
International Centre for Radio Astronomy Research

# ICRAR Machine Learning Workshop



ICRAR

International Centre for Radio Astronomy Research

# Introduction

A Brief History of Our Future Rulers

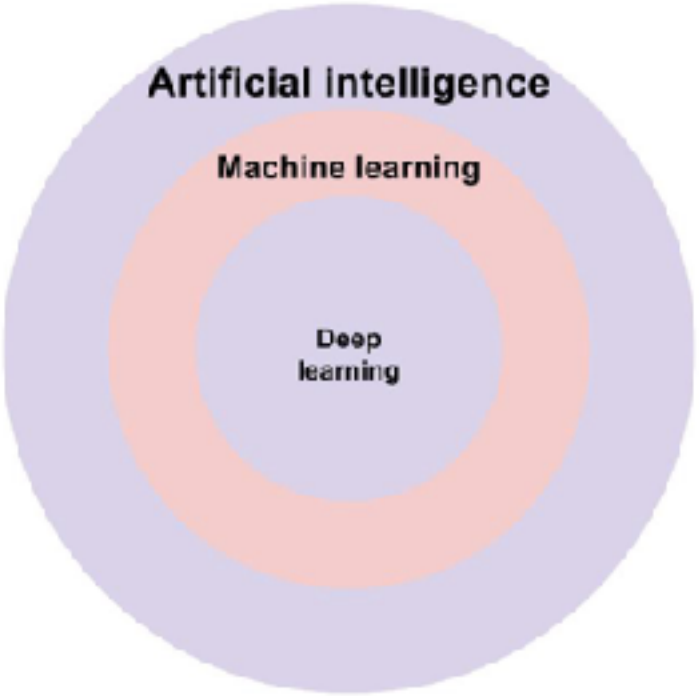
Curtin University

The University of Western Australia

Government of Western Australia
Department of the Premier and Cabinet
Office of Science

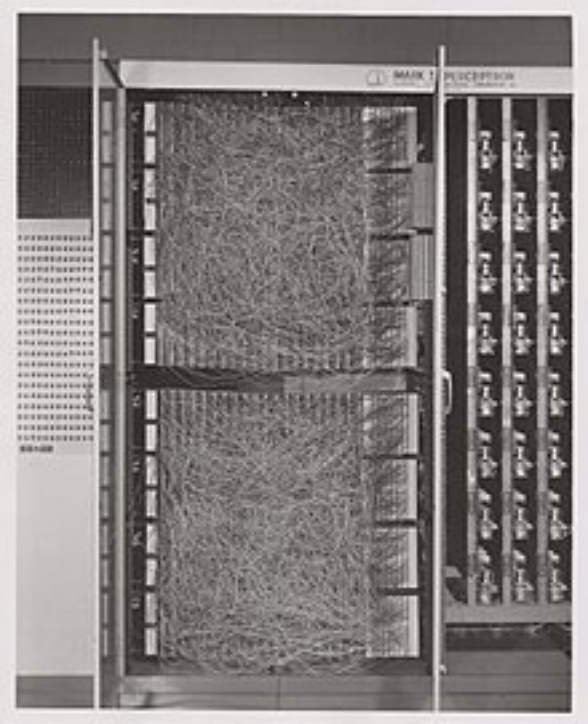
# What is Machine Learning?

- A relatively recent term to avoid saying artificial intelligence
- It is now a recognisably distinct subset of AI, containing the type of “deep learning” that we will mostly be focussed on today.



# The (Inglorious) History of AI

– The reasons for this rebrand is the slightly chequered history of AI research over the last 60 years.



- The first thing approaching recognisable AI was in fact what we might call a neural network, be it a physical one.
- This was a 20x20 pixel image recognition “perceptron”.
- It used a single layer of light sensors and motors to tune to “learning”.
- It was what we would today call a single layer feed forward neural network.
- Alas, claims made about what it could do, and how soon, rapidly spiraled out of control.
- When proof came that it could not tackle many simple tasks (e.g. XOR gate) funding was pulled, and we entered the first of many “AI Winters”

# AI Winters: A Regular Hype Cycle

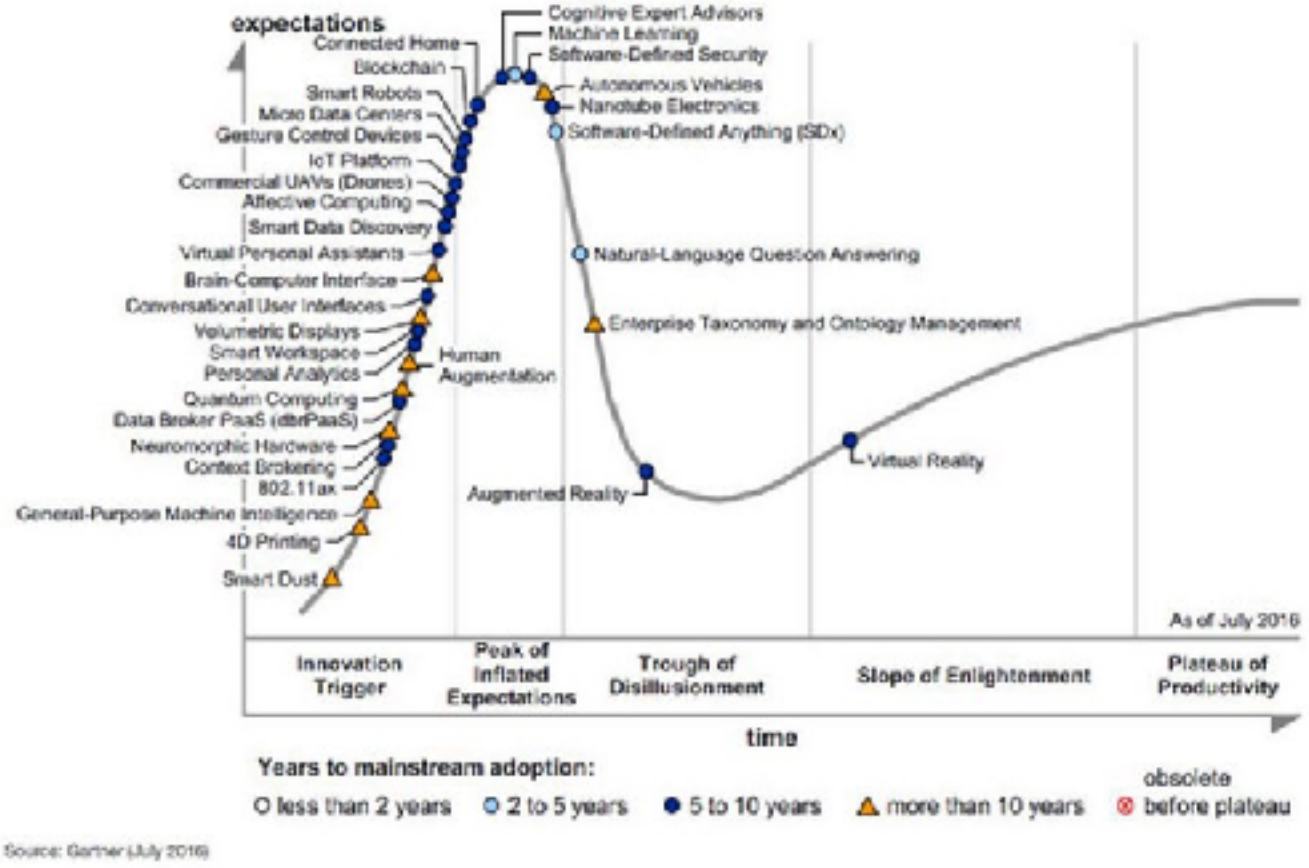
- Hype and then failure of the Perceptron:
  - **AI Winter of 1970-1980**

- Hype and then failure of LISP/expert machines, also the failure of the various attempts to build "fifth generation" AI machine:
  - **AI Winter of 1990-2000**

- In the mid 2000s a conscious effort was made to re-brand AI into various academic sub-domains, which is what we have today:
  - Informatics
  - Machine Learning
  - Data Mining
  - Big Data
- These sub-fields have regained credibility by reigning in the promises, and largely leaving the bombast of "thinking machines".

# Where is Machine Learning on the Hype Cycle?

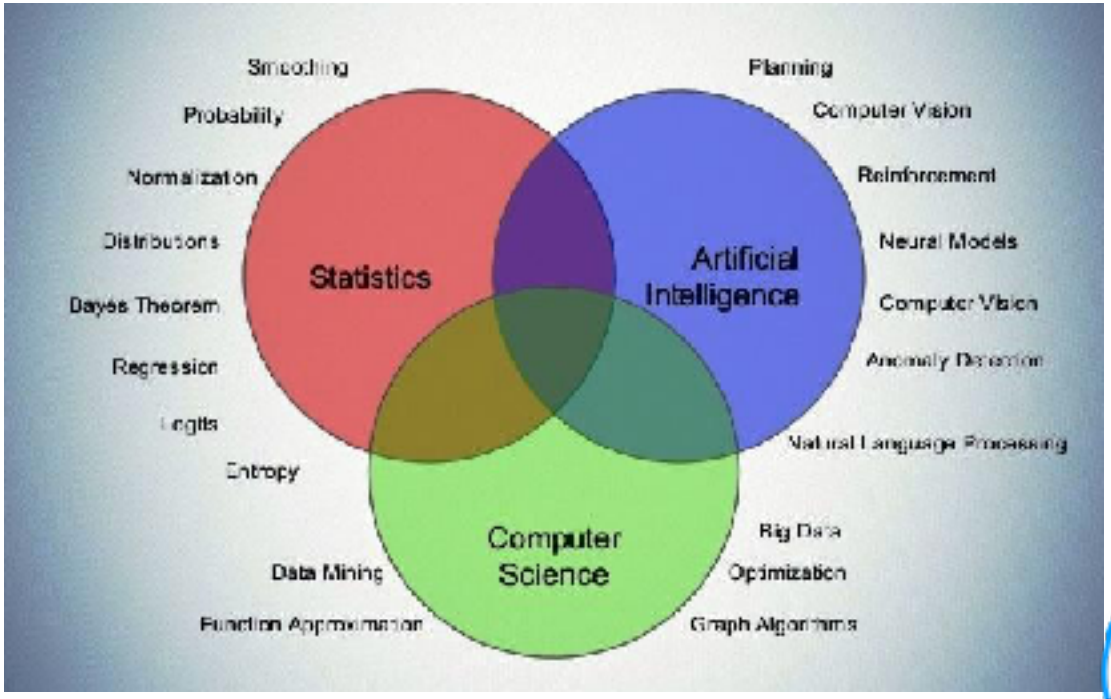
Figure 1. Hype Cycle for Emerging Technologies, 2016

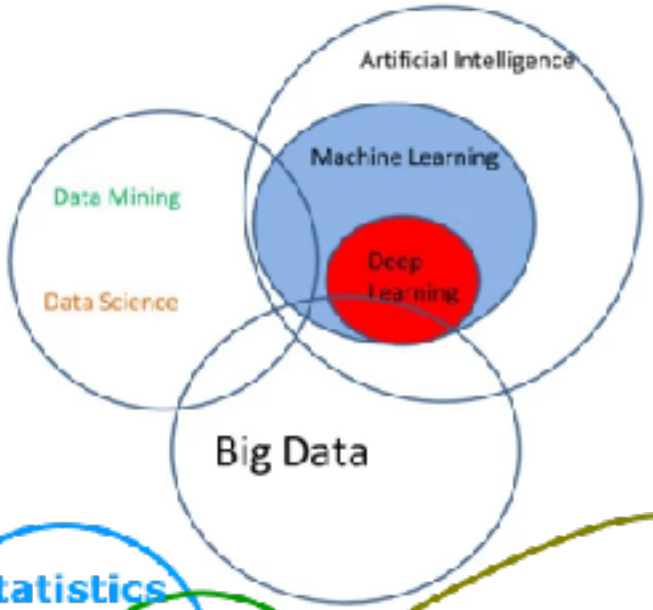


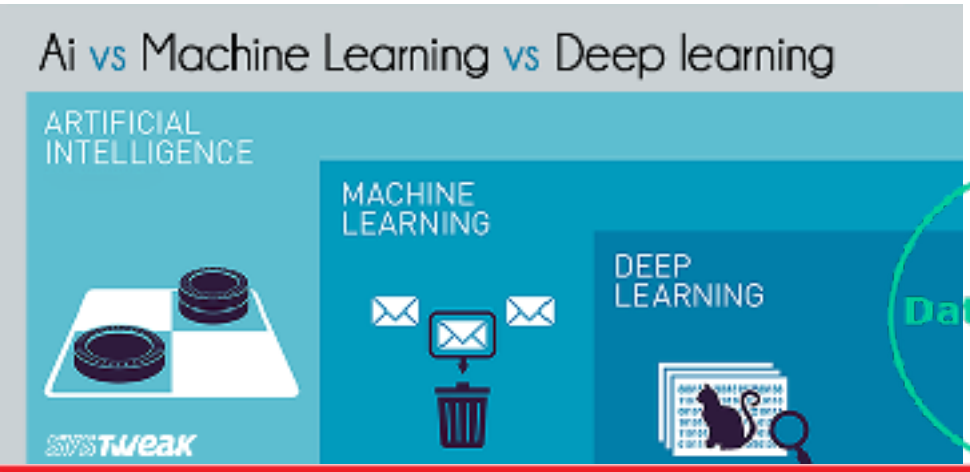

Source: Gartner (July 2016)

Source: Gartner (August 2016)

# Where Does Machine Learning Sit?



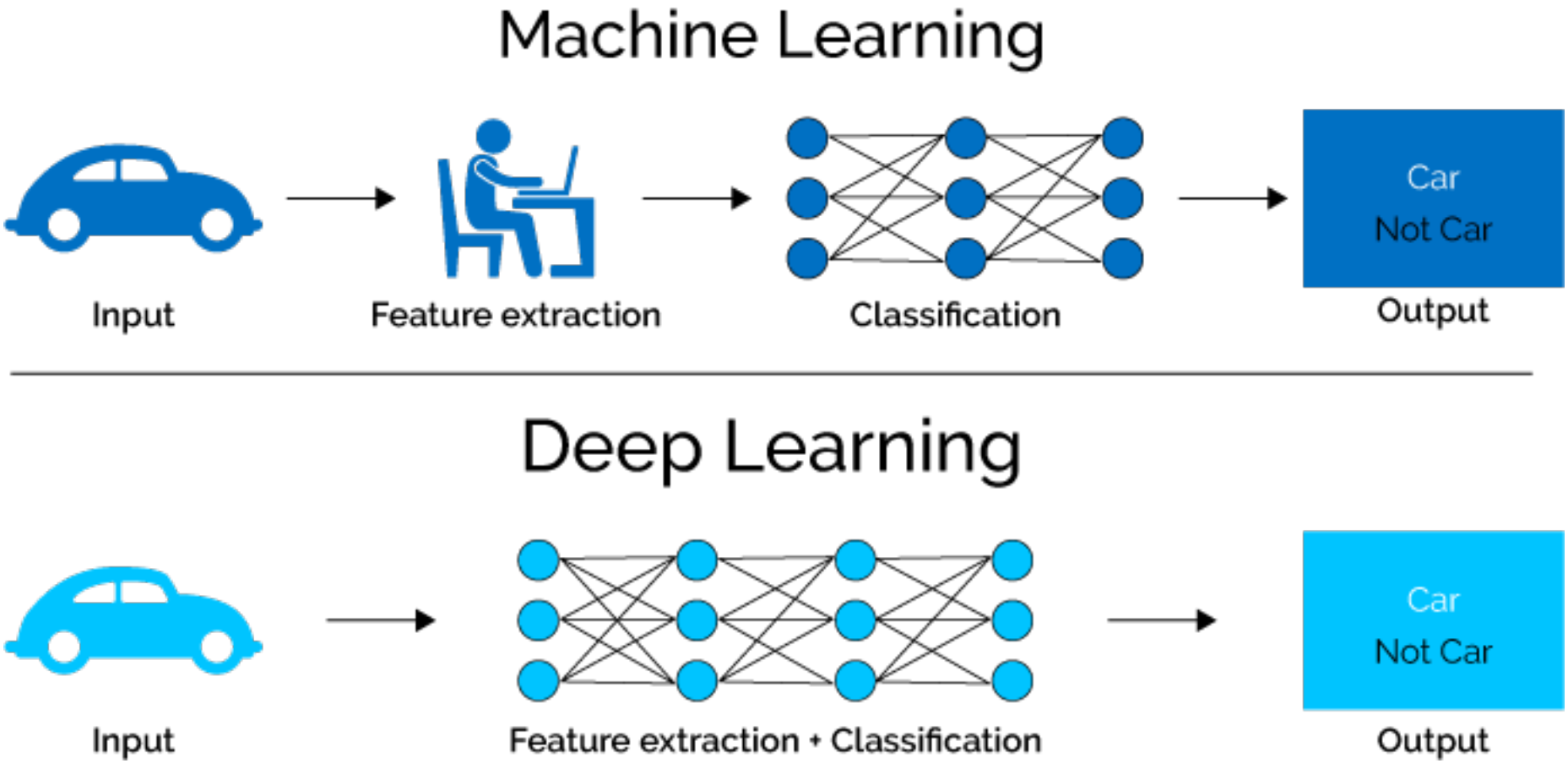




# Machine Learning or Statistics?

| | MACHINE LEARNERS | STATISTICIANS |
|---|---|---|
| Network/Graphs vs. Models | Network/Graphs to train and test data | Models to create predictive power |
| Weights vs. Parameters | Weights used to maximize accuracy scoring and hand tuning | Parameters used to interpret real-world phenomena – stress on magnitude |
| Confidence Interval | There is no notion of uncertainty | Capturing the variability and uncertainty of parameters |
| Assumptions | No prior assumption (we learn from the data) | Explicit a-priori assumptions |
| Distribution | Unknown a priori | A-priori well-defined distribution |
| Fit | Best fit to learning models (generalization) | Fit to the distribution |

# Machine Learning or Deep Learning?

– Much of the recent big news has involved complex neural nets:



# Deep Learning Goals

– Even these highly complex networks are still playing catch up.

## BRAIN VS AI

**Brain**

- Massive parallelism: 100 000 000 000 "cores"
- Extreme "bandwidth": 700 000 000 000 000 connections between "cores"
- ~10^18 "transistors"
- Asynchronous
- Adaptive hardware: neuroplasticity
- "Analog", but suitable for differential and integral computations

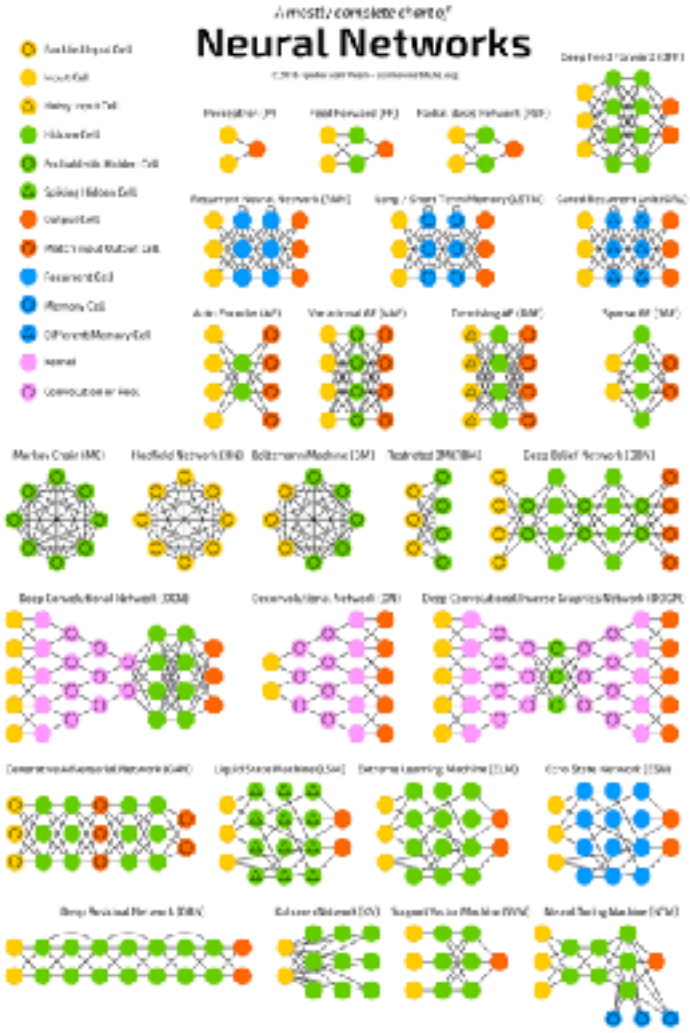
**Present day computer**

- Non-parallel architecture
- Low bandwidth
- ~10^9 transistors
- Synchronous (clock rate)
- Static hardware
- Digital, but linear computations

This need for extreme parallelism of simple operations opens up the possibilities of GPUs and TPUs (this is discussed later today).

# Deep Learning Goals

– Even these highly complex networks are still playing catch up.



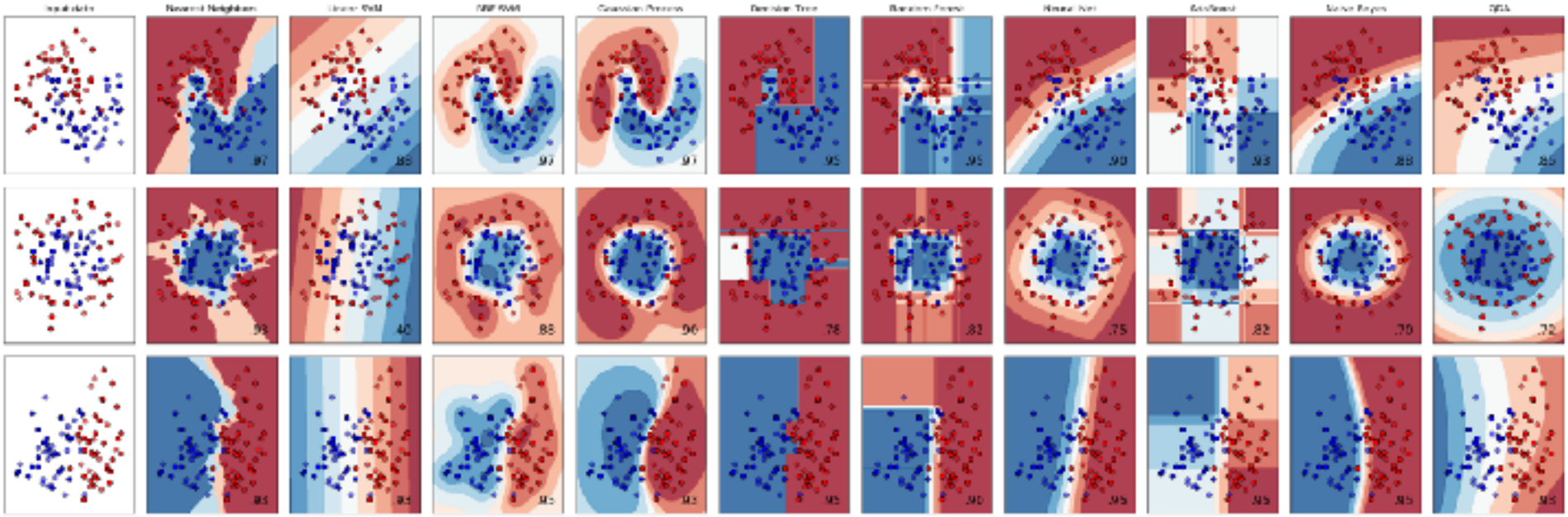

- There are a lot of options to take when creating and developing a modern neural network.
- The rule of thumb is to make them as simple as possible and no simpler (since learning is slow and expensive).
- This is where popular new frameworks like “Tensorflow” come in, allowing the construction of highly complicated NN graphs in relatively simple language.
- It understands about general useful classes of network, and can build complicated graphs quickly for rapid prototyping.

Some Classic Practical Examples in 1D and 2D

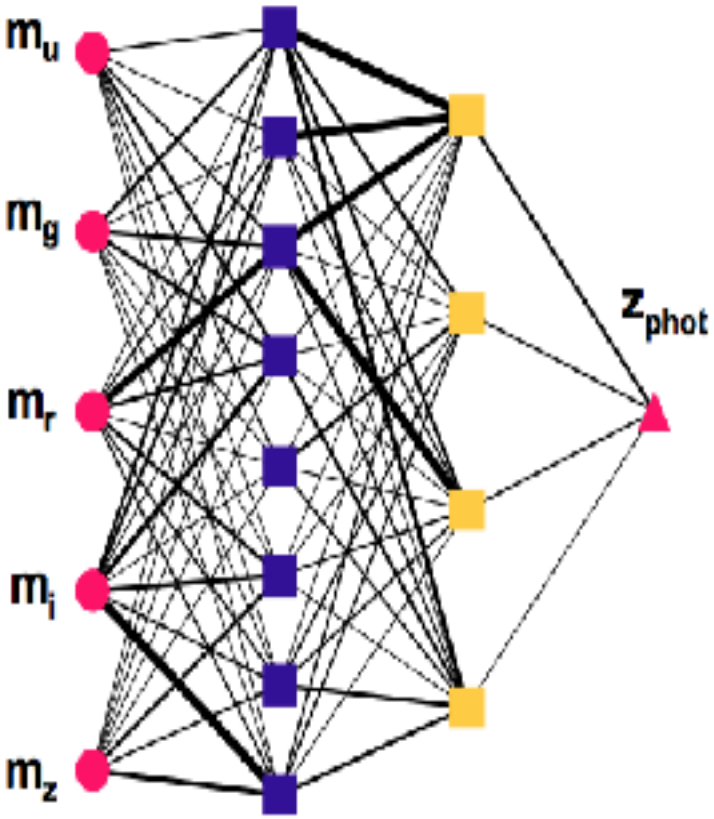
# Machine Learning in Practice

# Simple Discrete Classification Schemes

- There are a lot of simple classification schemes available freely in e.g. Python and R.
- Regularly used in astronomy are Gaussian Processes / Random Forests / LDA / QDA / Decision Trees.
- These tend to be used as black-boxes, but they are easily trained and verified, so a very low cost of entry.

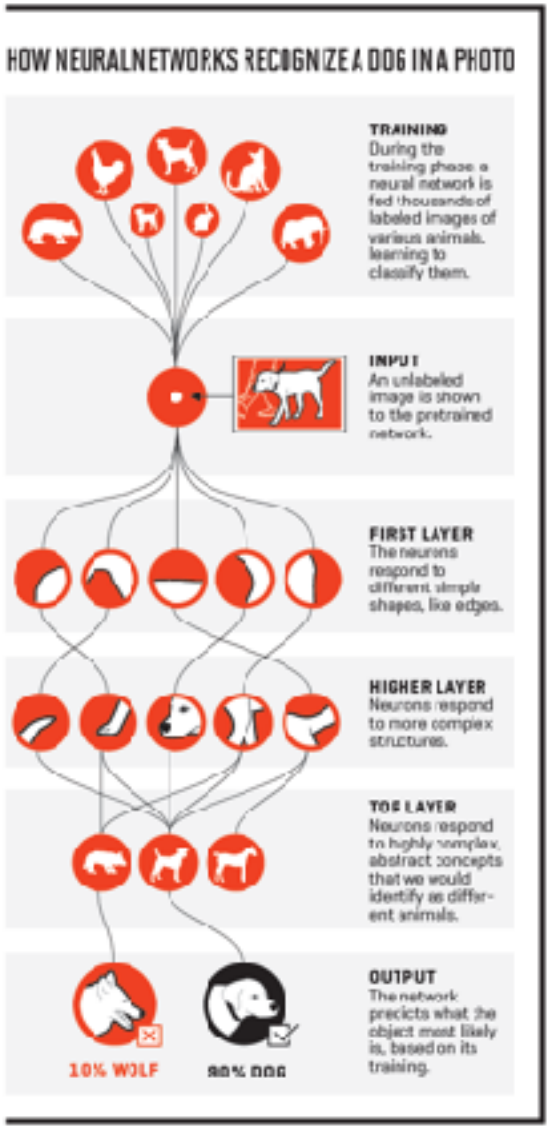


# 1D Feature Based Neural Networks (photo-z)




- ANNz and ANNz2 are classic application of 1D neutral networks.
- ANNz uses a multi layered perceptron (see, it turned out to be useful) to create a fairly simple and quick to train network.
- Note the inputs are magnitudes.
- Selecting the inputs in this manner is feasible for 1D problems- it is very obvious what are the useful properties to measure for an unresolved high-z object. Magnitude is all you can extract.
- This process is called “feature extraction”, but it breaks down in 2D since it is very hard to choose the right/best features.

# Deep Learning in 2D (image recognition)




- Image recognition has recently been successfully tackled through the use of ultra deep “convolution” networks that naturally extract features themselves.
- Many of the lowest level convolvers are actually much like traditional shapes used to sharpen and smooth images in e.g. Photoshop.
- Building up local associations of edges and corners allows more complex sub images to be formed.
- The next layer assembles these sub images etc etc.
- Typically one network per classification class (not always true though).
- This type of neural network is usually highlighted as an example of “deep learning”. More expensive computational, but less human intervention.

# An Abstract World View

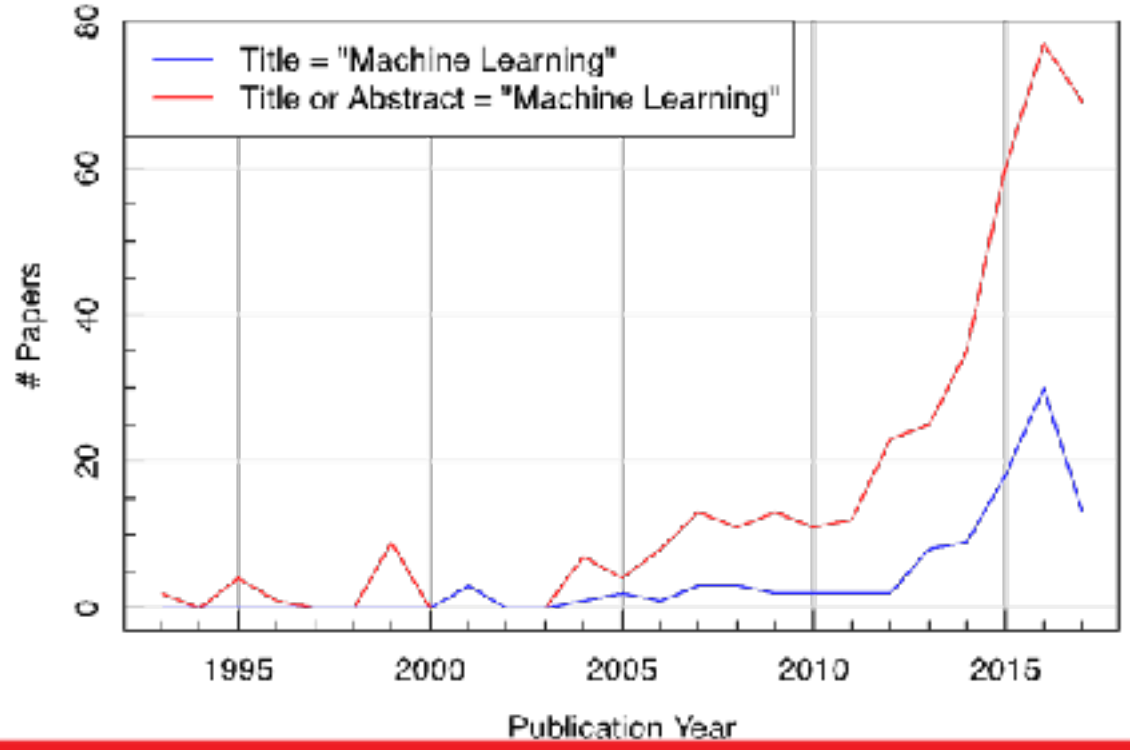
# Google's Deep Dream / Nightmare

Uptake in astronomy, and prospects for the future

# The State of Play

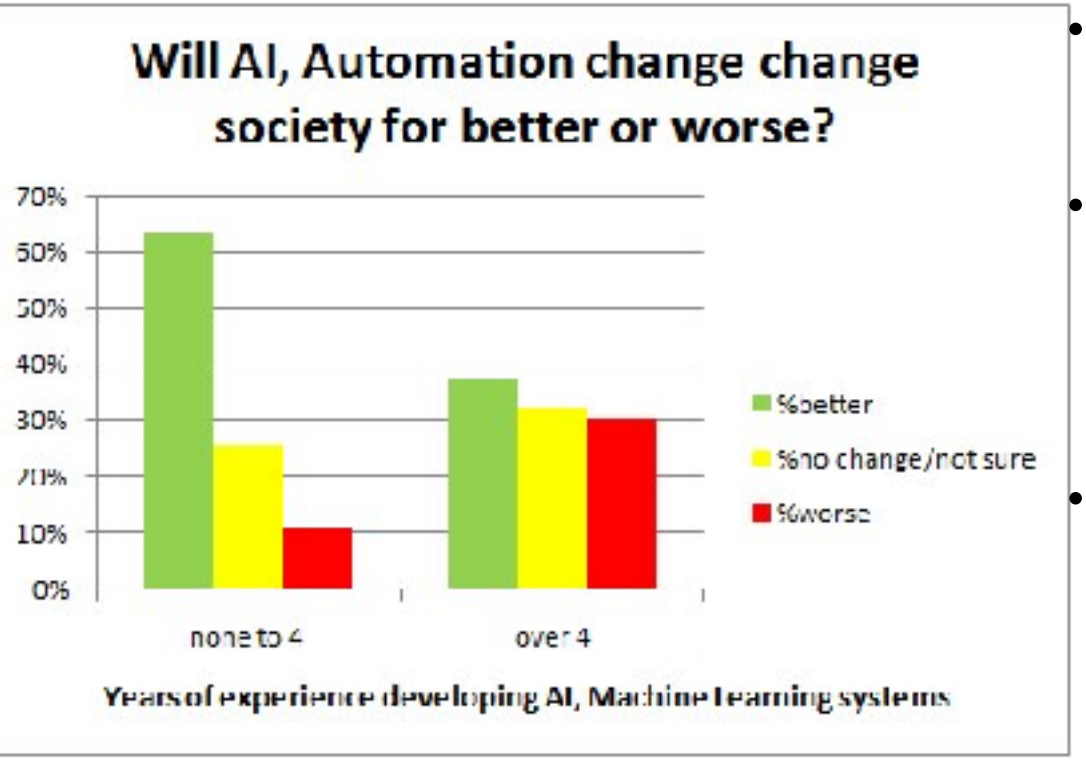
# Machine Learning Use In Astronomy

- Astronomers have been quick to pick up on the new wave of machine learning, shortly after the big mid 2000s rebrand.
- 380 total papers and 4300 citations for machine learning.
- Most cited is modestly ~100.
- Many of the simpler tools are used routinely and without fanfare.



# Future Prospects of Machine Learning

- "AI" still has a bad reputation, not helped by folk like Elon Musk who concentrate much more on the sci-fi side of the field.
- Academic scale machine learning has more positive prospects given the huge amount of open source support being provided by Google, Facebook and Microsoft.



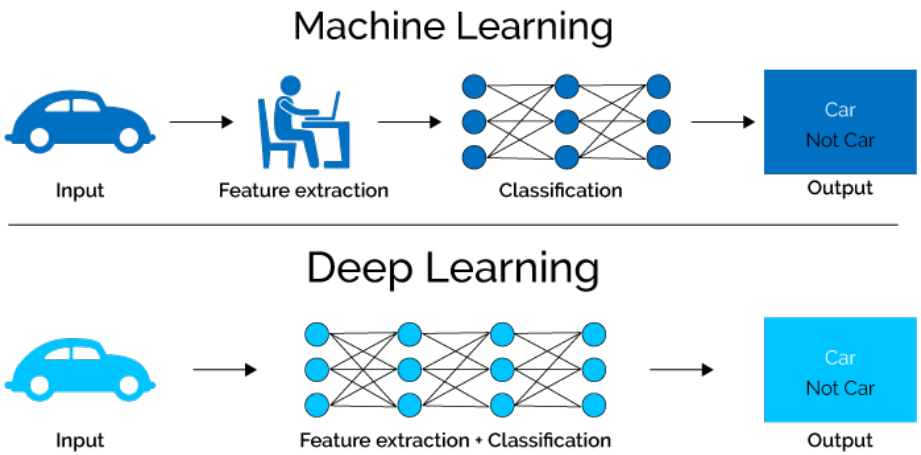

- For certain classes of problem the bar is lower now, but still high.
- Scientists tend to get alarmed by the lack of inference (parameter errors etc) and confidence intervals. Should we care?
- Tend to use them conservatively, so vanishingly few examples of original insight via machine learning in astronomy. So far!

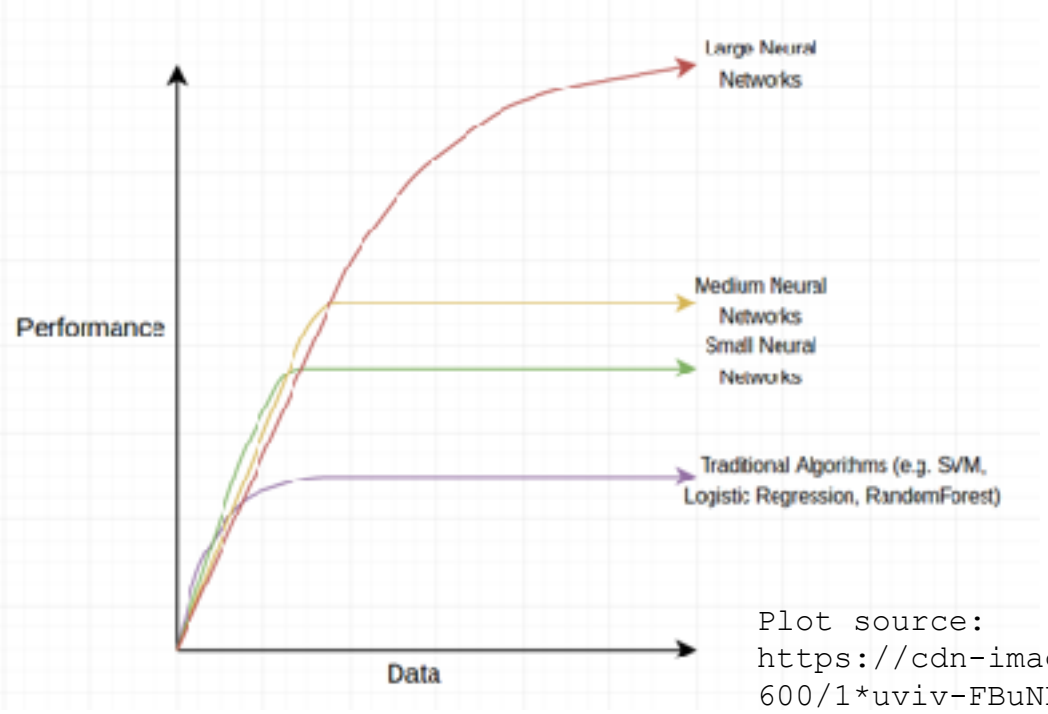

International Centre for Radio Astronomy Research

# Image and Convolutional Neural Networks



# Motivation of DL and CNN



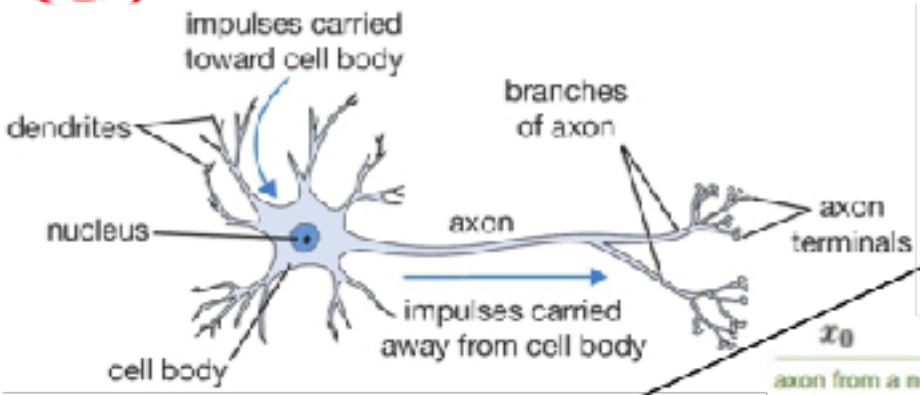



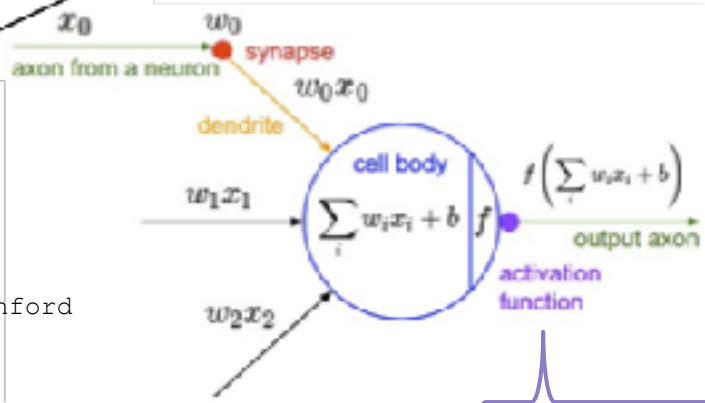

- End-to-end pipeline
- Little feature engineering
- Fully data-driven

- Data hungry
- Data sensitive
- Performance can be improved continuously as long as we keep feeding more data

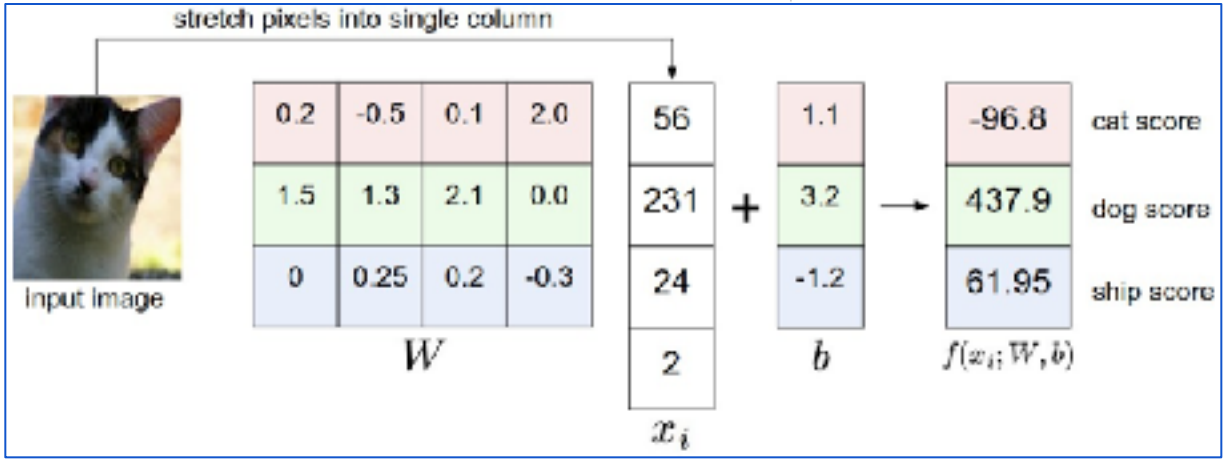
Plot source:
https://cdn-images-1.medium.com/max/600/1*uviv-FBuNKSbOiGwVSyvKw.png

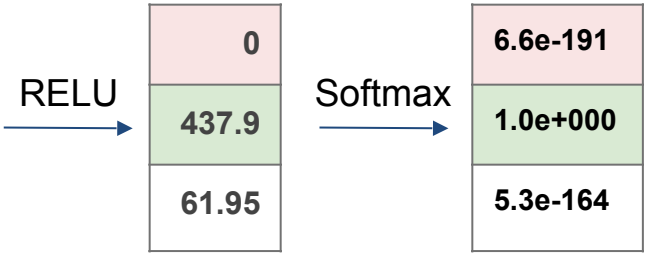
# Brain-inspired image classifier




Credit: (Li, Karpathy, Johnson 2016) cs231 stanford



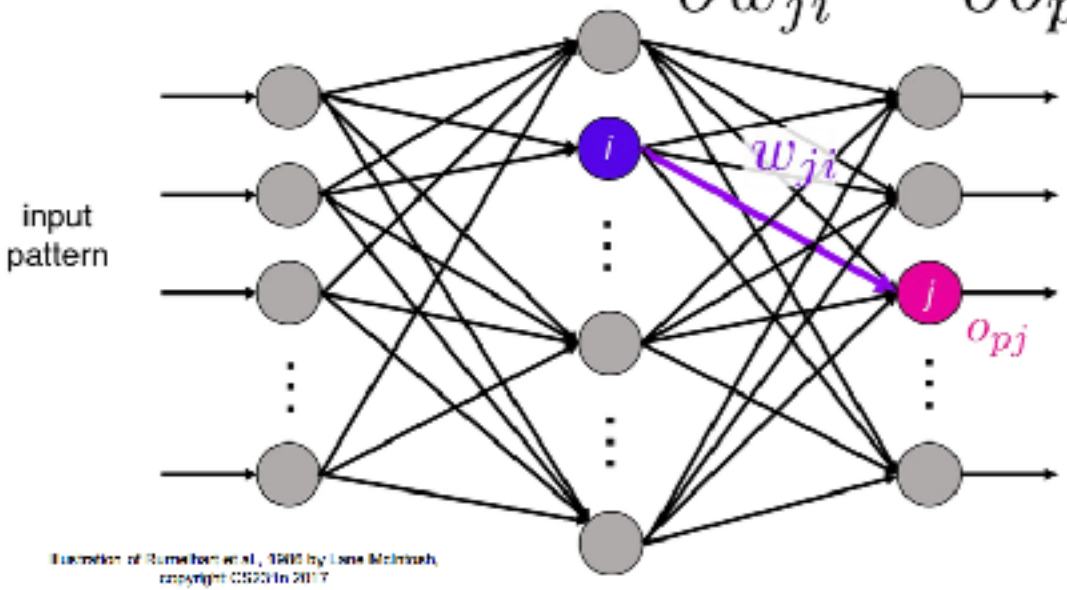



# The “Aha” moment of training NN

A bit of history...

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

recognizable math




Rumelhart et al., 1986: First time back-propagation became popular

Learning representations by back-propagating errors, Nature 323, 533–536 (09 October 1986)
David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

# A naive example of back prop

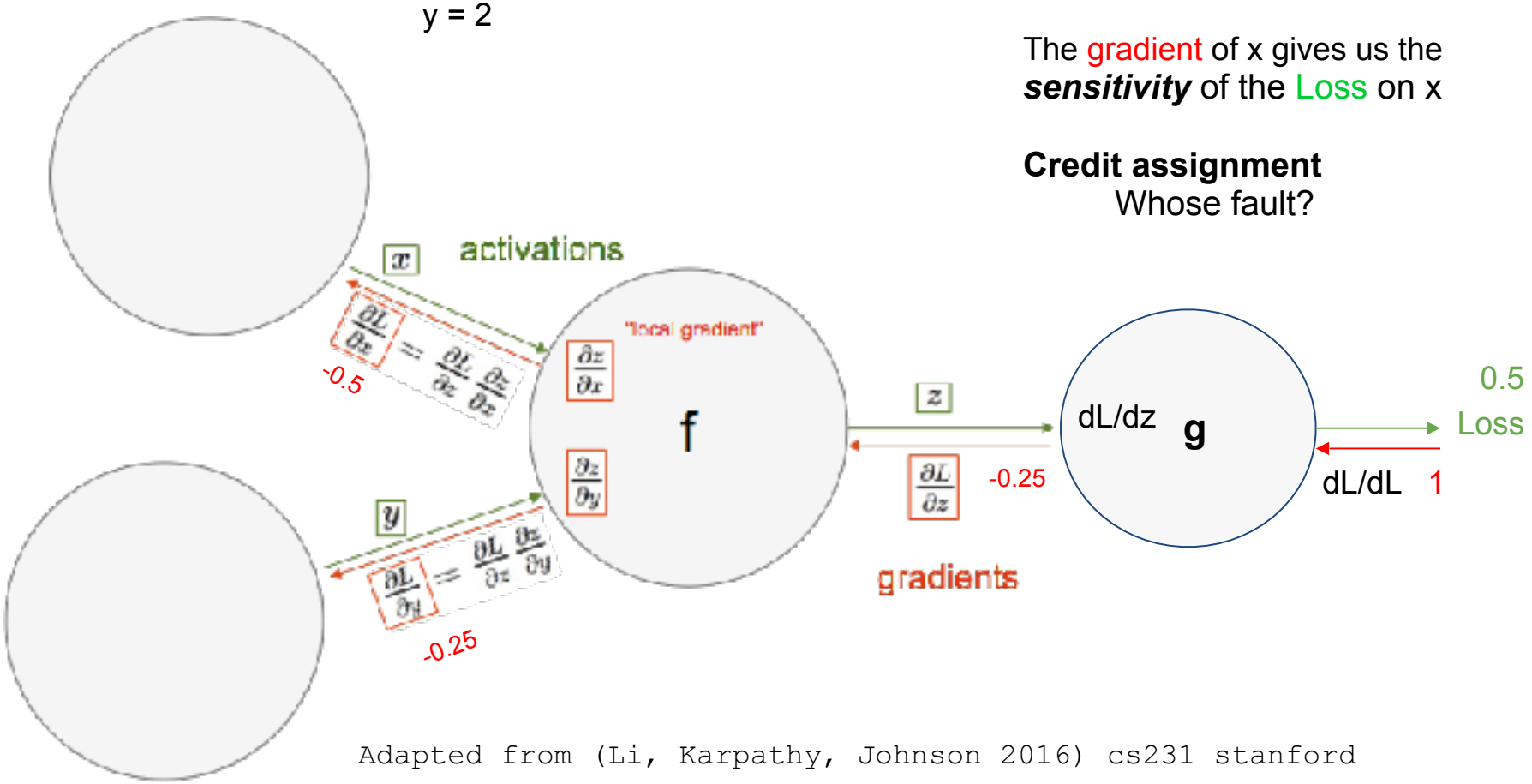
L = g(z) = 1 / z
z = f(x, y) = x * y

x = 1
y = 2

How can we change x and y in order to reduce the loss?

The gradient of x gives us the ***sensitivity*** of the Loss on x

**Credit assignment**
Whose fault?

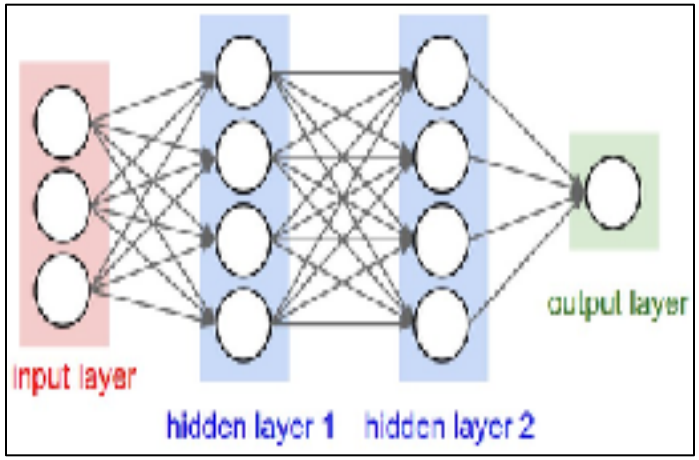



Adapted from (Li, Karpathy, Johnson 2016) cs231 stanford

# Mini-batch SGD

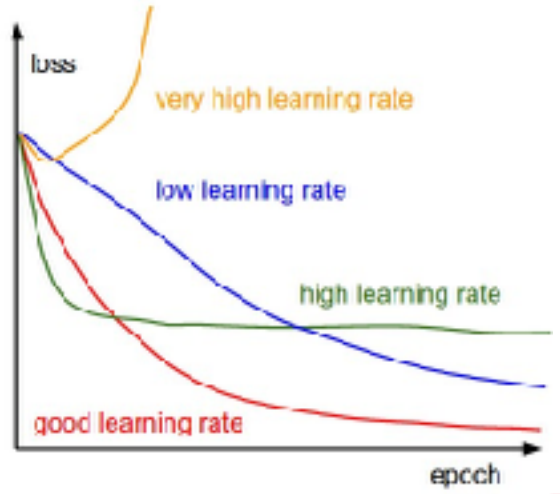
Loop: (THE **compute engine** of the entire Deep Learning industry)

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient




Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{1}{B} \sum_{t'=Bt+1}^{B(t+1)} \frac{\partial L(z_{t'}, \theta)}{\partial \theta}.$$



# ConvNet inspiration - Receptive field mapping

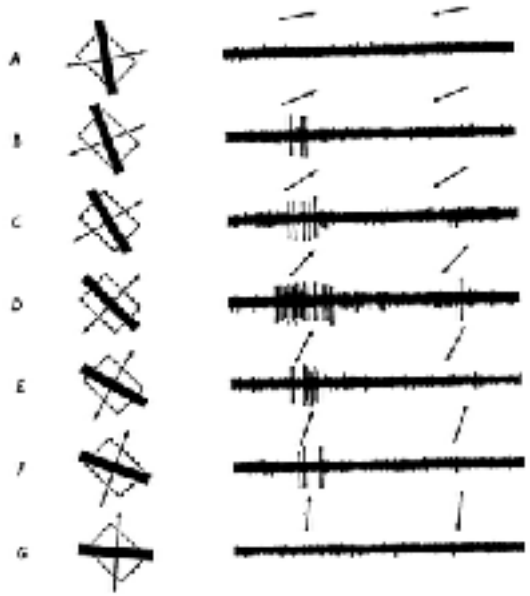
**Hubel & Wiesel 1959, 1962, 1968**

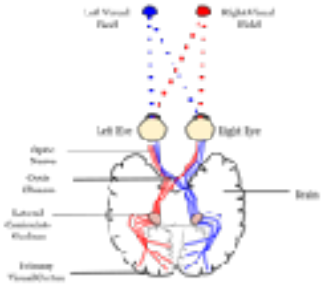
Responses of single cells along the visual pathway

**Simple** cell (Cat)

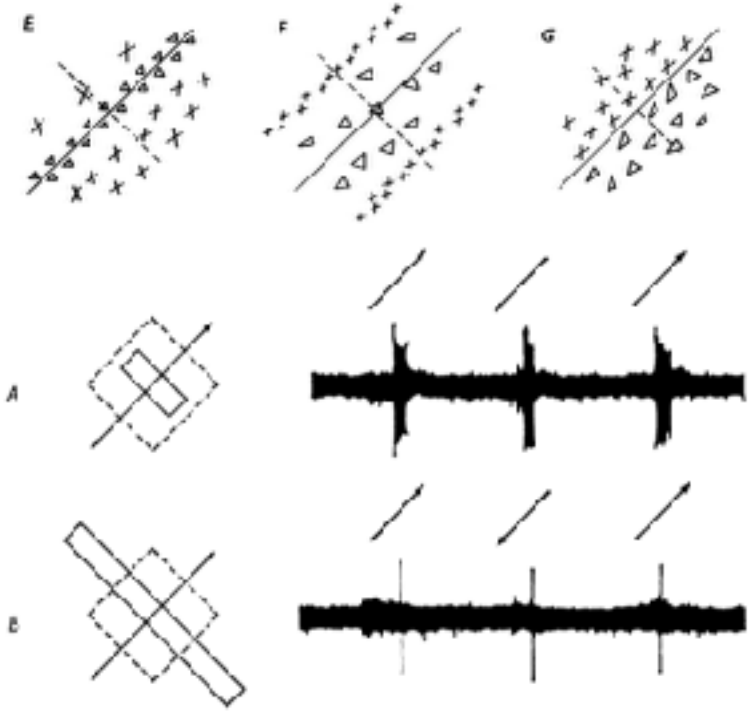
An anatomically-wired **Hierarchical System**, in which

1. Geniculate cell converge on simple cortex cells
2. Simple cells converge upon complex cells
3. Complex on hypercomplex





**Complex** cell (Monkey left eye Receptive Field)



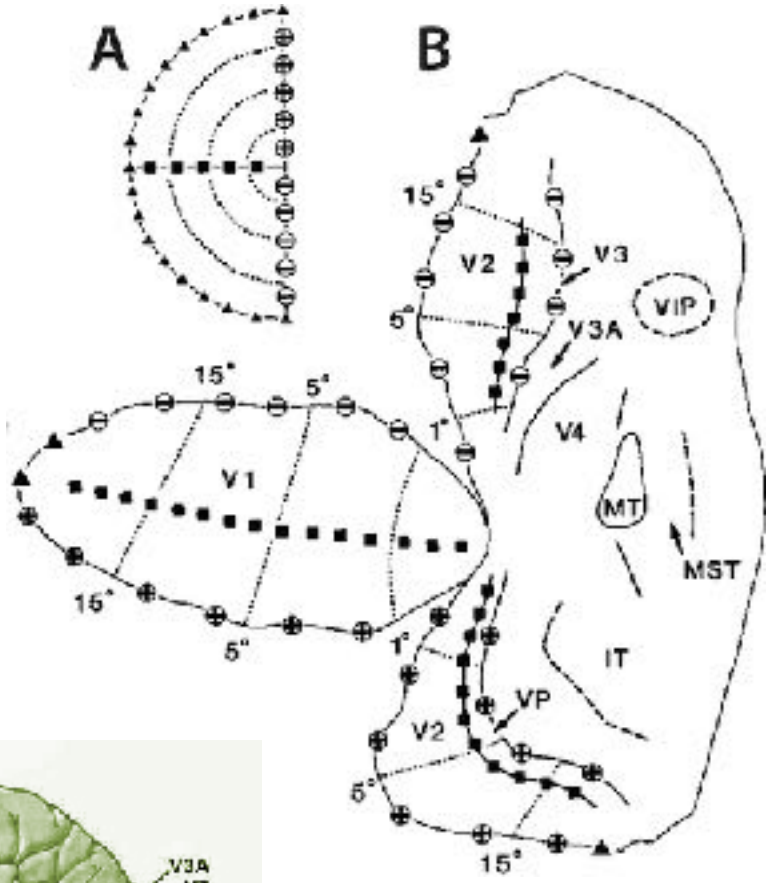
**HyperComplex** cell (Monkey left eye Receptive Field)

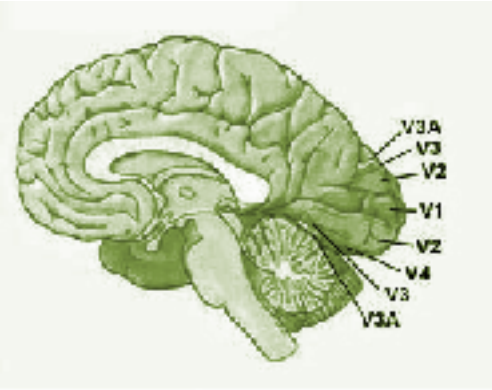
# ConvNet inspiration - Topographical mapping

**Topographical mapping in the cortex:**
(Patel, Michael, Snyder 2014)

1. nearby cells in cortex represent nearby regions in the visual field

1. one map completely fills each cortical area, so that topographic map boundaries coincide with areal boundaries



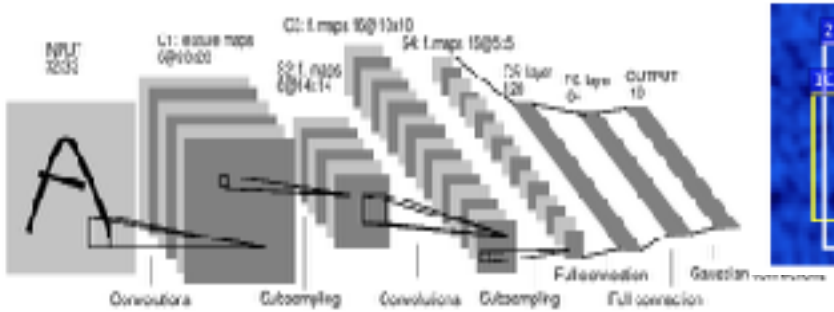



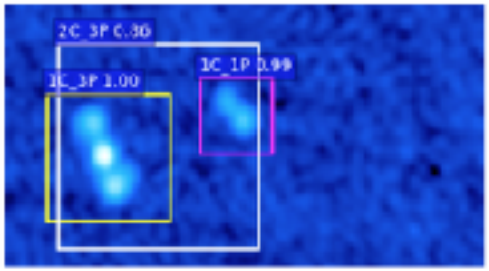

Maunsell JHR, van Essen DC. The connections of the middle temporal visual area (MT) and their relationship to a **cortical hierarchy in the macaque monkey**. J Neurosci. 1983;3:2563–2586

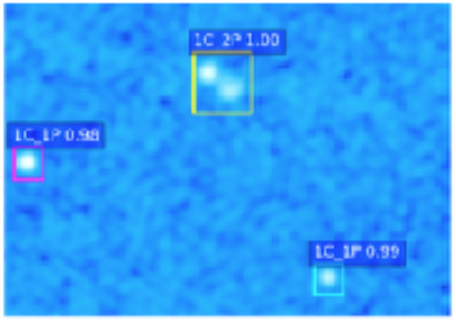
# Today ConvNet is used everywhere

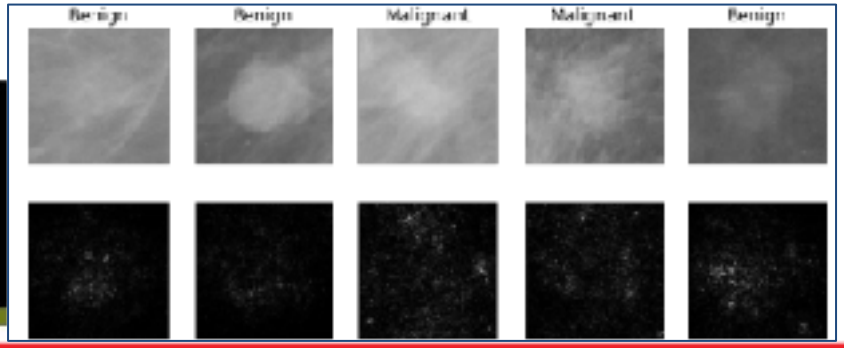
[LeCun et al., 1998]

Let's completely forget about the brain stuff from this point forward

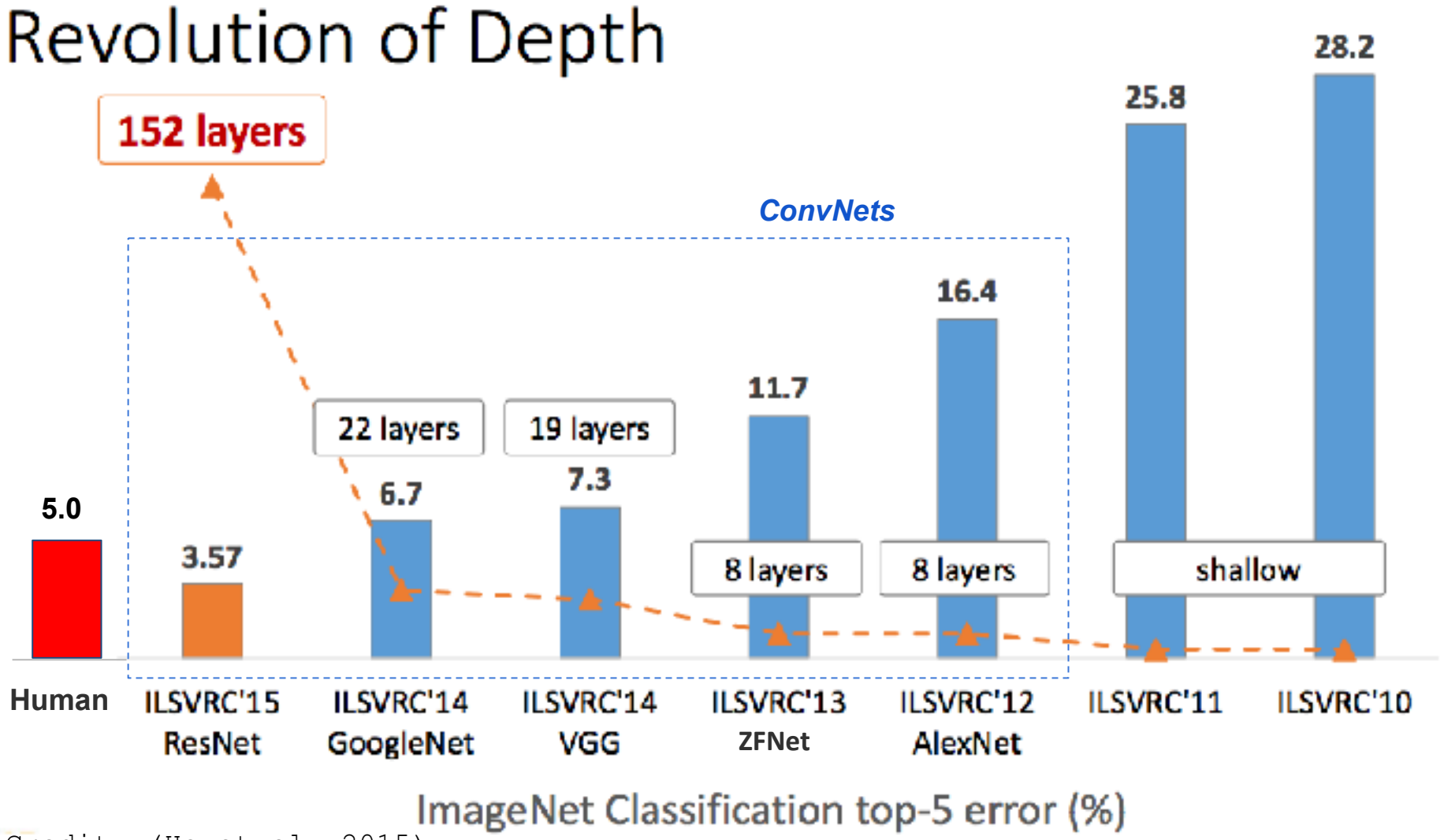







# Deep ConvNets beat human-level recognition




Credit: (He et al. 2015)
http://kaiminghe.com/ilsvrc15/ilsvrc2015_deep_residual_learning_kaiminghe.pdf

# Recent ConvNet development

AlexNet in 2012 (Krizhevsky et al., 2012), the annual ImageNet classification challenge (ILSVRC) has been dominated by progressively **deeper networks** with

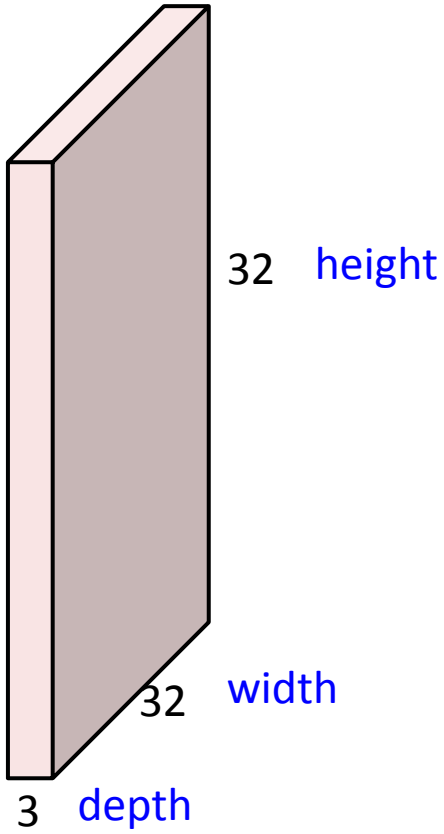
**smaller kernels** (Szegedy et al., 2015; Simonyan & Zisserman, 2014b).

Recent solutions to the issues of *vanishing and exploding gradients* (Glorot & Bengio, 2010) have allowed these networks to extend even **deeper**,

with the ILSVRC15 winner (ResNet (He et al., 2016)) being 8-fold deeper than VGG.

# Convolution Layer

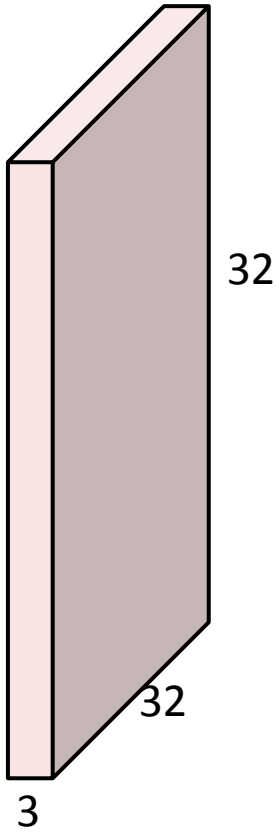
32x32x3 image



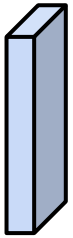

Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# Convolution Layer

32x32x3 image




5x5x3 filter



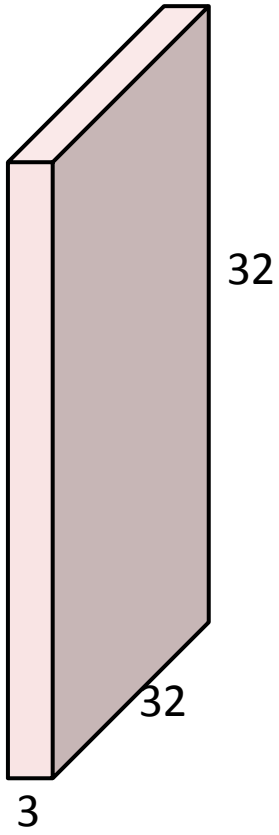
**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”

Credit: (Li, Karpathy, Johnson 2016) cs231 stanford

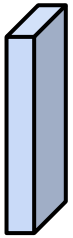
# Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



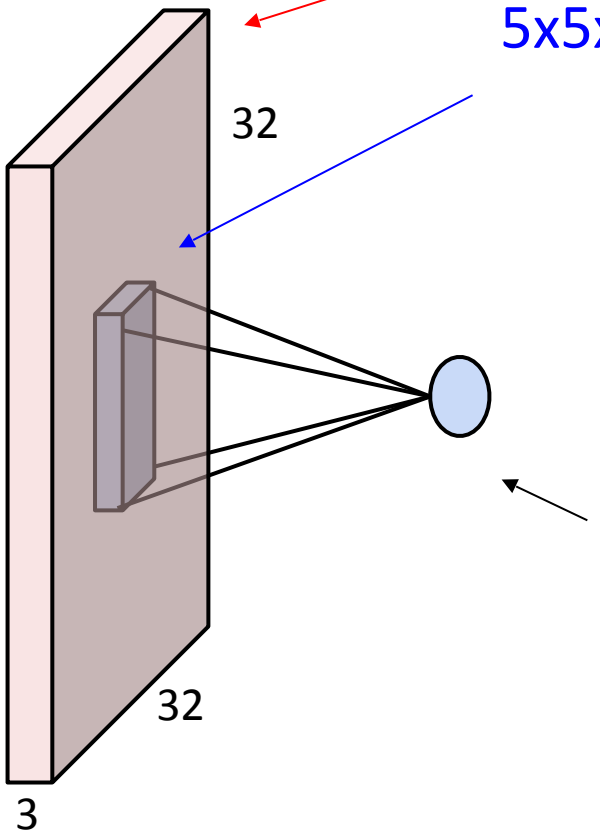

5x5x3 filter



**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”

Credit: (Li, Karpathy, Johnson 2016) cs231 stanford

# Convolution Layer

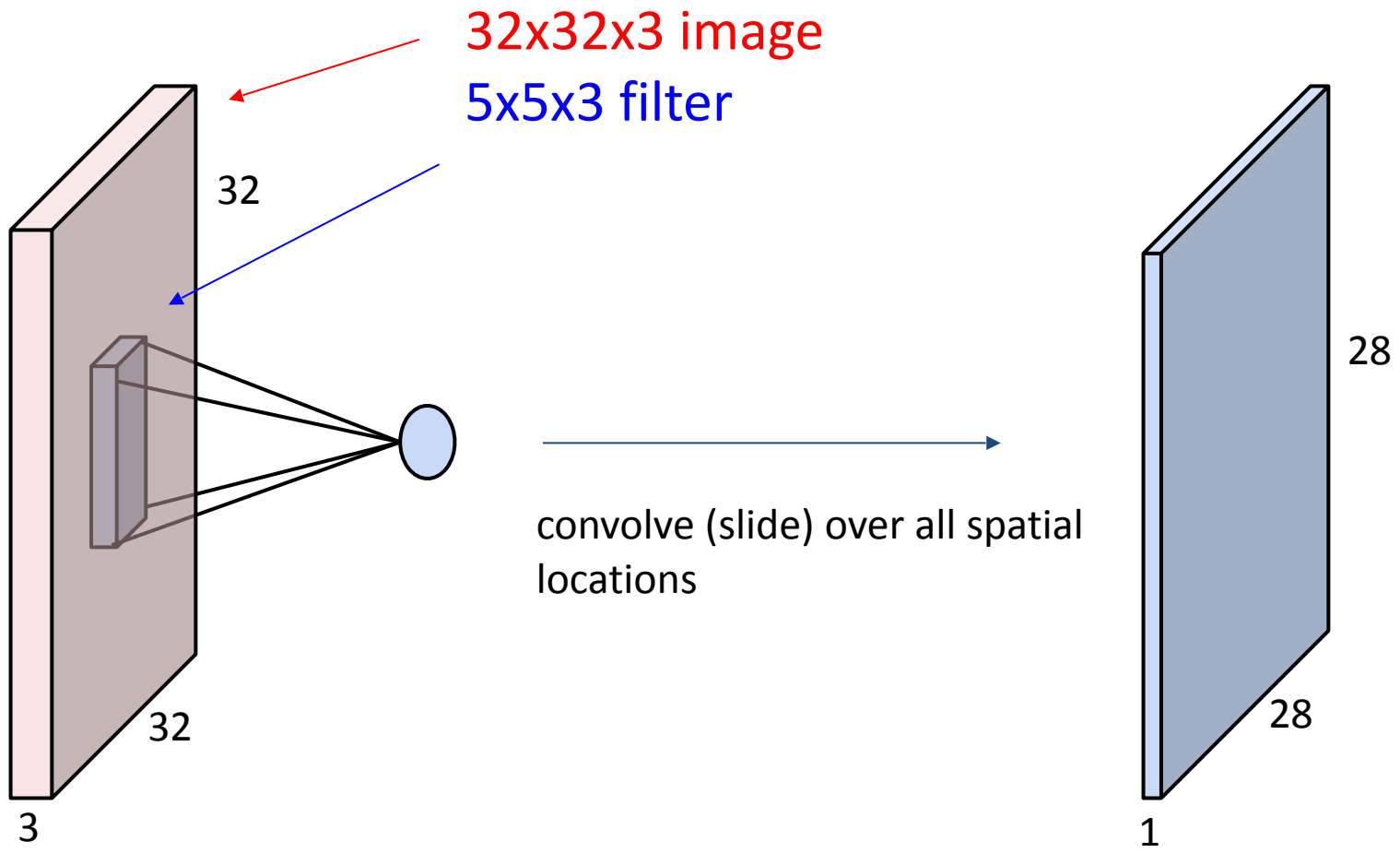



**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

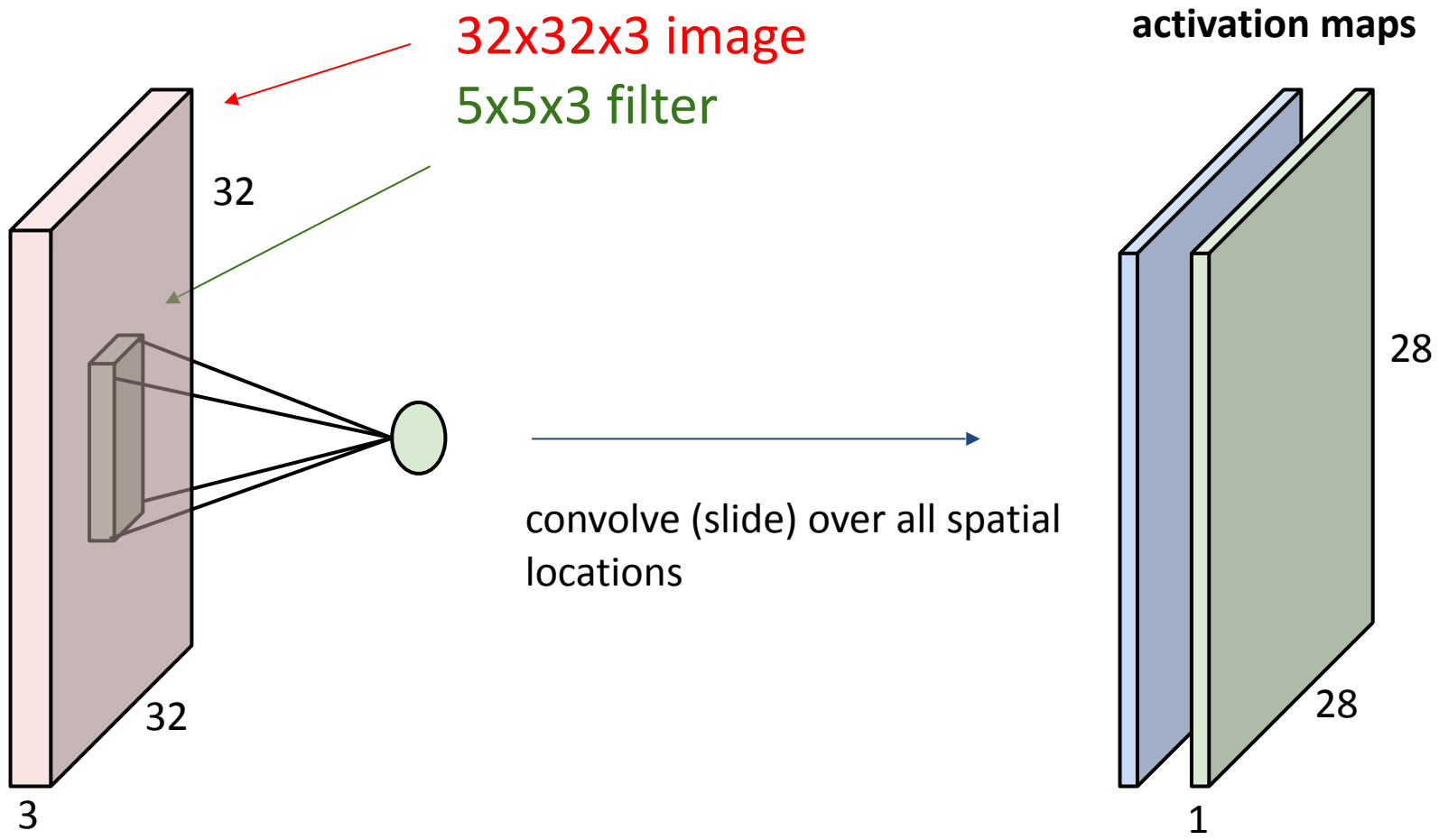
Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# Convolution Layer




Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

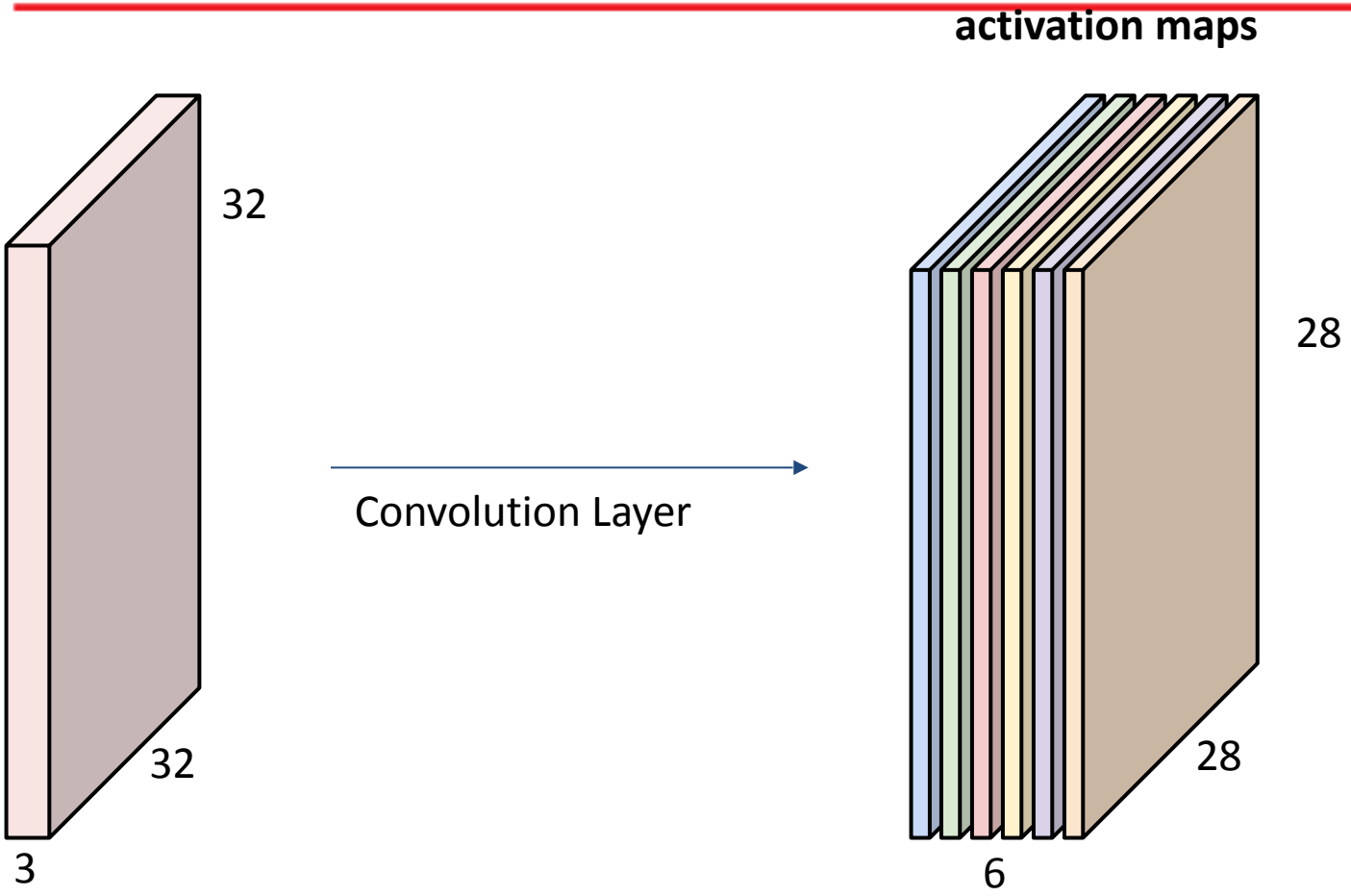
# Convolution Layer




consider a second, green filter

Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

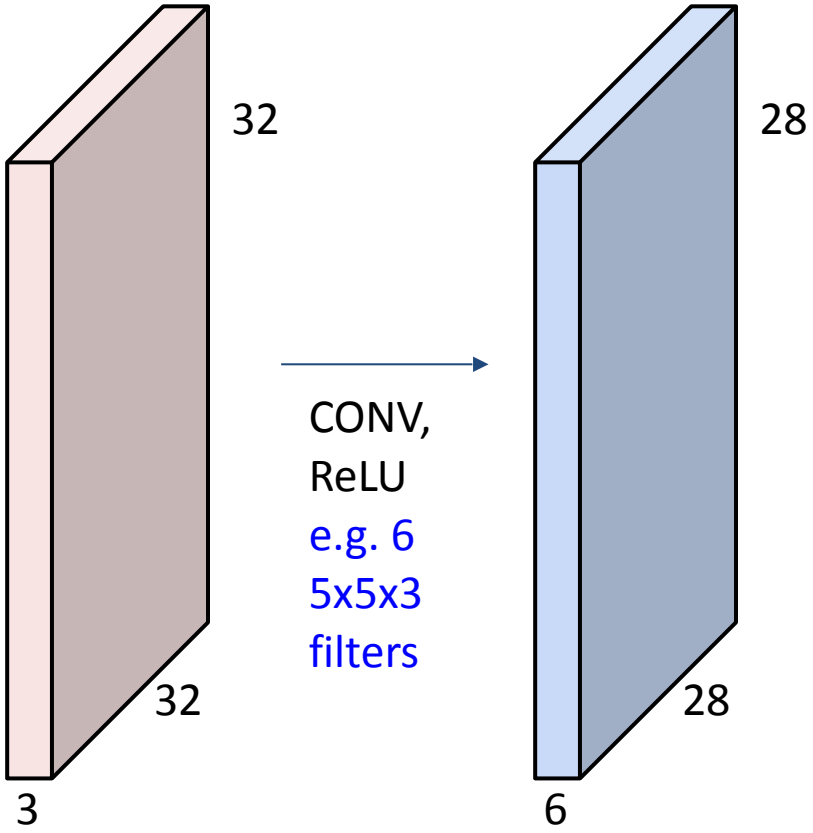
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:




We stack these up to get a "new image" of size 28x28x6!

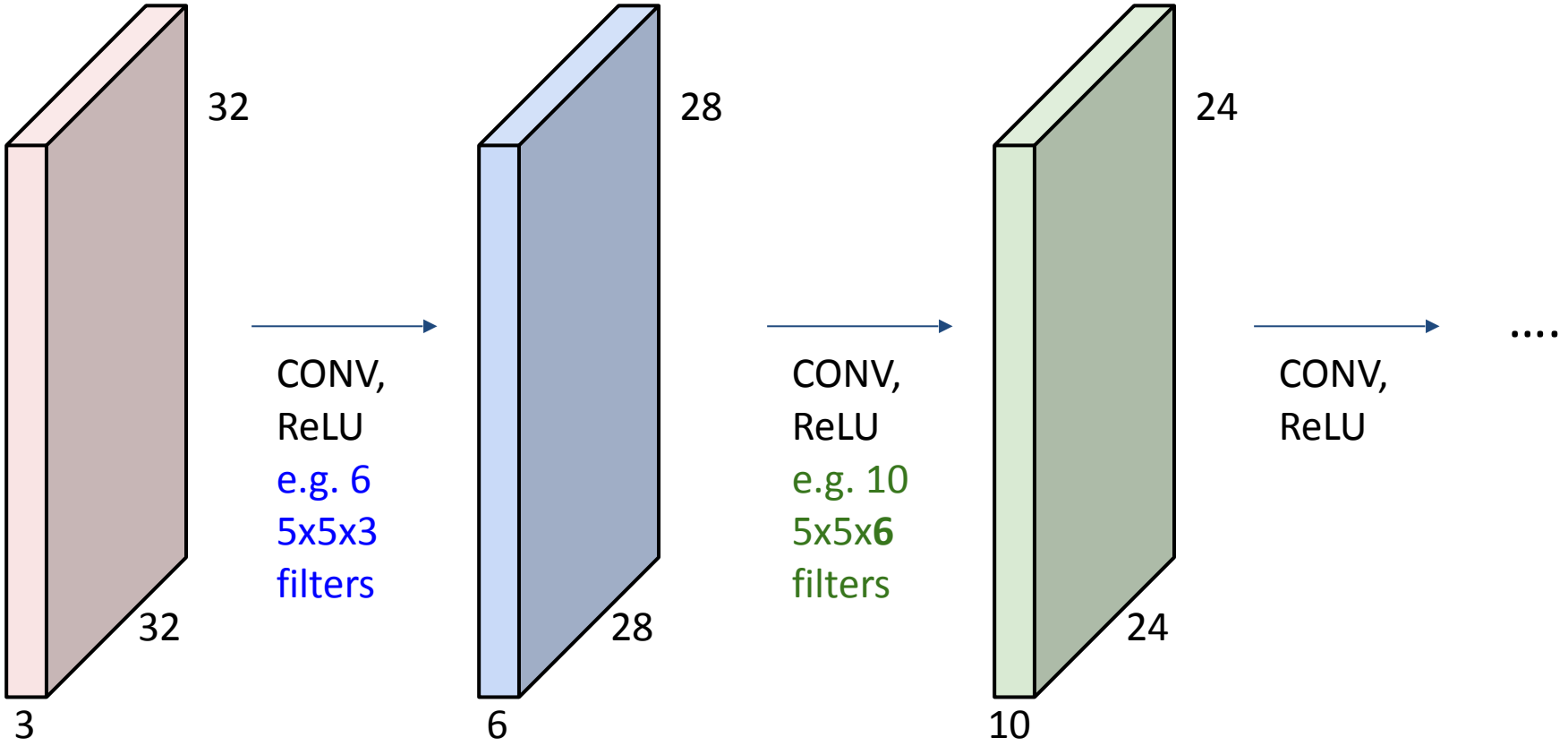
Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions



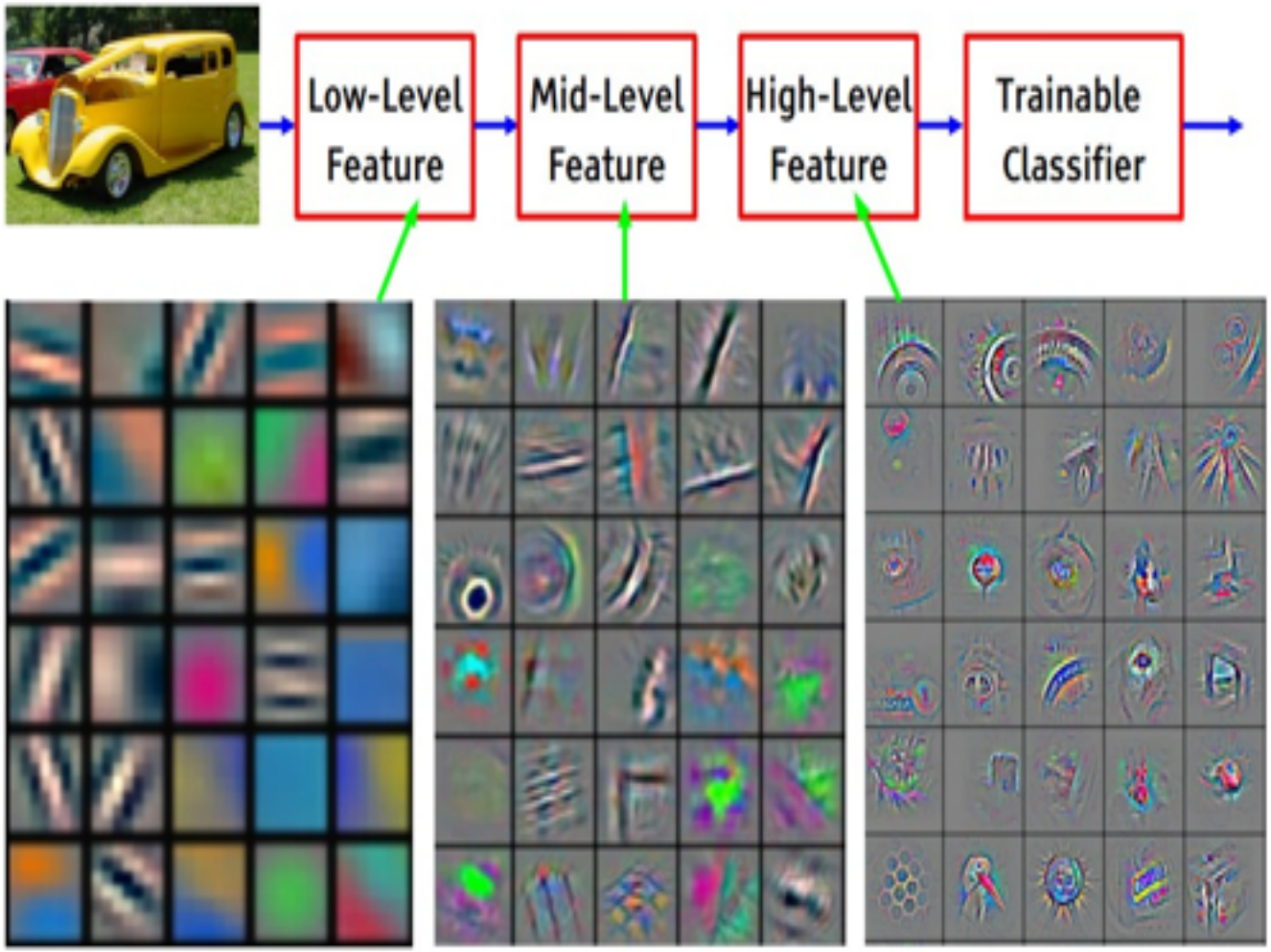

Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions




Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

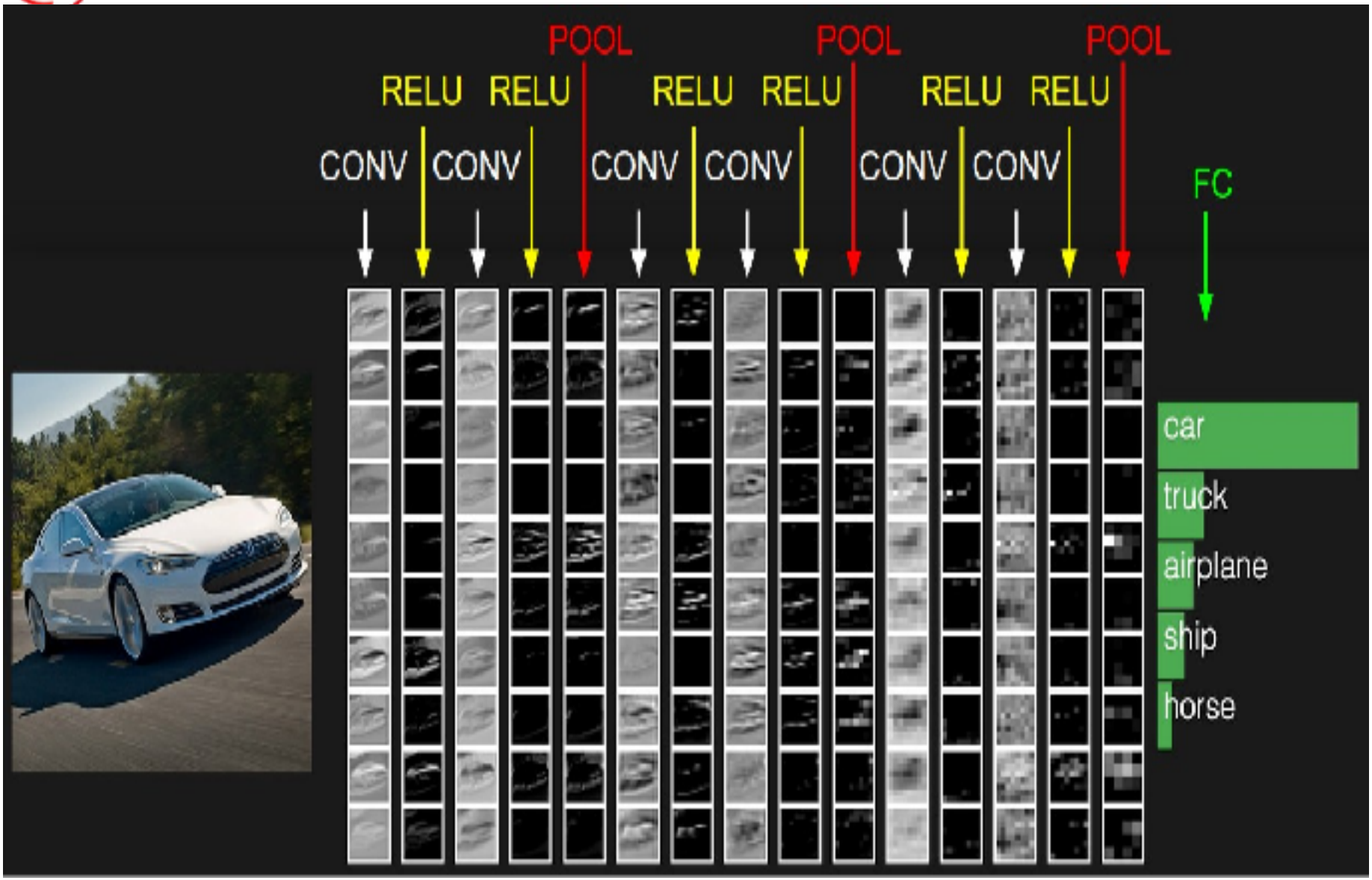
*[From recent Yann LeCun slides]*



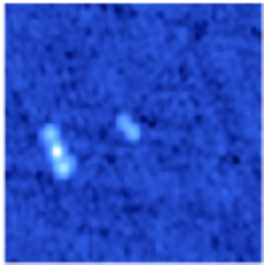

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

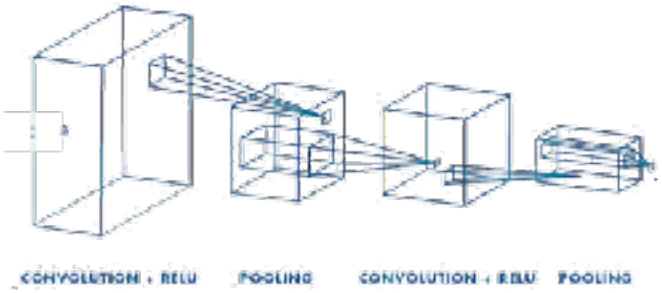
preview:

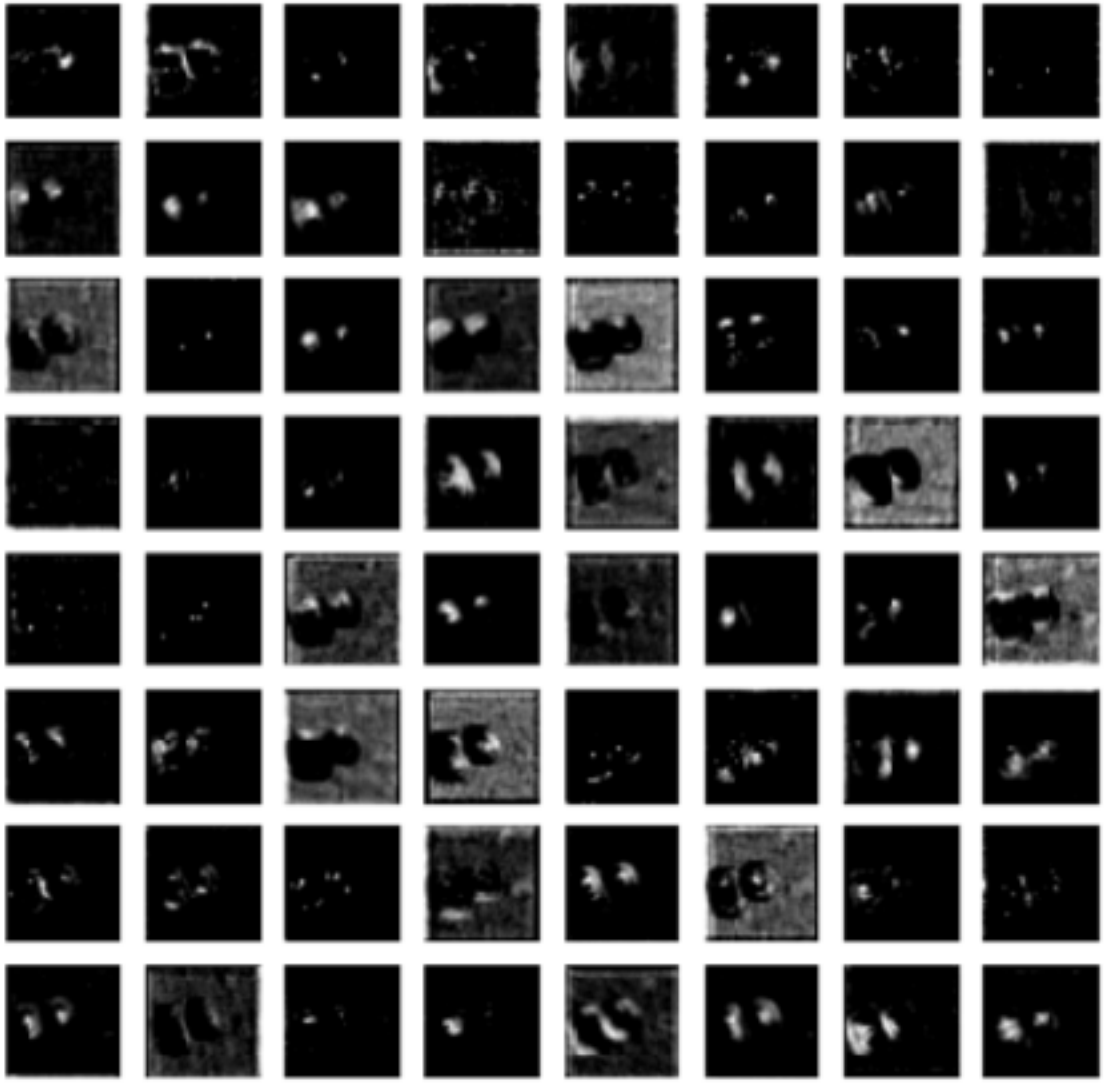
Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford



# R-CNN for Radio Galaxy Zoo (Chen Wu & Ivy Wong, 2017)

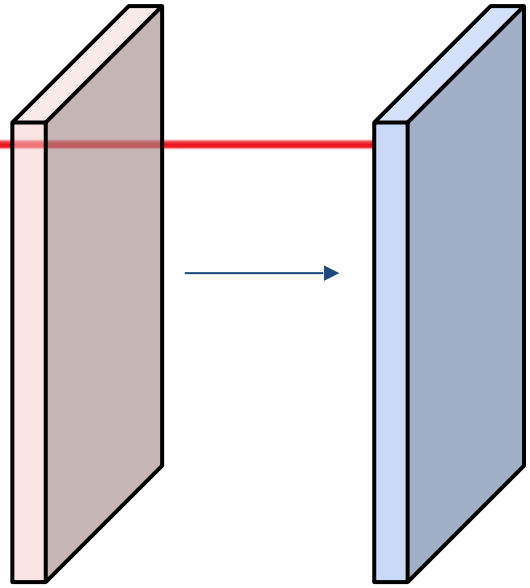




"high-level" feature maps

# Quiz time:

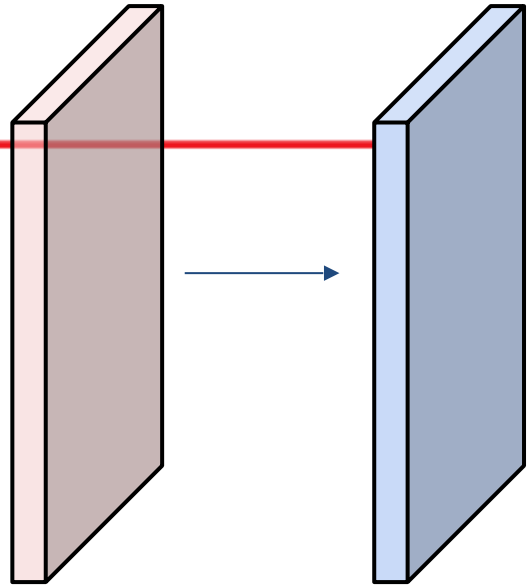


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of learnable parameters in this layer?

Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# Quiz time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

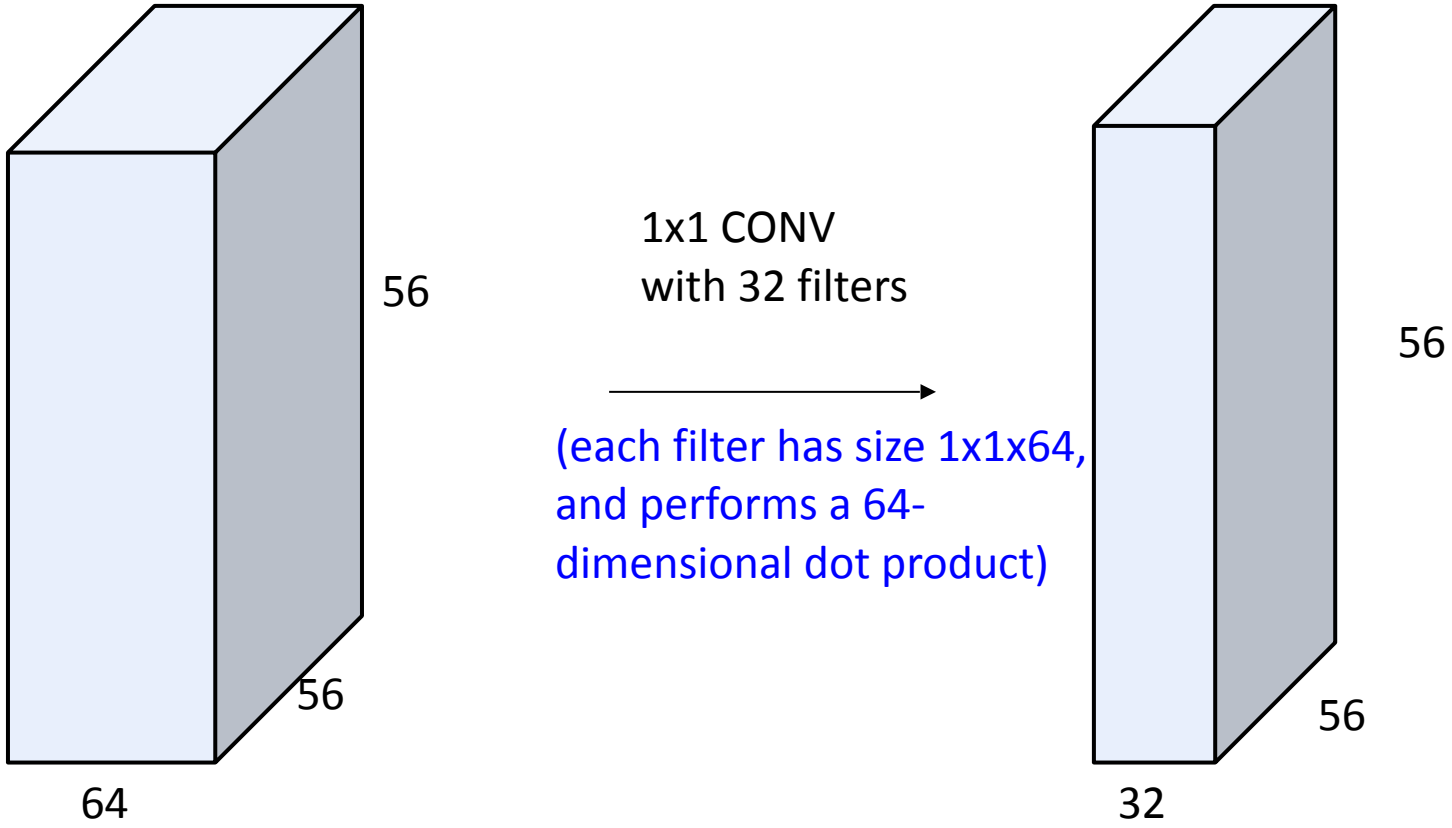
Number of learnable parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

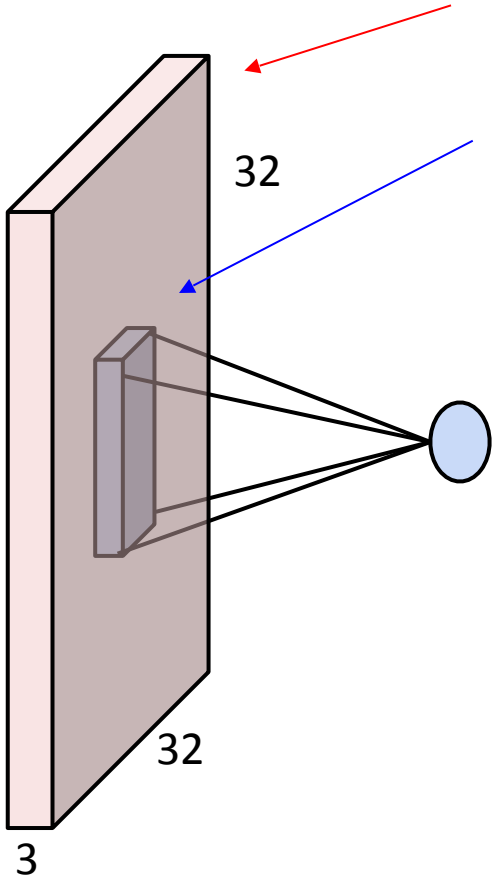
Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# (btw, 1x1 convolution layers make perfect sense)




Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

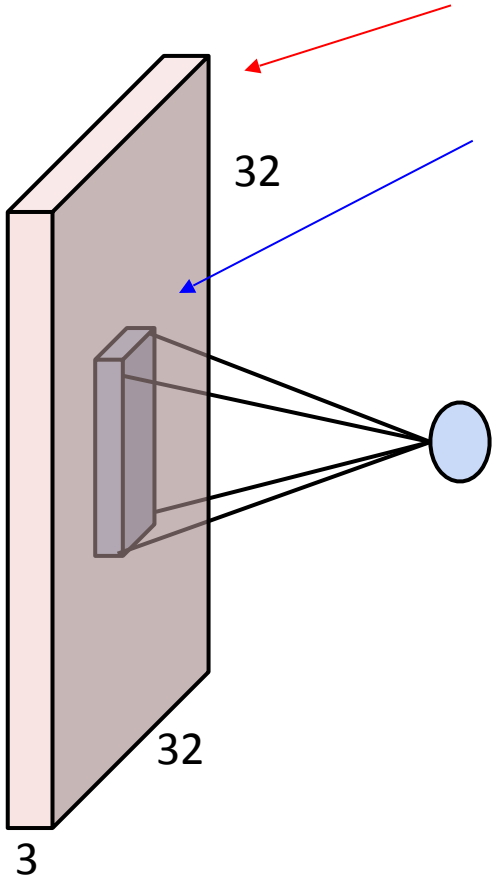
# The brain/neuron view of CONV Layer




**1 number:**
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

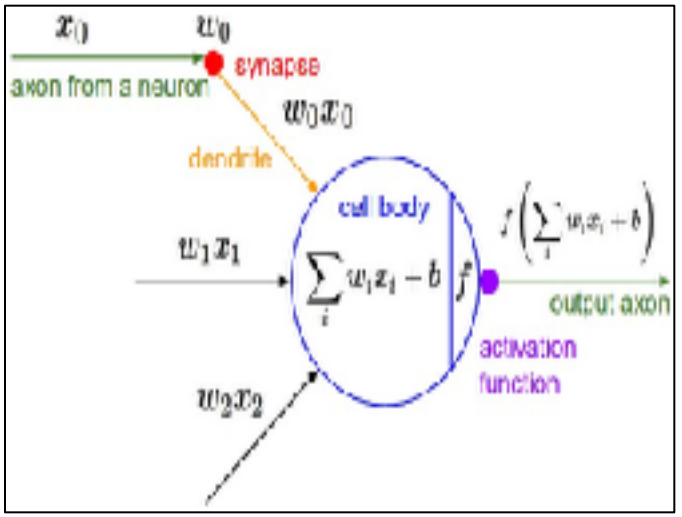
Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# The brain/neuron view of CONV Layer




32x32x3 image

5x5x3 filter

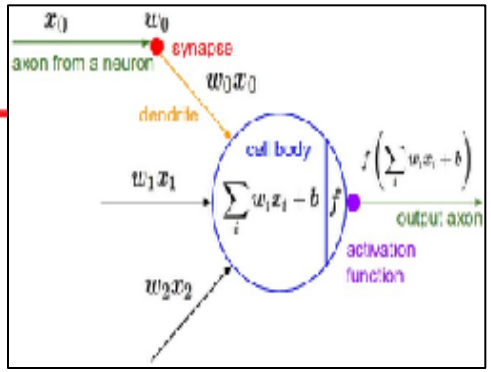


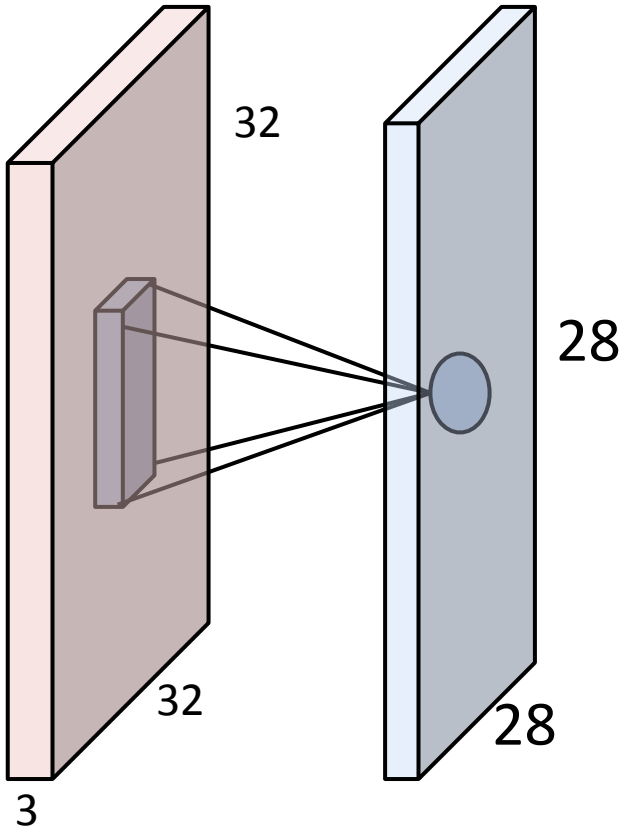

It's just a neuron with local connectivity...

**1 number:**
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# The brain/neuron view of CONV Layer



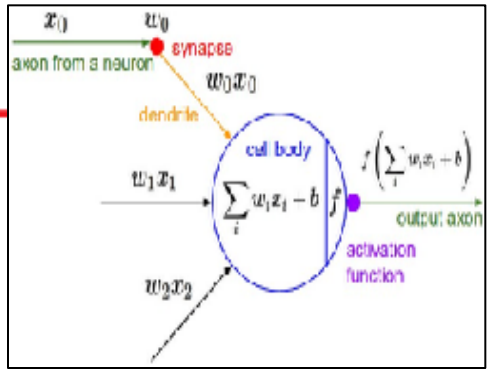

An activation map is a 28x28 sheet of neuron outputs:

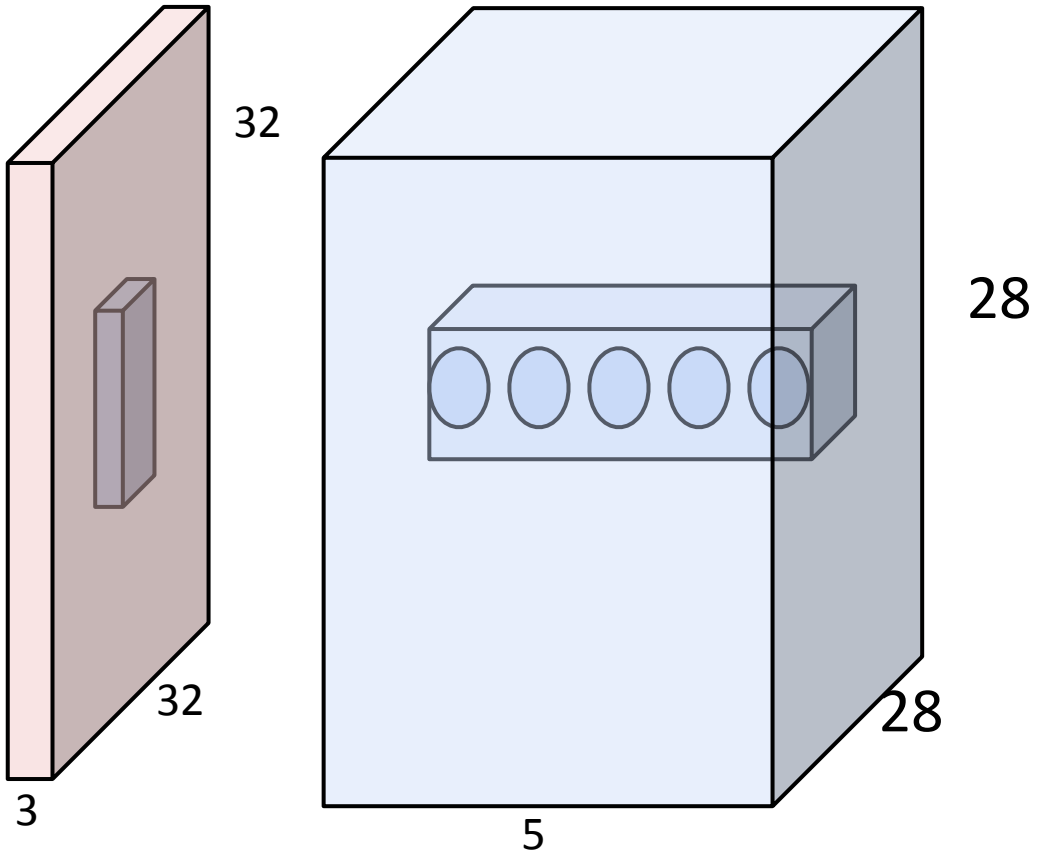
1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# The brain/neuron view of CONV Layer



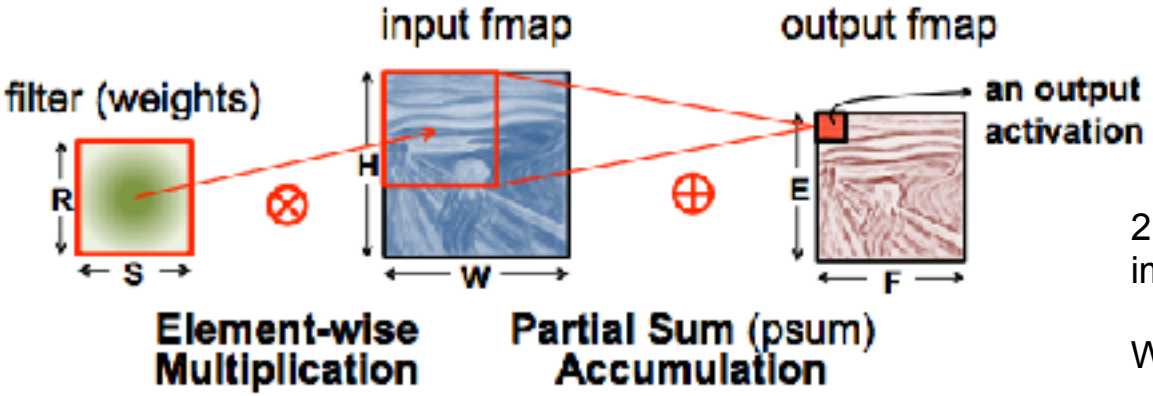




E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

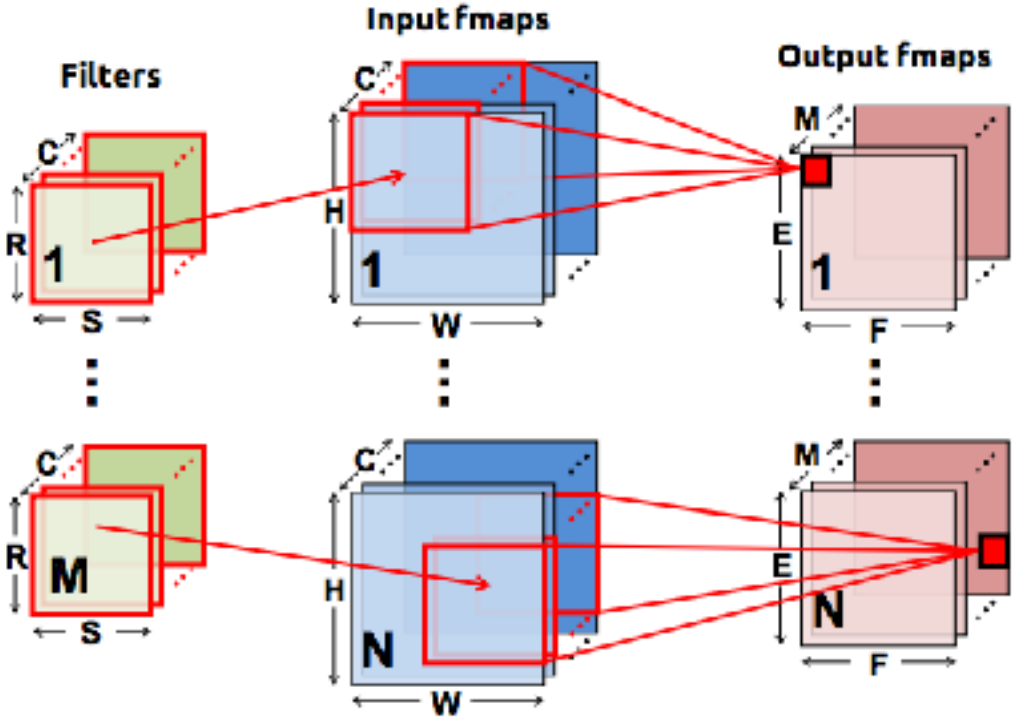
Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# Computational view of CONV Layer




2D Conv in traditional image processing

With pre-defined filters




2D Conv in CNN and

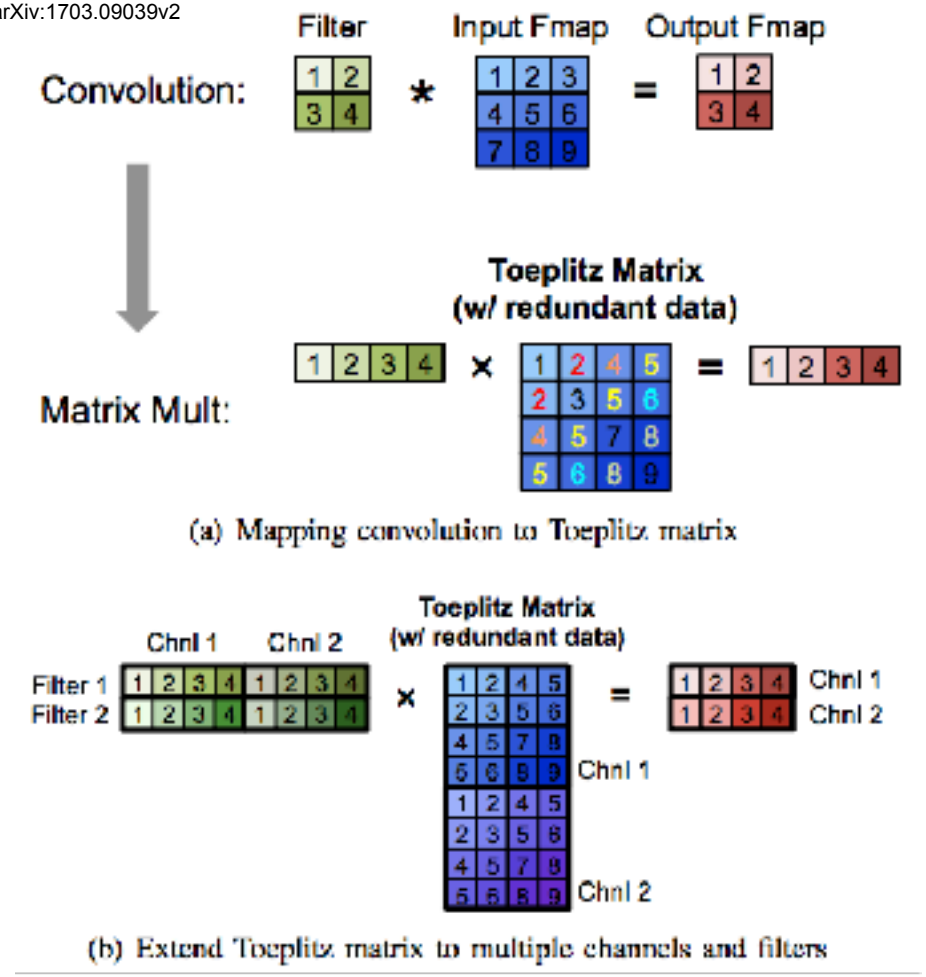
**Filters are unknowns!**

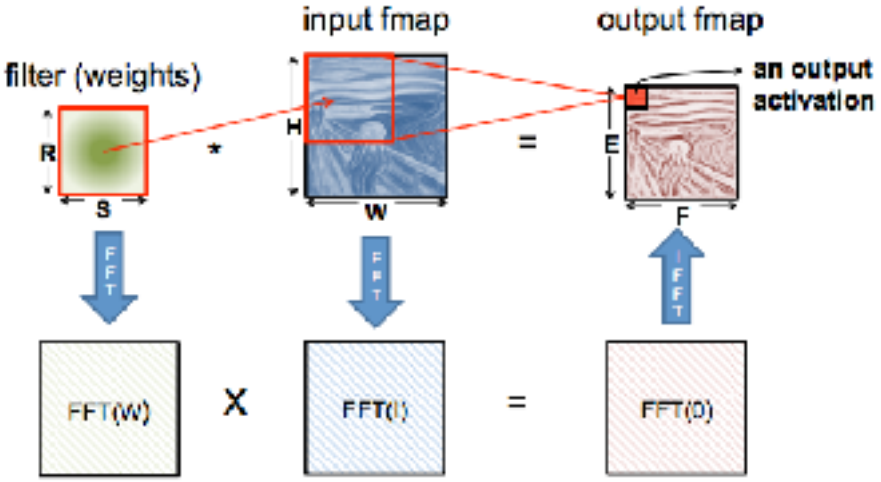
Sze et al. 2017
arXiv:1703.09039v2

# What do DIA folks like about ConvNet

The fundamental computation in ConvNet is **multiply-and-accumulate** (**MAC**) operations, which can be easily parallelised and transformed as:

Sze et al. 2017
arXiv:1703.09039v2

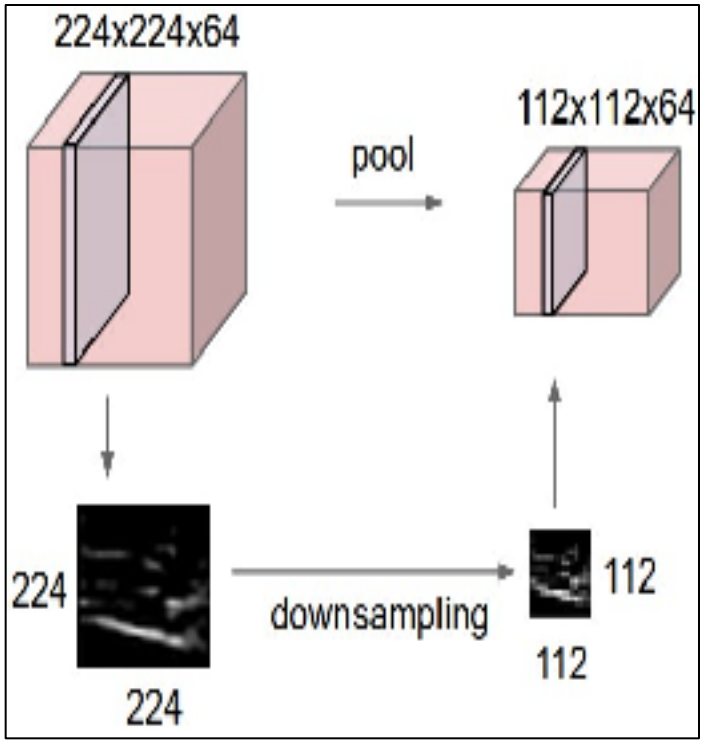



- In practice, different algorithms might be used for different layer shapes and sizes (e.g., FFT for filters greater than 5×5, and Winograd for filters 3×3 and below).
- Existing platform libraries, such as **MKL** and **cuDNN**, dynamically chose the appropriate algorithm for a given shape and size

# Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently
- make the network robust and **invariant** to **small** shifts and distortions




2x2 pooling, stride 2

| | | | |
|---|---|---|---|
| 9 | 3 | 5 | 3 |
| 10 | 32 | 2 | 2 |
| 1 | 3 | 21 | 9 |
| 2 | 6 | 11 | 7 |

**Max** pooling

| | |
|---|---|
| 32 | 5 |
| 6 | 21 |

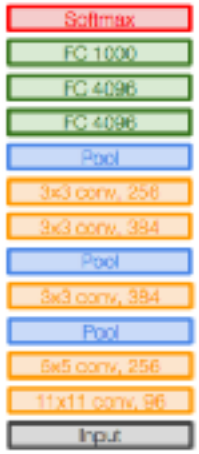
**Average** pooling

| | |
|---|---|
| 18 | 3 |
| 3 | 12 |

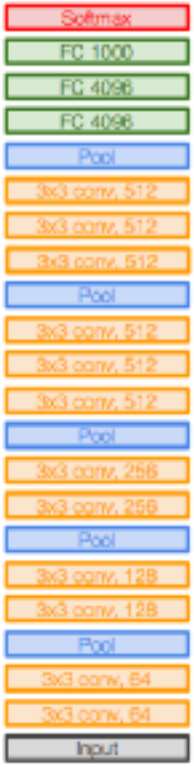
*"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster"*

*Geoff Hinton*

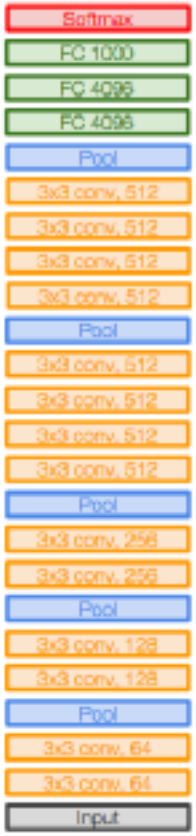
# Which ConvNet architecture should I use



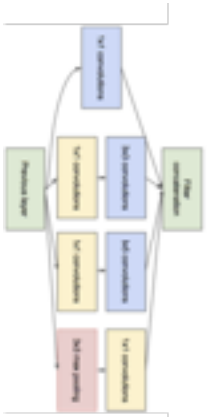

AlexNet




VGG16

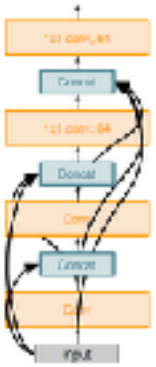


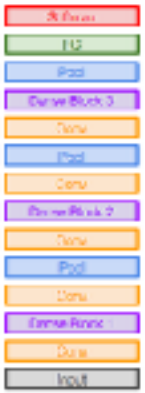

VGG19



GoogleNet

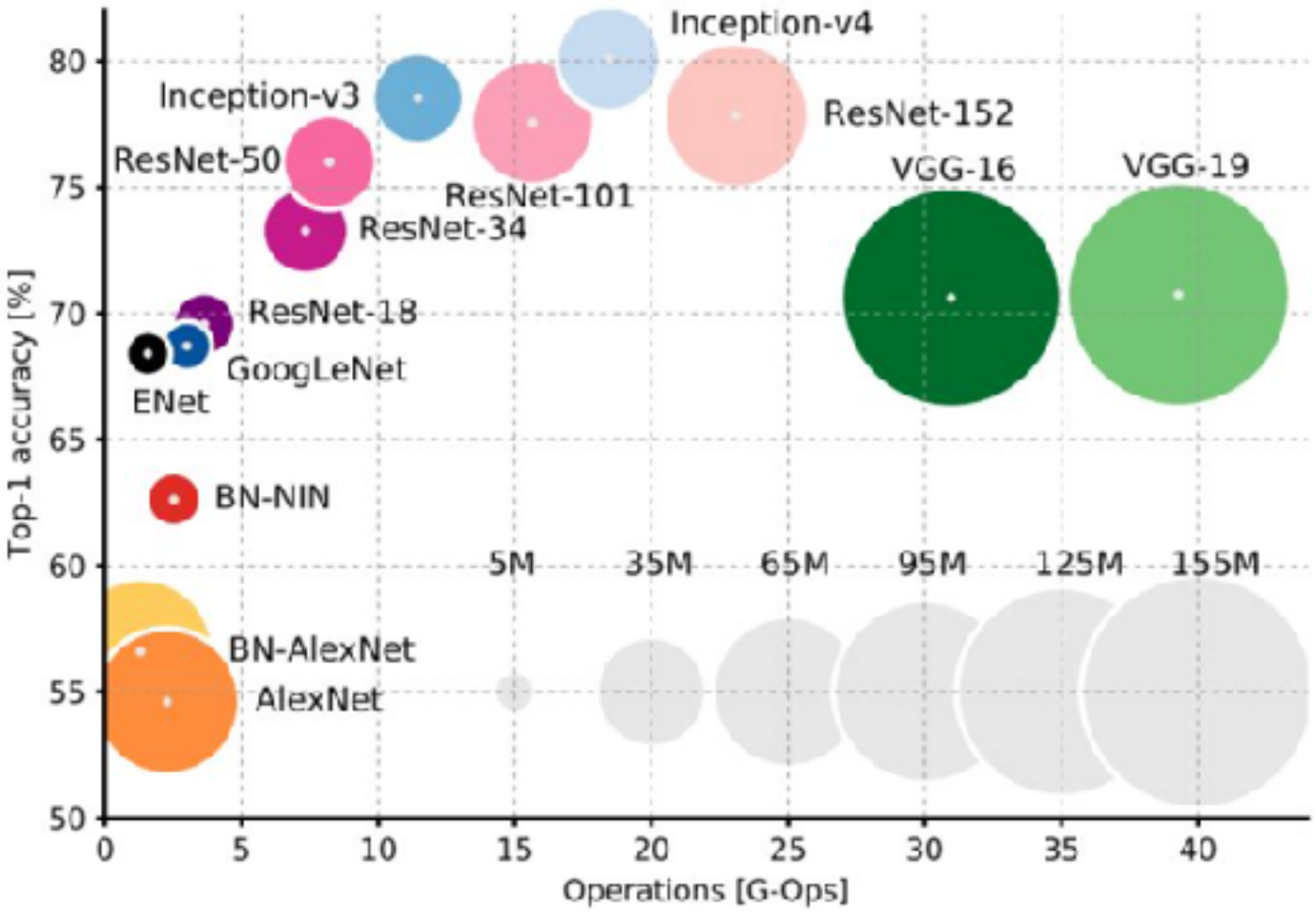
ResNet





DenseNet

# Accuracy vs. Compute vs. Memory



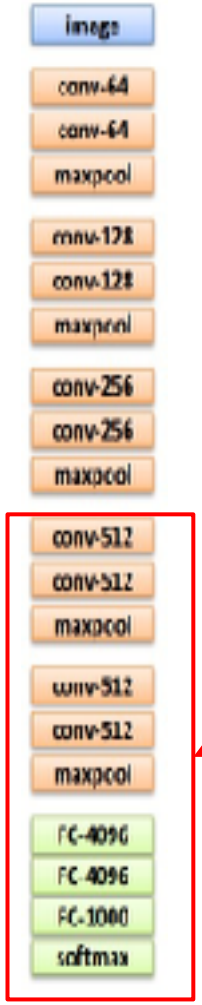

Credit: (Li, Johnson, Yeung 2017)
cs231 stanford

# Almost never train ConvNets from scratch → transfer learning


1. Train on ImageNet


2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end




3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

Credit: (Li, Karpathy, Johnson 2016)
cs231 stanford

# Summary

**What we covered**

- Motivation and brain inspiration
- What's inside neural networks
- Recent ConvNet development
- The inner workings of ConvNet
- ConvNet Architecture and transfer learning

**What we have not covered**

- Spatial details of ConvNet (stride and padding)
- Activation functions and Regularisation (e.g. Dropout)
- Learning rate and batch size fine tuning
  - but **kevin** and **Foivos** will talk about it
- Interpretability of ConvNet output (or DL in general)
- Rotation-invariant Networks → what is wrong with ConvNet
  - Spatial Transformer Networks and The Capsule Networks

# AI Winters: A Regular Hype Cycle

- Hype and then failure of the Perceptron:
  - **AI Winter of 1970-1980**

Kevin was working on this until the funding stopped

- Hype and then failure of LISP/expert machines, also the failure of the various attempts to build "fifth generation" AI machine:
  - **AI Winter of 1990-2000**

Kevin was working on this until the funding stopped

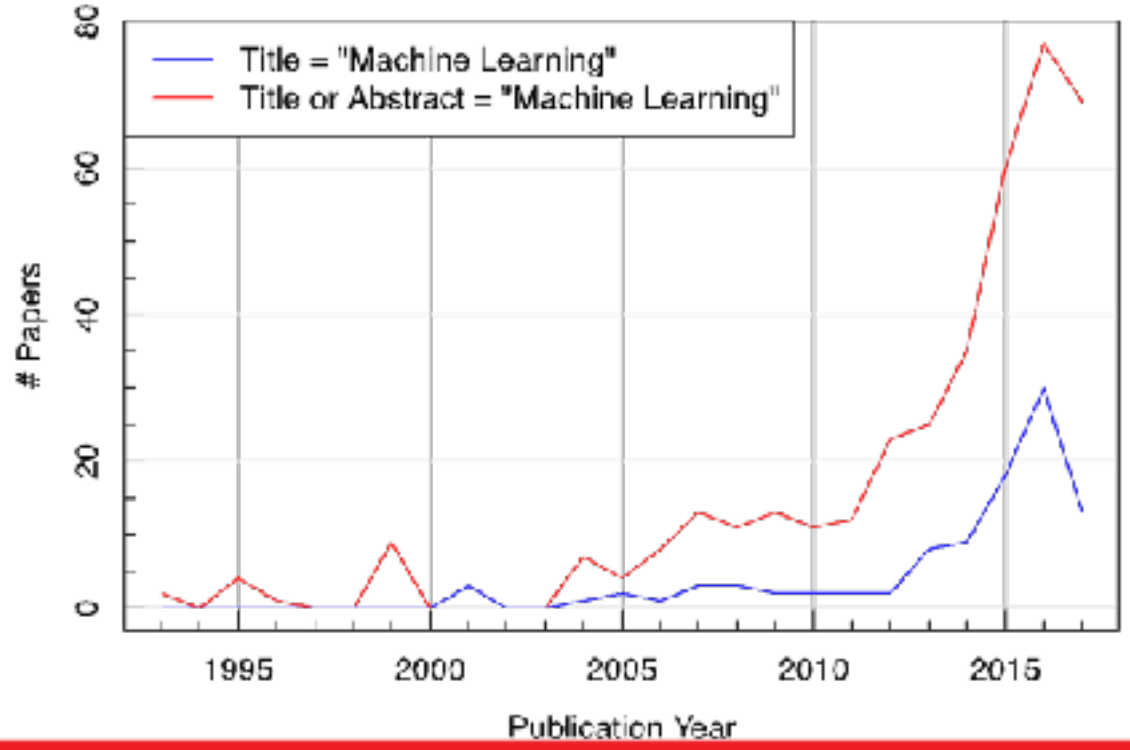
- In the mid 2000s a conscious effort was made to re-brand AI into various academic sub-domains, which is what we have today:
  - Informatics
  - Machine Learning
  - Data Mining
  - Big Data

ML rule:
If Kevin is working on it - it'll soon be time for a Winter of Disillusionment

- These sub-fields have regained credibility by reigning in the promises, and largely leaving the bombast of "thinking machines".

# Machine Learning Use In Astronomy

- Astronomers have been quick to pick up on the new wave of machine learning, shortly after the big mid 2000s rebrand.
- 380 total papers and 4300 citations for machine learning.
- Most cited is modestly ~100.
- Many of the simpler tools are used routinely and without fanfare.

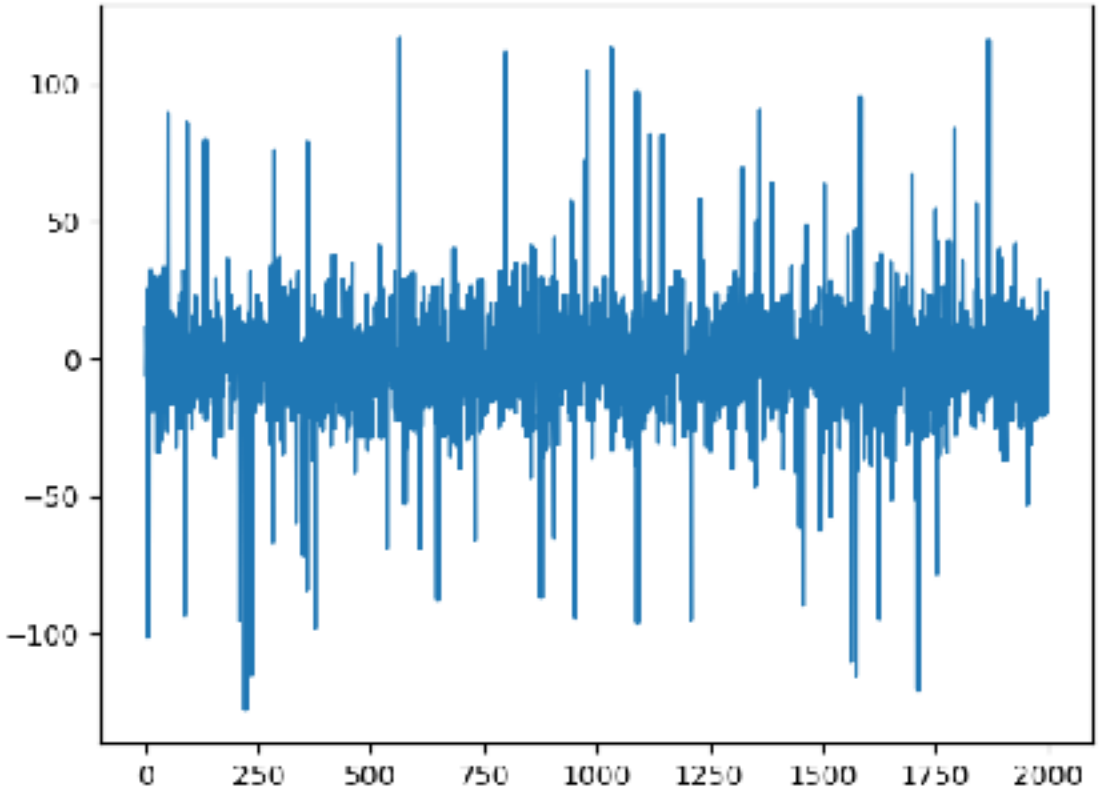


# Time-series

- Time-series
  - 1D Convolution
  - Recurrent Neural Networks  (RNN)
  - Long Short Term Memory (LSTM)
  - Vanilla Flavoured Deep Neural Networks

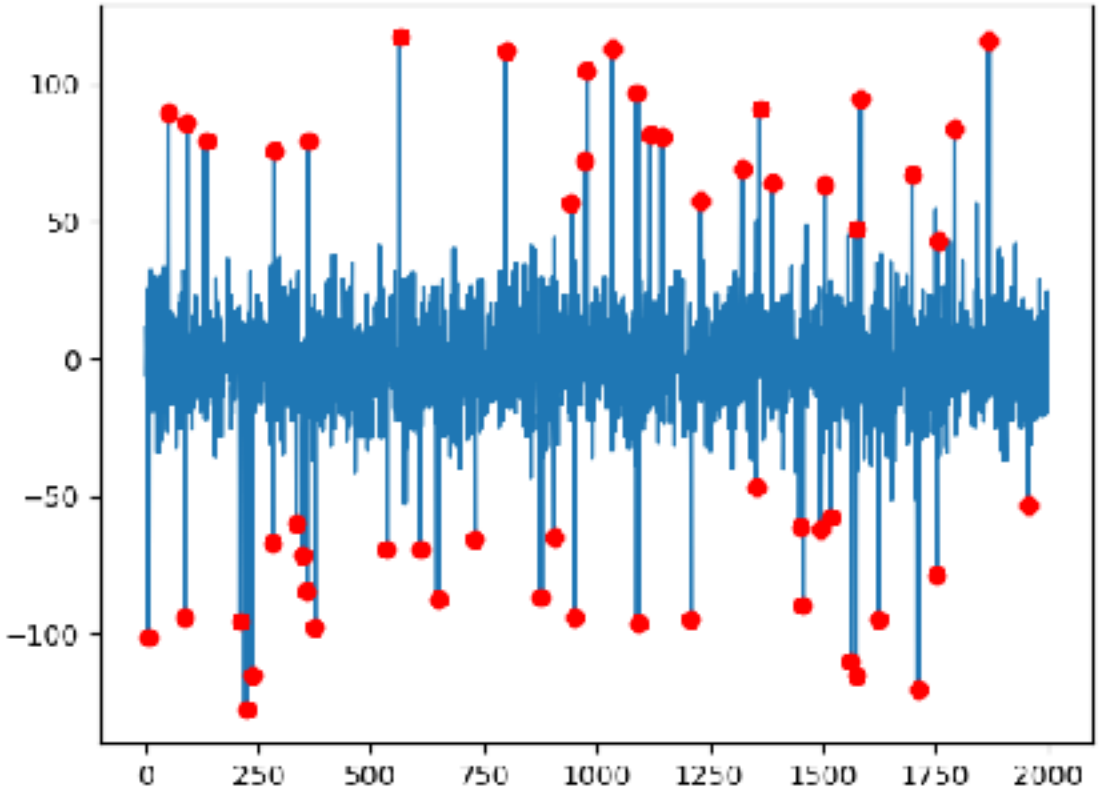
# Time-series

– This is simulated output from GMRT - no signal, just the noise



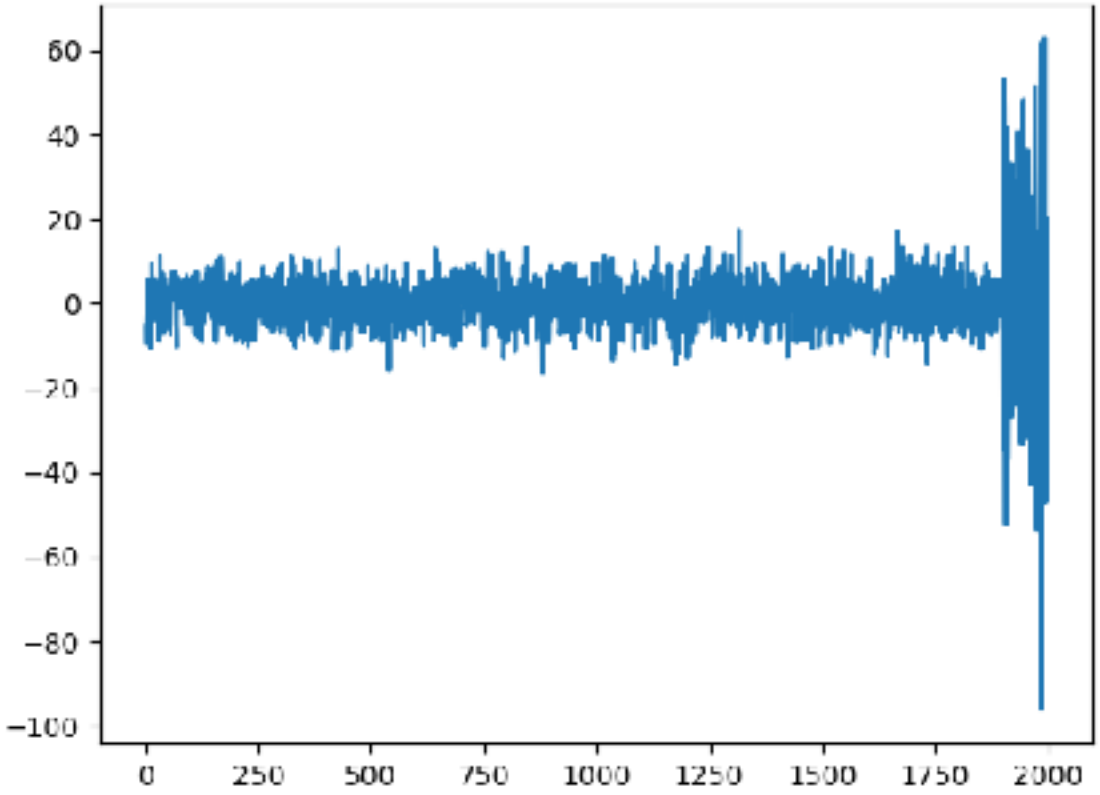
# Time-series

– This is simulated output from GMRT - no signal, just the noise



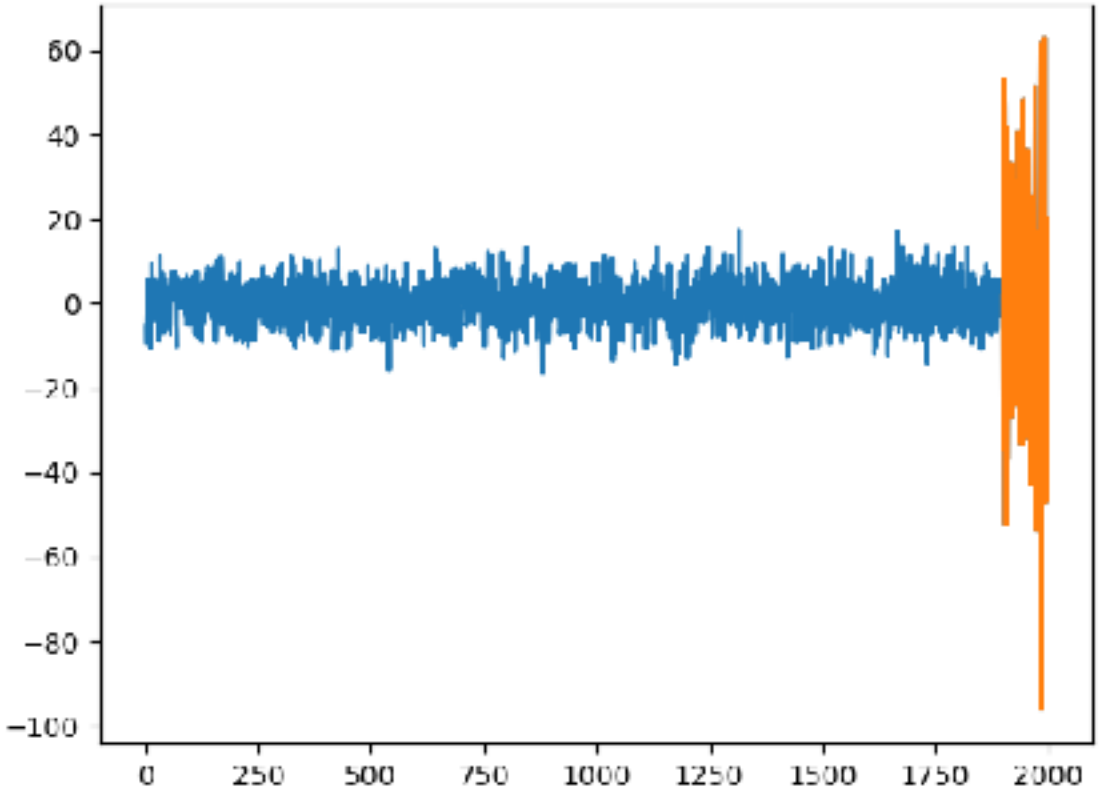
# Time-series

– This is simulated output from GMRT - no signal, just the noise



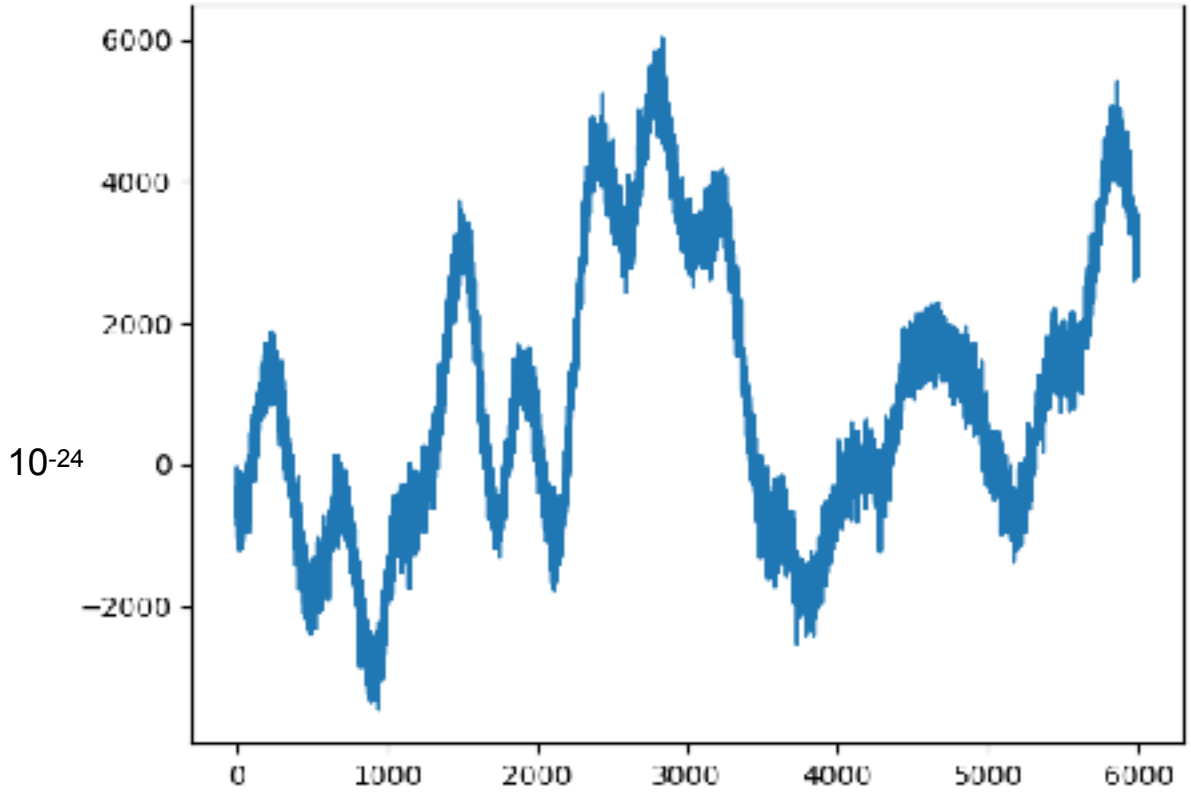
# Time-series

– This is simulated output from GMRT - no signal, just the noise



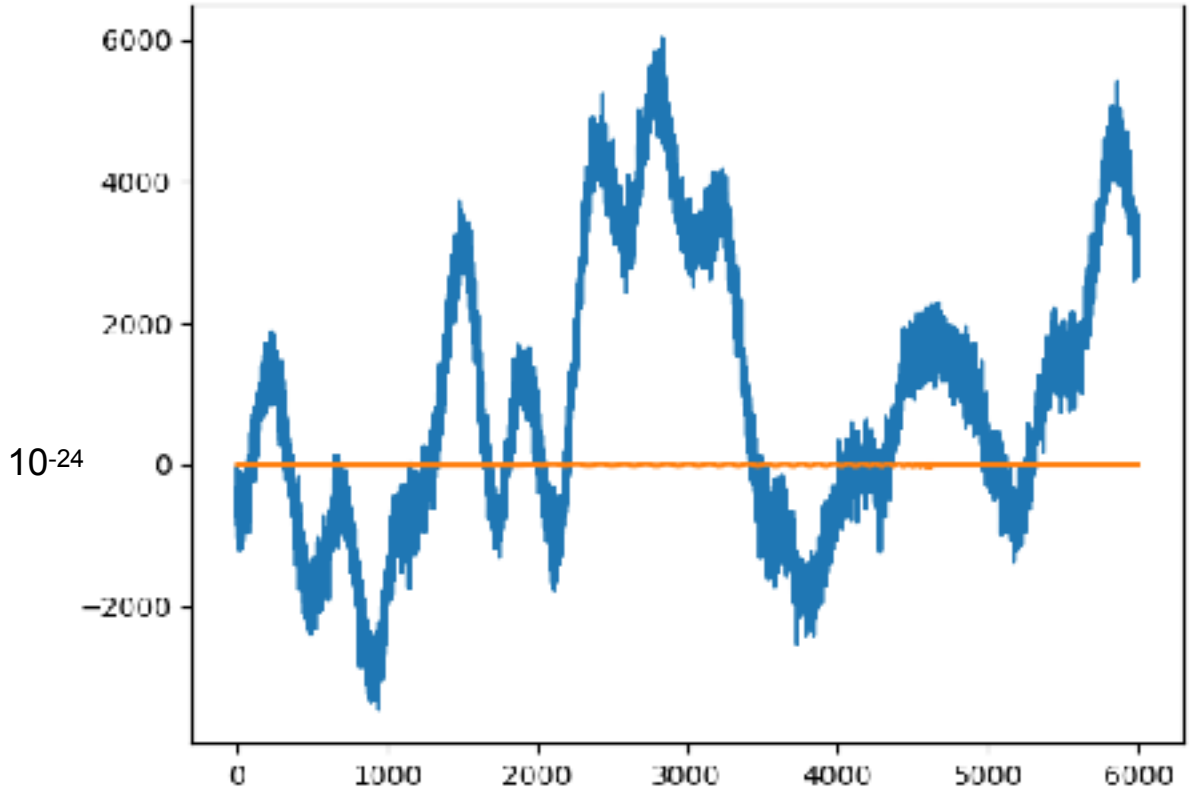
# Time-series

– Or simulated Gravity Waves



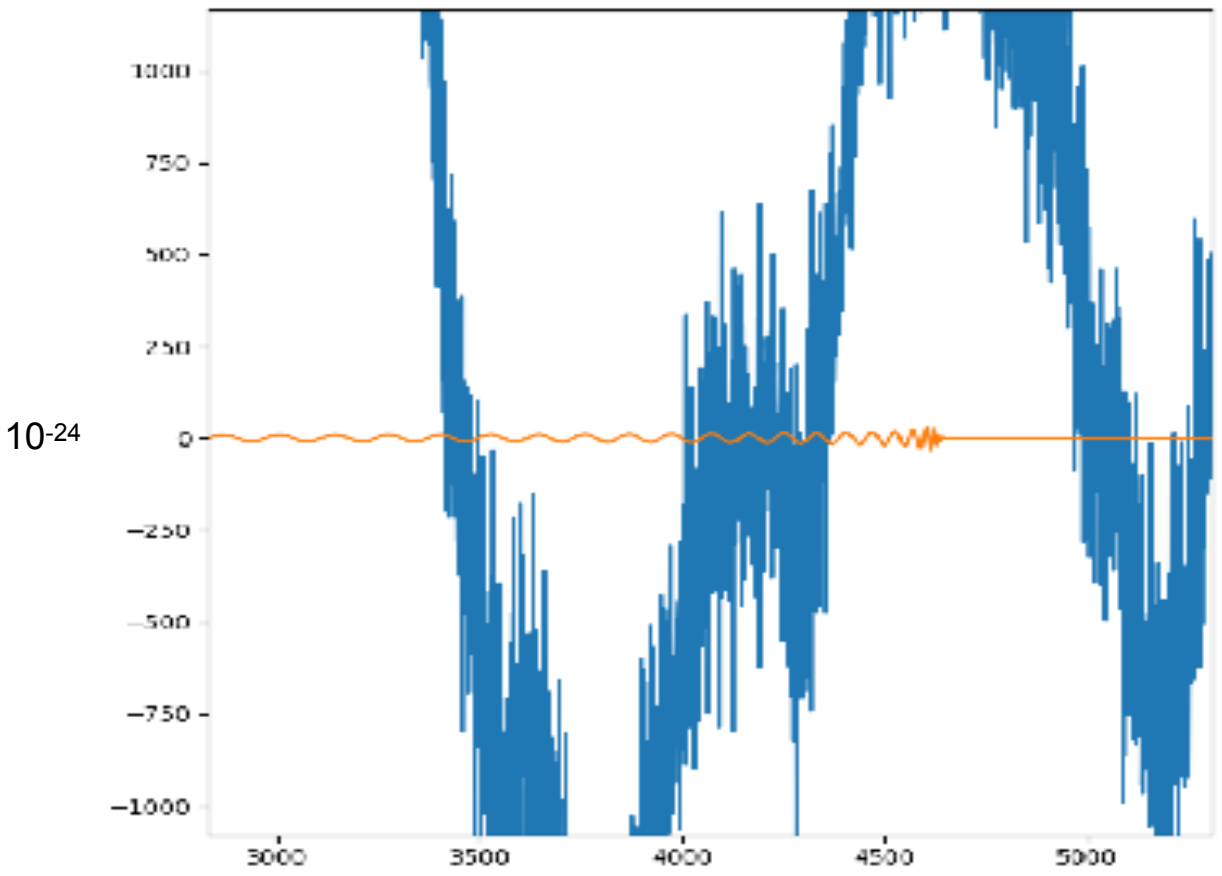
# Time-series

– Or simulated Gravity Waves

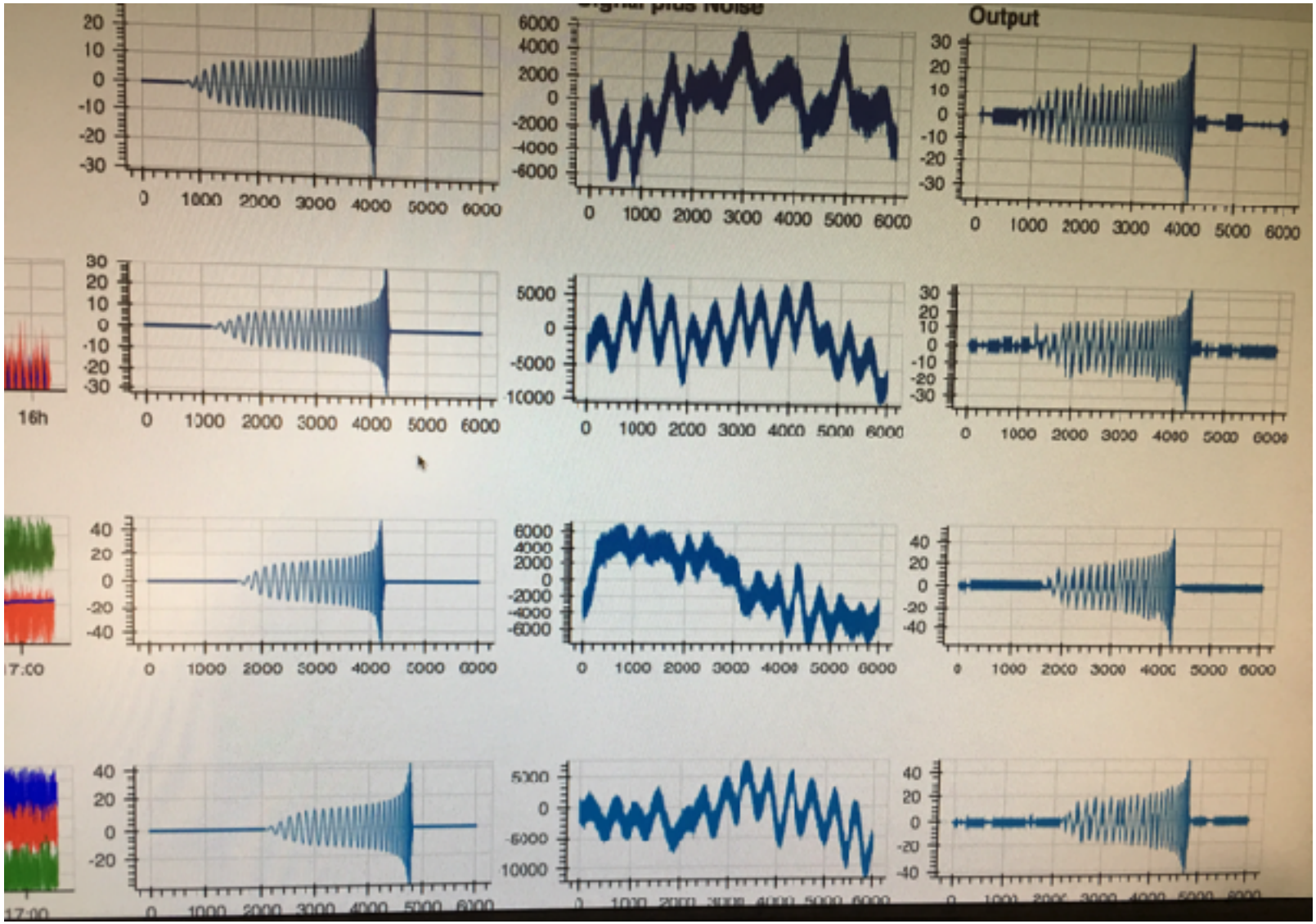


# Time-series

– Or simulated Gravity Waves



# Time-series



# Activation functions

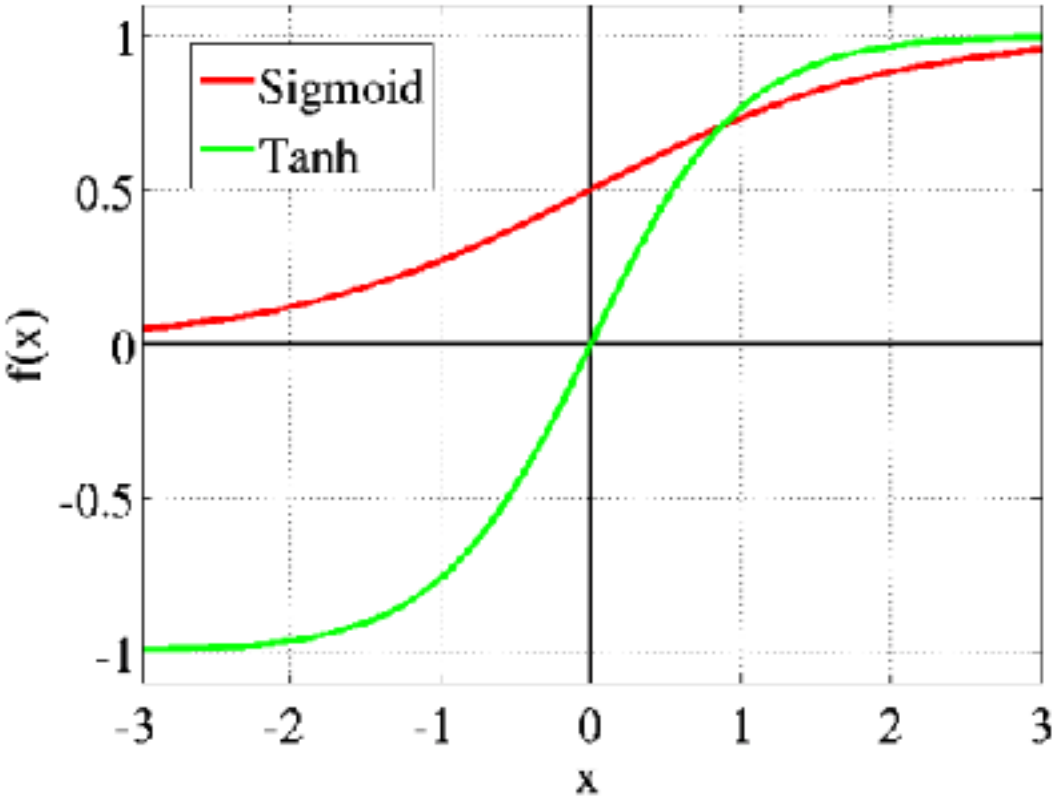



Fig: tanh v/s **Logistic Sigmoid**

# Activation functions

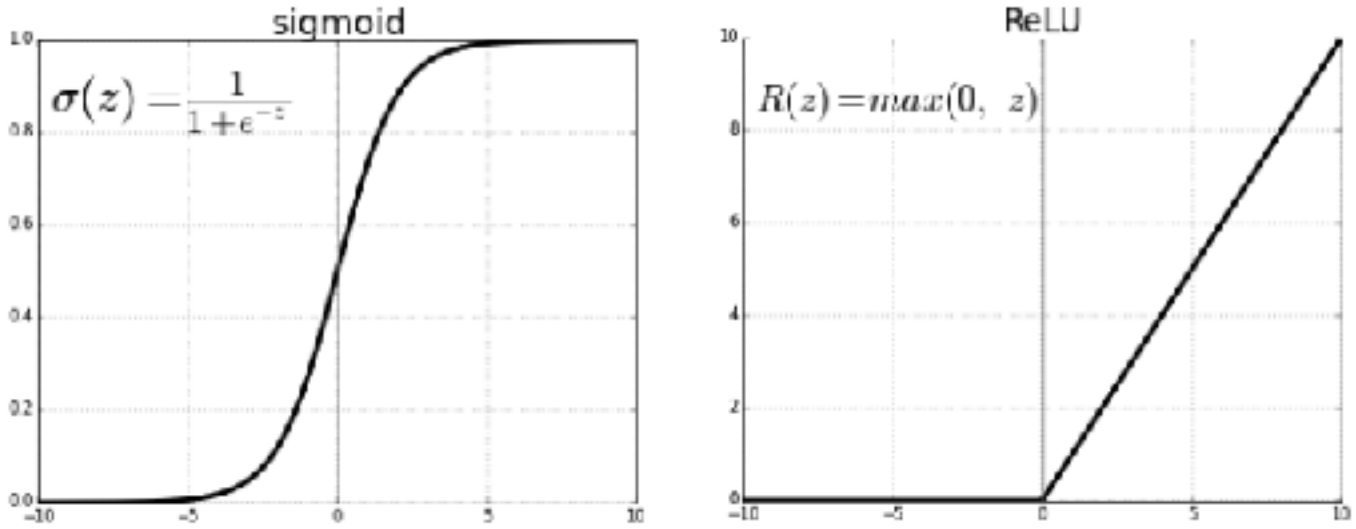



Fig: ReLU v/s Logistic Sigmoid

# Activation functions

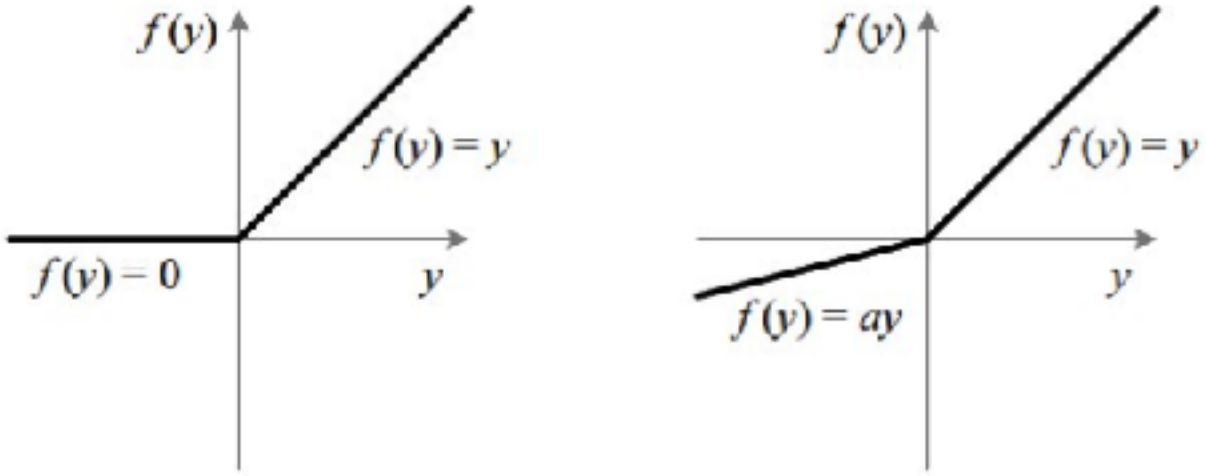



Fig : ReLU v/s Leaky ReLU

# One Hot Encoding

- Many machine learning algorithms cannot work with categorical data directly.
- The categories must be converted into numbers.
- This is required for both input and output variables that are categorical.

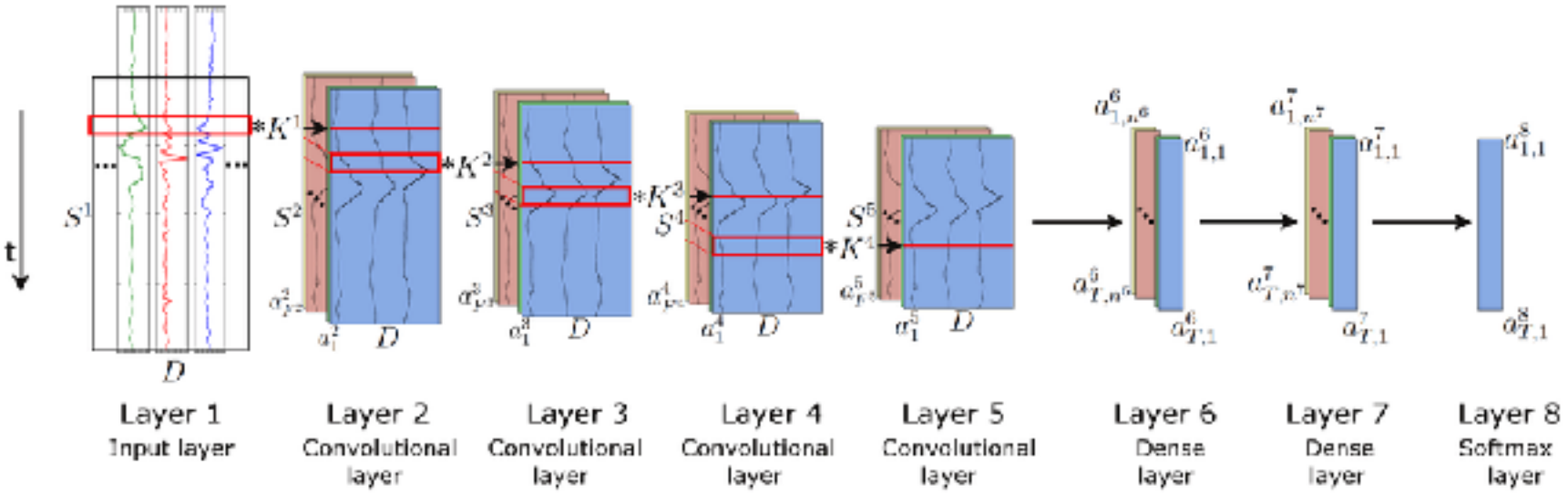
- The integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

- [red, red, green, blue] becomes:
- [[1,0,0], [1,0,0], [0,1,0],[0,0,1]]

# Convolution

- Can be used for time-series

- 1D & 2D Convolution
  - The same as Chen described earlier
  - You define a series of convolutions on the 1D multi-channel data



# Convolution

```
class GmrtCNN(nn.Module):
    def __init__(self, keep_probability=0.5):
        super(GmrtCNN, self).__init__()
        self.keep_probability = keep_probability
        self.conv1 = nn.Conv1d(NUMBER_CHANNELS, 18, kernel_size=2, stride=1).double()
        self.max_pool1 = nn.MaxPool1d(2, stride=2)
        self.conv2 = nn.Conv1d(18, 36, kernel_size=2, stride=1).double()
        self.max_pool2 = nn.MaxPool1d(2, stride=2)
        self.conv3 = nn.Conv1d(36, 72, kernel_size=2, stride=1).double()
        self.max_pool3 = nn.MaxPool1d(2, stride=2)
        self.fc1 = nn.Linear(144, 36).double()
        self.fc2 = nn.Linear(36, 36).double()
        self.fc3 = nn.Linear(36, NUMBER_OF_CLASSES).double()

    def forward(self, input_data_raw, input_periodogram):
        x = self.max_pool1(functional.relu(self.conv1(input_data_raw)))
        x = self.max_pool2(functional.relu(self.conv2(x)))
        x = self.max_pool3(functional.relu(self.conv3(x)))
        x = x.view(x.size(0), -1)
        x = functional.dropout(x, p=self.keep_probability, training=self.training)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)

        x = functional.sigmoid(x)
        return x
```

# Recurrent Neural Network

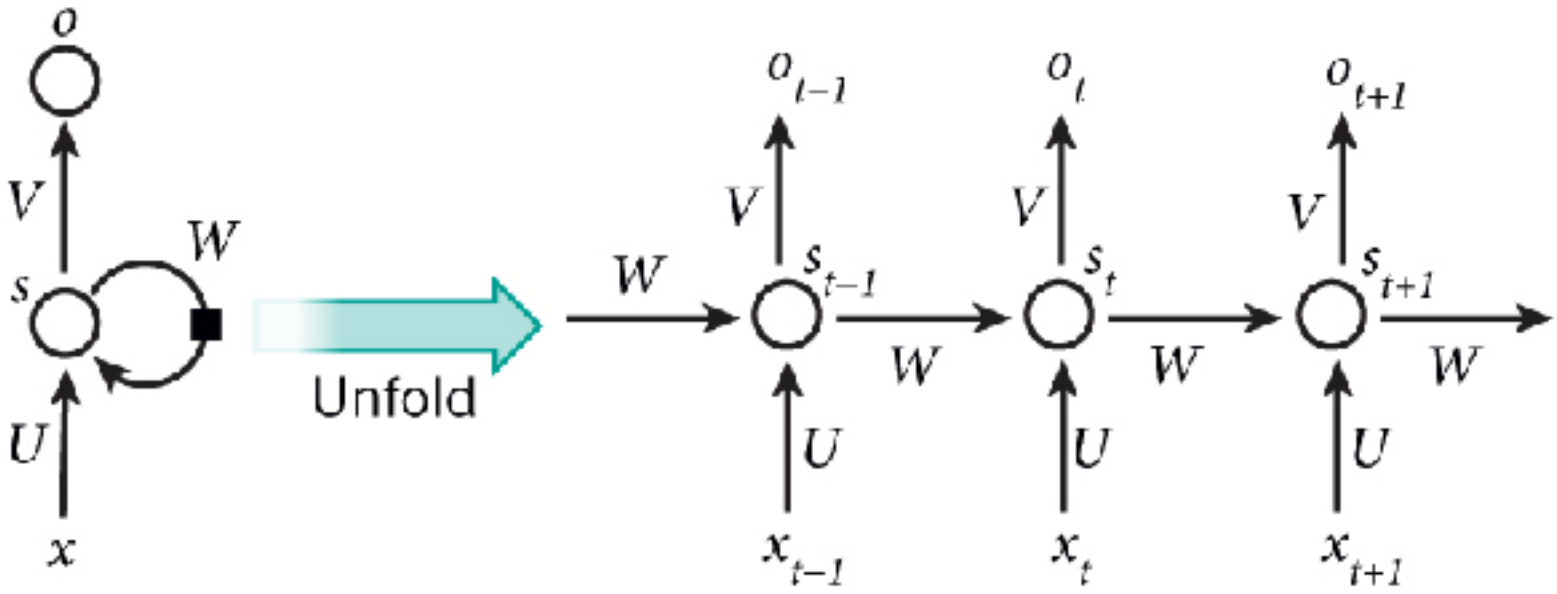
- Mostly used on Natural Language Processing because that is the “hot” topic.
- I’ve used them on data from GMRT with some success

# Recurrent Neural Network

- The idea behind RNNs is to make use of sequential information
- In a traditional neural network we assume that all inputs (and outputs) are independent of each other.
- If you want to predict the next word in a sentence you better know which words came before it.
- RNNs can be thought of as having a “memory” which captures information about what has been calculated so far.

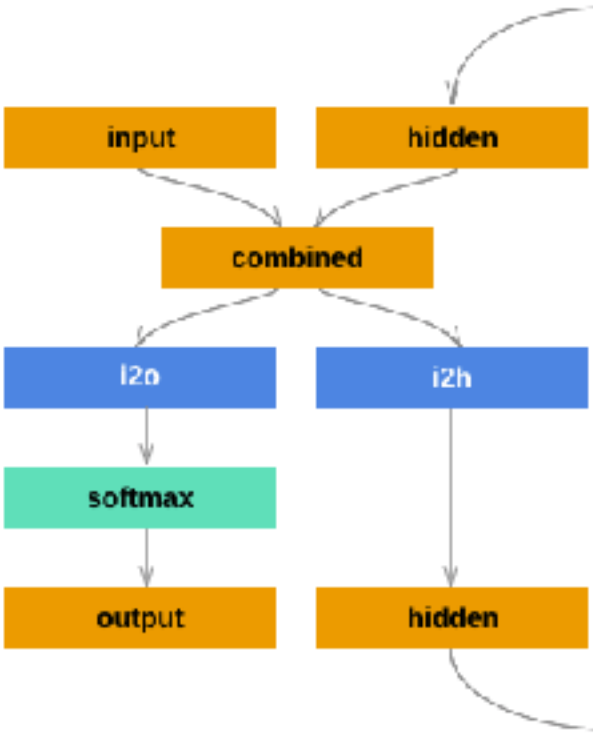
# Recurrent Neural Network

– A little like this



# Recurrent Neural Network

– Show me the code… From the Pytorch Tutorial

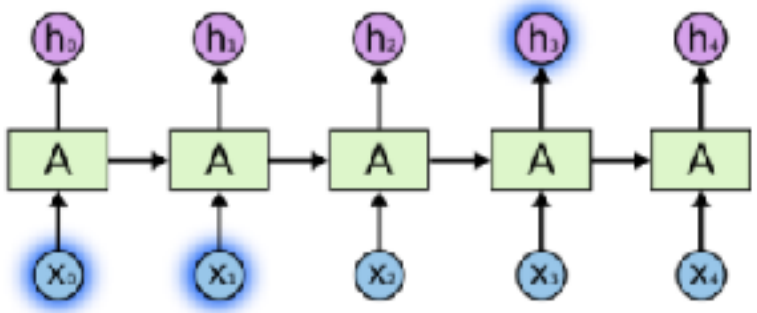


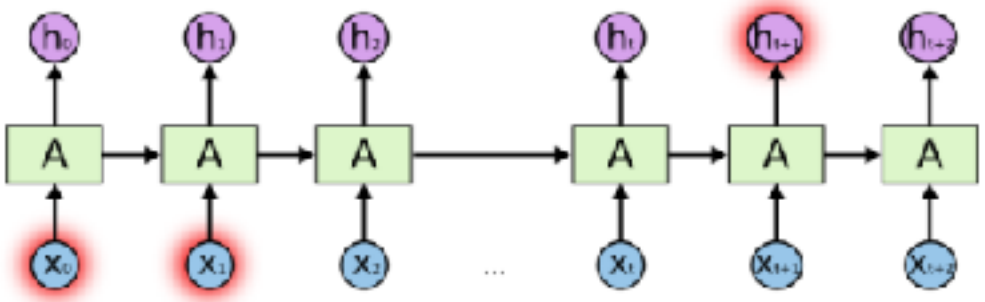

```
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1, self.hidden_size))

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```


# Vanishing Gradient

- In many RNN we notice something odd: the weights closer to the end of the network change a lot more than those at the beginning.
- The deeper the network, the less and less the beginning layers change.
- This is problematic, because our weights are initialized randomly. If they're barely moving, they're never going to reach the right values, or it'll take them years.

# Vanishing Gradient



# Long Short Term Memory

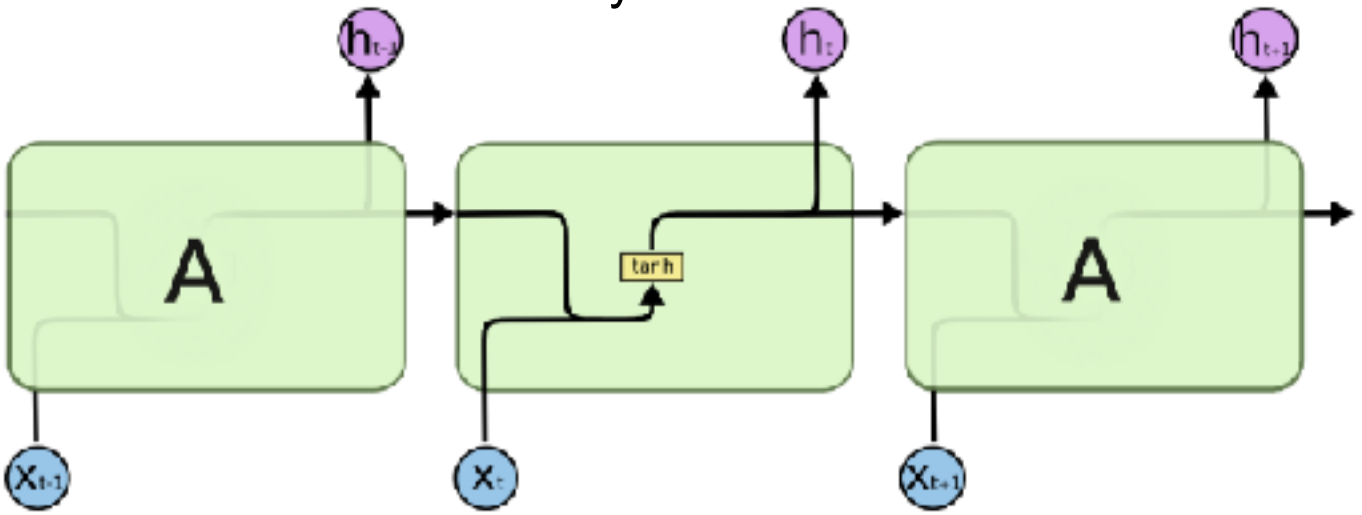
- The Long Short-Term Memory network, or LSTM, is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem.
- It can be used to create large recurrent networks that in turn can be used to address difficult sequence problems.
- Instead of neurons, LSTM networks have memory blocks that are connected through layers
- A block contains gates that manage the block's state and output.
- A block operates upon an input sequence and each gate within a block uses the sigmoid activation units to control whether they are triggered or not, making the change of state and addition of information flowing through the block conditional.

# Long Short Term Memory

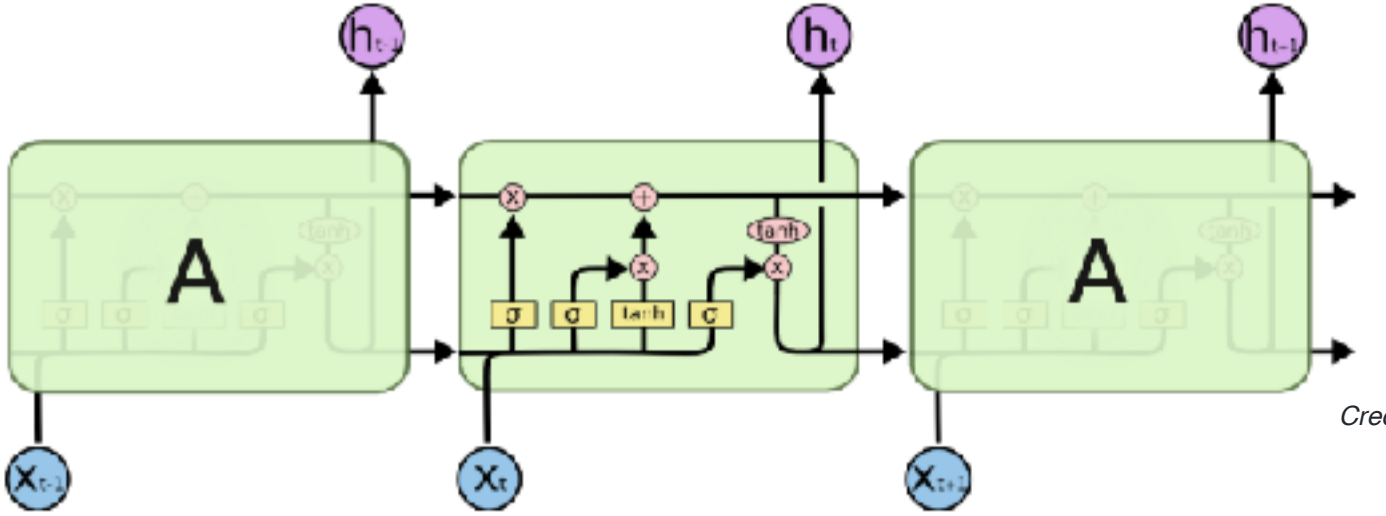
- There are three types of gates within a unit:
  - Forget Gate: conditionally decides what information to throw away from the block.
  - Input Gate: conditionally decides which values from the input to update the memory state.
  - Output Gate: conditionally decides what to output based on input and the memory of the block.
- Each unit is like a mini-state machine where the gates of the units have weights that are learned during the training procedure.

# Long Short Term Memory

– A RNN has one hidden layer




– LSTM contains 4 interacting layers




Credit: colah.github.io


# Long Short Term Memory

- Many different variants
  - Peephole connections
  - Coupled forget and input gates
  - Gated Recurrent Unit (GRU)

# RNN & LSTM

- Used a lot in Natural Language processing
- Some use in signal processing


# DNN

- You can use “vanilla” deep networks with sequences.
- Similar to RNN and have the same problems

```
class GmrtLinear(nn.Module):
  def __init__(self, keep_probability, sequence_length):
    super(GmrtLinear, self).__init__()
    self.keep_probability = keep_probability
    self.input_layer_length = 6 + (sequence_length * 7)

    self.fc1 = nn.Linear(self.input_layer_length, HIDDEN_LAYERS).double()
    self.fc2 = nn.Linear(HIDDEN_LAYERS + self.input_layer_length, HIDDEN_LAYERS).double()
    self.fc3 = nn.Linear(HIDDEN_LAYERS, HIDDEN_LAYERS).double()
    self.fc4 = nn.Linear(HIDDEN_LAYERS, HIDDEN_LAYERS).double()
    self.fc5 = nn.Linear(HIDDEN_LAYERS, HIDDEN_LAYERS).double()
    self.fc6 = nn.Linear(HIDDEN_LAYERS, NUMBER_OF_CLASSES).double()

  def forward(self, input_data_values):
    x = functional.leaky_relu(self.fc1(input_data_values))
    x = functional.leaky_relu(self.fc2(torch.cat((x, input_data_values), dim=1)))
    x = functional.dropout(x, p=self.keep_probability, training=self.training)
    x = functional.leaky_relu(self.fc3(x))
    x = functional.leaky_relu(self.fc4(x))
    x = functional.dropout(x, p=self.keep_probability, training=self.training)
    x = functional.leaky_relu(self.fc5(x))
    x = functional.leaky_relu(self.fc6(x))

    x = functional.softmax(x)
    return x
```

# Which is best?

- It all depends on what you want to do
- One size does not fit all
- We have used all the methods I have described successfully

# Thanks

To my colleague Dr Aaron Robotham and Dr Chen Wu for letting me “borrow” many of their slides