Imperial College
London

8th November 2018

Deep learning: applications to biological data

`https://bitbucket.org/mfumagal/statistical_inference`

Matteo Fumagalli

Imperial College London

Intended Learning Outcomes

By the end of this session, you will be able to:

- Describe the three key components of a classifier: score function, loss function, optimisation
- Identify the elements of a neural networks, including neurons and hyper-parameters
- Illustrate the specific layers in a neural network for visual recognition
- Appreciate the use of deep learning to solve biological problems
- Demonstrate how to implement, train and evaluate a deep neural network in `python`

Imperial College London

Who is the *deepest learner*?

It's a competition!

The challenge: predict whether a species is endangered, vulnerable or of least concern from genomic data.

Ursus arctos marsicanus

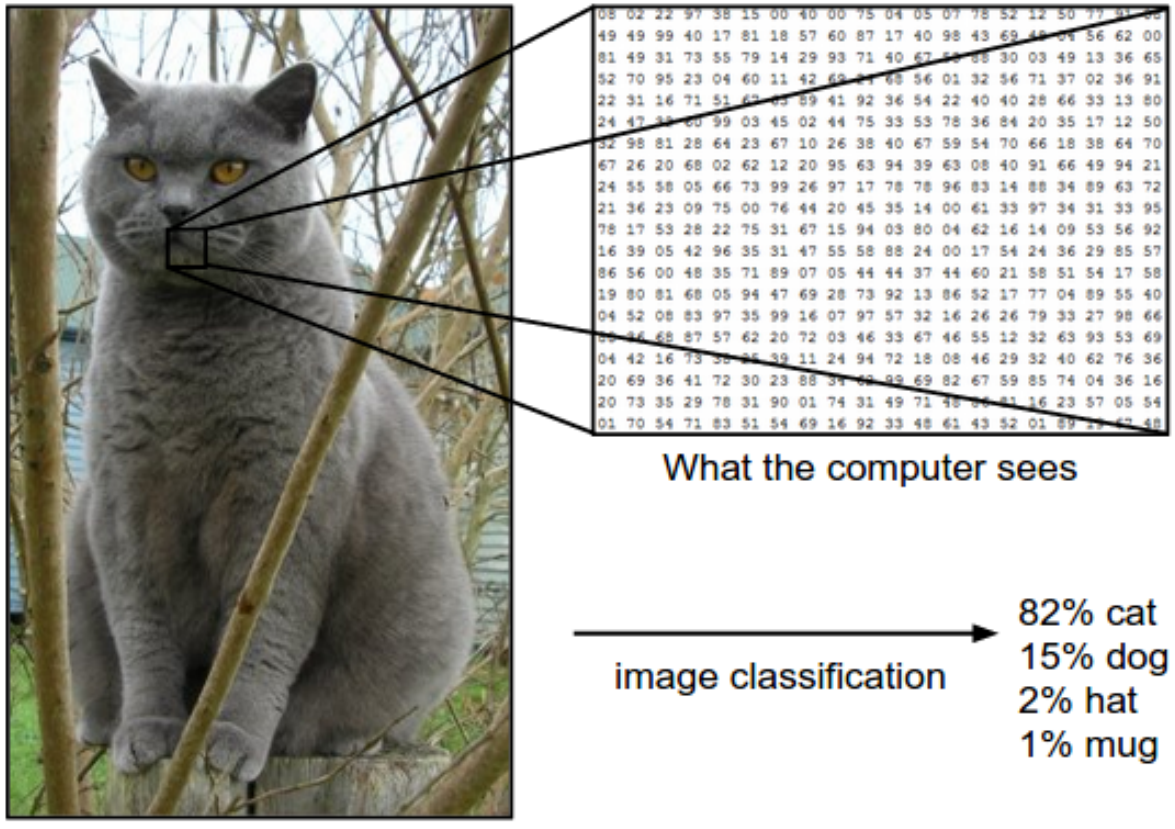
The score to beat: 75% by me.
The prize: TBA

Imperial College London

What do you see?

Imperial College London

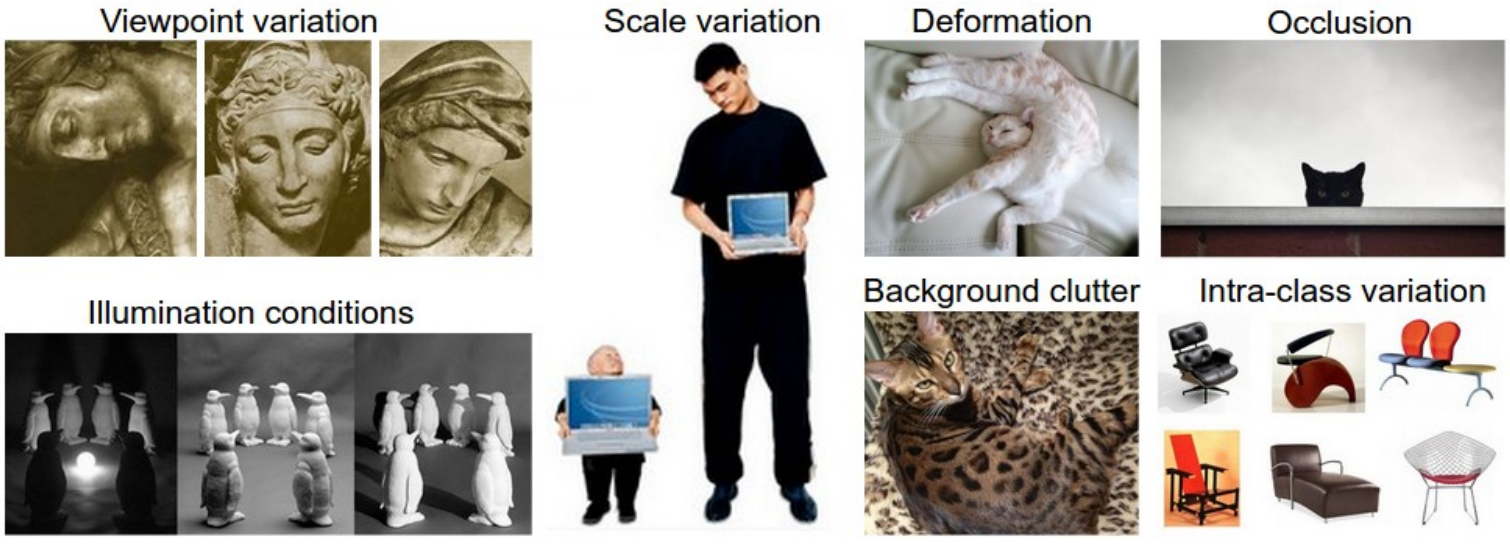
What does the computer see?



Is it THAT difficult?

Imperial College London

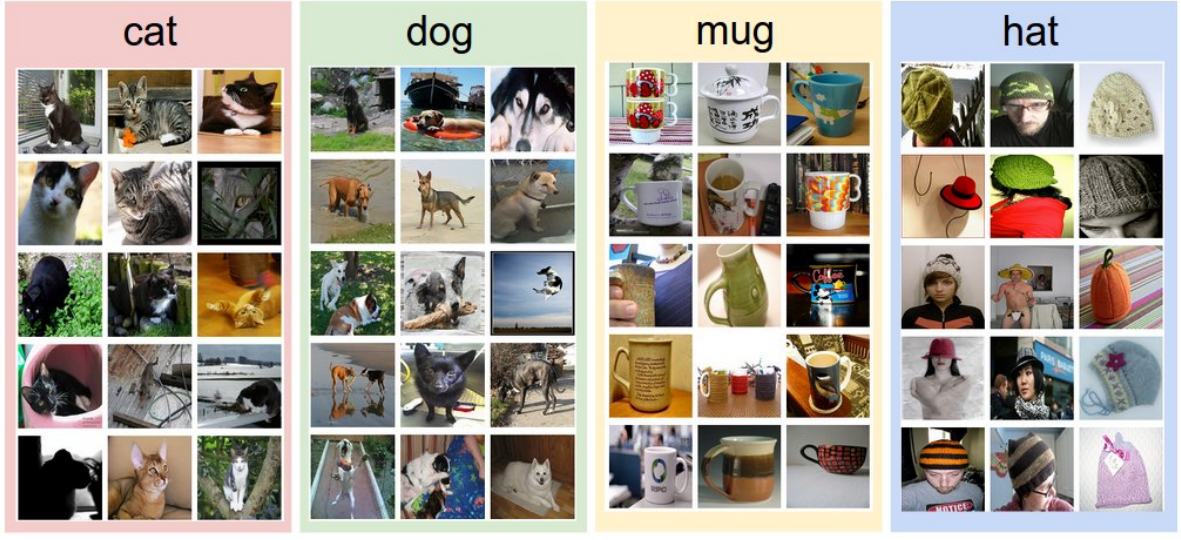
Challenges



- invariant to the cross product of all these variations
- retaining sensitivity to the inter-class variations

Imperial College London

Data-driven approach

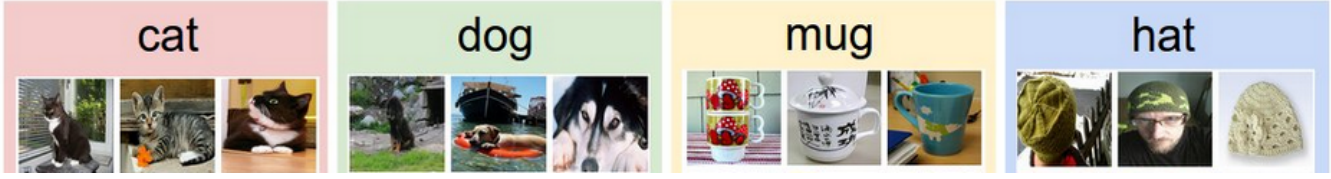


We need a (large) training dataset of labeled images.

Imperial College London

Pipeline for image classification

1. Training set: *N* images of *K* classes



2. Learning: training a classifier

3. Evaluation: against the *ground truth*

Imperial College London

Nearest Neighbour Classifier

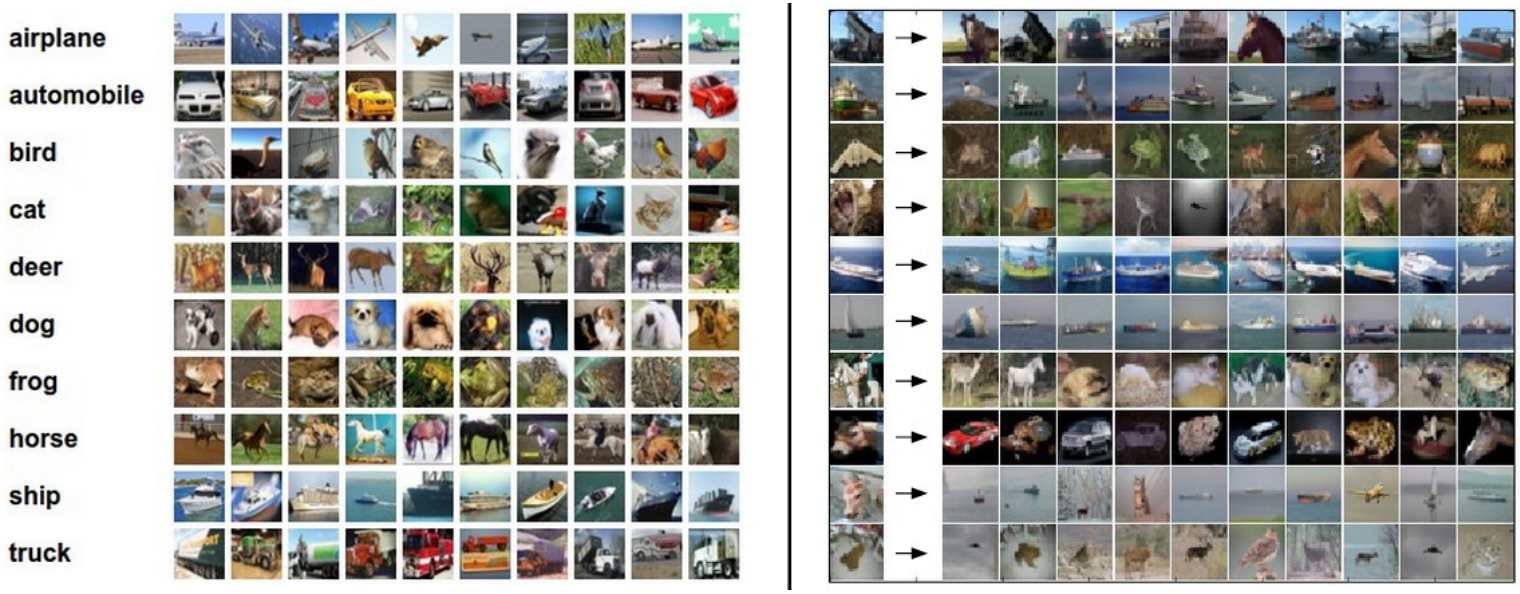


Figure 1: CIFAR-10 dataset: 60k tiny images of 10 classes.

The nearest neighbour classifier will take a test image, **compare** it to every single one of the training images, and predict the label of the closest training image.

Nearest Neighbour Classifier

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

Imperial College London

Nearest Neighbour Classifier

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

Imperial College London

Nearest Neighbour Classifier

test image			
56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

$-$

training image			
10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

$=$

pixel-wise absolute value differences			
46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

$\rightarrow 456$

Imperial College London

The choice of distance

L1 distance: $d_1(I_1, I_2) = \sum_{pixel} |I_1^p - I_2^p|$

L2 distance: $d_1(I_1, I_2) = \sqrt{\sum_{pixel} (I_1^p - I_2^p)^2}$

What's their accuracy?
What's human accuracy?
What's state-of-the-art neural networks' accuracy?

Imperial College London

k-Nearest Neighbour Classifier

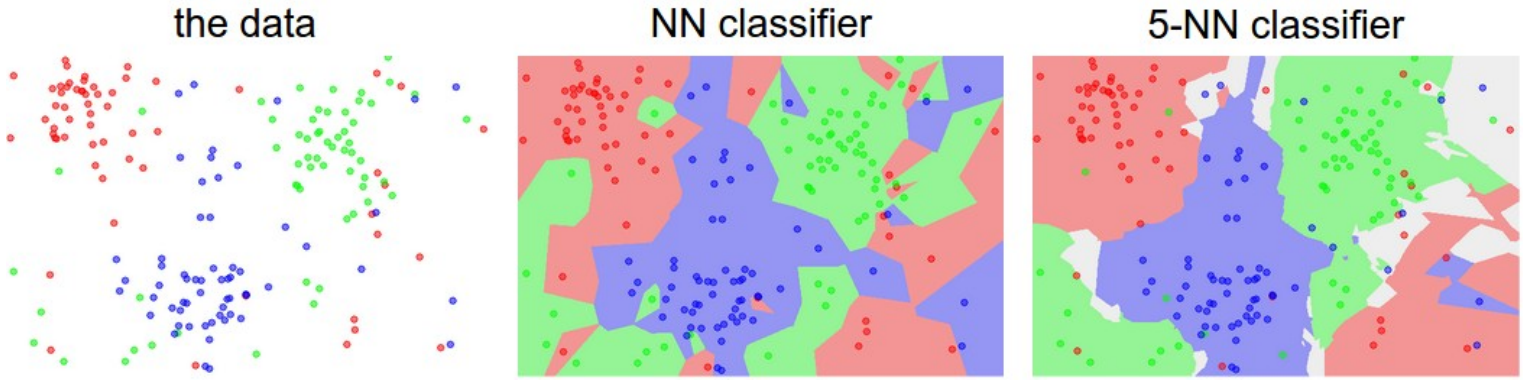


Figure 2: An example of the difference between Nearest Neighbor and a 5-Nearest Neighbor classifier, using 2-dimensional points and 3 classes (red, blue, green).

What value of k should we use? Which distance?

Imperial College London

Hyperparameter tuning

The engineer says: "We should try out many different values and see what works best."

Agree or disagree?

Imperial College London

Validation test

The good engineer says:
"Evaluate on the test set only a single time, at the very end."

- Split your training set into training set and a validation set.
- Use validation set to tune all hyperparameters.
- At the end run a single time on the test set and report performance.

Imperial College London

Data splits

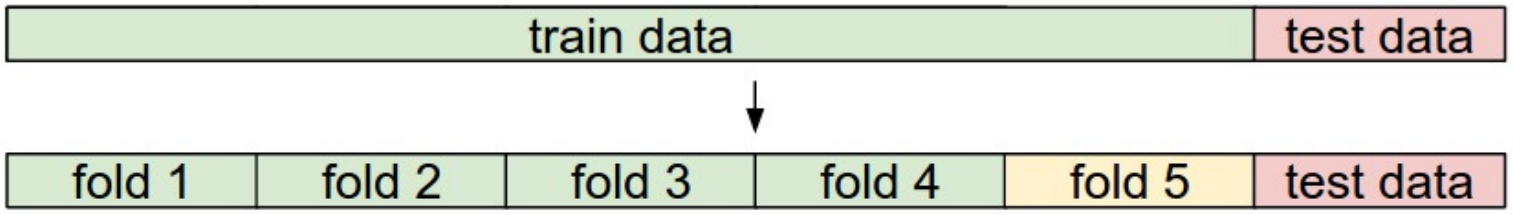


Figure 3: The training set is split into folds: 1-4 become the training set while 5 is the validation set used to tune the hyperparameters.

Where is the Nearest Neighbour classifier spending most of its (computational) time?

Imperial College London

Wrap up

- the problem of image classification: predicting labels for novel test entries
- training set vs testing set
- a simple Nearest Neighbor classifier requires hyperparameters
- validation set to tune hyperparameters
- Nearest Neighbor classifier has low accuracy (distances based on raw pixel values!) and is expensive at testing

Our aim: a solution which gives 90% accuracy, discards the training set once learning is complete, and evaluates a test image in less than a millisecond!

Linear classification

New approach based on:

- **score function** to map raw data to class scores
- **loss function** to quantify the agreement between predicted and true labels

Parameterised mapping from images to label scores

Our aim is to define the score function that maps the pixel values of an image to confidence scores for each class.

Assuming that:

N images, each with dimensionality D, and K distinct classes

$x_i \in R^D$ is image i-th with dimensions D and label y_i, with $i = 1...N$ and $y_i \in 1...K$

then we define a **score function**: $f : R^D \rightarrow R^K$

Linear classifier

Linear mapping: $f(x_i; W, b) = Wx_i + b$

W are called **weights** and b is the **bias** vector.

What are the dimensions of x_i, W and b?

Linear classifier

Linear mapping: $f(x_i; W, b) = Wx_i + b$

W are called **weights** and b is the **bias** vector.

What are the dimensions of x_i, W and b?

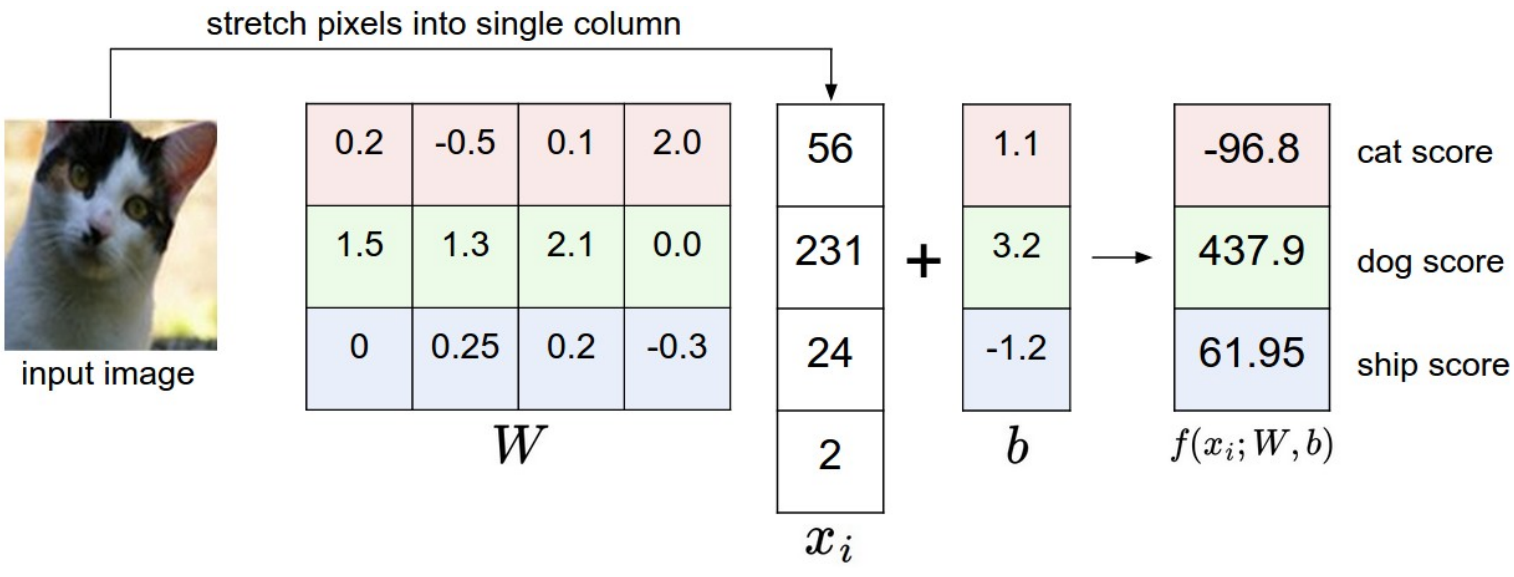
x_i has size [D x 1]

W has size [K x D]

b has size [K x 1]

Imperial College London

Linear classifier

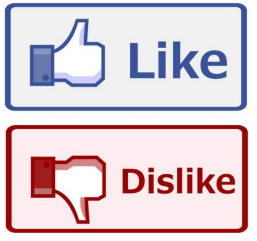


Imperial College London

Interpreting a linear classifier (i)

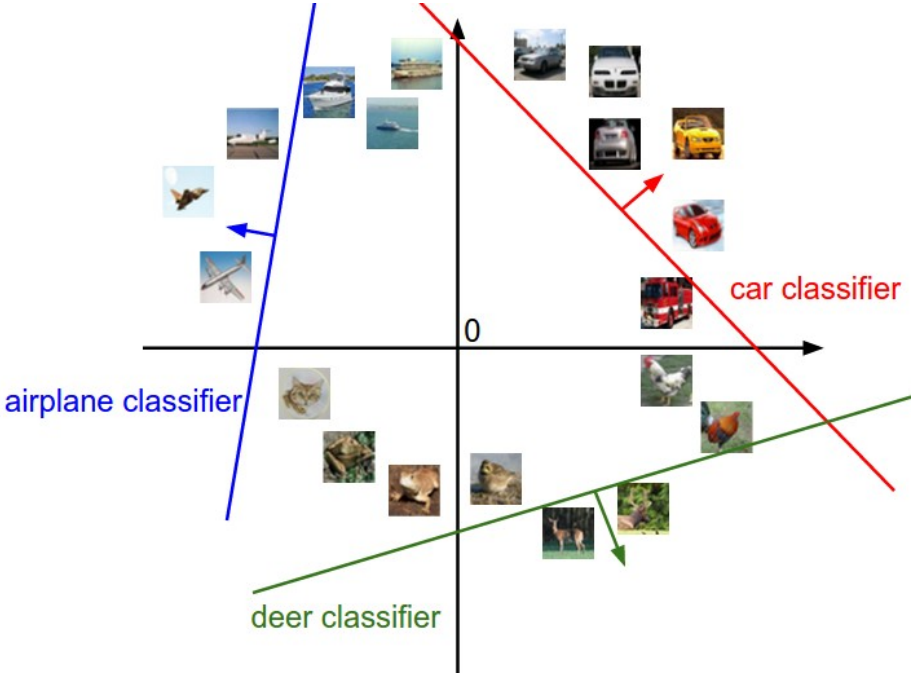
0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

W



Imperial College London

Interpreting a linear classifier (ii)



$$W = \begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix} \quad x_i = \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} \quad + \quad b = \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}$$

Imperial College London

Interpreting a linear classifier (iii)

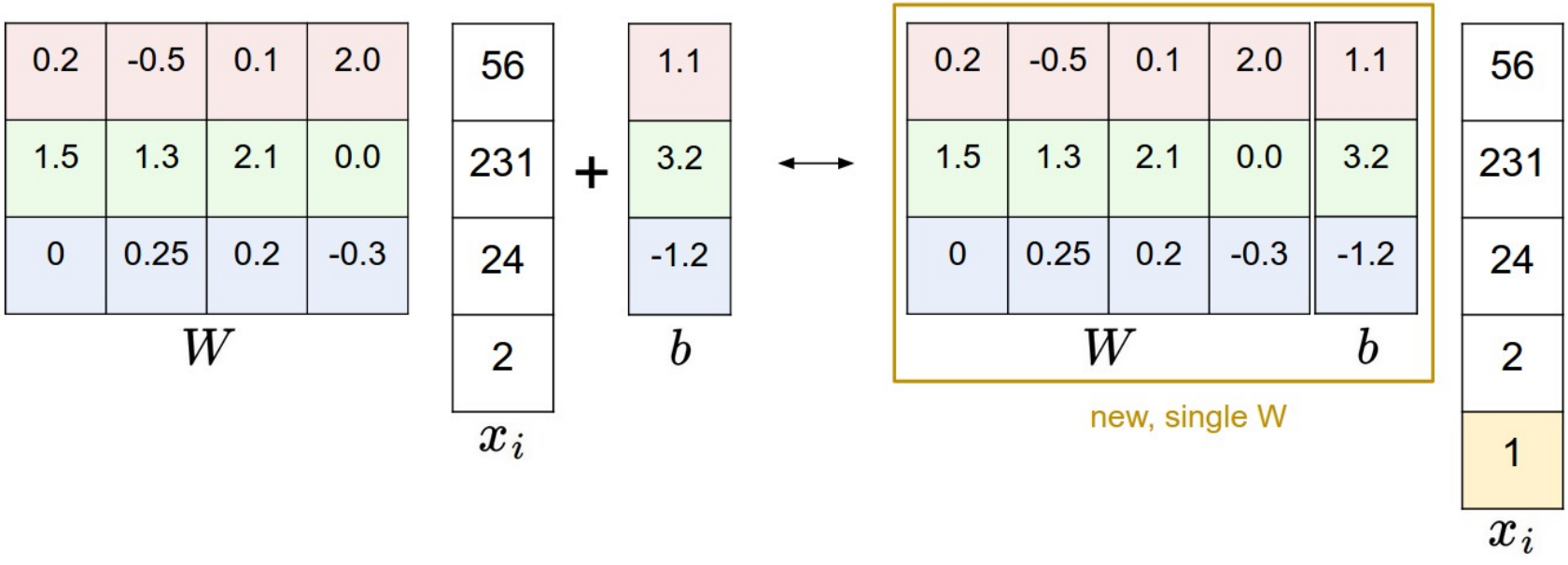
Template (or prototype) matching.

Imperial College London

Bias trick

Our new **score function**:

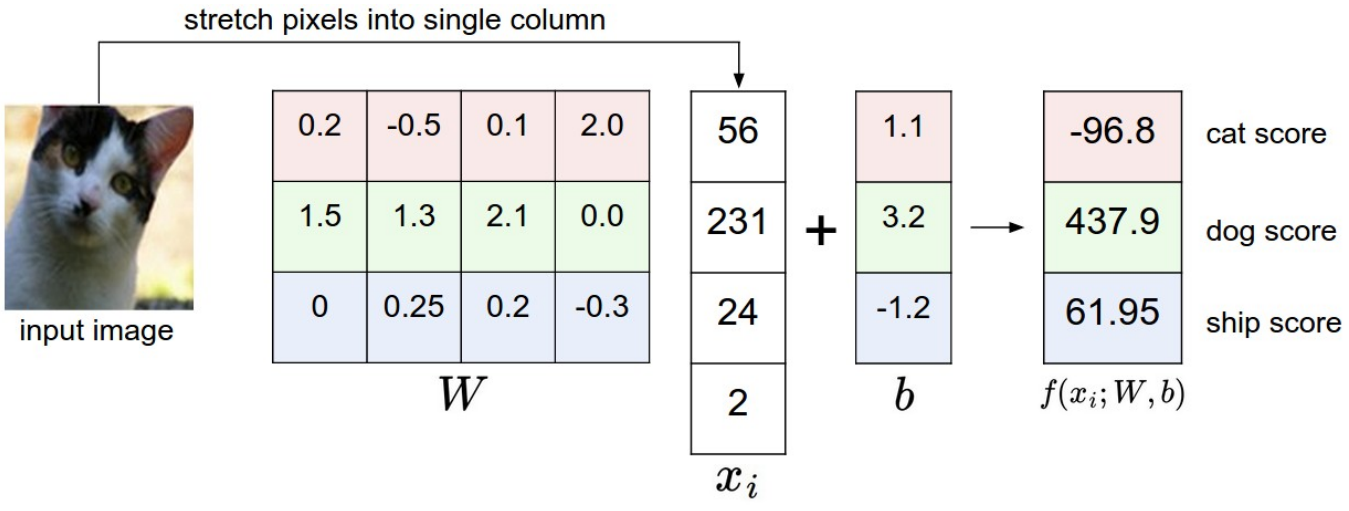
$f(x_i; W) = Wx_i$



Imperial College London

Loss function*

To measure our "unhappiness" with predicted outcomes.



* sometimes called cost function or objective

Imperial College London

Multiclass Support Vector Machine (SVM) loss

The SVM loss is set so that the SVM "wants" the correct class for each image to a have a higher score than the incorrect ones by some fixed margin.

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \delta)$$

Example:
$s = [13, -7, 11], y_i = 0, \delta = 10$
$L_i =$

Imperial College London

Multiclass Support Vector Machine (SVM) loss

The SVM loss is set so that the SVM "wants" the correct class for each image to a have a higher score than the incorrect ones by some fixed margin.

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \delta)$$

Example:
$s = [13, -7, 11], y_i = 0, \delta = 10$
$L_i = 8$

Imperial College London

Hinge loss

$max(0, -)$ or $max(0, -)^2$

Regularisation

If W correctly classifies each sample, then all λW with $\lambda > 1$ will have zero loss.

Which W should we choose?

Regularisation

If W correctly classifies each sample, then all λW with $\lambda > 1$ will have zero loss.

Which W should we choose?

Our new multiclass SVM loss function is:

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

including one data loss and one regularisation loss term $\lambda R(W)$, specifically L2 penalty.

Softmax classifier

Generalisation of the binary logistic regression classifier to multiple classes.

Cross-entropy loss function:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \tag{1}$$

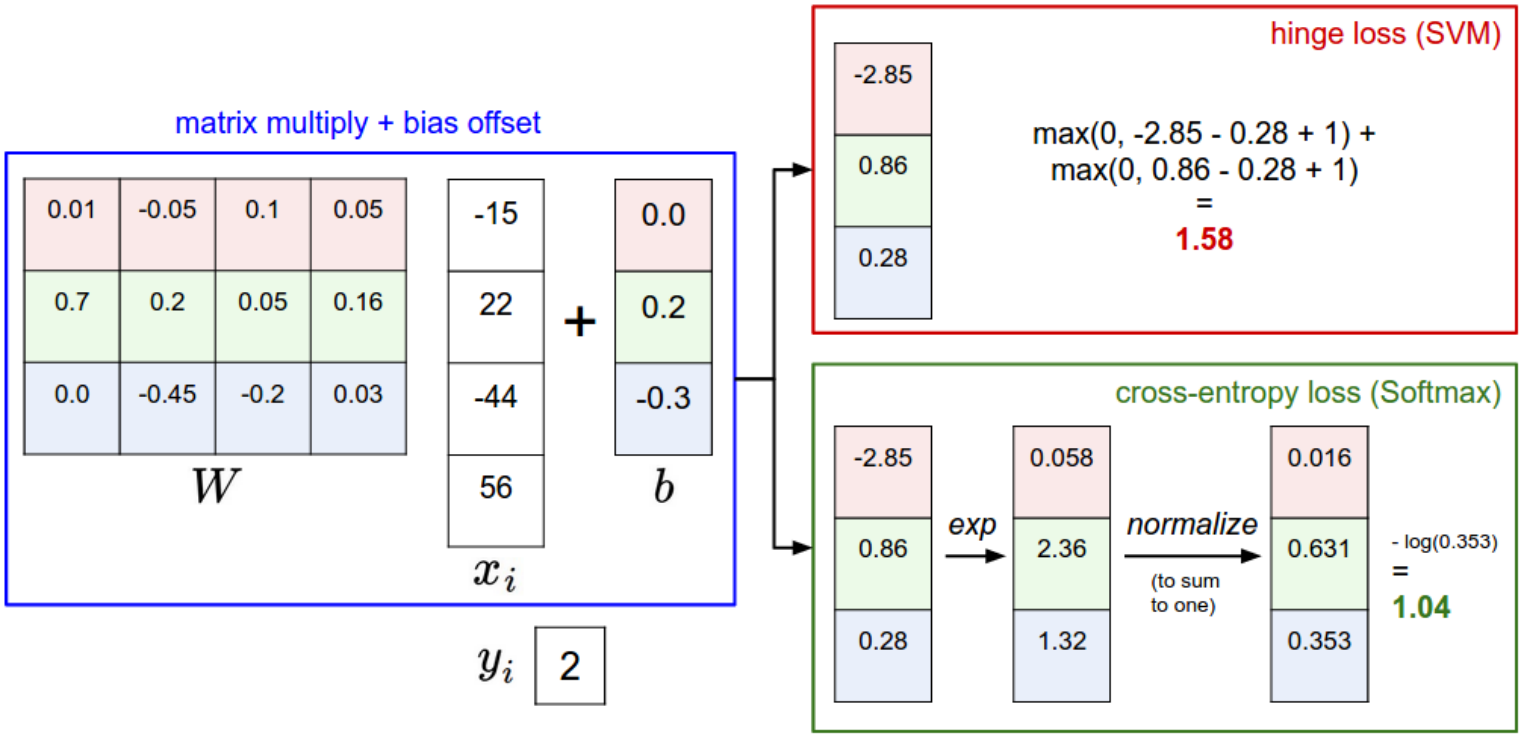
Probabilistic interpretation of Softmax scores

$$P(y_i|x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \tag{2}$$

Likelihood or Bayesian?

Imperial College London

SVM vs. Softmax classifier



Wrap up

- A score function maps image pixels to class scores (using a linear function that depends on W and b).
- Once we learning is done, we can discard the training data and prediction is fast.
- A loss function (e.g. SVM and Softmax) measures how compatible a given set of parameters is with respect to the ground truth labels in the training dataset.

How do we determine (optimise) the parameters that give the lowest loss?

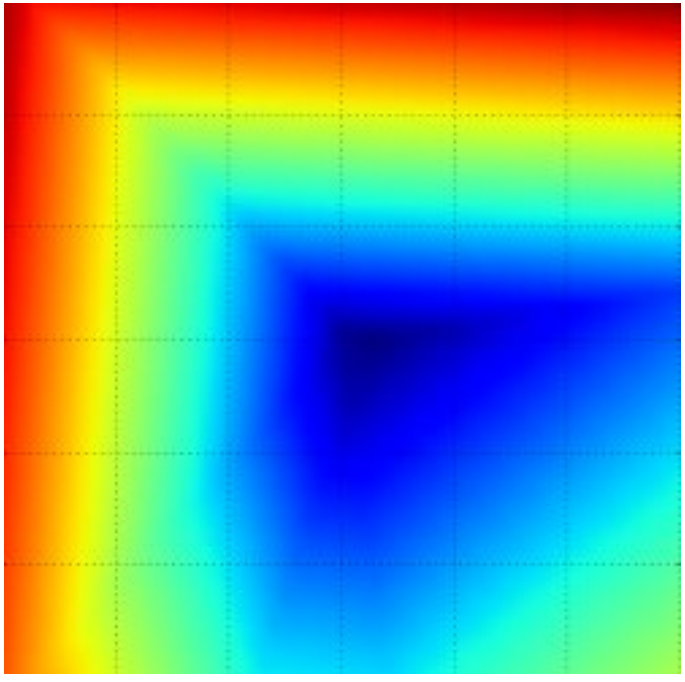
Key components for image classification

1. score function
2. loss function
3. optimisation

Optimisation is the process of finding the set of parameters W that minimise the loss function L.

Visualising the loss function

If W_0 random starting point, W_1 random direction, then compute $L(W_0 + aW_1)$ for different values of a.



(averaged across all images, x_i)

Imperial College London

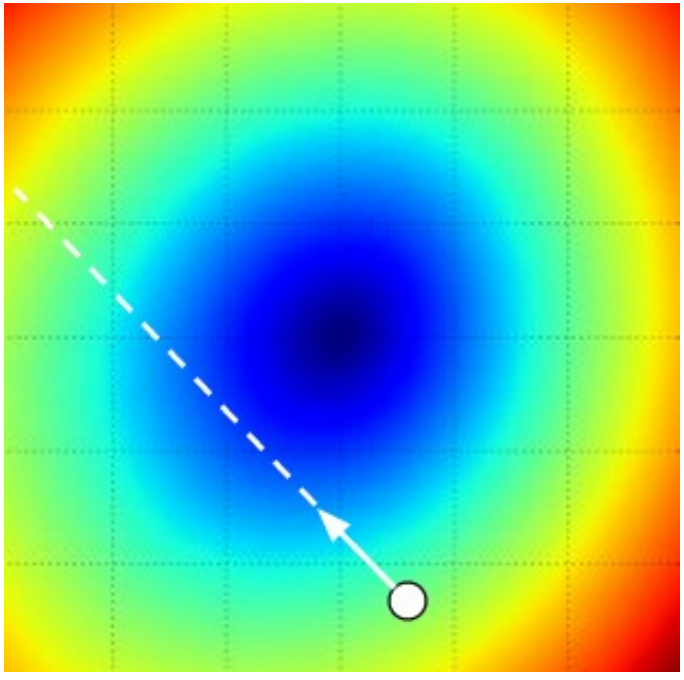
Optimisation

- Random search
- Random local search
- Gradient descent (numerical or analytical)

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Hyperparameters

Step size or learning rate



Batch size:
Compute the gradient over batches (e.g. 32, 64, 128...) of the training data.

Imperial College London

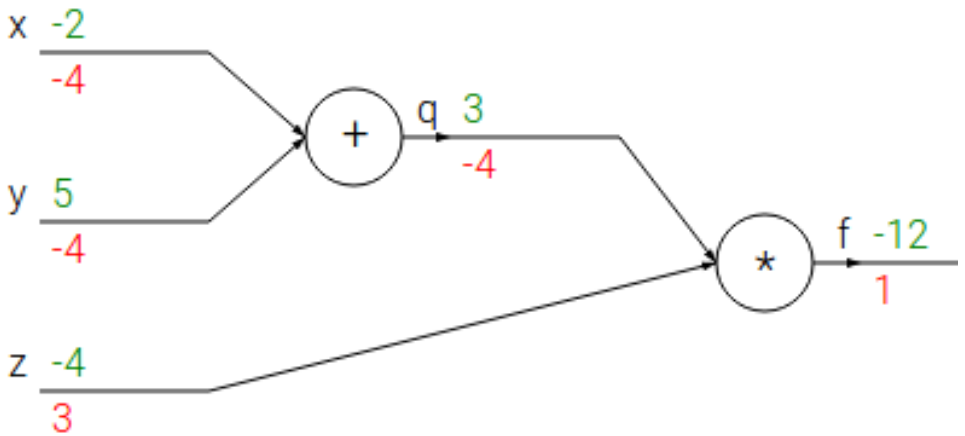
Backpropagation

We can compute the gradient analytically using the chain rule.

$$f(x, y, z) = (x + y)z$$

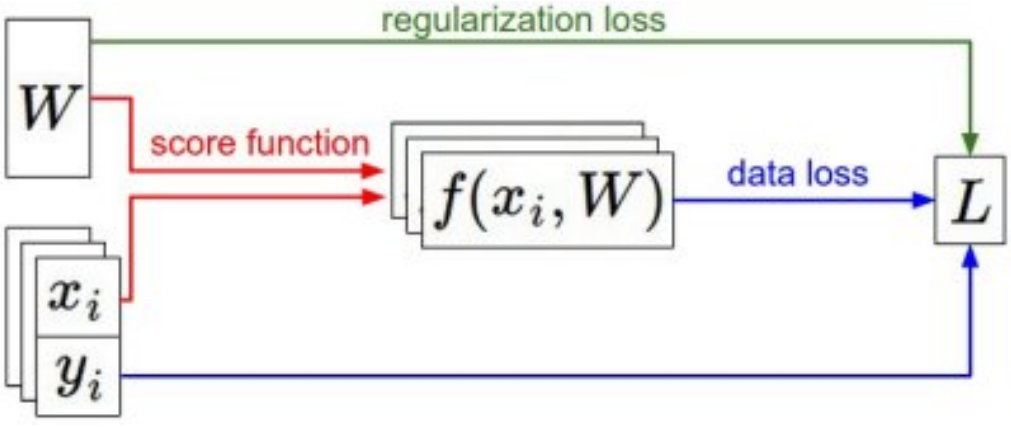
$$q = x + y \text{ and } f = qz$$

$$\frac{df}{dx} = \frac{df}{dq}\frac{dq}{dx}$$



Imperial College London

Wrap up

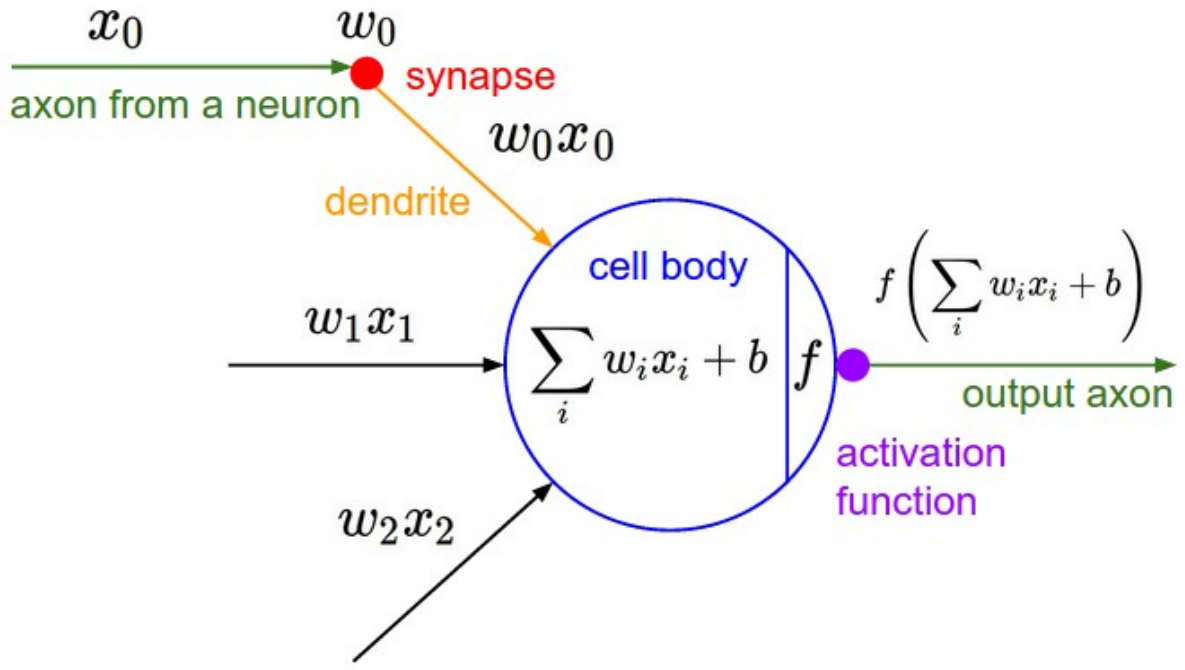


The 3 elements: score function, loss function, optimisation.

Next: let's put them all together in a neural network.

Imperial College London

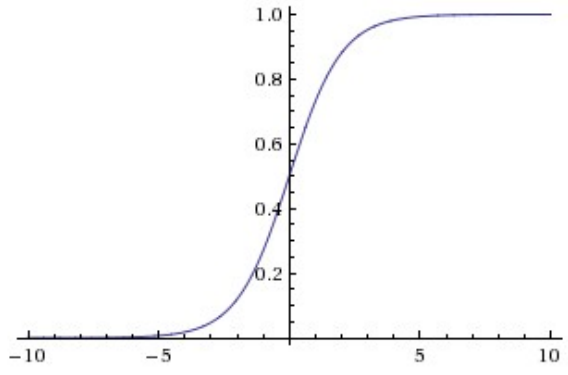
Neurons



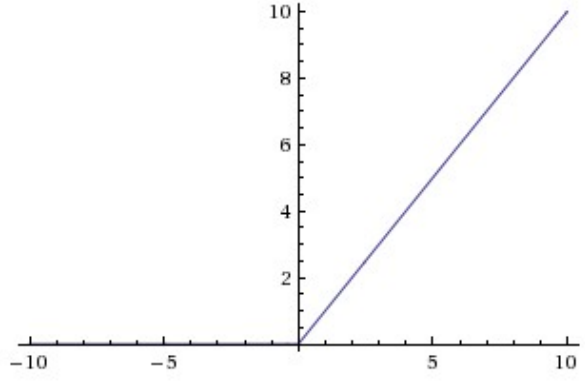
Imperial College London

Activation functions

It defines the *firing rate*



Sigmoid non-linearity squashes real numbers to range between $[0, 1]$



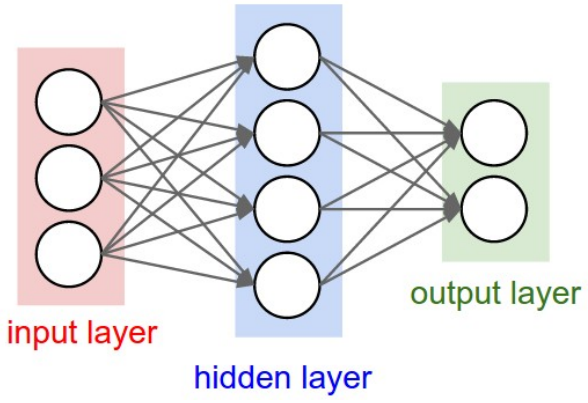
Rectified Linear Unit (ReLU): $f(x) = max(0, x)$

Imperial College London

Neural network architecture

Collection of neurons connected in an acyclic graph.

Last output layer represents class scores.



A 2-layer Neural Network

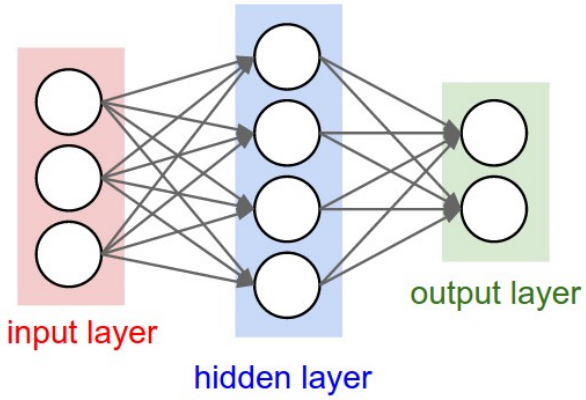
Size:

Imperial College London

Neural network architecture

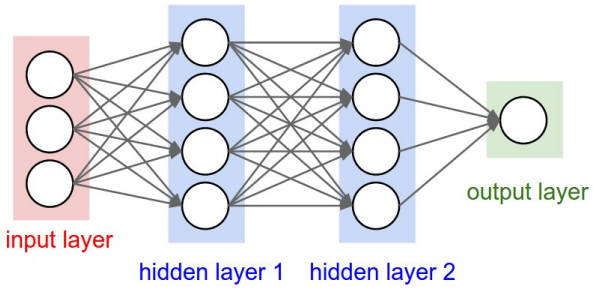
Collection of neurons connected in an acyclic graph.

Last output layer represents class scores.



A 2-layer Neural Network

Size: $4 + 2 = 6$ neurons, $[3 \times 4] + [4 \times 2] = 20$ weights and $4 + 2 = 6$ biases, for a total of 26 learnable parameters.



A 3-layer Neural Network

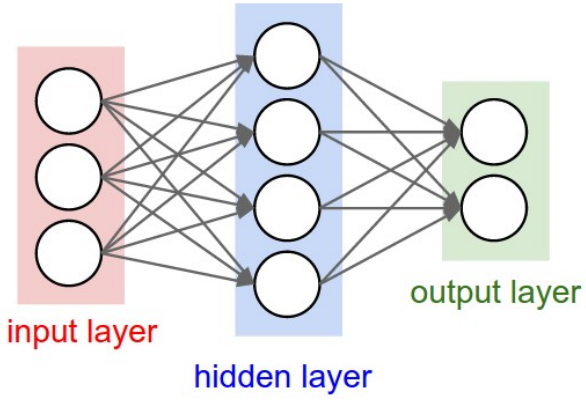
Size:

Imperial College London

Neural network architecture

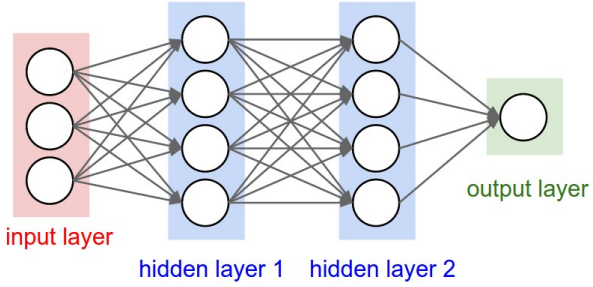
Collection of neurons connected in an acyclic graph.

Last output layer represents class scores.



A 2-layer Neural Network

Size: $4 + 2 = 6$ neurons, $[3 \times 4] + [4 \times 2] = 20$ weights and $4 + 2 = 6$ biases, for a total of 26 learnable parameters.



A 3-layer Neural Network

Size: $4 + 4 + 1 = 9$ neurons, $[3 \times 4] + [4 \times 4] + [4 \times 1] = 12 + 16 + 4 = 32$ weights and $4 + 4 + 1 = 9$ biases, for a total of 41 learnable parameters.

Representational power

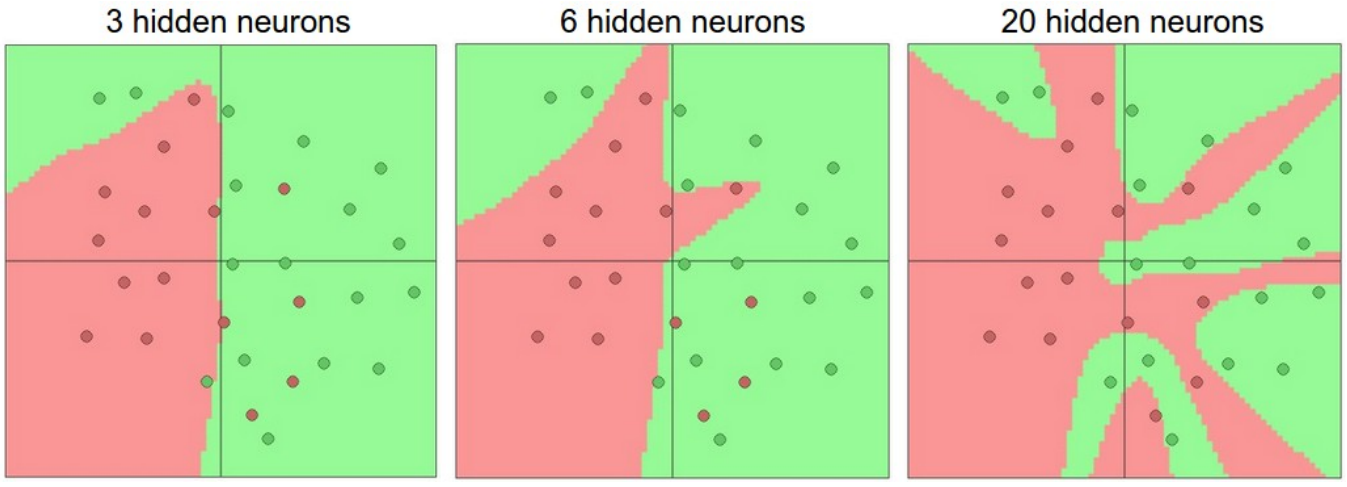
Given any continuous function $f(x)$ and some $\epsilon > 0$, there exists a Neural Network $g(x; W)$ with one hidden layer (with a reasonable choice of non-linearity, e.g. sigmoid) such that for all x, $| f(x) - g(x) | < \epsilon$.

In other words, the neural network can approximate any continuous function.

In practice, more layers work better...

Imperial College London

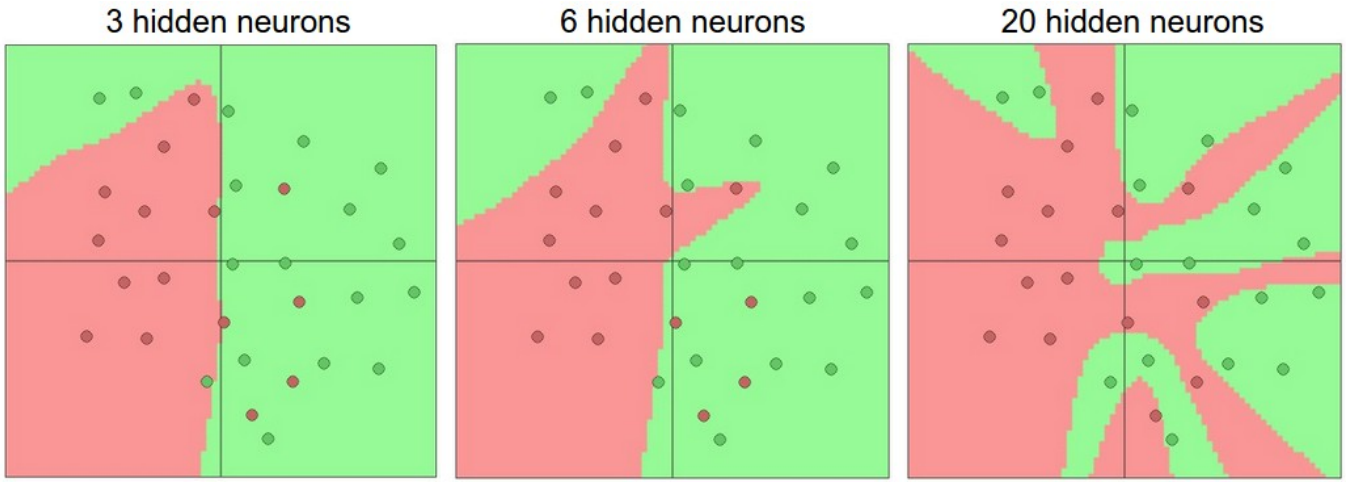
Setting up the architecture



Capacity vs. ?

Imperial College London

Setting up the architecture



Capacity vs. ? Overfitting

We aim at a better **generalisation**.

Setting up the data

Data preprocessing:

- mean subtraction
- normalisation
- PCA and Whitening

Setting up the model

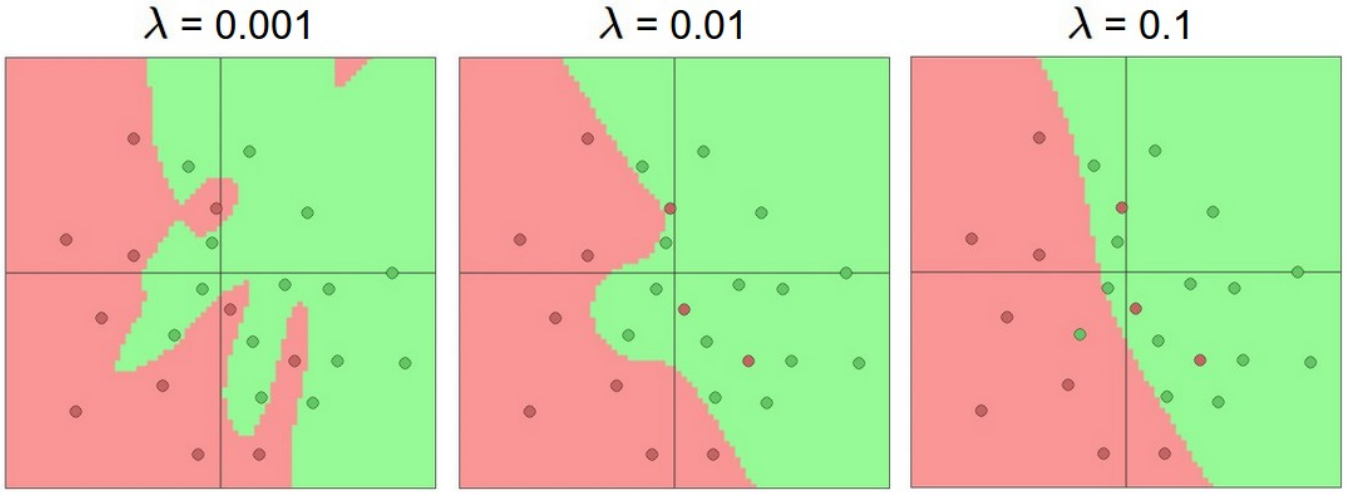
Weight initialisation:

- all zero
- small random numbers
- calibrate the variances
- sparse

Imperial College London

Setting up the model

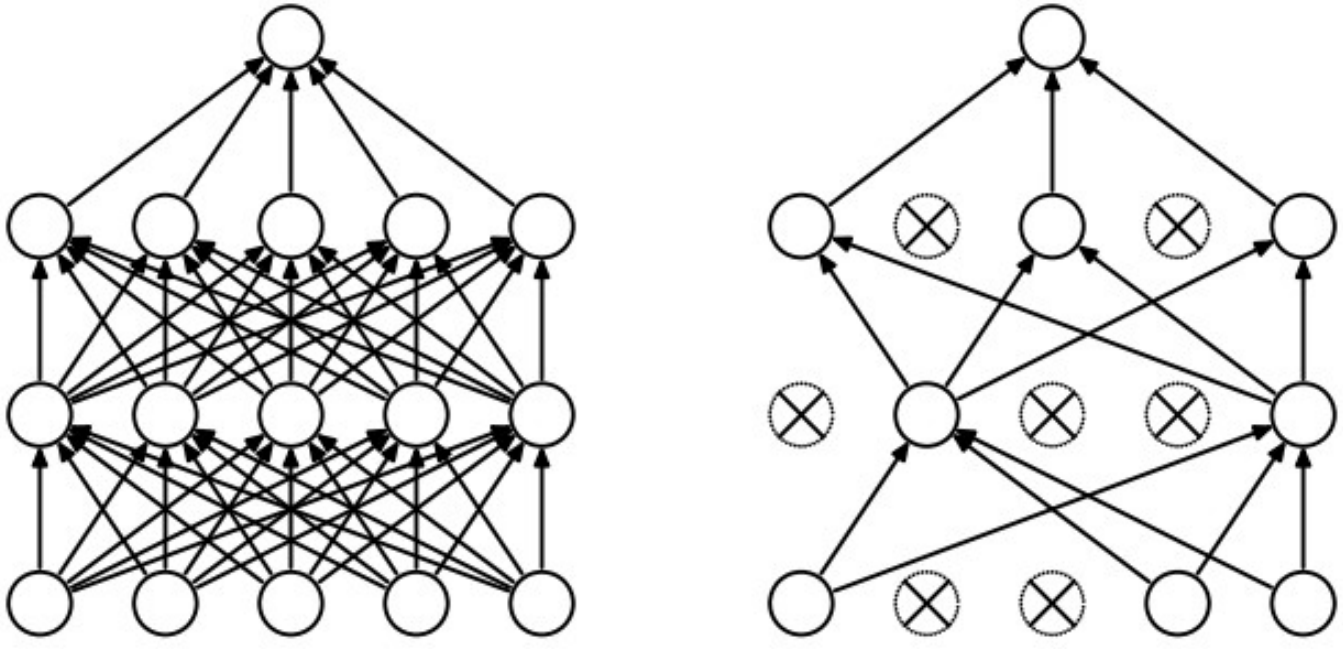
Regularization



Options: L2, L1, maxnorm and dropout.

Imperial College London

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Dropout can be interpreted as sampling a Neural Network within the full Neural Network, and only updating the parameters of the sampled network based on the input data.

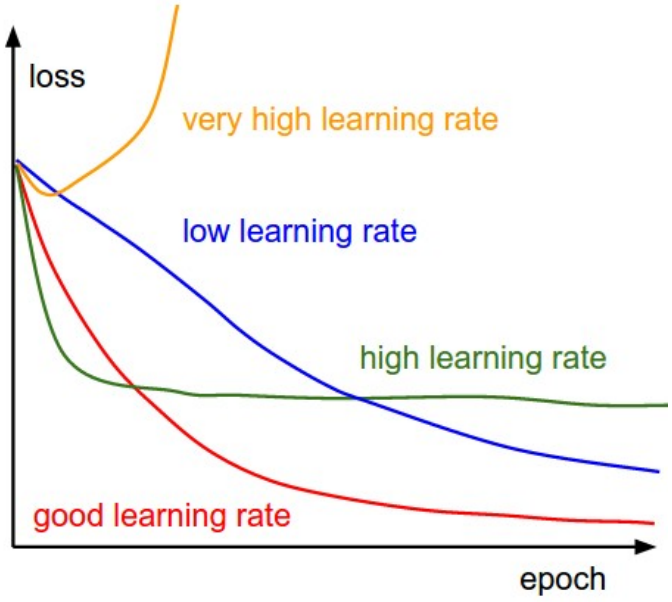
Setting up the model

Loss functions:

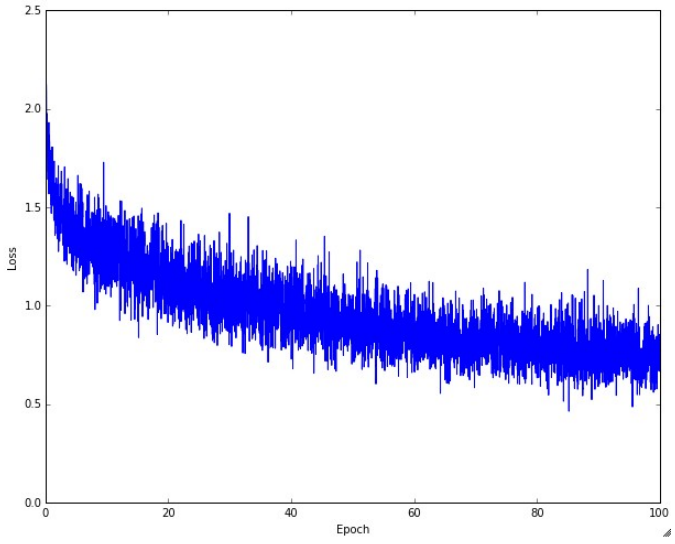
- SVM (hinge loss)
- cross-entropy
- hierarchical softmax
- attribute classification
- regression (?)

Imperial College London

Setting up the learning



effects of different learning rates

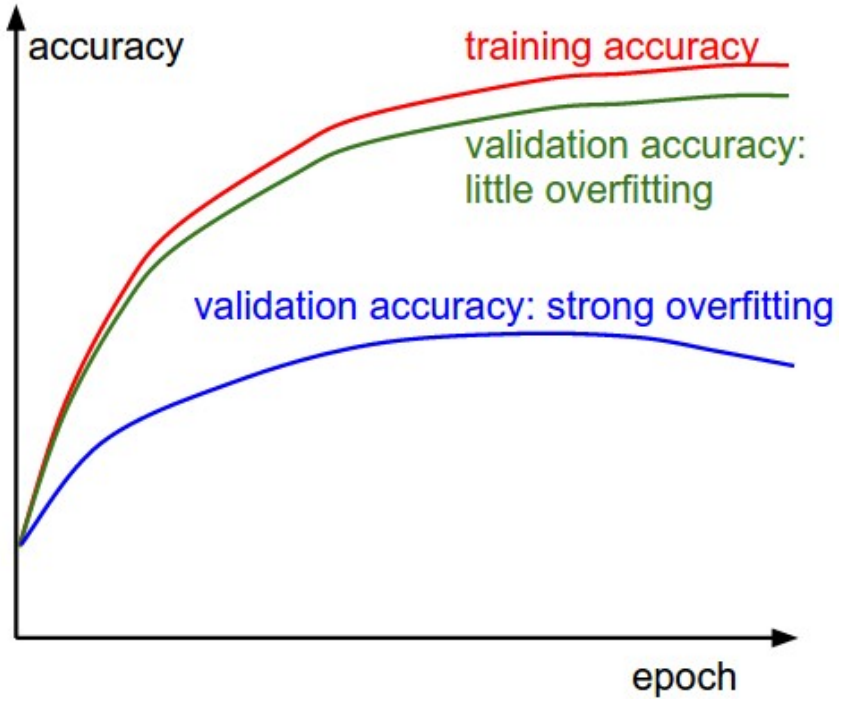


loss decay

Imperial College London

Setting up the learning

Training vs. validation accuracy



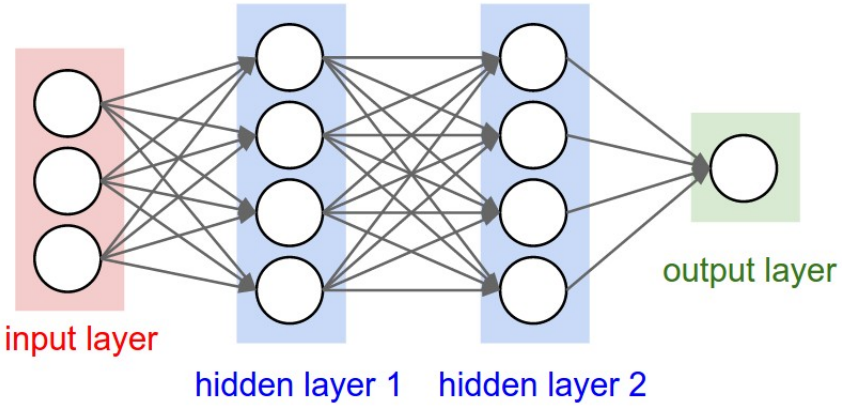
Wrap up

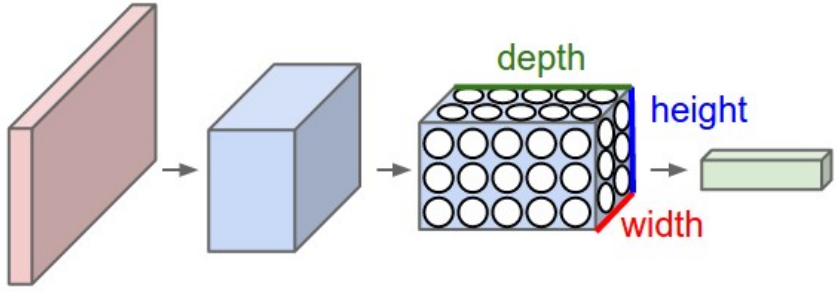
- Neural Networs are made of layers of neurons/units with activation functions
- Choice of the architecture: capacity vs overfitting
- Preprocessing of the data and choice of hyperparameters for the model and learning

What about images? Can we use neural networks straight from images? What's the issue?

Imperial College London

Convolutional Neural Networks (CNN)





A CNN arranges its neurons in three dimensions (width, height, depth). Every layer of a CNN transforms the 3D input volume to a 3D output volume of neuron activations. Neurons in a layer are connected only to a small region of the layer before it.

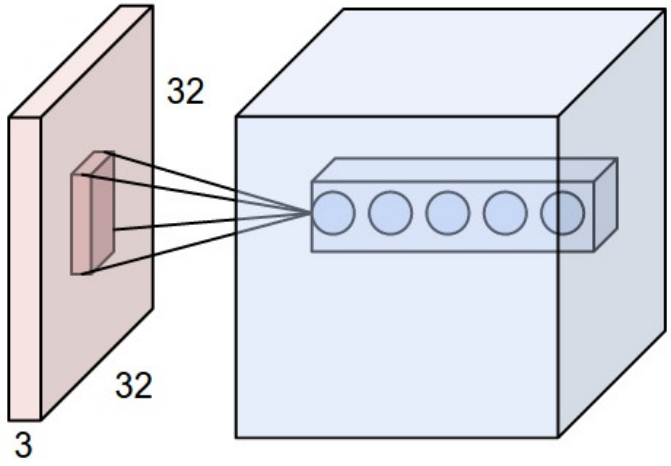
CNN architecture

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

We will stack these layers to form a full CNN architecture.

Imperial College London

Convolutional layer



Set of learnable filters which *slides* across the width and height of the input volume.

3 hyper-parameters: depth (nr of filters), stride, zero-padding.

Convolutional layer

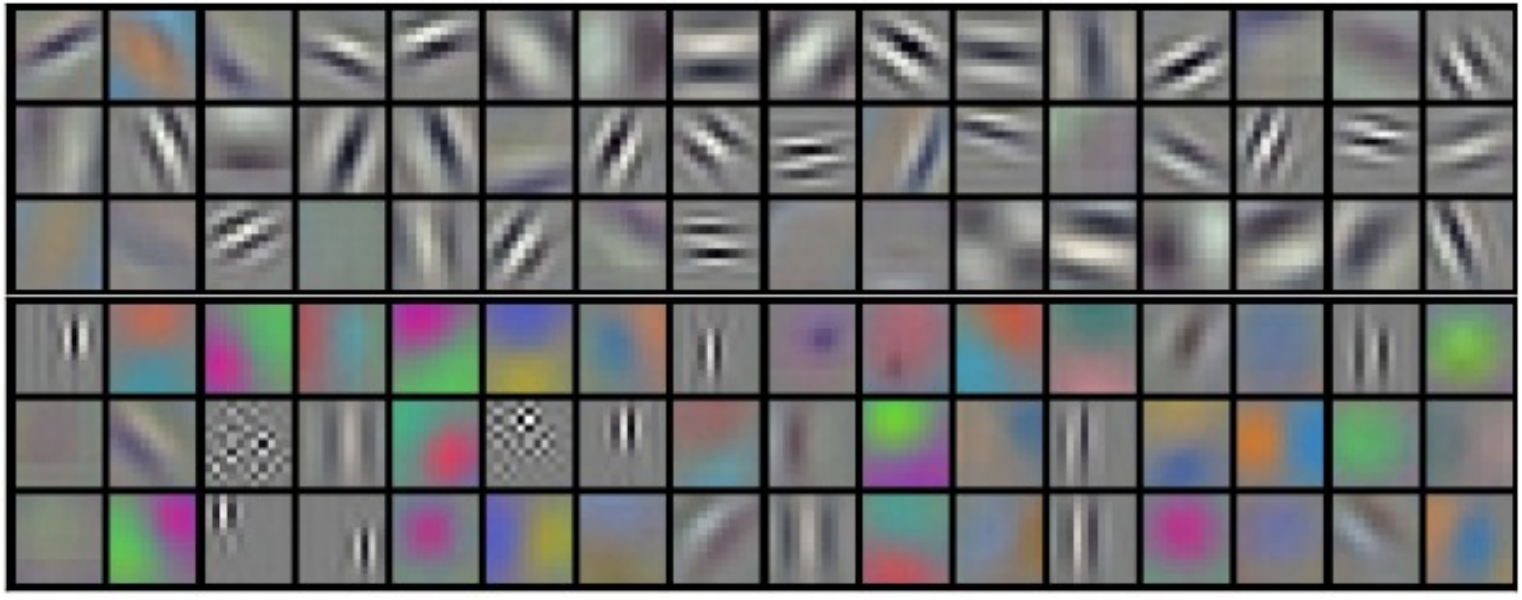
- Accepts a volume of size $W_1 x H_1 x D_1$
- Requires 4 parameters: number of filters K, their size F, the stride S, the amount of zero padding P
- Produces a volume of size $W_2 x H_2 x D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$

Usually: $F = 3, S = 1, P = 1$.

Demo: `http://cs231n.github.io/convolutional-networks/`

Imperial College London

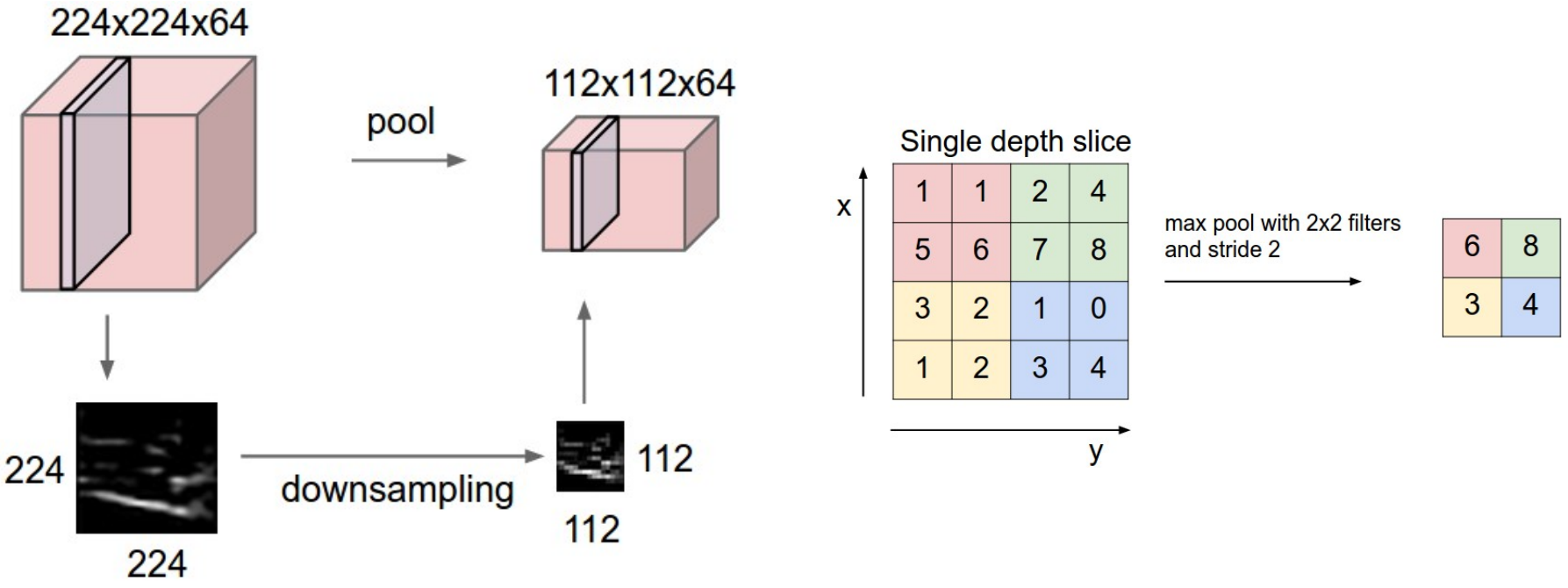
Weights in filters



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice

Imperial College London

Pooling layer



Size will influence the proportion of weights retained.

Imperial College London

Layer patterns

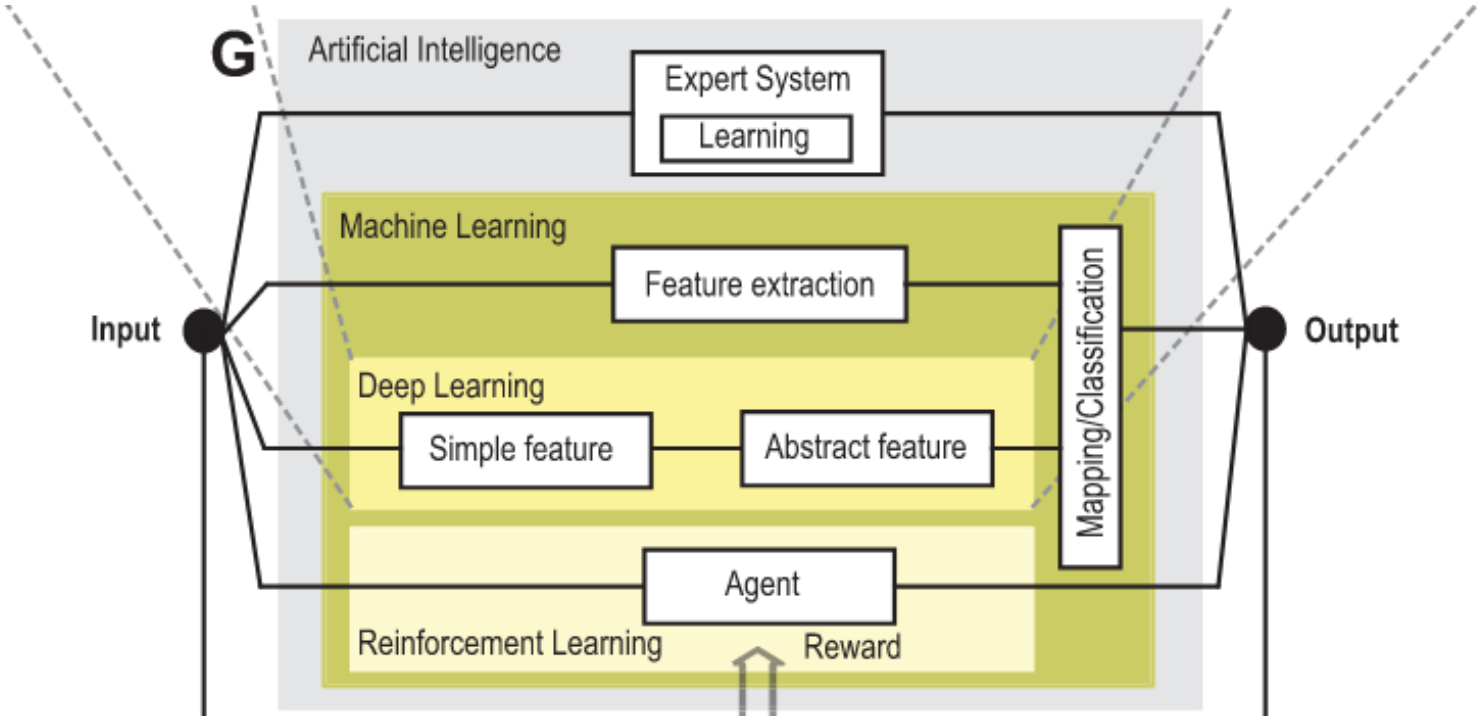
INPUT → [[CONV → RELU]*N → POOL?]*M → [FC → RELU]*K → FC

More examples:

`http://cs231n.github.io/convolutional-networks/`

Imperial College London

Evolution of AI



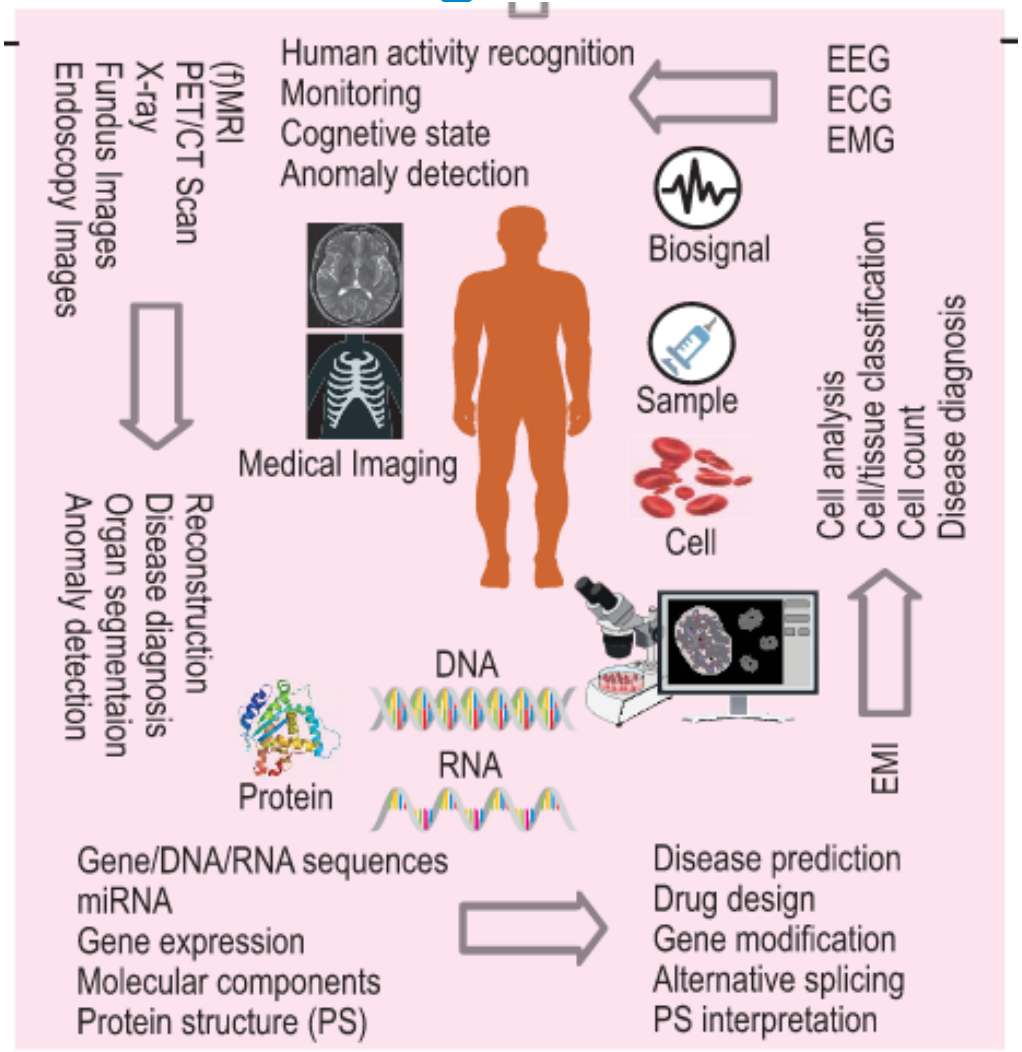
Applications to biological data

From various sources:

- -omics (gen-, transcript-, epigen-, prote-, metabol-)
- bioimaging (cellular images, ...)
- medical images (clinical imaging)
- brain/body machine interfaces (ECG, EEG, ...)

Imperial College London

Applications to biological data



Omics

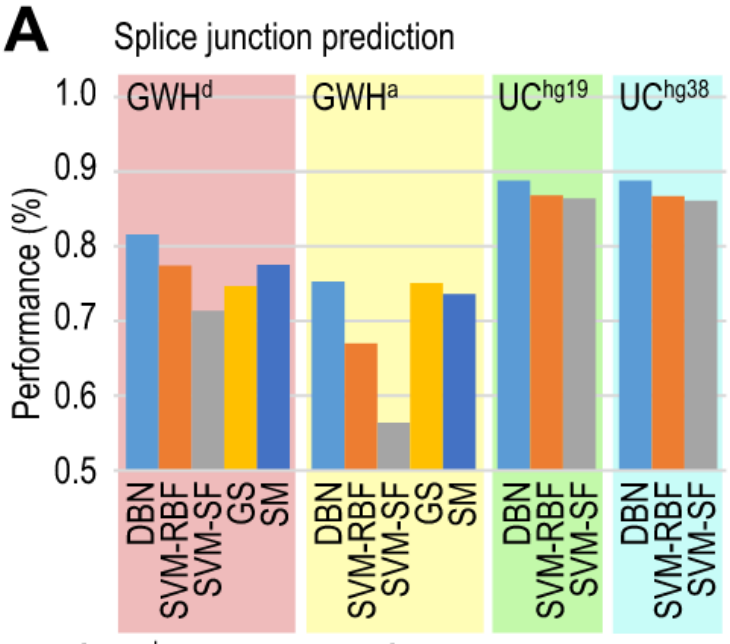
Mining DNA/RNA sequence data to:

- identify splicing junction
- classify somatic point mutation-based cancer
- predict DNA- and RNA-binding motifs
- relate disease-associated variants to gene expression
- estimate DNA methylation patterns
- ...

Imperial College London

Splice junctions

Deep neural networs outperform other methods.



GWH^{d}: Whole Human Genome-Donor dataset; GWH^{a}: GWH-Acceptor dataset; UC^{hg19}: UCSC-hg19 dataset; UC^{hg38}: UCSC-hg38 dataset; SVM-RBF: SVM with Radial Basis Function; SVM-SF: SVM with Sigmoid Function; GS: Gene Splicer; SM: Splice Machine.

Open issues

- theory of deep learning is not completely understood making outcomes difficult to interpret
- susceptible to misclassification and overclassification
- uncertainty in building architectures
- bootstrapping not possible

Imperial College London

Future perspectives

- improving theoretical foundations on the basis of experimental data
- assessment of model's computational complexity and learning efficiency
- novel data visualization tools
- in biology: reduce data redundancy and extract novel information
- *ad hoc* computational infrastructures

Wrap up

- CNN arranges its neurons in three dimensions
- Different type of layers (convolution, pooling, ...)
- Weights in filters are learned
- Promising applications to biological data.

Imperial College London

TensorFlow

easy and intuitive way to do ML

Imperial College London

TensorFlow

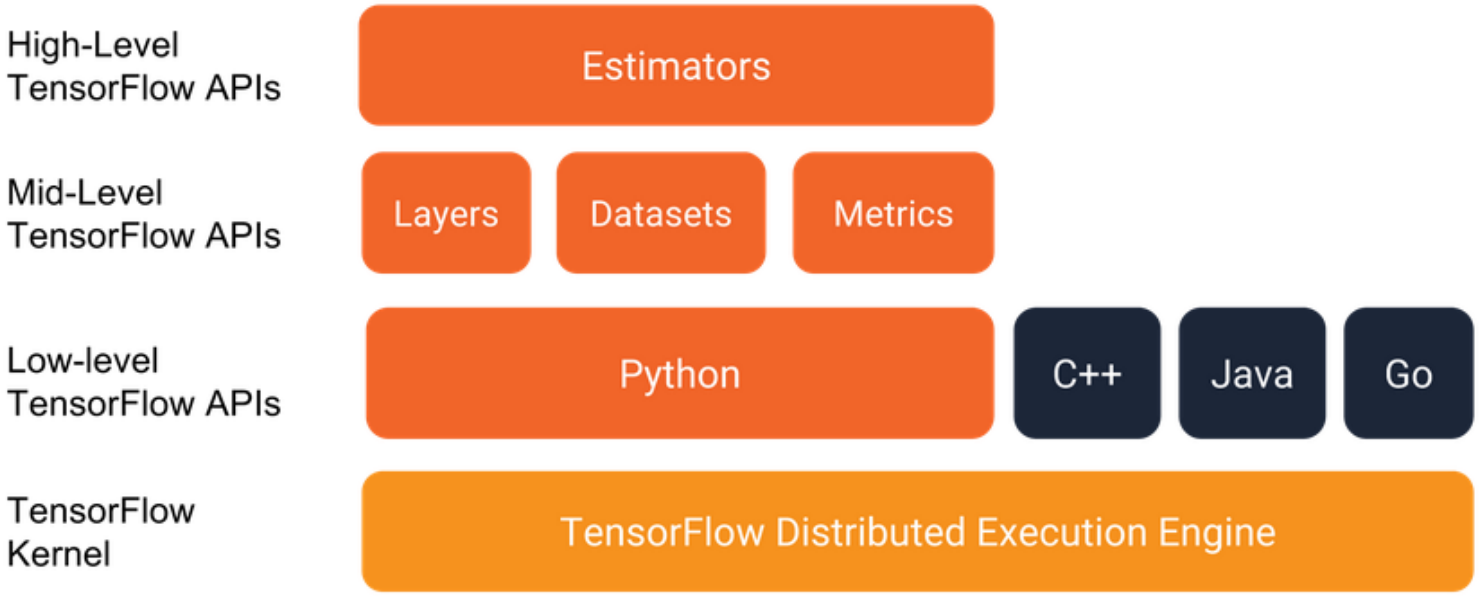
```python
import tensorflow as tf
mnist = tf.keras.datasets.mnist

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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

concept-heavy but code-light

many parameters, but only few are important to adjust

Imperial College London

TensorFlow



Imperial College London

Low-level APIs

What is a tensor?

```
3. # a rank 0 tensor; a scalar with shape [],
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

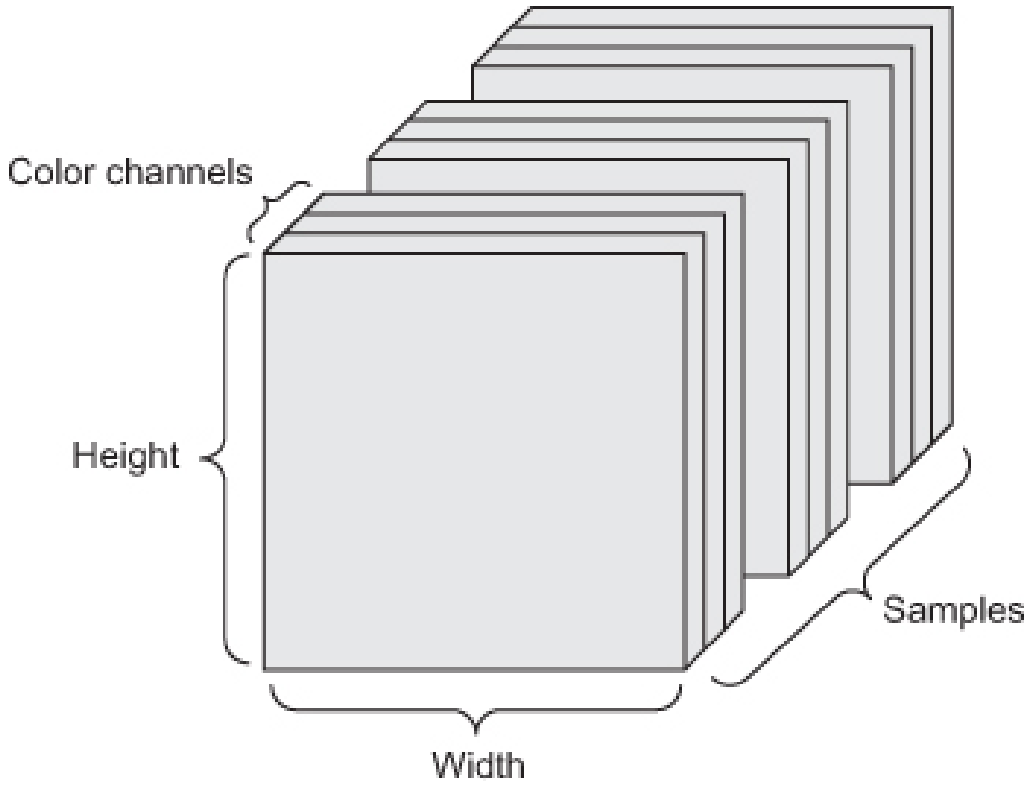
Examples of data tensors

- vector data: 2D tensors of shape `(samples, features)`
- timeseries or sequence data: 3D tensors of shape `(samples, timesteps, features)`
- images: 4D tensors of shape `(samples, height, width, channels)`
- video: 5D tensors of shape `(samples, frames, height, width, channels)`

The first axis can be the sample or batch dimension.

Imperial College London

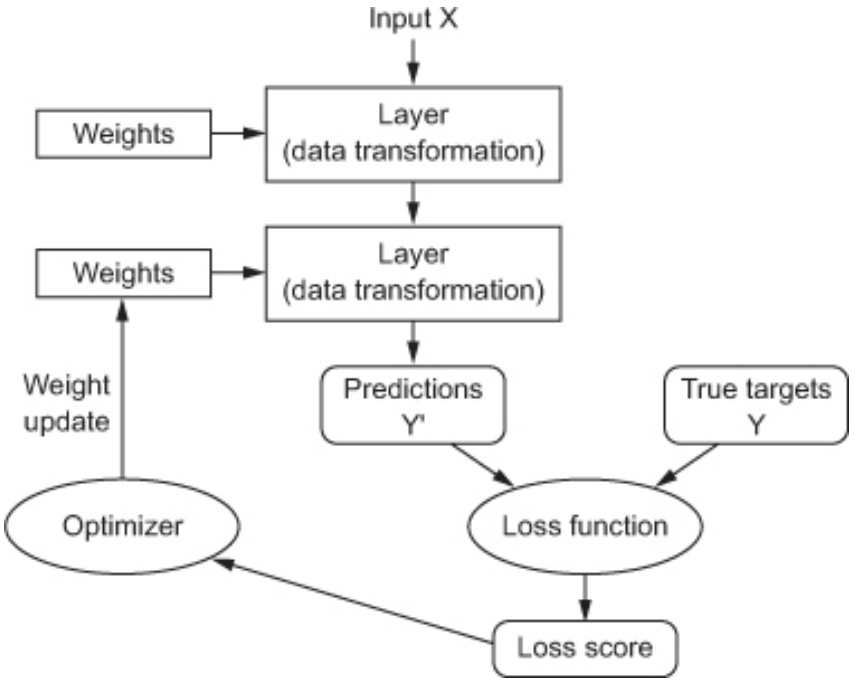
Image data



A batch of 128 colour images can be stored in a 4D tensor with shape `(128,256,256,3)`

Imperial College London

Anatomy of a neural network



Imperial College London

Build your first neural network

1. collect and preprocessing a dataset: most of the actual work

Imperial College London

Build your first neural network

1. collect and preprocessing a dataset: most of the actual work
2. build your model: few lines of code

Imperial College London

Build your first neural network

1. collect and preprocessing a dataset: most of the actual work
2. build your model: few lines of code
3. train: one line of code

Imperial College London

Build your first neural network

1. collect and preprocessing a dataset: most of the actual work
2. build your model: few lines of code
3. train: one line of code
4. evaluate: one line of code

Imperial College London

Build your first neural network

1. collect and preprocessing a dataset: most of the actual work
2. build your model: few lines of code
3. train: one line of code
4. evaluate: one line of code
5. predict: one line of code

source: Get started with TensorFlow's High-Level APIs (Google I/O '18)

Imperial College London

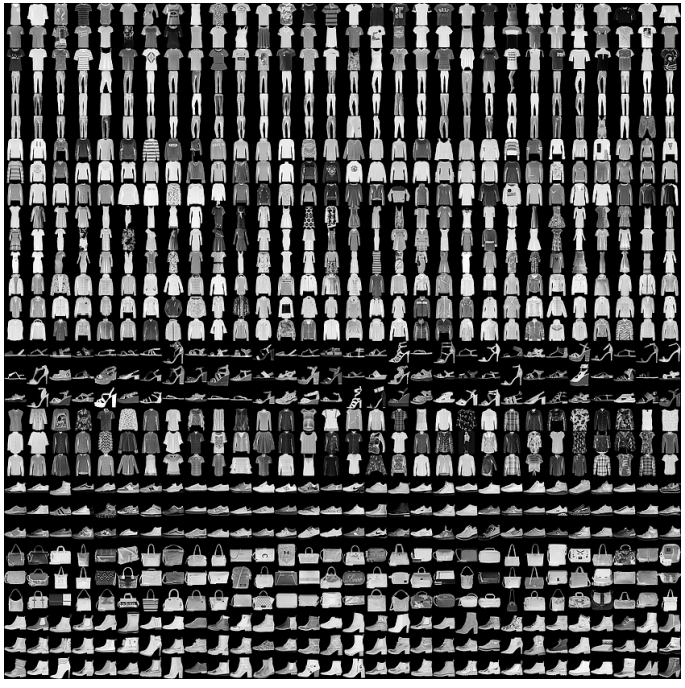
Step 1: collect a dataset

Import the data and spend a lot of time asking questions on: rank, shape, number of objects, printing, format, data type, ...: very basic questions!

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
```

Imperial College London

Step 1: collect a dataset



70,000 28x28 grayscale images in 10 categories of clothing articles

Imperial College London

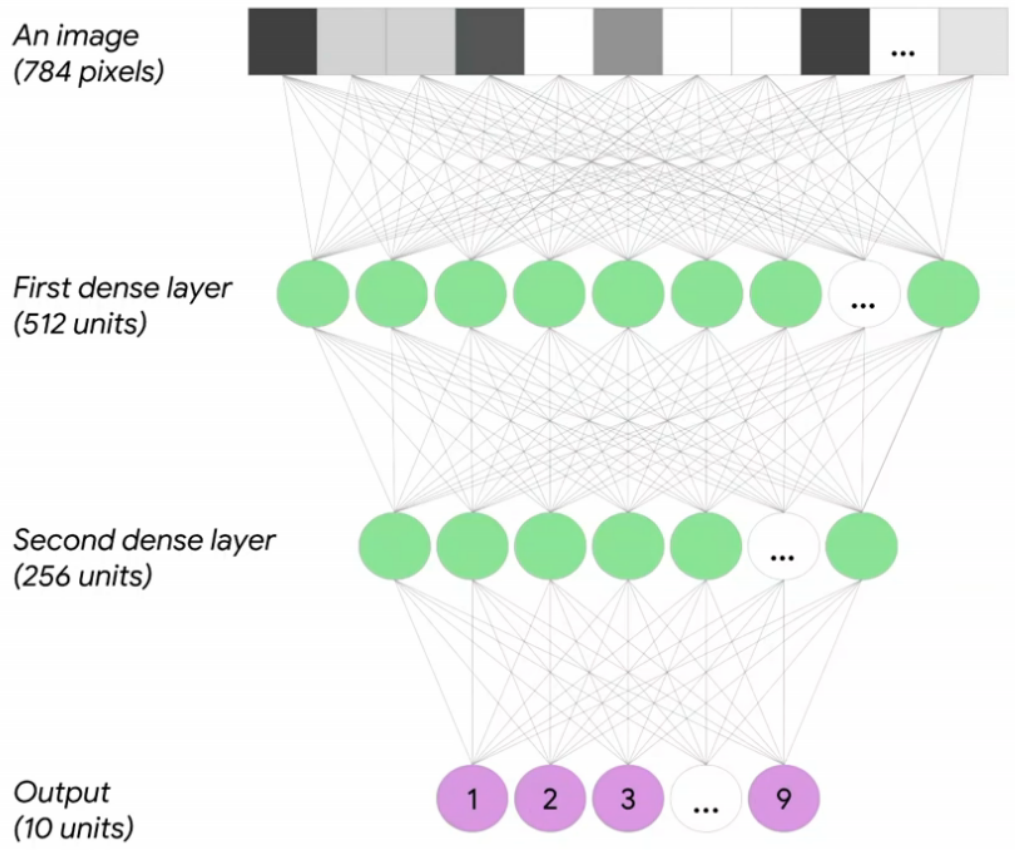
Step 1: collect a dataset

```
fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Imperial College London

Step 2: build a model




```
model.add(Dense(512,
 activation=tf.nn.relu,
 input_shape=(784,)))
```

```
model.add(Dense(256,
 activation=tf.nn.relu))
```

```
model.add(Dense(10,
 activation=tf.nn.softmax))
```

Step 2: build a model

- start simple! do not overlearn the training set.
- define loss function
- define optimization (important but defaults are good)

```
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

When you choose a network topology, you constrain your space of possibilities (hypothesis space) to a specific series of tensor operations, mapping input data to output data.

Imperial College London

Step 3: train the model

```
model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 5s 87us/step - loss: 0.5033 - acc: 0.8242
Epoch 2/5
60000/60000 [==============================] - 5s 80us/step - loss: 0.3803 - acc: 0.8643
Epoch 3/5
60000/60000 [==============================] - 5s 85us/step - loss: 0.3399 - acc: 0.8758
Epoch 4/5
60000/60000 [==============================] - 5s 87us/step - loss: 0.3141 - acc: 0.8855
Epoch 5/5
60000/60000 [==============================] - 5s 88us/step - loss: 0.2941 - acc: 0.8917
```

Only "epochs" (and "batch size") matter here.

Imperial College London

Step 4: evaluate

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
10000/10000 [==============================] - 1s 50us/step
Test accuracy: 0.8734
```

Imperial College London

Step 5: predict

```
predictions = model.predict(test_images)
```

Here, the model has predicted the label for each image in the testing set. Let's take a look at the first prediction:

```
predictions[0]
```

```
array([4.2577299e-06, 7.2840301e-08, 2.3979945e-08, 2.0671453e-06,
       9.1094840e-08, 1.2096325e-01, 1.5182156e-06, 1.9717012e-01,
       1.2066002e-05, 6.8184656e-01], dtype=float32)
```

https://www.tensorflow.org/tutorials/keras/basic_classification

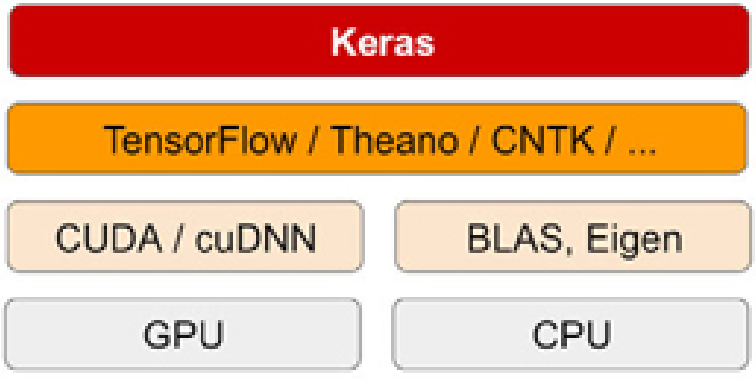
Imperial College London

Keras

"Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano."

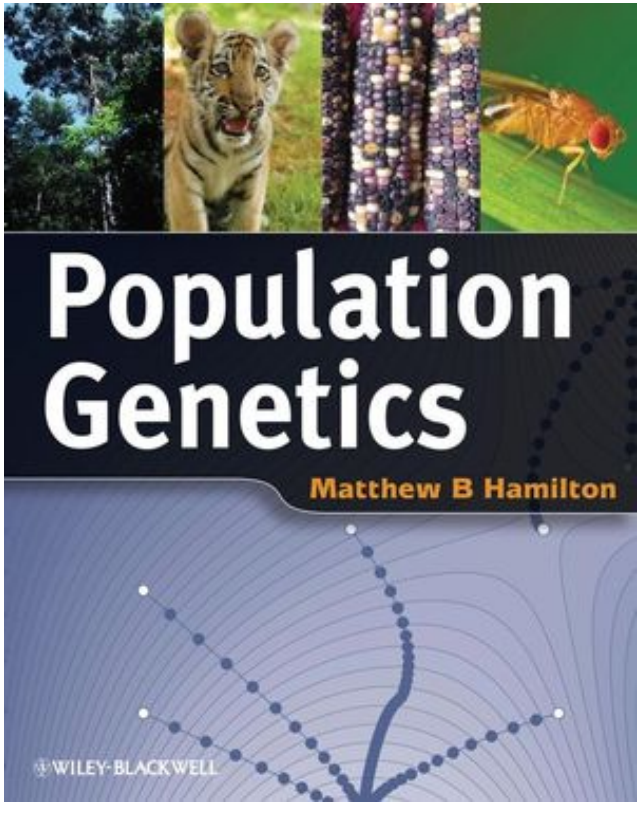
Imperial College London

Keras



Imperial College London

Population genetics



Imperial College London

Population genetics

PLOS | COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

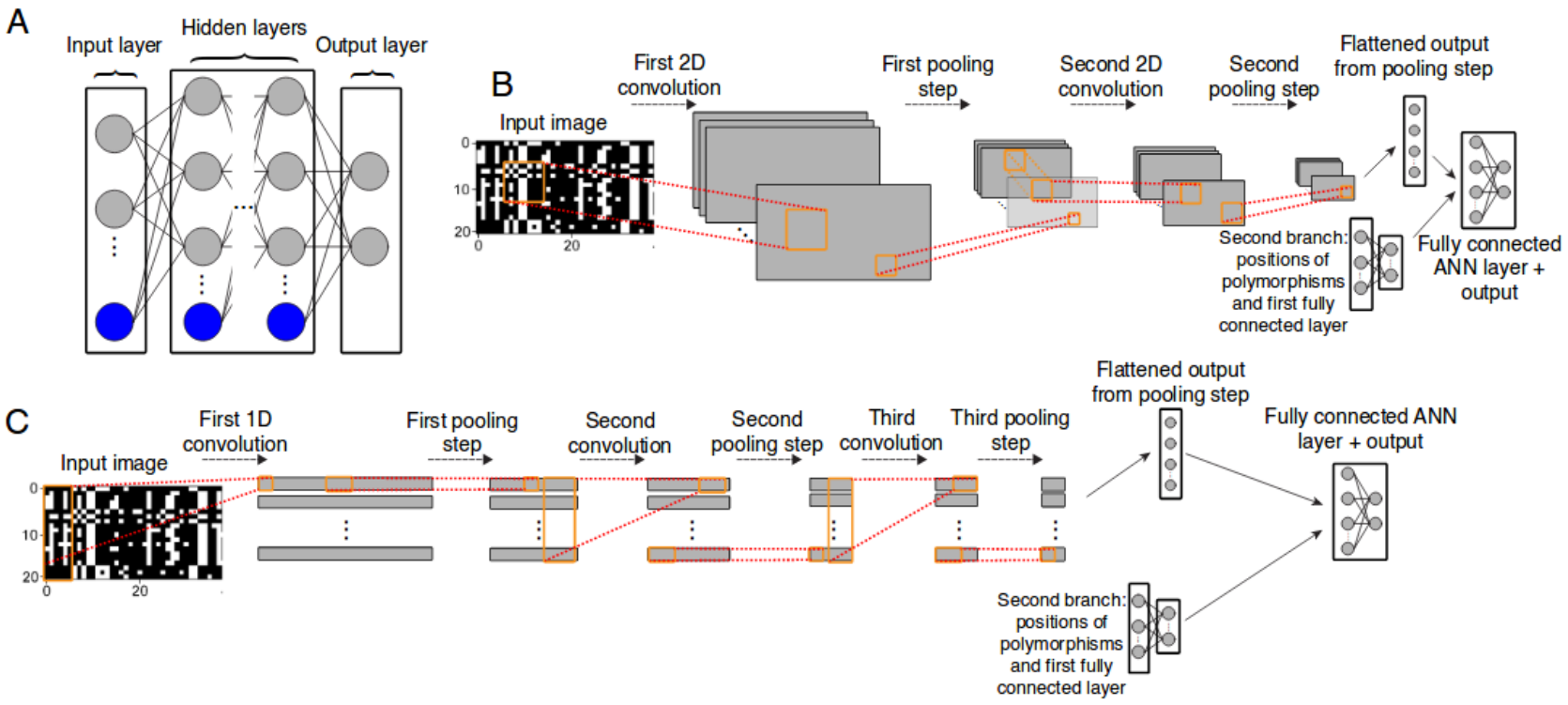
Deep Learning for Population Genetic Inference

Sara Sheehan[1,2]*, **Yun S. Song**[2,3,4,5,6]*

1 Department of Computer Science, Smith College, Northampton, Massachusetts, United States of America, **2** Computer Science Division, UC Berkeley, Berkeley, California, United States of America, **3** Department of Statistics, UC Berkeley, Berkeley, California, United States of America, **4** Department of Integrative Biology, UC Berkeley, Berkeley, California, United States of America, **5** Department of Mathematics, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America, **6** Department of Biology, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America

Imperial College London

Population genetics



Imperial College London

IUCN Red List of Threatened Species

LC: least concern

EN: endangered

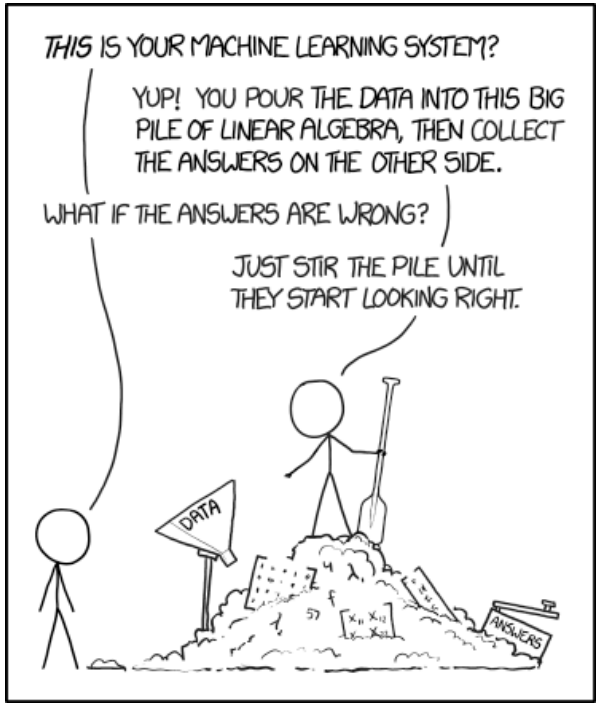
VU: vulnerable

CR: critically endangered

Imperial College London

Well done!

You are all data scientists now!



postdoc and phd position in machine learning applied to population genetics (deadlines Nov 21st and Jan 16th)