Data Science: Principles and Practice

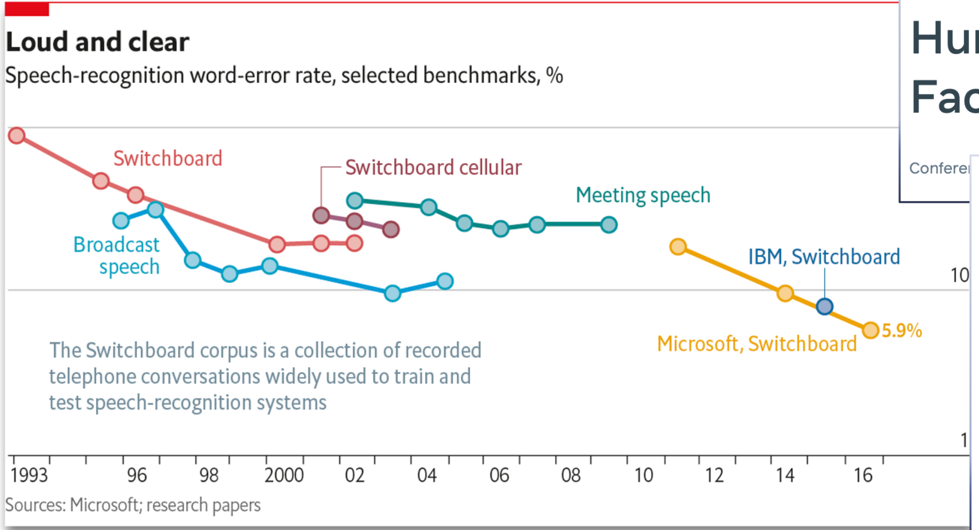
Lecture 7: Further DL Architectures

Ekaterina Kochmar

UNIVERSITY OF CAMBRIDGE

Perception

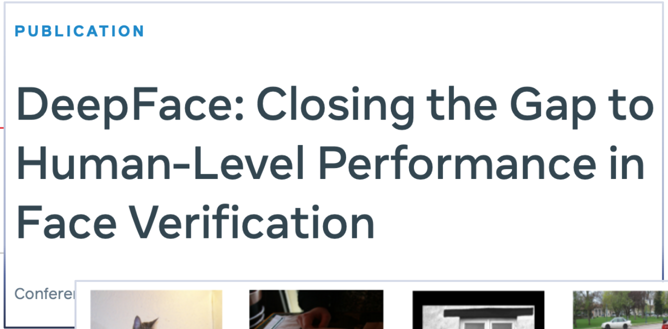
- So far we've been looking at simpler models
- What is so peculiar about recognising images and language?
- Look closely into sensory modules


Loud and clear

Speech-recognition word-error rate, selected benchmarks, %

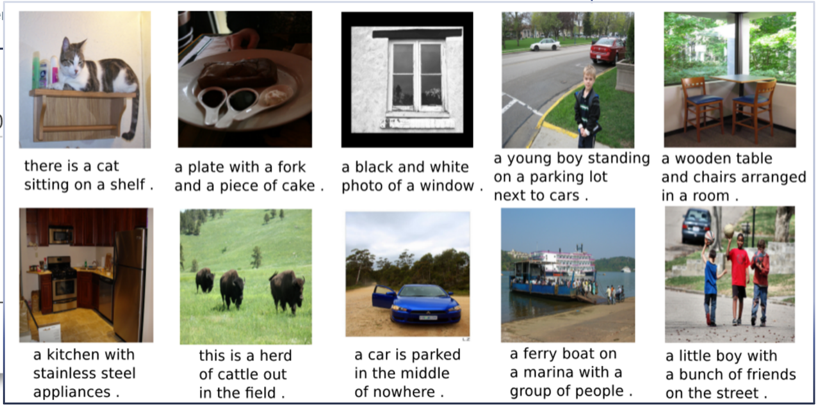
The Switchboard corpus is a collection of recorded telephone conversations widely used to train and test speech-recognition systems

Sources: Microsoft; research papers


PUBLICATION

DeepFace: Closing the Gap to Human-Level Performance in Face Verification

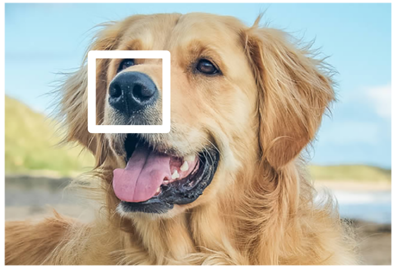
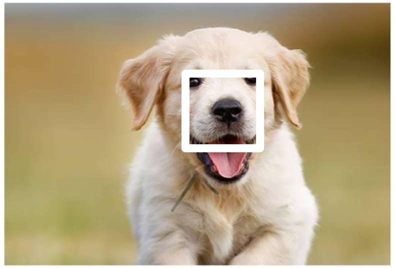
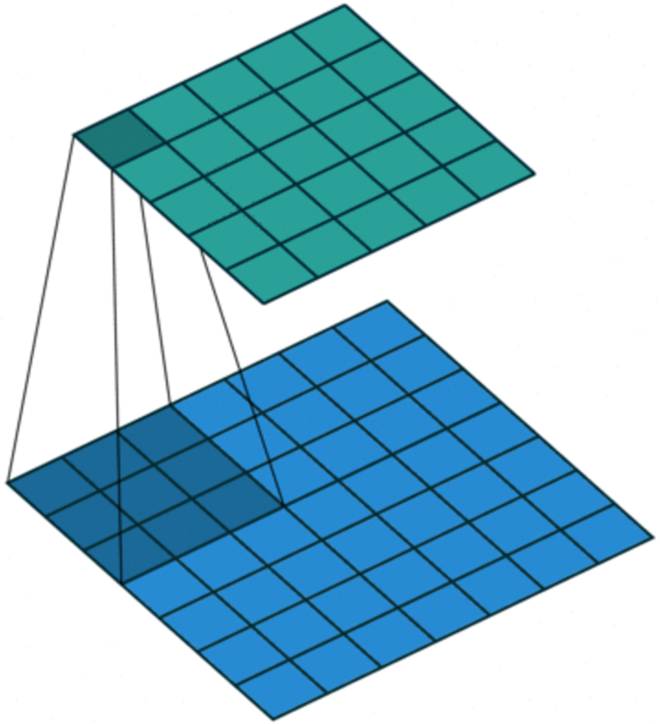
Confere



there is a cat sitting on a shelf .
a plate with a fork and a piece of cake .
a black and white photo of a window .
a young boy standing on a parking lot next to cars .
a wooden table and chairs arranged in a room .
a kitchen with stainless steel appliances .
this is a herd of cattle out in the field .
a car is parked in the middle of nowhere .
a ferry boat on a marina with a group of people .
a little boy with a bunch of friends on the street .

Convolutional Neural Networks

- Inspired by research on the brain's visual cortex
- Have been used in image recognition since 1980s
- Are used in image search, self-driving cars, video classification
- Are also used in other fields e.g., NLP, voice recognition

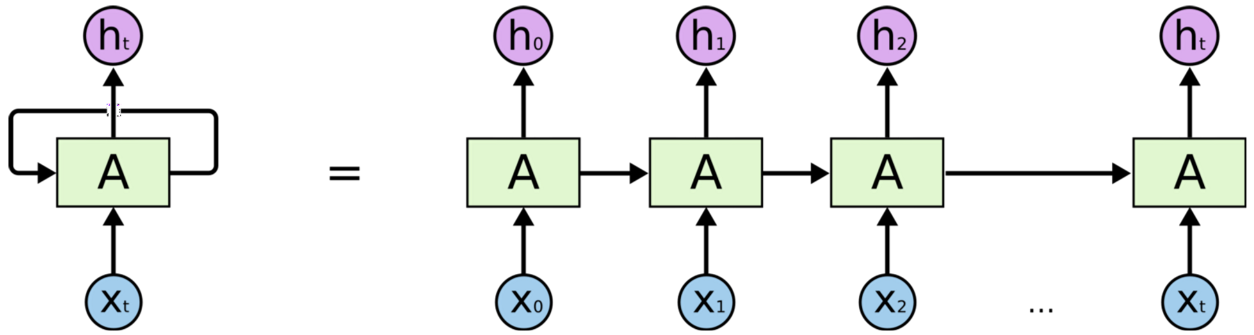




Recurrent Neural Networks

Designed to process **input sequences** of arbitrary length.

Each hidden state is calculated based on the **current input** and the **previous hidden state**.

Main neural architecture for **processing text**, with each input being a word representation.



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

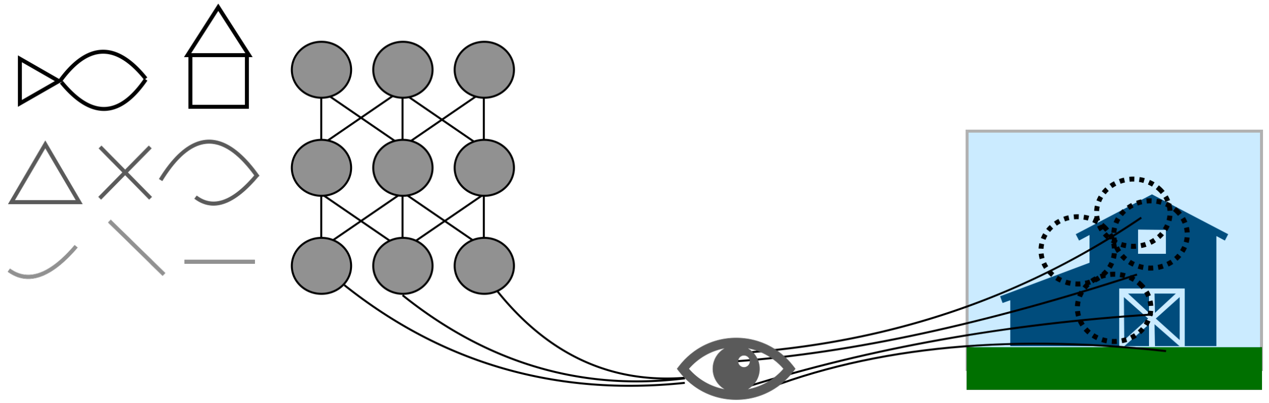
Data Science: Principles and Practice

Convolutional Neural Networks

From Visual Cortex

Experiments on cats in 1958-59: many neurons in the visual cortex have a small **local receptive field**

Receptive fields of different neurons **may overlap**

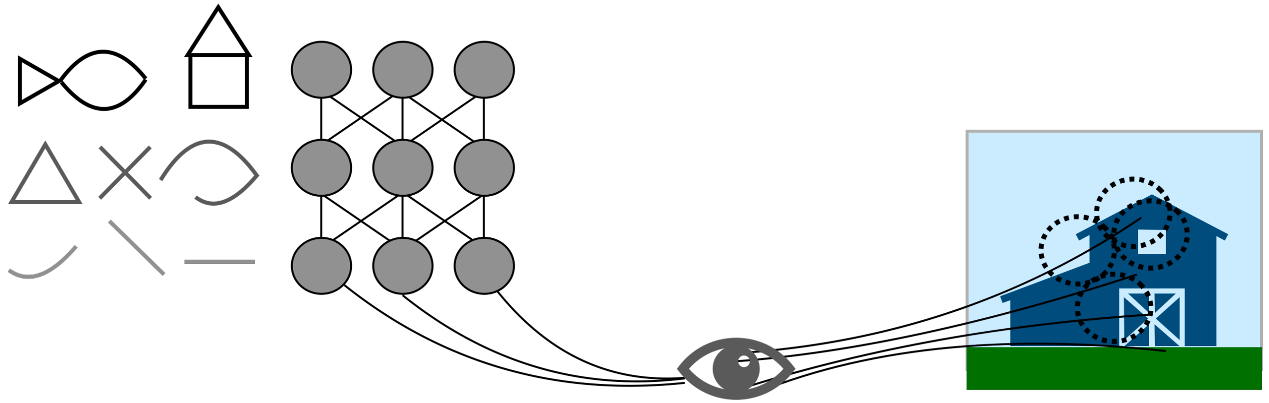


Hubel and Wiesel (1958). *Single Unit Activity in Striate Cortex of Unrestrained Cats*
Hubel and Wiesel (1959). *Receptive Fields of Single Neurones in the Cat's Striate Cortex*

From Visual Cortex

Some neurons react only to images of horizontal lines, others – only to lines with **different orientations**

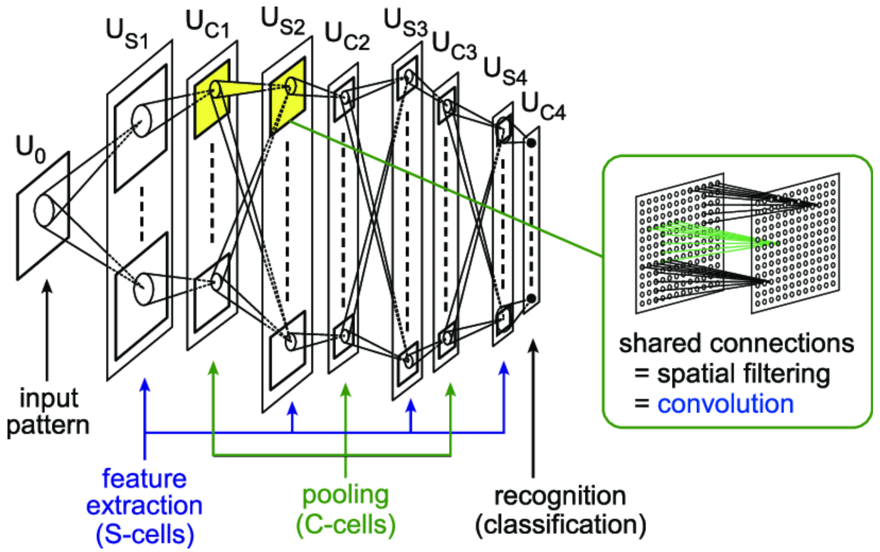
Some neurons react to **more complex patterns** that are combinations of the lower-level patterns



From Visual Cortex to CNNs

Neurocognitron (1980) gave early inspiration for and gradually evolved into convolutional neural networks.

LeNet-5 architecture introduced new building blocks – **convolutional layers** and **pooling layers**



U_0
U_{S1}
U_{C1}
U_{S2}
U_{C2}
U_{S3}
U_{C3}
U_{S4}
U_{C4}
input pattern
feature extraction (S-cells)
pooling (C-cells)
recognition (classification)
shared connections = spatial filtering = convolution

Fukushima (1980). *A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position*
LeCun et al. (1998). *Gradient-Based Learning Applied to Document Recognition*

From Visual Cortex to CNNs

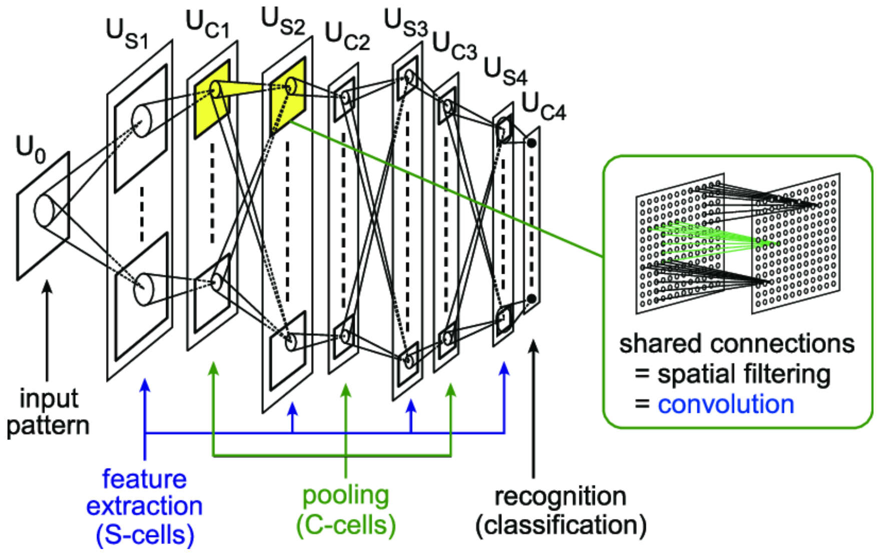
Neurocognitron (1980) gave early inspiration for and gradually evolved into convolutional neural networks.

LeNet-5 architecture introduced new building blocks – **convolutional layers** and **pooling layers**

Can you apply a regular DNN with fully connected layers to image recognition?

- Input of 100 × 100 pixels = 10^4
- First layer with 1,000 neurons

  ⇒ 10 million connections!


U_{S1}
U_{C1}
U_{S2}
U_{C2}
U_{S3}
U_{C3}
U_{S4}
U_{C4}
U_0
input pattern
feature extraction (S-cells)
pooling (C-cells)
recognition (classification)
shared connections = spatial filtering = convolution

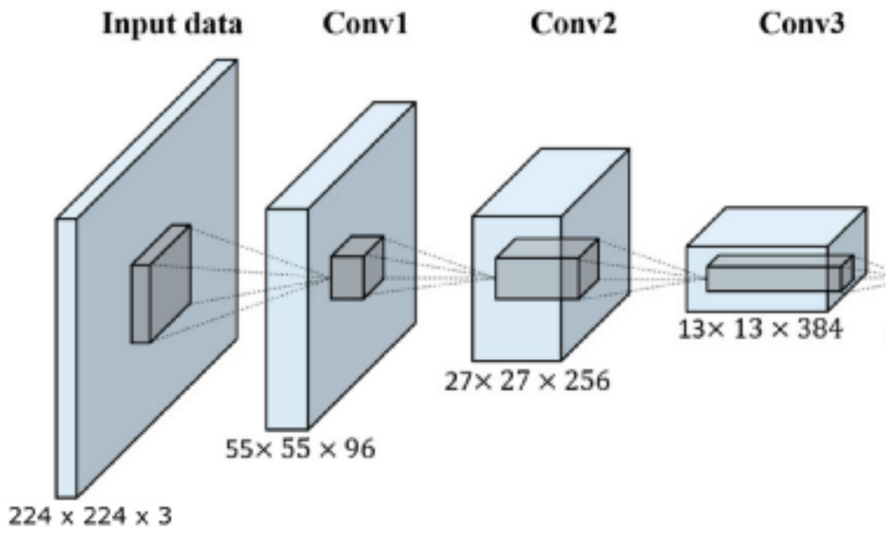
Fukushima (1980). *A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position*
LeCun et al. (1998). *Gradient-Based Learning Applied to Document Recognition*

Convolutional Layers

Only connect neurons in the first convolutional layer to the input pixels from the **receptive field** (focus on low-level features)

Only connect neurons in the second convolutional layer to the relevant **small rectangle** from the first layer (higher-level features)

Hierarchical structure at work

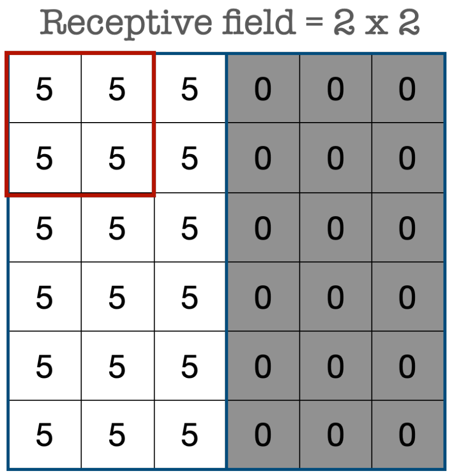


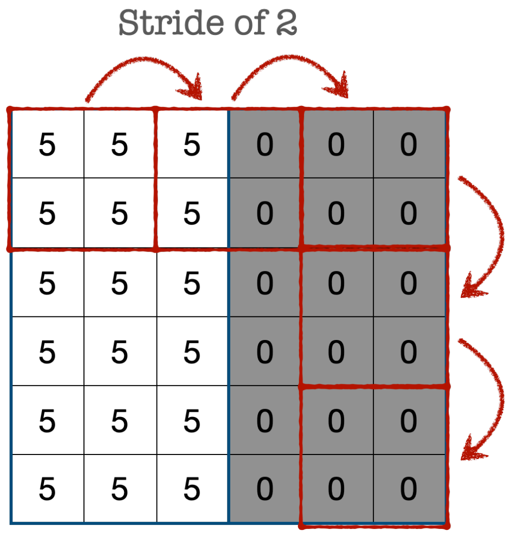
https://cs231n.github.io/convolutional-networks/#convert

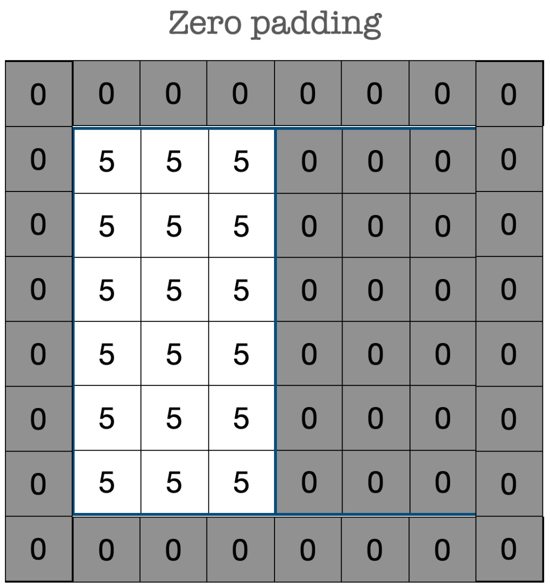
Convolutional Layers

Stride – distance between two consecutive receptive fields

Zero padding – adding zeros around the input to make it fit certain dimensionality





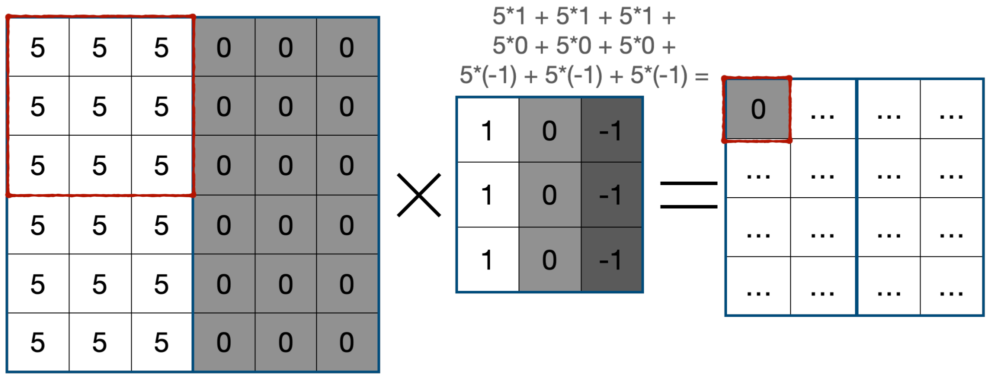


Filters

Filter (convolution kernel) is a set of weights

Feature map – a layer full of neurons using the same filter; highlights areas in an image that are most similar to the filter

Power of CNNs: if it learns to recognise a pattern in one location of an image, it can recognise it elsewhere. Traditional DNN can only recognise a pattern in a specific location

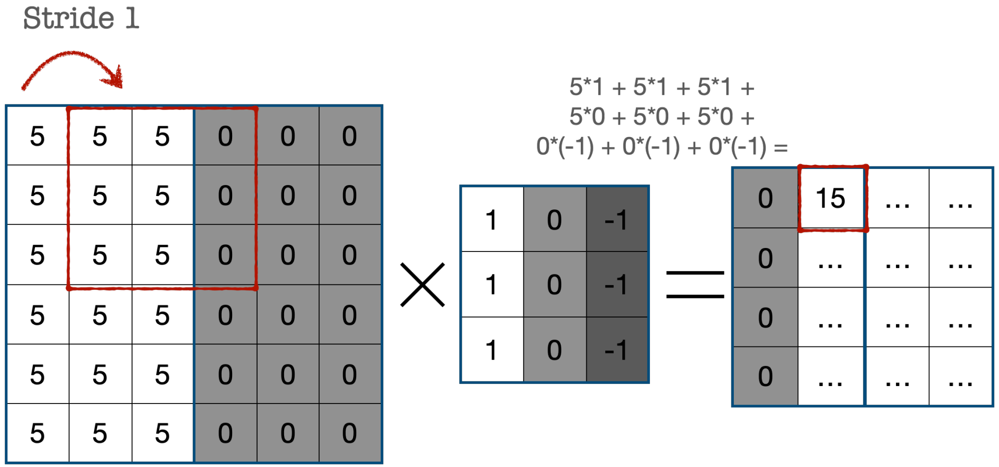


Filters

Filter (convolution kernel) is a set of weights

Feature map – a layer full of neurons using the same filter; highlights areas in an image that are most similar to the filter

Power of CNNs: if it learns to recognise a pattern in one location of an image, it can recognise it elsewhere. Traditional DNN can only recognise a pattern in a specific location



5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0

1	0	-1
1	0	-1
1	0	-1

0	15	...	...
0	...	...	...
0	...	...	...
0	...	...	...

Filters

Filter (convolution kernel) is a set of weights

Feature map – a layer full of neurons using the same filter; highlights areas in an image that are most similar to the filter

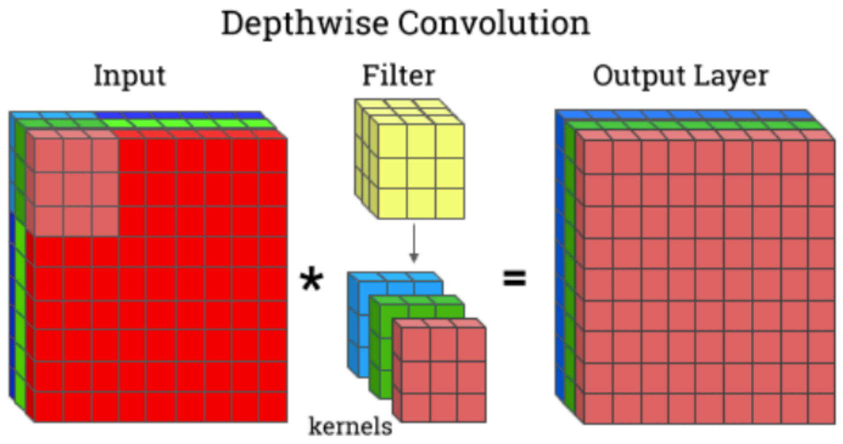
Power of CNNs: if it learns to recognise a pattern in one location of an image, it can recognise it elsewhere. Traditional DNN can only recognise a pattern in a specific location

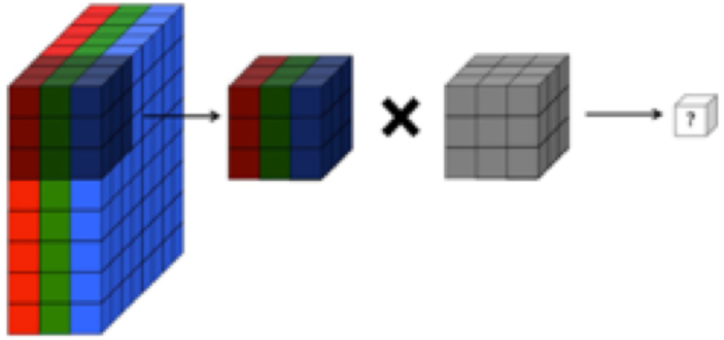
$$\begin{bmatrix} 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 15 & 15 & 0 \\ 0 & 15 & 15 & 0 \\ 0 & 15 & 15 & 0 \\ 0 & 15 & 15 & 0 \end{bmatrix}$$

Stacking Multiple Feature Maps

Multiple features maps are stacked on top of each other – 3D representation more appropriate

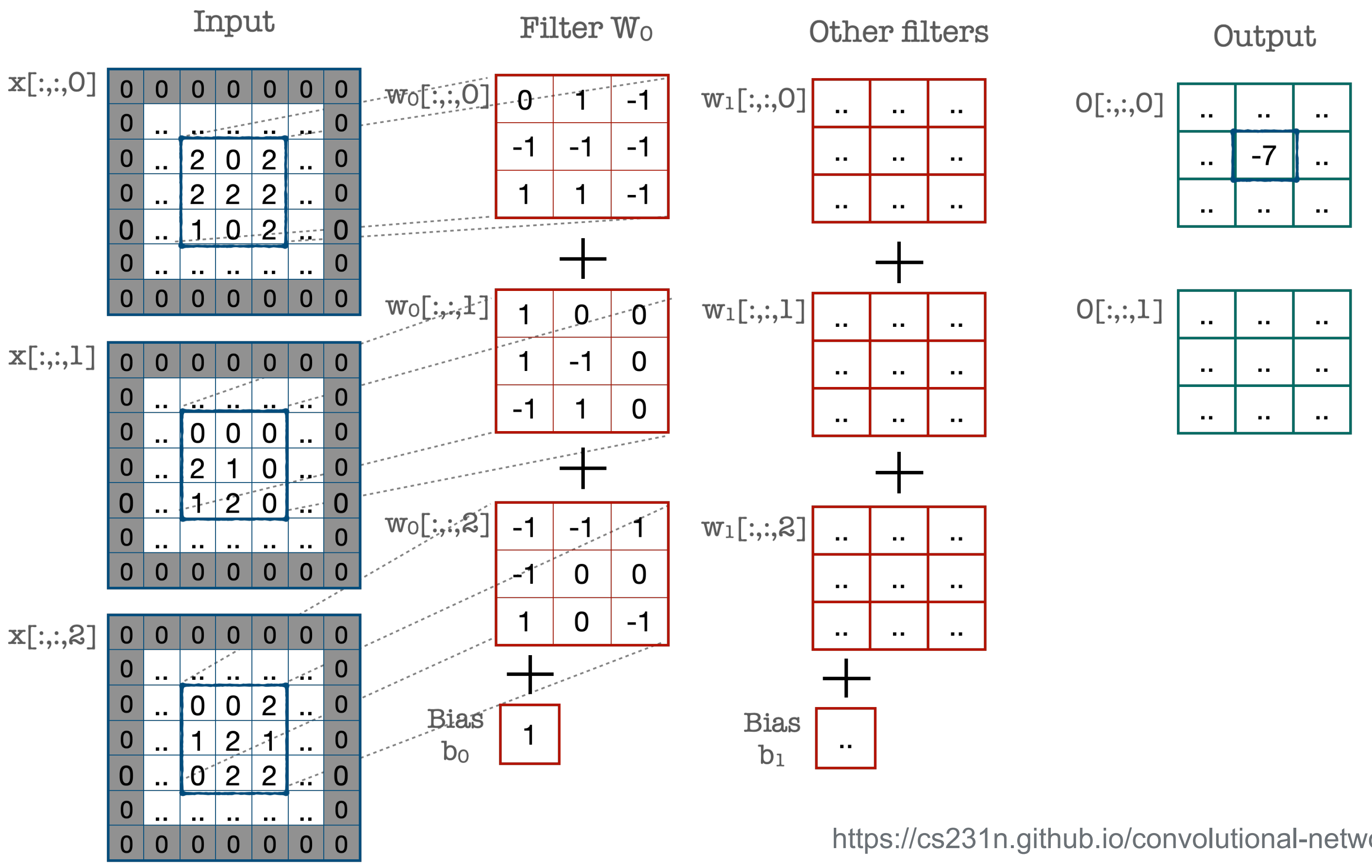
Feature maps applied to each of the RGB channels





https://www.researchgate.net/figure/Example-of-a-depthwise-separable-convolution-on-an-RGB-image-with-N-1_fig1_338593957

Stacking Multiple Feature Maps



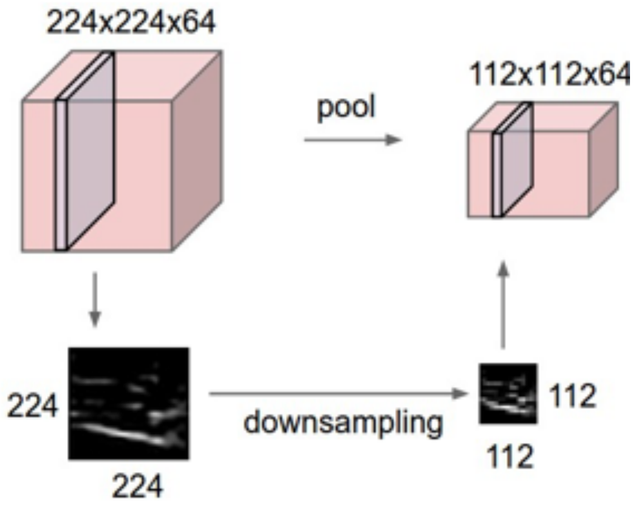
https://cs231n.github.io/convolutional-networks/#convert

Pooling Layers

Pooling layers' goal is to **subsample** (shrink) the image to reduce the computational load, the memory usage, the number of parameters (i.e., to avoid overfitting), and make the network tolerate a bit of image shift (i.e., introduce **location invariance**).

Each neuron in the pooling layer is connected to the outputs of a limited number of neurons from the previous layer.

Need to define the size, the stride, and the padding type as before.



https://cs231n.github.io/convolutional-networks/#convert

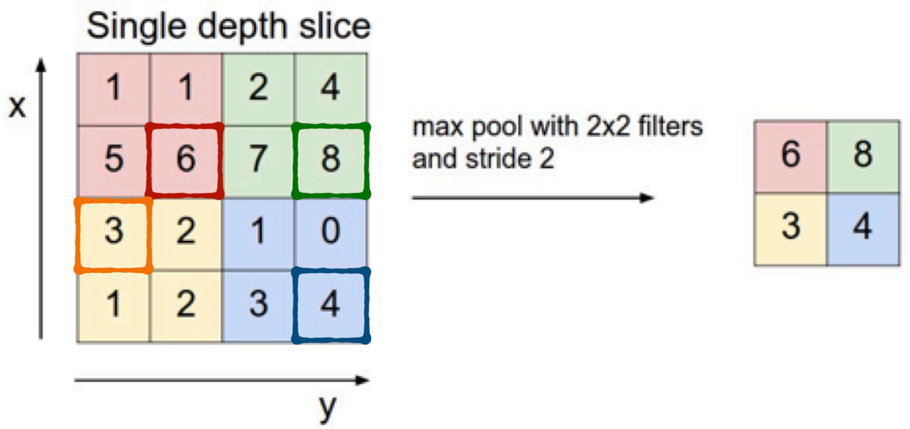
Types of Pooling

Pooling layers don’t contain weights. All they do is aggregate the input from the previous layer, e.g. using *max* or *mean (avg)*

With **max pooling,** only the **max** input value in each kernel makes it to the next layer. The other inputs are dropped.

This technique shrinks the image quite a lot: 2-by-2 kernel with a stride of 2 results in 4-times smaller image (75% drop!)

You may also apply it to channels (**depth-wise**), in which case image dimensions stay the same but depth is reduced



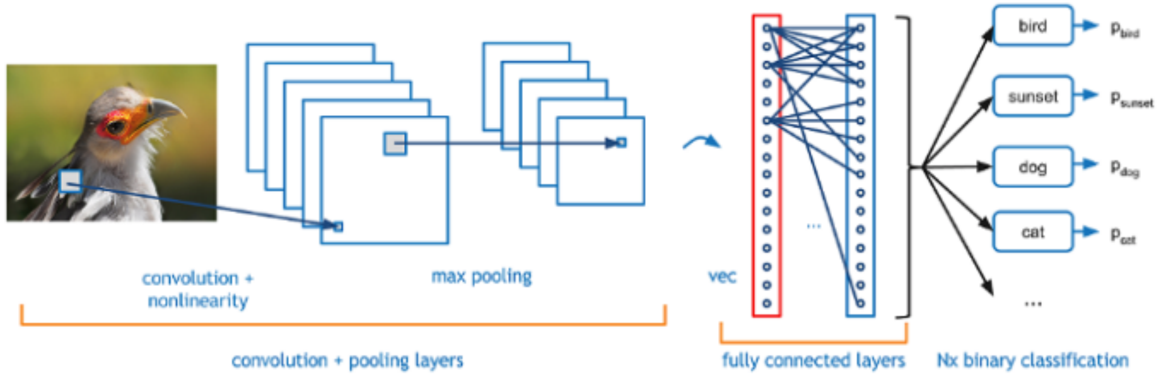
https://cs231n.github.io/convolutional-networks/#convert

Assemble a CNN

The image gets **smaller and smaller** as it progresses through the network, but also **deeper and deeper** as more feature maps are added

At the top of the stack – regular feedforward neural network composed of a few fully connected layers and ReLU activation

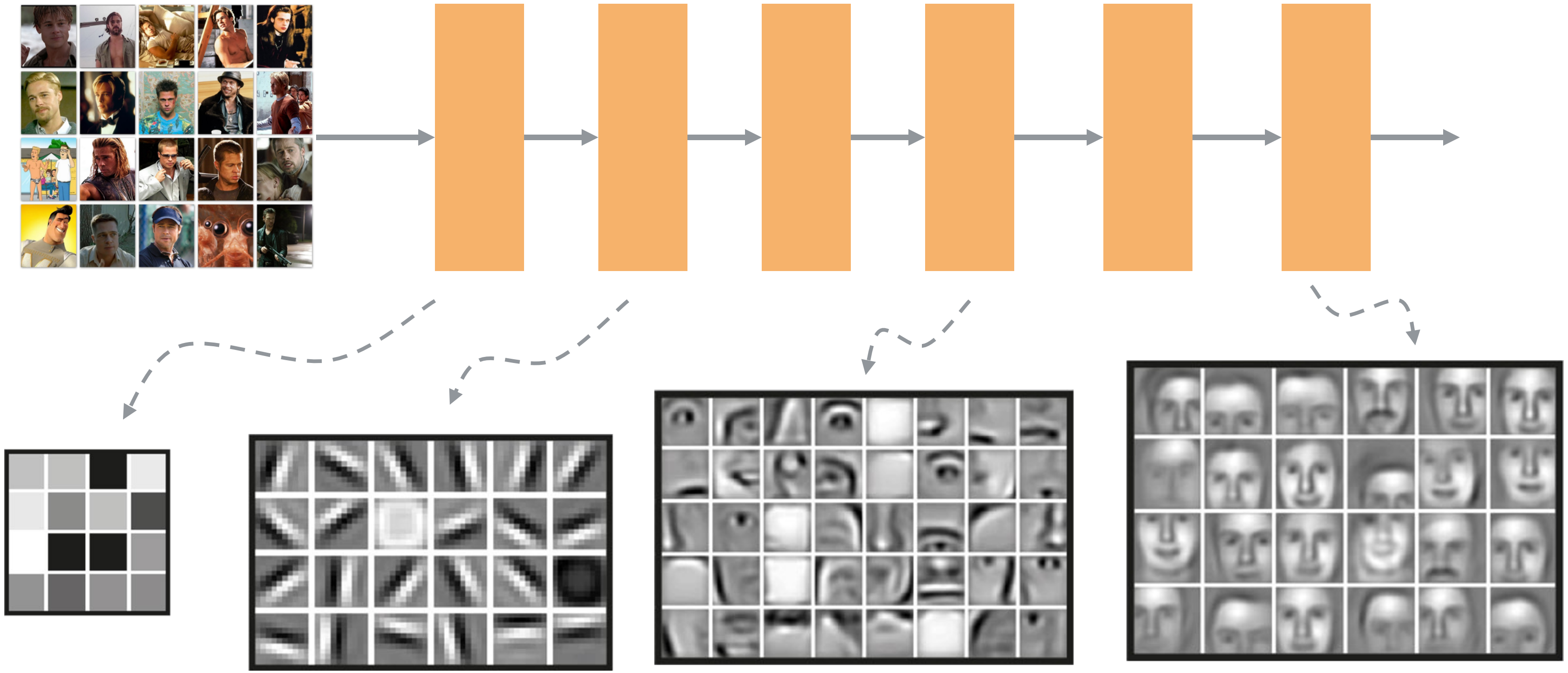
The final layer outputs predictions using softmax



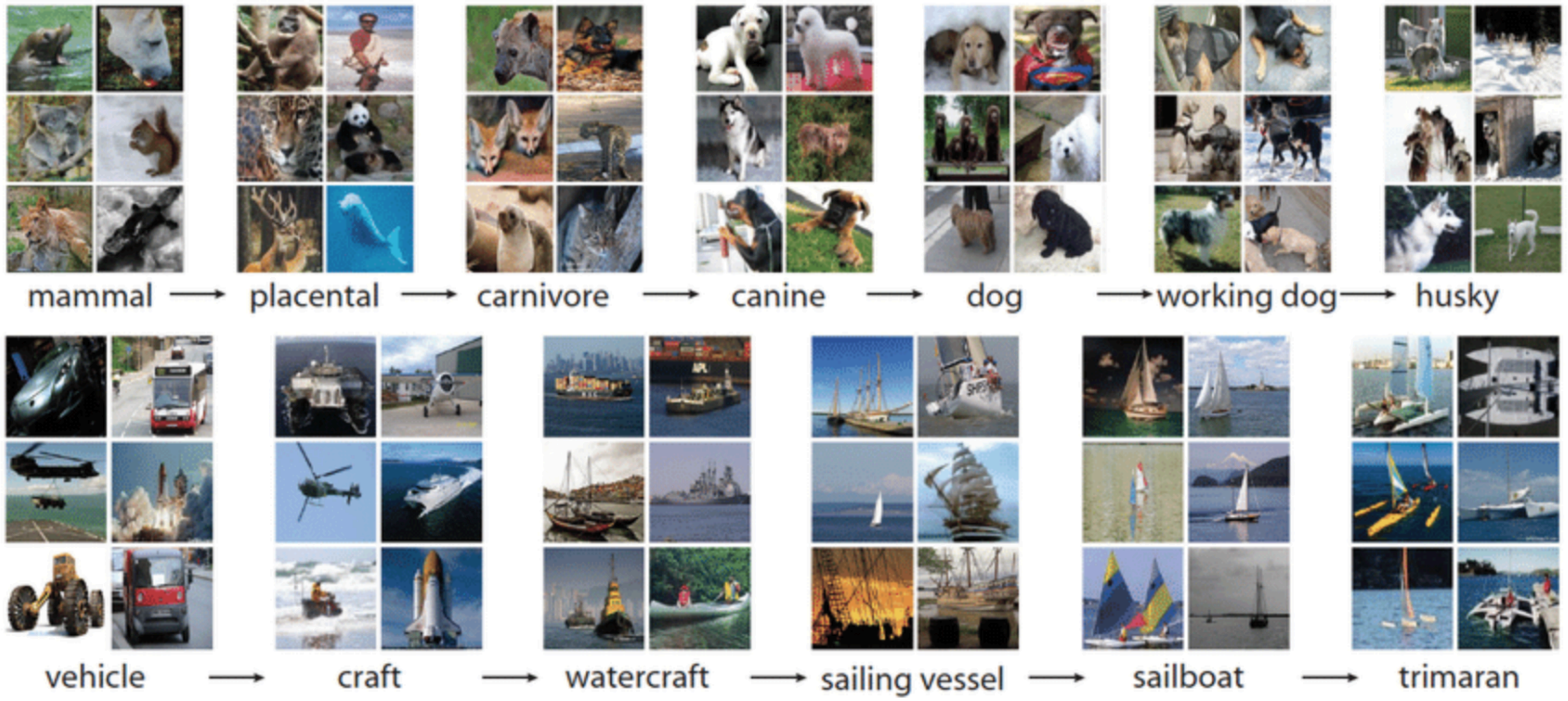
https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

Recap: Learning Representations & Features

Automatically learning increasingly more complex feature detectors from the data.

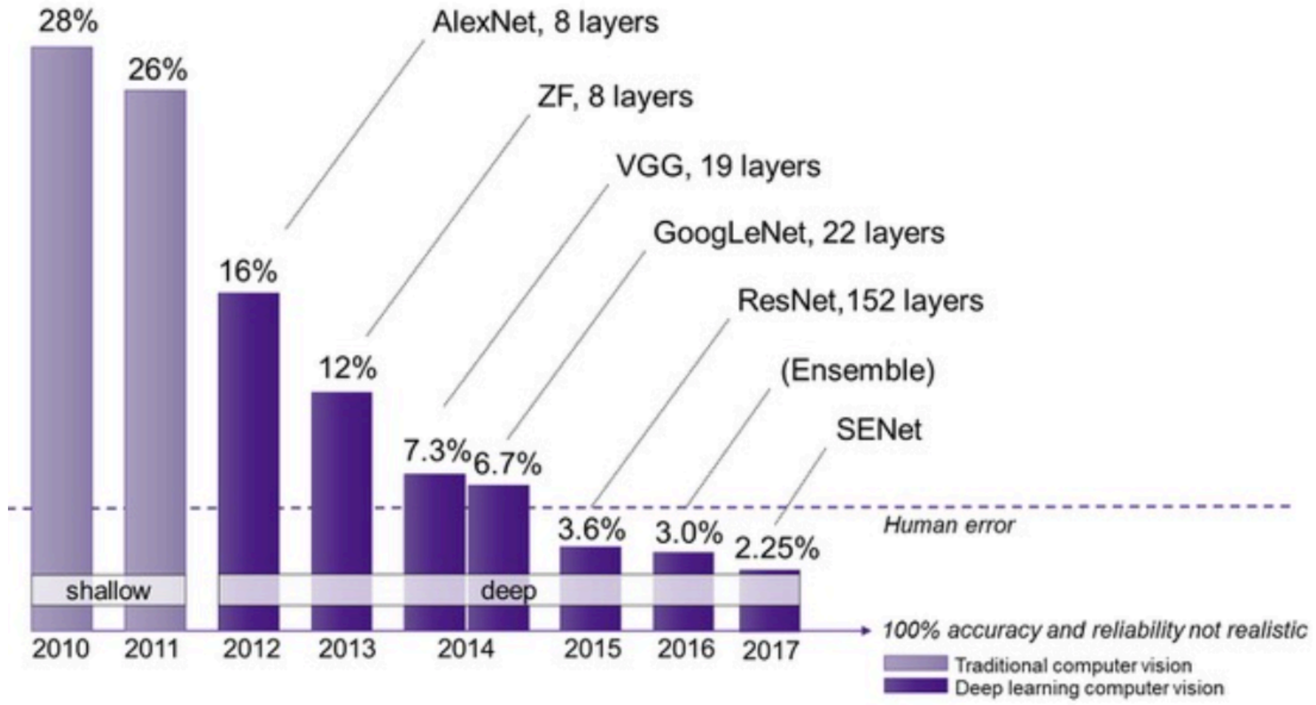


ImageNet Large Scale Visual Recognition Challenge


mammal → placental → carnivore → canine → dog → working dog → husky
vehicle → craft → watercraft → sailing vessel → sailboat → trimaran

Ye (2018). *Visual Object Detection from Lifelogs using Visual Non-lifelog Data*
https://devopedia.org/imagenet

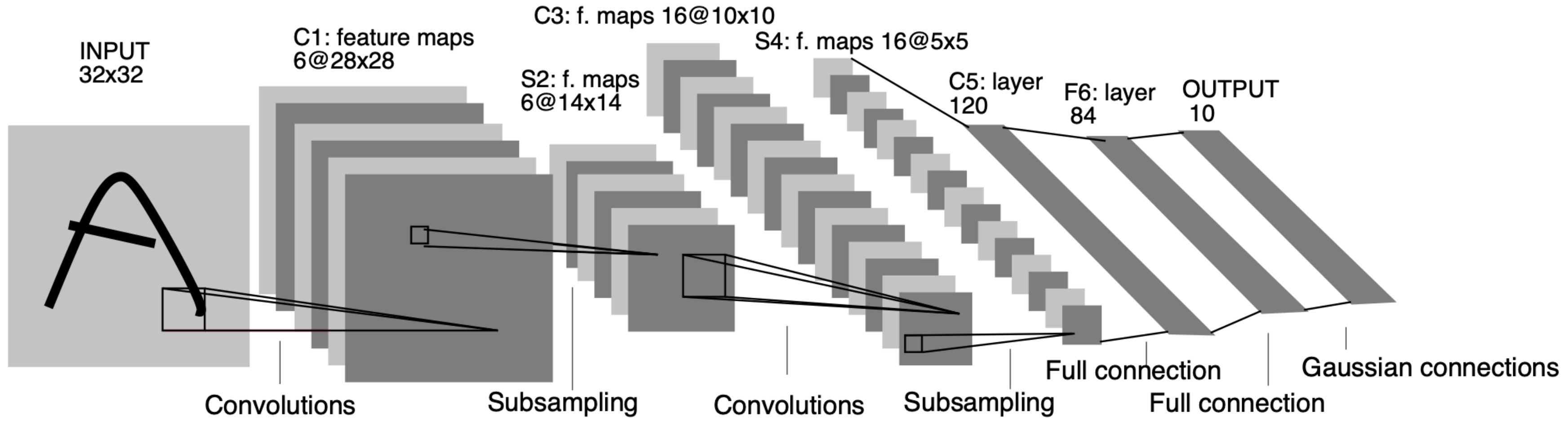
ImageNet Large Scale Visual Recognition Challenge



https://semiengineering.com/new-vision-technologies-for-real-world-applications/

LeNet-5

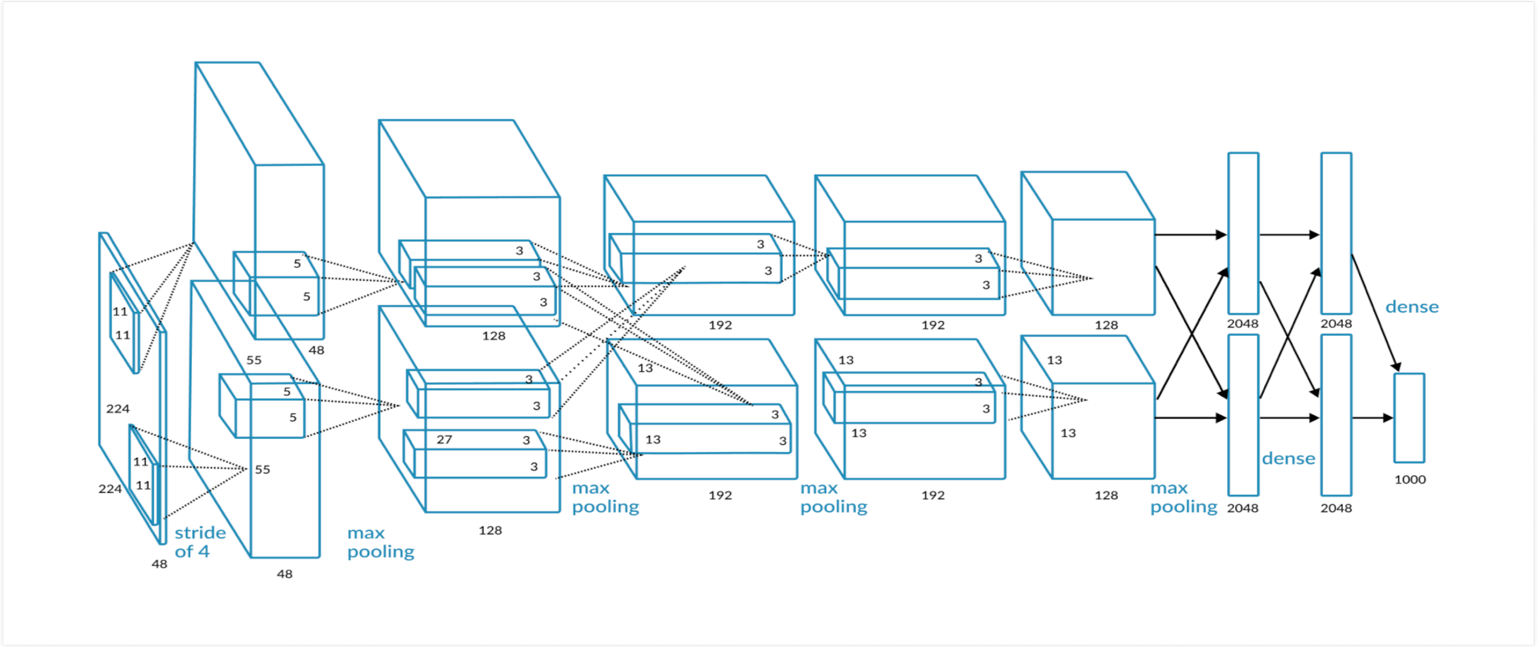
MNIST hand-written digit recognition



LeCun et al. (1998). *Gradient-Based Learning Applied to Document Recognition*
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

AlexNet

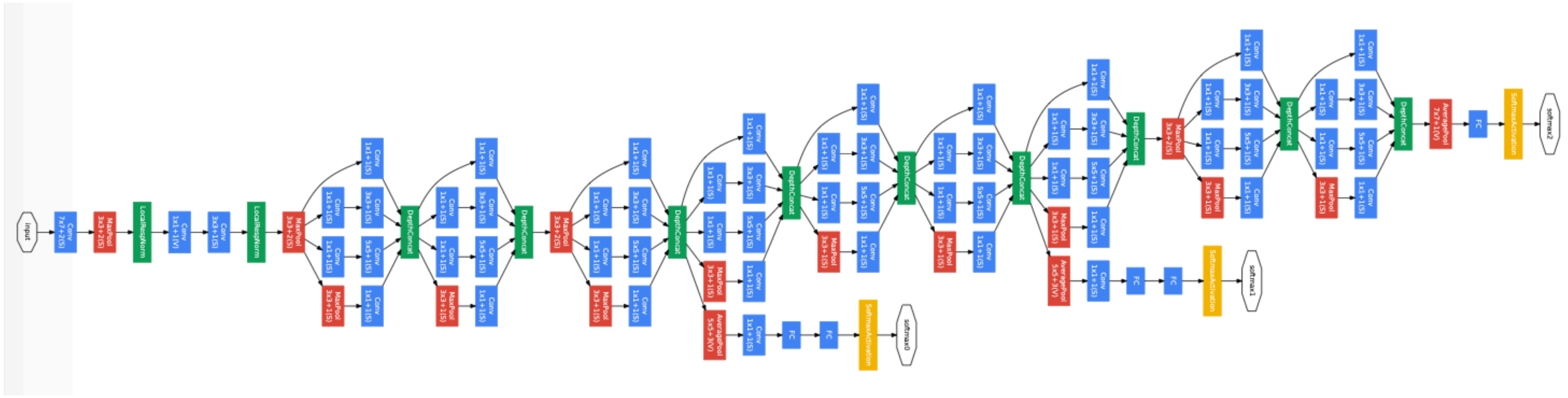
Top-5 error rate down to 17% (from 26%) on ImageNet



Krizhevsky et al. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*
https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/

GoogleNet

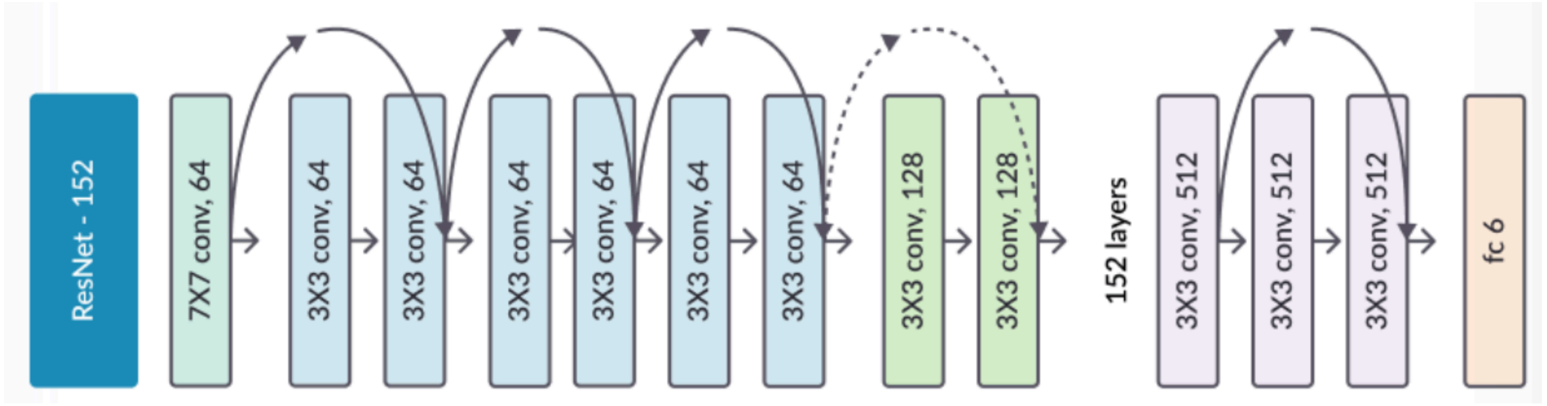
Top-5 error rate down to 7% on ImageNet



Szegedy et al. (2014). *Going Deeper with Convolutions*
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43022.pdf

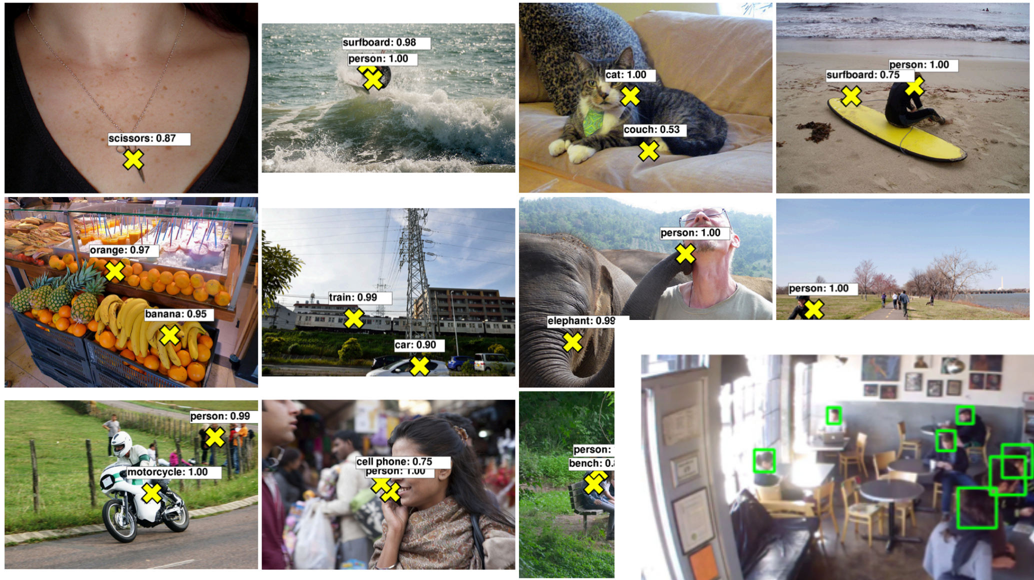
Finally, Residual Network (ResNet)

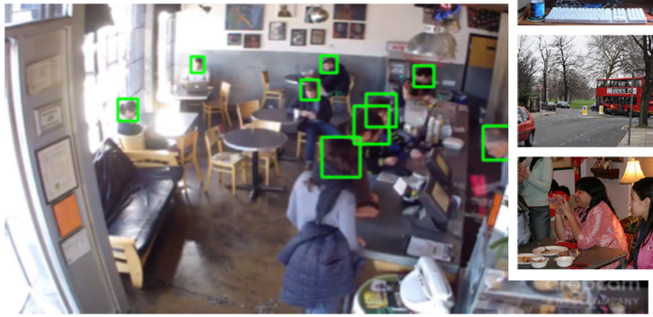
Top-5 error rate under 3.6% on ImageNet… using 152 layers

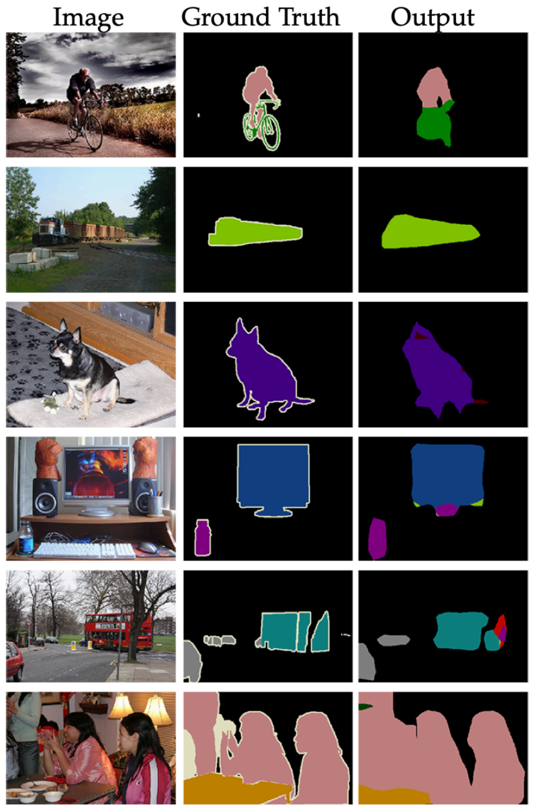


He et al. (2015). *Deep Residual Learning for Image Recognition*
https://missinglink.ai/guides/keras/keras-resnet-building-training-scaling-residual-nets-keras/

Other Visual Tasks







Oquab et al. (2015). *Is object localization for free? – Weakly-supervised learning with convolutional neural networks*; Stewart & Andriluka (2015). *End-to-end people detection in crowded scenes*; Shelhamer et al. (2016). *Fully Convolutional Networks for Semantic Segmentation*

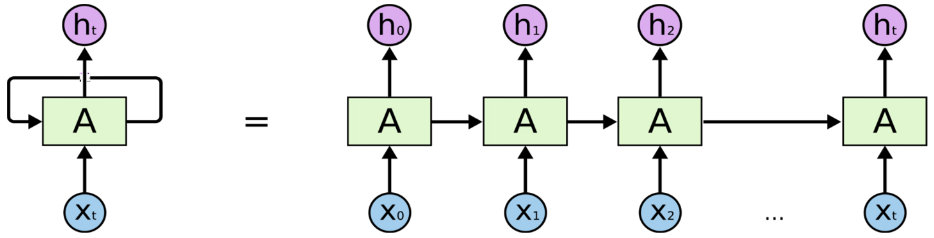
Recurrent Neural Networks

Predicting the Future

RNNs are good for predicting future events (to a certain extent) – e.g., future stock market prices, next word in a sequence, next note in a melody, next move in a scene, etc.

Can work on **sequences of arbitrary length** unlike architectures we've discussed so far

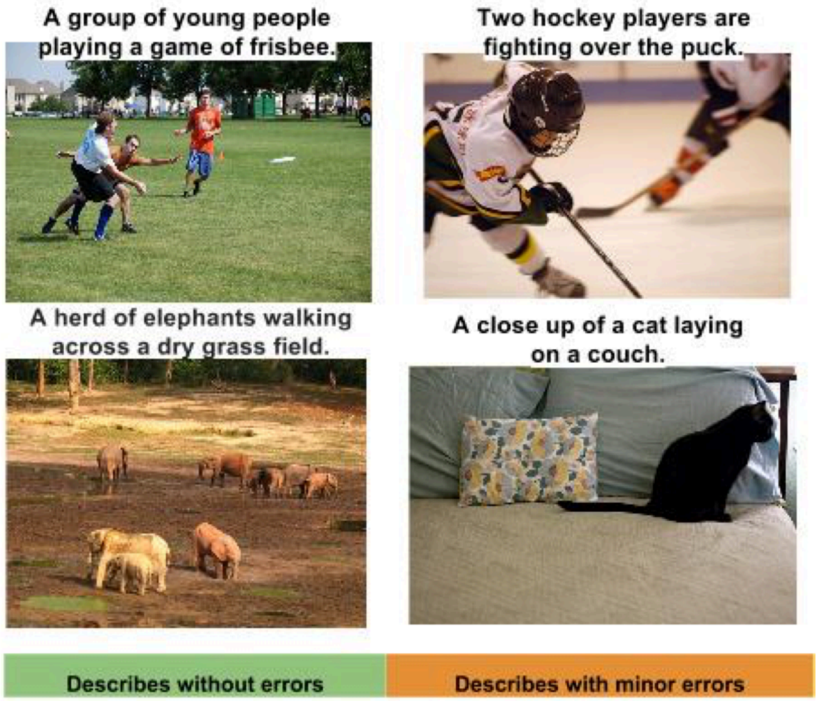
Suitable for **time-series data analysis**



Sequence Generation with RNNs

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```


A group of young people playing a game of frisbee.
Two hockey players are fighting over the puck.
A herd of elephants walking across a dry grass field.
A close up of a cat laying on a couch.
Describes without errors
Describes with minor errors

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

Vinyals et al. (2015) *Show and Tell: A Neural Image Caption Generator*
https://arxiv.org/pdf/1411.4555v2.pdf

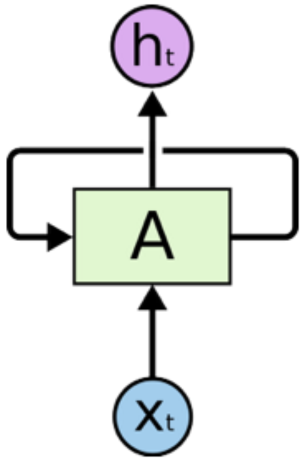
RNNs in a Nutshell

In traditional feedforward neural networks information flows in one direction only

At every time step, start learning “from scratch”

Recurrent units – connections pointing backwards

Block **A** here looks at the input from $\mathbf{x_t}$ and outputs $\mathbf{h_t}$



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNNs in a Nutshell

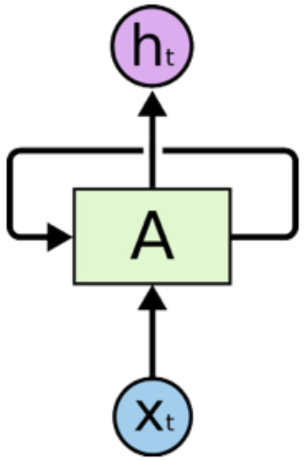
Let's unroll this

In traditional feedforward neural networks information flows in one direction only

At every time step, start learning "from scratch"

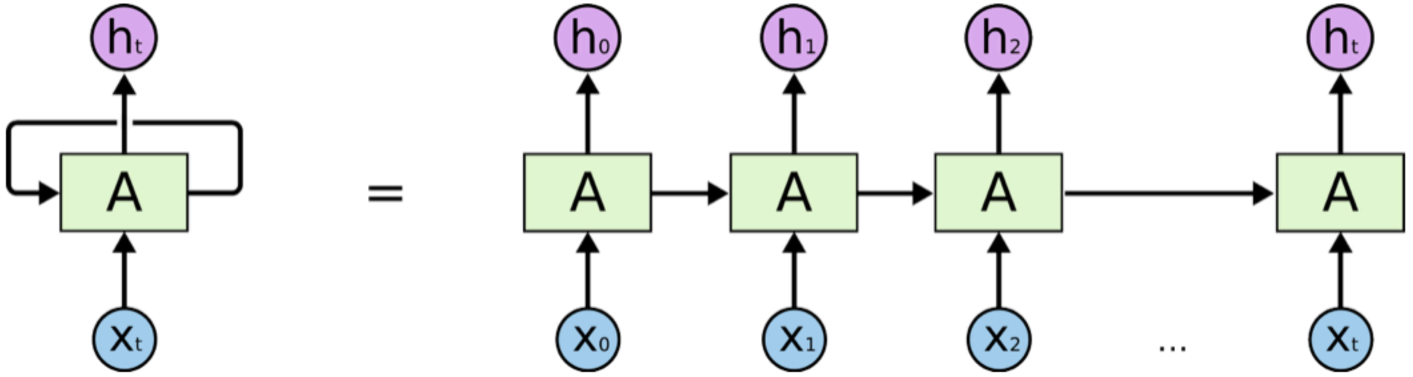
Recurrent units – connections pointing backwards

Block **A** here looks at the input from $\mathbf{x_t}$ and outputs $\mathbf{h_t}$



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolling the Network through Time

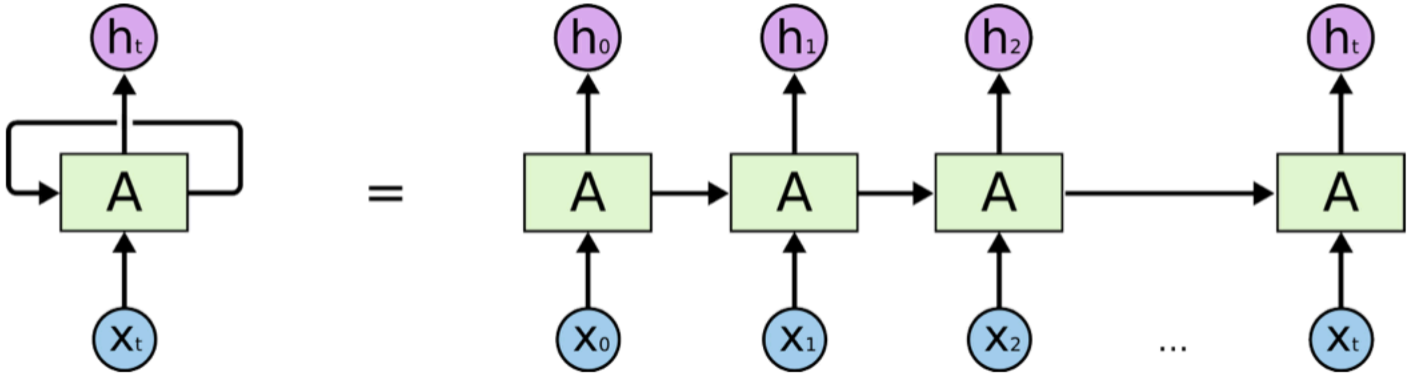


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN contains multiple copies of the same network, each passing a message to a successor

At each time step **t** (frame) a recurrent neuron receives the inputs $\mathbf{x}_{(t)}$ as well as its own output from the previous time step $\mathbf{h}_{(t-1)}$

Unrolling the Network through Time

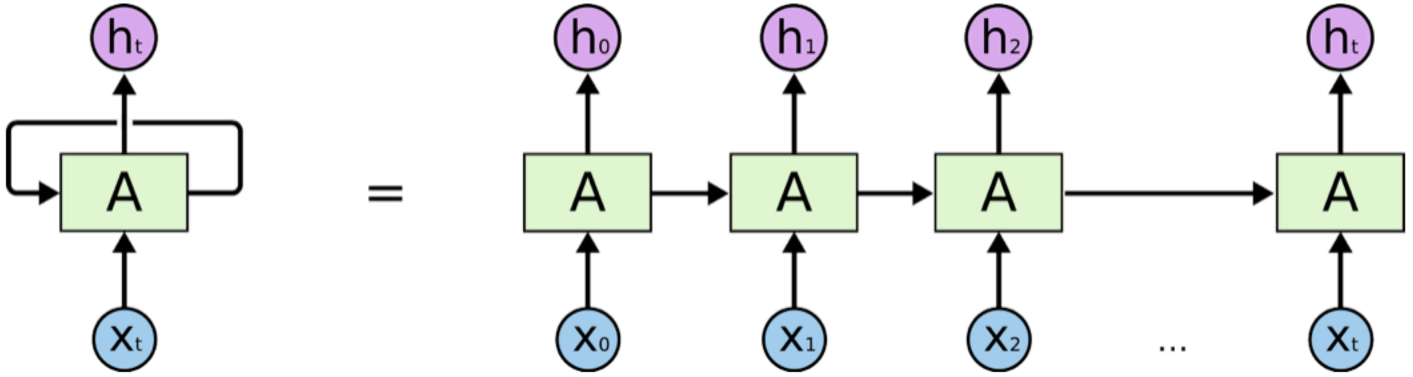


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Now we have two sets of weights: one for the inputs $\mathbf{x}_{(t)}$ and the other for the outputs of the previous time step $\mathbf{h}_{(t-1)}$

Let's call them $\mathbf{w}_x$ and $\mathbf{w}_h$

Mathematical Definition



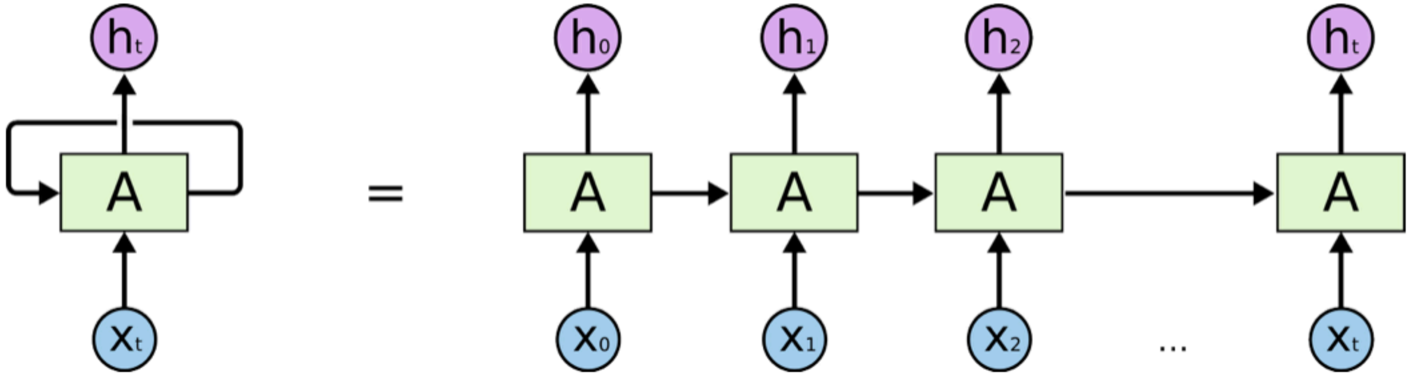
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$h_t = \phi(x_{(t)}^T \cdot w_x + h_{(t-1)}^T \cdot w_h + b)$$

Vectorised form: $$H_{(t)} = \phi(X_{(t)} \cdot W_x + H_{(t-1)} \cdot W_h + b)$$

$$H_{(t)} = \phi([X_{(t)} \quad H_{(t-1)}] \cdot W + b) \text{ with } \mathrm{W} = \begin{bmatrix} W_x \\ W_h \end{bmatrix}$$

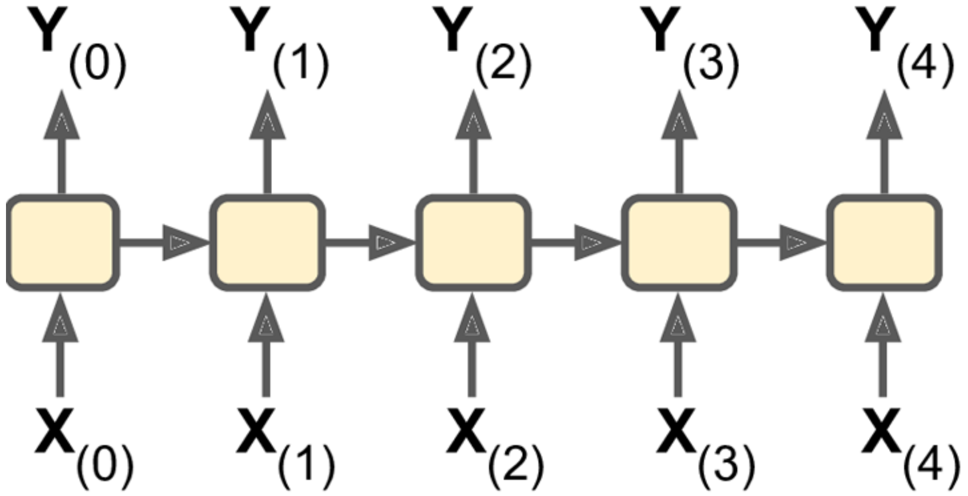
Memory Cells



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Note that $\mathbf{h}_{(t)}$ is a function of $\mathbf{x}_{(t)}$ and $\mathbf{h}_{(t-1)}$, where
$\mathbf{h}_{(t-1)}$ is a function of $\mathbf{x}_{(t-1)}$ and $\mathbf{h}_{(t-2)}$, where
$\mathbf{h}_{(t-2)}$ is a function…
⇒ $\mathbf{h}_{(t)}$ is a function of all the inputs since time **t=0**
You can say that each cell has a form of **memory**

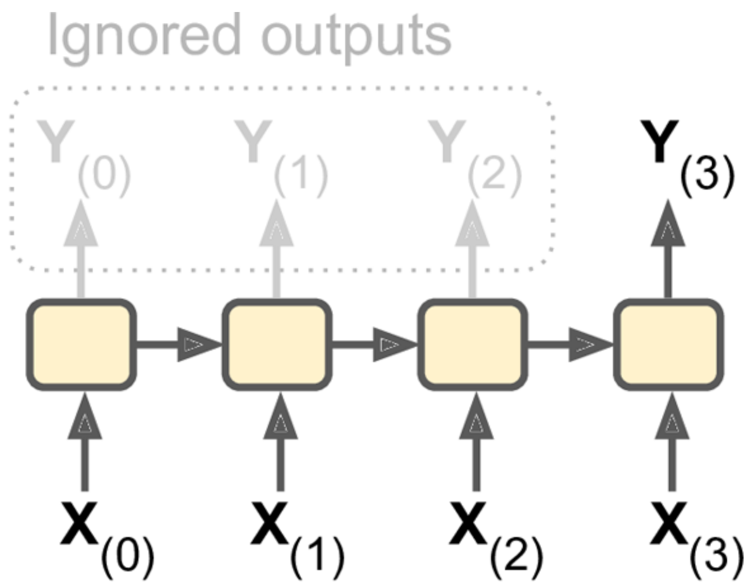
Sequence-to-Sequence



- **Example:** Stock prices
- **Input** – prices for the last N days
- **Output** – prices predictions starting from N-1 to tomorrow

Geron (2017). Hands-on Machine Learning with Scikit-Learn & TensorFlow

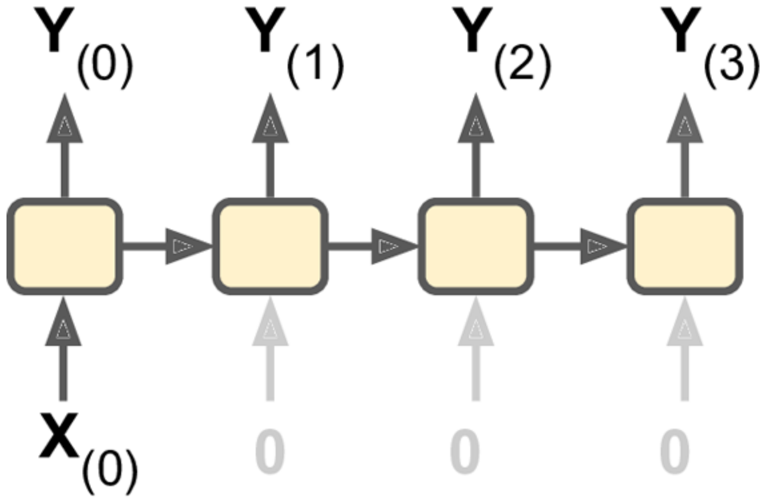
Sequence-to-Vector



- **Example:** Sentiment analysis
- **Input** – sequence of words (e.g., in a review)
- **Output** – single sentiment score prediction (e.g., 0=hate, 1=love)

Geron (2017). Hands-on Machine Learning with Scikit-Learn & TensorFlow

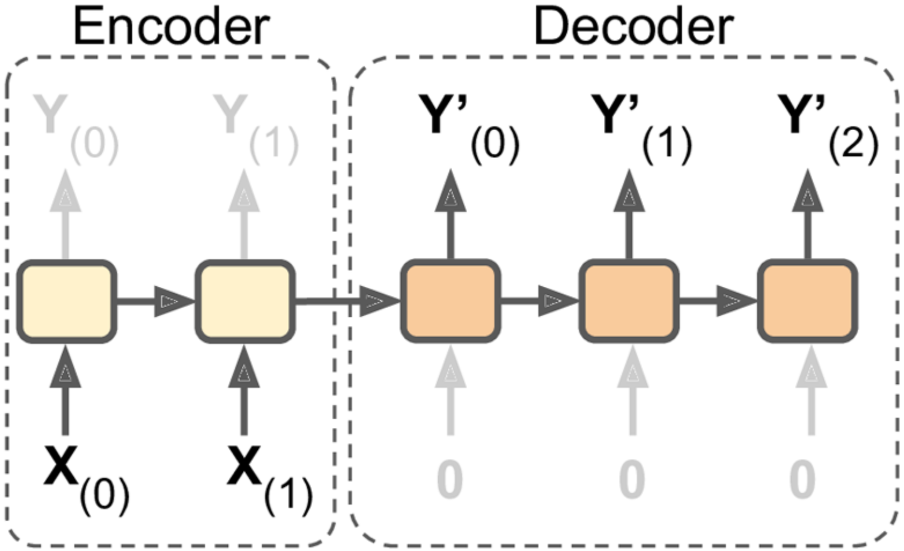
Vector-to-Sequence



- **Example:** Caption generation
- **Input** – single image
- **Output** – sequence of words in image caption

Geron (2017). Hands-on Machine Learning with Scikit-Learn & TensorFlow

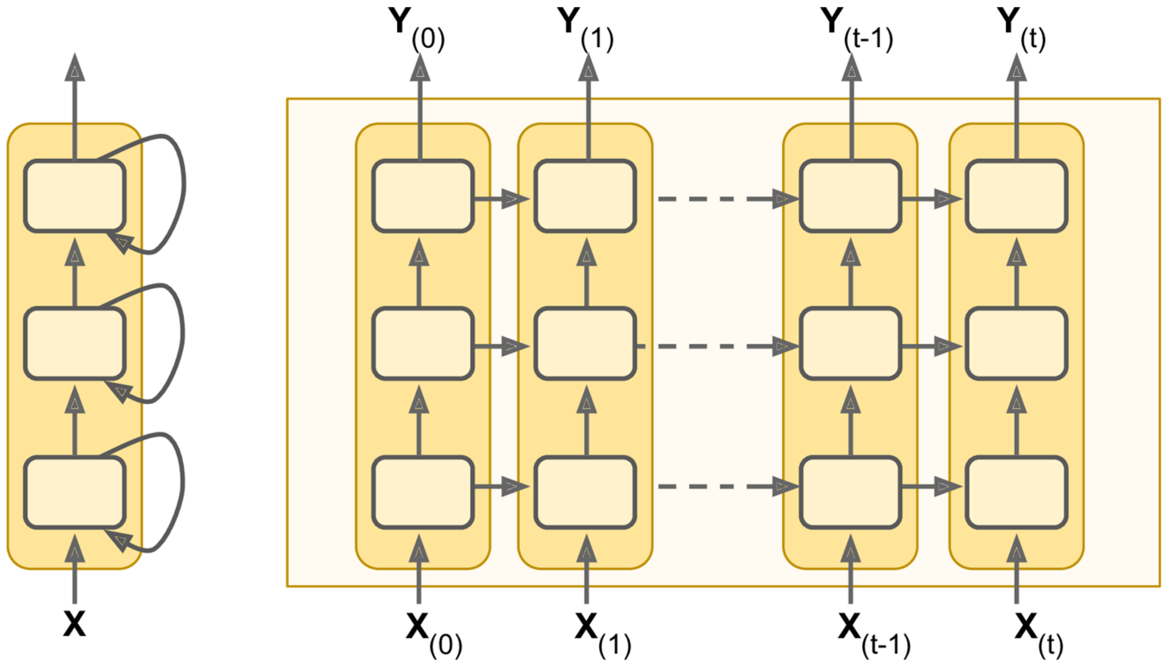
Delayed Sequence-to-Sequence



- **Example:** Machine Translation
- **Input** – sequence of words in L1 (need to “wait” to the end to get the message)
- **Output** – sequence of words in L2

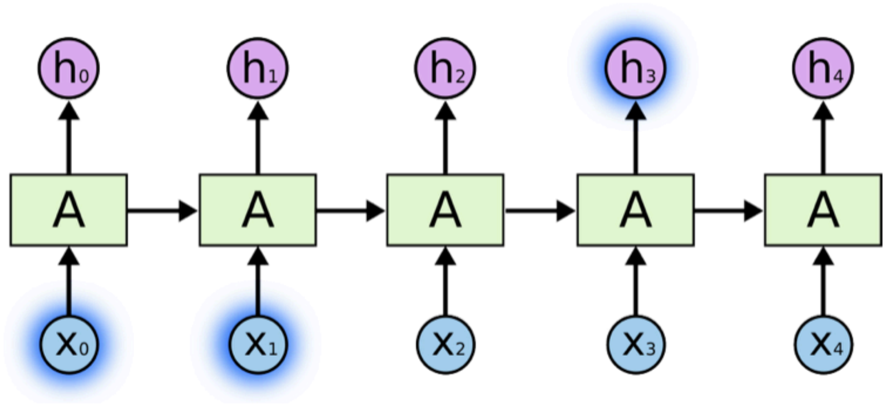
Geron (2017). Hands-on Machine Learning with Scikit-Learn & TensorFlow

Deep RNN



Geron (2017). Hands-on Machine Learning with Scikit-Learn & TensorFlow

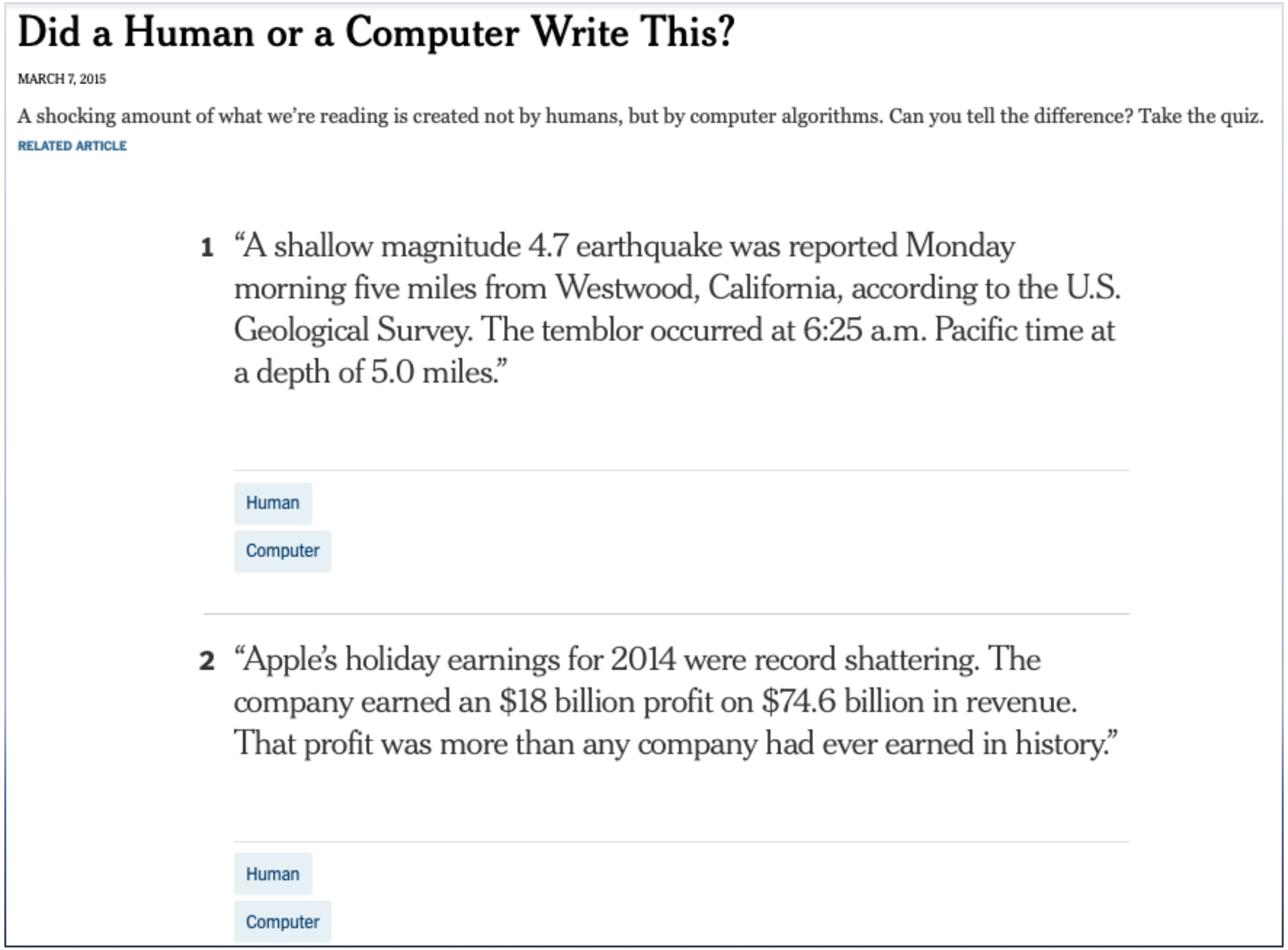
Problems with RNNs



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Problems with RNNs:

- Slow to train
- Vanishing and exploding gradients


Did a Human or a Computer Write This?

MARCH 7, 2015

A shocking amount of what we're reading is created not by humans, but by computer algorithms. Can you tell the difference? Take the quiz.

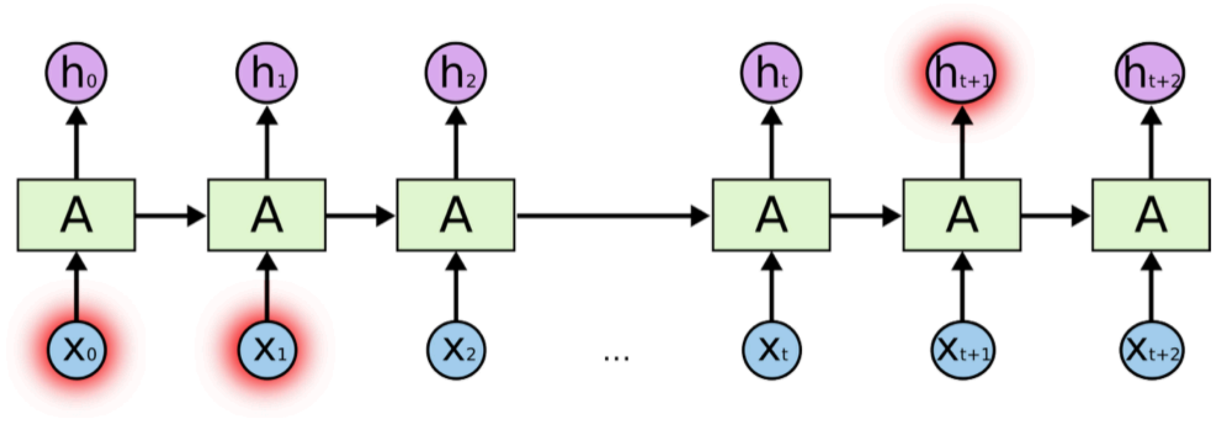
RELATED ARTICLE

1 "A shallow magnitude 4.7 earthquake was reported Monday morning five miles from Westwood, California, according to the U.S. Geological Survey. The temblor occurred at 6:25 a.m. Pacific time at a depth of 5.0 miles."

2 "Apple's holiday earnings for 2014 were record shattering. The company earned an $18 billion profit on $74.6 billion in revenue. That profit was more than any company had ever earned in history."

https://www.nytimes.com/interactive/2015/03/08/opinion/sunday/algorithm-human-quiz.html

Long Term Dependencies Problem

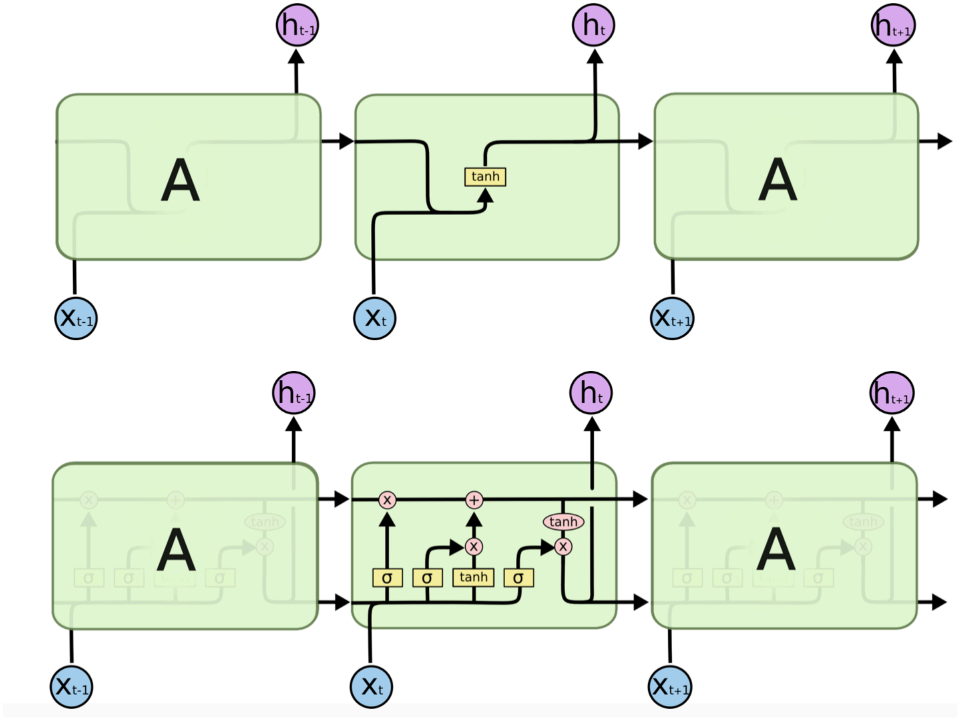


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Solution – Long Short-Term Memory (LSTM) cells
Proposed in 1997 by Hochreiter and Schmidhuber, and improved over the years
Can be used pretty much like any other cell – implementation available in TensorFlow

Hochreiter & Schmidhuber (1997). *Long Short-Term Memory*

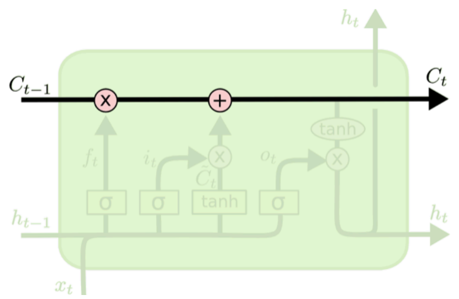
Long Short-Term Memory (LSTM) Networks

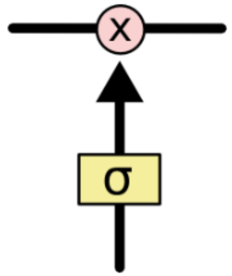


What's under the hood?

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM) Networks

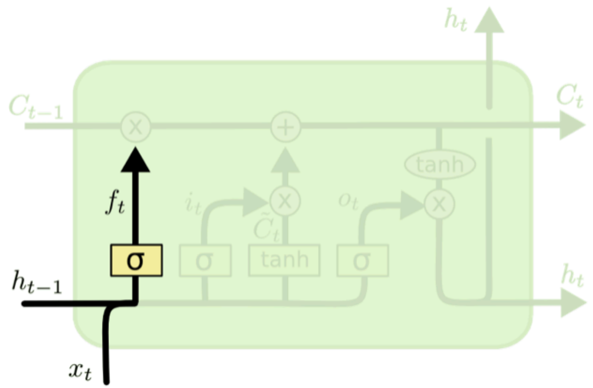




https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- “Conveyor belts” in LSTMs
- Sigmoid function controlling how much information passes though the gates: from 0 = none to 1 = “let it all through”
- $\mathbf{h}_{(t)}$ – short-term state, $\mathbf{c}_{(t)}$ – long-term state
- There are 3 types of gates in the cell: *forget gate*, *input gate*, and *output gate*

(1) Forget Gate Layer

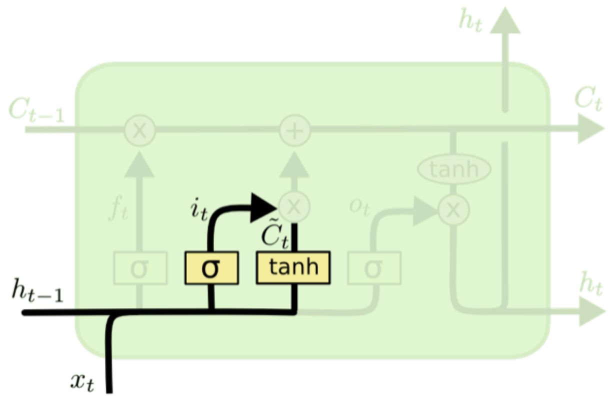


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- How much of the information from previous layers should be forgotten: 1 = “keep this in full”, 0 = “get rid of this”
- **Example**: forgetting the gender of the story characters when the narrative switches to a different character

(2) Input Gate Layer

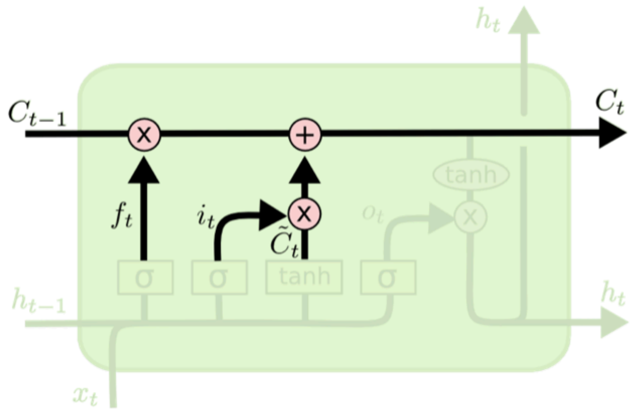


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- First, decide which values to update and then update these values ⇒ update the state: 1 = “keep this in full”, 0 = “get rid of this”
- **Example**: adding the gender of the new character in the story to replace the old one we are forgetting

(2) Update to the Old Cell

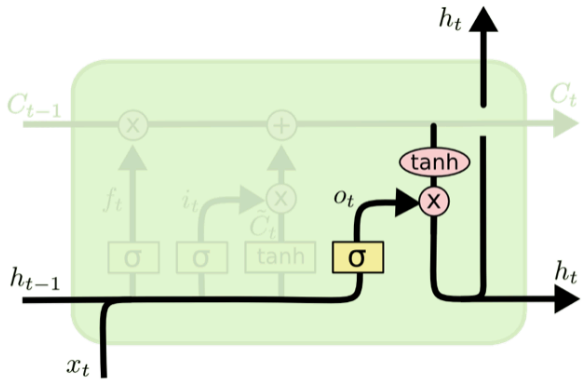


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Multiply the old state by $\mathbf{f_t}$ "forgetting" things that we've decided to forget
- Add $i_t \times \tilde{C}_t$ which is the new candidate, scaled by how much we need to update the state value
- **Example**: actually updating the information on the subject's gender

(3) Output Gate



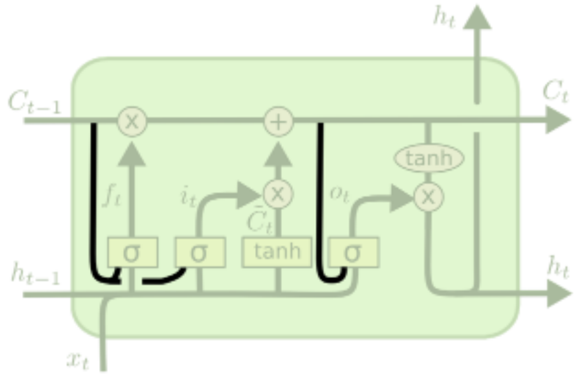
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Decide what we are going to output
- **Example**: output whether the subject of a verb is singular or plural so that we know what form of the verb to use next

Peephole Connections



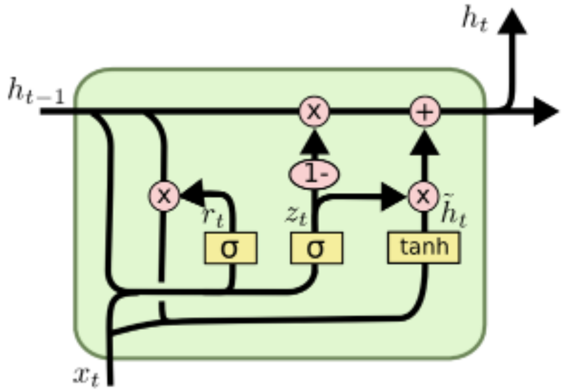
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- In basic LSTM, gate controllers can only look at the input $\mathbf{x}_{(t)}$ and the previous short-term state $\mathbf{h}_{(t-1)}$
- It may be useful to give them more context by allowing them to peek at the long-term state as well
- **Peephole connections**: add $\mathbf{c}_{(t-1)}$ to the forget and input gate, and $\mathbf{c}_{(t)}$ to the output

Gers & Schmidhuber (2000). *Recurrent Nets that Time and Count*

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- **Replace both state vectors** with a single vector **h$_{(t-1)}$**
- Introduce a **single gate controller** to control both forget and input gate: whenever a memory must be stored, the location where it will be stored should first be erased
- There is **no output gate**: a new gate controller controls which part of the previous state will be shown to the main layer

Cho et al. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*

Practical 5

Your task: Learning objectives

- The basics of CNNs

- Implementation of two building blocks in CNNs – convolutional and pooling layers with Keras and TensorFlow

- Application of CNNs to image analysis

- Classification of images from two popular image datasets

- **Assignment**: Build a CNN model to classify digits in the MNIST dataset and compare the results to other ML models (e.g., from previous practicals)

Practical 5 Logistics

- Data and code for Practical 5 can be found on: Github (https://github.com/ekochmar/cl-datasci-pnp-2021/tree/main/DSPNP_practical5)

- Practical ('ticking') session over Zoom at the time allocated by your demonstrator

- At the practical, be prepared to discuss the task and answer the questions about the code to get a 'pass'

- Upload your solutions (Jupyter notebook or Python code) to Moodle by the deadline (Thursday 26 November, 4pm)