Deep Learning: Supervised Learning

Adam Moss

School of Physics and Astronomy

adam.moss@nottingham.ac.uk

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

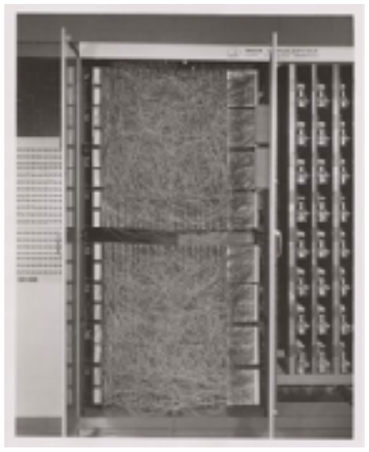
Early Learning

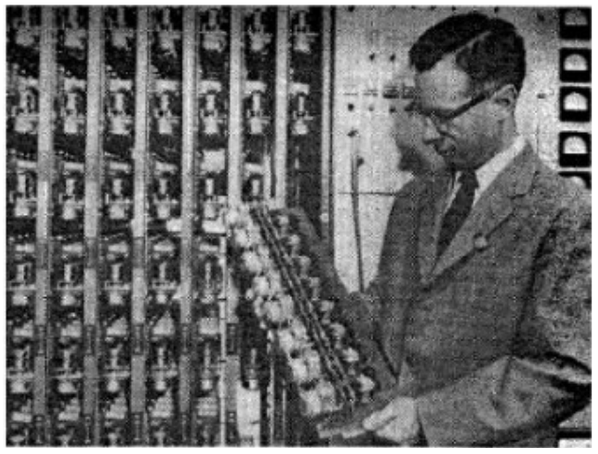
- Feature extraction

Hand crafted feature extraction

Trainable classifier

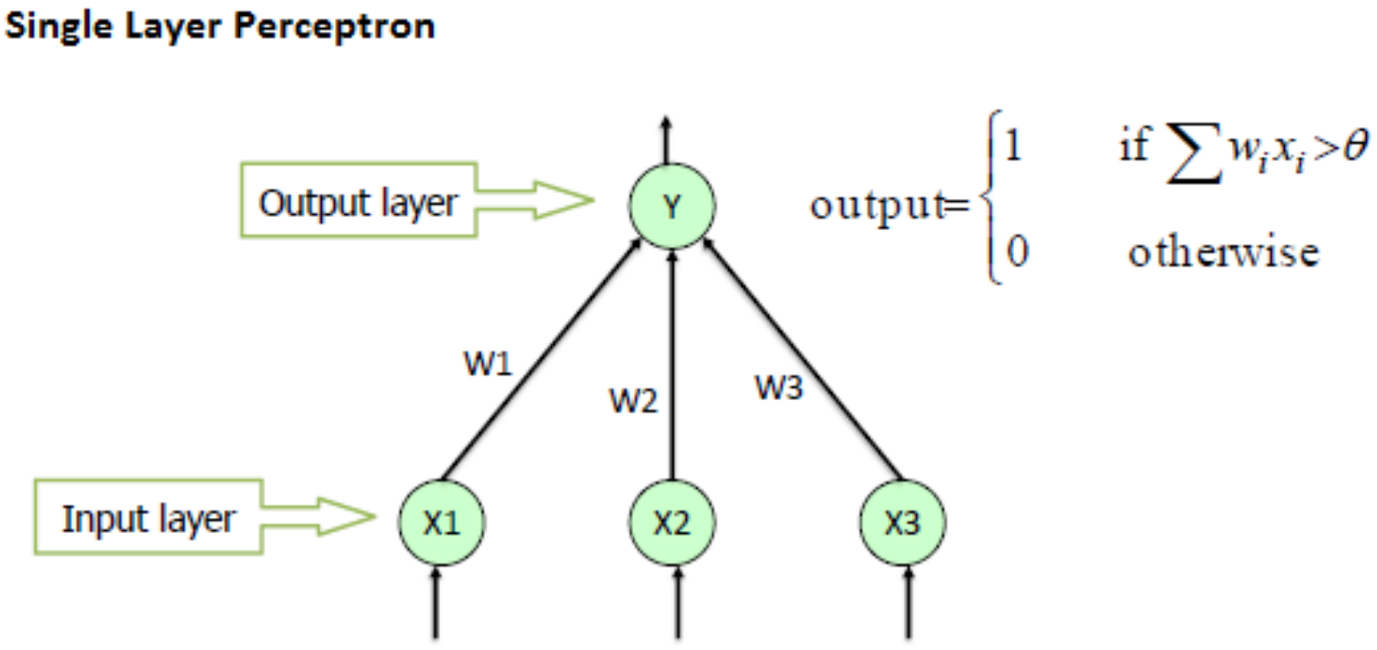
- Perceptron (binary classifier). Mark 1 perceptron machine (1957) used motors, potentiometers!





Perceptron

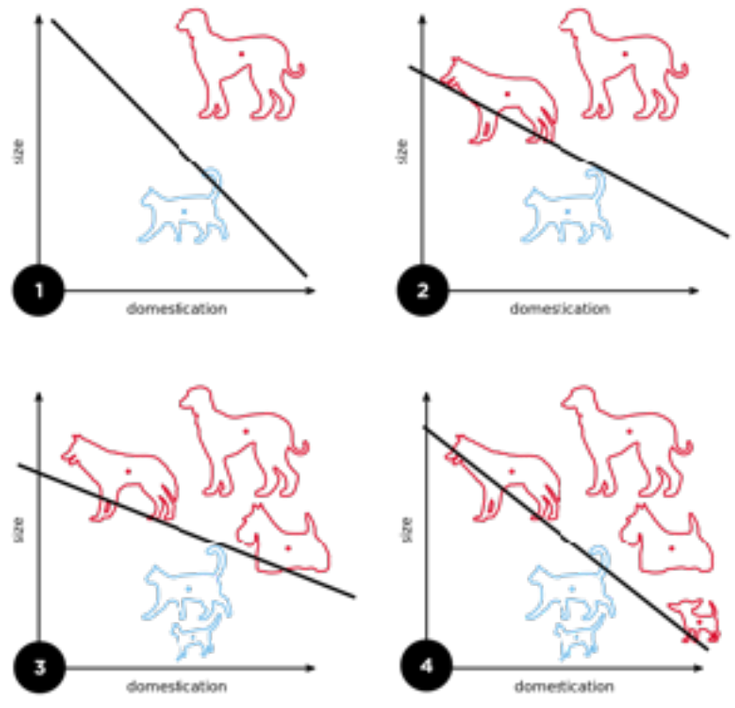
- Simplest perceptron: set of inputs x_i mapped to output y



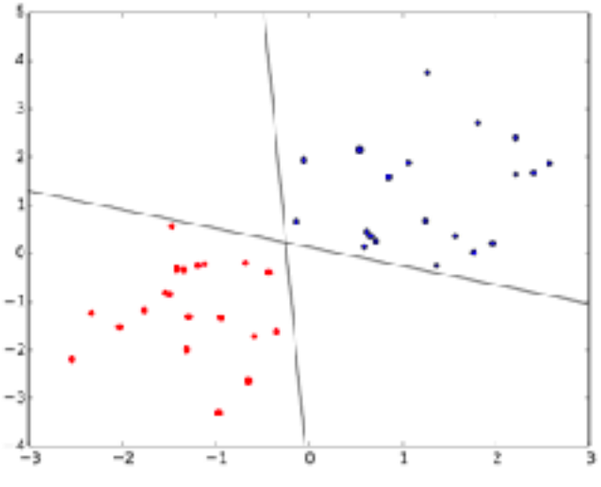
- Each input has a weight w_i
- Weights are trained using **supervised learning**
- Training sets of $D = \{x_{i,j}, d_j\}$ where j is the sample number and d_j the desired output for that sample
- Weights are updated to minimise $\sum_j (d_j - y_j)^2$

Perceptron

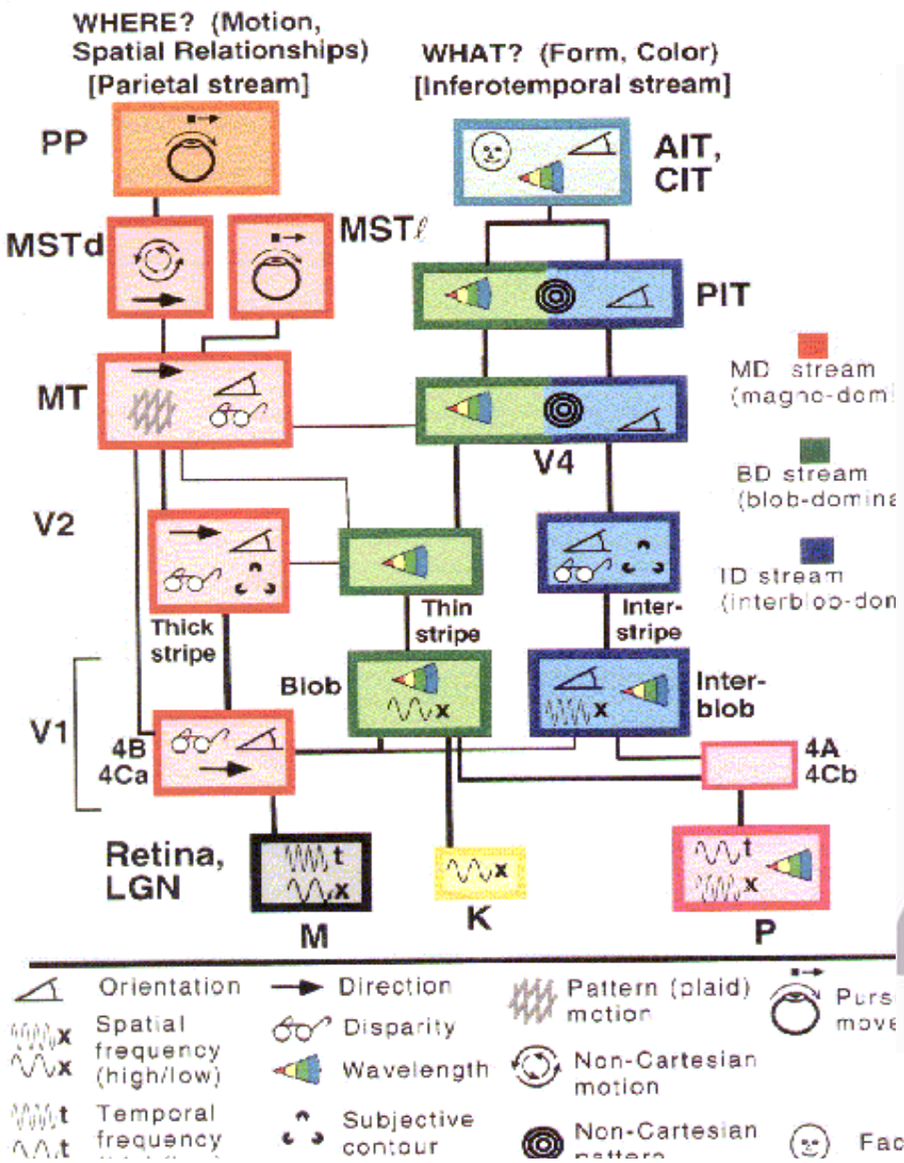
- The perceptron is only able to classify linearly separable training sets
- E.g. if two features are size and domestication

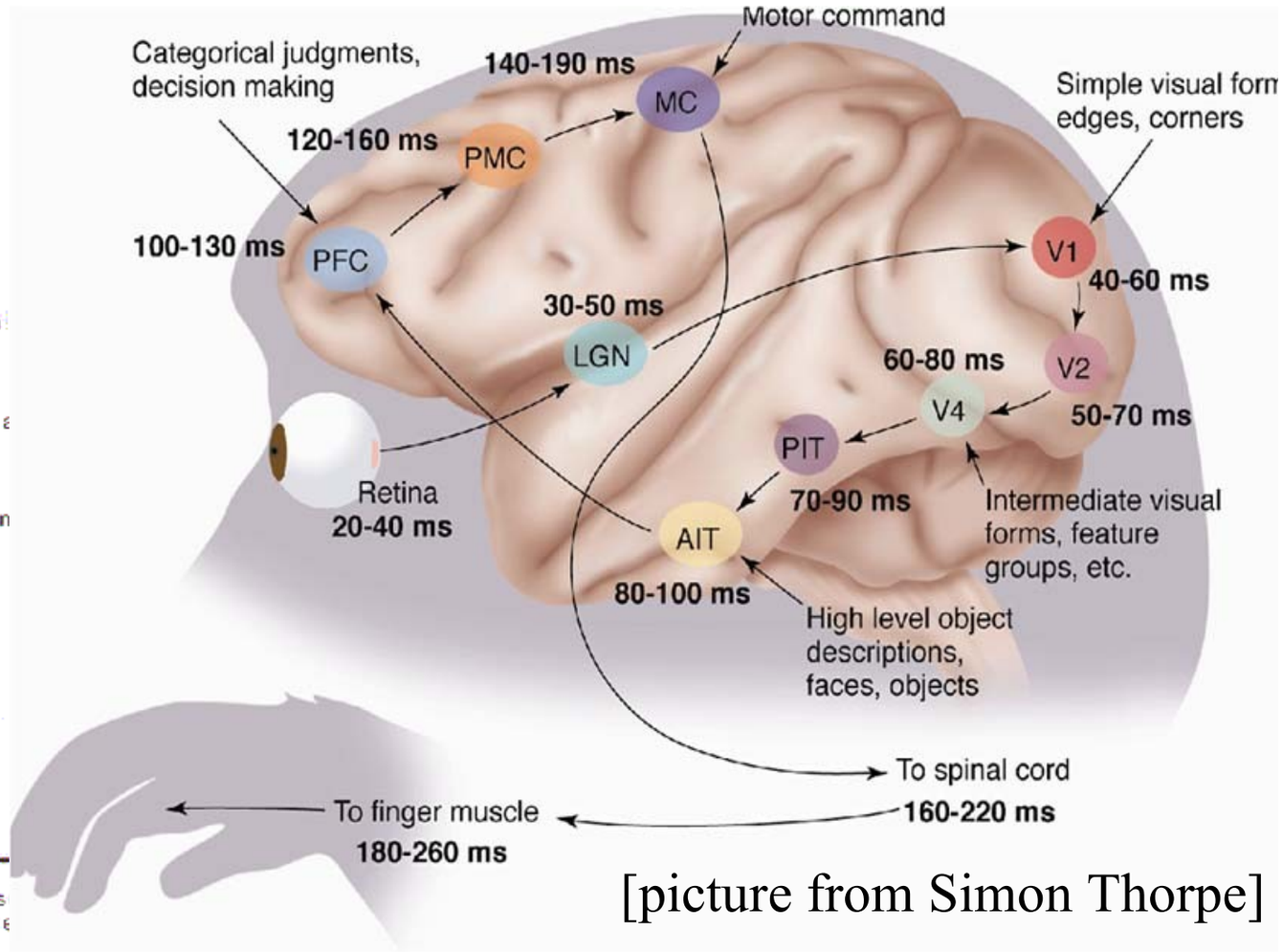


- May admit solutions of different quality (general problem for machine learning if training data is not representative)



Visual Cortex





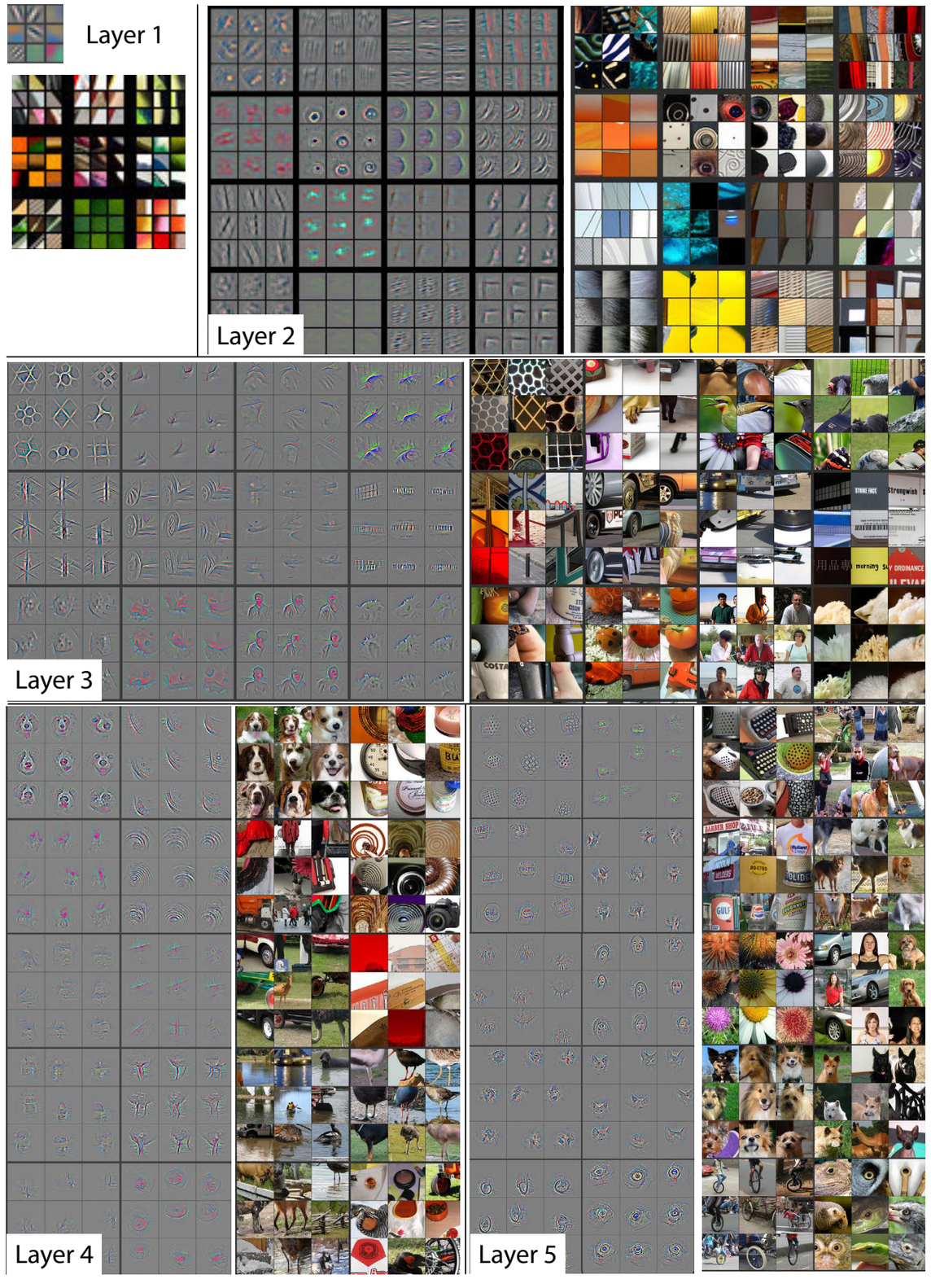
[picture from Simon Thorpe]

[Gallant & Van Essen]

Shallow vs Deep

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

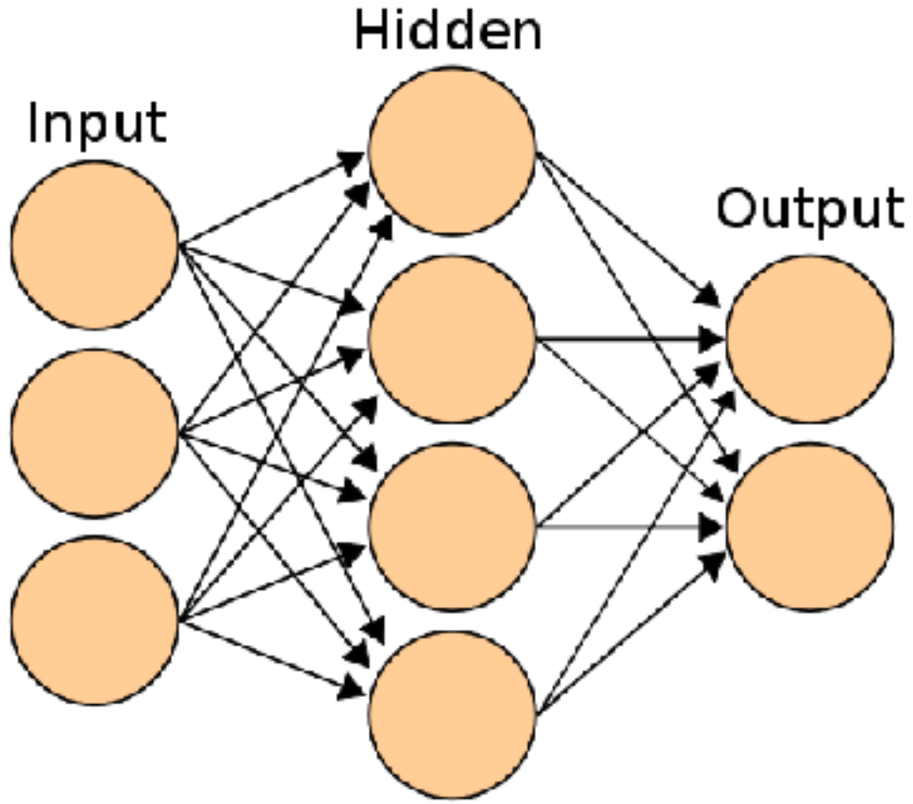
- A network is **deep** if it has more than one layer of non-linear feature abstraction
- Hierarchy of representations with increasing levels of abstraction (e.g. pixel -> edge -> eye -> face)
- Deep networks can store more memory than equivalent number of units in a single layer



MultiLayer Perceptron

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

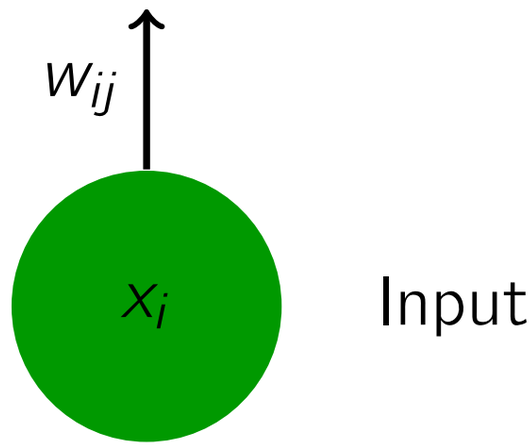
- Type of **feedforward** artificial neural network
- Can distinguish data which is not linearly separable
- Some neurons use non-linear **activation** functions (functions which map the weighted input to the output of a neuron)
- The brain is thought to work in a similar way when biological neurons are fired
- MLPs use supervised learning to update network weights using **back-propagation**



Back-propagation

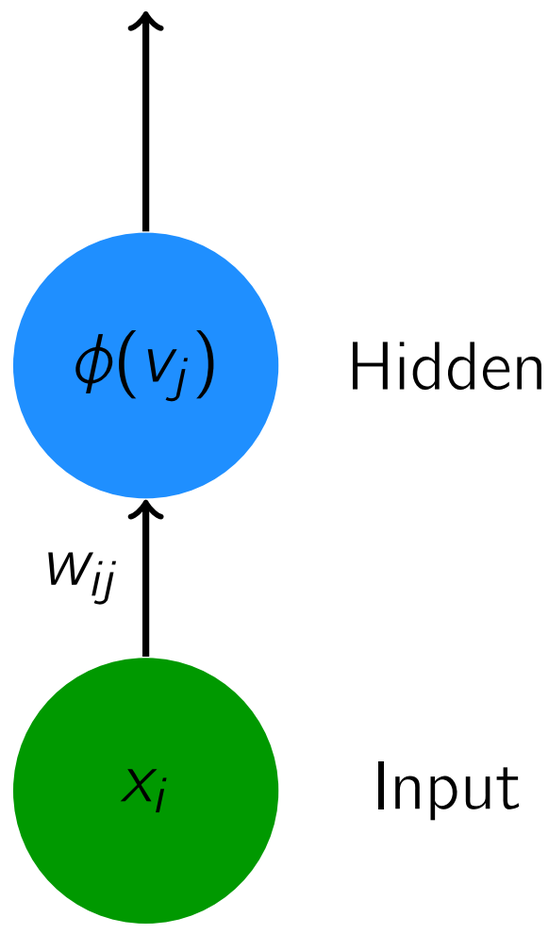
The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

- Weighted inputs $v_j = w_{ij} x_i$



Back-propagation

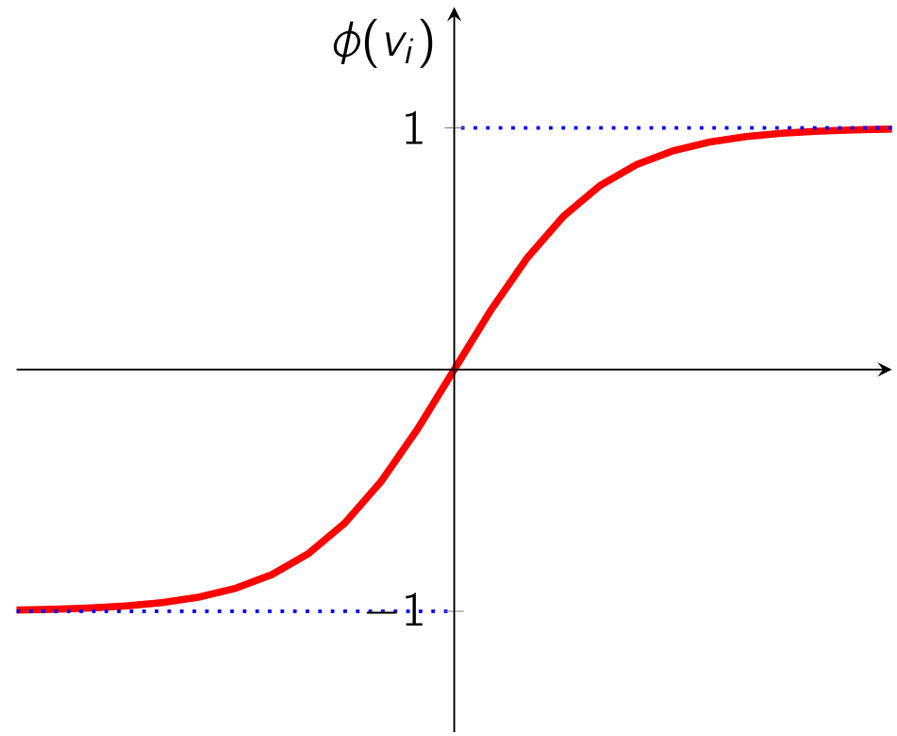
- Weighted inputs $v_j = w_{ij} x_i$
- Activation function $\phi(v_j)$



Back-propagation

- Weighted inputs $v_j = w_{ij} x_i$
- Activation function $\phi(v_j)$

$$\phi(v_i) = \tanh(v_i)$$



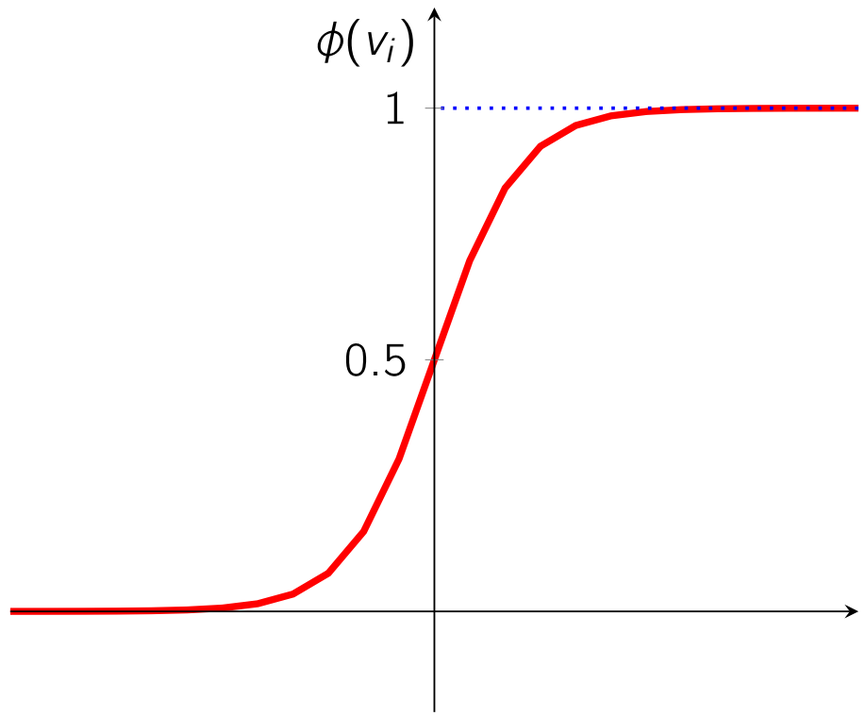
Tanh function

Back-propagation

- Weighted inputs $v_j = w_{ij} x_i$
- Activation function $\phi(v_j)$

$$\phi(v_i) = \tanh(v_i)$$

$$\phi(v_i) = (1 + e^{-v_i})^{-1}$$



Sigmoid function

Back-propagation

- Weighted inputs $v_j = w_{ij} x_i$
- Activation function $\phi(v_j)$

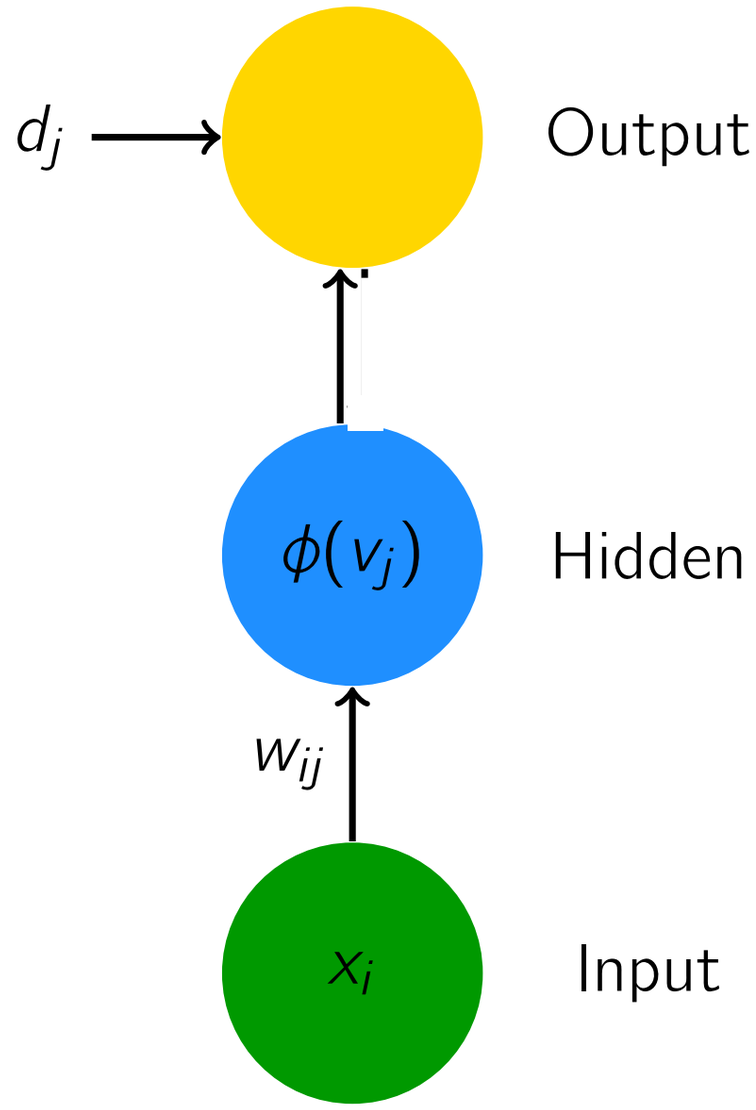
$$\phi(v_i) = \tanh(v_i)$$

$$\phi(v_i) = (1 + e^{-v_i})^{-1}$$

- Loss function

$$e_j = d_j - \phi(v_j)$$

$$\mathcal{E} = \sum_j e_j^2$$



Back-propagation

- Weighted inputs $v_j = w_{ij} x_i$
- Activation function $\phi(v_j)$

$$\phi(v_i) = \tanh(v_i)$$

$$\phi(v_i) = (1 + e^{-v_i})^{-1}$$

- Loss function

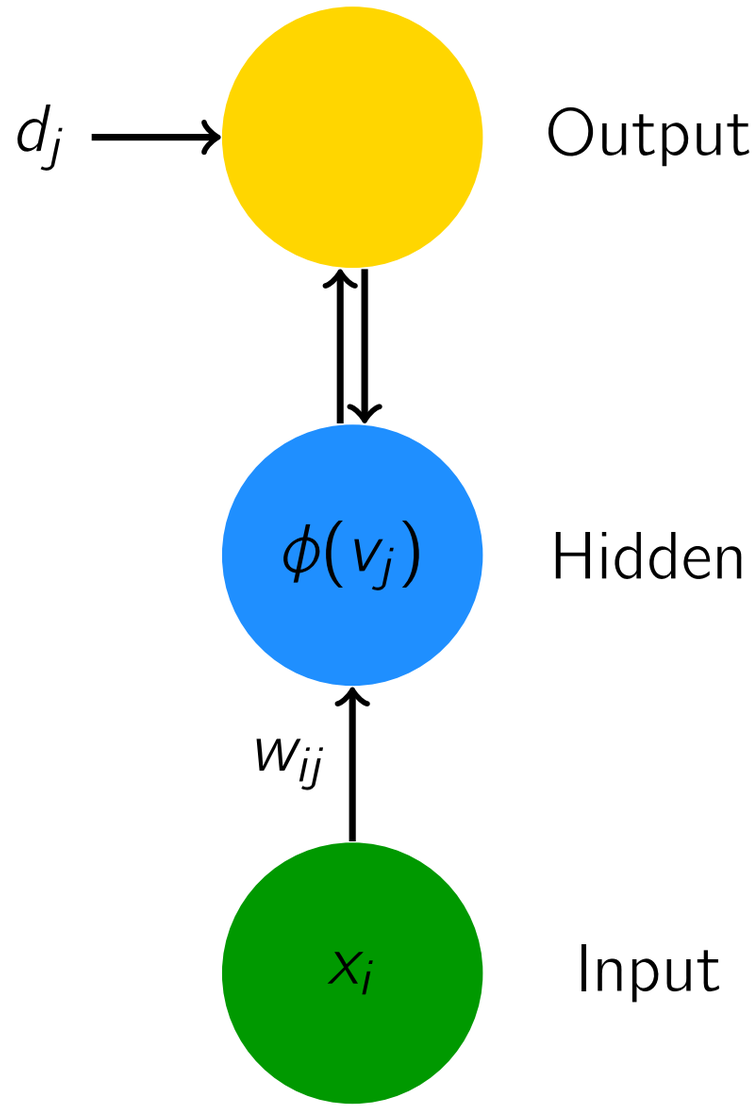
$$e_j = d_j - \phi(v_j)$$

$$\mathcal{E} = \sum_j e_j^2$$

- Back-propagation

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}}{\partial w_{ij}}$$

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}}{\partial \phi(v_j)} \frac{\partial \phi(v_j)}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}}$$



Back-propagation

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

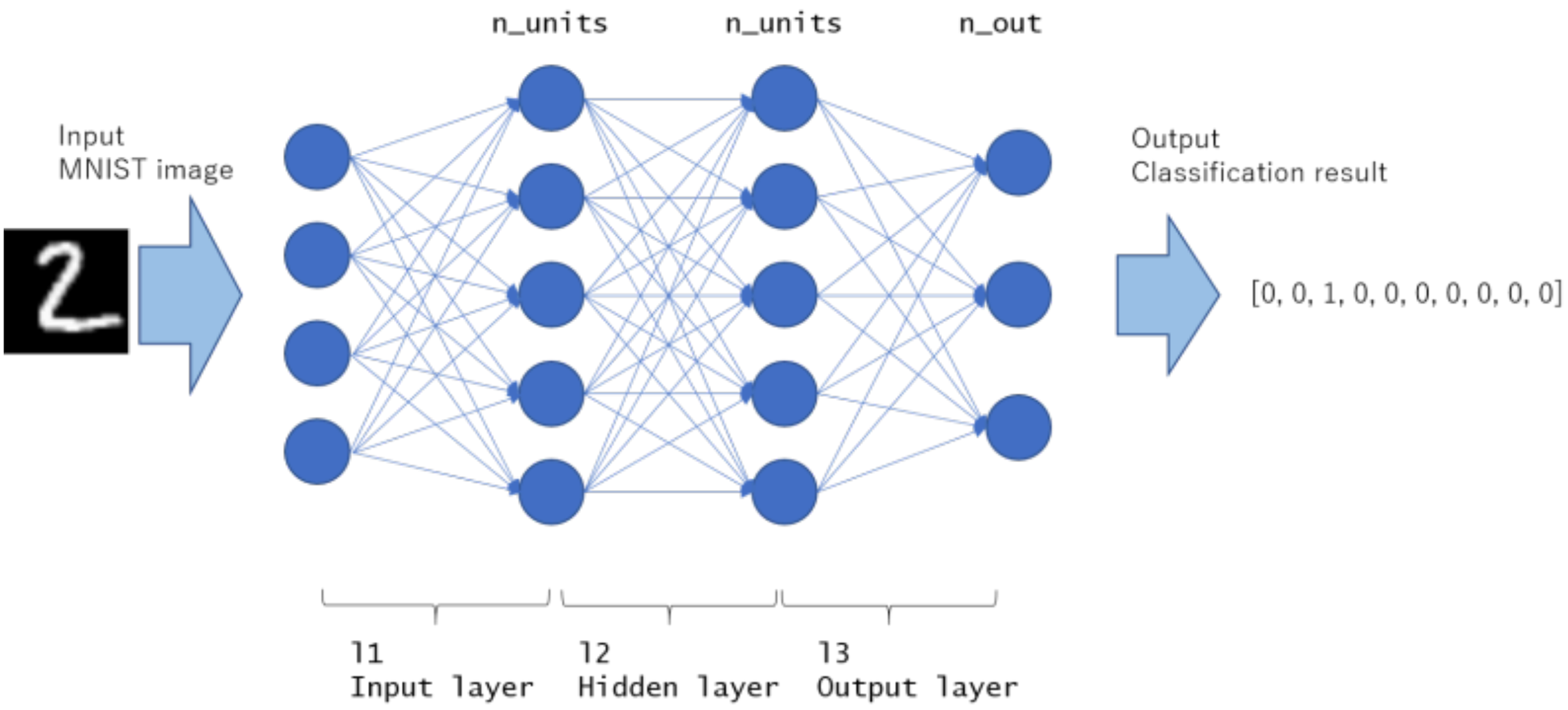
- Can the cortex do back-propagation?
- Maybe or maybe not!
- Many neuroscientists think the brain **can't** back-propagate
  - ➡ Source of supervision signal?
  - ➡ Neurons send all-or-nothing spikes
  - ➡ Neurons must be able to send different signals forward and backward
- But see work of Geoffrey Hinton (e.g. Lillicrap et. al., Nature Communications **7,** 2016) who presents arguments how the brain can back-propagate
- How much should we be led by the function of the brain when developing deep learning algorithms?

Hello World

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

- MNIST is `Hello Word' of deep learning
- Black and white images of integers from 0 to 9
- 28 x 28 pixel images
- 60,000 training images and 10,000 test images

MNIST MLP

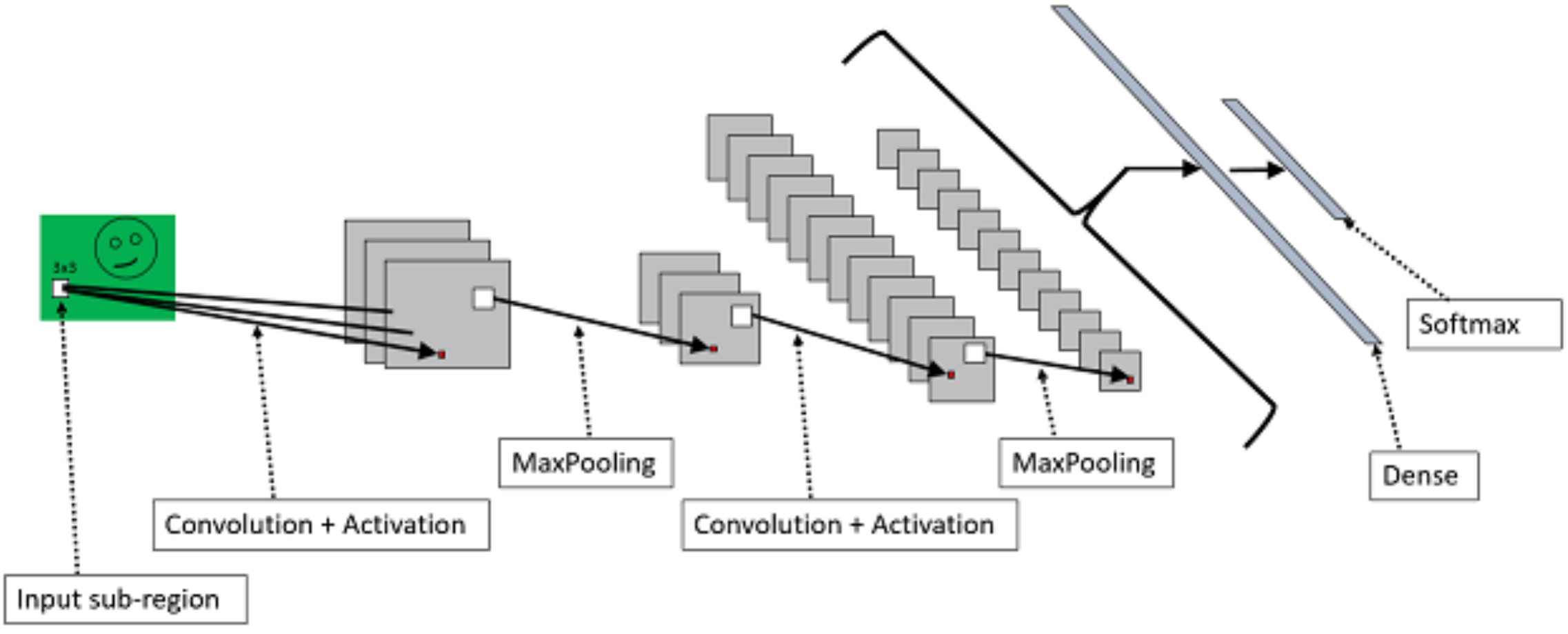


MNIST MLP

- Keras Python library (uses TensorFlow or Theano as backend)
- 2 hidden layers each with 512 units
- Dropout switches off random fraction of connections during training
- Softmax activation turns output into classification probabilities
- Gets to 98.40% test accuracy after 20 epochs

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop


batch_size = 128
num_classes = 10
epochs = 20

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

MNIST CNN



MNIST CNN

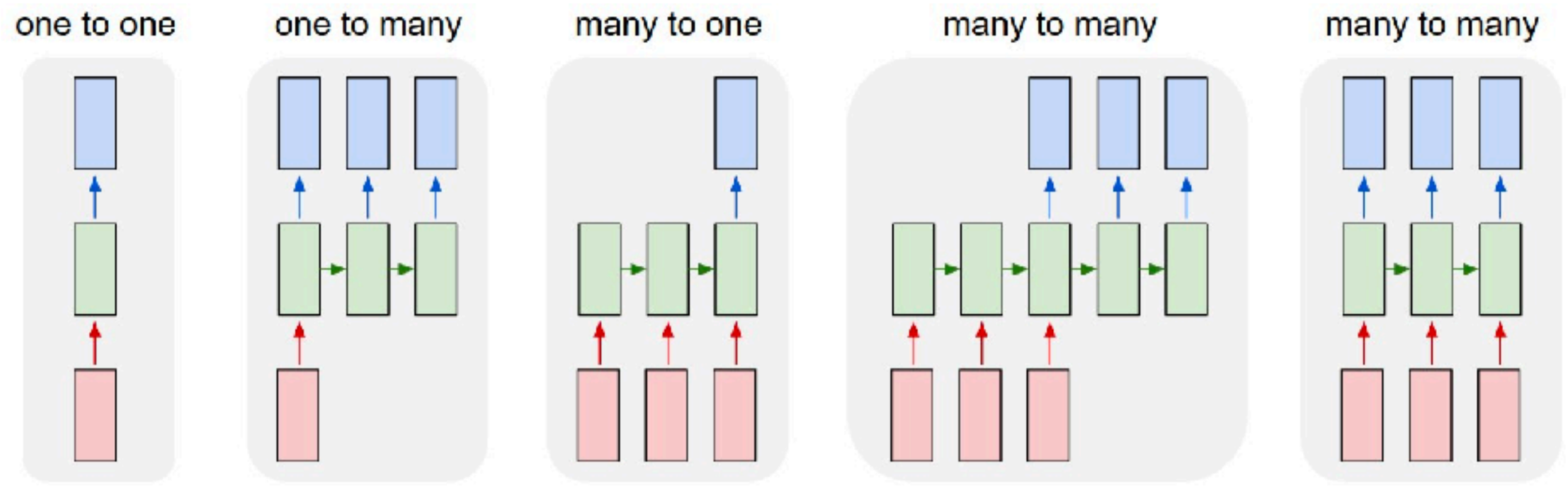
- First uses 32 convolutional units with 3x3 kernel
- Next 64 convolutional units with 3x3 kernel
- Max pooling down-samples units
- Dropout
- Dense fully connected layer
- Additional dropout and softmax activation
- Gets to 99.25% test accuracy after 12 epochs
- Each epoch takes ~16 seconds on K520 GPU

```
batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Recurrent Network

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

- Recurrent neural networks (RNNs) are a class of neural network that can learn about sequential data (e.g. time series, natural language)



E.g. image classification

E.g. image caption

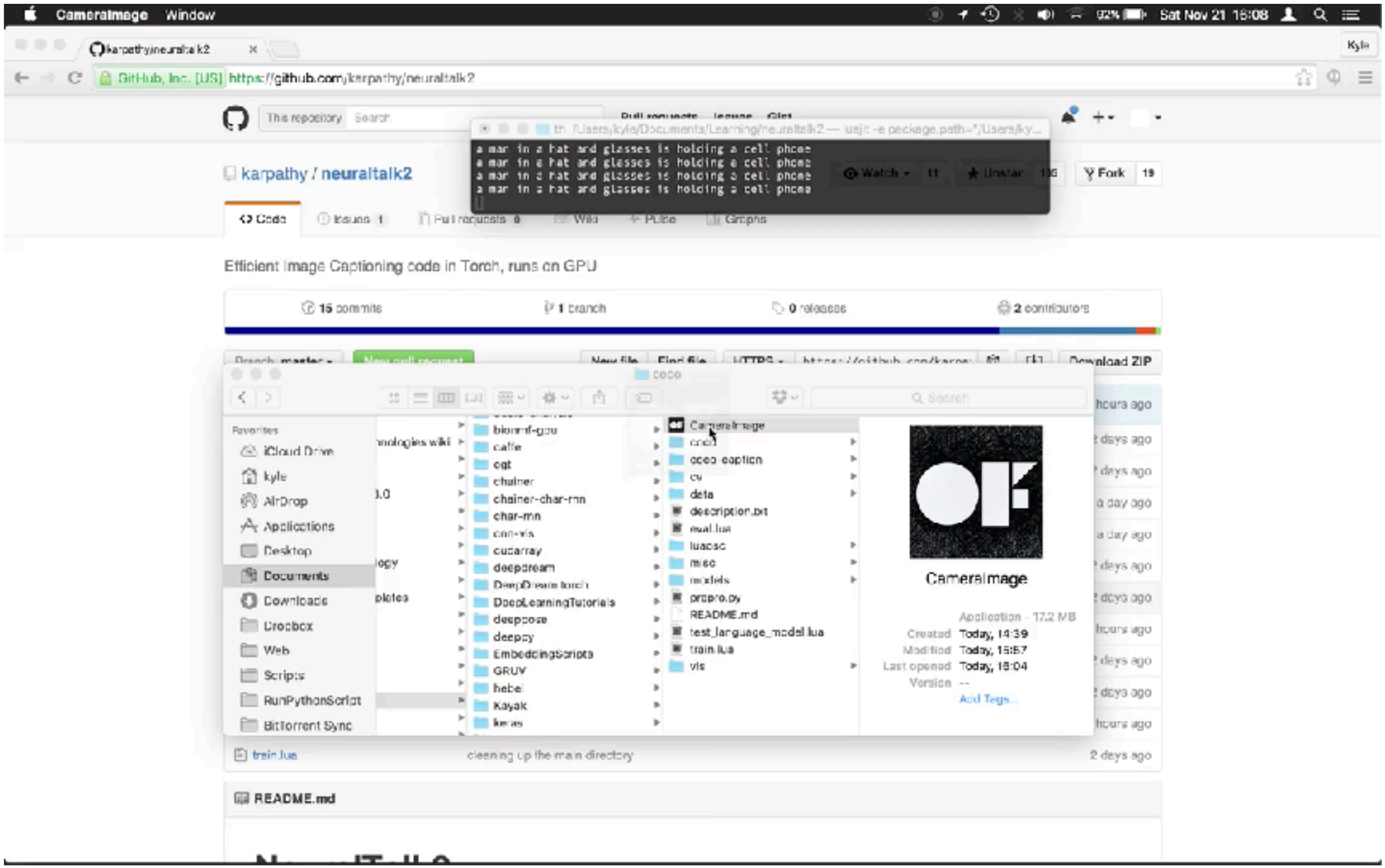
E.g. sentiment analysis

E.g. language translation

E.g. predict next word in sentence

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Network



Recurrent Network

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

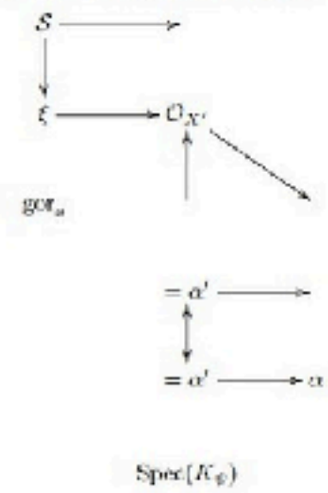
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "Field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\text{étale}}}) \longrightarrow \mathcal{O}_{X_i}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\gamma}^{\vee})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/