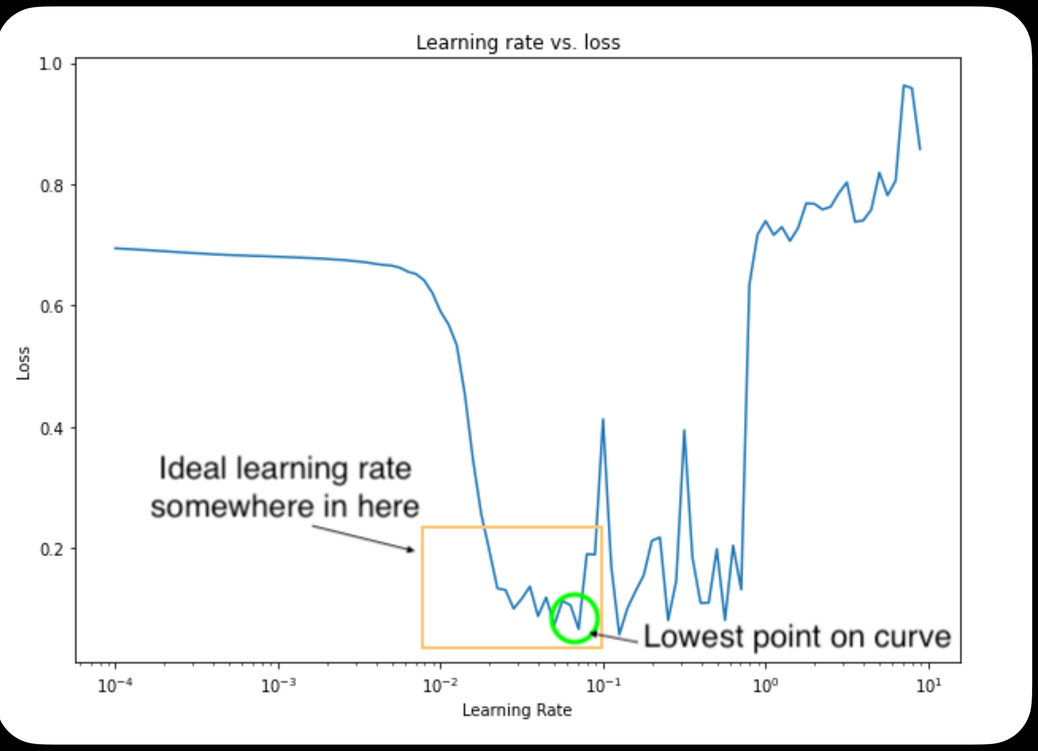
# Finding the ideal learning rate

```
# Model code
...

# Create a learning rate scheduler callback
# (traverse set of learning rate values from 1e-4, increasing by 10**(epoch/20) every epoch)
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-4 * 10**(epoch/20))

# Fit the model (passing the lr_scheduler callback)
history = model.fit(X_train,
                    y_train,
                    epochs=100,
                    callbacks=[lr_scheduler])

# Plot the learning rate versus the loss
lrs = 1e-4 * (10 ** (np.arange(100)/20))
plt.figure(figsize=(10, 7))
plt.semilogx(lrs, history.history["loss"])
plt.xlabel("Learning Rate")
plt.ylabel("Loss")
plt.title("Learning rate vs. loss");
```



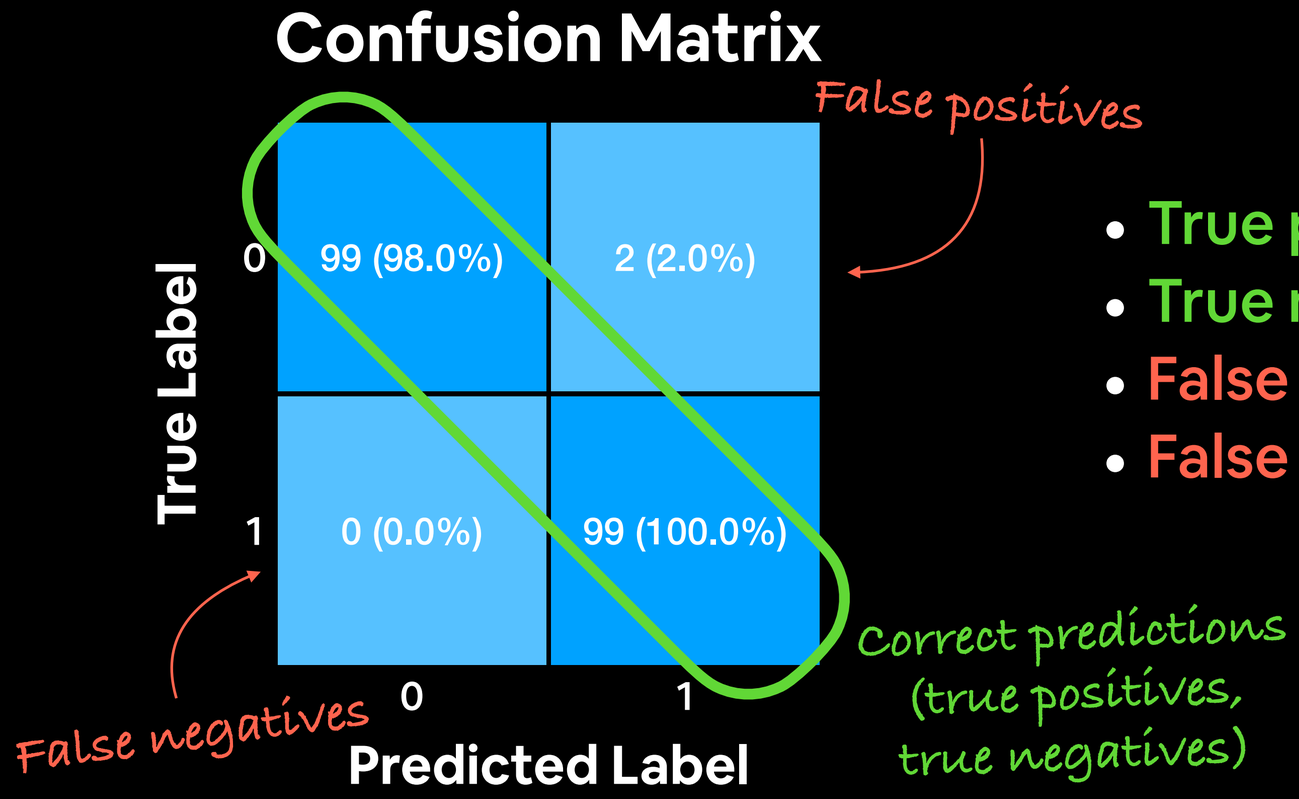
*(some common)*

# Classification evaluation methods

**Key:** **tp** = True Positive, **tn** = True Negative, **fp** = False Positive, **fn** = False Negative

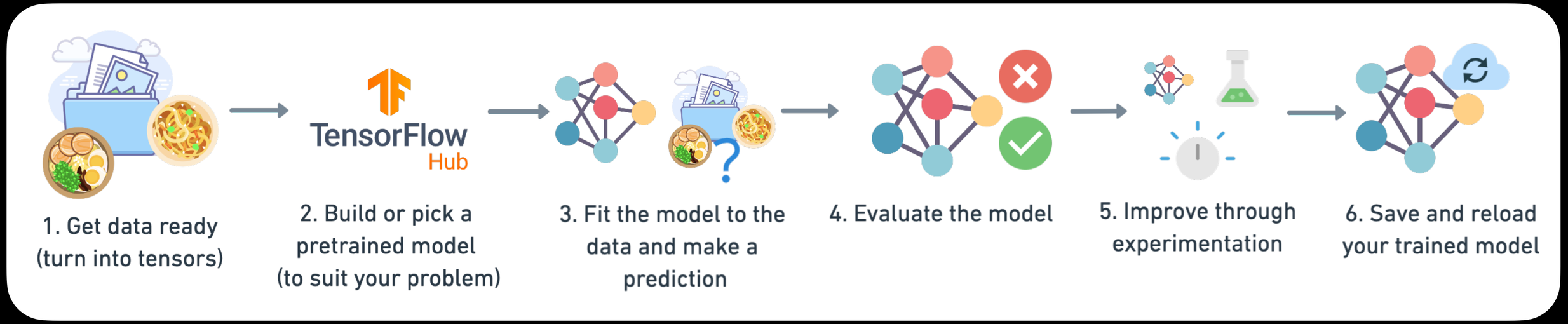
| Metric Name | Metric Forumla | Code | When to use |
|---|---|---|---|
| Accuracy | $\textbf{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$ | `tf.keras.metrics.Accuracy()` or `sklearn.metrics.accuracy_score()` | Default metric for classification problems. Not the best for imbalanced classes. |
| Precision | $\textbf{Precision} = \frac{tp}{tp + fp}$ | `tf.keras.metrics.Precision()` or `sklearn.metrics.precision_score()` | Higher precision leads to less false positives. |
| Recall | $\textbf{Recall} = \frac{tp}{tp + fn}$ | `tf.keras.metrics.Recall()` or `sklearn.metrics.recall_score()` | Higher recall leads to less false negatives. |
| F1-score | $\textbf{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$ | `sklearn.metrics.f1_score()` | Combination of precision and recall, usually a good overall metric for a classification model. |
| Confusion matrix | NA | `Custom function` or `sklearn.metrics.confusion_matrix()` | When comparing predictions to truth labels to see where model gets confused. Can be hard to use with large numbers of classes. |

# Anatomy of a confusion matrix




- True positive = model predicts 1 when truth is 1
- True negative = model predicts 0 when truth is 0
- False positive = model predicts 1 when truth is 0
- False negative = model predicts 0 when truth is 1

# Steps in modelling with TensorFlow




1. Turn all data into numbers (neural networks can't handle strings)

2. Make sure all of your tensors are the right shape (inputs and outputs)

3. Scale features — normalize or standardize, neural networks tend to prefer normalization (values between 0 & 1)

# Inputs and outputs (layer by layer)

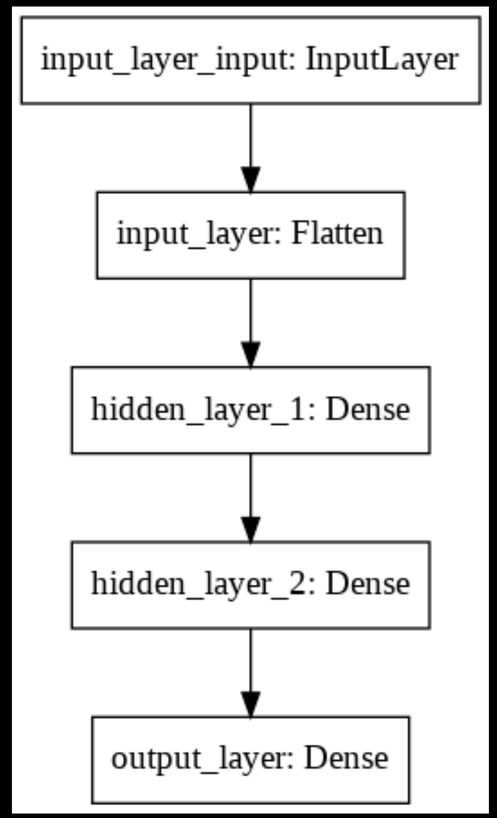
```
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
…,
```

```python
import tensorflow as tf

# Create the model
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28), name="input_layer"),
  tf.keras.layers.Dense(4, activation="relu", name="hidden_layer_1"),
  tf.keras.layers.Dense(4, activation="relu", name="hidden_layer_2"),
  tf.keras.layers.Dense(10, activation="softmax", name="output_layer")
], name="10_class_classification_model")

# Plot the model
tf.keras.utils.plot_model(model)
```

Takes inputs →




Outputs to

Outputs to

Outputs to

Outputs to

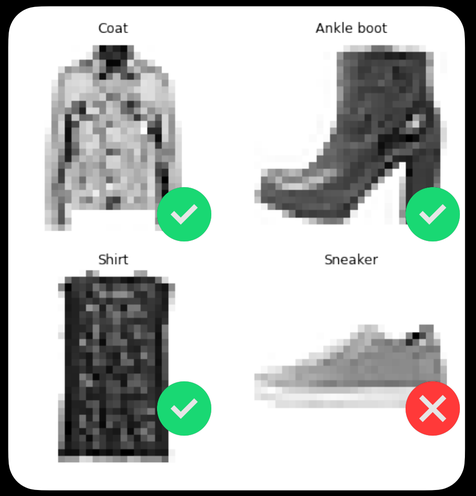
Outputs to

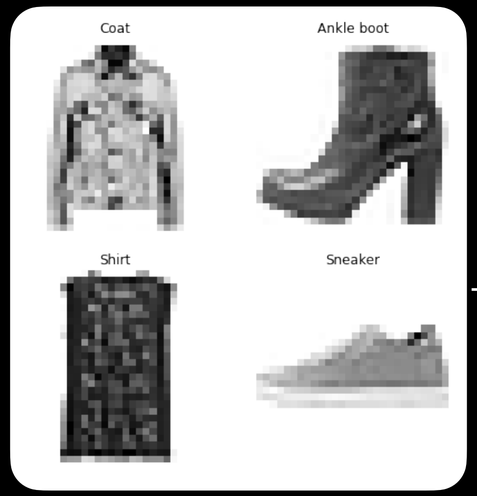
```
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,
```

2. Show examples

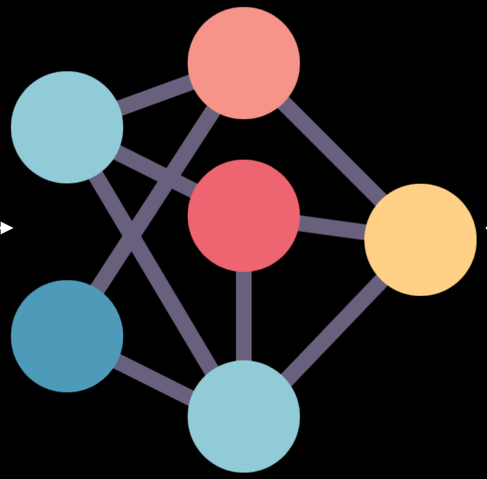
1. Initialise with random weights (only at beginning)

```
[[0.092, 0.210, 0.415],
[0.778, 0.929, 0.030],
[0.019, 0.182, 0.555],
…,
```




```
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
…,
```



```
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,
```

Coat,
Ankle boot,
Shirt,
Sandal

3. Update representation outputs (weights & biases)

4. Repeat with more examples

| Inputs | Numerical encoding | Learns representation (patterns/features/weights) | Representation outputs | Outputs |
|---|---|---|---|---|

# The machine learning practitioner's motto

"Experiment, experiment, experiment"


(try lots of things and see what tastes good)