Quick Review

Perceptron

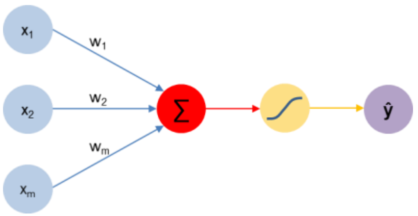
- A linear sum
- Non-linear activation function

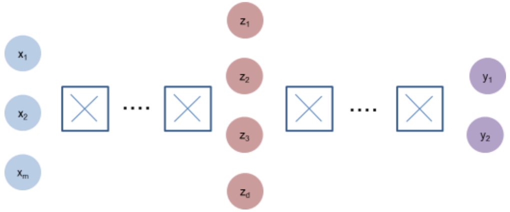
Neural Network

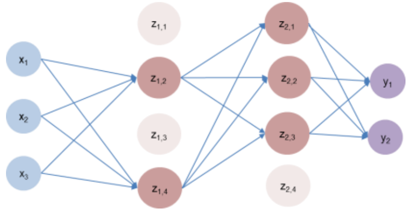
- Stacking of perceptrons
- Optimisation through back propagation

Training

- Regularisation optimization
- Learning rate







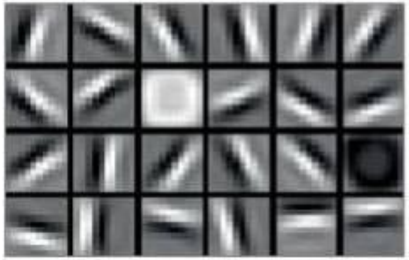
Convolutional Neural Networks

Until now we looked at fully connected multi-layer perceptrons.

However these aren’t particularly good with images on their own!

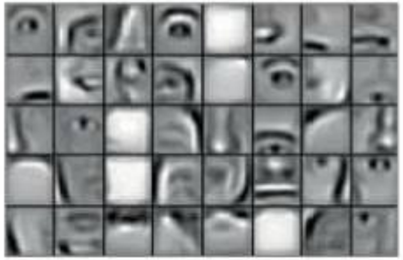
Millions of images

Low level features



Lines & Edges

Mid level features



Eyes & Nose & Ears

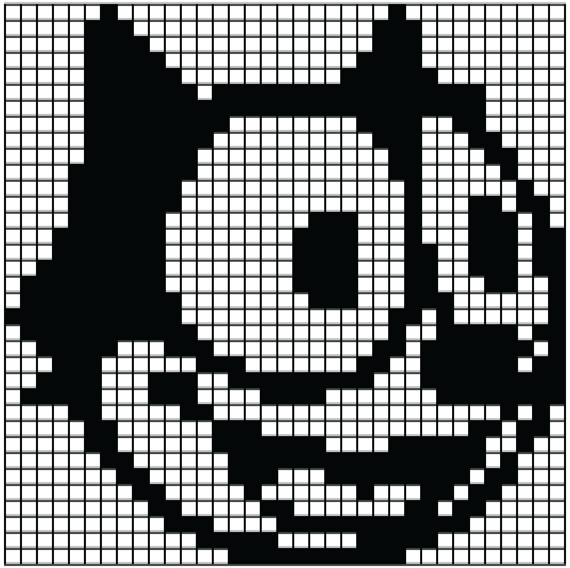
High level features

Facial Structure

Convolutional Neural Networks

How can we help a computer ‘see’?

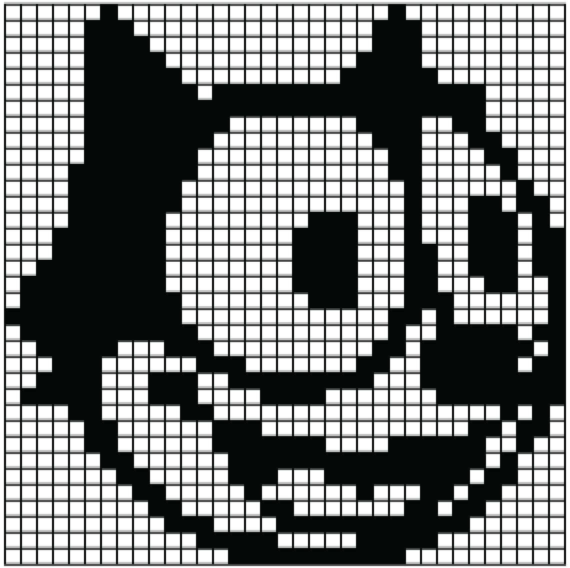
Black: 0
White: 1



Convolutional Neural Networks

How can we help a computer ‘see’?

Black: 0
White: 1



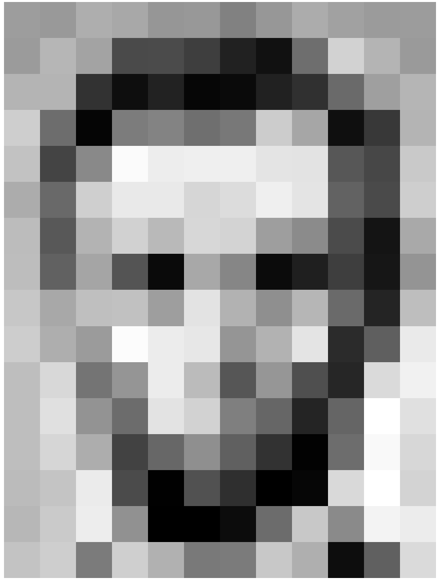
1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	1	1	1
1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1
1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	1	1	1	0	0	0	1	0	1
1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	0	1	1	1	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	1	1	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	1	1	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	1	1	1	0	0	1	1	0
1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0
1	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1
1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0	1	1
1	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1
1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1
1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

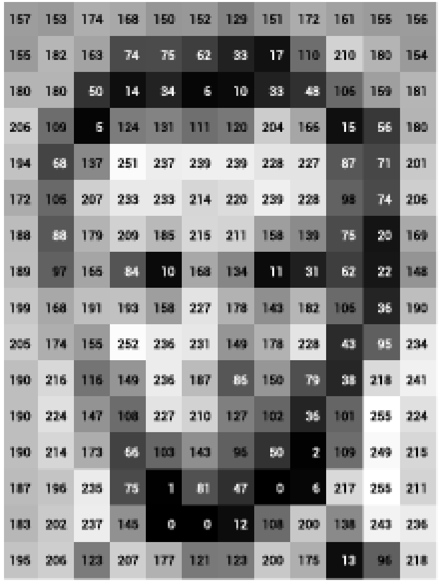
35x35

Convolutional Neural Networks

How can we help a computer ‘see’?

Grey scale





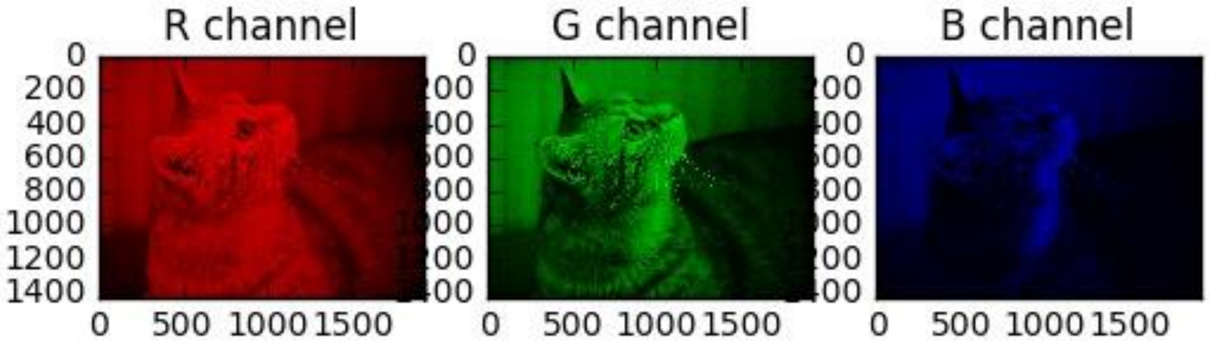
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Convolutional Neural Networks

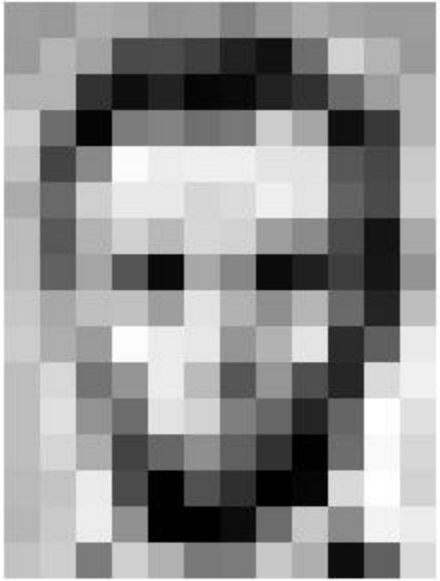
How can we help a computer ‘see’?

For colour images: break into RGB channels

We get a 3d matrix that describes our image.



Convolutional Neural Networks



Input image

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel representation

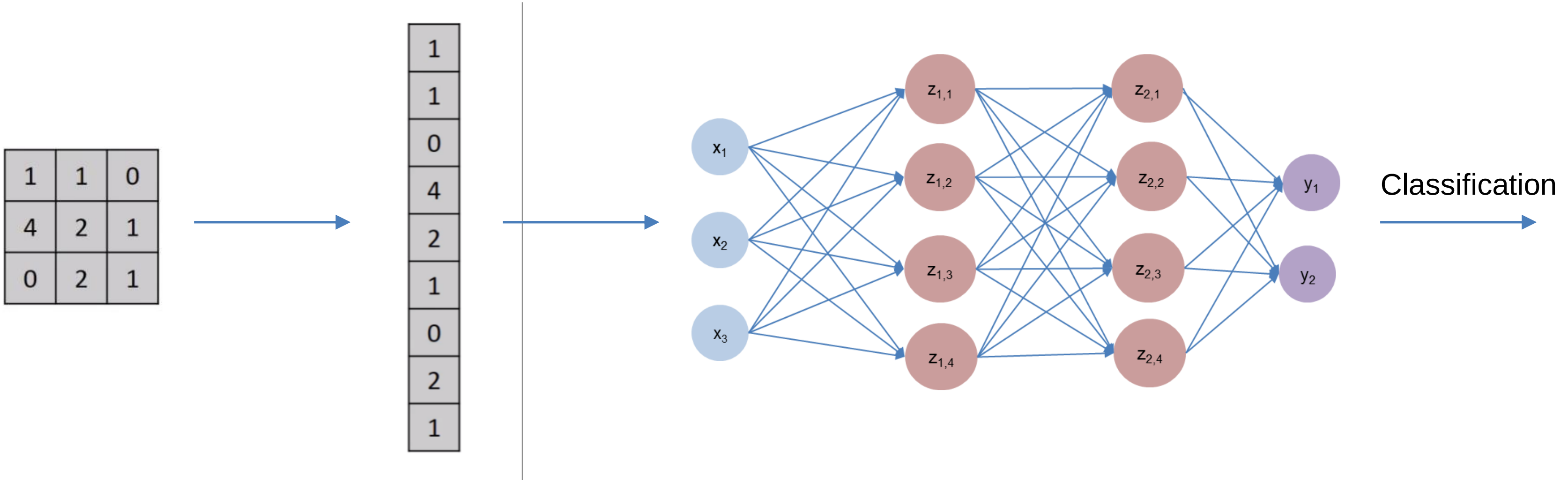
Classification

Lincoln

Trump

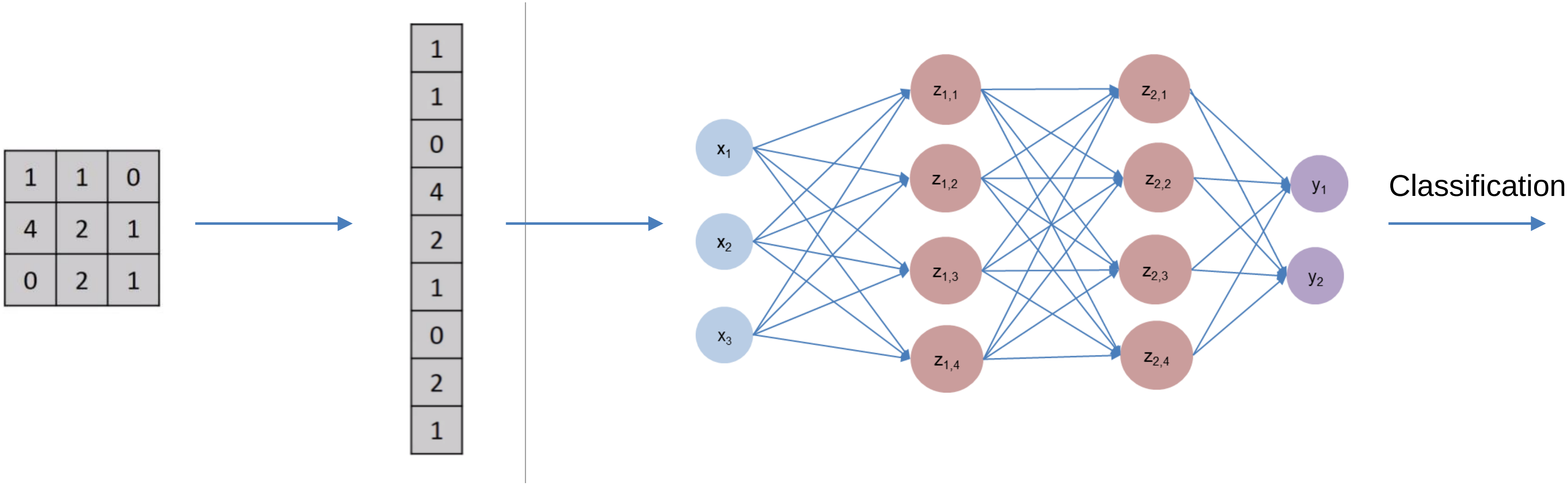
Why the convolutions?!

We could vectorise the image?
Stick it straight into an ANN



Why the convolutions?!

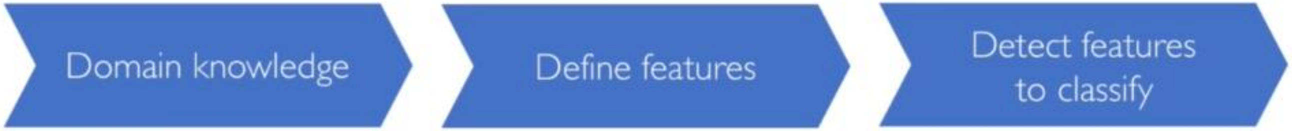
We could vectorise the image?
Stick it straight into an ANN



- *We lose the spatial relationships between pixels!*
- *We introduce a very large number of parameters*

Why the convolutions?!

We could define features manually…



Why the convolutions?!

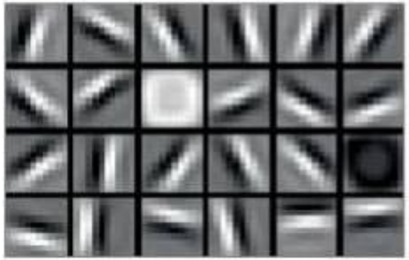
We could define features manually…

Learning feature representations

A hierarchy of features that help us describe the image

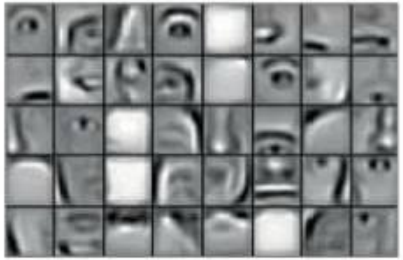
Millions of images

Low level features



Lines & Edges

Mid level features



Eyes & Nose & Ears

High level features

Facial Structure

Identifying similar objects

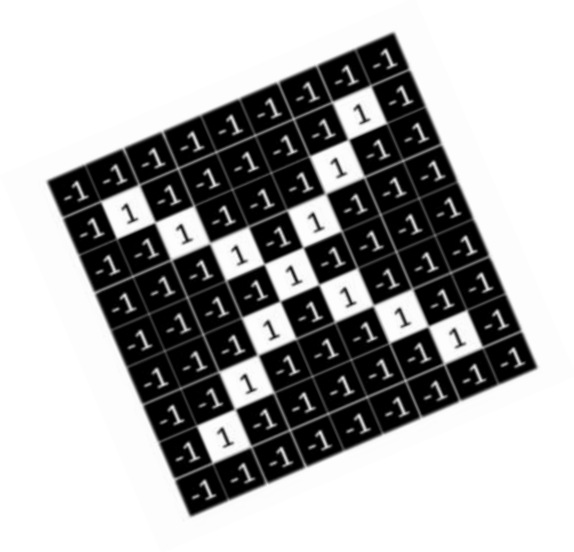
Both images show an X.
Directly comparing the elements of each image would suggest they are different images
We want to identify common features across the two images!

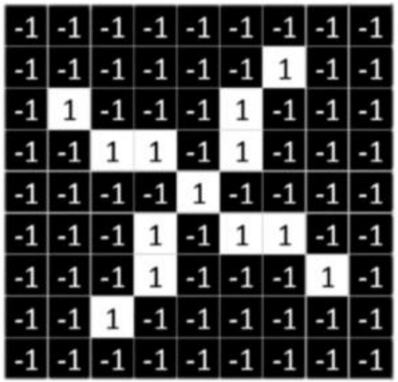
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Identifying similar objects

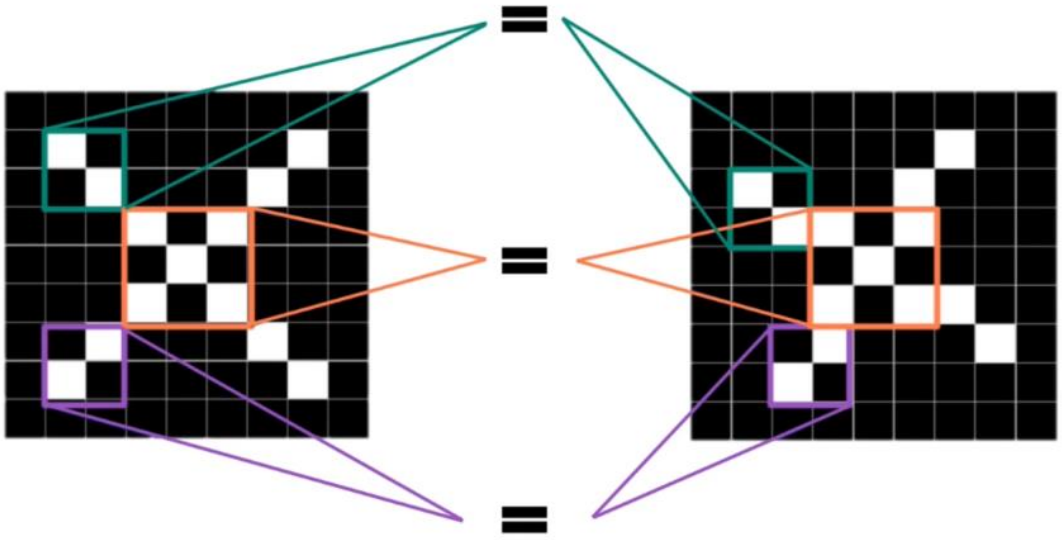
Both images show an X.
Directly comparing the elements of each image would suggest they are different images
We want to identify common features across the two images!





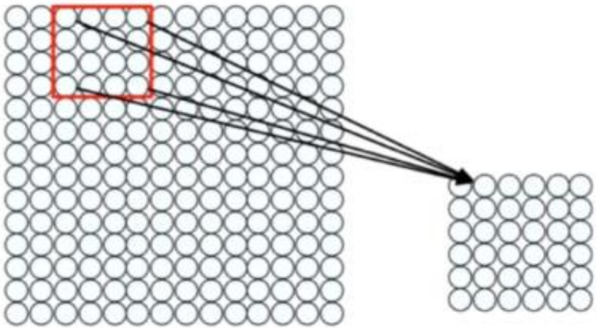
Identifying similar objects

There are common features in both that we can learn.



Learning feature representations

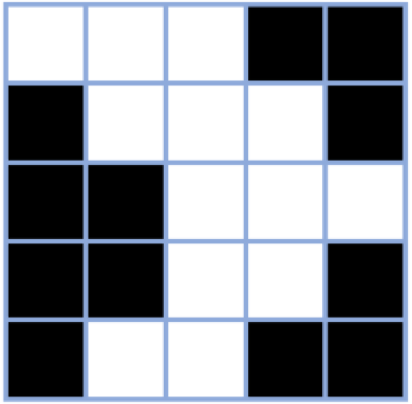
Apply a filter → Shift filter across image → Construct a new ‘convolved’ matrix



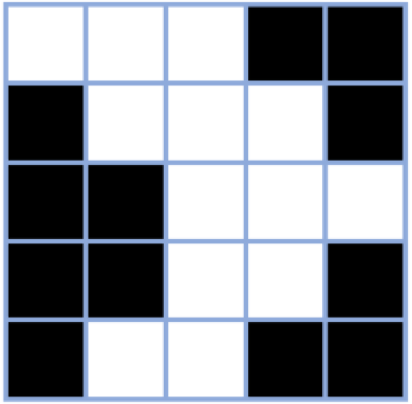
Example:

- 4x4 filter
- Apply filter to the input image
- Element-wise multiplication of pixels with filter matrix
- Shift by 1 pixel to the right and repeat

Convolution operation



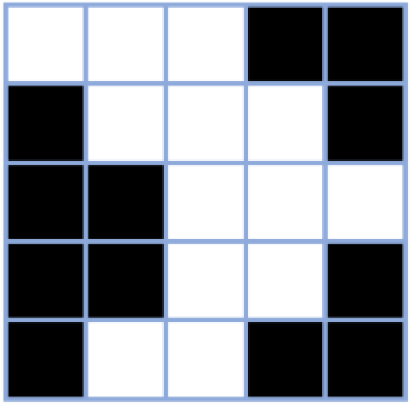
Convolution operation

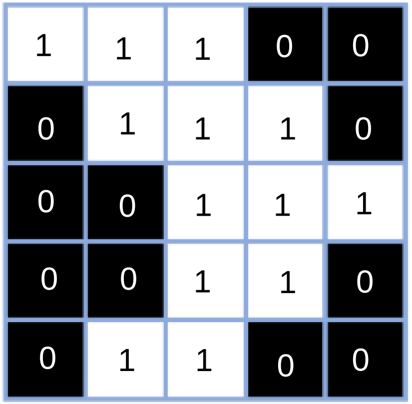


1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

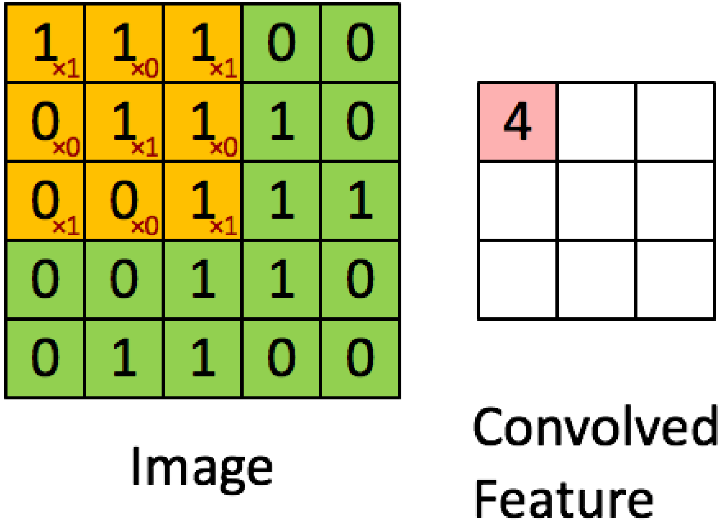
$$kernel = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Convolution operation

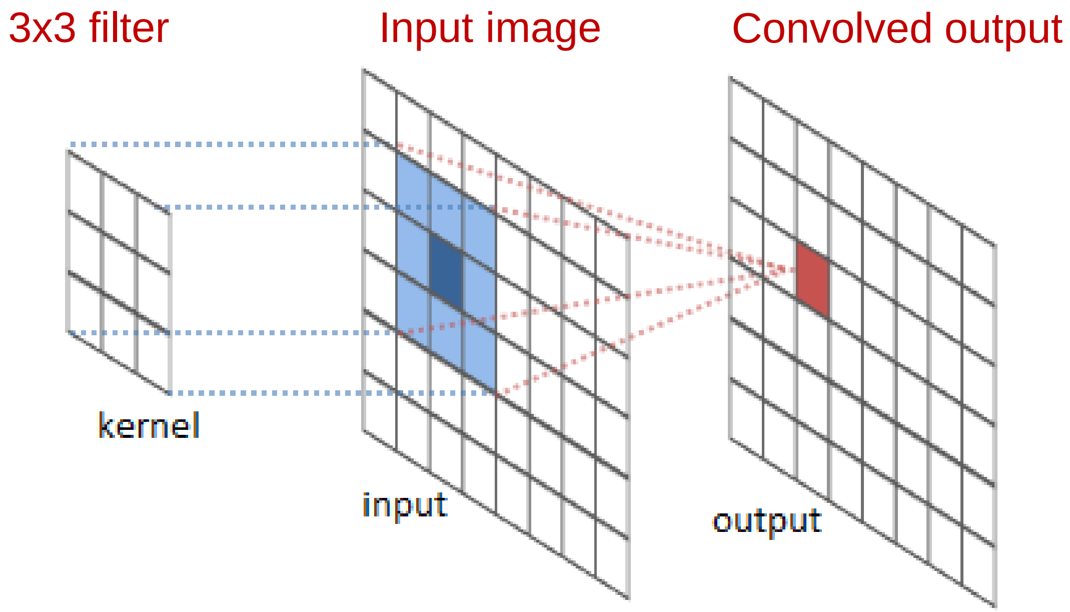




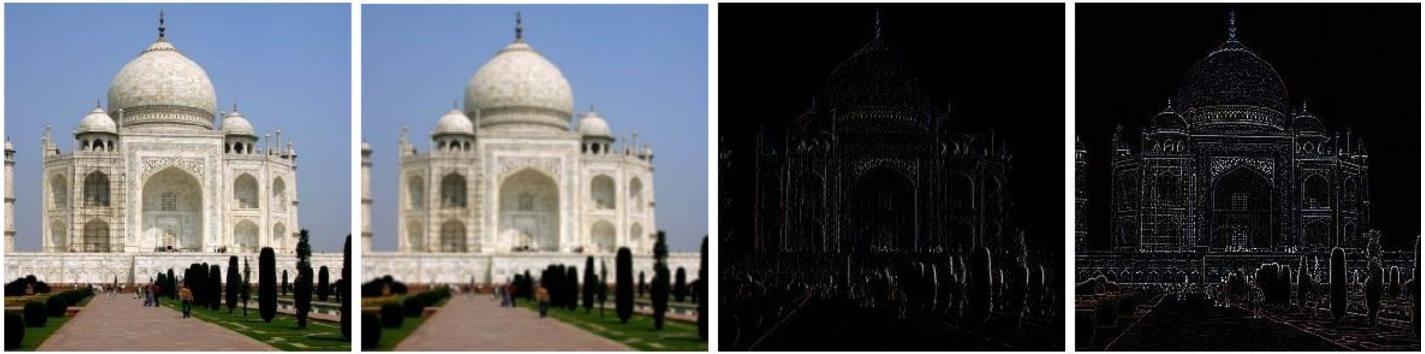
$$kernel = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



Kernels


3x3 filter
Input image
Convolved output
kernel
input
output

Kernels



(a) Original image.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

(b) Blurred.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(c) Detect vertical edges.

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

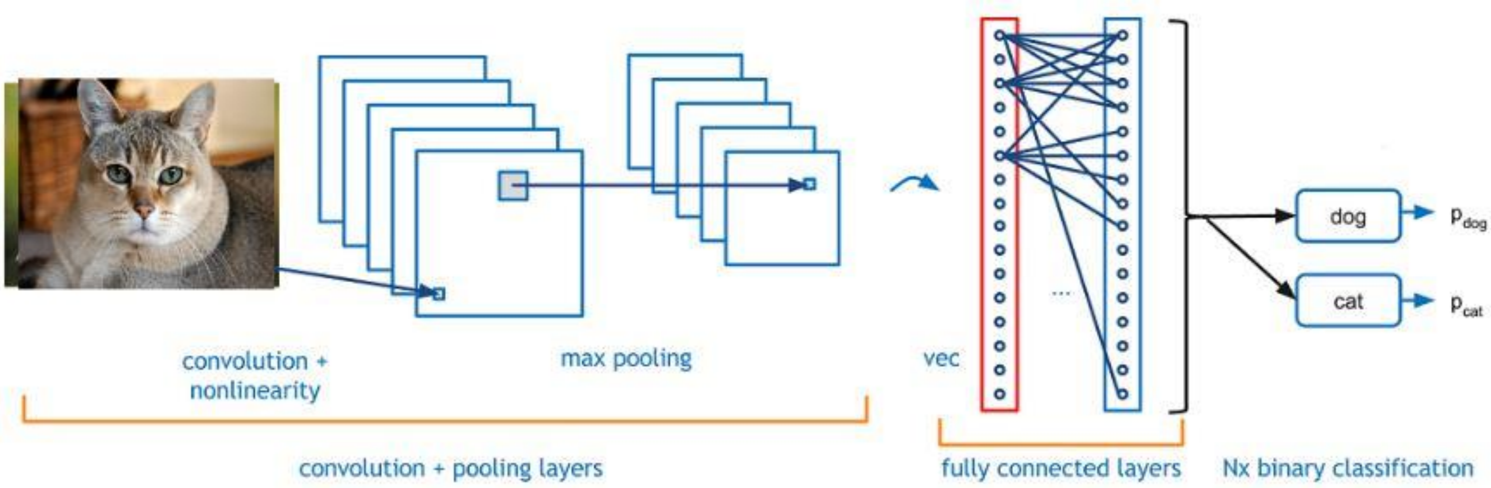
(d) Detect all edges.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

c) we subtract two adjacent pixels. When side by side pixels are similar, this gives us approximately zero. On edges, however, adjacent pixels are very different in the direction perpendicular to the edge. Knowing that results differs from zero will result in brighter pixels, you can already guess the result of this type of kernel.

4 main steps for CNN classification

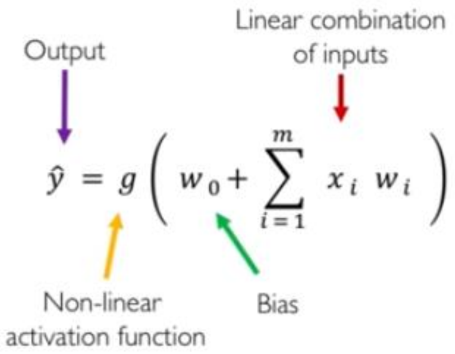
1. Convolution – Apply filters (*we learn the filters!)*
2. Non-linear function – Often ReLU
3. Pooling operation – reduce matrix size, often using max pooling
4. Fully connected MLP



Learning weights of filter

In lecture 1 we saw a neuron in a hidden layer.

Each neuron, weighted combination of inputs plus a bias put through a non-linear function



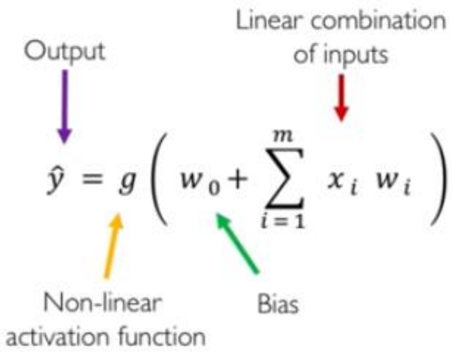
Learning weights of filter

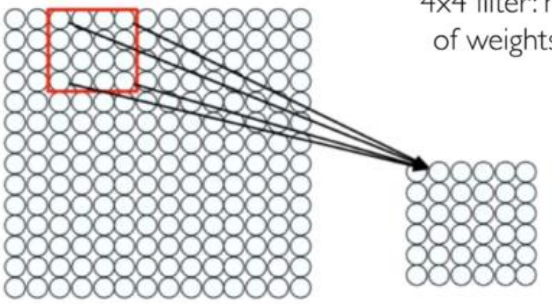
In lecture 1 we saw a neuron in a hidden layer.

Each neuron, weighted combination of inputs plus a bias put through a non-linear function

Here each neuron only ‘sees’ a patch before it. Not fully connected. Defines local connectivity.

1. Apply a window of weights
2. Computer linear combinations
3. Activating with non-linear function





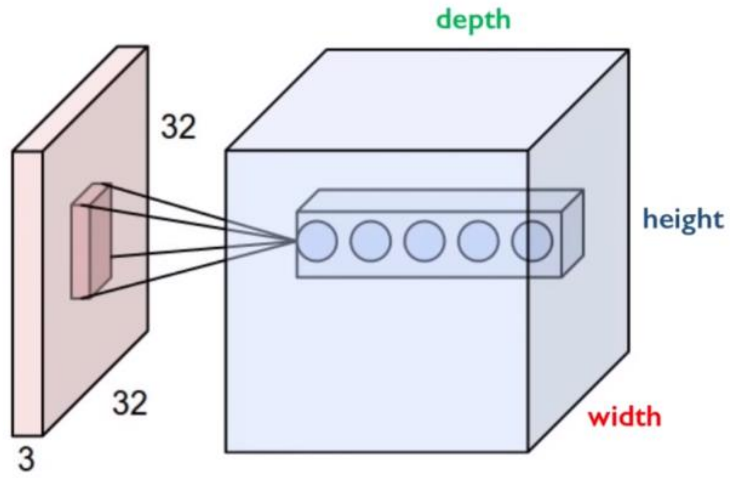
4x4 filter: matrix of weights w_{ij}

$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}\, x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

We can learn many filters

Using multiple features gives us an output **volume** instead of matrix.



Layer Dimensions:
H x W x D

H and D are spatial dimensions of our images

D = number of filters

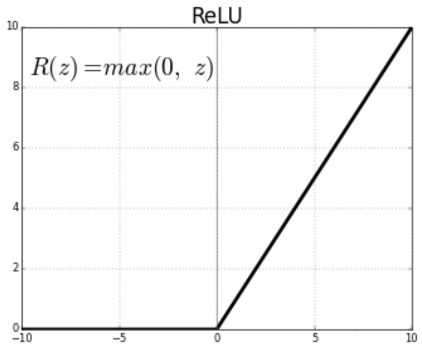
Stride:
Filter step size

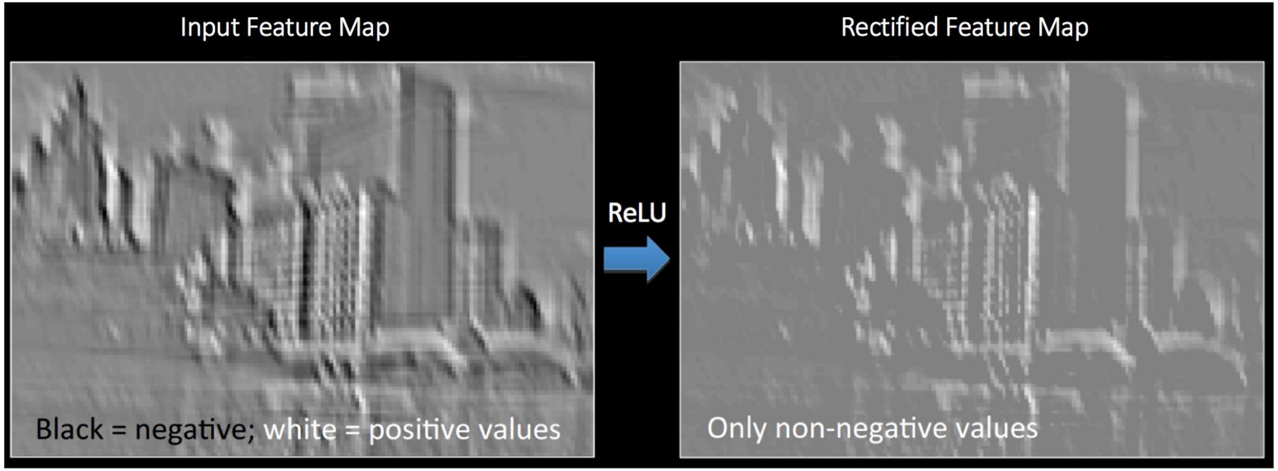
Receptive field:
Locations in input image that a node is connected to.

Introducing non-linearity

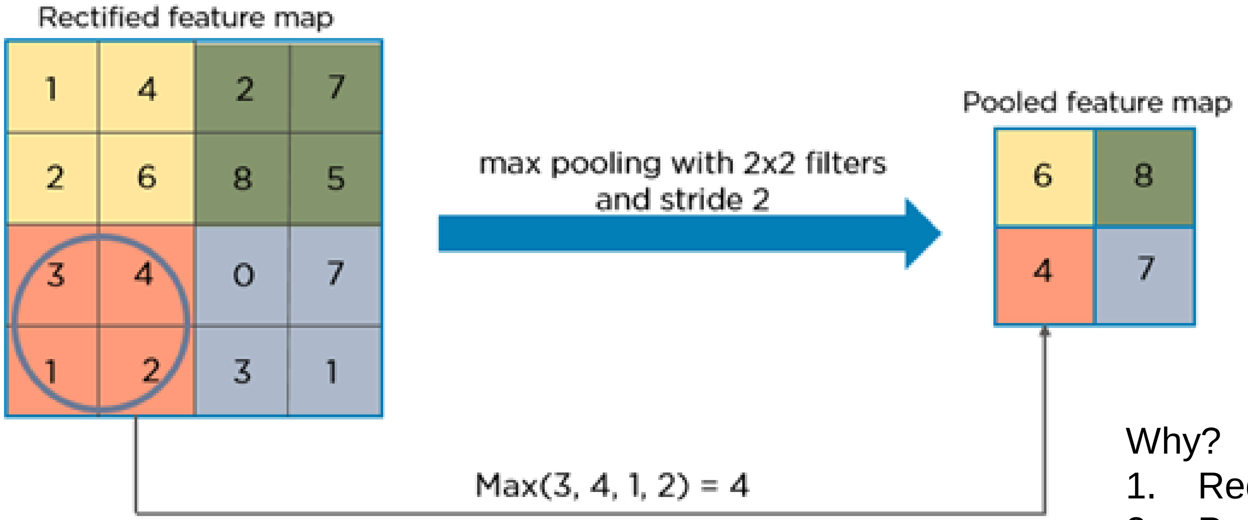
ReLU function is most common for CNN’s

It sets all negative values in our feature map to zero!





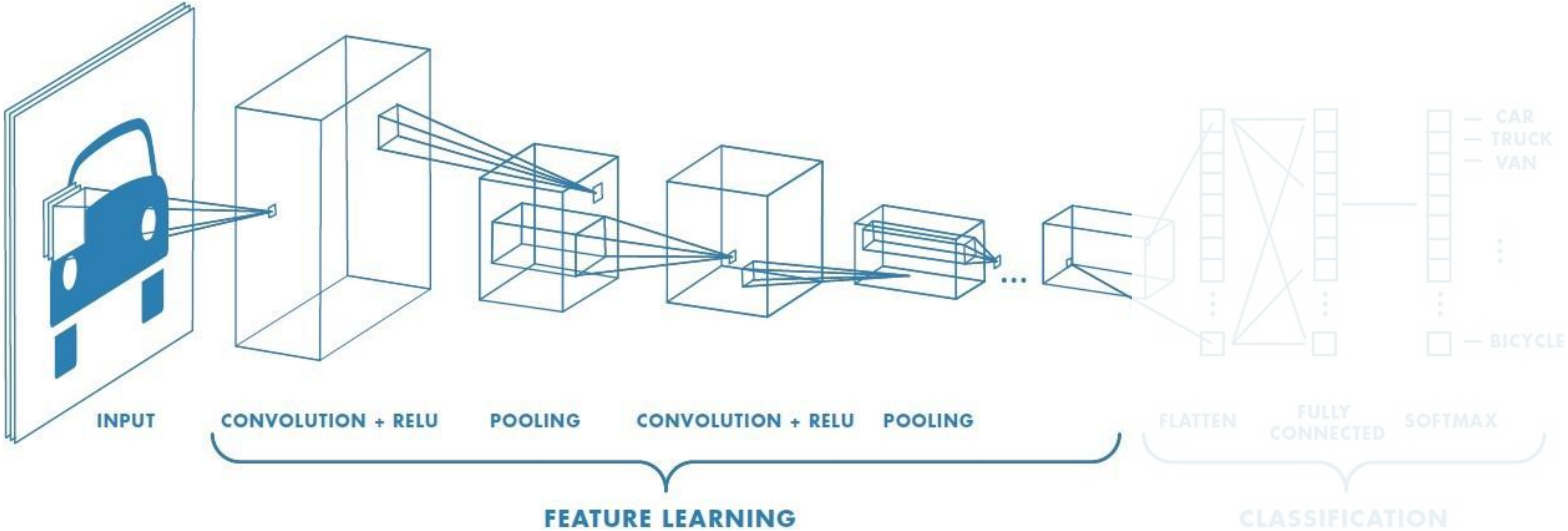
Pooling operations



Why?

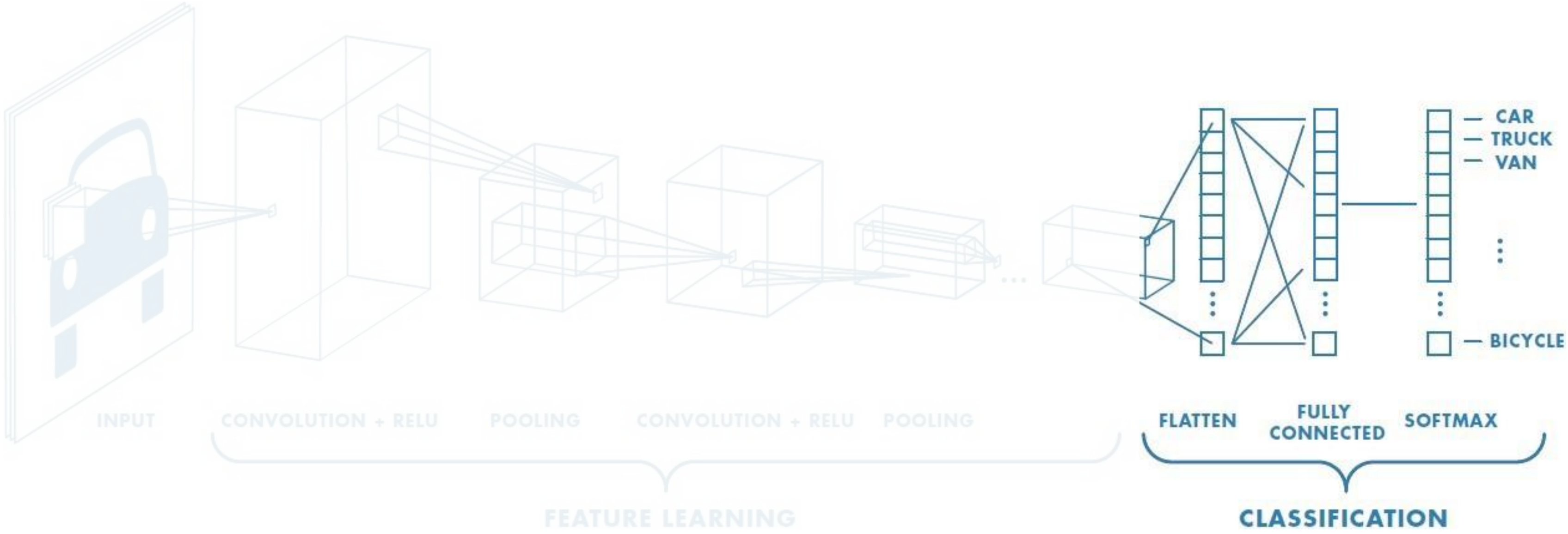
1. Reduce dimensionality
2. Preserve spatial invariance

CNN as a whole



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** (real world data isn’t linear!)
3. Reduce dimensionality and preseve
4. Preserve spatial invariance using **pooling**

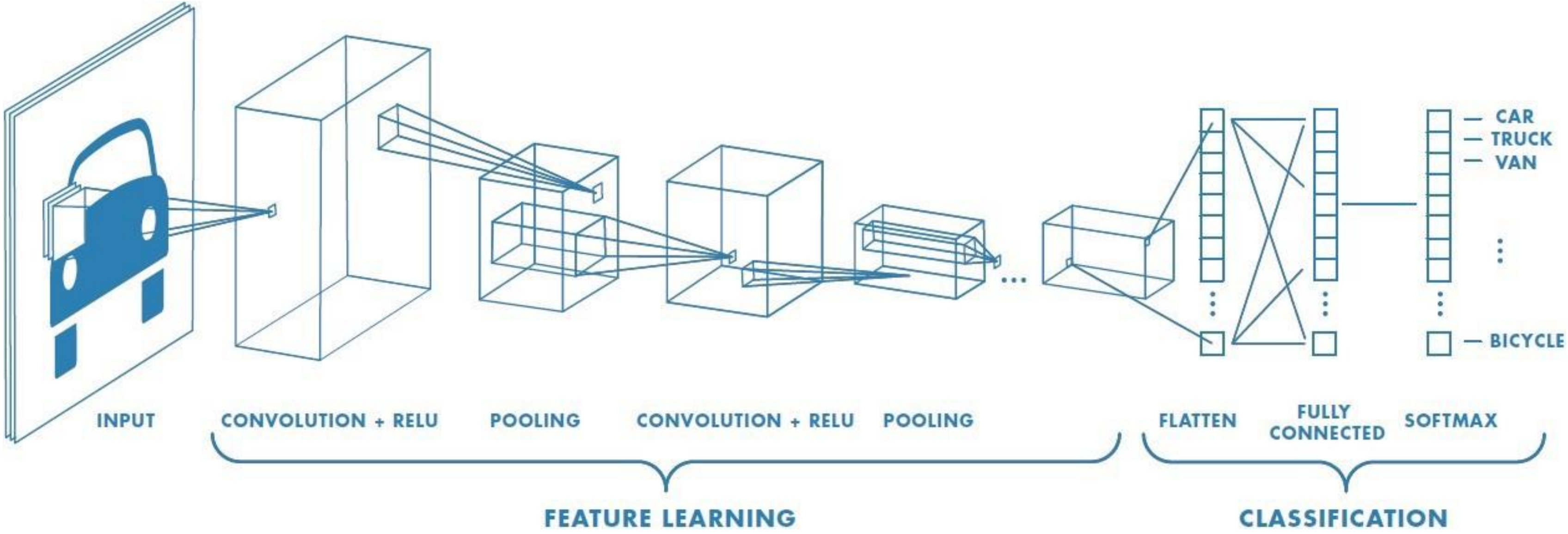
CNN as a whole



1. Outputs from feature learning are input into fully connected ANN
2. Fully connected layers users the generated features for classifying the input image
3. Express output as a probability of image belonging to a class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

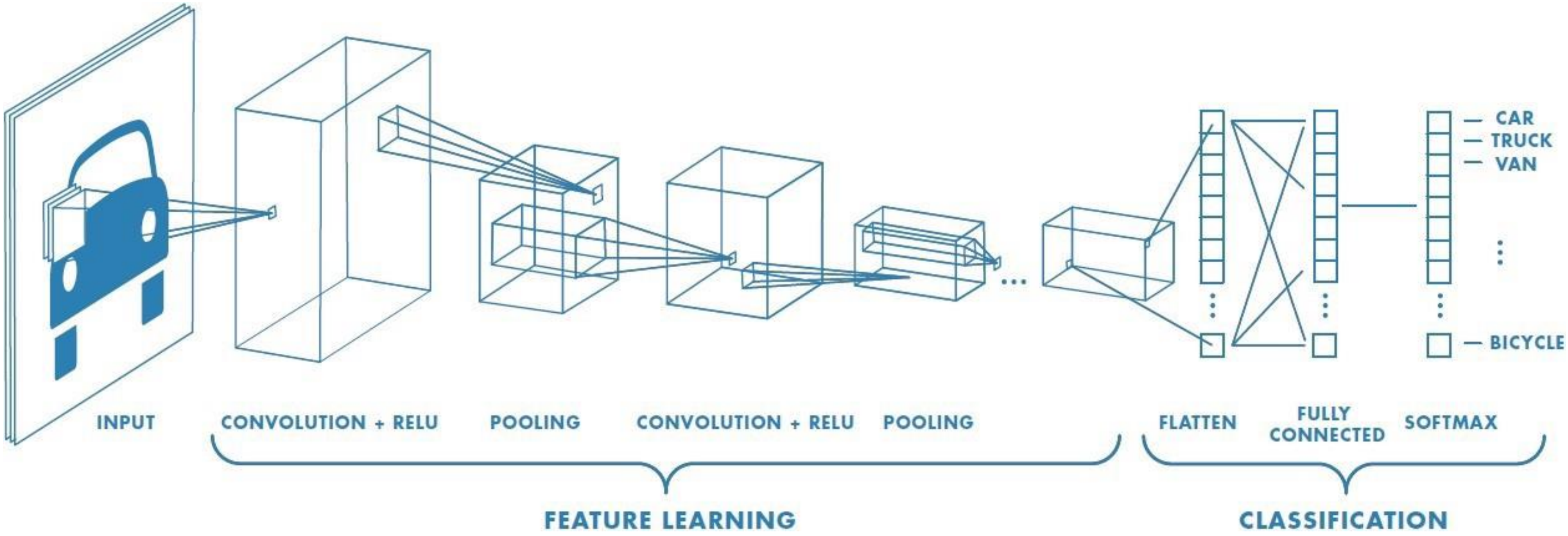
CNN as a whole



How do we train? – Exactly the same as the ANN. Use back propagation!!

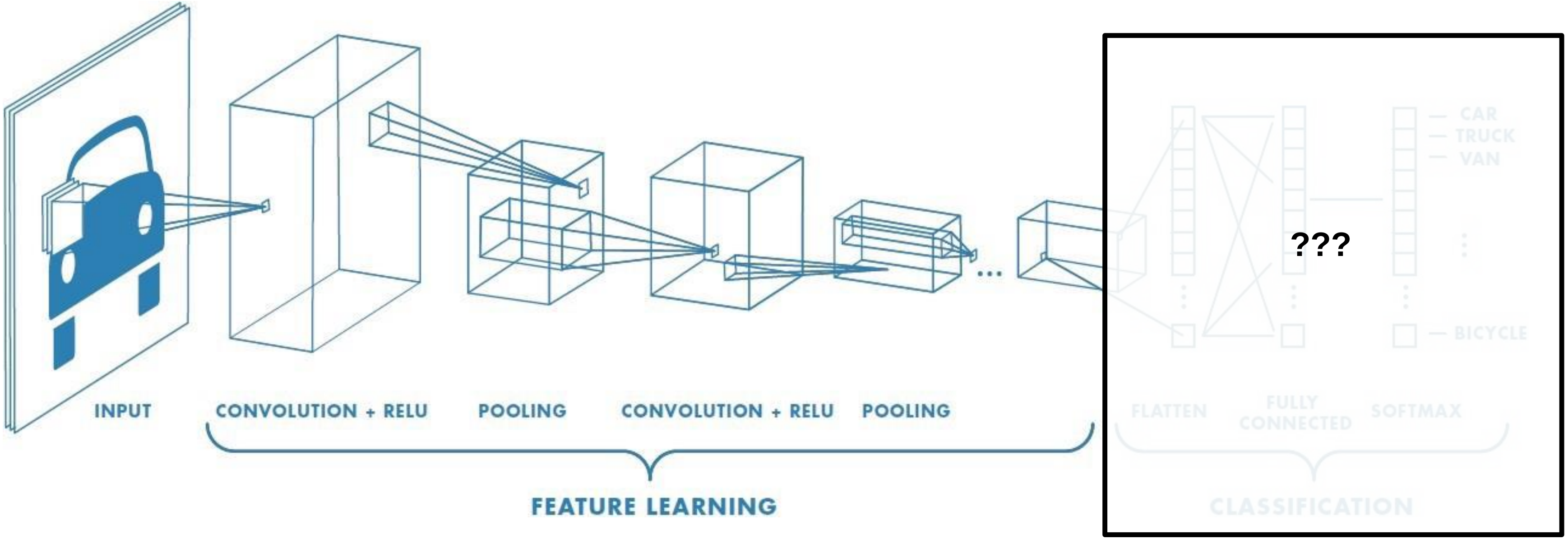
Use some loss function – cross entropy loss. $J(\boldsymbol{\theta}) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$

CNN as a whole



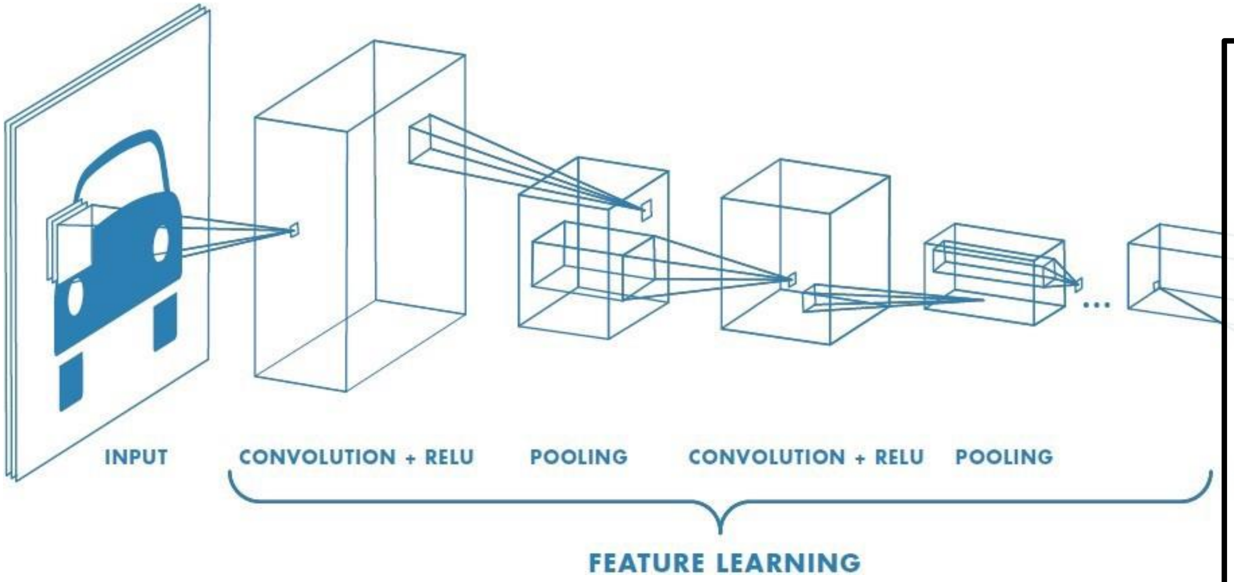
This full architecture is purposed for **image classification**!

An Architecture for many applications

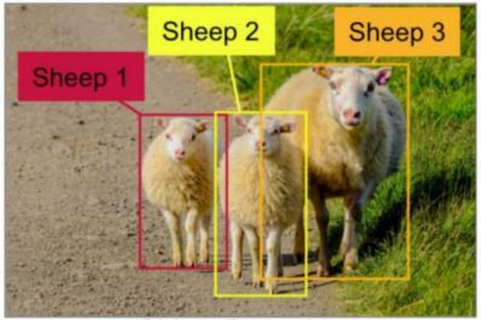


The convolution and pooling stages (*feature learning*) can be used for many other applications!

An Architecture for many applications

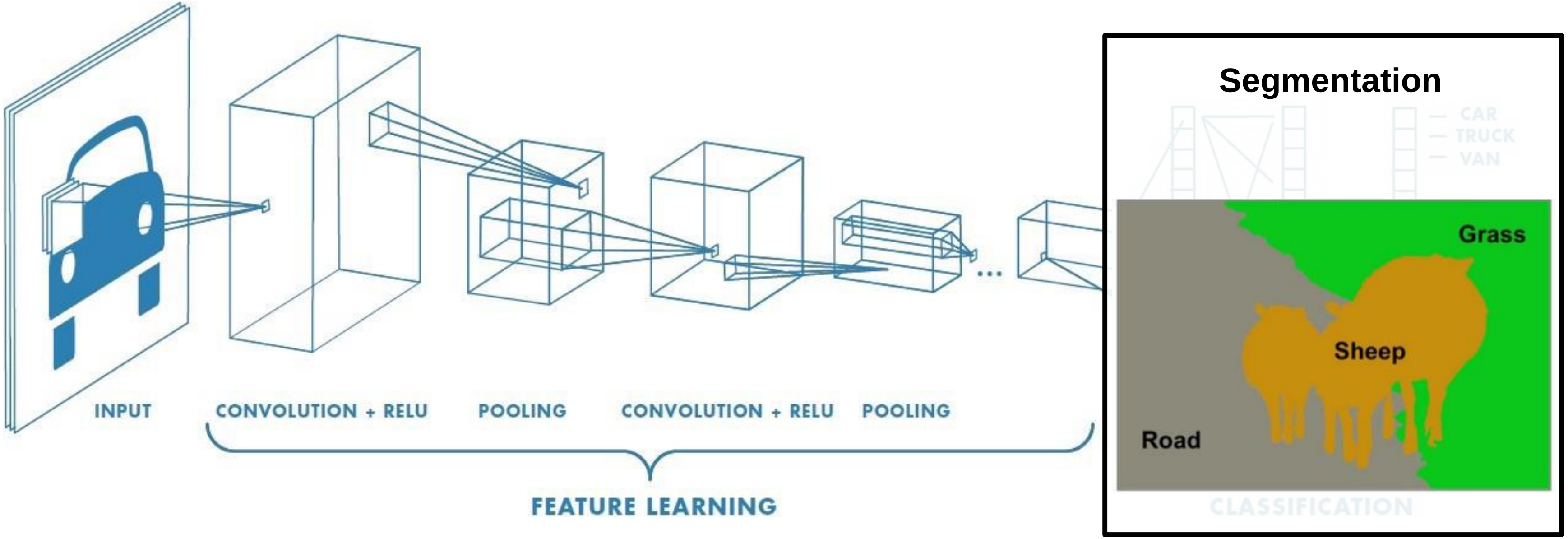


Object detection with R-CNNs



The convolution and pooling stages (*feature learning*) can be used for many other applications!

An Architecture for many applications



The convolution and pooling stages (*feature learning*) can be used for many other applications!

An Architecture for many applications

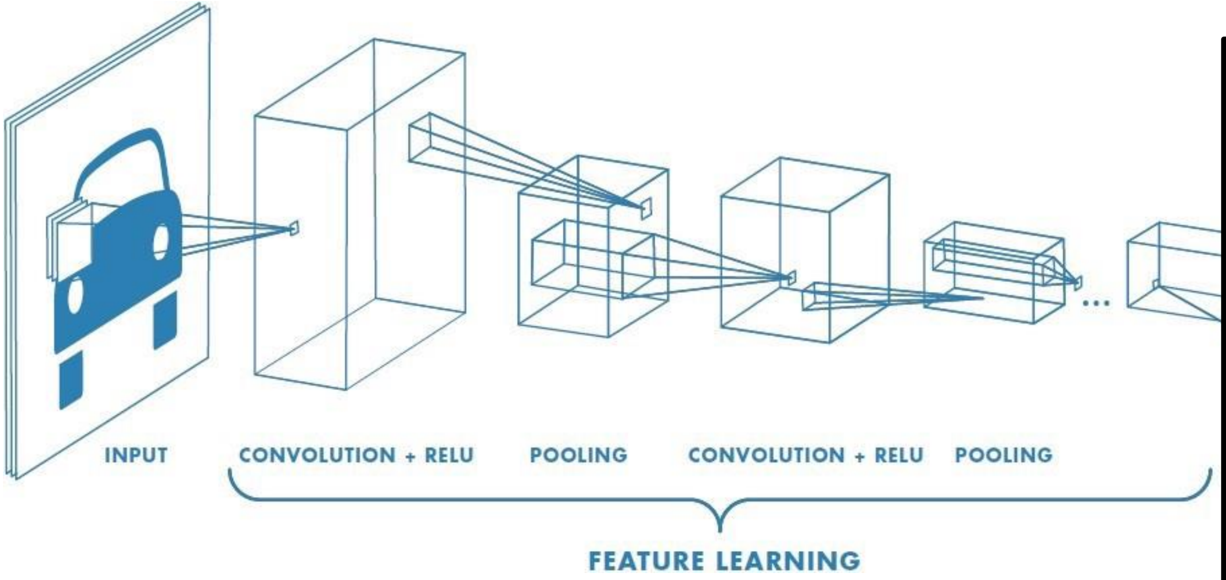


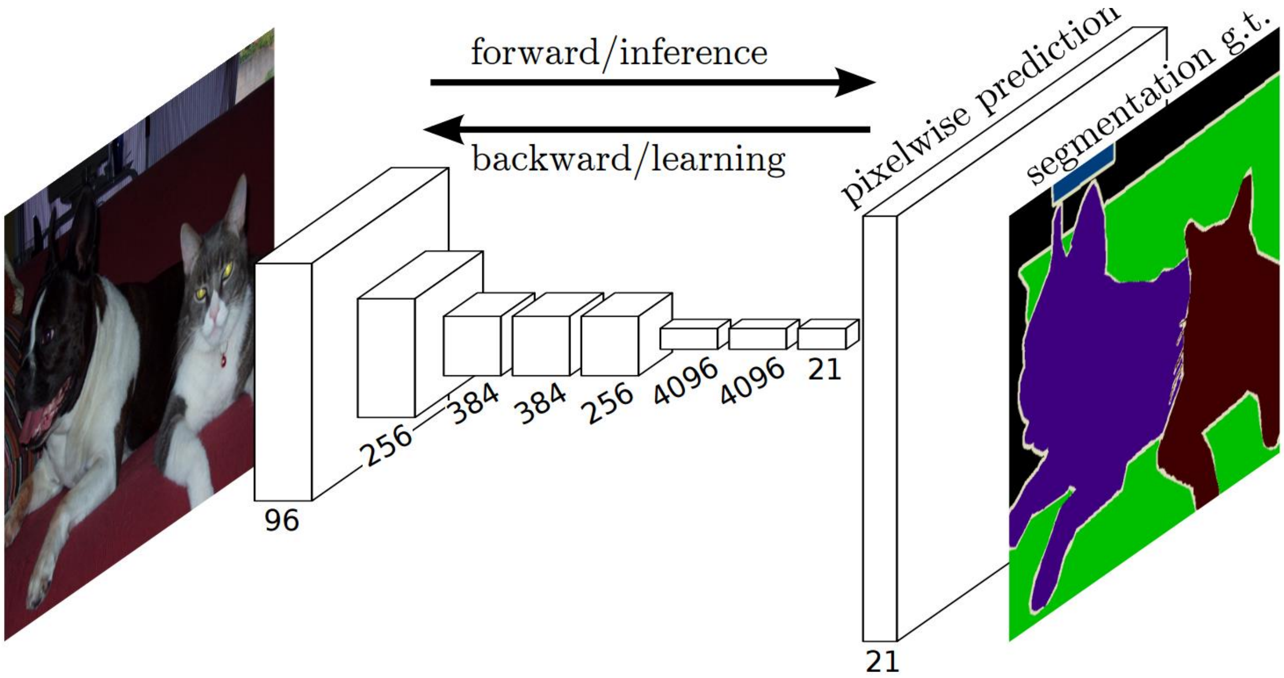
Image captioning with RNNs

The convolution and pooling stages (*feature learning*) can be used for many other applications!

Semantic Segmentation

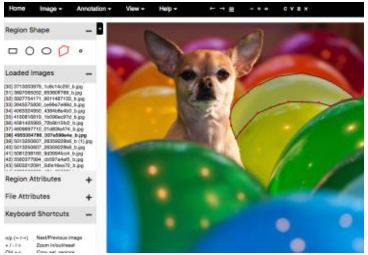
Assign each pixel in the image a class!

This has the flavour of an autoencoder



Creating your labelled datasets can be time consuming….

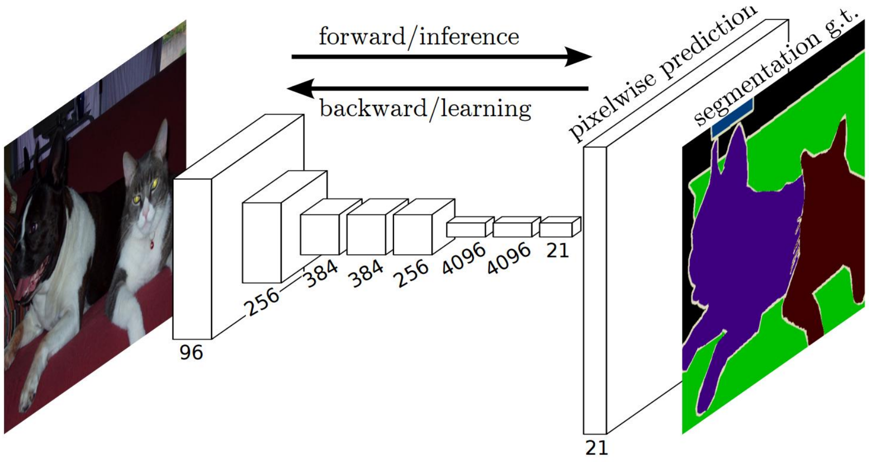
Annotate images manually…



Semantic Segmentation

Assign each pixel in the image a class!

This has the flavour of an autoencoder

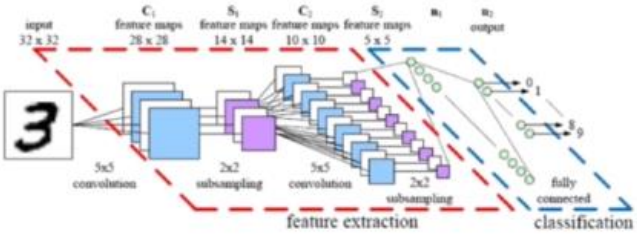


We used **down sampling** (convolutions and max pooling) to capture semantic/contextual information

We then implement **up sampling** to recover spatial information! Take our learned features and map them back into the original spatial image.

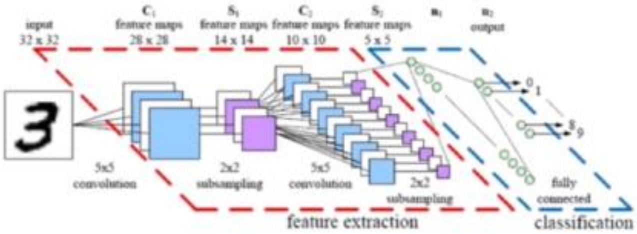
Object Detection and Instance Segmentation

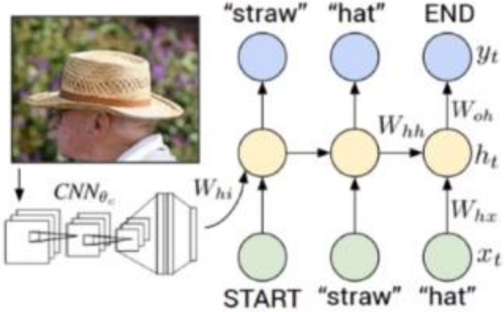
Identify objects and the pixels that relate to that object!



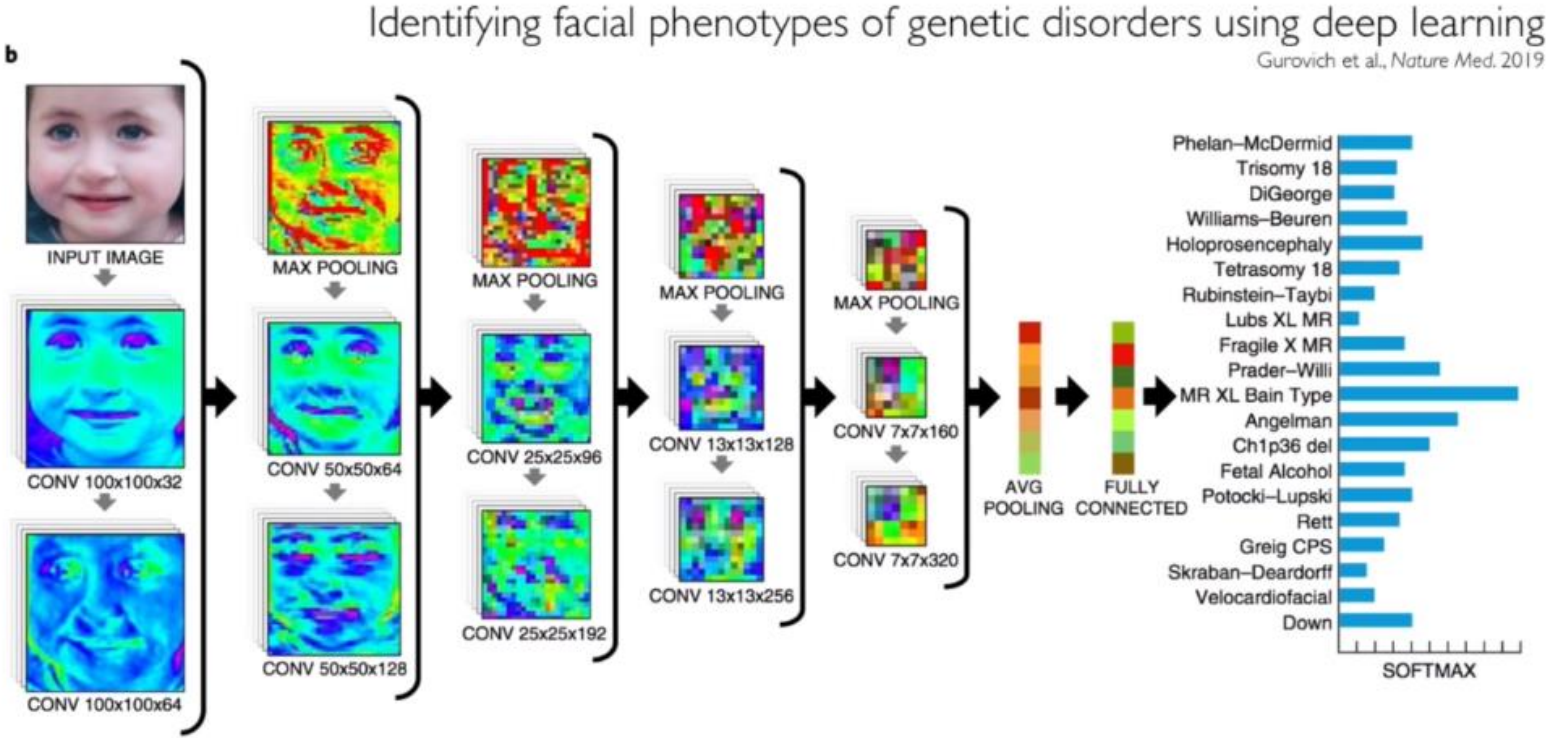
Object Detection and Instance Segmentation

Identify objects and the pixels that relate to that object!





Application of CNN



Quick Review

Foundations

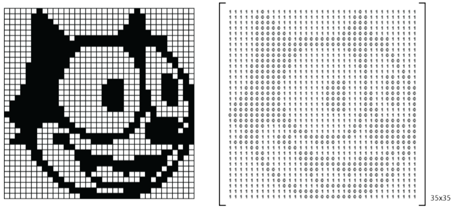
- Representing images for computer vision
- Convolution feature learning and pooling operations

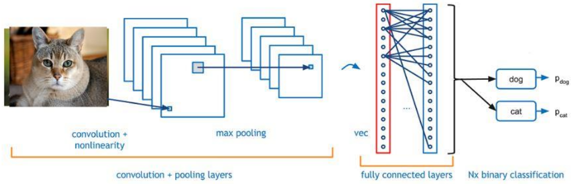
CNNs

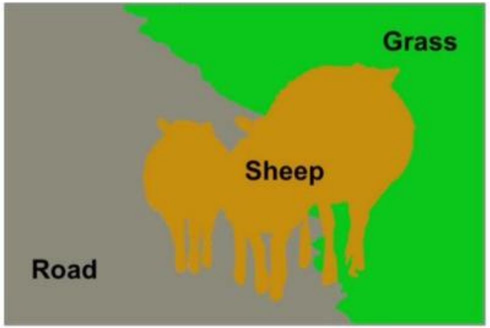
- The complete architecture
- Stacking multiple kernels into 3-dimensions

Applications

- Segmentation
- Object detection
- Image captioning



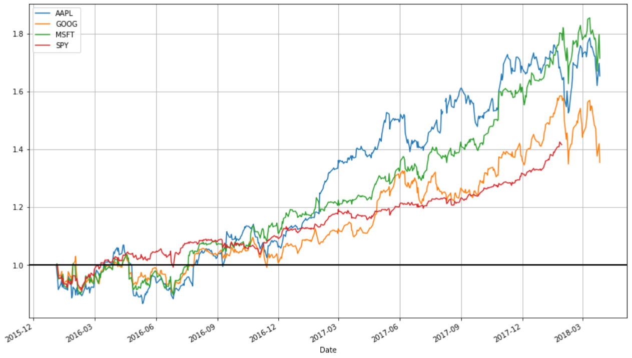




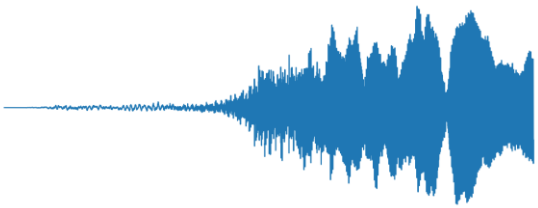
Recurrent Neural Networks

We have considered images… what about time-series data?

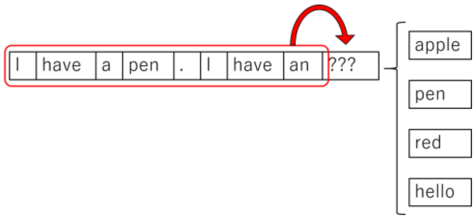
Stock Data



Audio



Text



Predicting next step in a trajectory

I took my dog for a …..

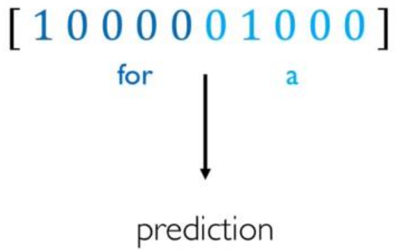
Predicting next step in a trajectory

I took my dog for a walk

Idea 1

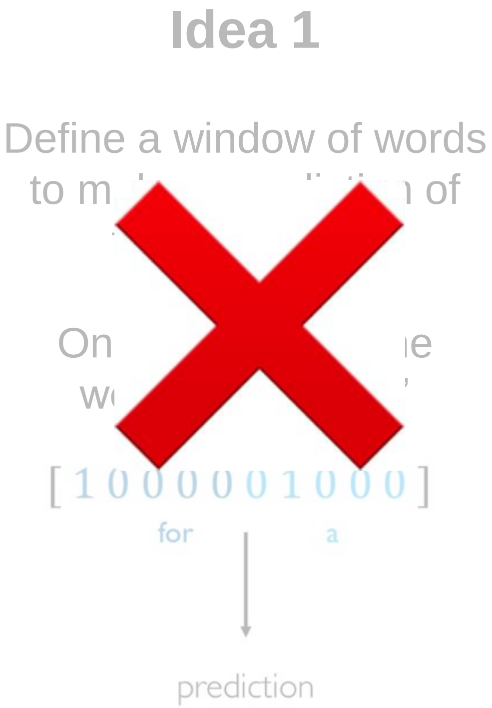
Define a window of words to make a prediction of the next word.

One-hot encode the words ‘for’ and ‘a’



Predicting next step in a trajectory

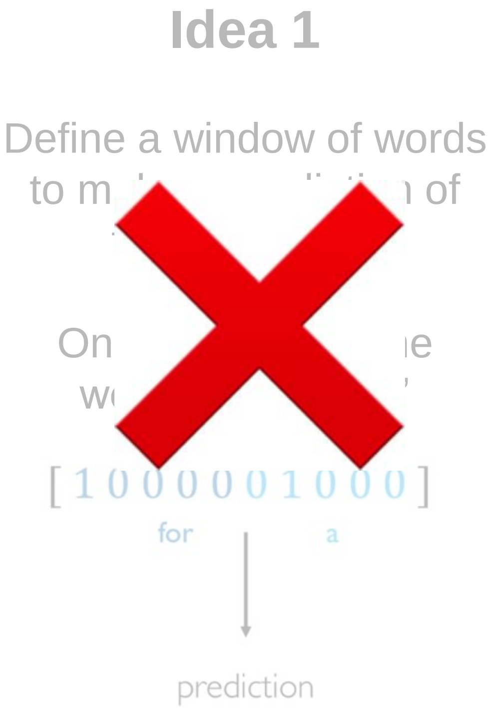
I took my dog for a walk



‘for’ and ‘a’ aren’t particularly predictive of the next word ‘walk’. We would need to define a larger window

Predicting next step in a trajectory

I took my dog for a walk



Can’t model long term dependencies

“China is where I grew up, but I now live in London. I speak fluent Chinese.”

We need to retain information from the distant past to predict the next correct word.

Predicting next step in a trajectory

I took my dog for a walk

Idea 1

Define a window of words
to m ... of

On ... e
w ...

[1 0 0 0 0 0 1 0 0 0]
for a

prediction

Idea 2

Use entire sequence as
set of counts

'bag of words'

[0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1]

prediction

Predicting next step in a trajectory

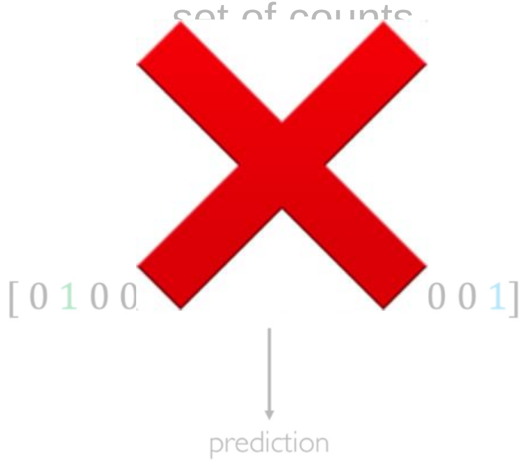
I took my dog for a walk

Idea 1

Define a window of words

Idea 2

Use entire sequence as



It lost the sequential information!

e.g.

The food was good, not bad at all.

Vs

The food was bad, not good at all.

Predicting next step in a trajectory

I took my dog for a walk

Idea 1

Define a window of words

[1 0 0 0 0 0 1 0 0 0]

for a

prediction

Idea 2

Use entire sequence as

[0 1 0 0 … 0 0 1]

prediction

Idea 3

Extend the window for

[1 0 0 0 0 0 0 … 1 0 …]

Introduces separate parameters for words at different position in sequence.

Features we learn about the sequence won't transfer if they appear elsewhere in the sequence.

Predicting next step in a trajectory

Traditional feed-forward neural network isn’t satisfactory for this problem.

We saw that standard ANN architecture wasn’t suitable for images

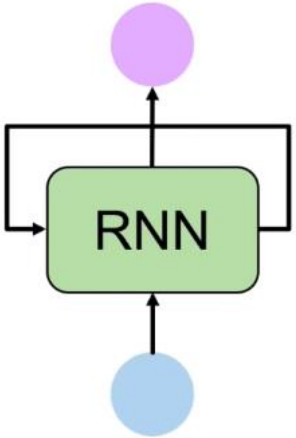
Similar problem for time-series!

Design Criteria

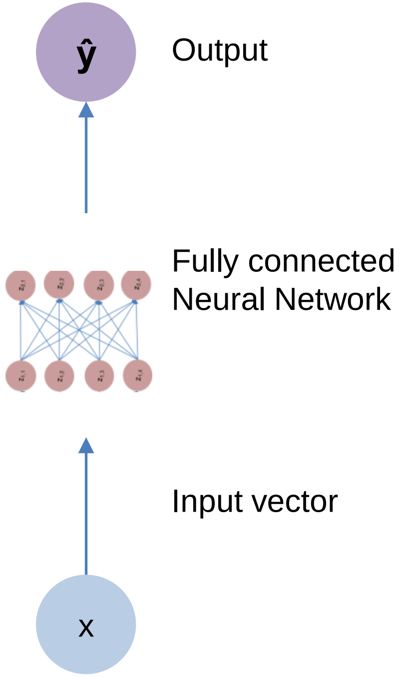
We want a model that can:

1. Take **variable length** sequences
2. Track (‘remember’) **long term dependencies**
3. Maintain information about **order**
4. **Share parameters** across the sequence

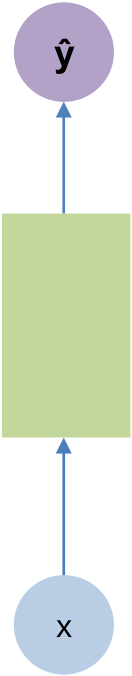
Use a Recurrent Neural Network (RNN).



Sequence modelling

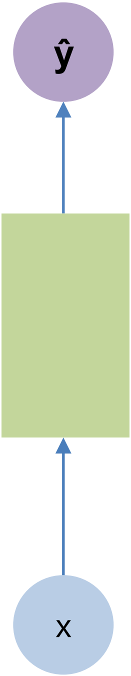


Sequence modelling

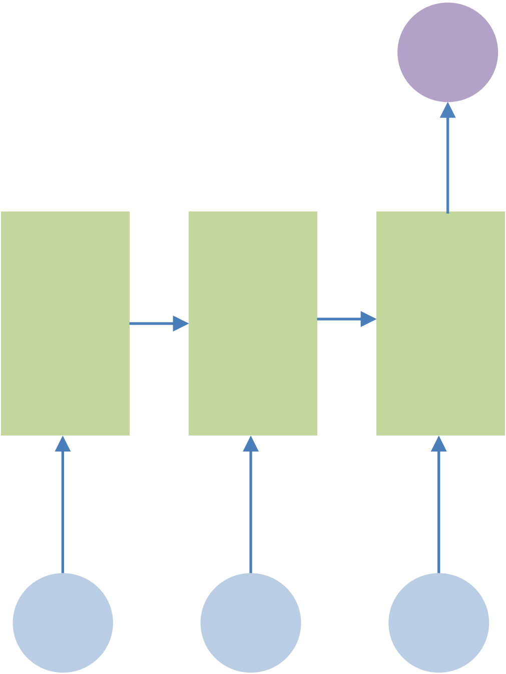


Our basic
Neural Network

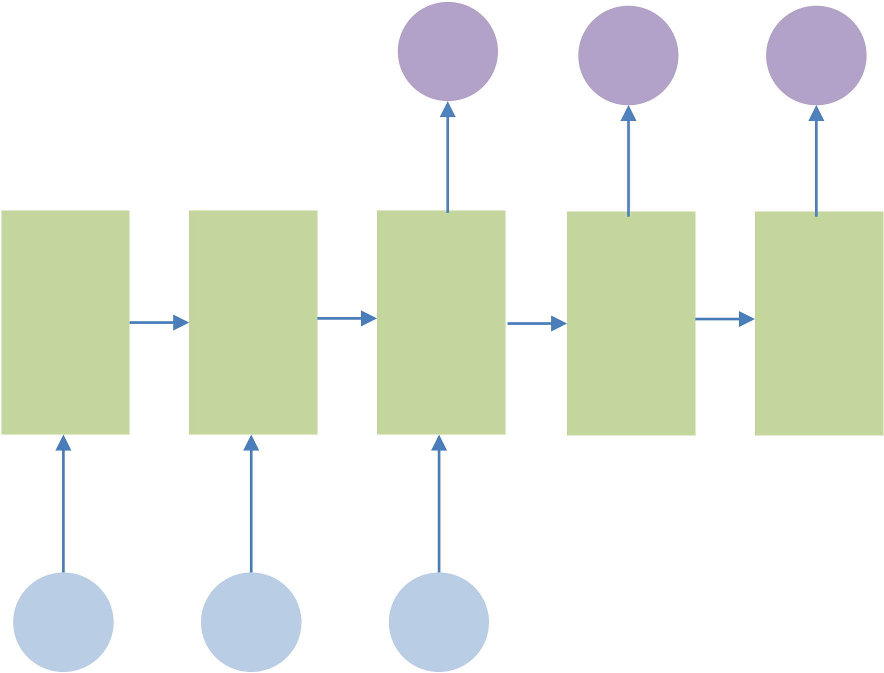
Sequence modelling



Our basic
Neural Network

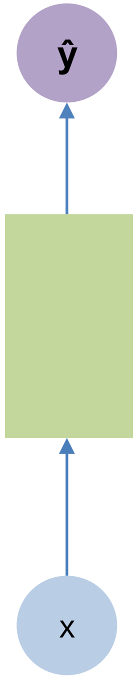


Many to one
e.g. Sentiment Analysis



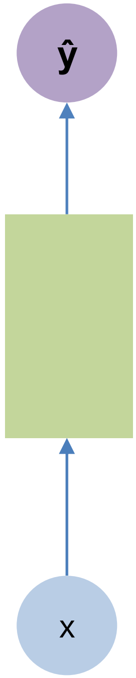
Many to many
e.g. predict stock prices

Recurrent Neural Network



Our basic
Neural Network

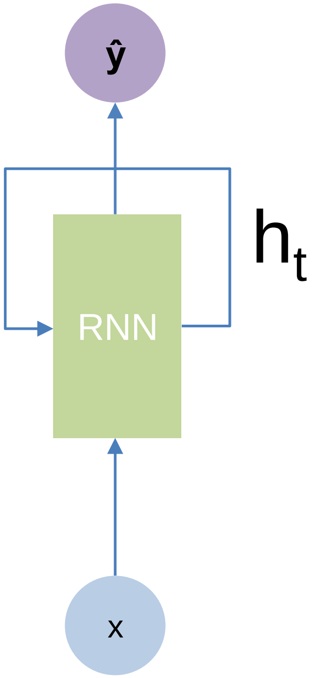
Recurrent Neural Network



Our basic
Neural Network

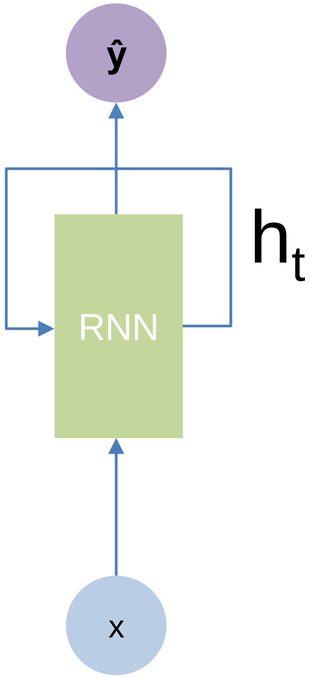
Passing information internally from one step in the network to the next.

This loop creates a recurrent relation.



Recurrent neural
network

Recurrent Neural Network

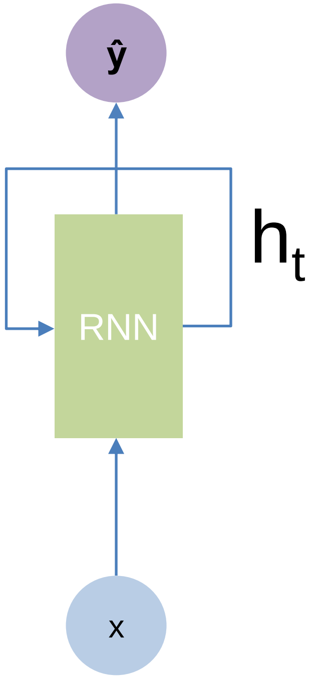


Recurrent neural network

$$\underbrace{\boxed{h_t}}_{\text{new state}} = \underbrace{\boxed{f_W}}_{\text{function parameterized by W}} (\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, \underbrace{\boxed{x_t}}_{\text{input vector at time step } t})$$

Some weighted function (learnt weights) that updates the state using the current state and the next step in the sequence.

Recurrent Neural Network

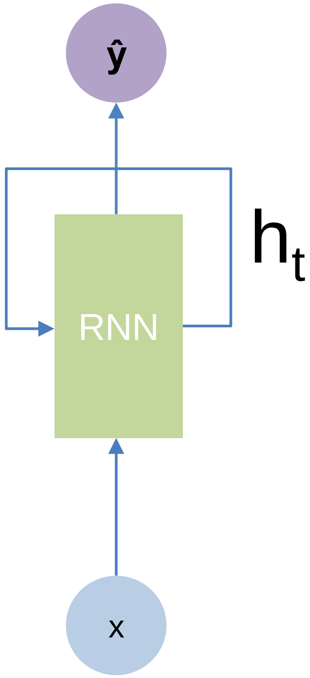


Recurrent neural network

Standard Neural network

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

Recurrent Neural Network



Recurrent neural network

Standard Neural network

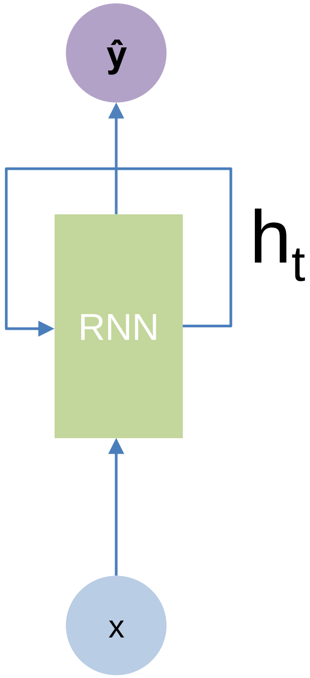
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

Recurrent Neural Network

Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh} h_{t-1} + \boldsymbol{W}_{xh} x_t)$$

Recurrent Neural Network



Recurrent neural network

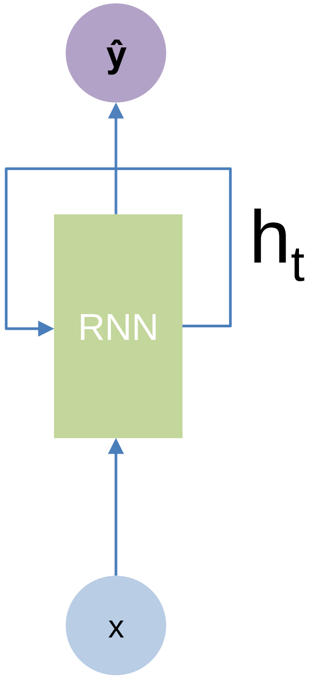
Recurrent Neural Network

Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}} x_t)$$

Two weight matrices:
1) Applied to hidden state
2) Applied to input

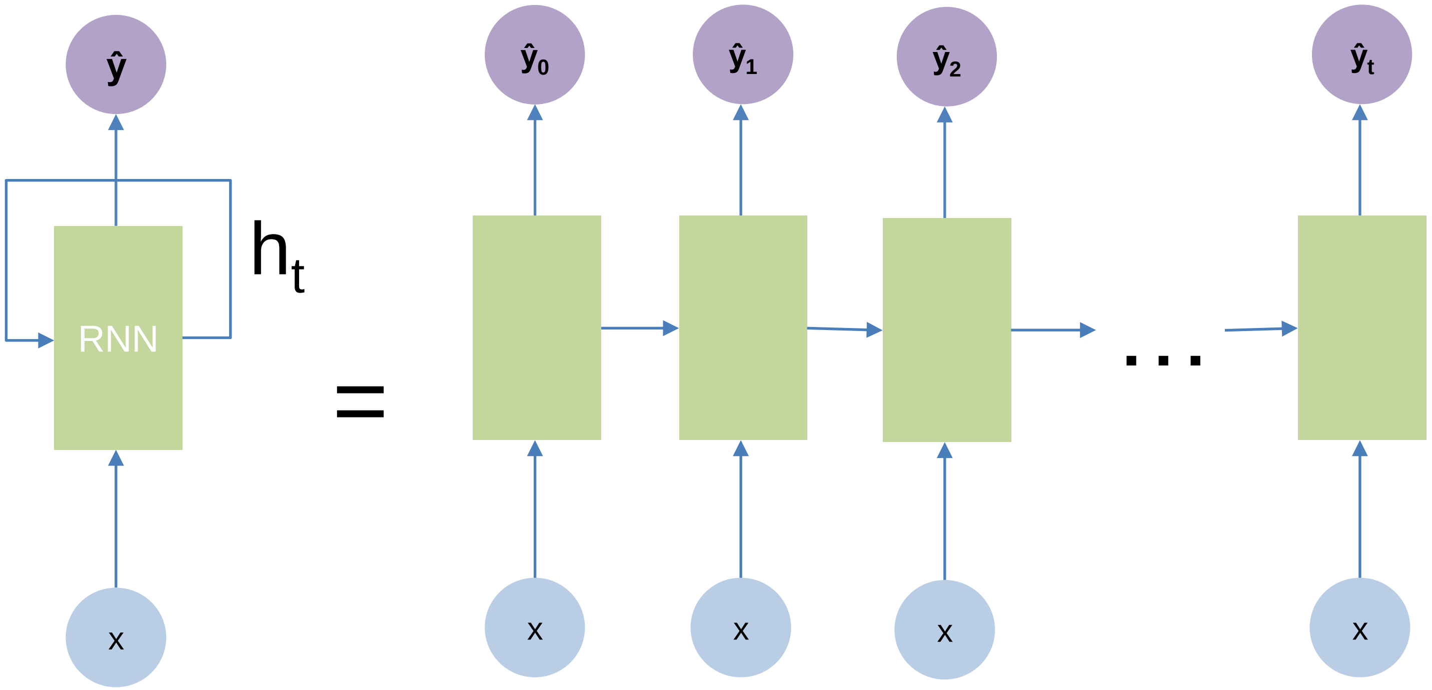
Unravelling RNN



=

Recurrent neural network

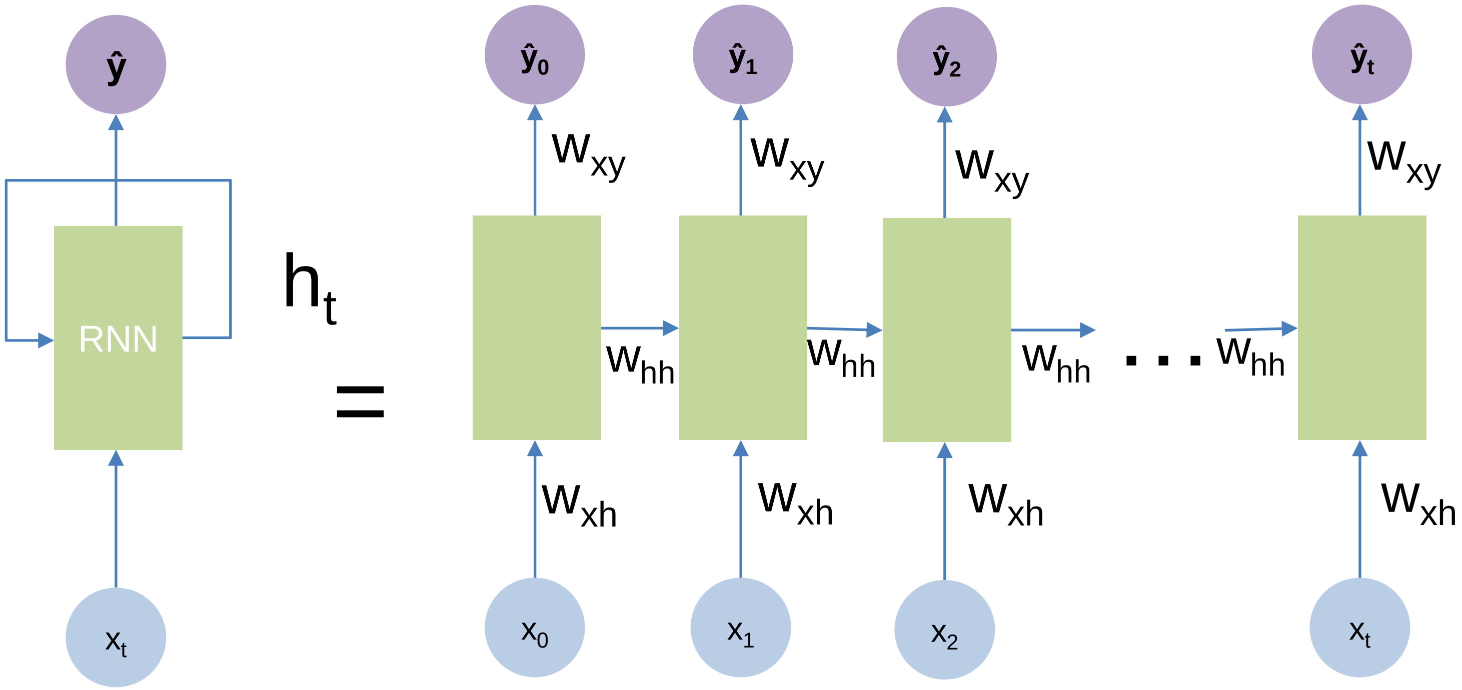
Unravelling RNN



Recurrent neural network

The Loop can be seen as a chain like structure

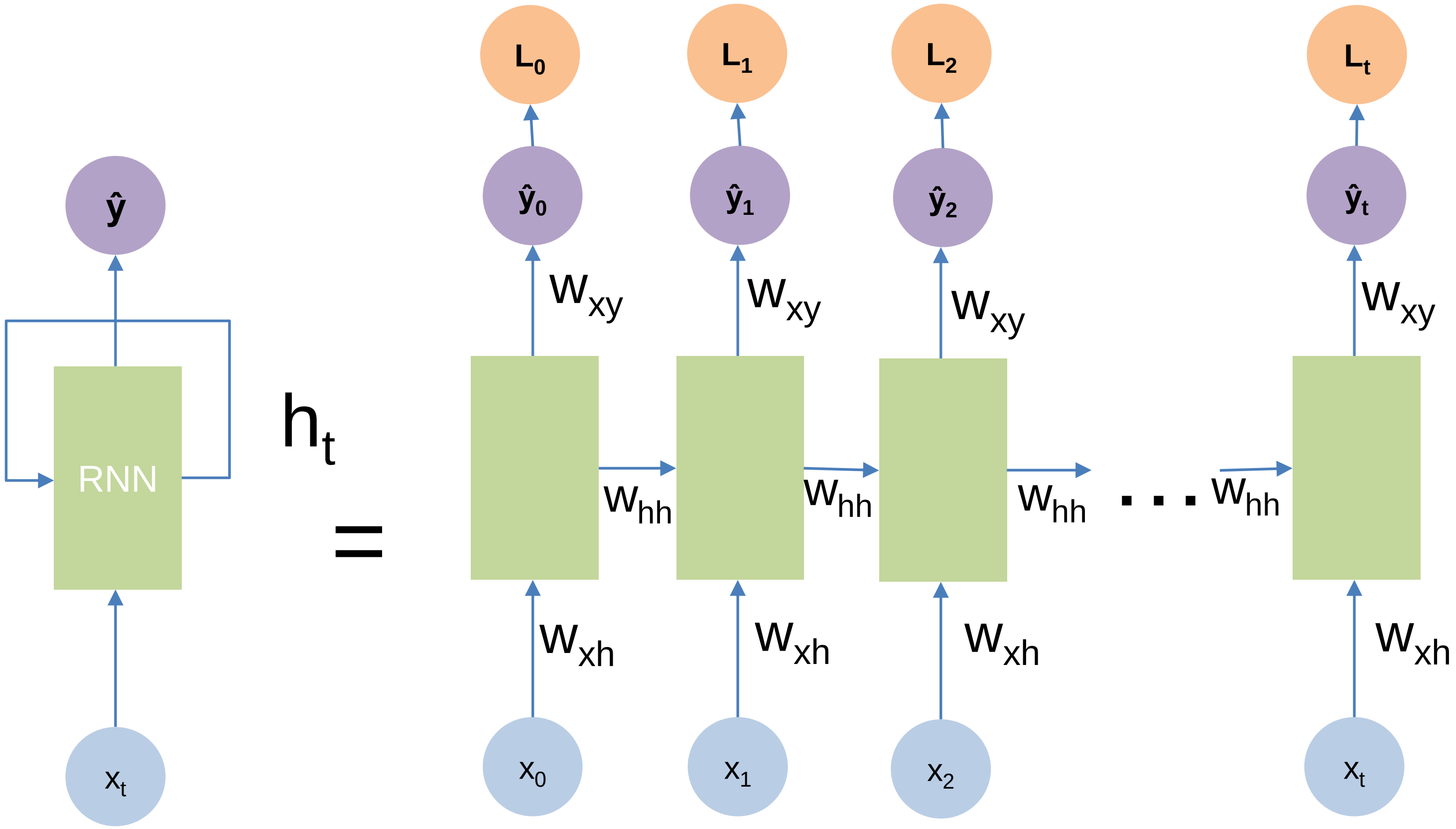
Unravelling RNN



Recurrent neural network

The Loop can be seen as a chain like structure

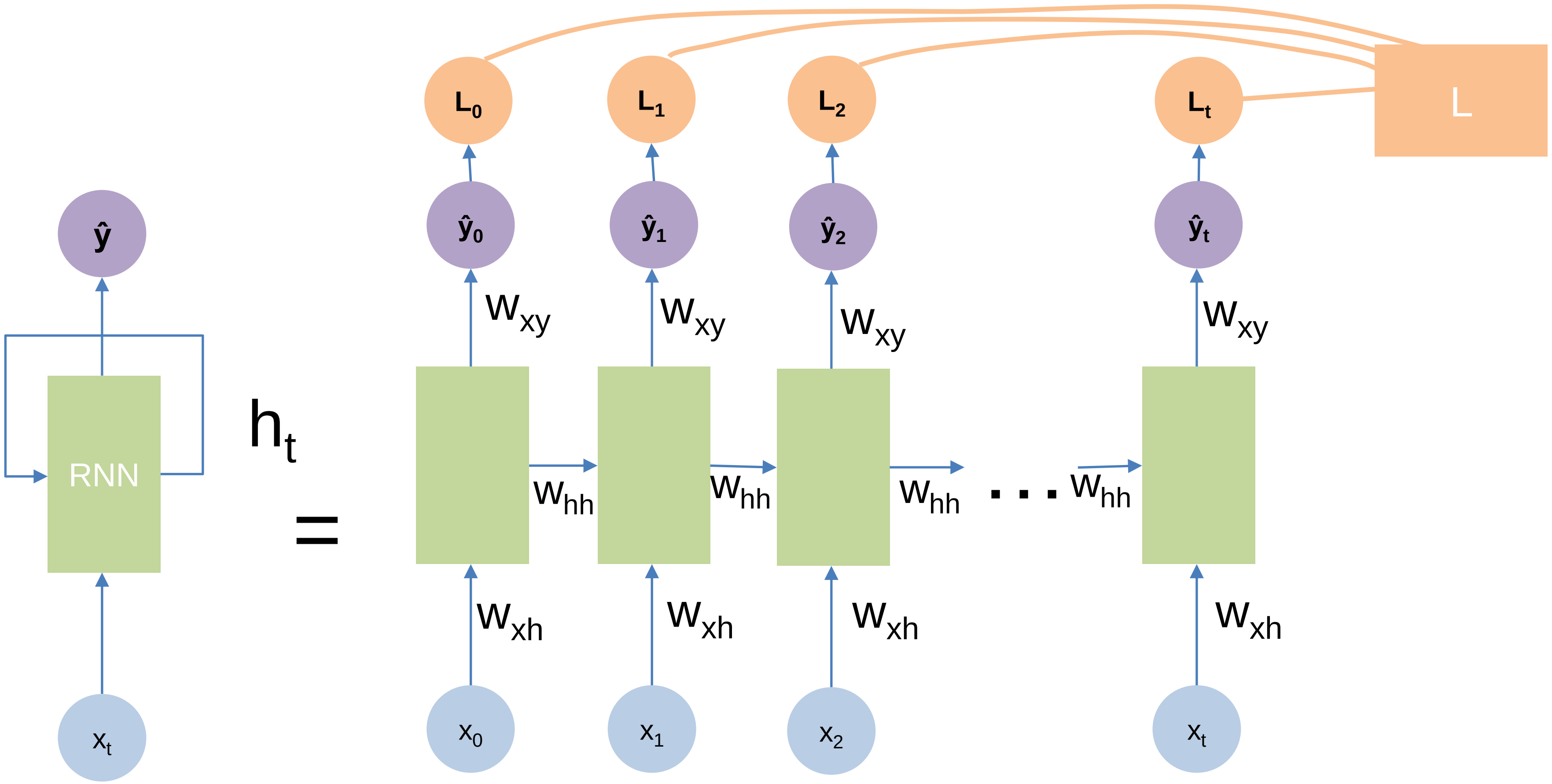
Using the same weight matrices through our network!



Recurrent neural network

The Loop can be seen as a chain like structure

Using the same weight matrices through our network!

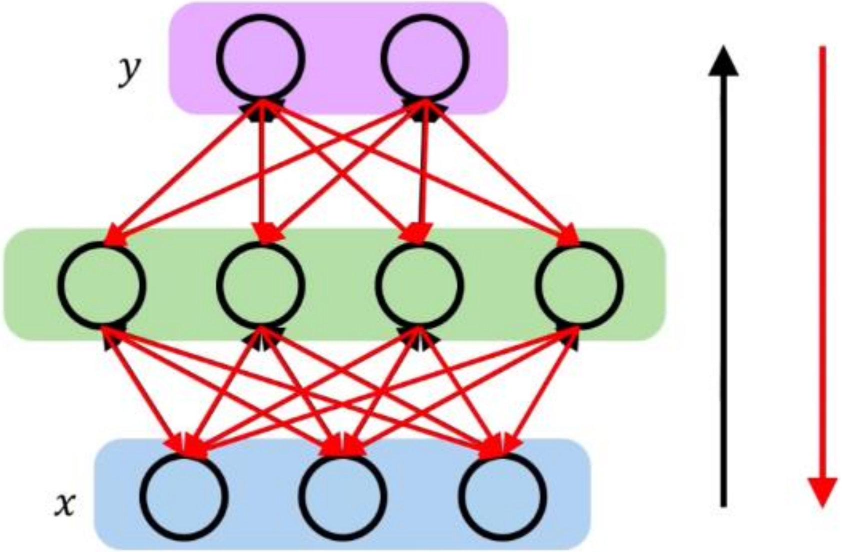


Recurrent neural network

Individual contributions to the loss across all steps in time!

Back propagation through time

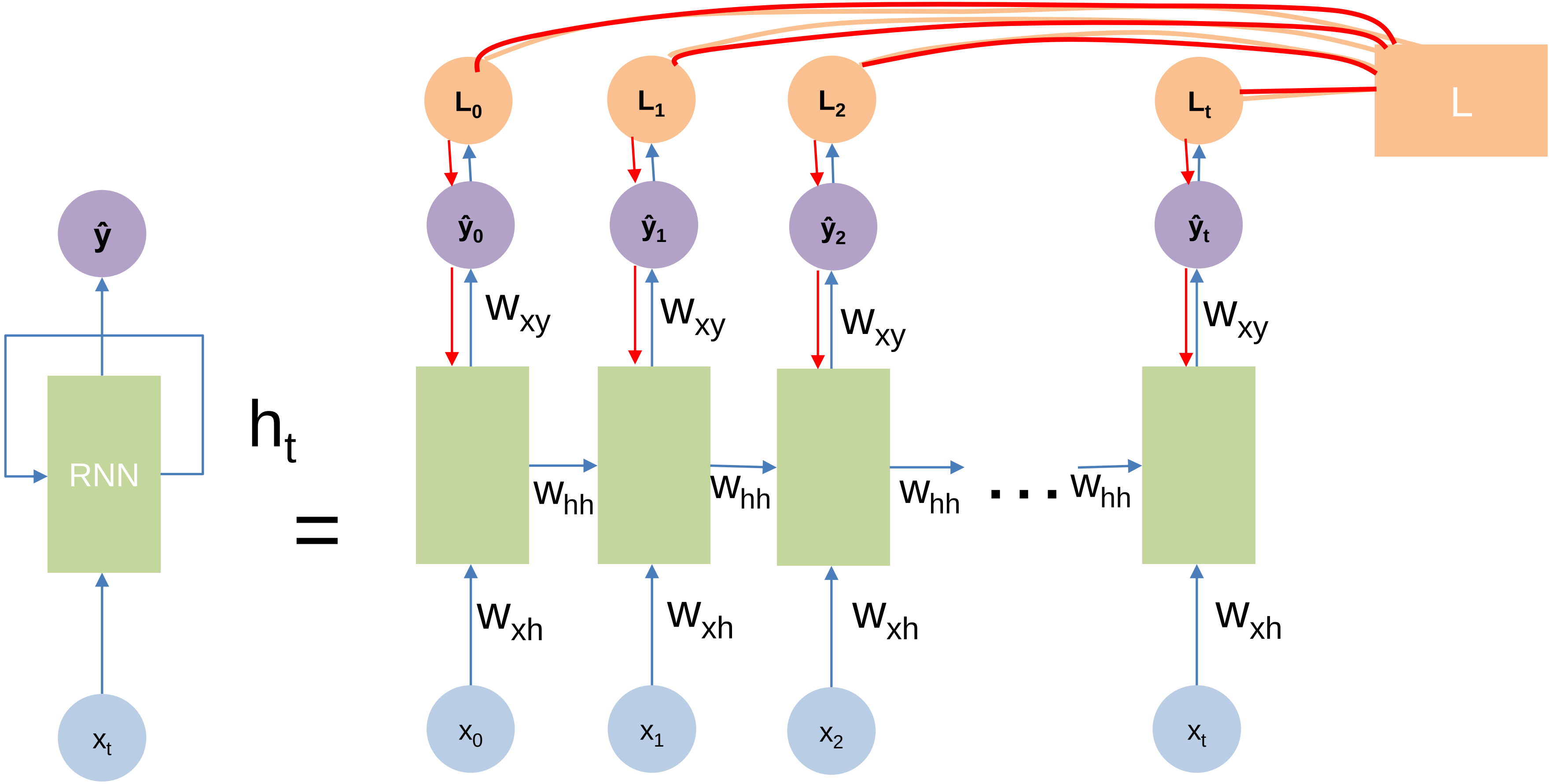
Recall the backpropagation in feed forward neural networks



Backpropagation algorithm:

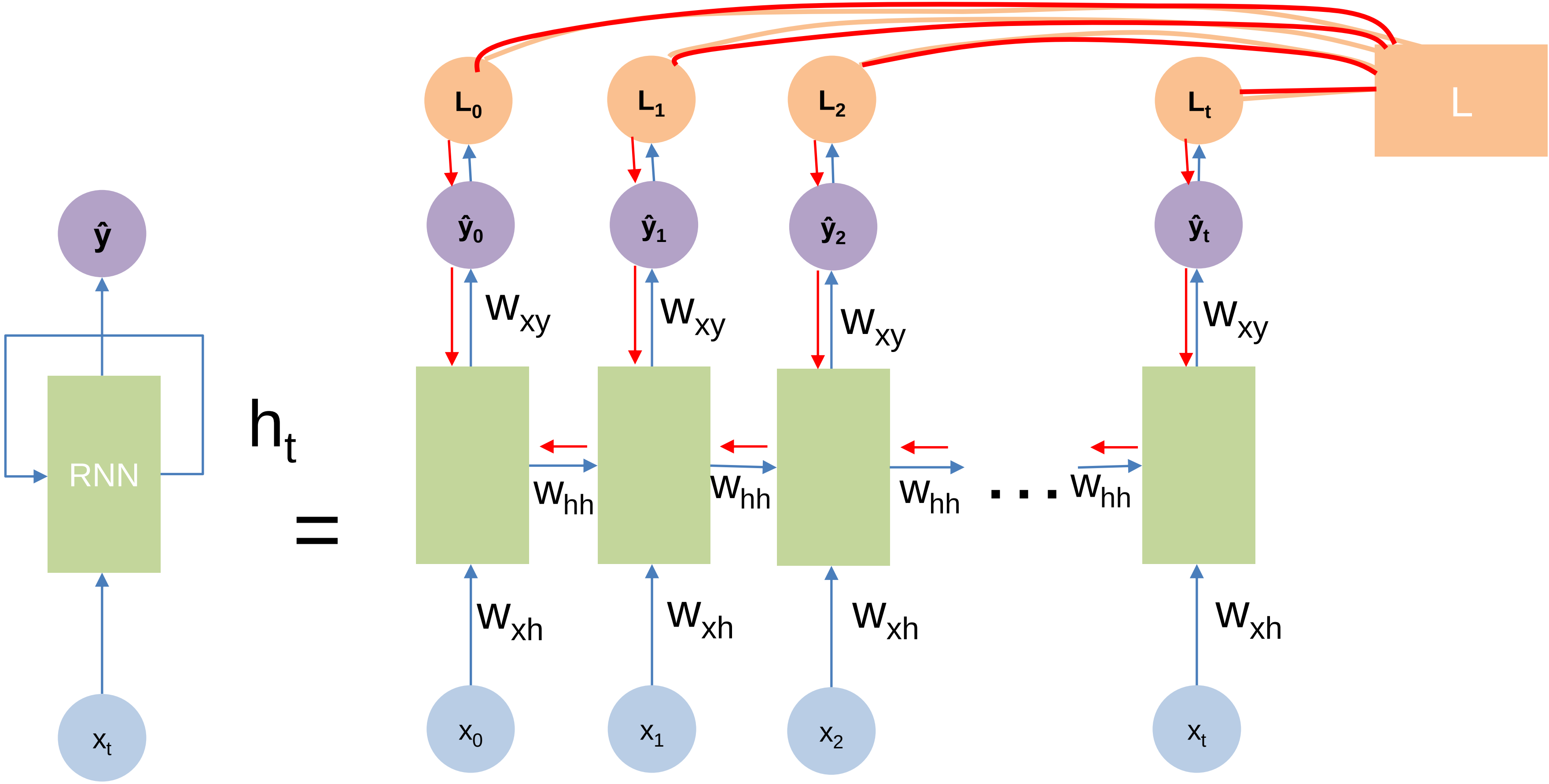
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$



Recurrent neural network

Individual contributions to the loss across all steps in time!



Recurrent neural network

Individual contributions to the loss across all steps in time!

We have a weight matrix W_{hh} that is repeated many times.

Calculating gradient wrt to h_o involves many factors of W_{hh}