Deep Learning

Slides credits – Stanford CS231n, Roger Grosse, Keyulu Xu

Outline

- General Overview
- Vanilla Neural Network/Multi layer Perceptron
- How to train your model?
- Best Practices
- Convolutional Neural Networks
- Glimpse of other techniques – sequence models, Generative models…

Deep learning systems are neural network models that were introduced in the ’80s and ’90s

But then why now?

But then why now?

- architectural and algorithmic innovations (e.g. many layers, ReLUs, Adam optimizer, dropout, batch-norm, residual connections, LSTMs, attention, ...)
- vastly larger data sets (e.g. Imagenet, Pascal VOC, …)
- vastly larger-scale compute resources (GPUs, TPUs, cloud)
- much better software tools (PyTorch, TensorFlow, Chainer, …)
- vastly increased industry investment and media hype

GPU

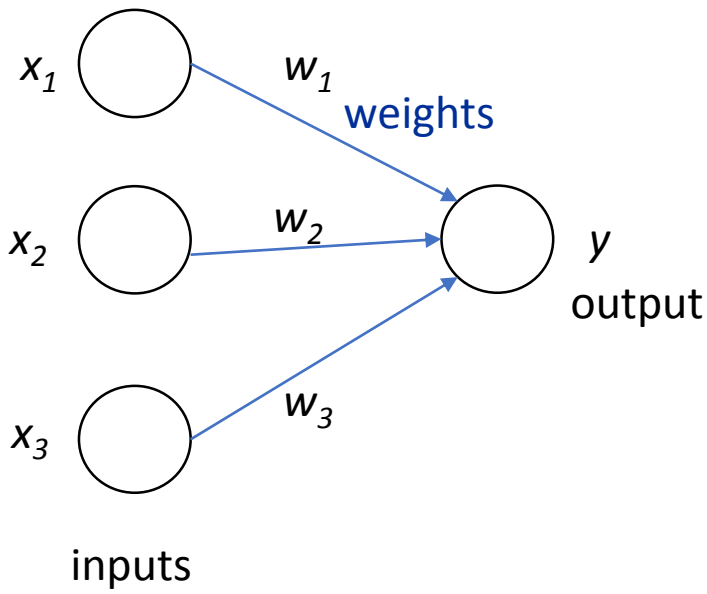
TPU

Zoubin Ghahramani - NIPS 2016

Neural Networks

https://bit.ly/2C080IK (Magician 3blue1brown's explanation)

Let's start with the simplest unit!

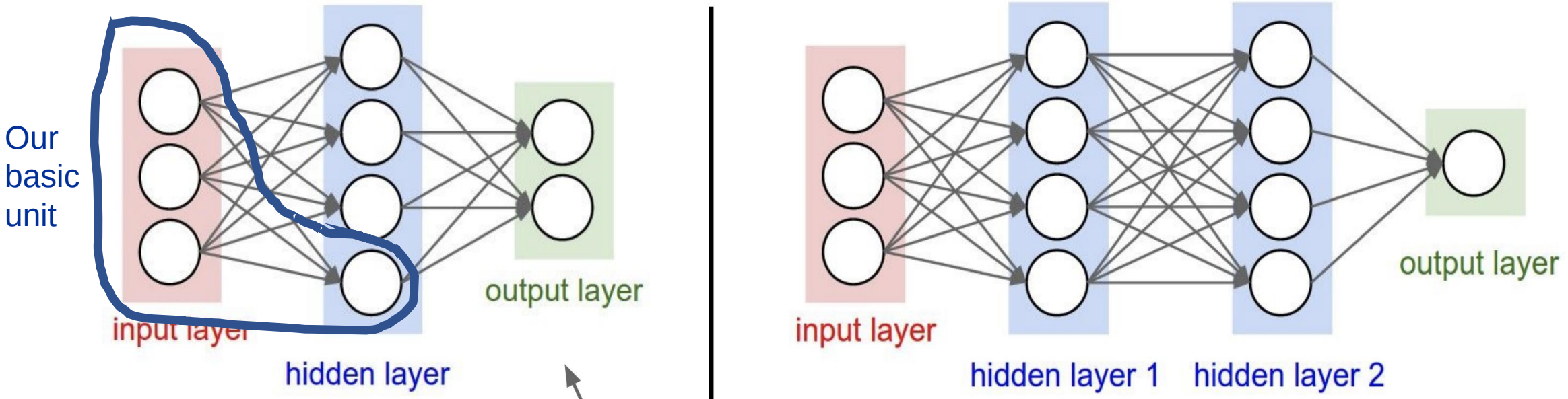


How to read this graphical representation

$y = \sigma(Wx)$, where φ is a non-linear function called as activation function

These units are much more powerful if we stack many of them together! This stacked network will be called a neural network!

Neural Network



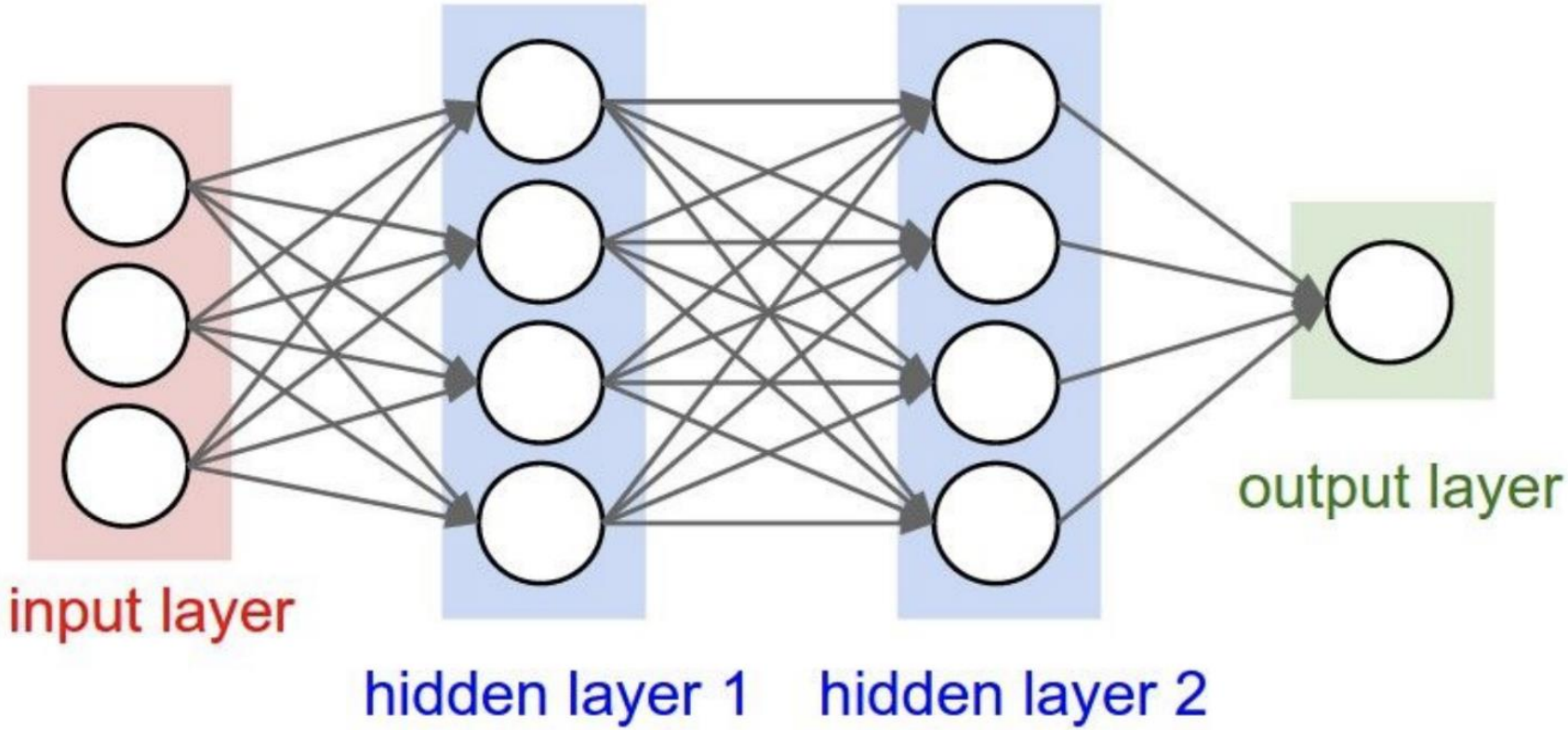
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Terminology in deep learning

- Deep learning terminology can be confusing sometimes
- Be aware of that!
- For e.g. a neural network with few hidden layers (roughly less that 4-5) is also called as Multi-layer perceptron (MLP)
- A fully connected layer is also called a feed-forward layer
- We will try to notify whenever there is any confusing terminology



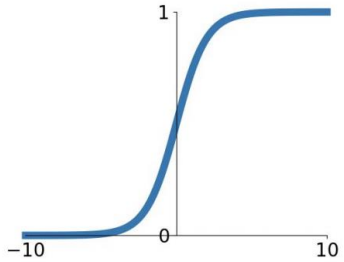
How many weights in each layer?

What if we didn’t use any activation function?

Common activation functions

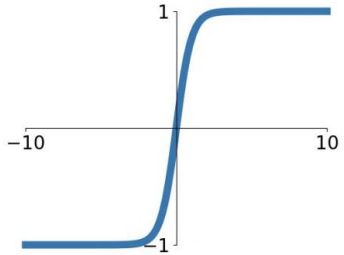
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



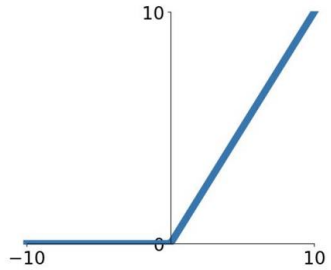
tanh

$$\tanh(x)$$



ReLU

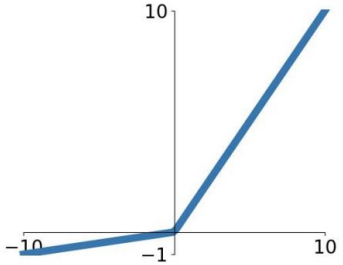
$$\max(0, x)$$



ReLU is a good default choice

Leaky ReLU

$$\max(0.1x, x)$$

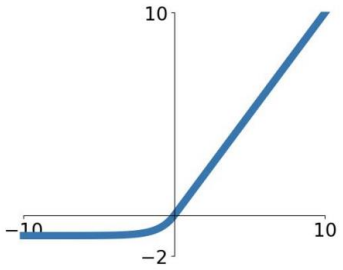


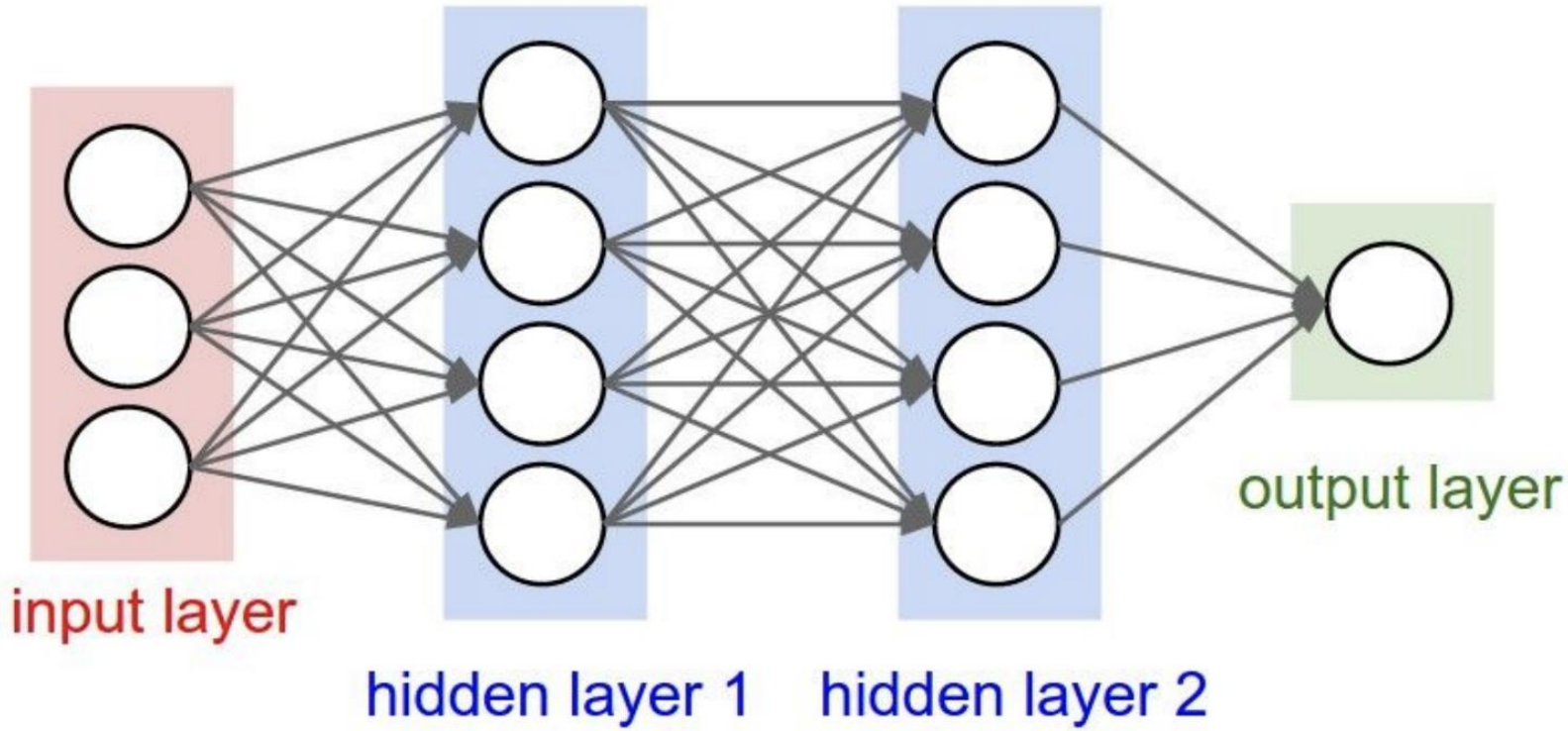
Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

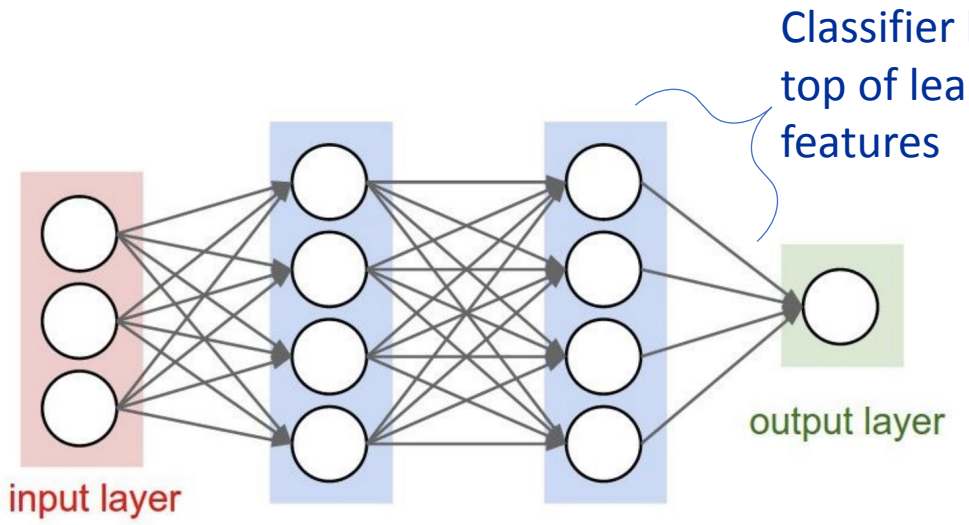
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$




input layer
hidden layer 1
hidden layer 2
output layer

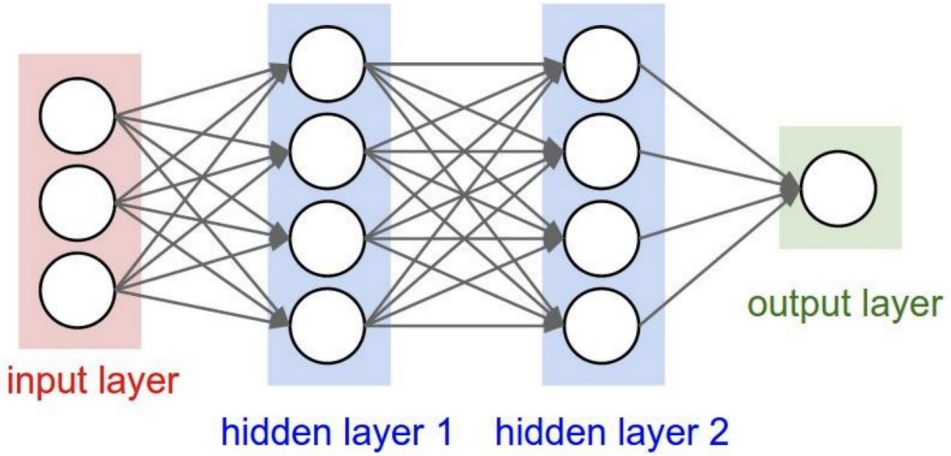
Different ways to think about a neural network

Automatic Feature Learning



Different ways to think about a neural network

Function Approximator

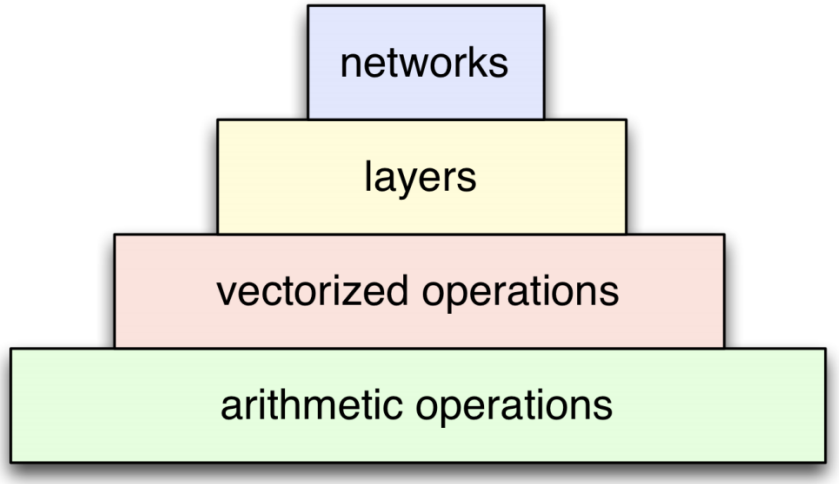


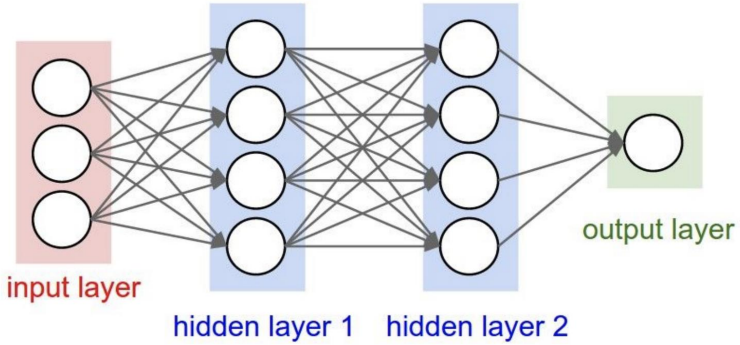
$x \longrightarrow f \longrightarrow y$

$$f(x) = \sigma[W_2\,[\sigma(W_1 x + b_1)] + b_2]$$

Levels of abstraction

When you design neural networks and machine learning algorithms, you'll need to think at multiple levels of abstraction.





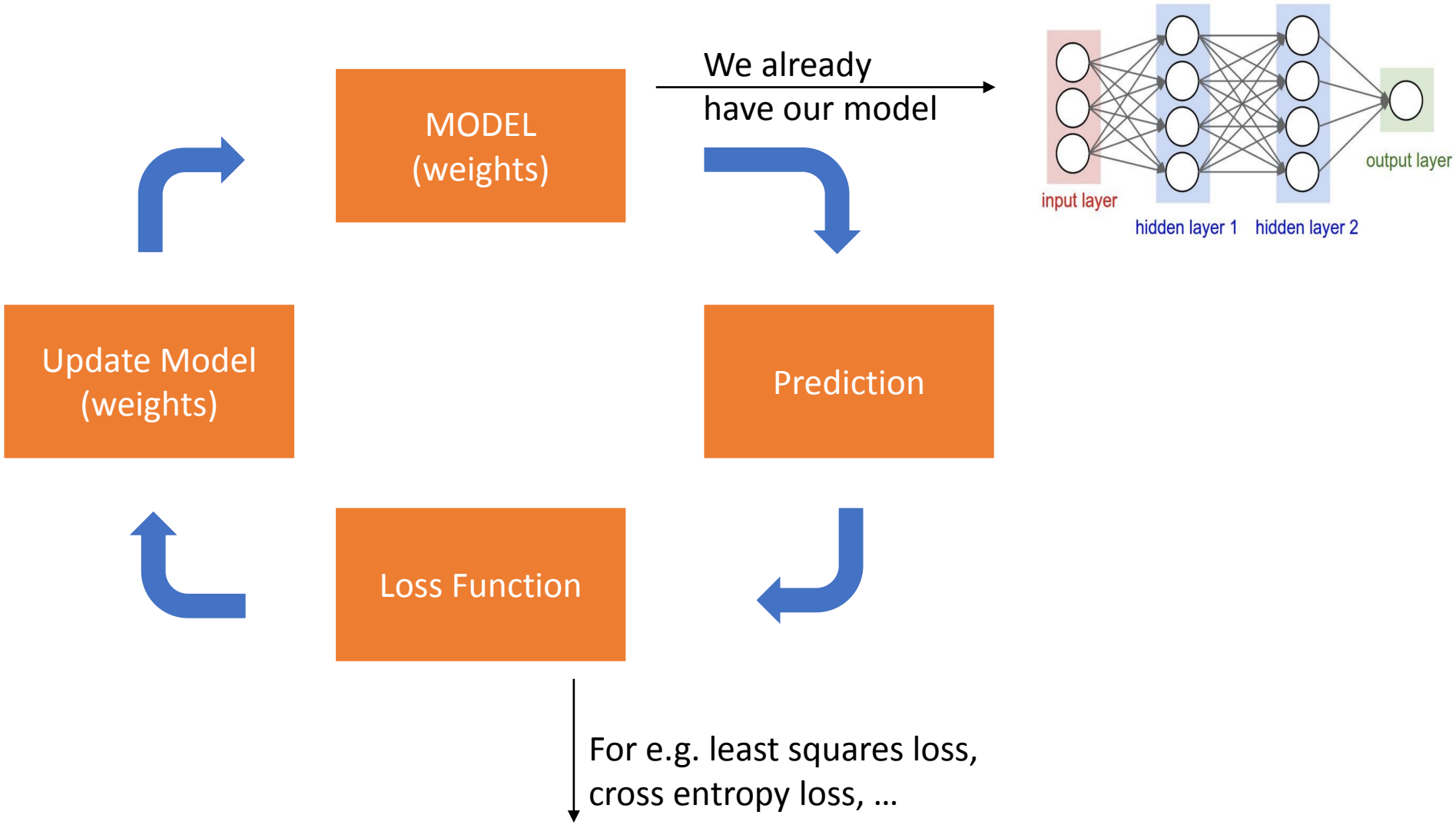
Now, we have our model!

How do we train it?

Backpropagation!

Training Neural Networks with Backpropagation

General (Supervised) ML Training Loop



How do we update our model?

- Goal: find weights w^* that minimize the total loss i.e.

$$w^* = \arg\max_w L$$ where L is total loss = sum of losses for each input

$$w^* = \arg\max_w \sum_{i=1}^{n} loss(x_i)$$

$$w^* = \arg\max_w \sum_{i=1}^{n} (y - y^*)^2$$ ← For least squares loss

- This is a continuous optimization problem.
- Let's use Gradient Descent to solve this problem.

Aside: Gradient Descent

- Generic algorithm to minimize a continuous objective function.
- In our case, loss function is the objective.
- Key idea: take steps proportional to the negative of the gradient at the current point.
- Key equation:
  - $w_{t+1} = w_t - \eta \cdot \nabla L(w_t)$, where η is the learning rate
- Essentially, the main thing we require now
  - gradient of the loss function with respect to the weights ($\nabla L(w_t)$)

https://bit.ly/2oKFTsM

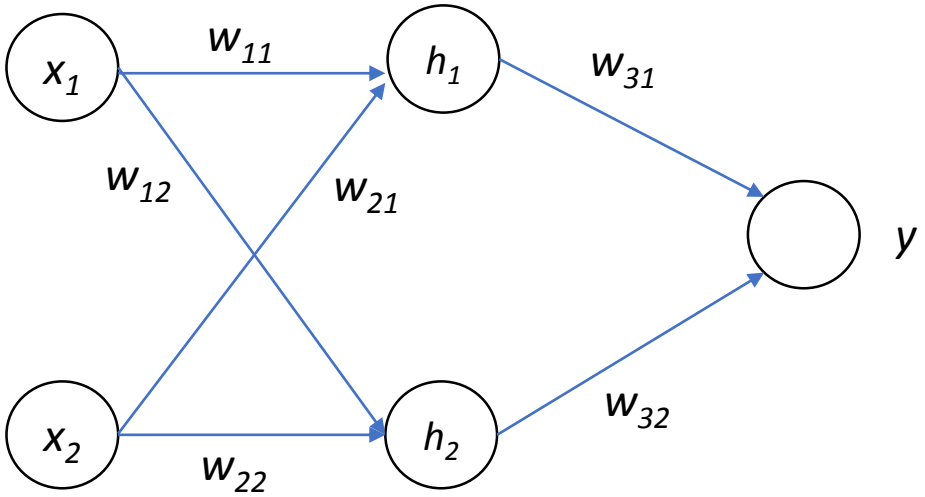
How do we compute the gradients?

Backpropagation

- Efficient algorithm to compute gradients w.r.t weights
- Key Idea: Chain Rule of Calculus
- Central algorithm for training neural networks
- Let’s consider a simple example to illustrate the idea

https://bit.ly/2FS5wPF

Backprop example on 1 hidden layer neural network

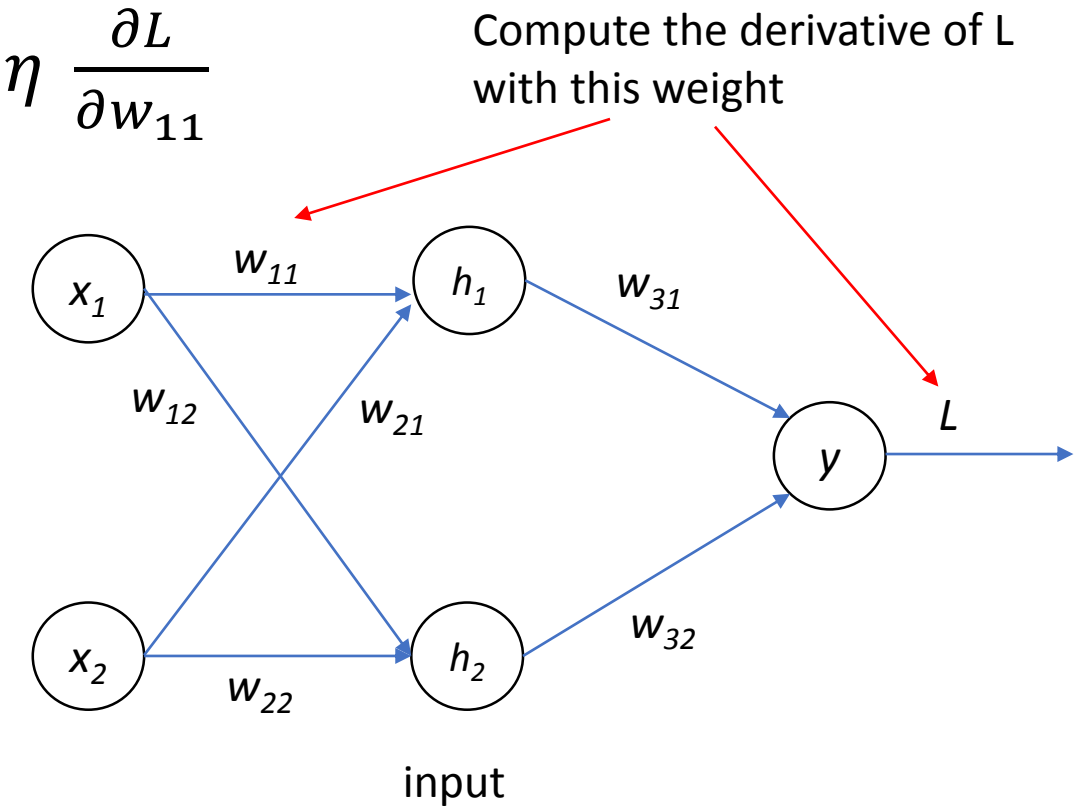


- $y = \sigma(w_{31}h_1 + w_{32}h_2)$ where $h_1 = \sigma(w_{11}x_1 + w_{21}x_2)$ and

$$h_2 = \sigma(w_{12}x_1 + w_{22}x_2)$$

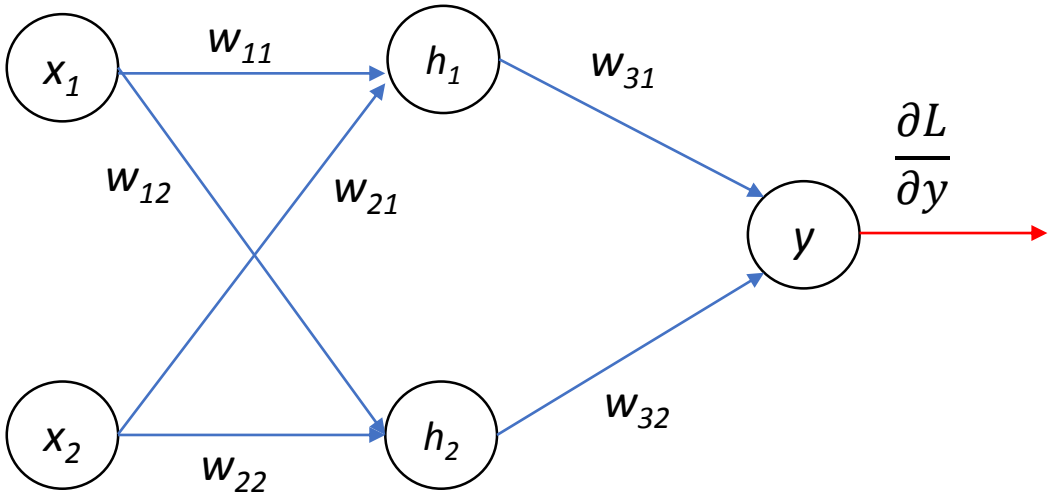
Backpropagation Example

- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



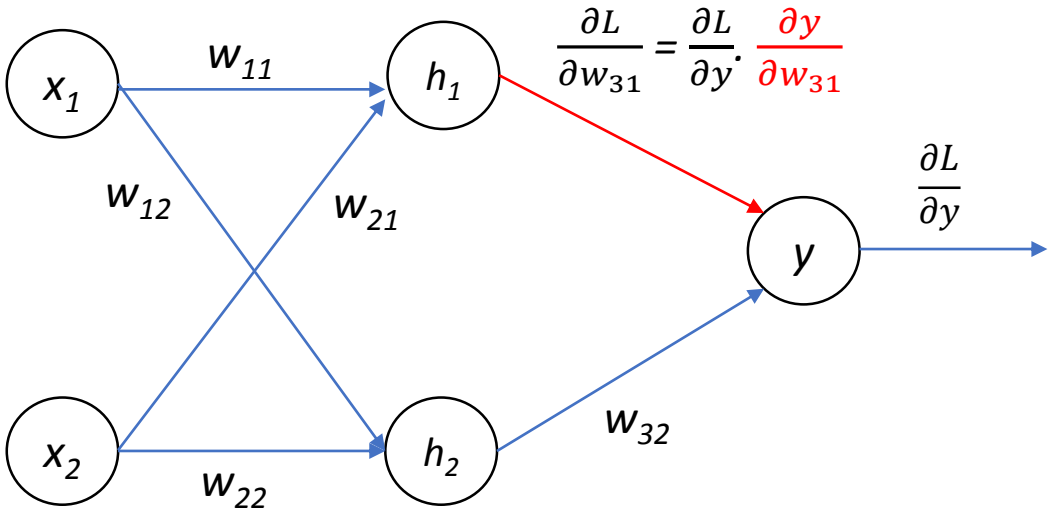
Backpropagation Example

- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



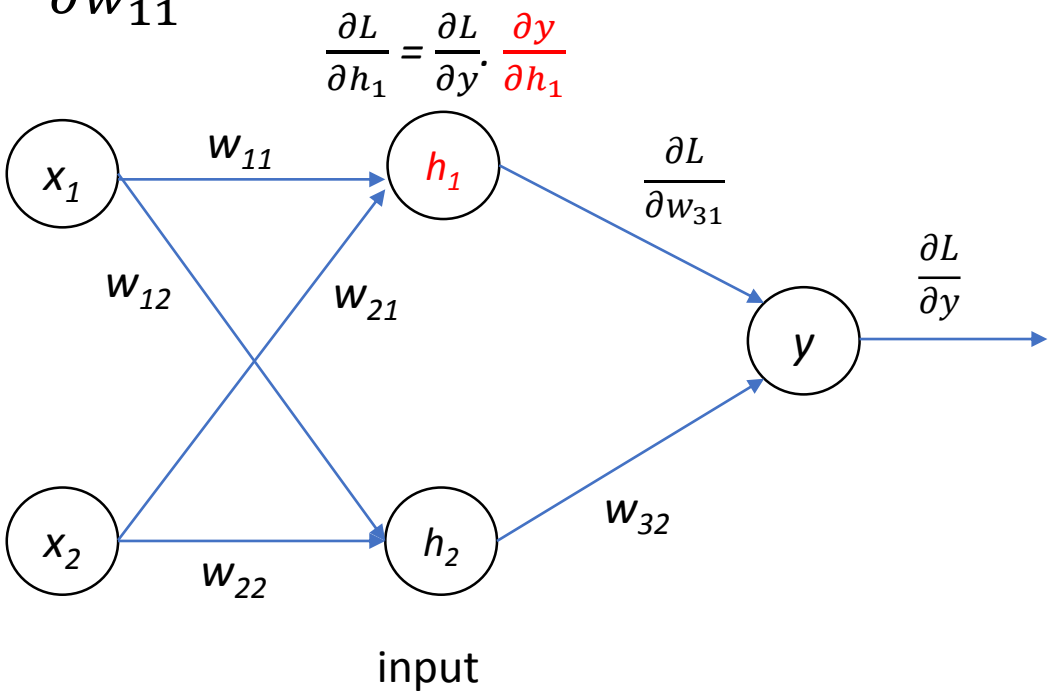
Backpropagation Example

- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



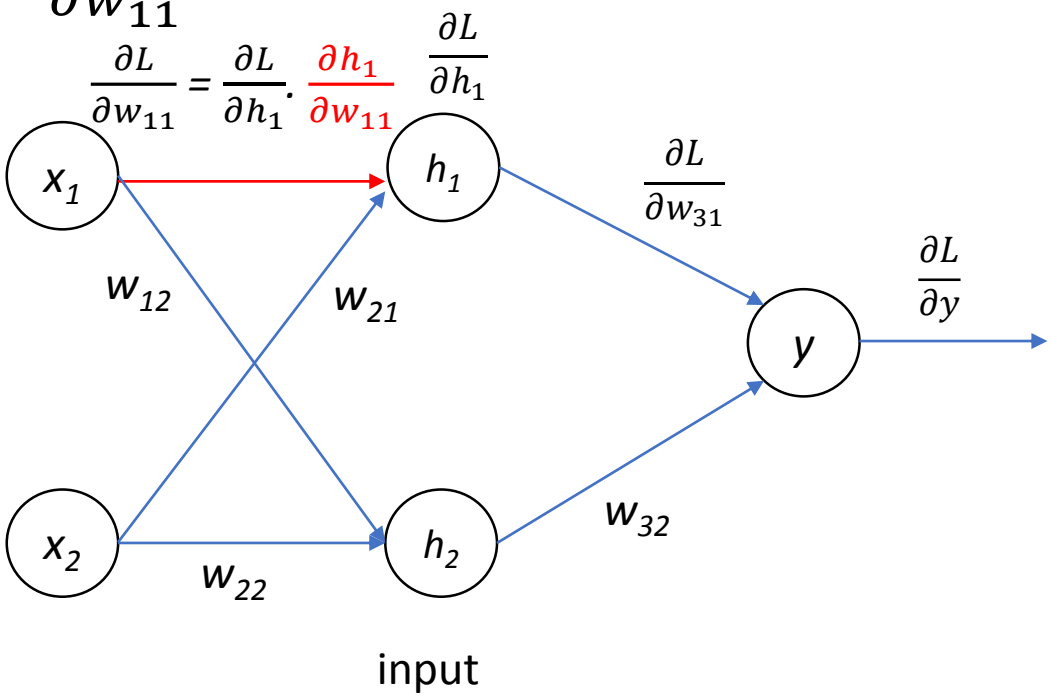
Backpropagation Example

- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



Backpropagation Example

- $w_{11}^{\{t+1\}} = w_{11}^{\{t\}} - \eta \frac{\partial L}{\partial w_{11}}$



Overall Training

- Same procedure can be applied to other weights
- Overall Training of Neural Networks:
  - *Till convergence (for e.g. when validation loss stops decreasing), repeat:*
    - *Forward Pass - Compute the predictions*
    - *Compute training loss*
    - *Backward Pass – Compute the gradients*
    - *Update the weights with gradient descent equation*

Are you wondering that you need to calculate all these gradients by hand for each weight?

Answer is No!

Automatic Differentiation is the solution

Auto differentiation

- General way of taking a program which computes a value, and automatically constructing a procedure for computing gradients of that value.
  - For e.g. gradients of loss function.
- Implementing backprop by hand is like programming in assembly language.
  - You'll probably never do it, but it's important for having a mental model of how everything works
- Most of the deep learning libraries (PyTorch, TensorFlow) have this functionality as default

Roger Grosse and Jimmy Ba notes

Auto differentiation

- An autodiff system will convert the program into a sequence of primitive operations (ops) which have specified routines for computing derivatives.
- In this representation, backprop can be done in a completely mechanical way.

Original program:

$$z = wx + b$$
$$y = \frac{1}{1 + \exp(-z)}$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Sequence of primitive operations:

$$t_1 = wx$$
$$z = t_1 + b$$
$$t_3 = -z$$
$$t_4 = \exp(t_3)$$
$$t_5 = 1 + t_4$$
$$y = 1/t_5$$
$$t_6 = y - t$$
$$t_7 = t_6^2$$
$$\mathcal{L} = t_7/2$$

You can create all kinds of functions by combining these primitive operations

Roger Grosse and Jimmy Ba notes

Implementation issues of neural networks

- Initialization
- Learning Rate
- Stochastic Gradient Descent
- Batch size
- Architecture
- Type of optimizer
- Numerical Stability

Inductive Biases and Architectures

- What is the general problem in machine learning?
- Construct algorithms that *learn* to predict a certain target output.
- How do we want the learner to achieve it?
- Learner is shown some examples for training and
- Asked to generalize even for examples not seen in training.
- Without assumptions, this problem is impossible to solve.
- These necessary assumptions are called inductive biases.

https://www.cs.cmu.edu/~tom/pubs/NeedForBias_1980.pdf

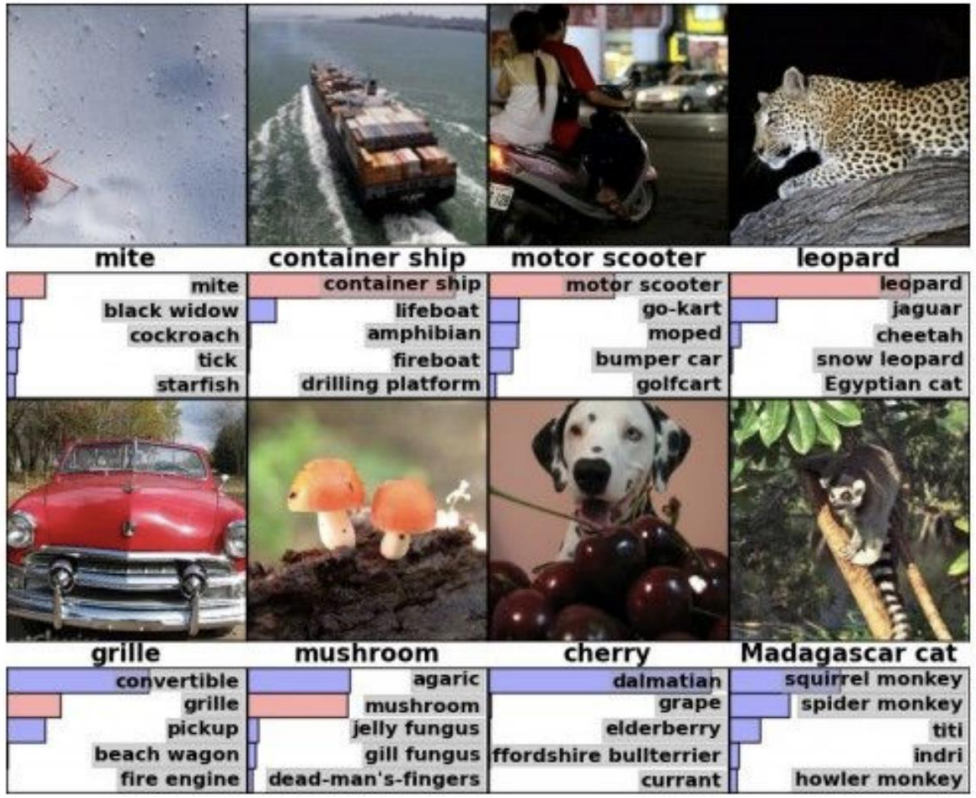
Inductive Biases and Architectures

- Similarly, Neural networks are a great tool!
- But they are very generic
- We need to add inductive biases in these models
- How do we do that?
- Different types of architectures!
- For e.g. Convolutional architecture for vision
- For e.g. Recurrent architecture for sequences, …

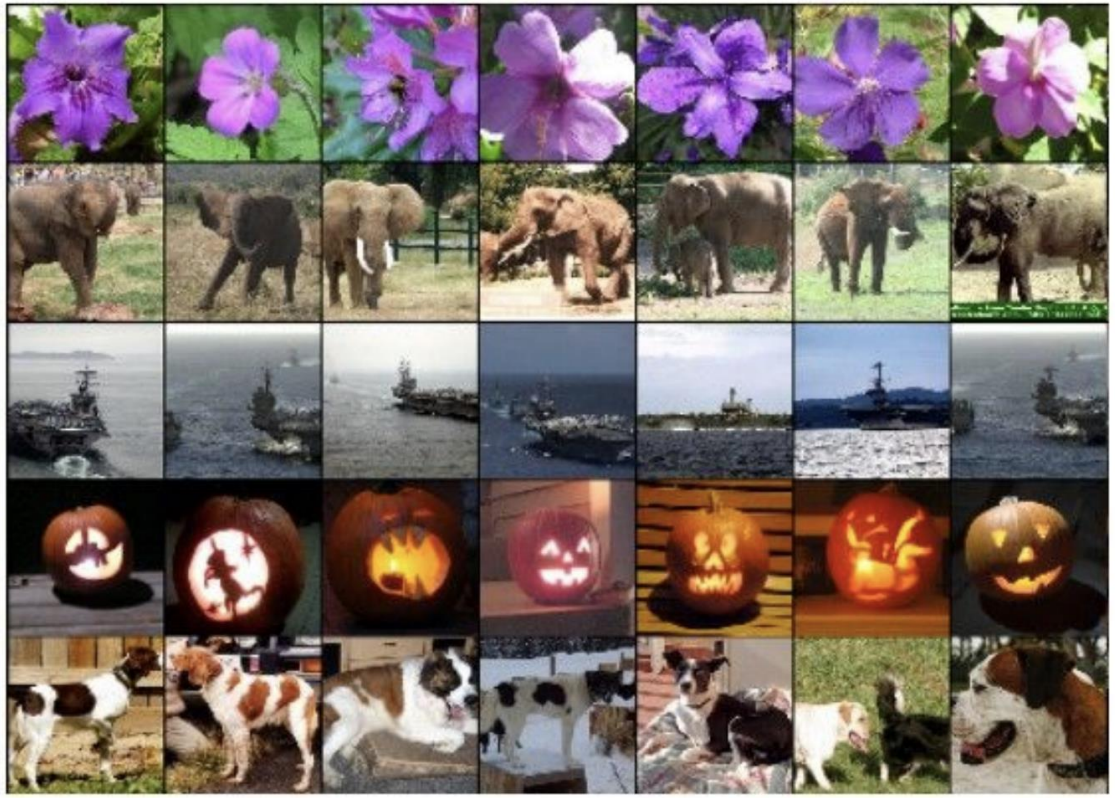
Convolutional Neural Networks

CNNs are everywhere

Classification



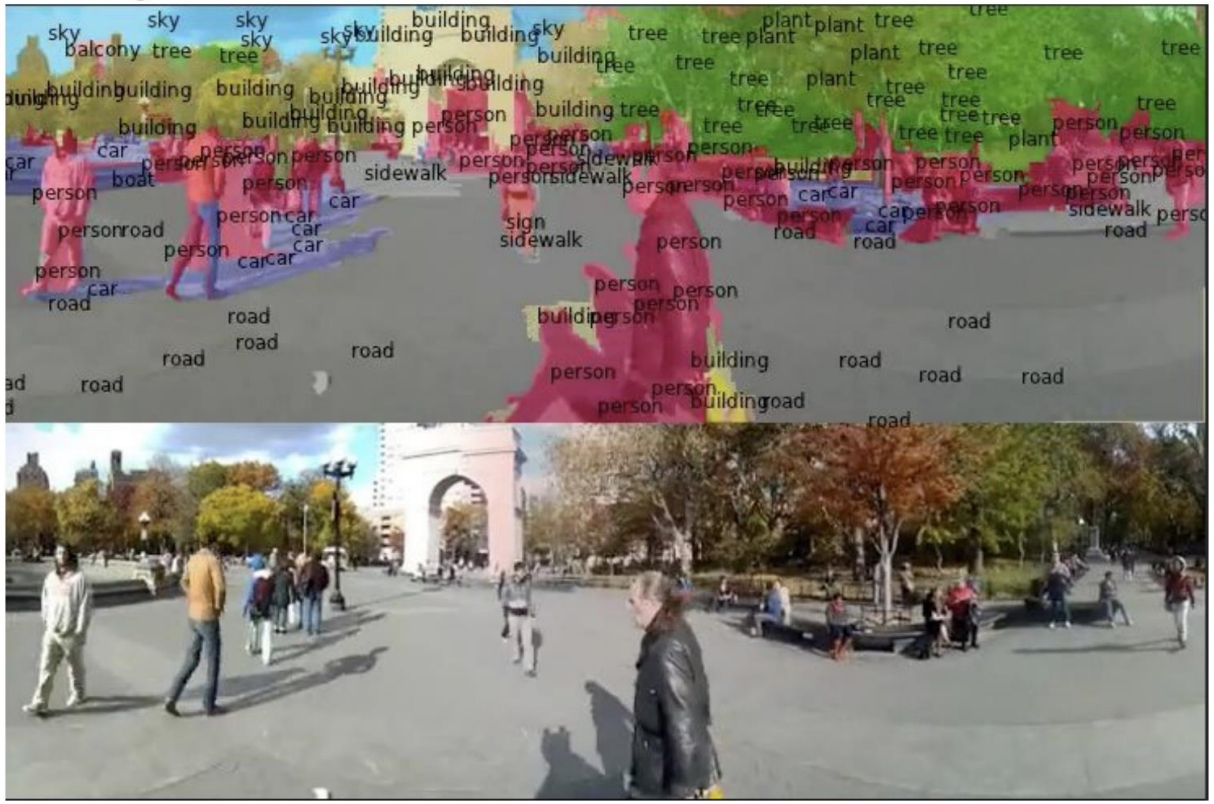
Retrieval



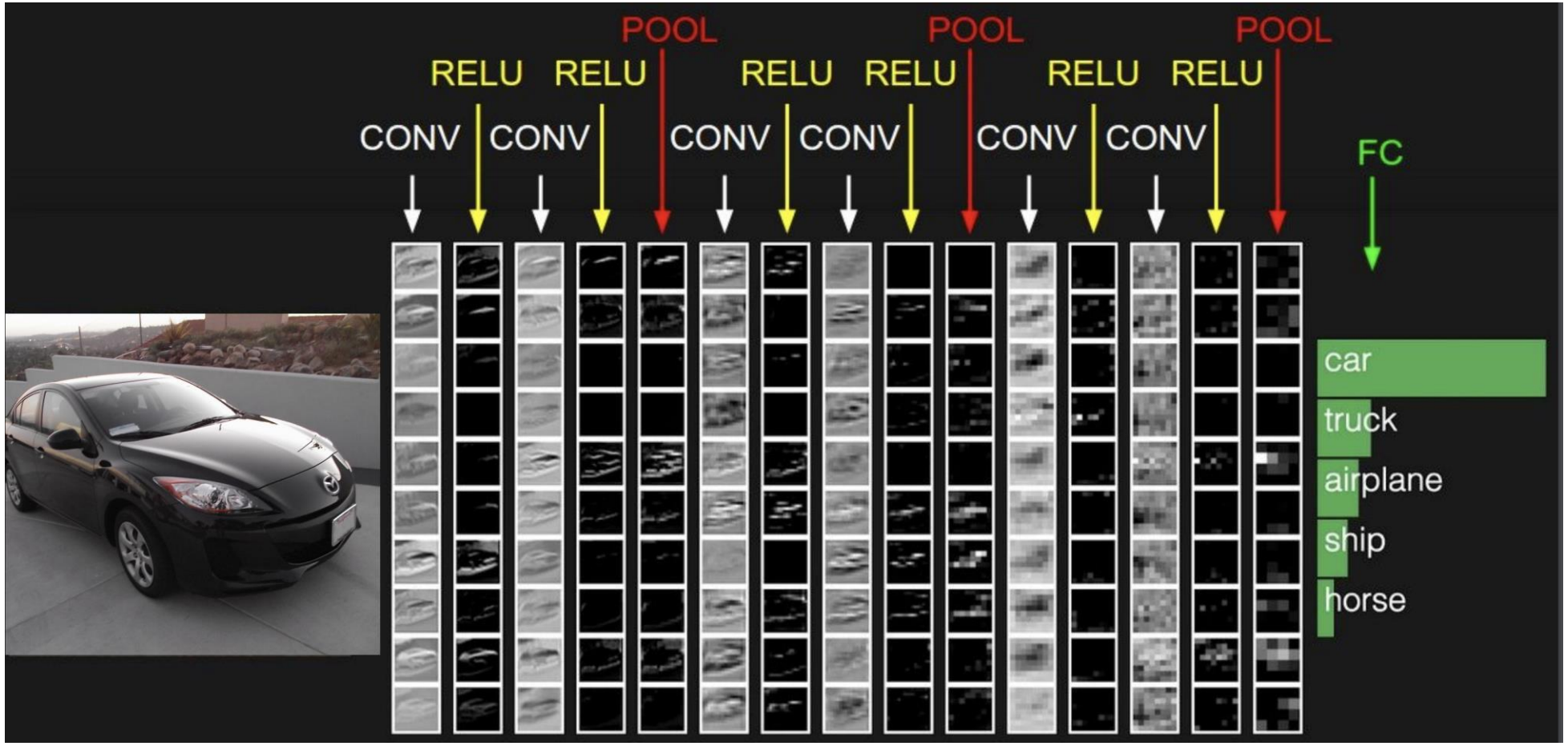
CNNs are everywhere

Detection

Segmentation



This is what a CNN looks like

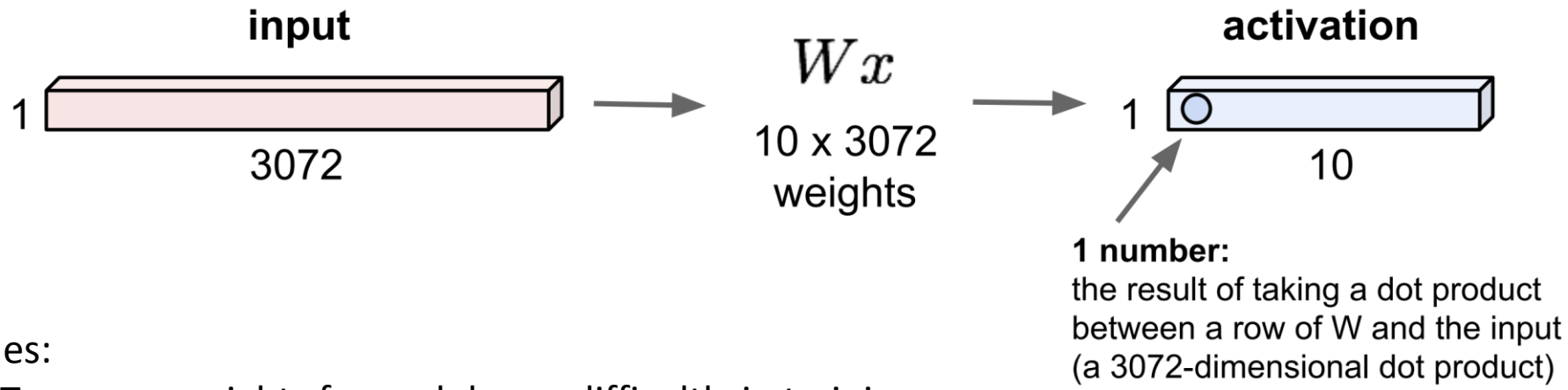


We will talk about each layer in detail shortly!

Image Classification with Vanilla Neural networks

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Issues:

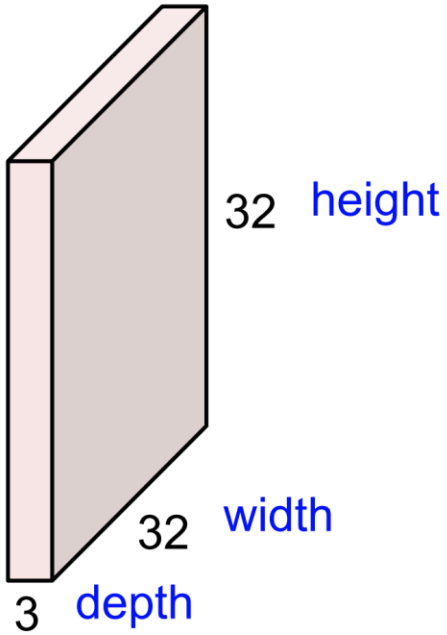
1. Too many weights for each layer; difficultly in training
2. Doesn't take into account any structure in the image

Architecture for vision based tasks

- What kind of assumptions do you think we need?
- The same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts as well.
- E.g., edges, corners, contours, object parts, …
- Let’s discuss a new layer: Convolution layer

Convolution Layer

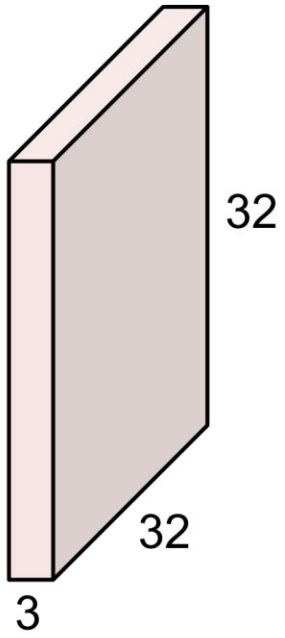
32x32x3 image -> preserve spatial structure



Convolution Layer

Convolution Layer

32x32x3 image



5x5x3 filter

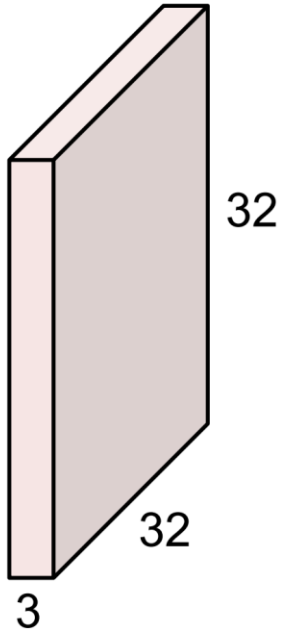
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

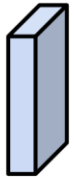
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



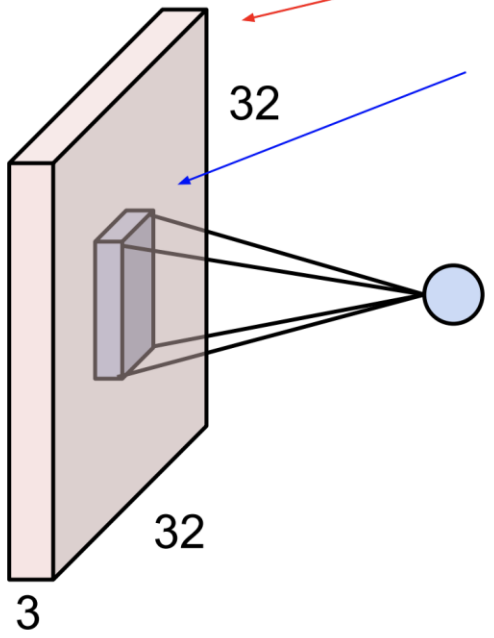
5x5x3 filter



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

Convolution Layer

Convolution Layer



32x32x3 image

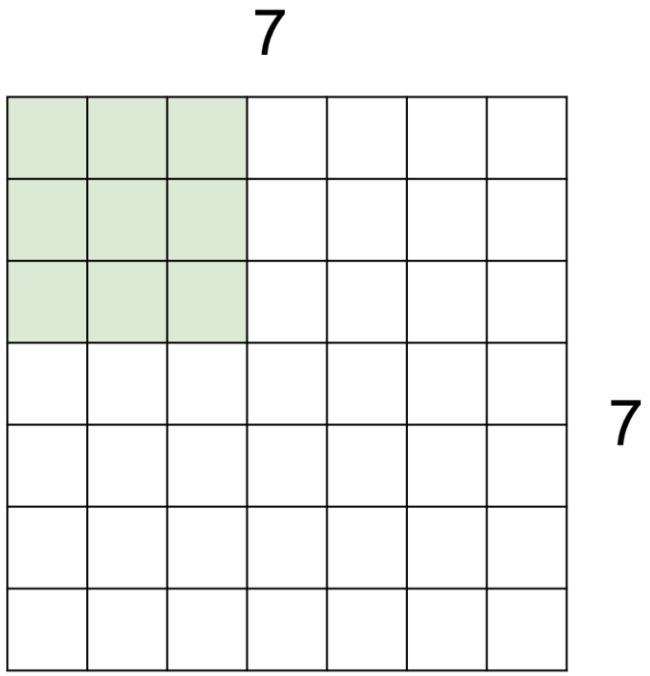
5x5x3 filter w

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)

$$w^T x + b$$

Example of Convolution Operation

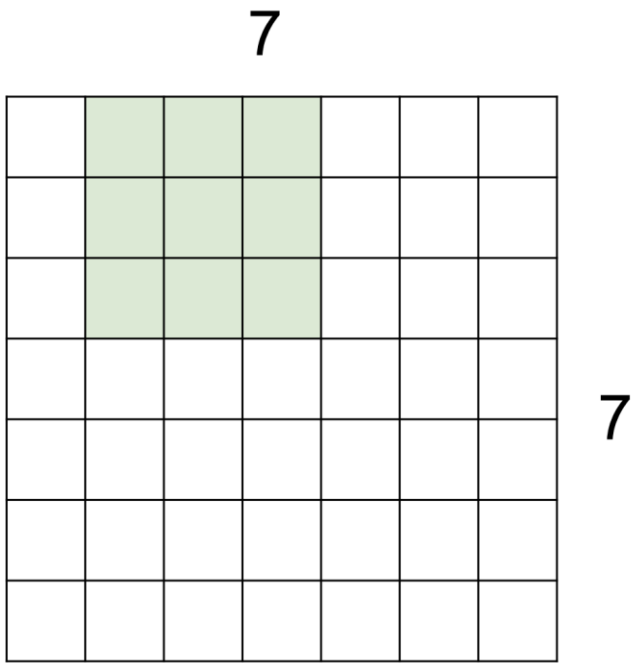
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Example of Convolution Operation

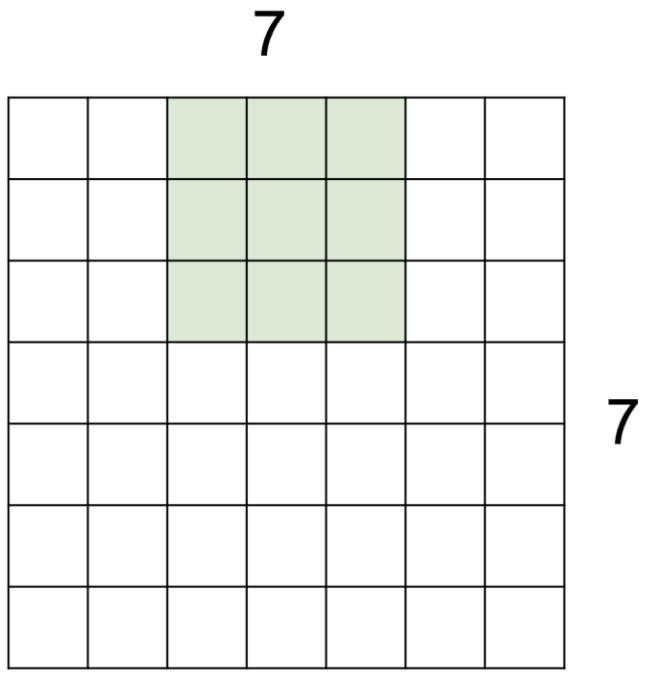
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Example of Convolution Operation

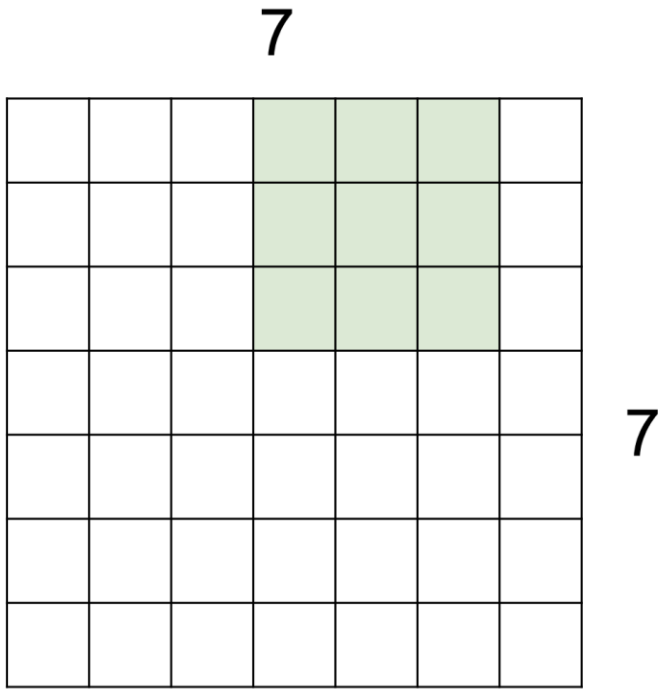
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Example of Convolution Operation

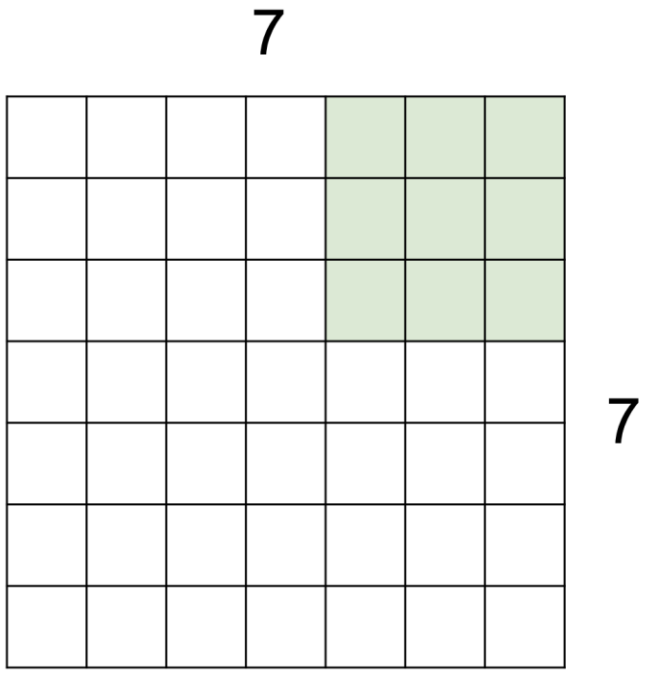
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Example of Convolution Operation

A closer look at spatial dimensions:

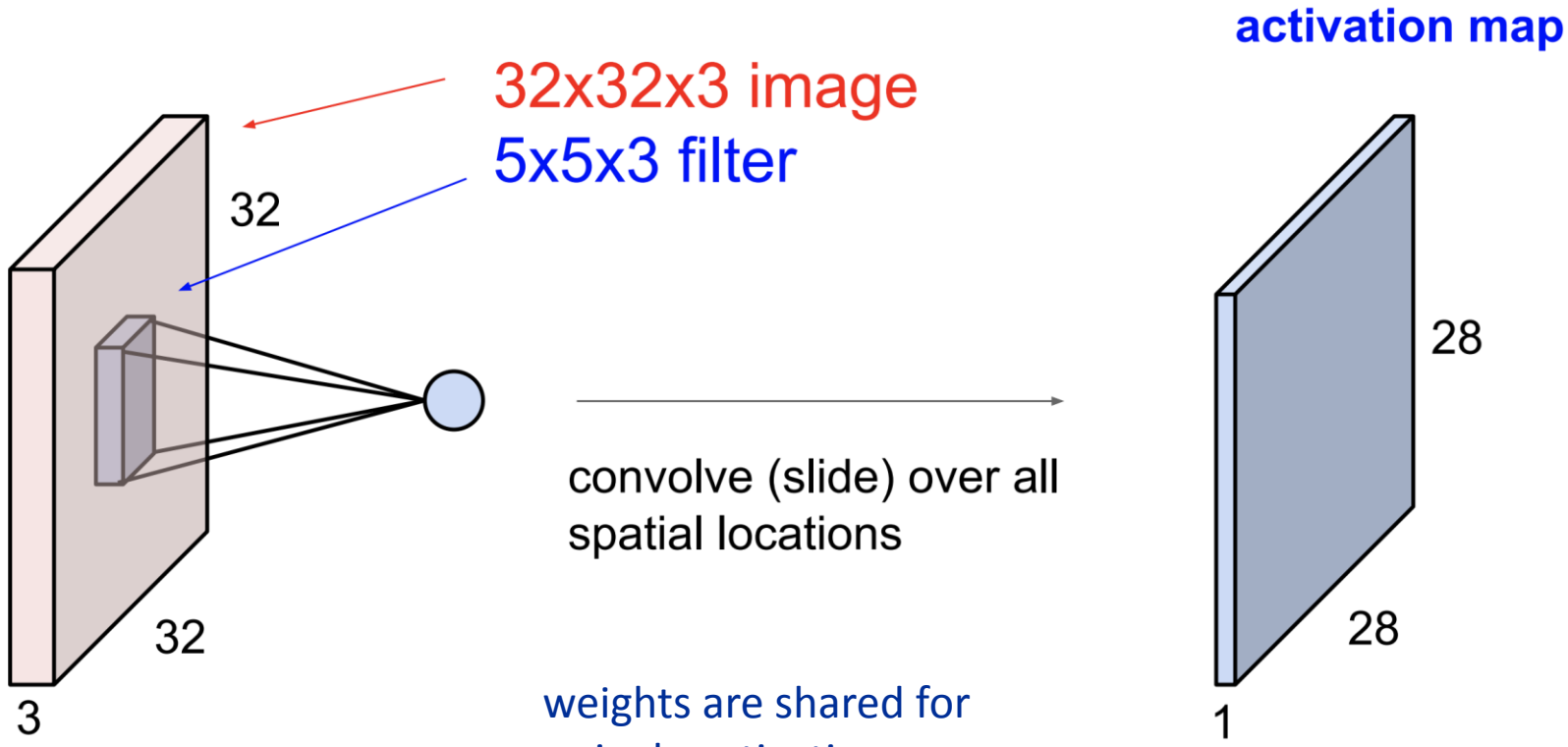


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Convolution Layer

Convolution Layer



weights are shared for
a single activation map

Convolution Layer

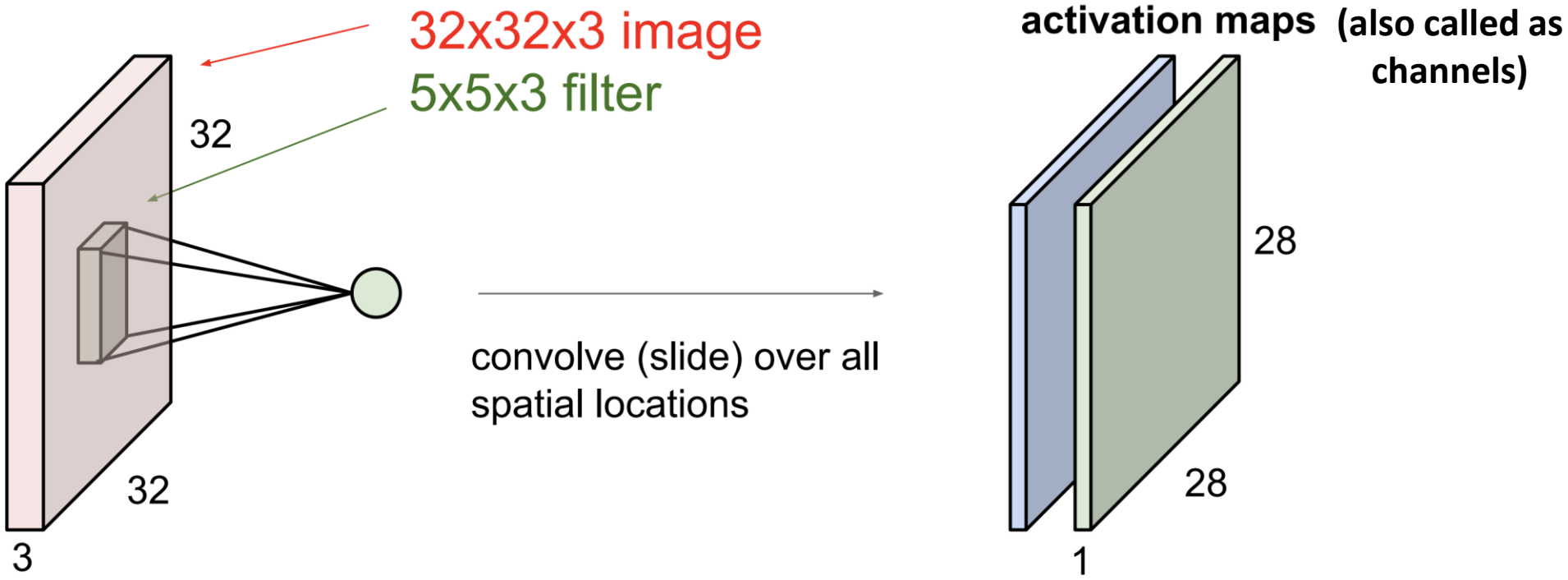
Convolution Layer

consider a second, green filter

Convolution Layer

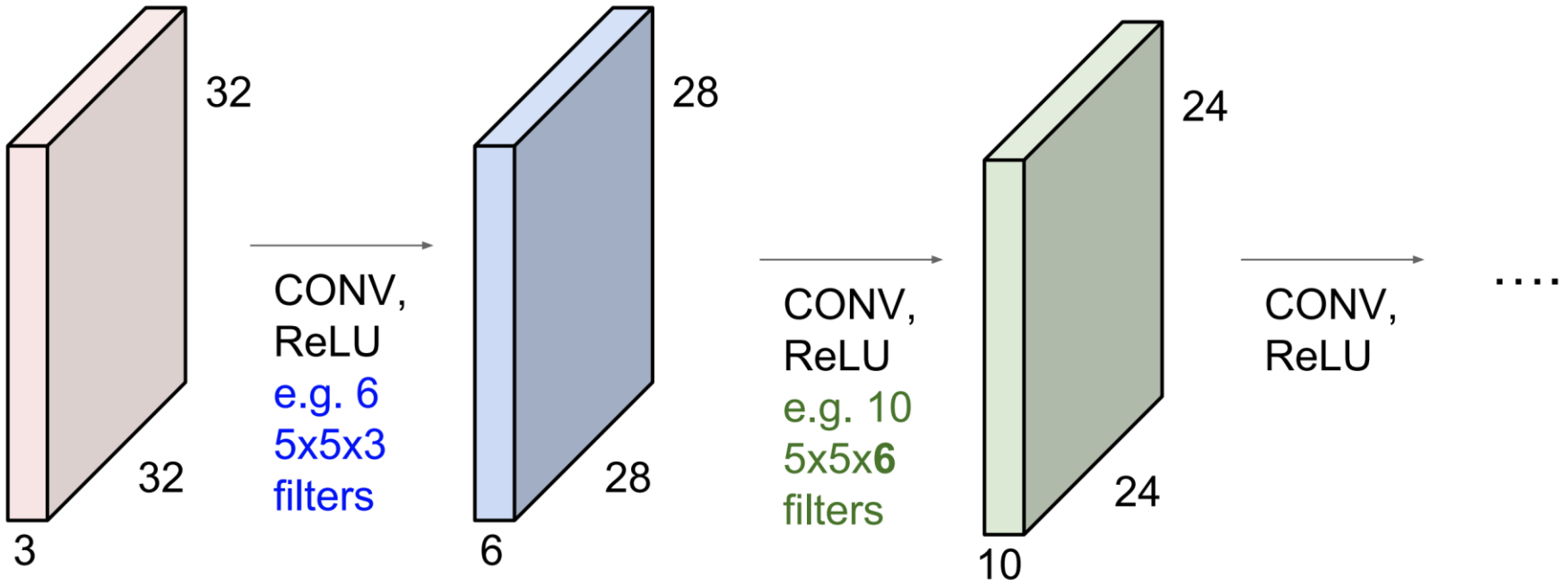
Convolution Layer

consider a second, green filter



ConvNet

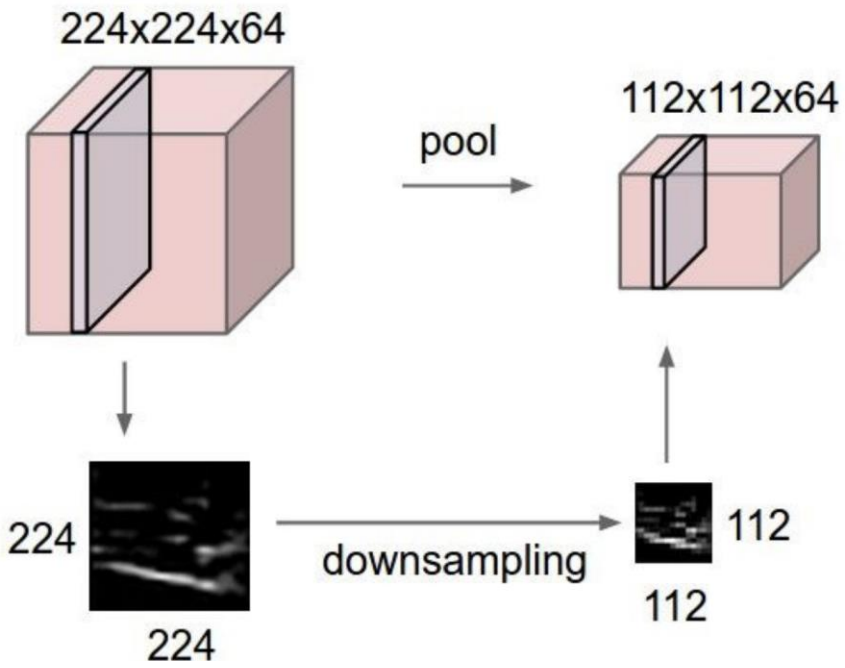
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



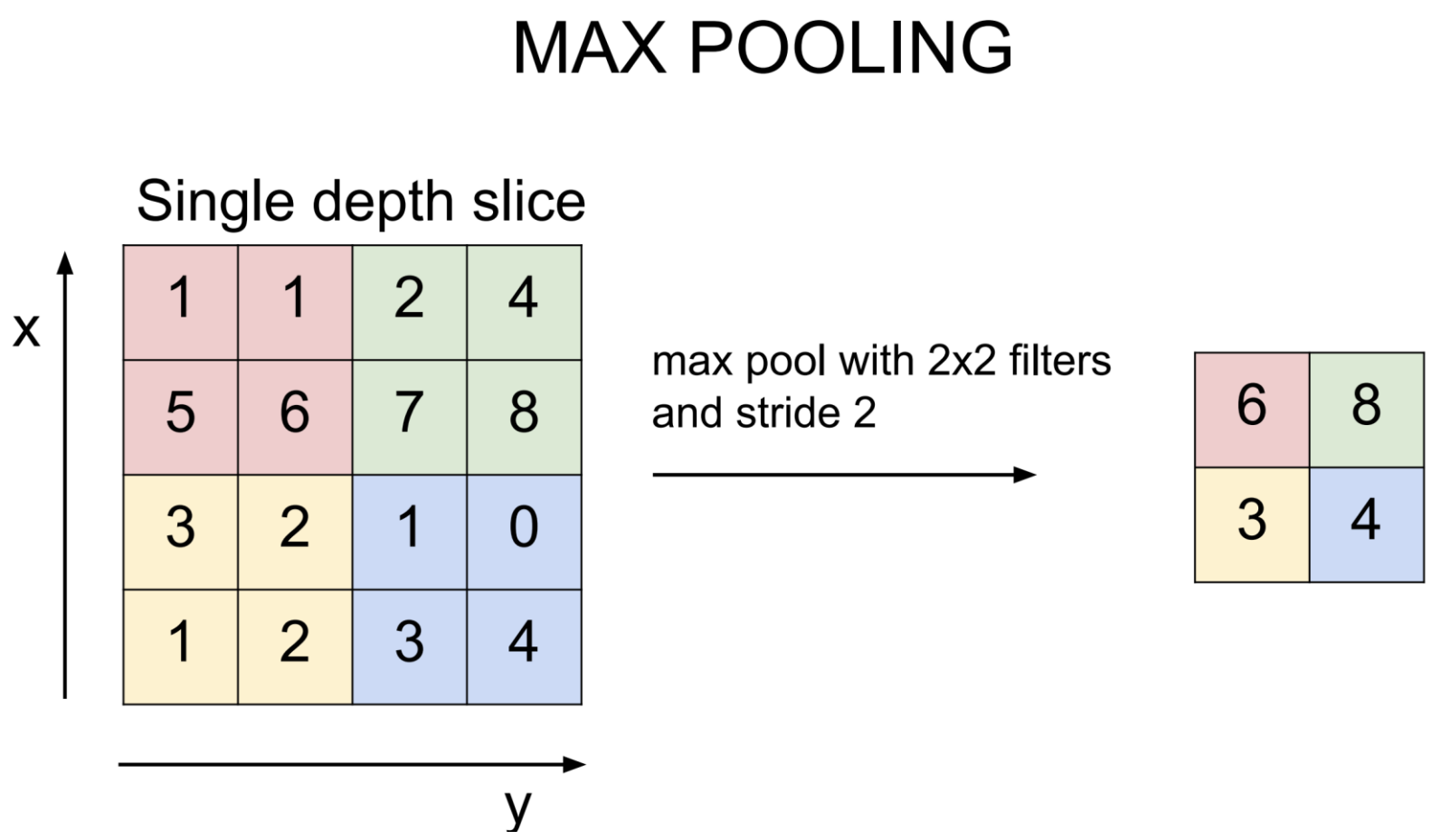
Pooling Layer

Pooling layer

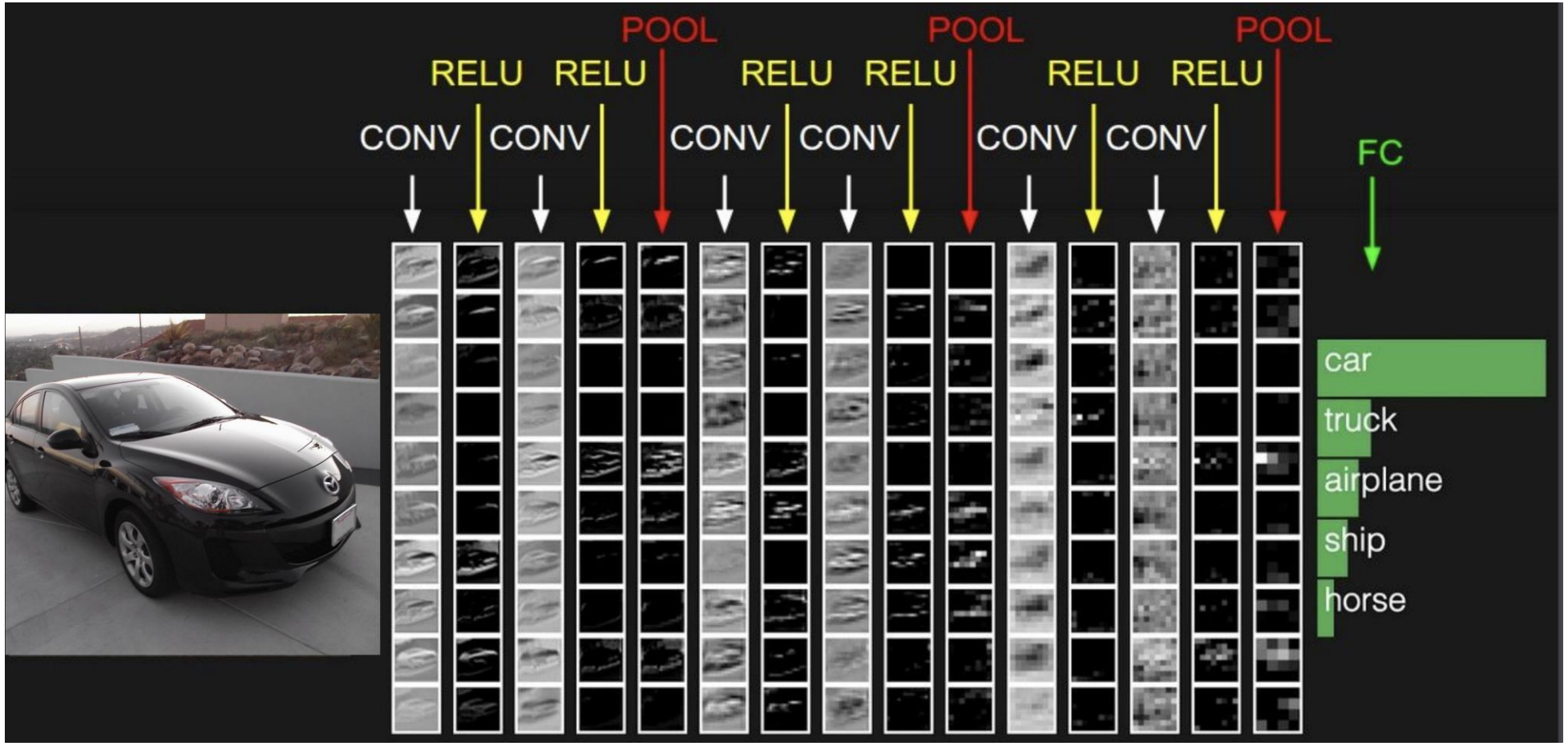
- makes the representations smaller and more manageable
- operates over each activation map independently:



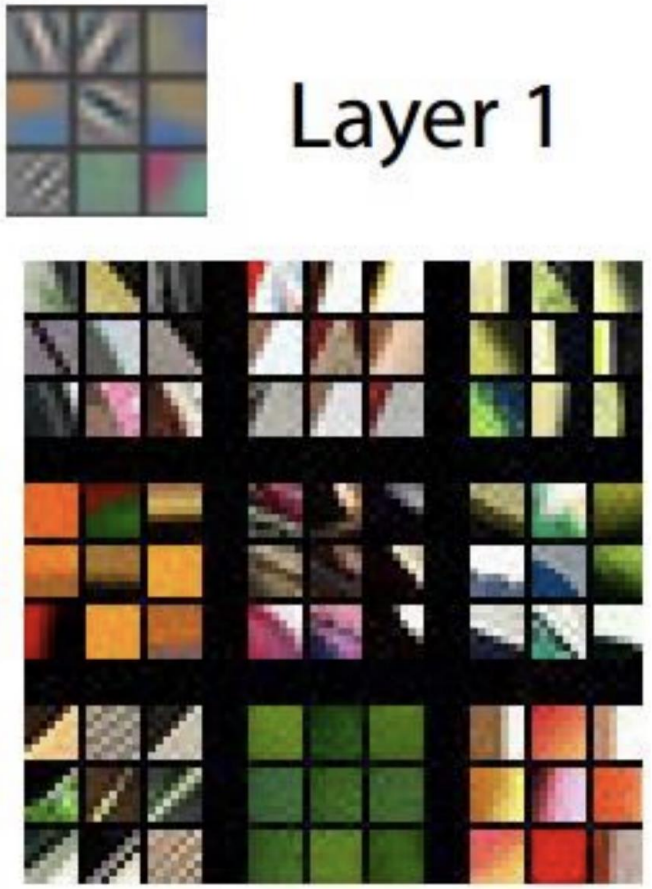
One example of Pooling



Convolutional Neural Network (CNN)

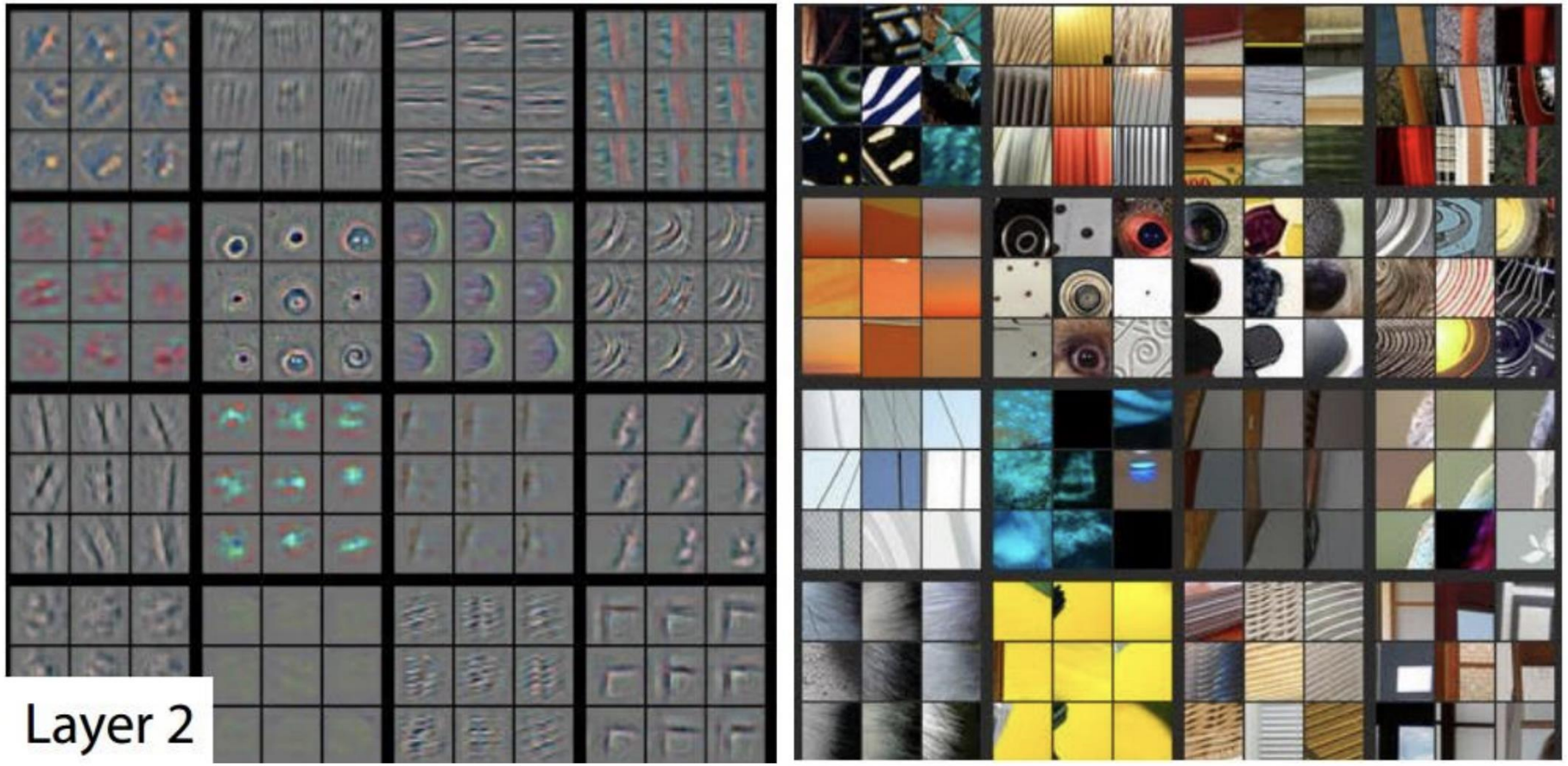


Visualizing what CNN learns

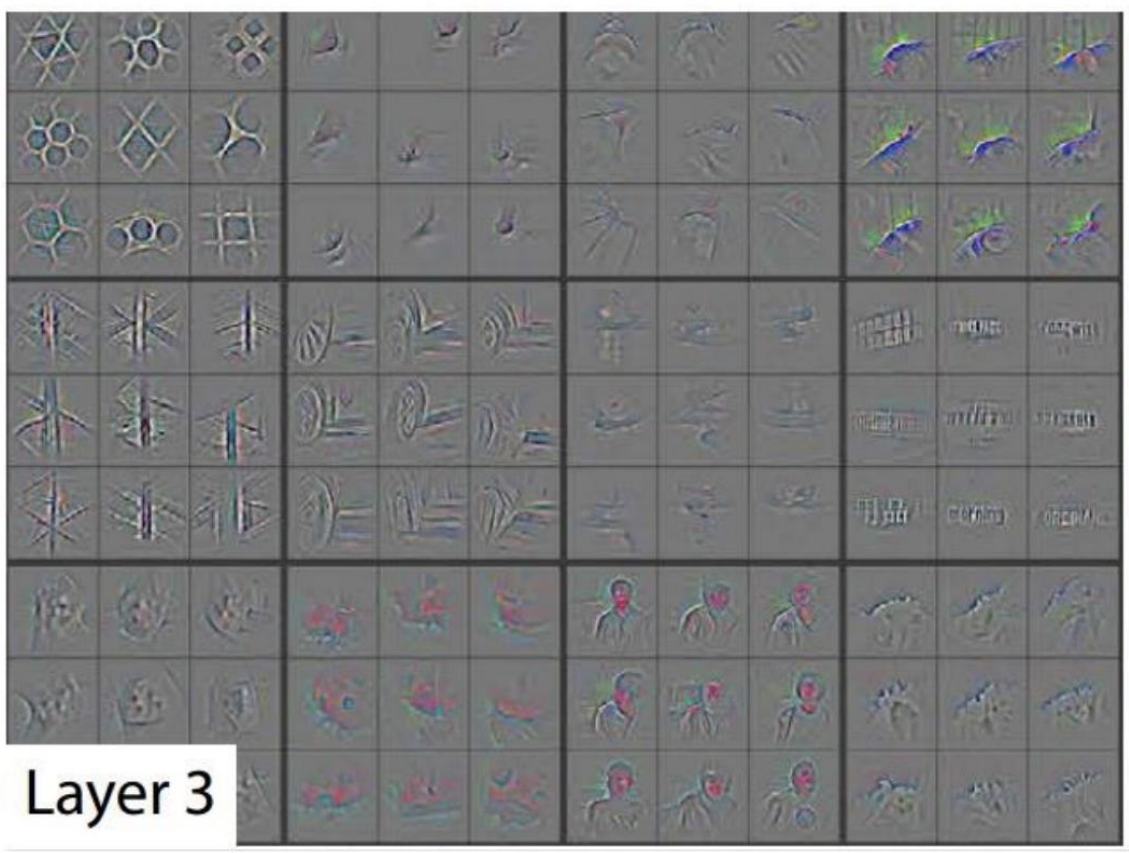


Increasingly complex features at each level
Starting from generic ones (lines, edges, colours etc.) at lower layers to specific ones at the higher layers!

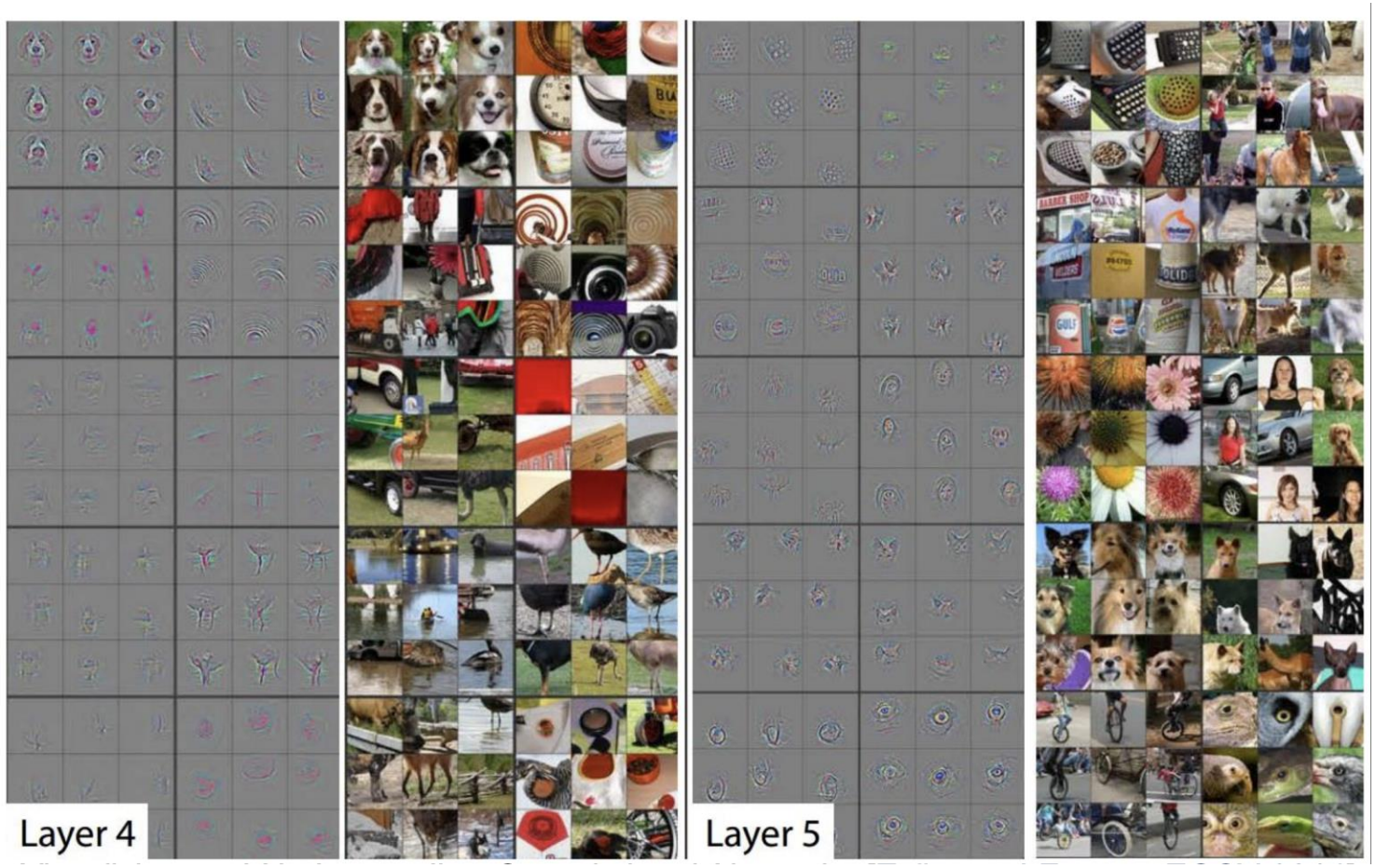
Visualizing what CNN learns



Visualizing what CNN learns



Visualizing what CNN learns



Sequence based Neural Networks

Applications of Sequence Models

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

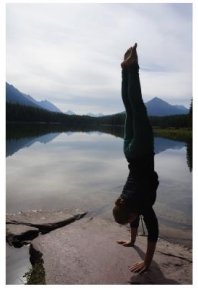
Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand



A woman standing on a beach holding a surfboard

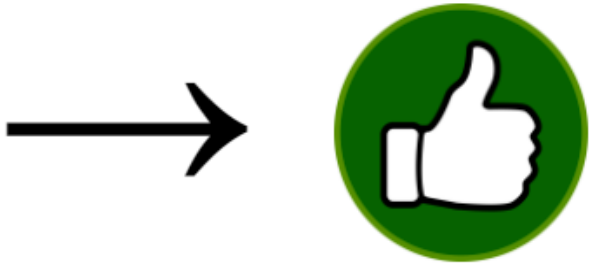
Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

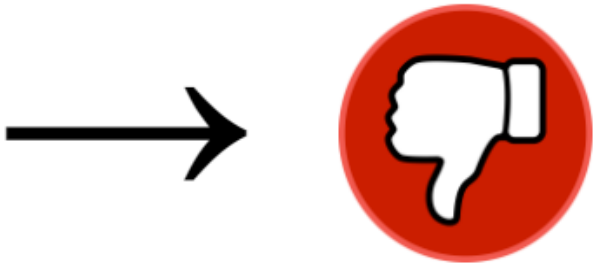
Captions generated by Justin Johnson using Neuraltalk2

Applications of Sequence Models

"I love this movie.
I've seen it many times
and it's still awesome."

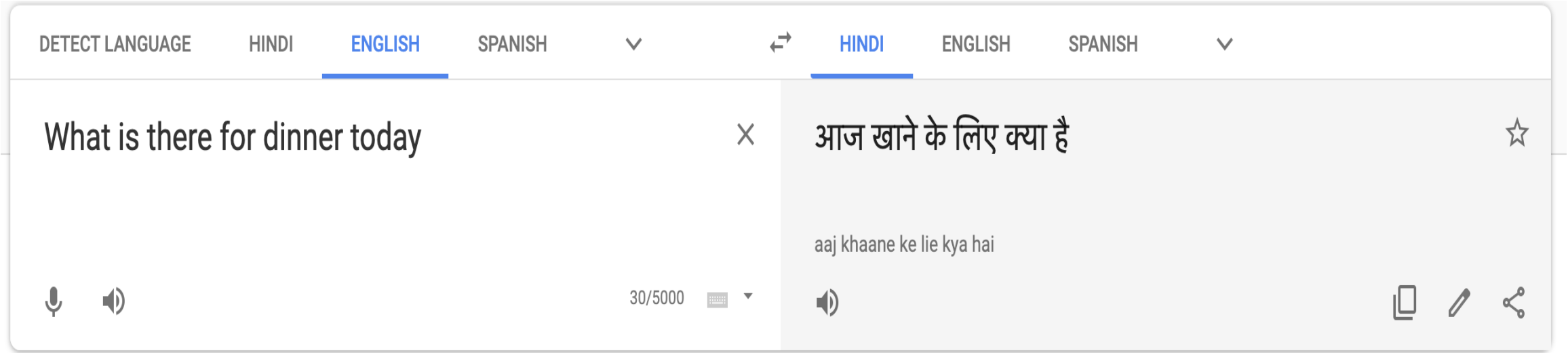


"This movie is bad.
I don't like it it all.
It's terrible."



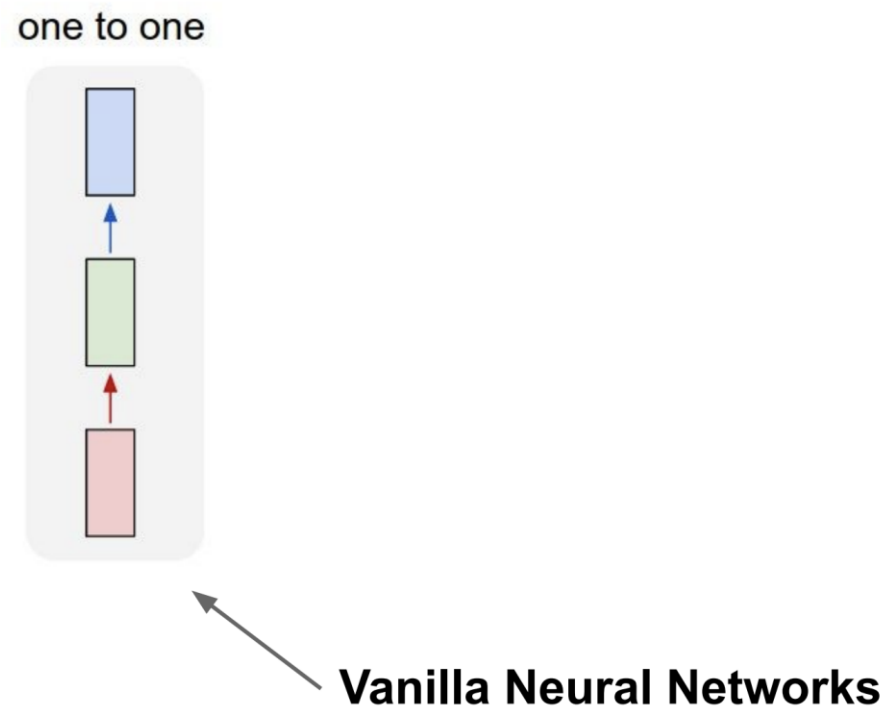
Sentiment
Classification

Applications of Sequence Models

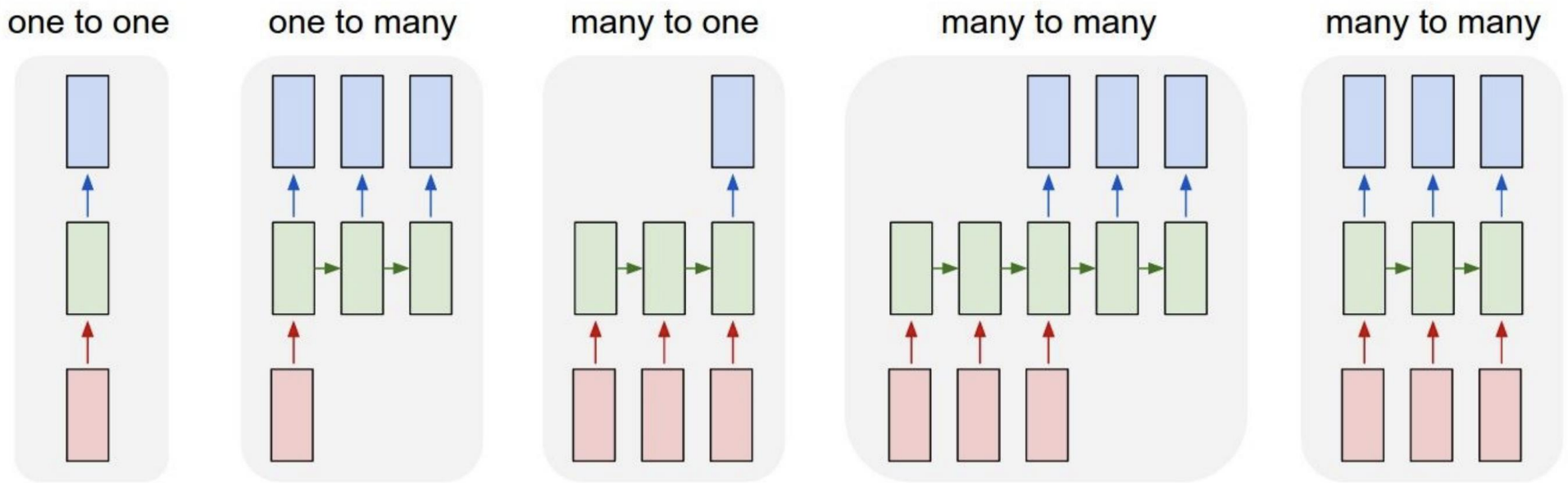


Machine Translation

“Vanilla” Neural Network

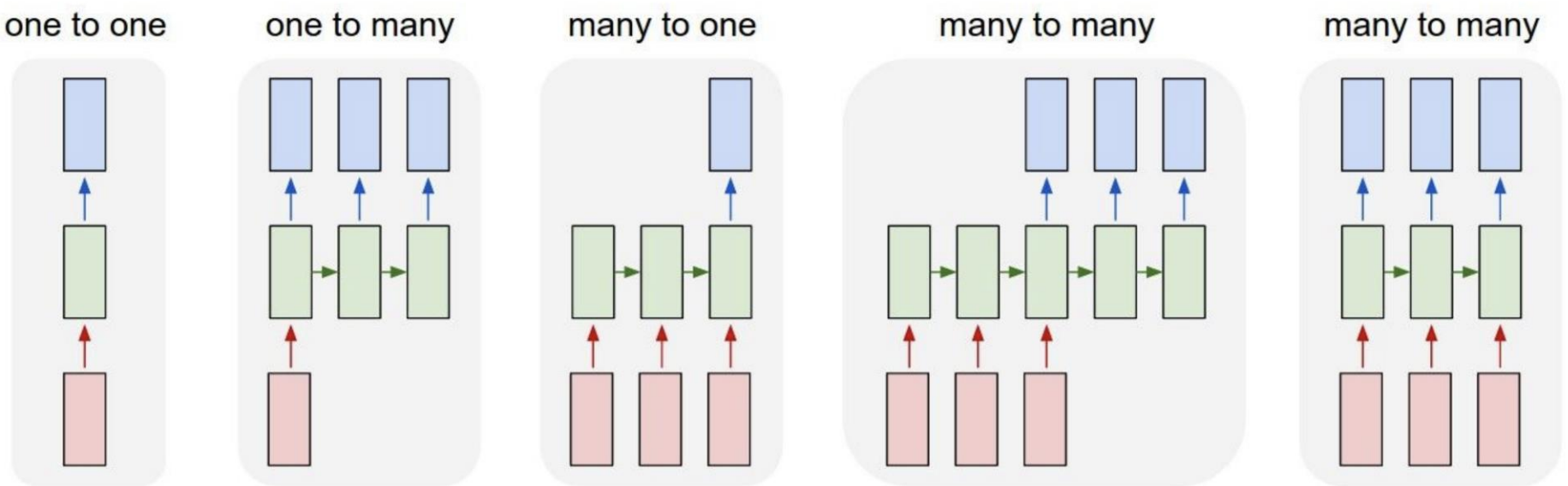


Recurrent Neural Networks: Process Sequences



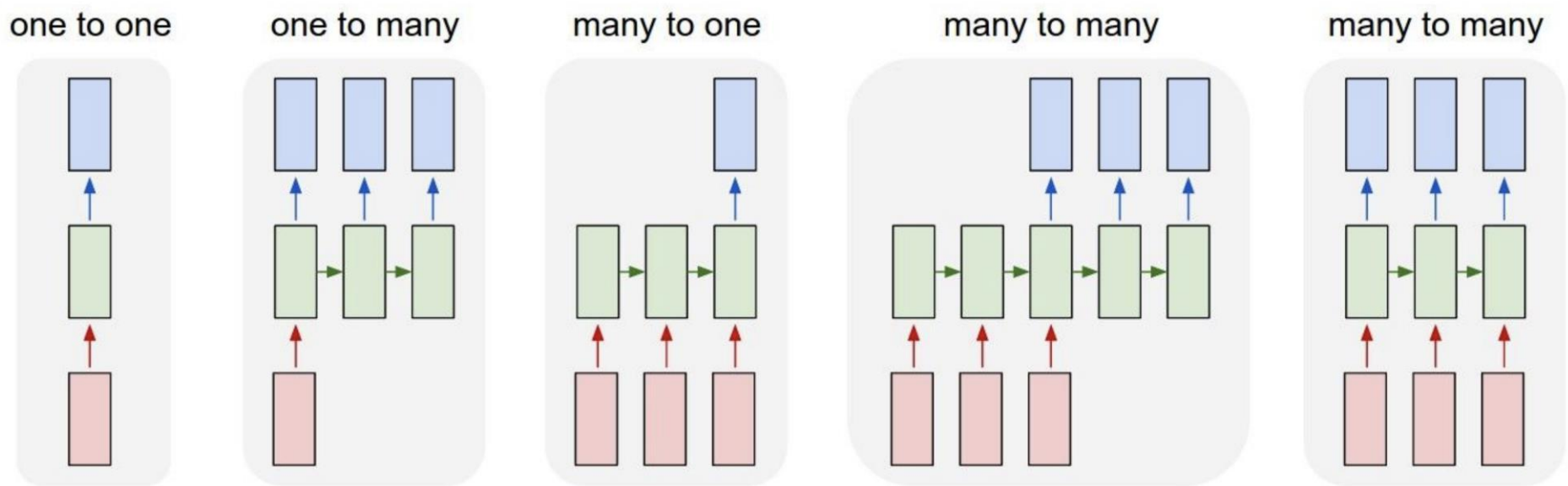
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



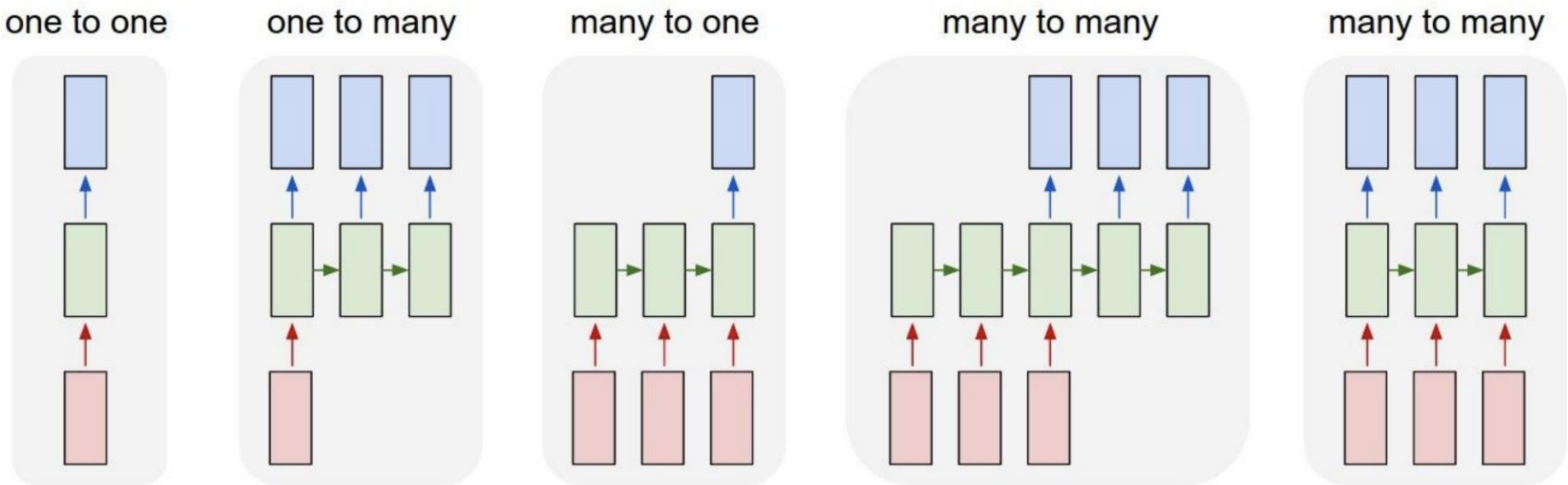
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Neural Networks: Process Sequences



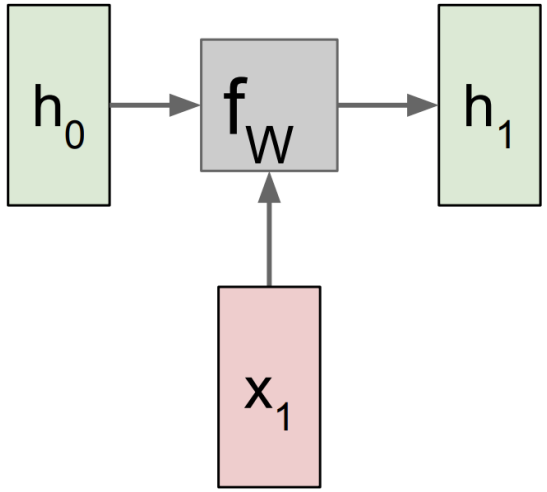
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Neural Networks: Process Sequences

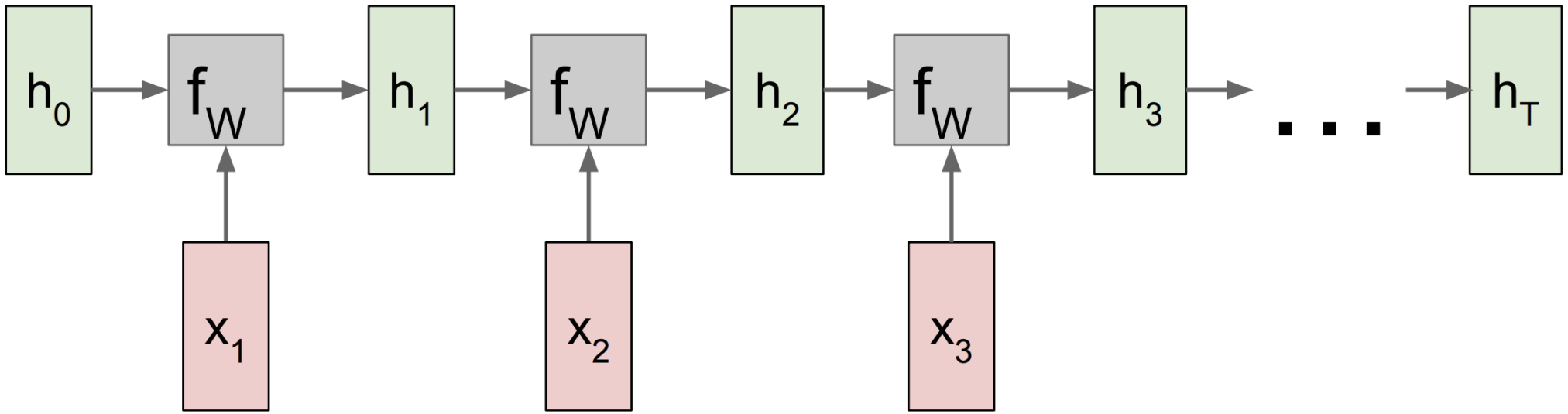


e.g. **Video classification on frame level**

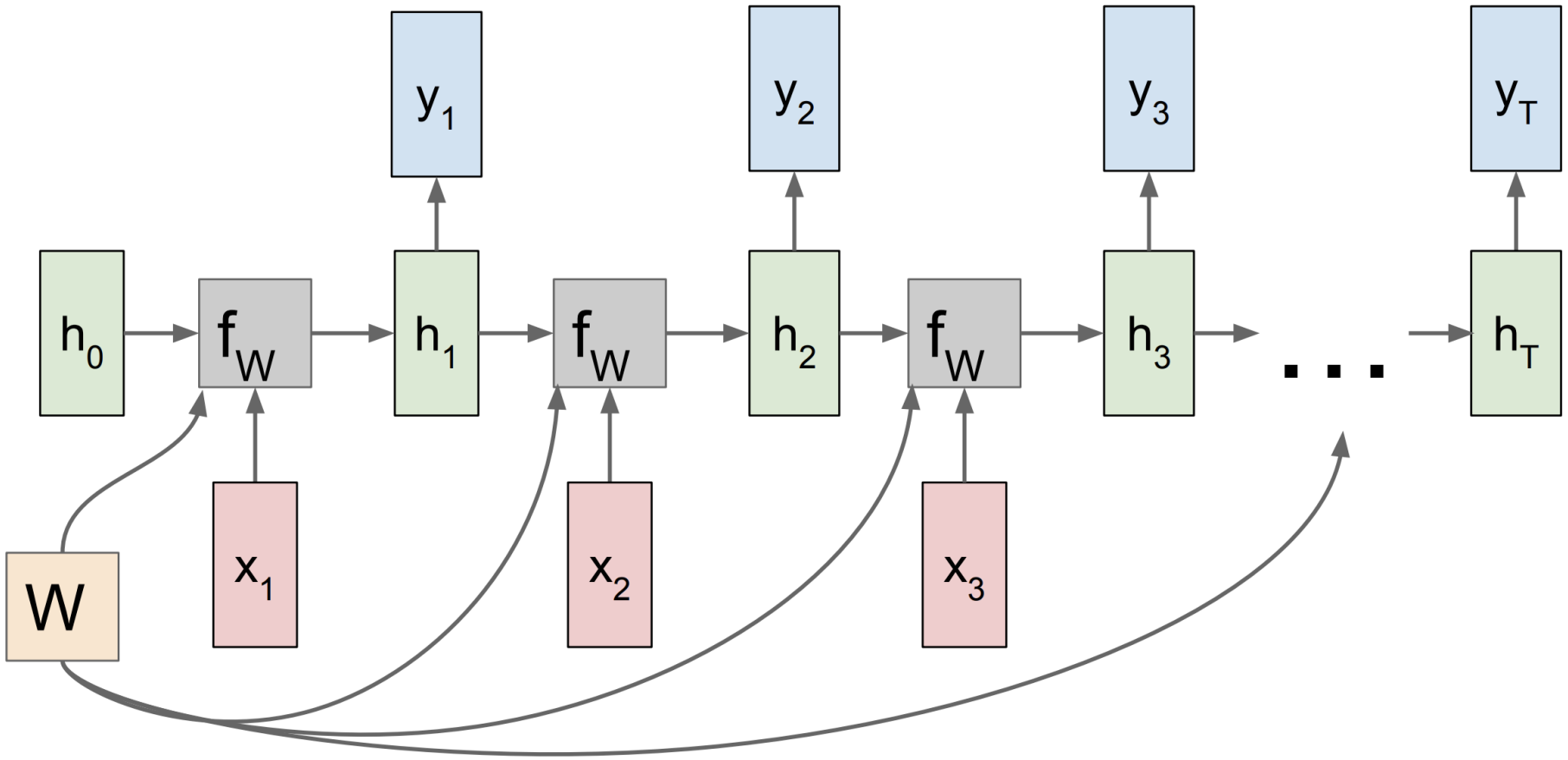
Basic (Vanilla) RNN architecture



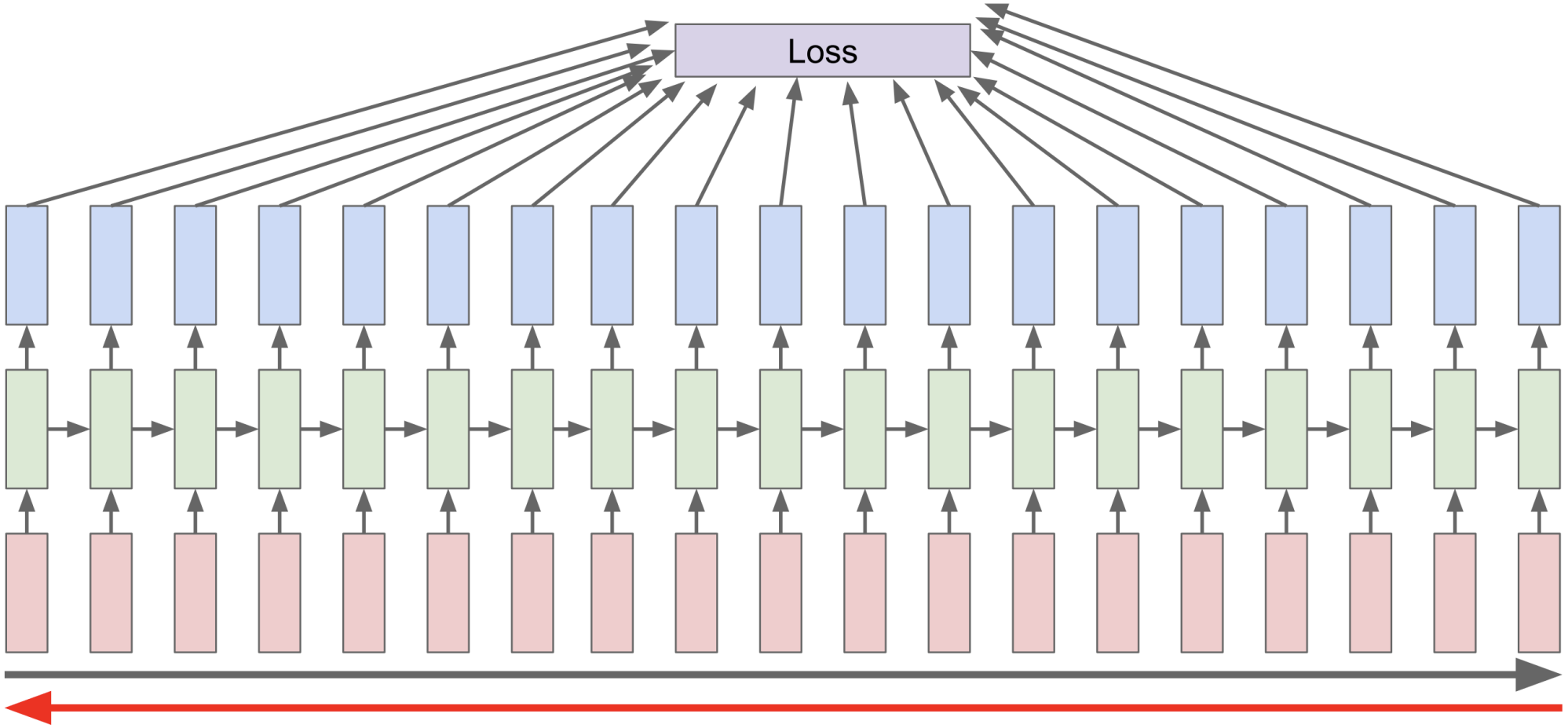
Basic RNN architecture



Basic RNN architecture



Backpropagation through time (BPTT)



Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

https://bit.ly/35jh0bg

Advanced Sequence Models

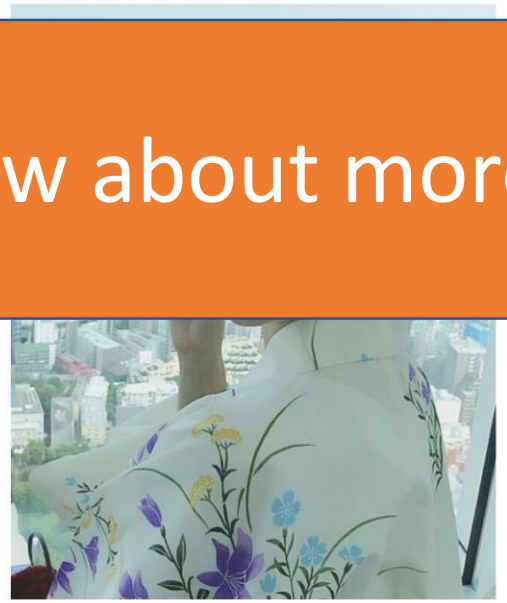
- Vanilla RNNs have almost fallen out of favor because of multiple issues with their training procedure
  - For e.g. Vanishing Gradients
- Multiple advanced techniques overcome these issues in different forms. Some of them are:
  - LSTMs, GRUs
  - Attention models

Advanced Techniques

Graph Neural Networks

Success of deep learning on grid/sequence structured data…

Images



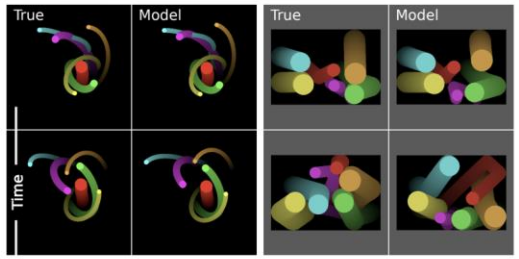
(Instagram:@kyokofukada_official)

Language

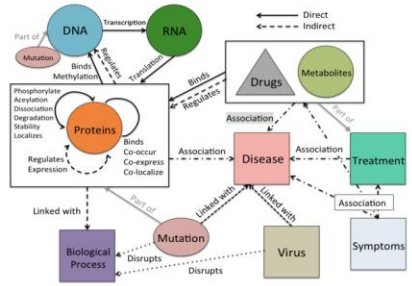
(Wikibooks)

How about more complex data

Graph Neural Networks (Motivation)

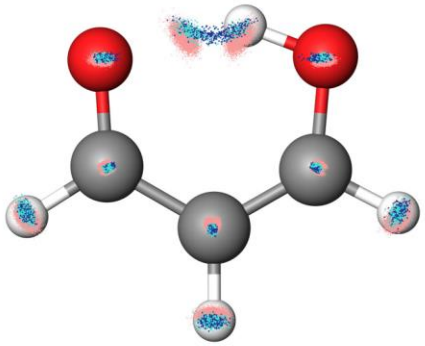


Intuitive Physics (Battaglia et al. 2016)



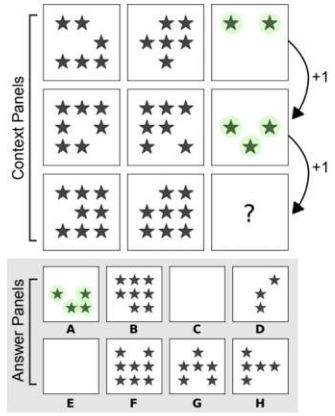
Biomedicine (Zitnik et al. 2018)

Complex graph data



Drug design (Duvenaud et al. 2015)

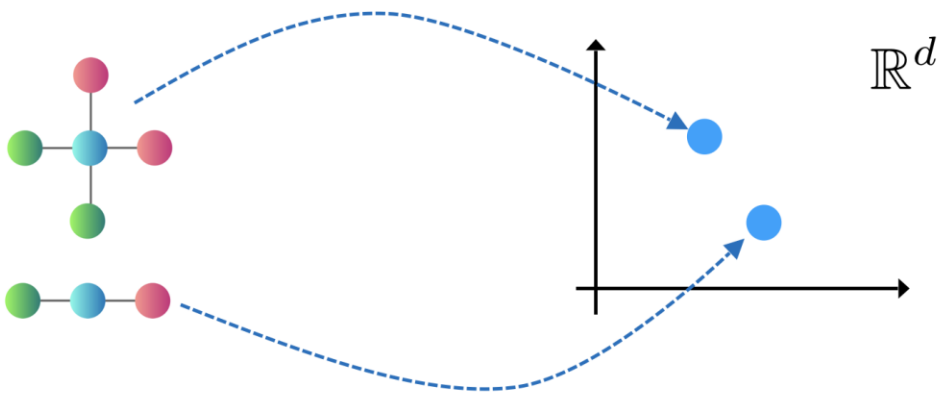
Socia networks and social media graphs (Hamilton et al. 2017)



Human IQ test (Barrett et al. 2018)

Graph Neural Networks (Example Problem)

Input: Graph with node features

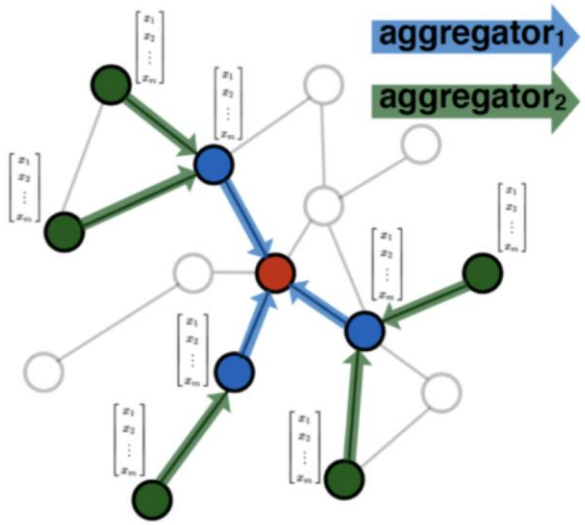


Task: Node/Graph classification

Goal: Learn node/graph embeddings that capture graph structure

Graph Neural Networks Formulation

GNNs learn node representations with **Neighborhood Aggregation**



Aggregate from neighbors

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

Combine with current node

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

…

For Graph classification use **readout to pool node embeddings**

$$h_G = \text{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right)$$

K GNN iterations captures K hop network structure

More resouces: https://www.youtube.com/watch?v=v3jZRkvIOIM, https://tkipf.github.io/graph-convolutional-networks/

More Resources

- Programmer/Developer oriented courses
  - fast.ai Practical Deep Learning for Coders** (https://course.fast.ai/)
  - Full Stack Deep Learning Bootcamp (https://fullstackdeeplearning.com/march2019)
- Courses with mathematical treatment
  - NPTEL's Deep Learning course (detailed derivations of each concept) (https://bit.ly/2VuGK06)
  - Andrew Ng's Deep Learning Specialization (https://bit.ly/2tT9aSC)
  - Hugo Larochelle's Neural Networks Course (https://bit.ly/1j2oViP)
- Courses with specialized applications
  - Stanford DL for Computer Vision course (http://cs231n.stanford.edu/)
  - Stanford DL for NLP course (http://web.stanford.edu/class/cs224n/)
- Advanced State of the art courses
  - Deep RL (https://bit.ly/2KwA5gi)
  - Deep Unsupervised Learning (https://bit.ly/2TODPfW)

**highly recommended

More Resources

- Demos
  - Backprop Demo (https://bit.ly/2OQ9zA7)
  - Neural Network Demo (https://bit.ly/2mnpFIp)
  - ConvNet Demo (https://stanford.io/1BNjsWA)
- Book
  - Deep Learning Book, Goodfellow et al. (https://www.deeplearningbook.org/)
- Optimizers in Deep Learning
  - Ruder's blogpost (http://ruder.io/optimizing-gradient-descent/)