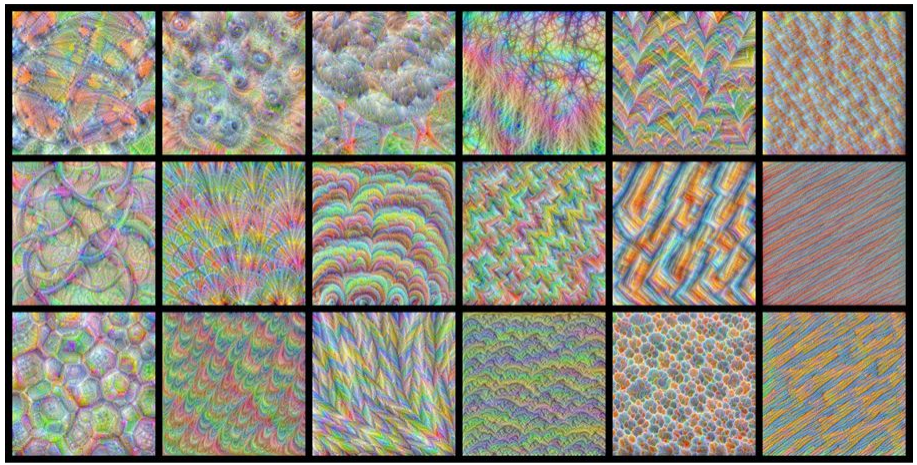
ICME Summer Workshops 2021

Introduction to Deep Learning

Session 3: 8:00—9:30 AM
Instructor: Sherrie Wang

icme-workshops.github.io/deep-learning



https://towardsdatascience.com/how-to-visualize-convolutional-features-in-40-lines-of-code-70b7d87b0030

Workshop Schedule

Day 1

Session 1 (Yesterday 8:00—9:30 AM)

- Introduction
- Examples of deep learning
- Math review
- Neural network basics

Session 2 (Yesterday 9:45—11:00 AM)

- Loss functions
- Gradient descent
- Backpropagation
- Walkthrough of an example

Day 2

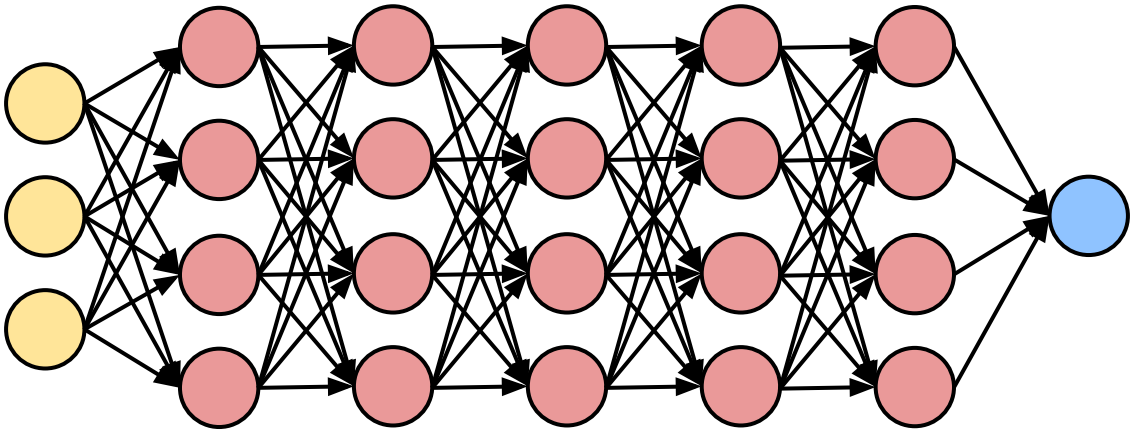
Session 3 (Today 8:00—9:30 AM)

- More on implementation: preprocessing, regularization, monitoring training
- Convolutional neural networks

Session 4 (Today 9:45—11:00 AM)

- Recurrent neural networks
- Deep learning libraries
- Failures of deep learning
- Walkthrough of an example

Quick recap



$$g\left(\begin{bmatrix} 2 & 4 & 2 \\ 1 & 3 & 7 \\ 0 & 7 & 3 \end{bmatrix}\begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} + \begin{bmatrix} -3 \\ 0 \\ 2 \end{bmatrix}\right)$$

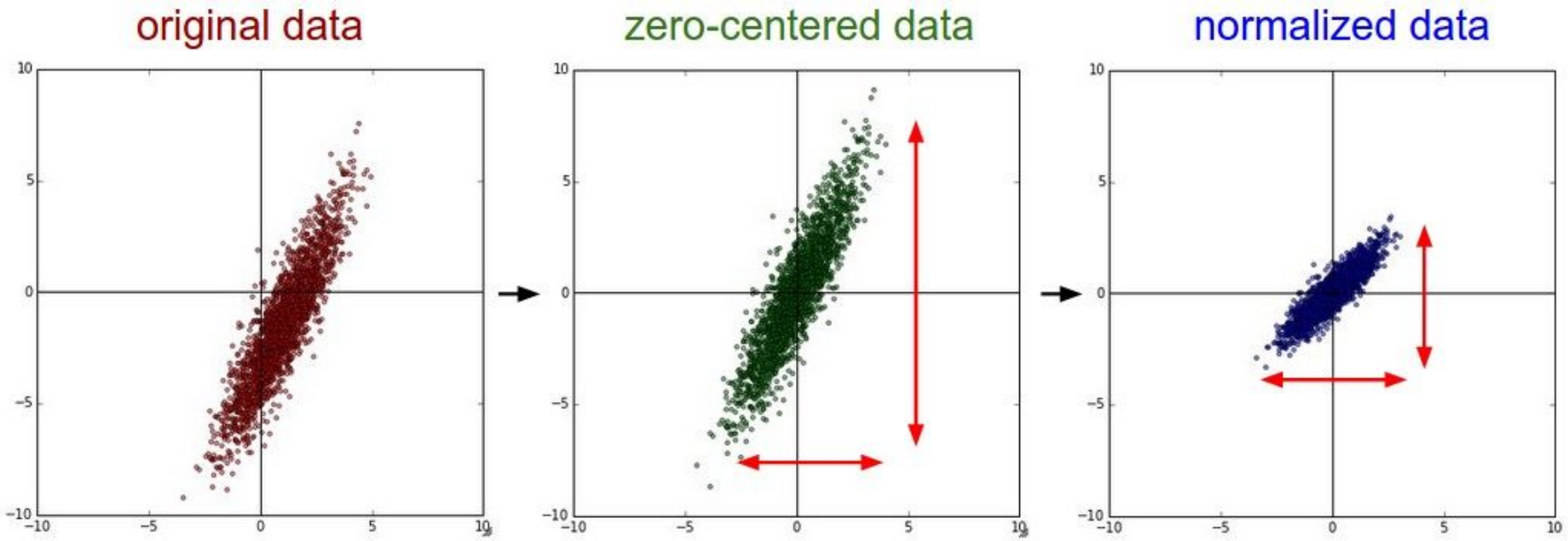
Trainable parameters learned through gradient descent and backprop on a loss function

Data Preprocessing

Three kinds of preprocessing

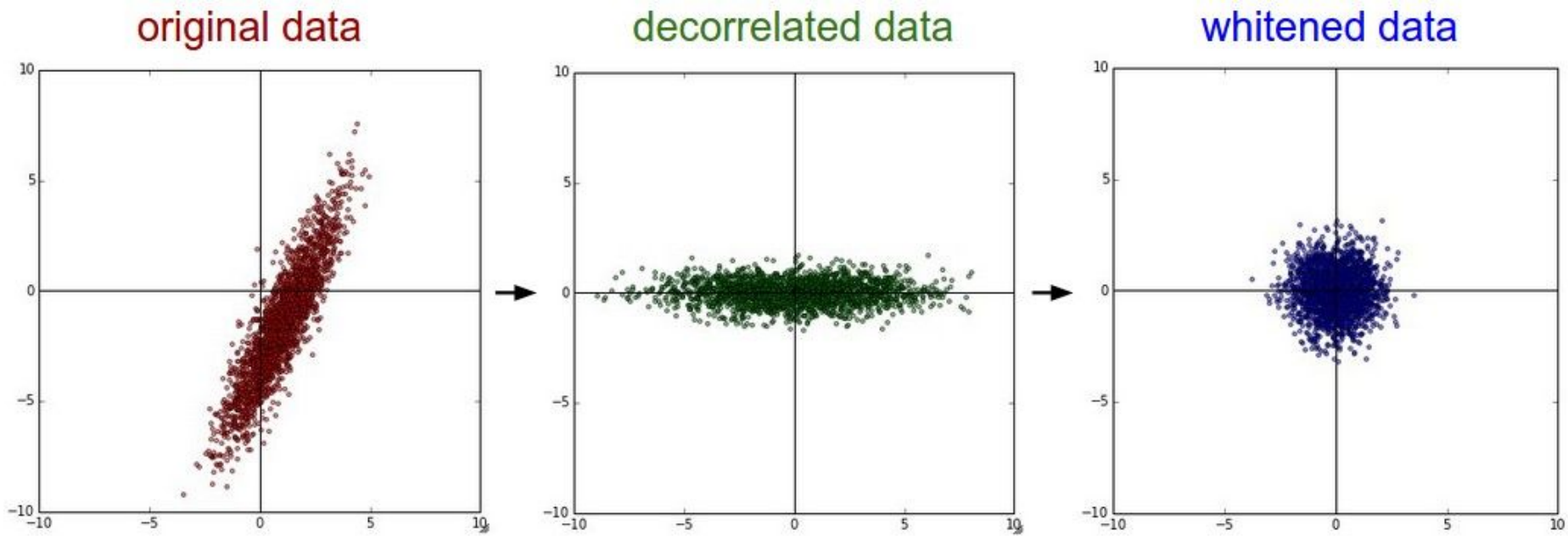
- **Mean subtraction**
  - Subtract the mean across samples for every feature in the data
- **Normalization**
  - Scale each feature to have standard deviation of 1
- **Whitening**
  - First center the data, then compute the covariance matrix, decorrelate and normalize the data

Mean subtraction and normalization



https://cs231n.github.io/neural-networks-2/

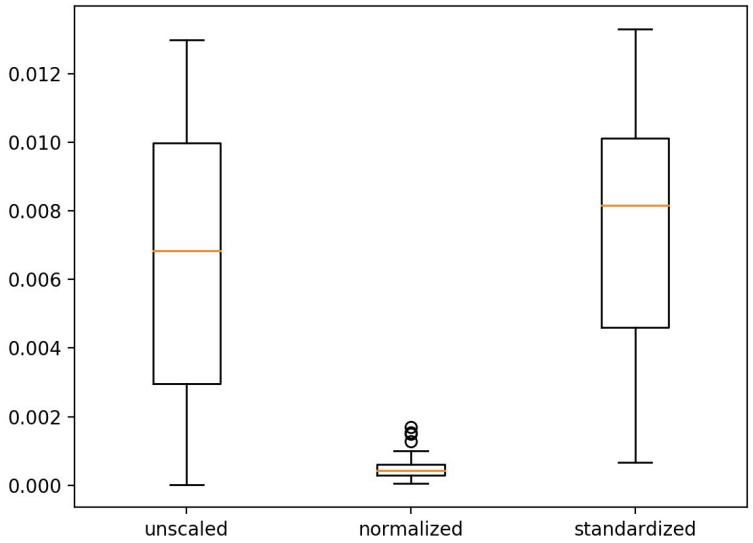
Whitening



https://cs231n.github.io/neural-networks-2/

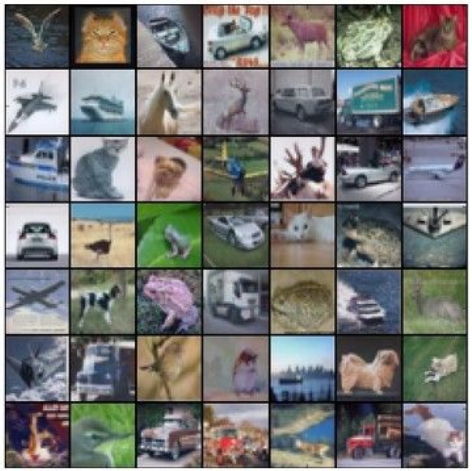
Which preprocessing to choose, if any?

Typically zero-center and normalize the data, but not whiten.

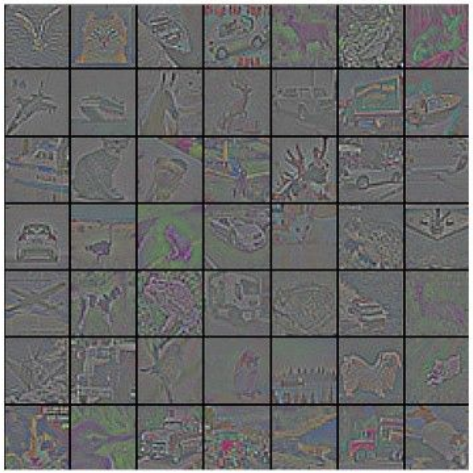


Which preprocessing to choose, if any?

original images



whitened images



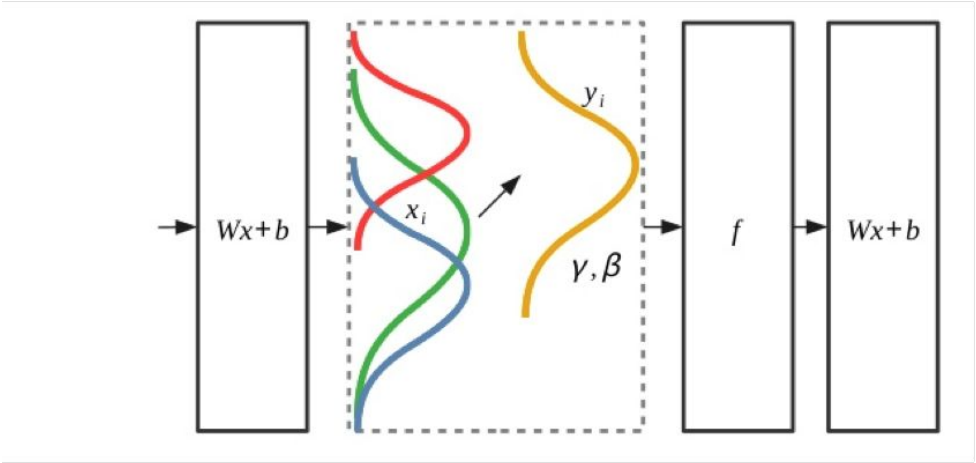
Batch normalization layer

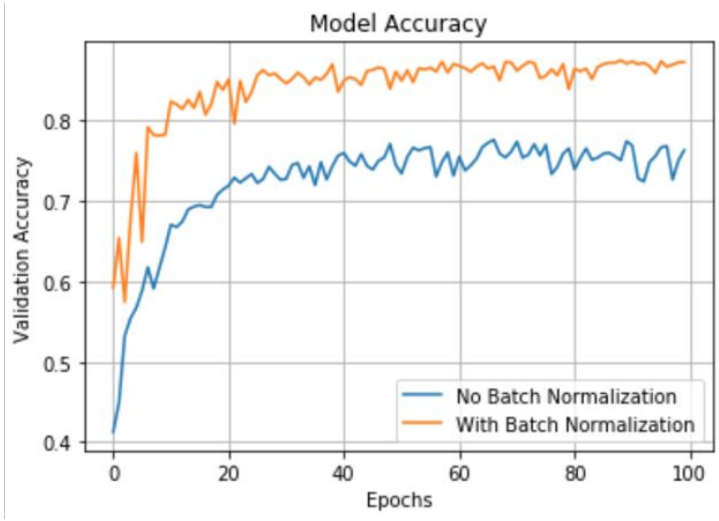
Goal is to control the mean and variance of each layer's output.

1. Normalize output of each neuron to be mean zero and unit variance across the batch at every training step.
2. Rescale the output of the neuron to be some specific mean and variance, both of which are trainable parameters.

Batch normalization layer

Empirically observed to stabilize and speed up training, and can result in overall better model performance.





Weight Initialization

What not to do: Initialize all weights to zero

If all weights are zero, then all neurons output the same thing, the gradient is the same for all neurons, and all neurons in a layer will be updated in identical manners.

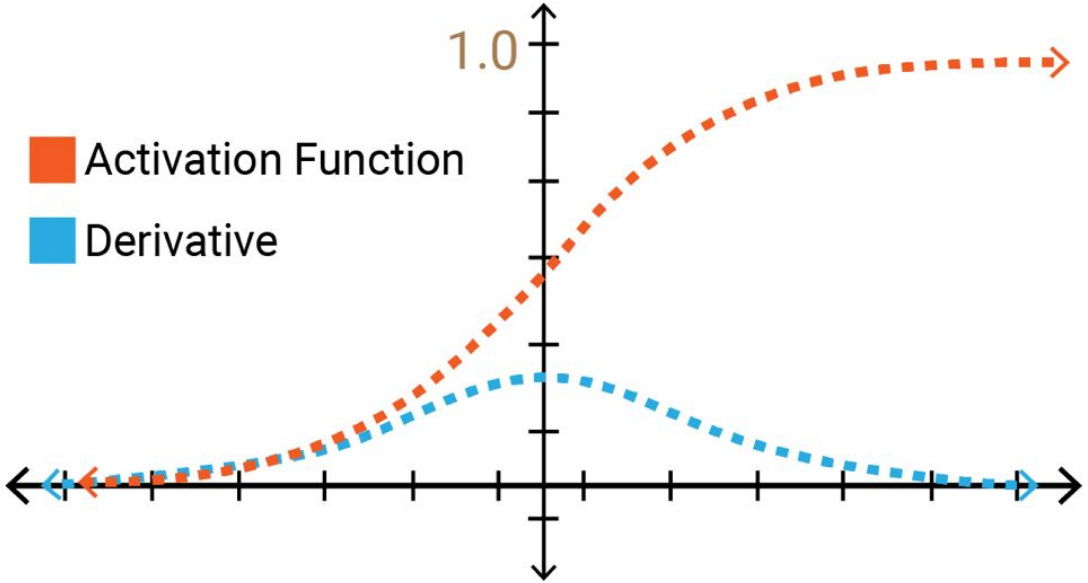
Instead, initialize weights randomly to break symmetry.

But how big should weight initializations be?

Vanishing and exploding gradient problem

Vanishing gradient = gradients become too small. Training is too slow.

Exploding gradient = gradients become too big. Weights can go to NaN.



Vanishing gradients

$$\hat{y} = a^{[L]} = W^{[L]} W^{[L-1]} W^{[L-2]} \dots W^{[3]} W^{[2]} W^{[1]} x$$

$$W^{[1]} = W^{[2]} = \dots = W^{[L-1]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$\hat{y} = W^{[L]} 0.5^{L-1} x$$

Exploding gradients

$$\hat{y} = a^{[L]} = W^{[L]} W^{[L-1]} W^{[L-2]} \dots W^{[3]} W^{[2]} W^{[1]} x$$

$$W^{[1]} = W^{[2]} = \dots = W^{[L-1]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

$$\hat{y} = W^{[L]} 1.5^{L-1} x$$

Criteria for weight initialization

We want:

1. The mean of activations to be zero
2. The variance of activations to be the same across every layer

So that the gradient can be backpropped from the loss to the input layer without vanishing or exploding gradients.

Common initializations: He and Xavier

For sigmoid and hyperbolic tangent activations, use Xavier initialization:

- Random initialization
- Mean zero
- Variance $Var(W^{[l]}) = \frac{1}{n^{[l]}}$

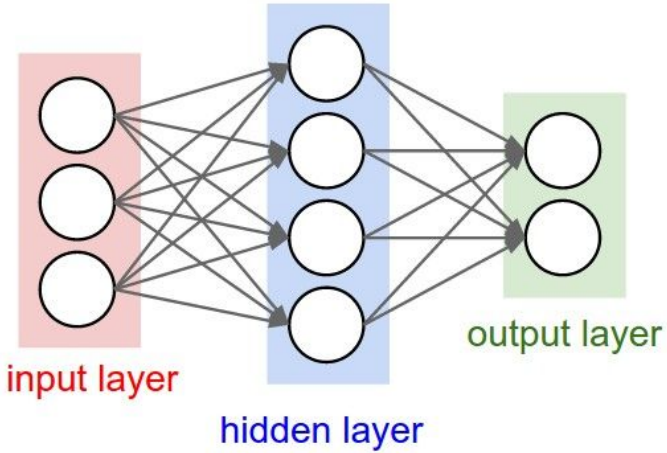
For ReLU activations, use He initialization:

- Random initialization
- Mean zero
- Variance 2x in Xavier

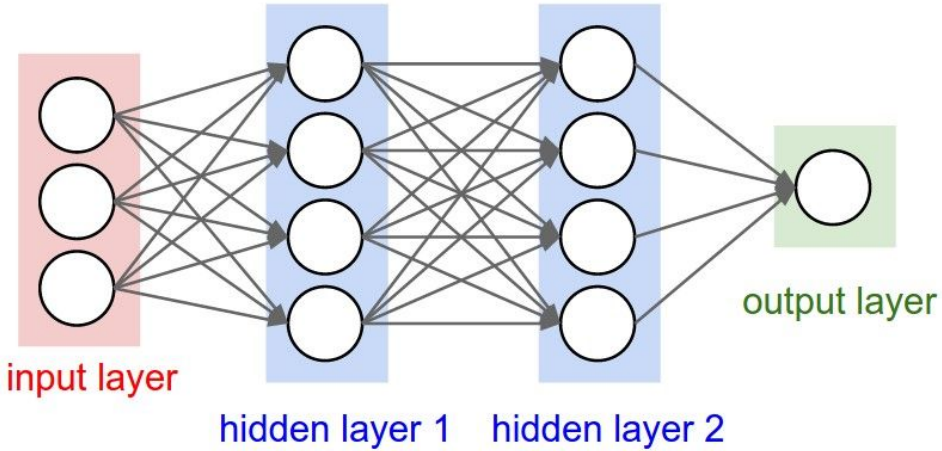
Network depth and width

Depth = number of layers

A 2-layer neural network:

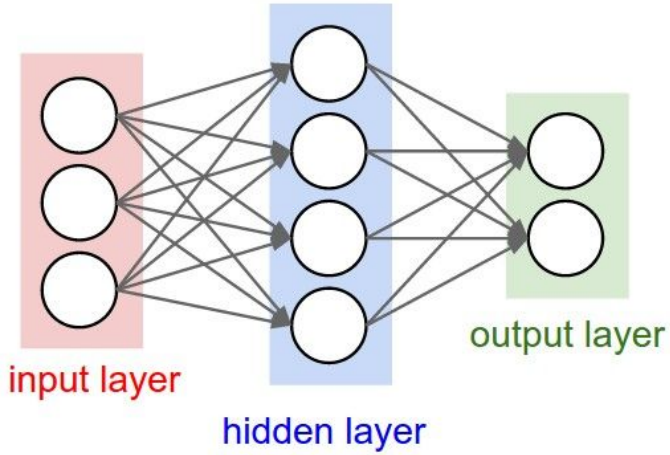


A 3-layer neural network:



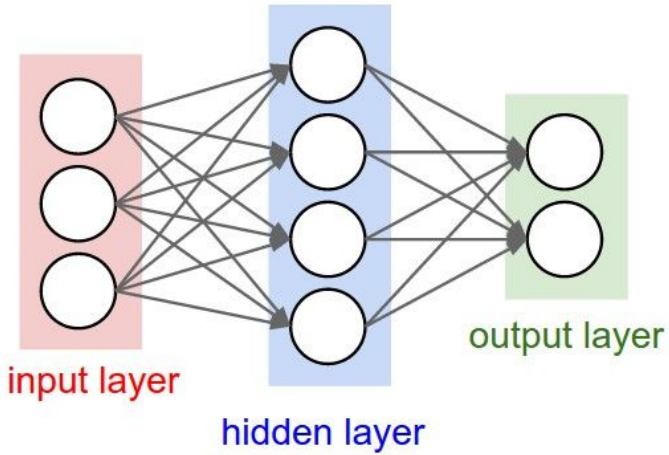
Width = number of neurons in a layer

The hidden layer below has width 4.



Measuring the size of a neural network

Neural network size is measured by the number of neurons or, more commonly, number of parameters.



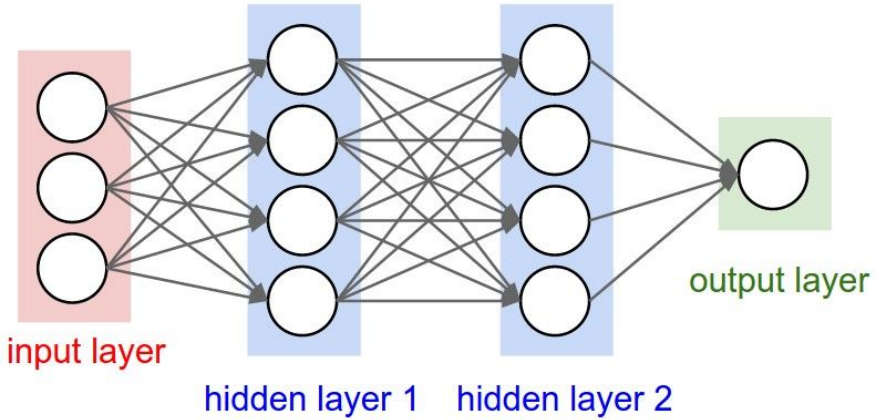
This network has:

- 4 + 2 = 6 neurons
- (3 x 4) + (4 x 2) = 20 weights
- 4 + 2 = 6 biases
- For a total of 26 learnable parameters

Poll: Model size

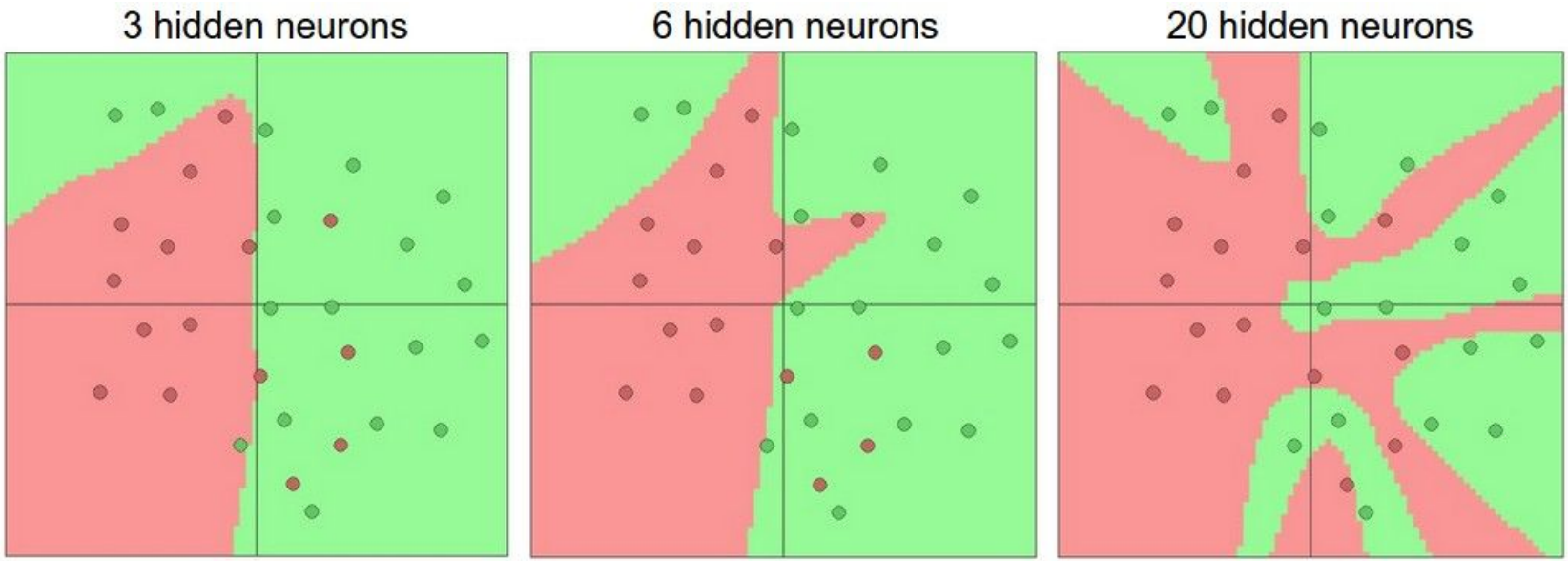
Go to **tinyurl.com/dlworkshop2021**

How many learnable parameters are there in the following neural network?



A. 20
B. 32
C. 41
D. 44

Expanding model capacity

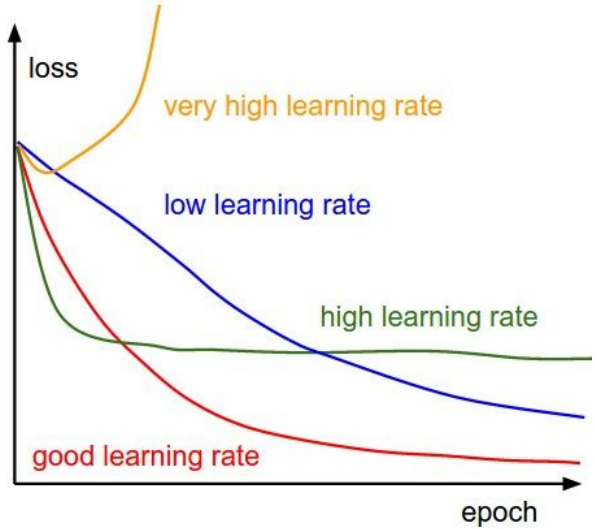


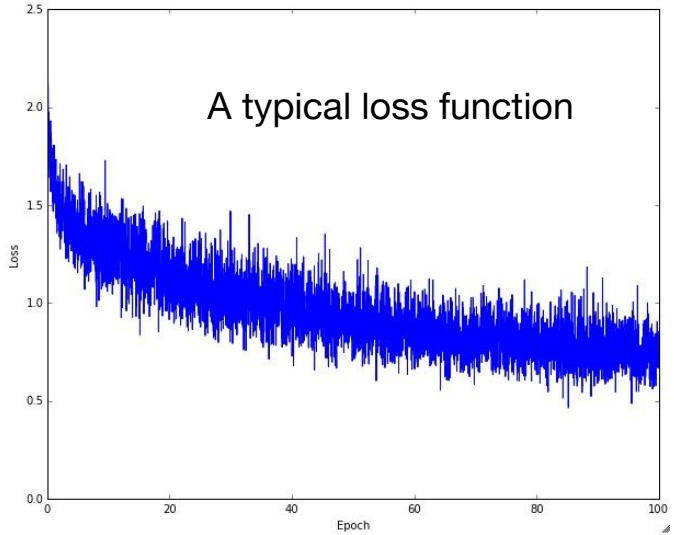
https://cs231n.github.io/neural-networks-1/

Monitoring the training process

Monitoring the loss

The shape of the loss function over time (as training progresses) tells you about how well the model is learning and gives clues about the hyperparameters.





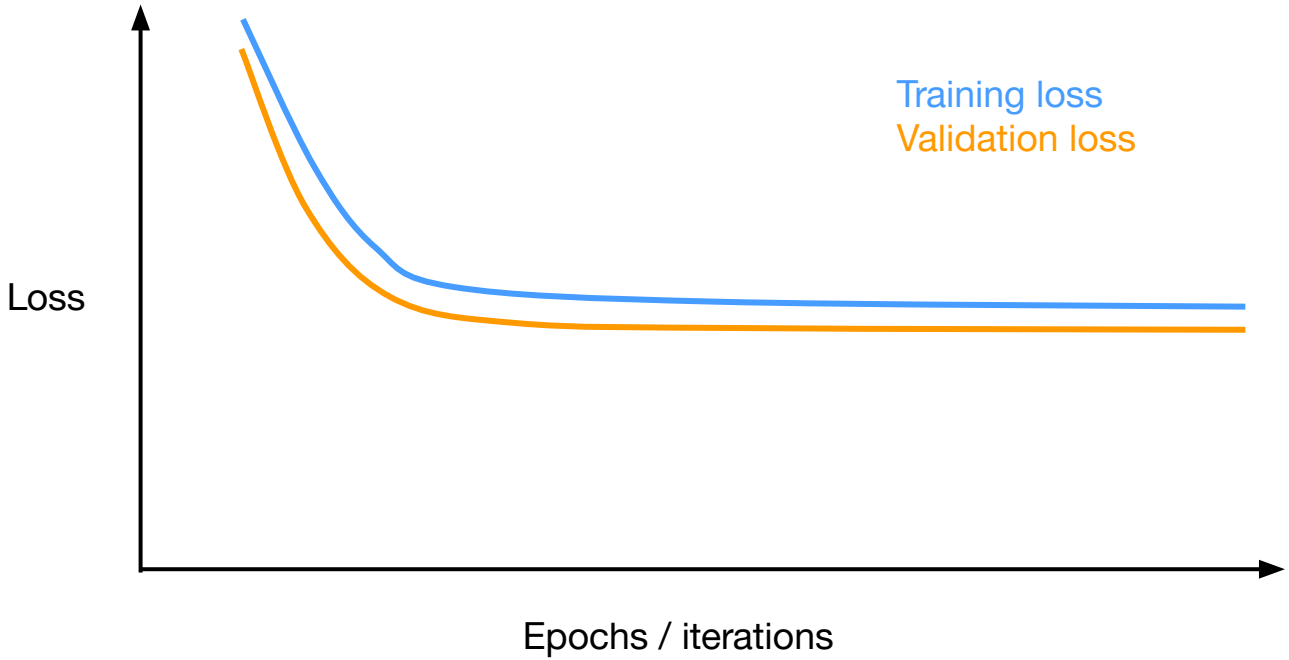
https://cs231n.github.io/neural-networks-3/

Overfitting and underfitting

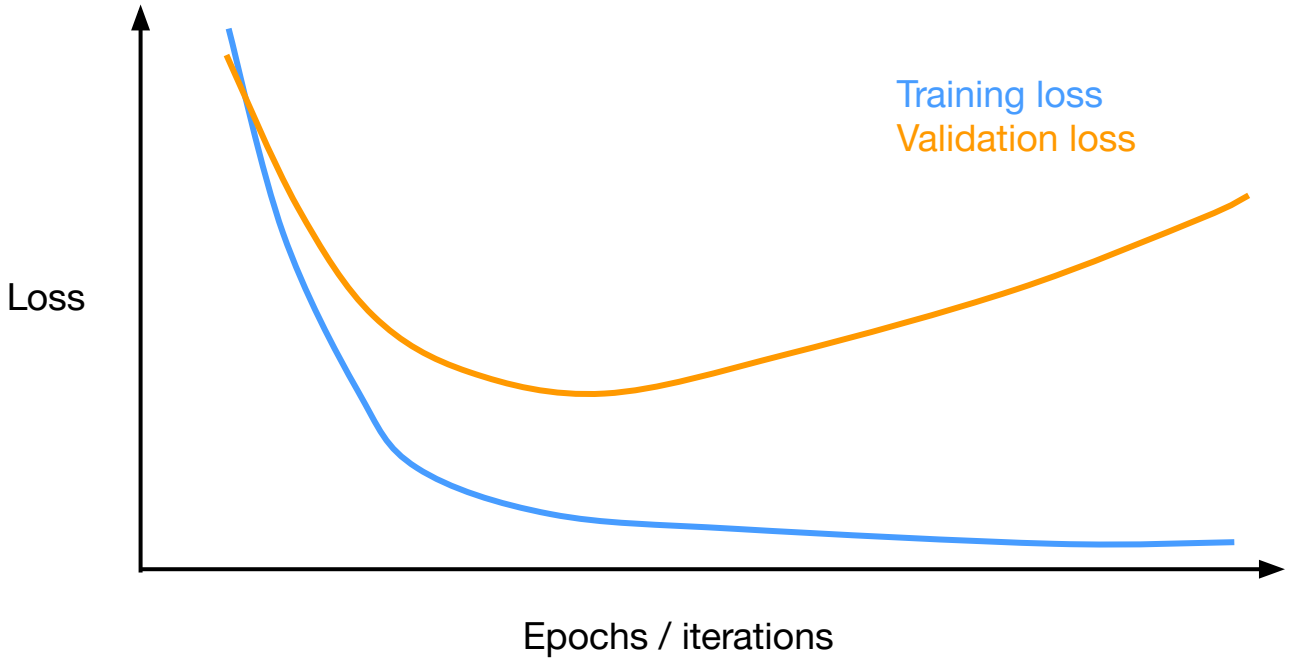
- **Underfitting:** when your model is not complex enough to capture the structure in the data. The capacity of the model is insufficient. Both training loss and validation loss are high.

- **Overfitting:** when your model has so much capacity that it memorizes your training data. It usually indicates insufficient training data relative to your model complexity. Training loss is low, but validation loss is high.

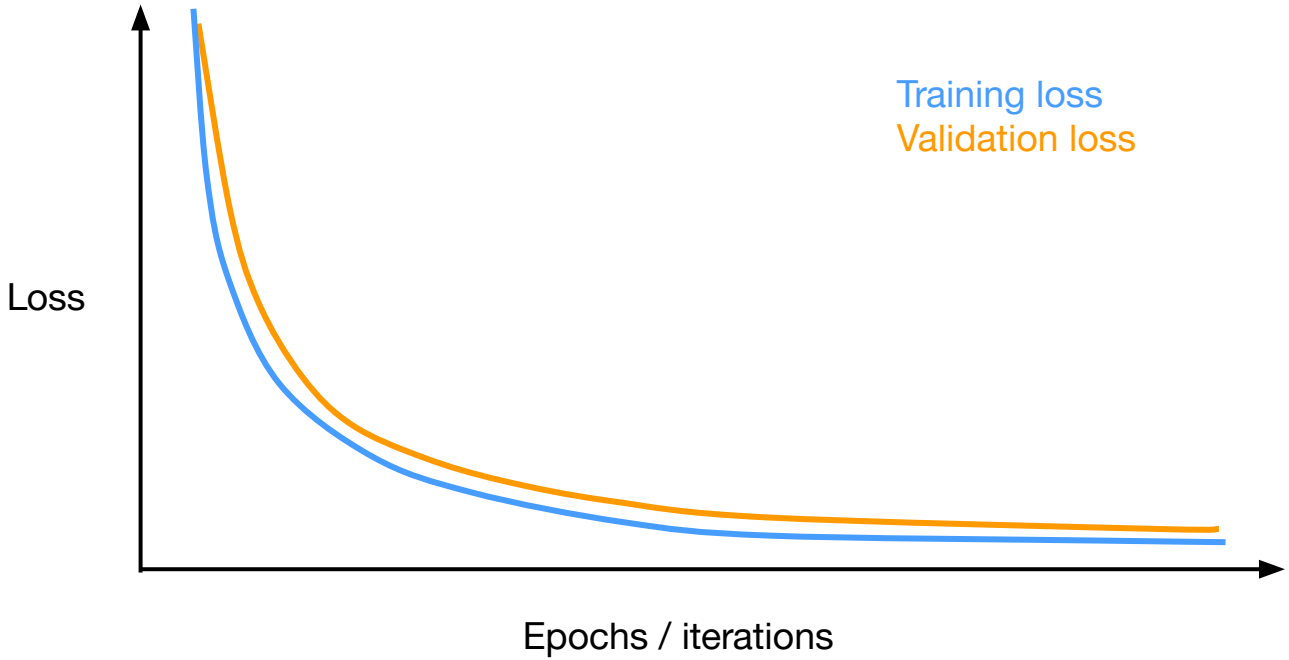
Underfitting



Overfitting

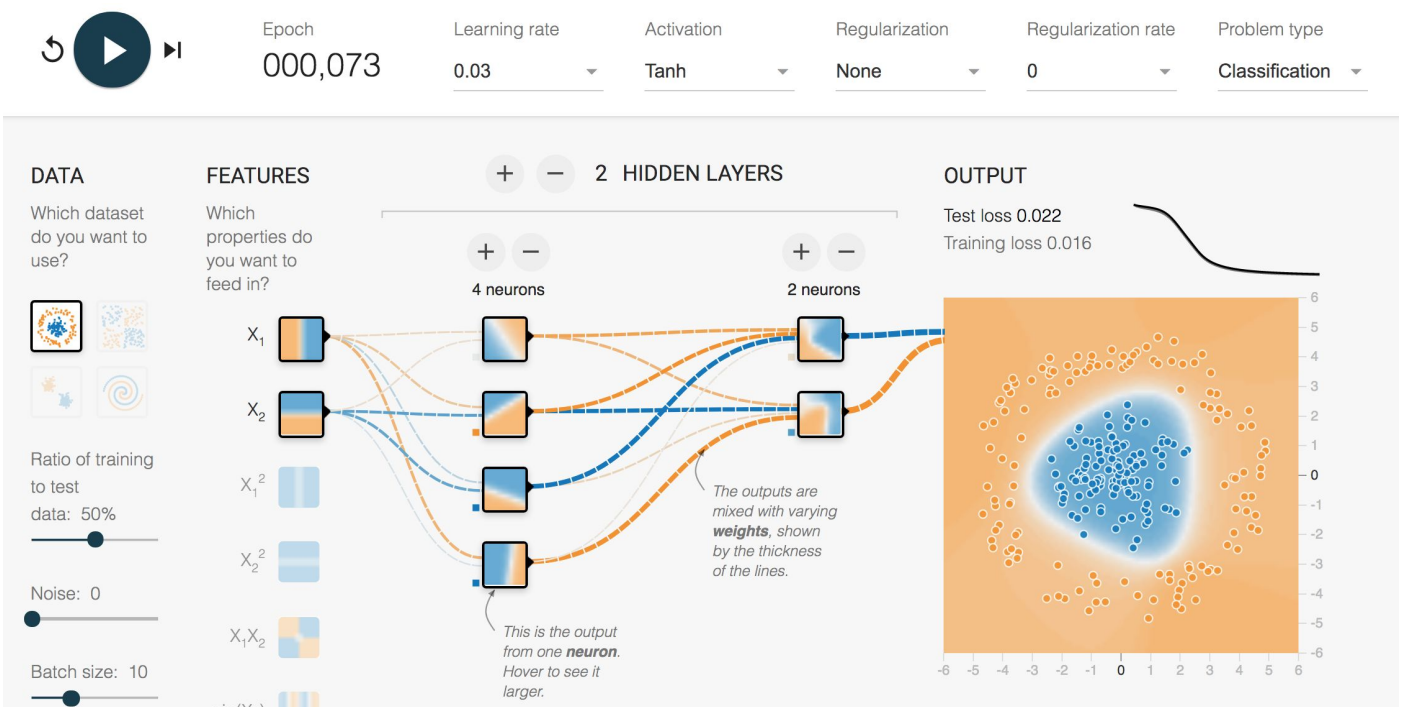


Good training



Exercise: Underfit a dataset in TF Playground

https://playground.tensorflow.org



Regularization

One way to combat overfitting is to add a regularization term to the cost function.

This constrains the capacity of the model so that it has a lower chance of overfitting.

$$J(W, b) + \boldsymbol{\lambda} R(W, b)$$

L2 regularization

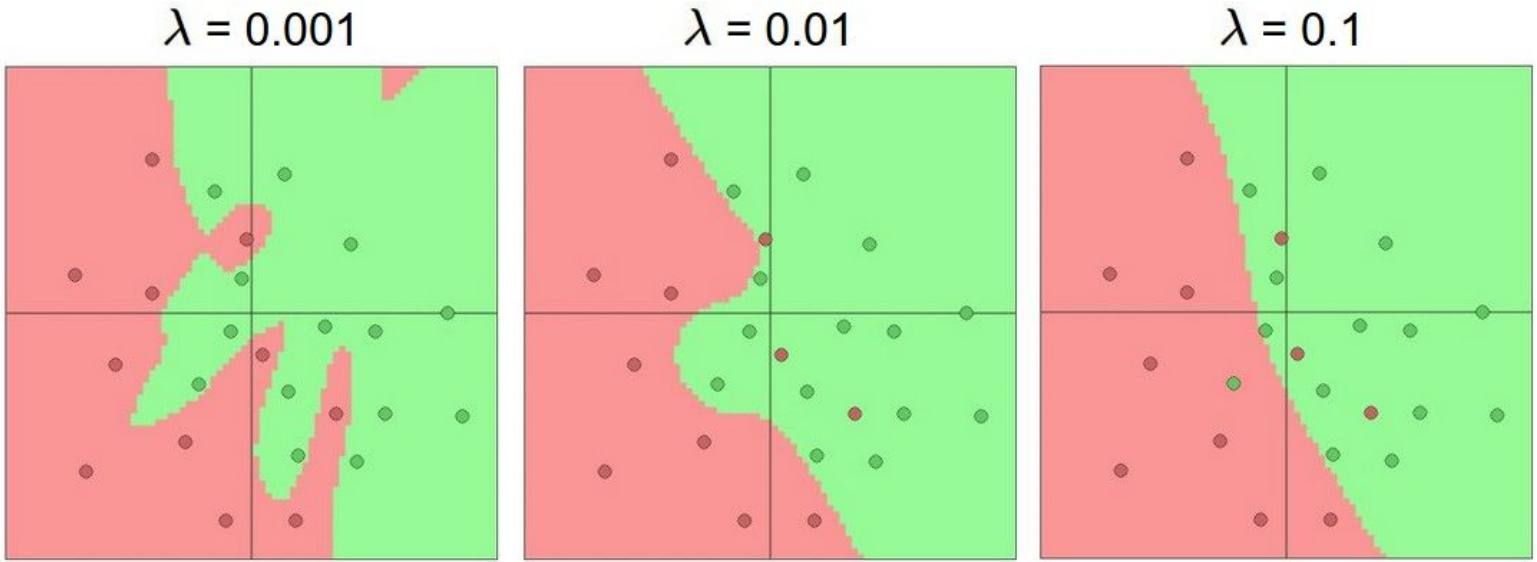
Penalize the squared sum of weights.

Squaring penalizes large weights, which encourages more diffuse weight vectors.

$$J(W, b) + \tfrac{1}{2}\lambda R(W, b)$$

$$R(W, b) = \| W \|^2$$

L2 regularization



https://cs231n.github.io/neural-networks-1/

L1 regularization

Penalize the sum of the absolute value of weights.

Leads to weight vectors becoming sparse (i.e. some values are zero or nearly zero). The network tends not to depend on parts of the input or intermediate values that are deemed less useful.

$$J(W, b) + \lambda \boxed{R(W, b)}$$

$$\| W \|_1$$

Max norm constraints

Enforce an upper bound on the magnitude of the weight vector for every neuron.

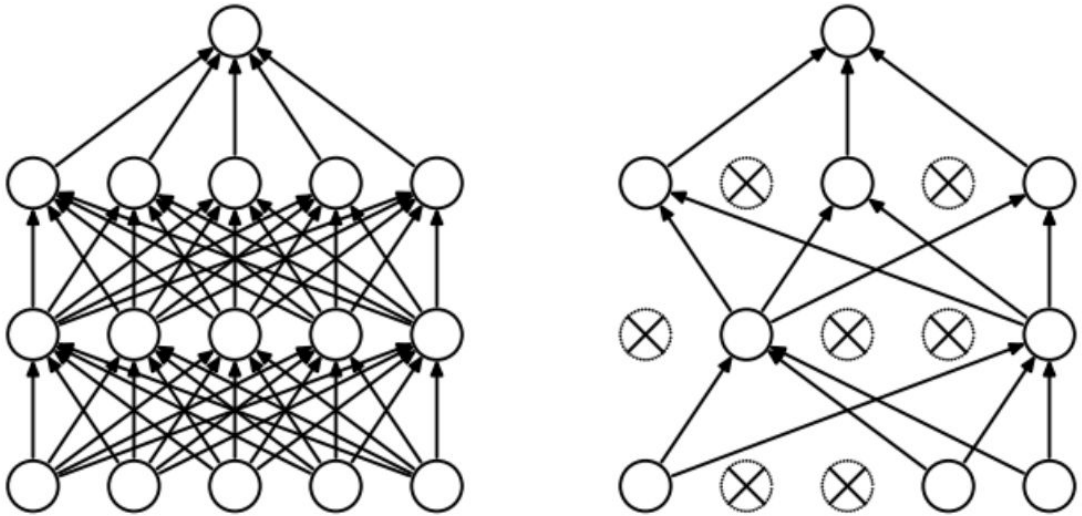
Perform parameter updates as usual, then enforce the constraint by clamping the weight vector of every neuron to a magnitude $< c$.

Typical values of c are around 3 or 4, since weights are usually small.

Dropout layers

Another way to regularize neural networks.

Randomly drop some fraction of neurons at a layer during training.



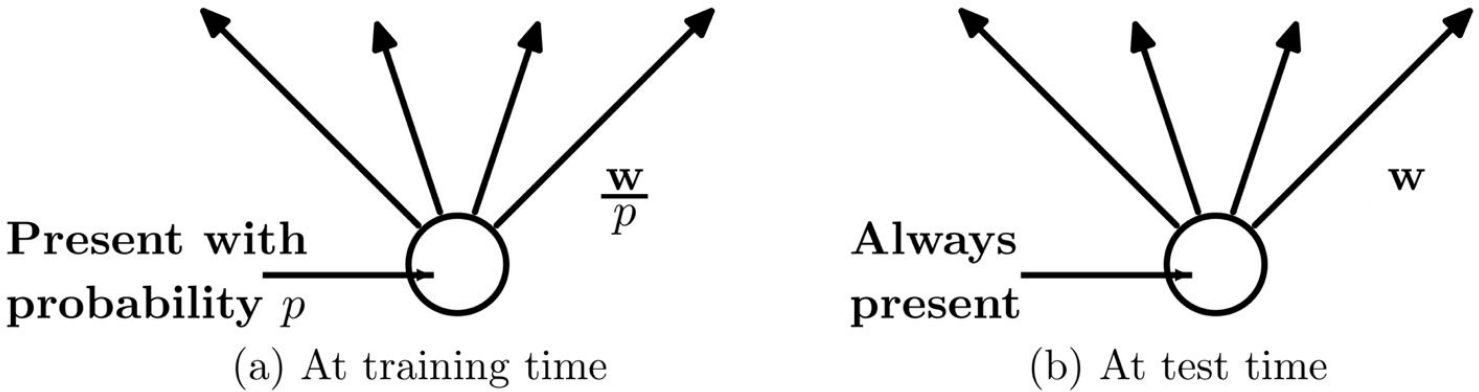
(a) Standard Neural Net

(b) After applying dropout.

Dropout layers

The output of non-dropped neurons is scaled by the inverse probability of keeping the neuron alive.

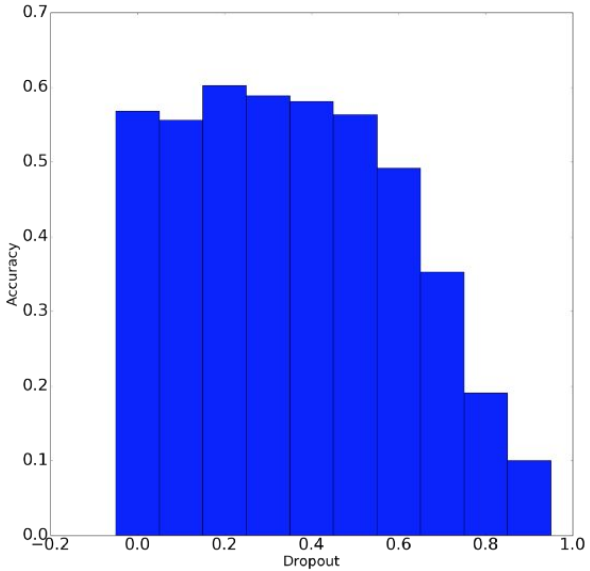
At test time, you do not use dropout.

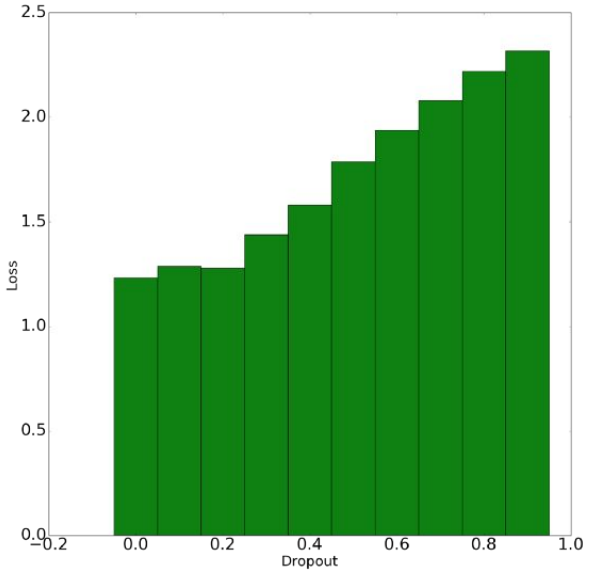


Dropout layers

The point is to reduce dependence across neurons.

Empirical results:





Tips for training

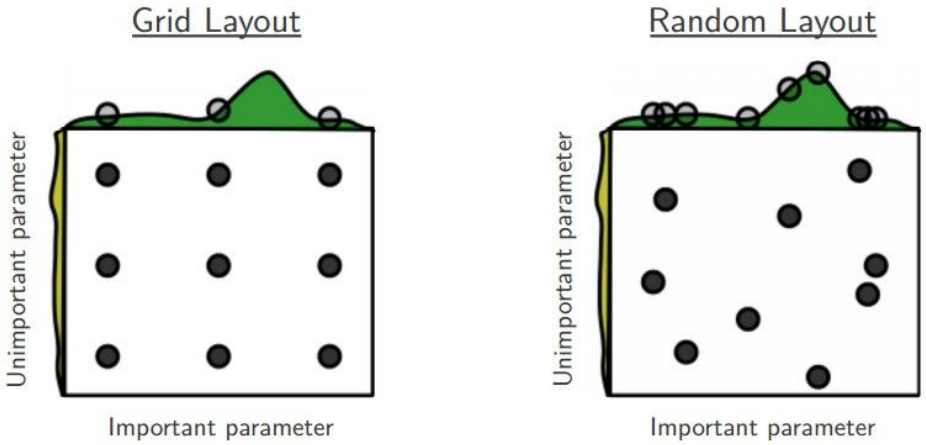
1. Try other machine learning methods first
2. If you have trouble with training, first try to overfit a small subset of the dataset
3. If loss does not decrease smoothly, try to increase the number of hidden neurons in each layer
4. Play around with the learning rate; try to increase it as well as decrease it over time
5. Check online for others who have tried to solve your problem before attempting your own solution

Recap: Things that could be tuned

- Learning rate
- Batch size
- Optimization algorithm (SGD, Adam, Adagrad…)
- Data normalization
- Parameter initialization
- Regularization type and strength
- Number of hidden layers
- Number of neurons per layer
- Type of activation function
- Epochs of training to obtain final model

Searching over hyperparameters

- Less computationally expensive: Train one model, observe its performance, try new hyperparameters based on observations during training
- More comprehensive: Grid search or random search (latter is recommended)
- Usually too computationally expensive: Exhaustive search



Why do we need a test set?

Since we use the validation set to choose architecture and hyperparameters, it is possible to overfit to the validation set.

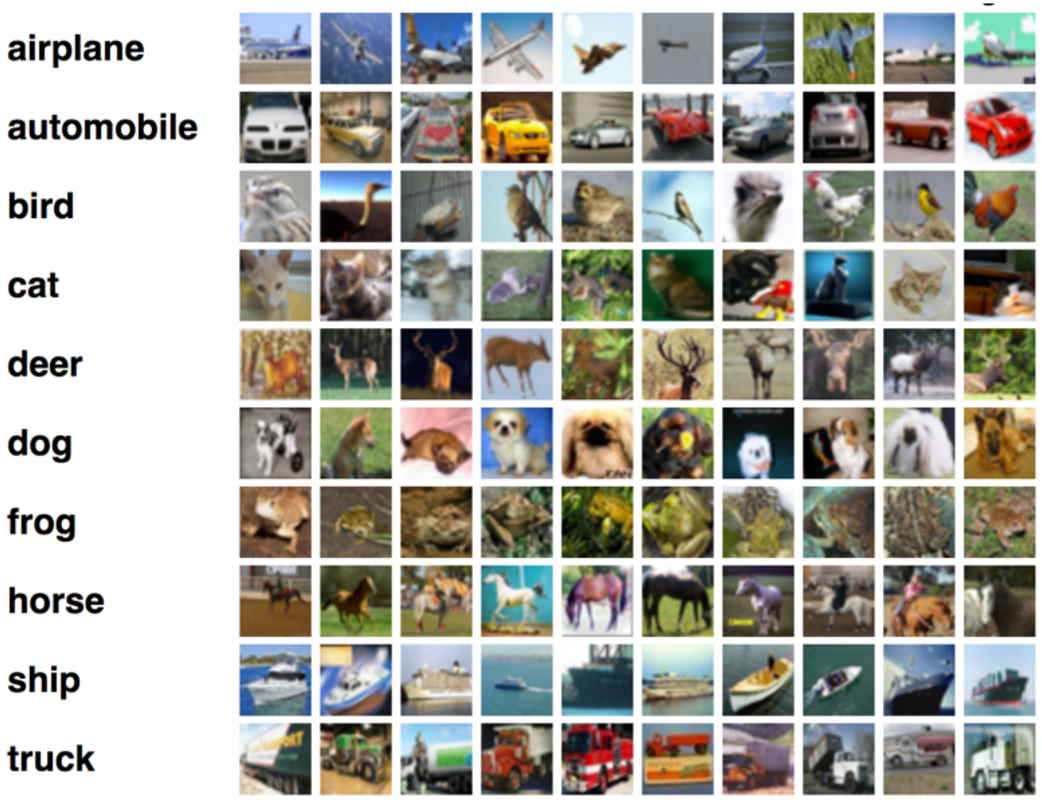
The test data set is used to give us an unbiased estimate of out-of-sample performance.

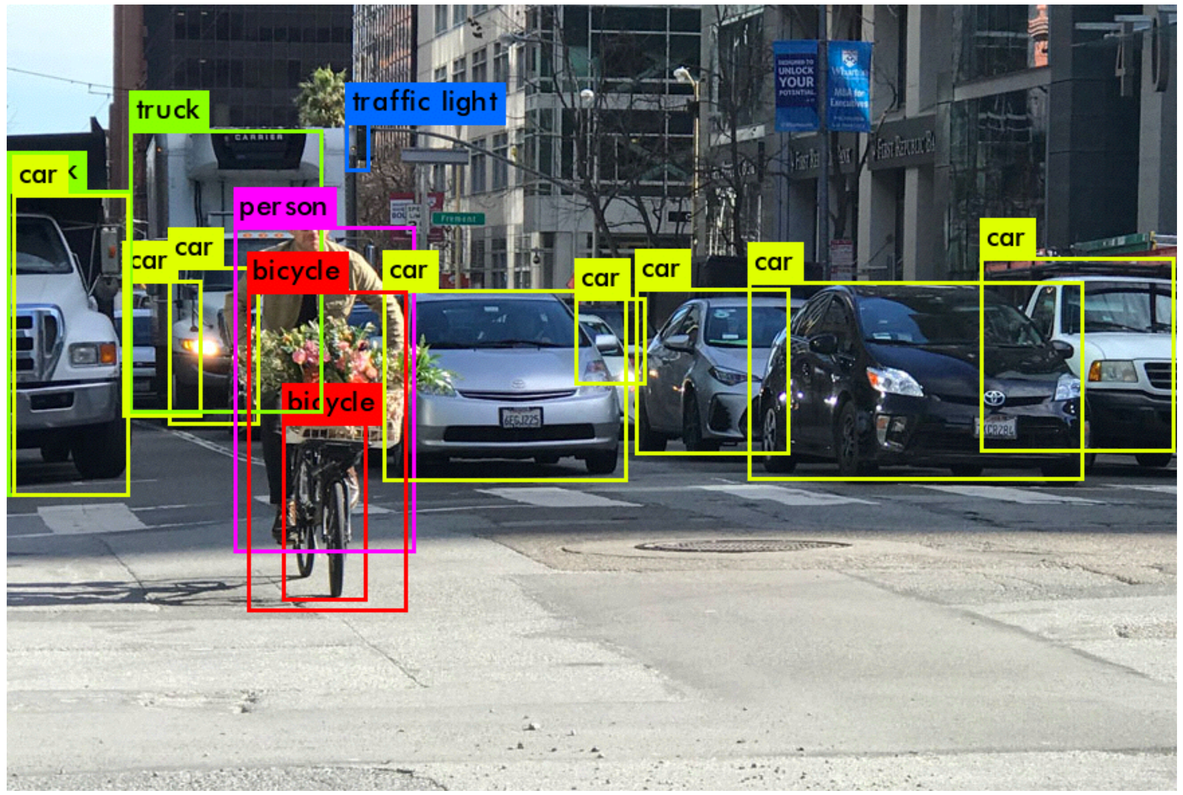
This underscores the importance of using the test set *only once*.

Convolutional Neural Networks

Convolutional neural networks

- Neural networks have been very successful at image classification and object detection



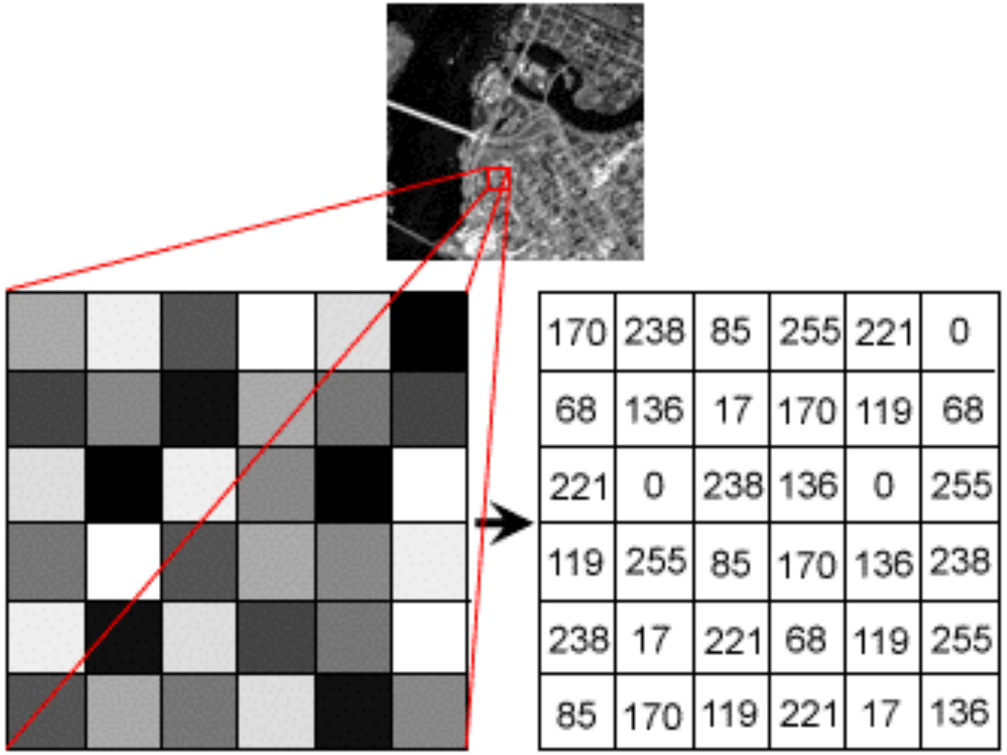


Convolutional layers

- Convolutional layers are a special type of layer for image inputs
- Convolutions replace matrix multiplication with the convolution operation

$$g(\mathbf{W}\mathbf{x} + \mathbf{b}) \longrightarrow g(\mathbf{f} * \mathbf{x} + \mathbf{b})$$

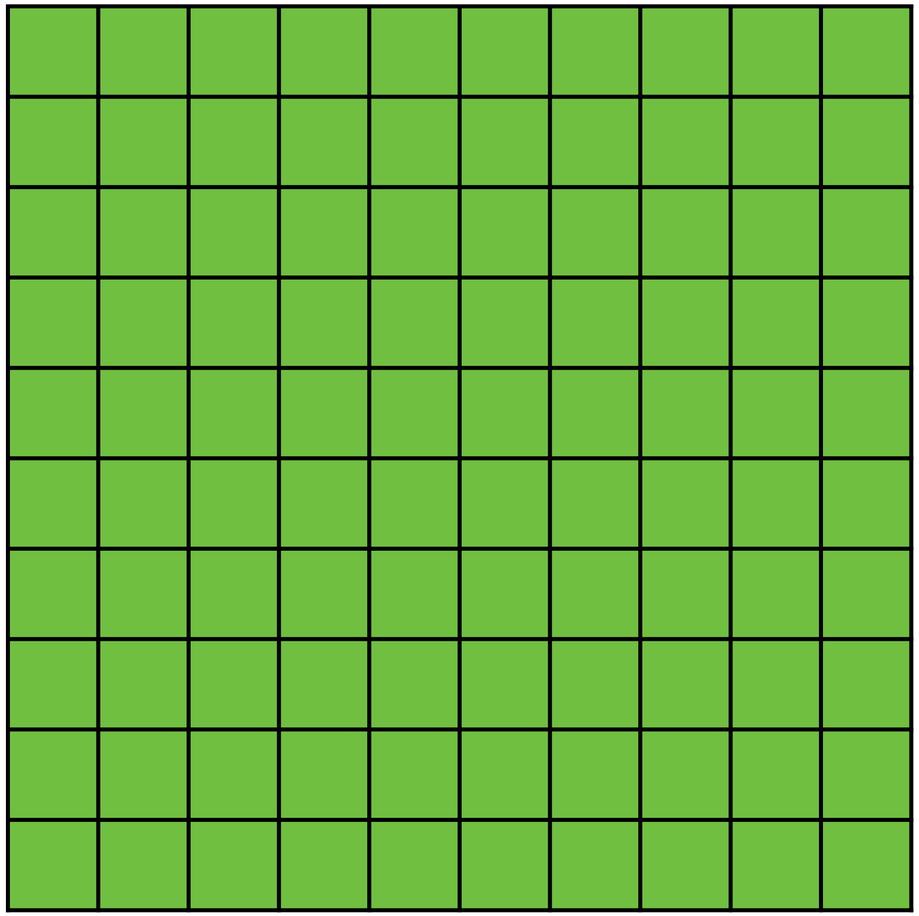
Quick reminder: images as tensors



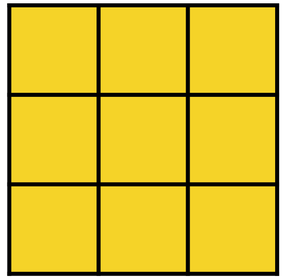
170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Convolutions

Image

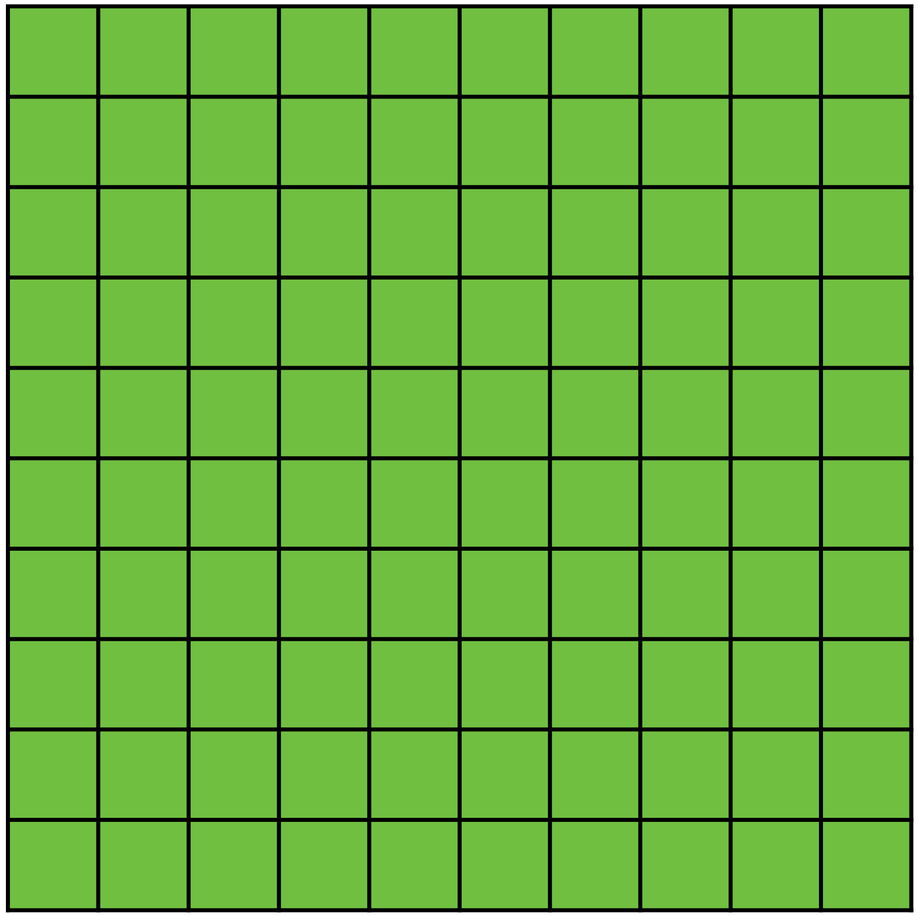


Filter



Convolutions

Image



Filter

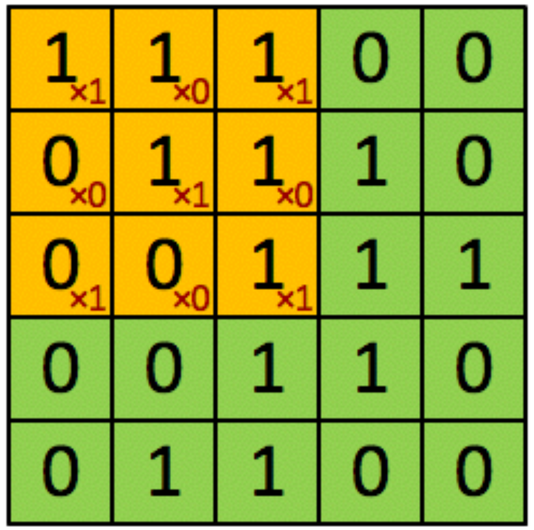
1	4	0
0	2	1
3	0	1

Learnable weights

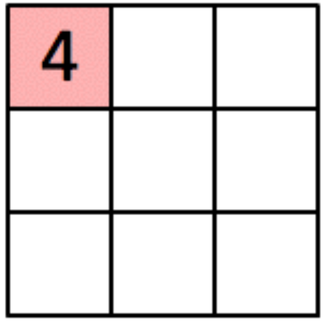
Convolutions

An example on an image with 1 channel

Notice that the dimensions of the output are smaller than the dimensions of the image by 2 pixels



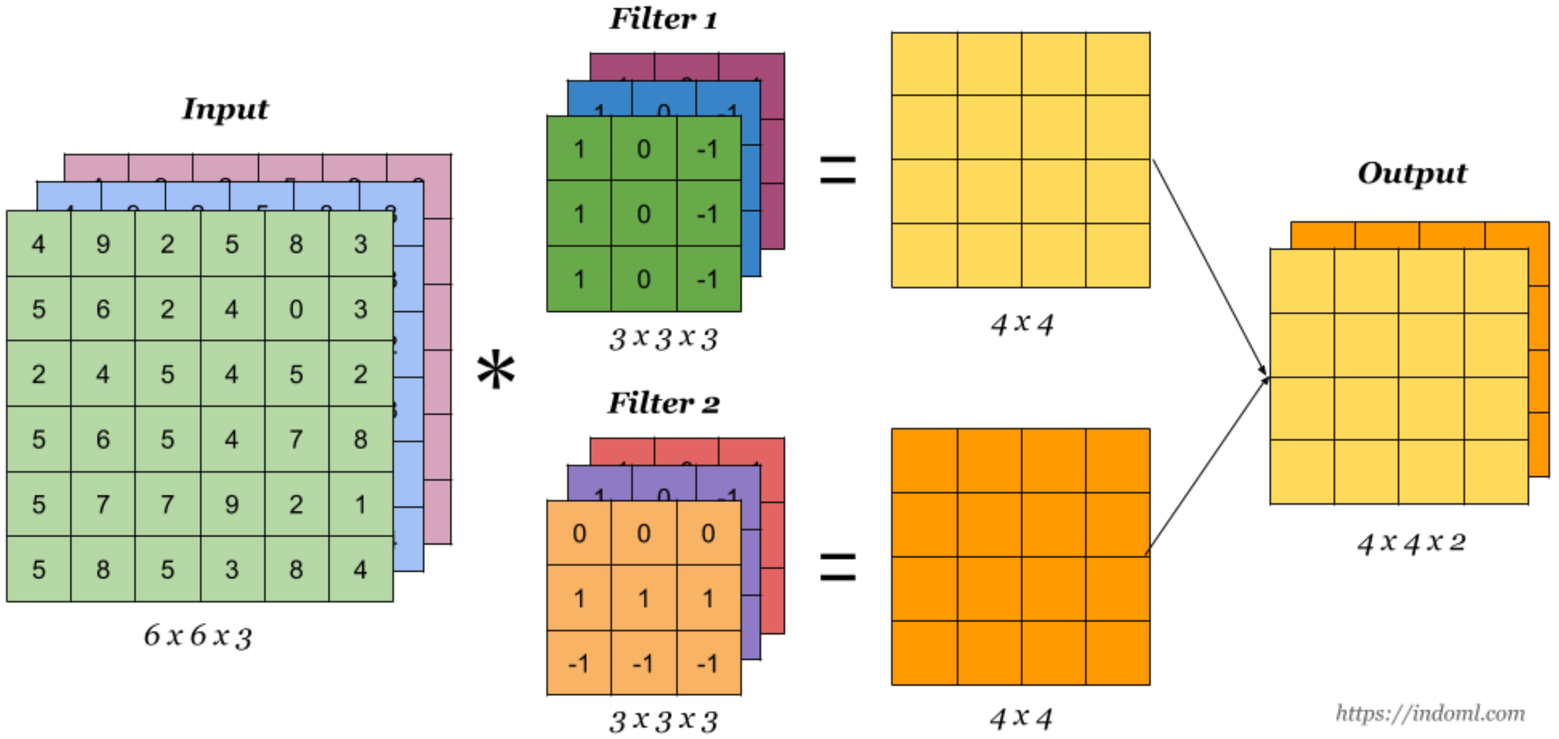
Image



Convolved Feature

Convolutions: 3 channels

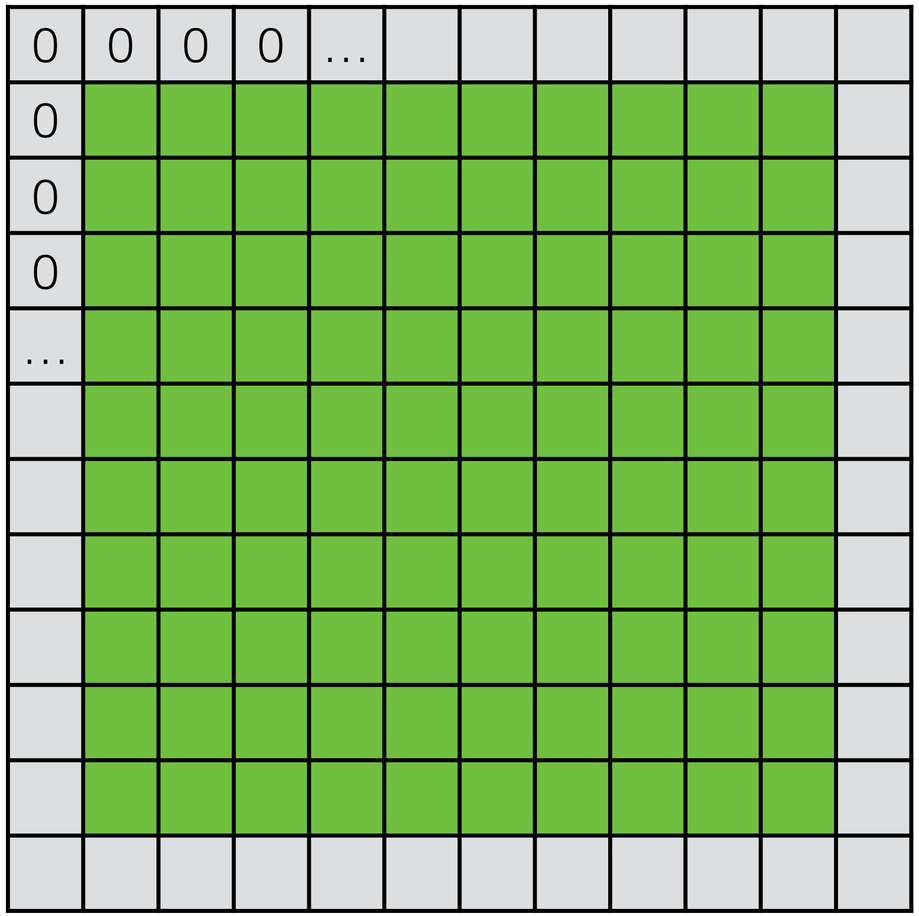
Most images are RGB



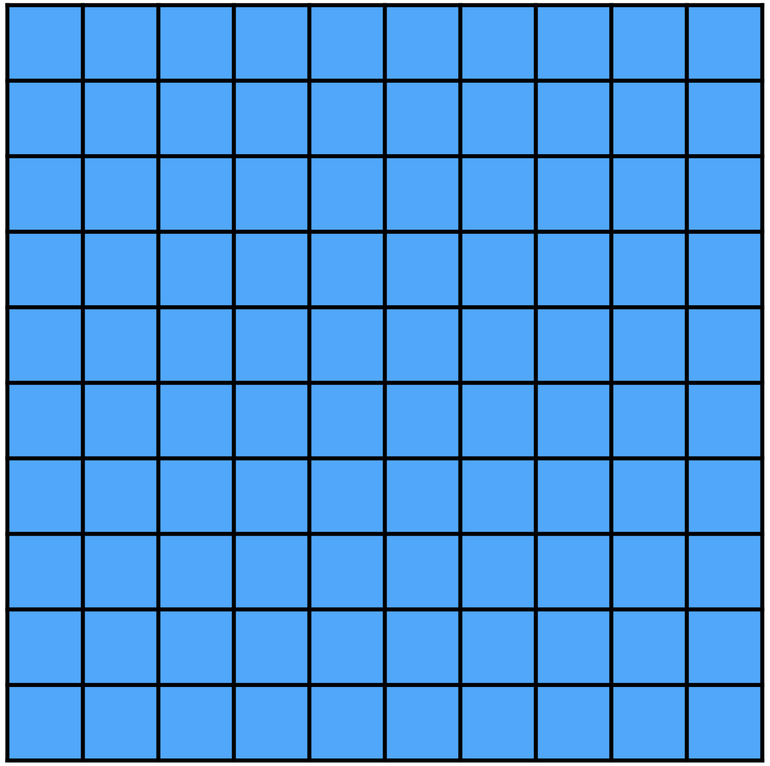
Padding

Zero-padding allows us to control the spatial size of the output — can be a hyperparameter

Image

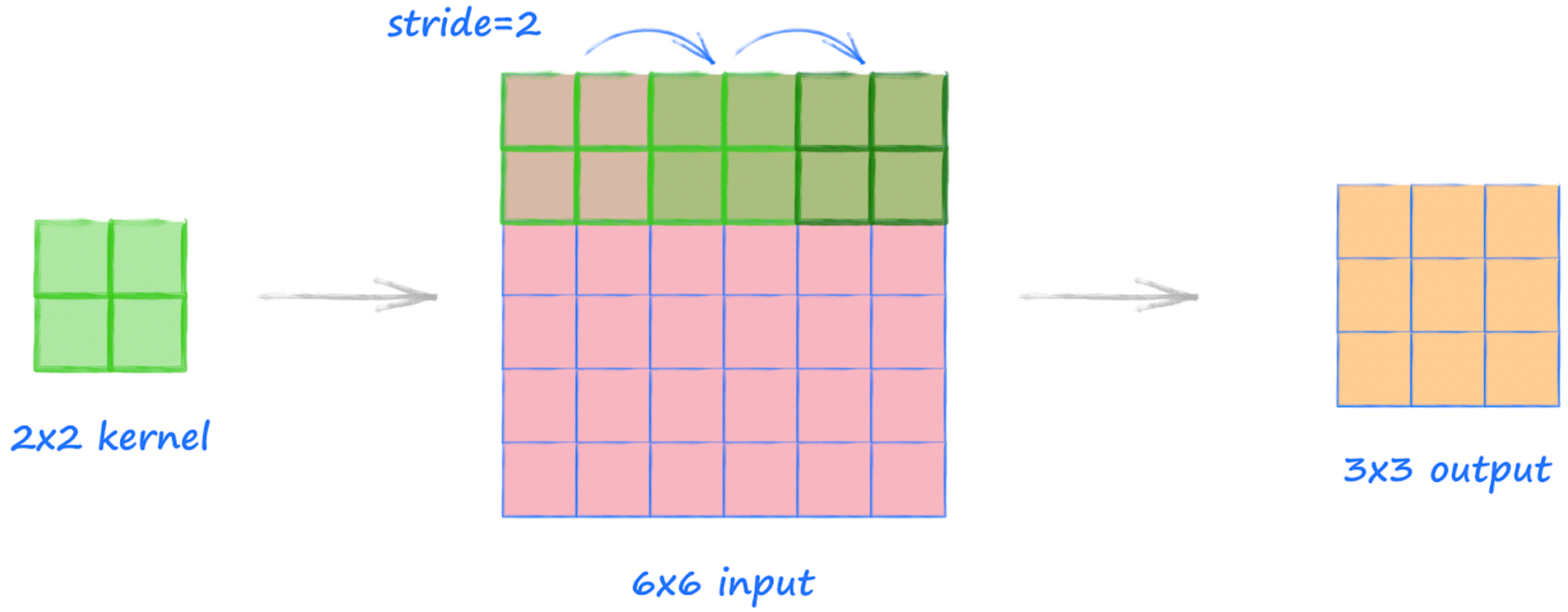


Output



Stride

Stride refers to how many pixels we shift the filter when convolving



http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html

Filter size

Filters can be of any spatial dimension smaller than or equal to the image size, but must match the number of channels in the image

The filter size is also known as the **receptive field** of the neuron

3x3

0.91	0.32	0.07
0.73	0.26	0.81
0.53	0.68	0.14

5x5

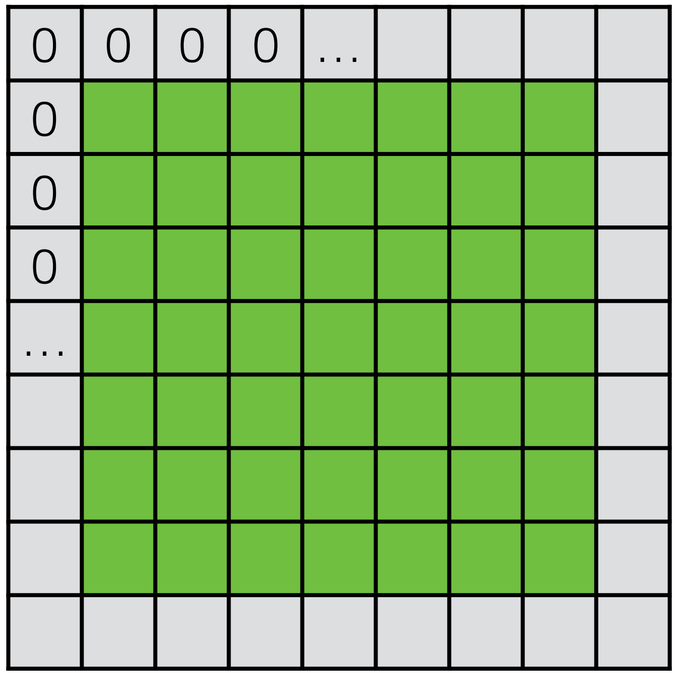
0.27	0.64	0.44	0.84	0.29
0.28	0.06	0.89	0.99	0.33
0.64	0.67	0.08	0.38	0.03
0.04	0.31	0.16	0.57	0.08
0.87	0.85	0.97	0.71	0.96

Bigger filters have wider receptive fields — but this can also be accomplished with a deeper CNN

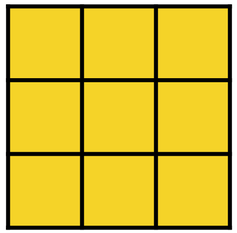
Poll: Output dimensions

Given an input image of dimension 7x7x1, one filter of size 3x3x1, 1-pixel thick zero-padding, and a stride of 3, what is the output size?

Image



Filter

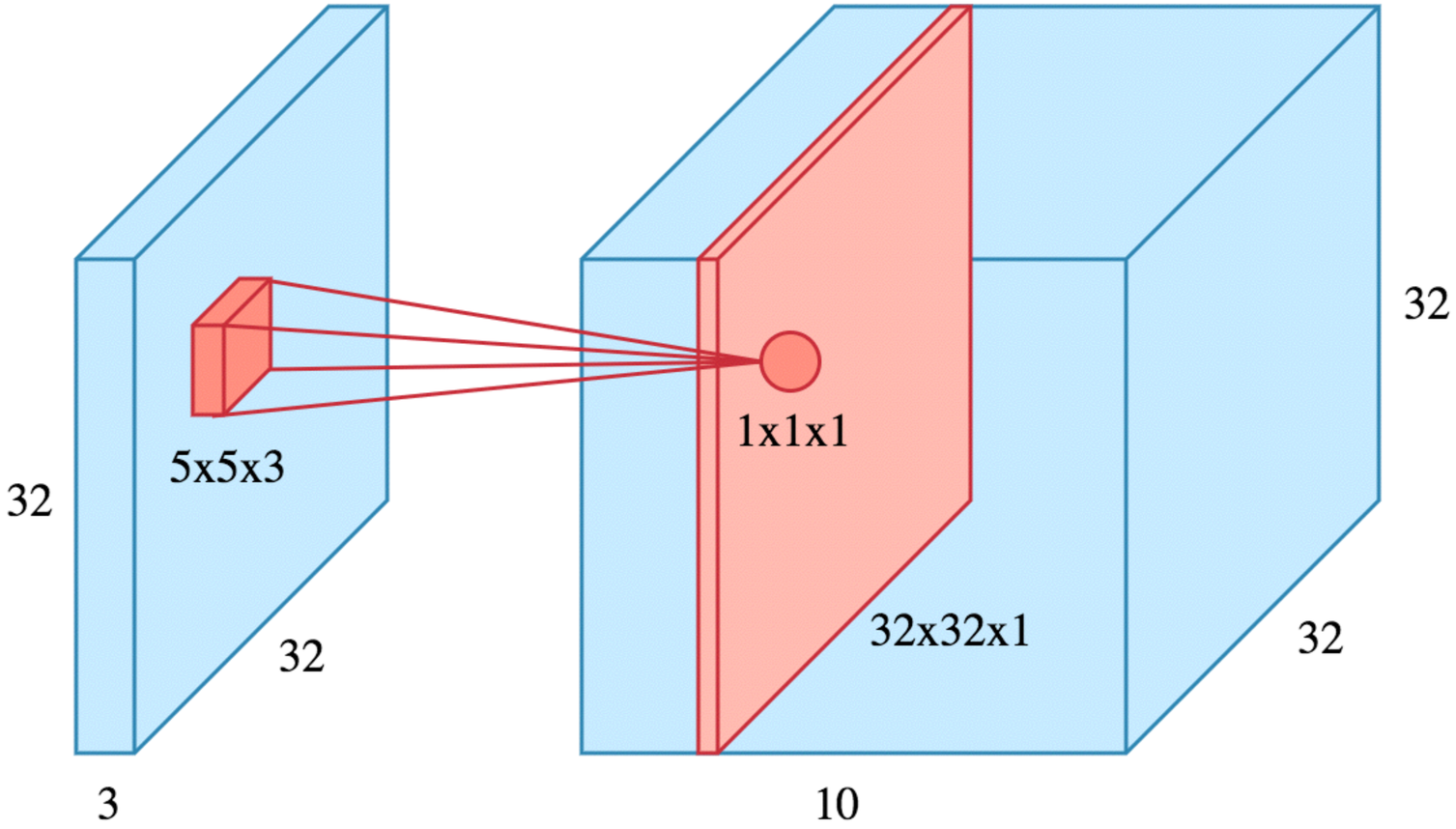


A. 3x3x1
B. 4x4x1
C. 5x5x1
D. 7x7x1

Go to PollEv.com/dlworkshop2020

Depth

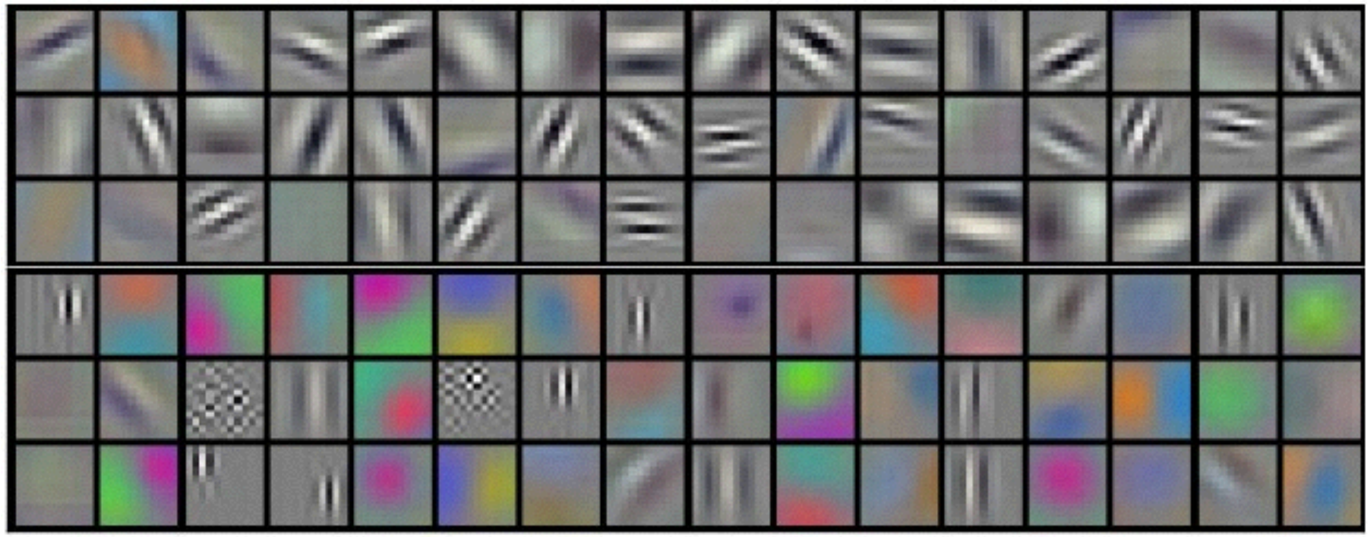
The number of filters used — more filters mean more learnable features



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

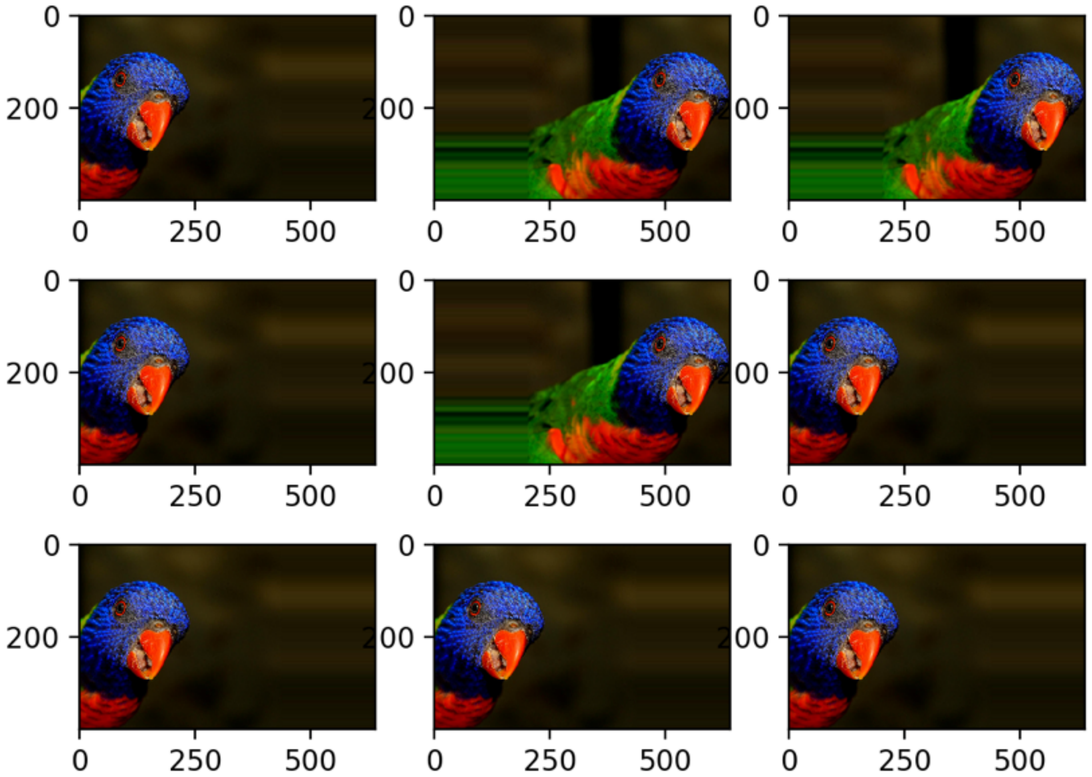
Example learned filters

- 96 filters learned by the AlexNet architecture trained on ImageNet
- Each filter is of dimension 11x11



Why use convolutions?

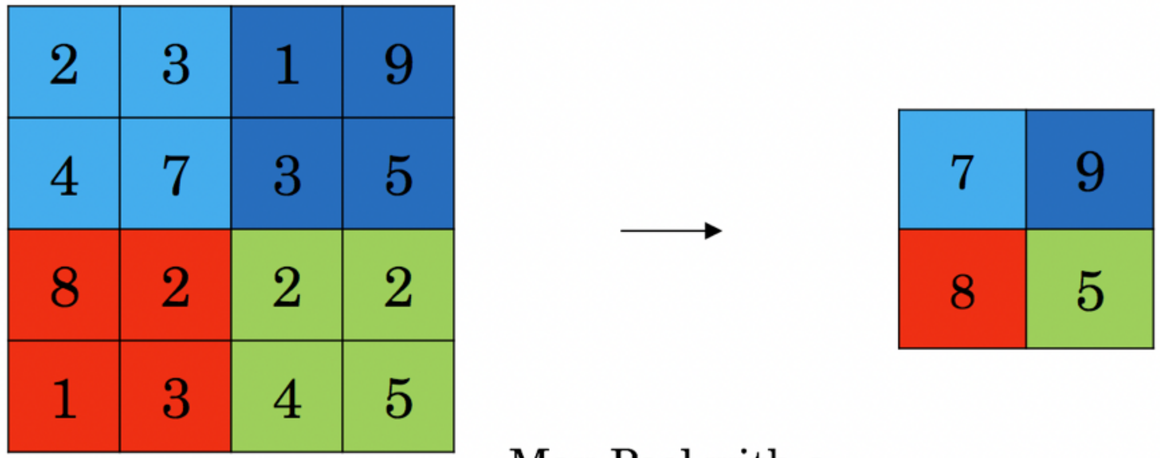
- Features in images should be invariant to translations
- Weight sharing — reduces the number of parameters to learn



No matter where the parrot is in the image, the image still contains a parrot

Pooling layers

It is common to insert pooling layers between successive Conv layers.
Pooling reduces the spatial size of the representation.
The most common pooling layer:

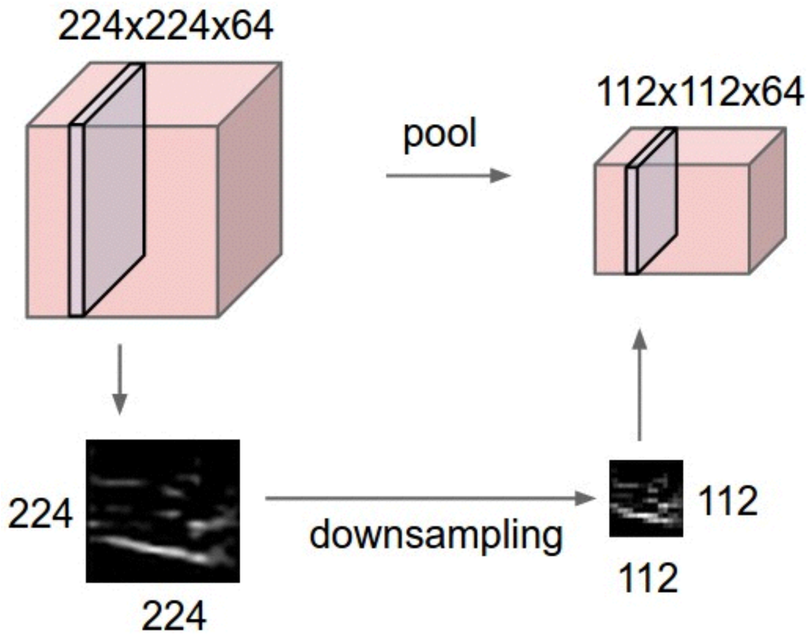


Max-Pool with a 2 by 2 filter and stride 2.

Andrew Ng

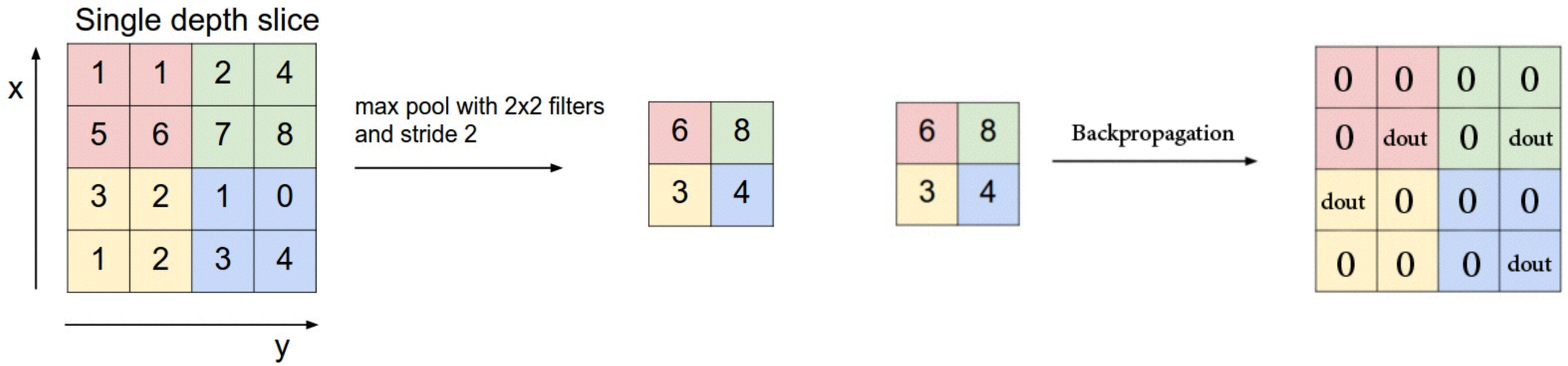
Pooling layers

Pooling layers contain no trainable parameters.
It is uncommon to use padding in pooling layers.



Recall backpropagation

Backprop through max-pooling layers routes the gradient through the index of the max activation



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

Stacking layers

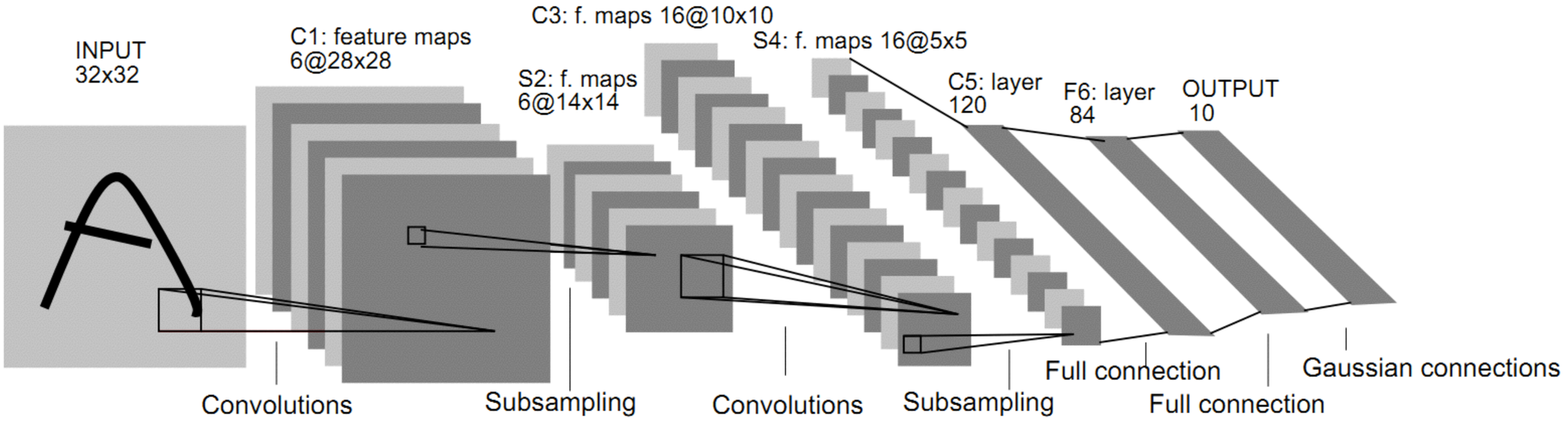
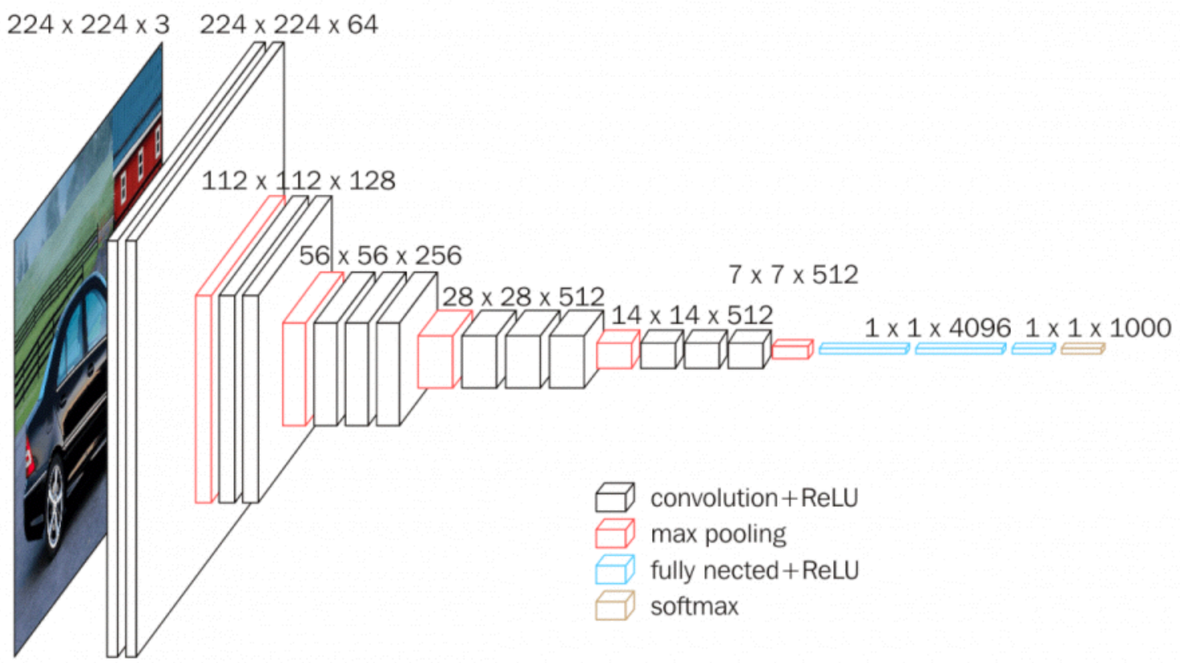


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Stacking layers

VGG-16

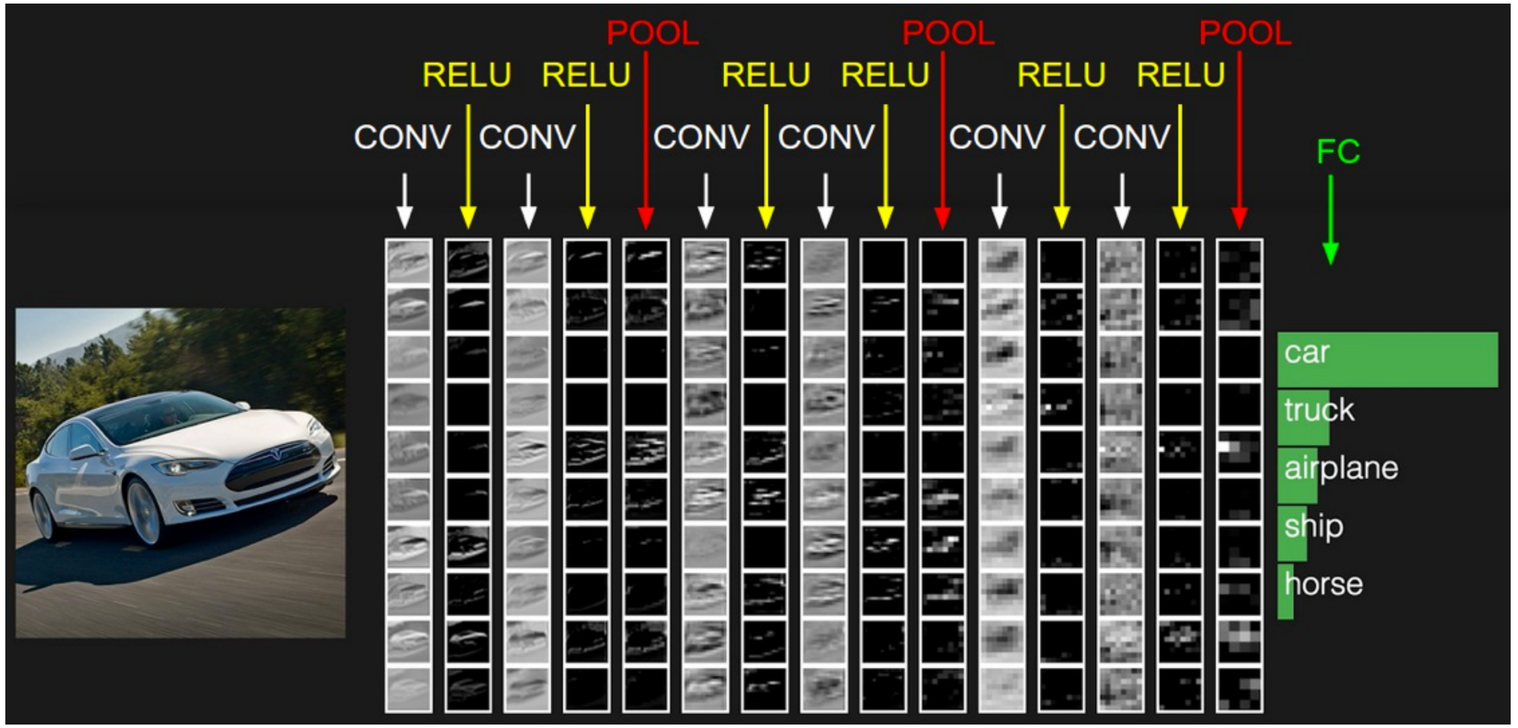


https://neurohive.io/en/popular-networks/vgg16/

Stacking layers

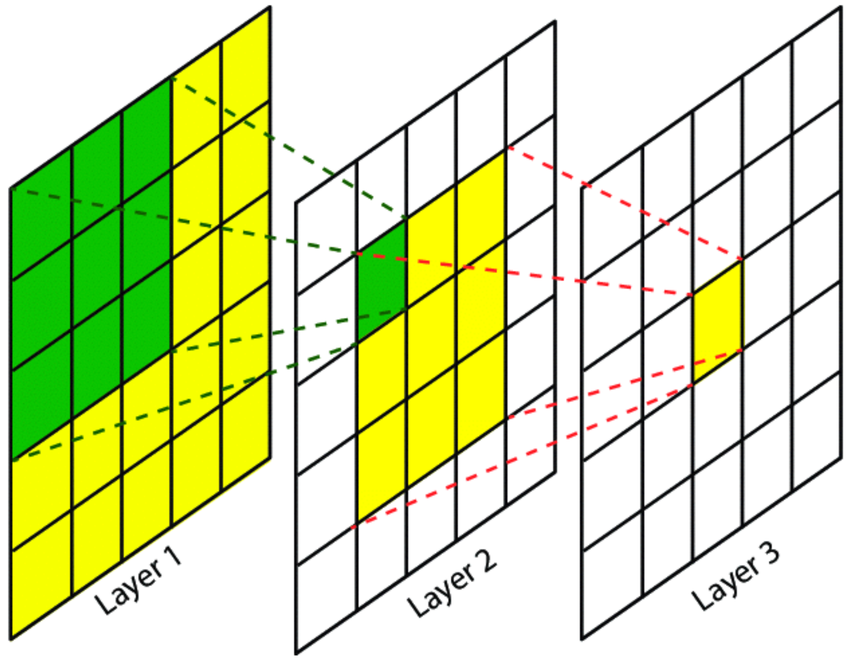
CNNs commonly follow the pattern:

Input —> ((Conv —> ReLU) * N —> Pool) * M —>
(FC —> ReLU) * K —> FC —> Output



Growing the receptive field

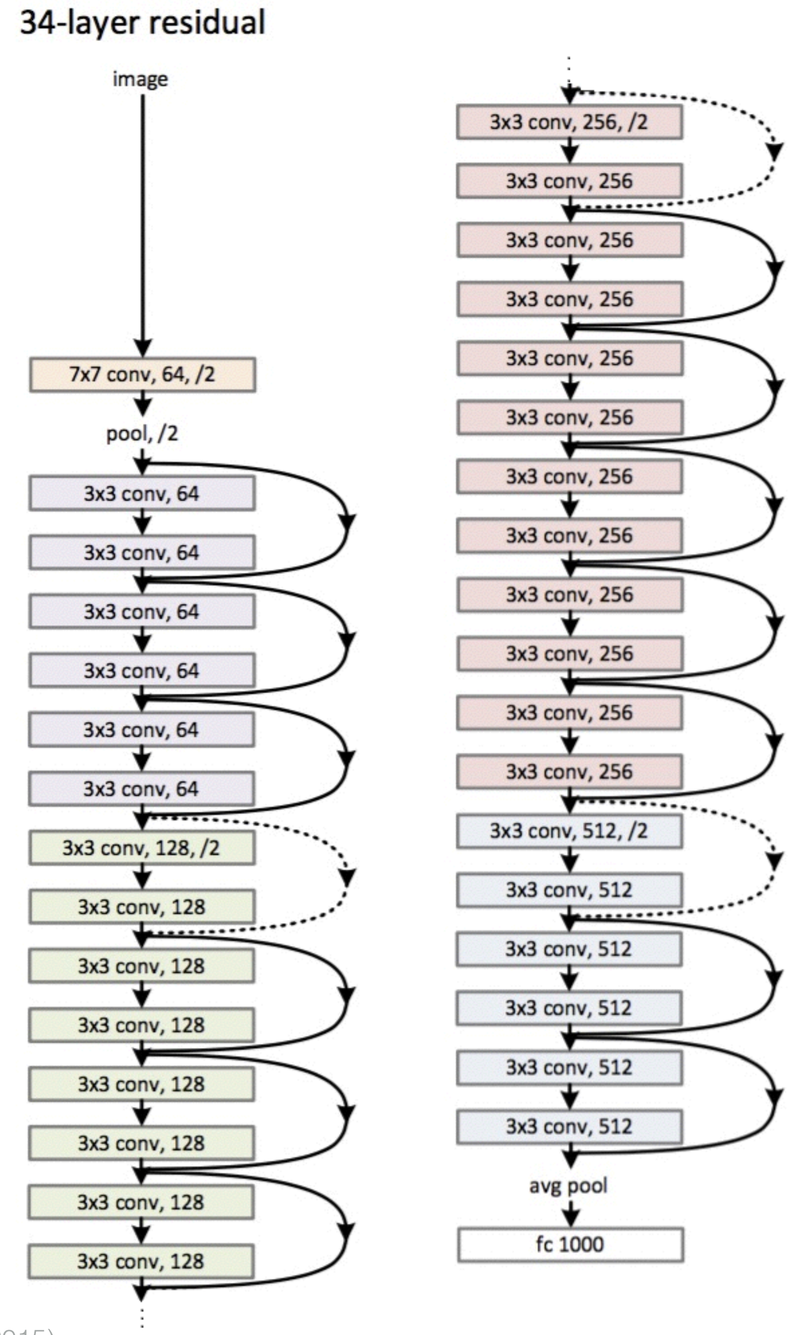
Deeper layers “see” more and more of the input all at once



https://dev.to/jai00271/cnn-general-terms-and-their-meaning-42nc

ResNet

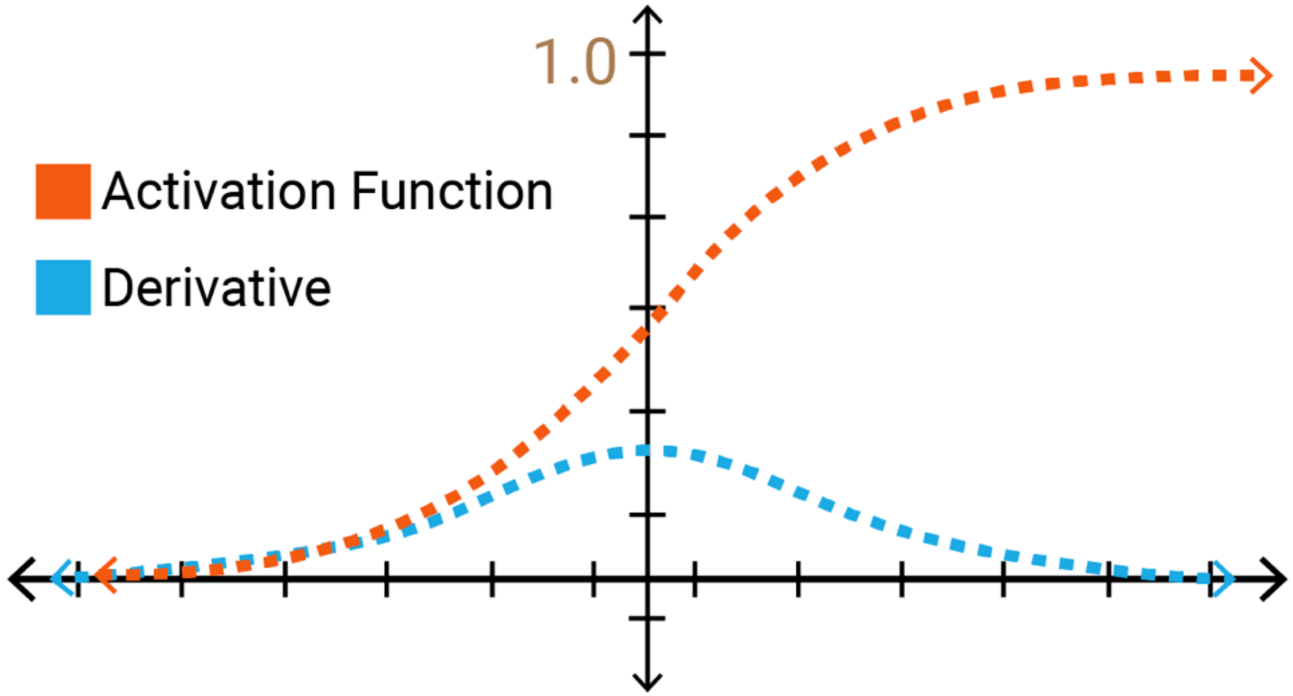
Residual neural network (ResNet) is one of the most commonly used architectures.
It contains **skip connections** that jump over some layers.
Skip connections improve training by avoiding vanishing gradients.



He *et al.* “Deep Residual Learning for Image Recognition” (2015)

Recall: Vanishing gradients

Gradients can become very small after stacking many layers of a neural network. This makes training difficult.



Model expressiveness

Number of Layers	Number of Parameters
ResNet 18	11.174M
ResNet 34	21.282M
ResNet 50	23.521M
ResNet 101	42.513M
ResNet 152	58.157M

Recap: CNNs

CNNs take advantage of the structure of images.

They use:
- Convolutional layers
- Pooling layers

Parameters are shared for more efficient learning and translational invariance.

Convolutional layer hyperparameters: filter size, padding, stride

ICME Summer Workshops 2020

Introduction to Deep Learning

Session 4: 3:30—4:45 PM
Instructor: Sherrie Wang

icme-workshops.github.io/deep-learning

Eykholt *et al.* “Robust Physical-World Attacks on Deep Learning Visual Classification” (2018)

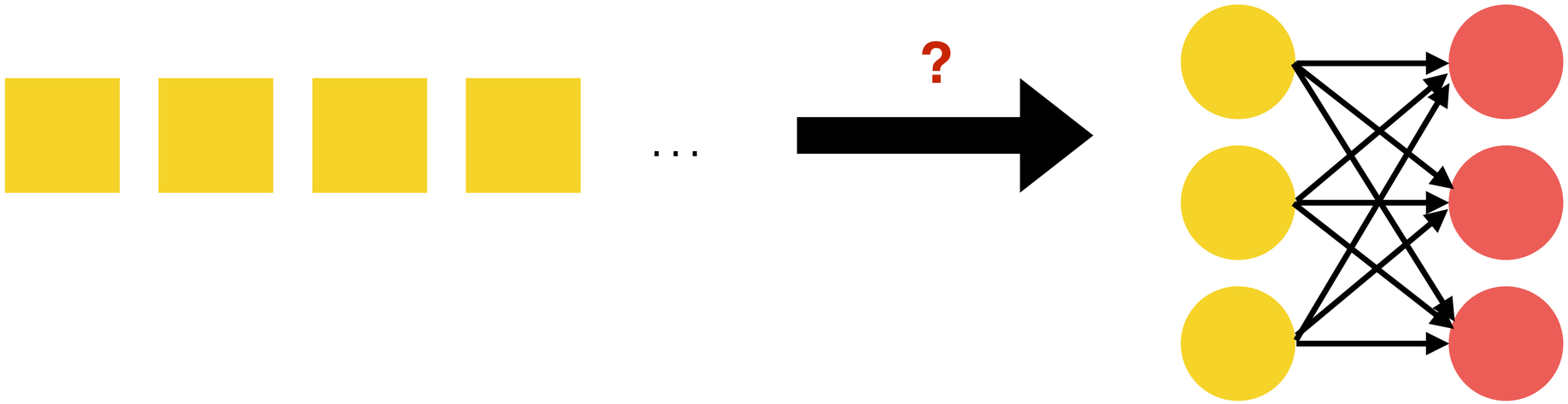
Recurrent Neural Networks

Recurrent neural networks

How to use neural networks for time series input?

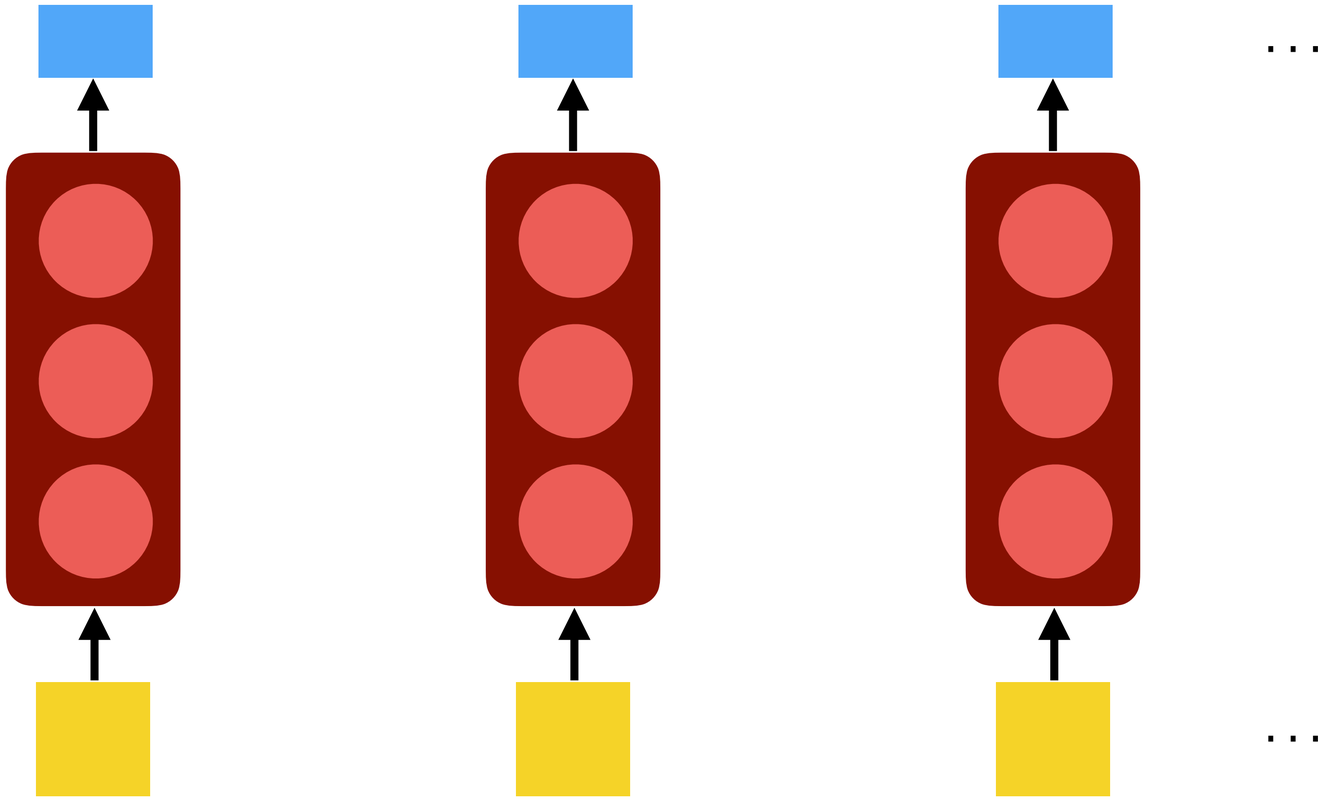
The length of input is not fixed or of indefinite length

Examples: audio, text, financial time series



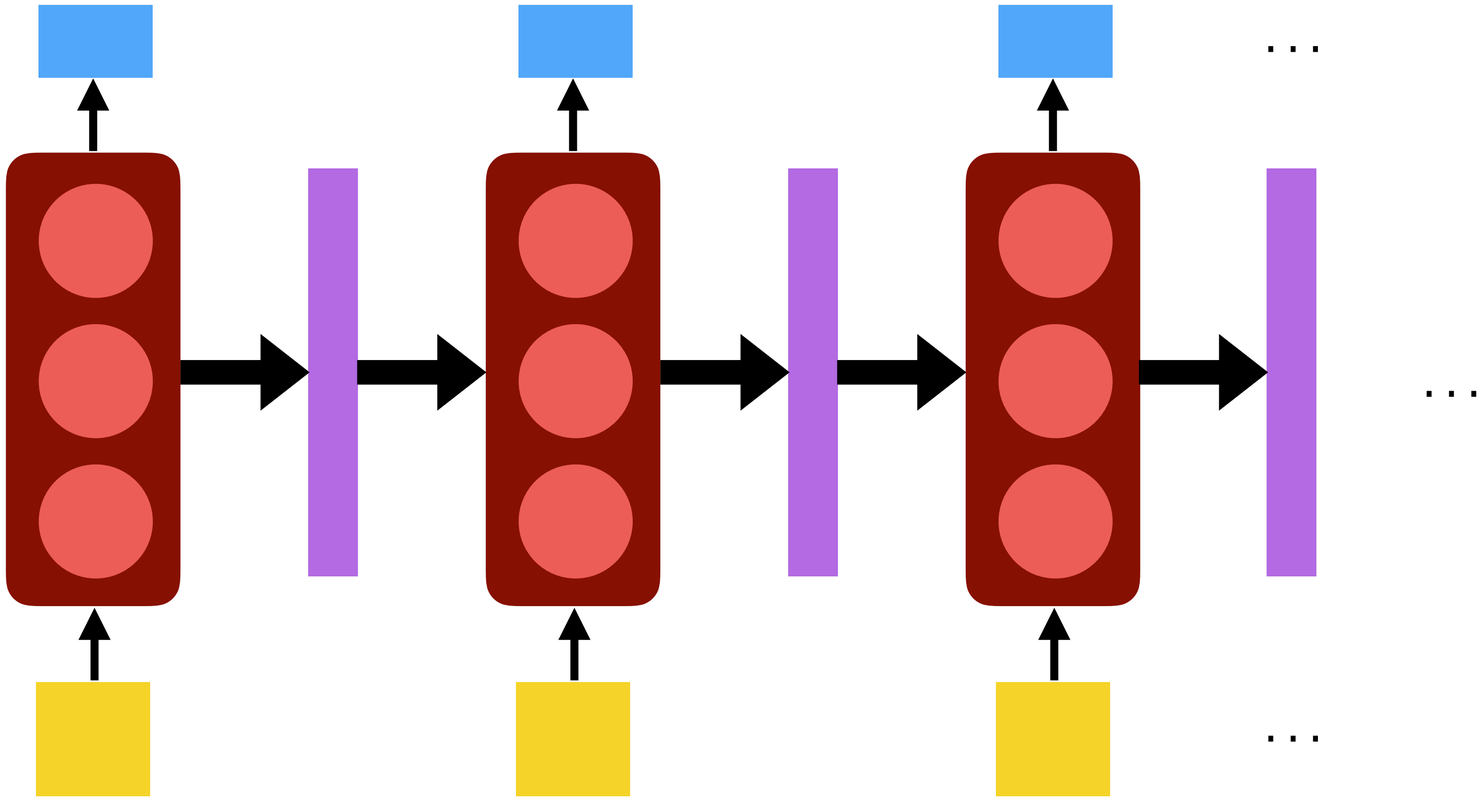
Recurrent neural networks

Share weights for each time step



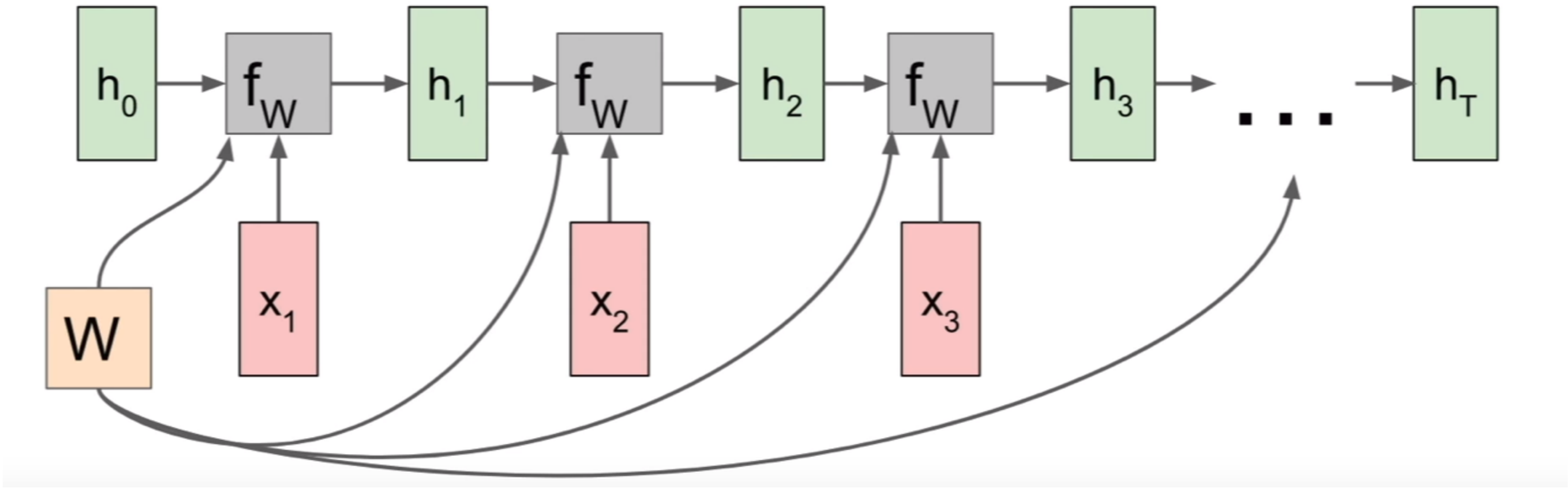
Recurrent neural networks

Pass information across time steps using **hidden states**



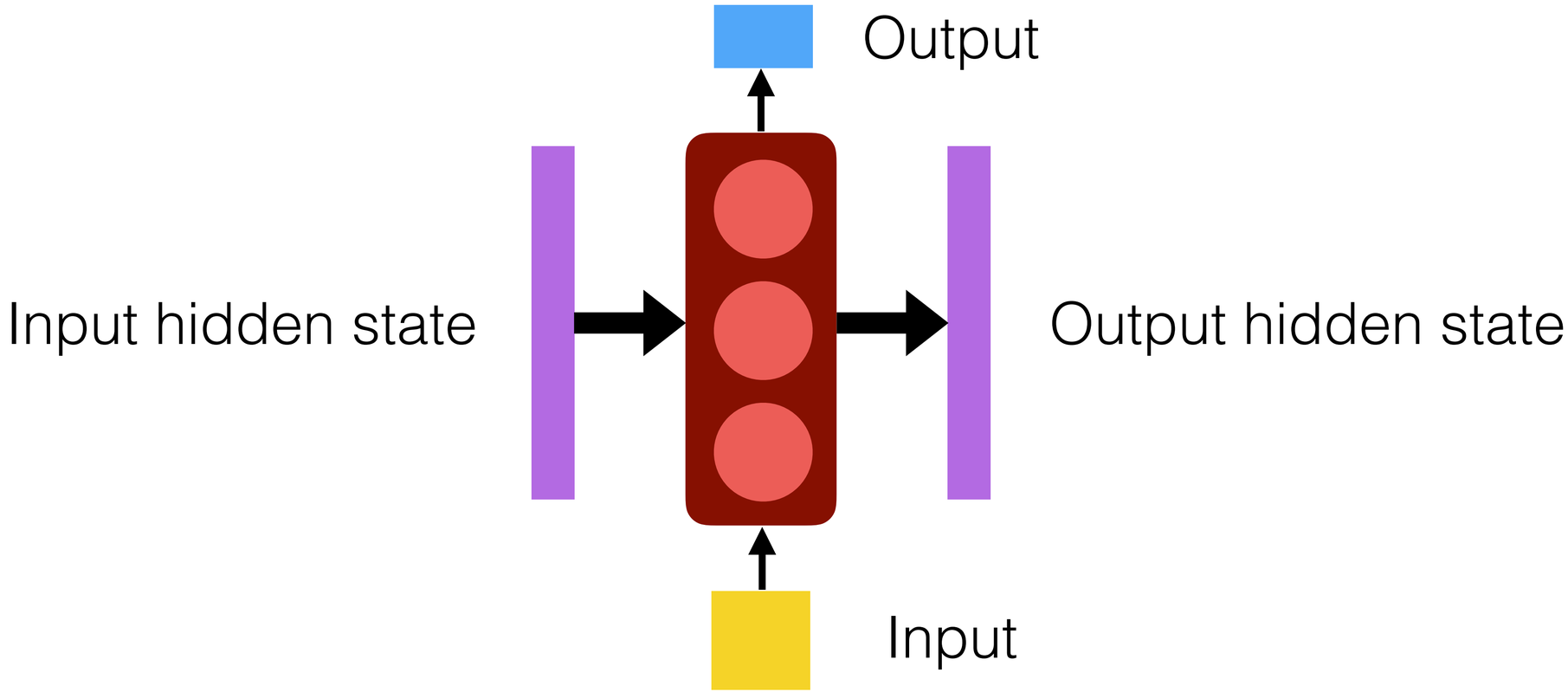
Recurrent neural networks

Computational graph view of things:

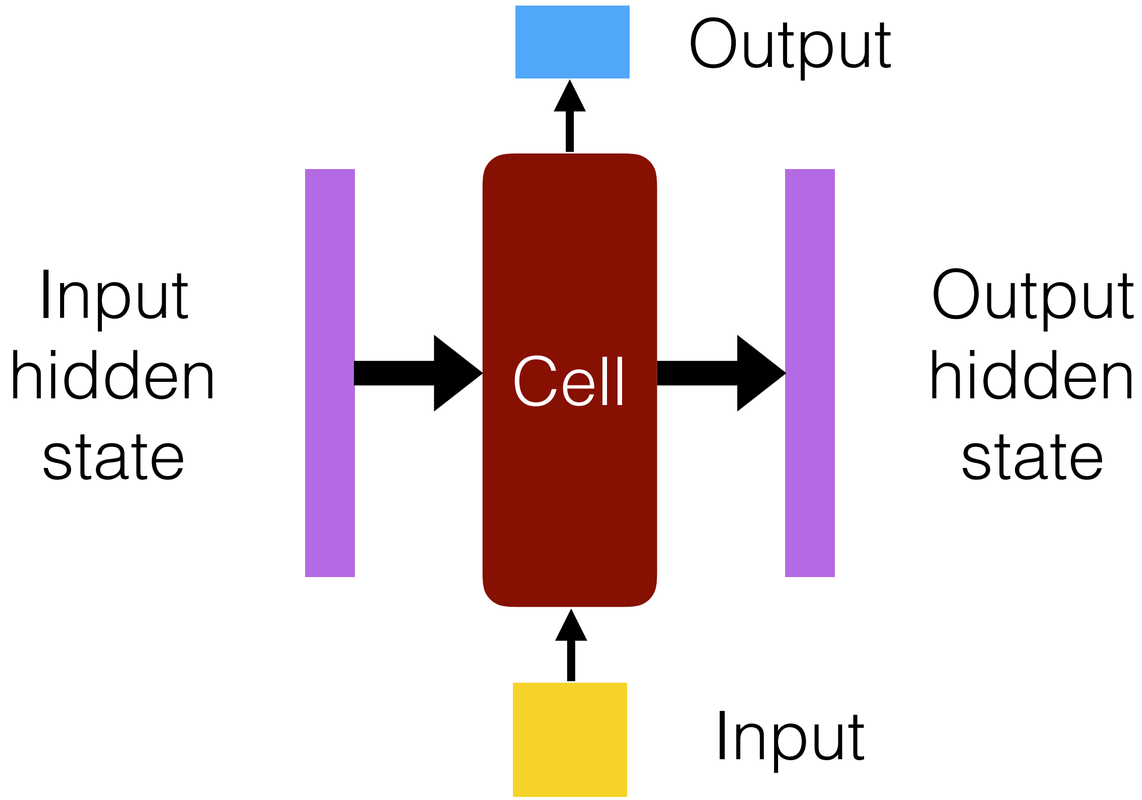


https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

Hidden states



Types of cells

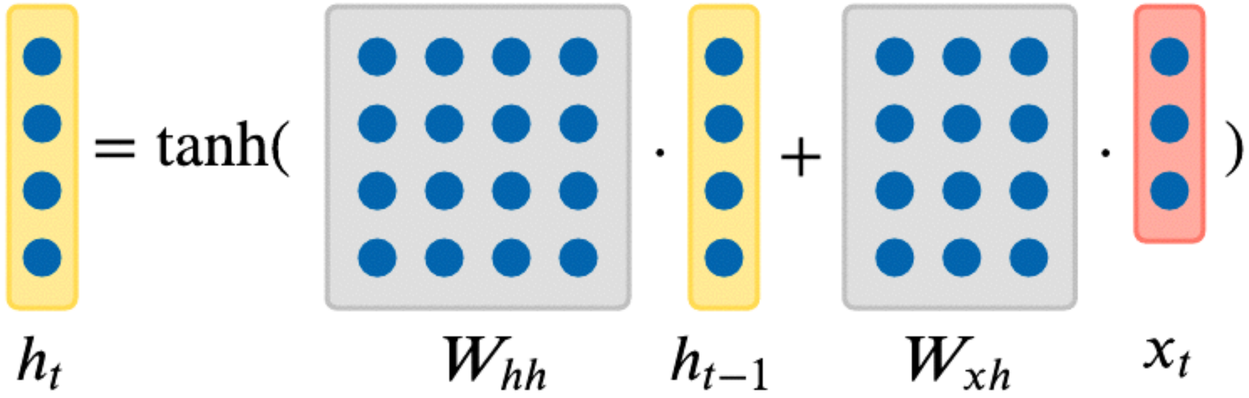


Many types of cells:

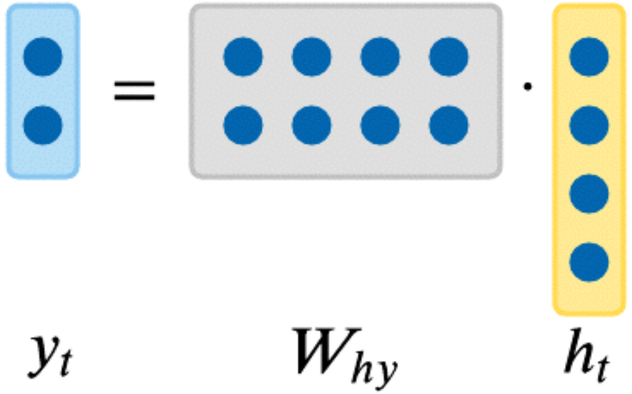
1. Vanilla RNN
2. LSTM - Long short term memory
3. GRU - Gated recurrent unit

Vanilla RNN

$$h_t = tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$



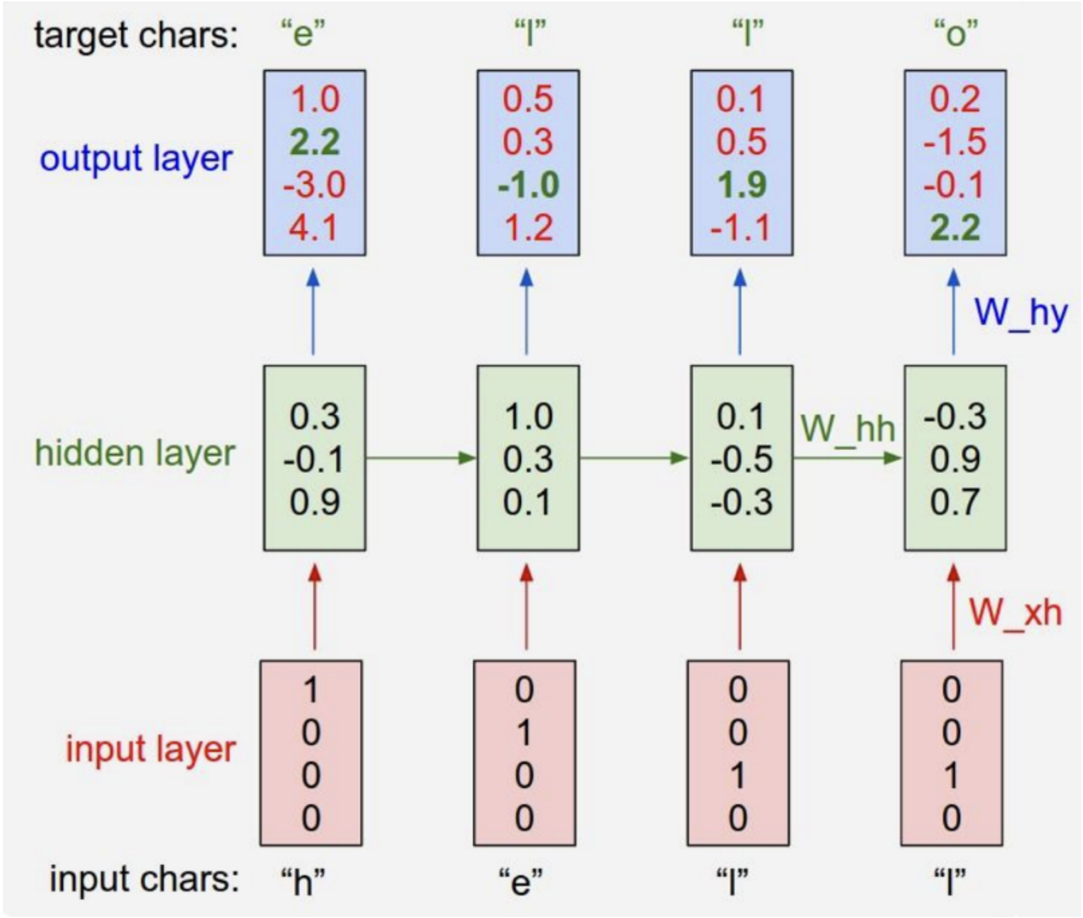
$$y_t = W_{hy} h_t$$



https://cs231n.github.io/rnn/

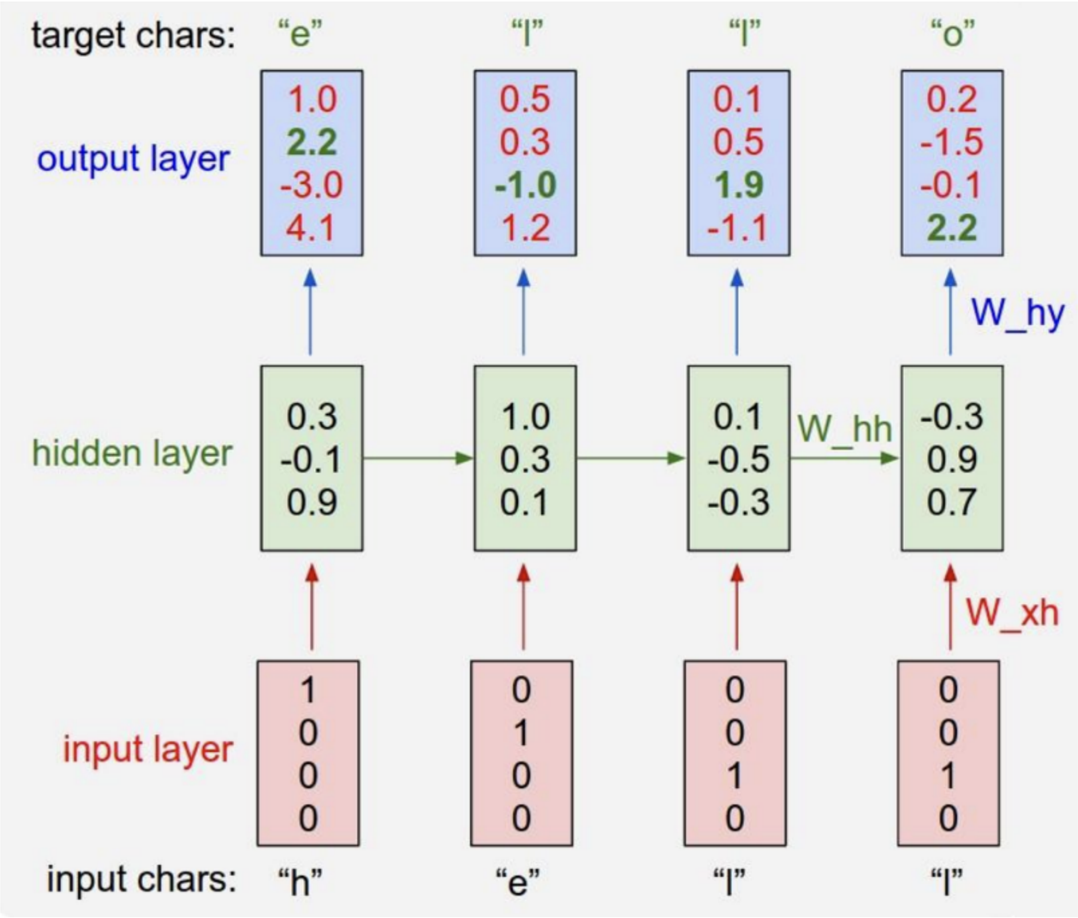
Example: Character prediction

Suppose we want to train an RNN to output the letters “hello” in order.



https://cs231n.github.io/rnn/

Example: Character prediction



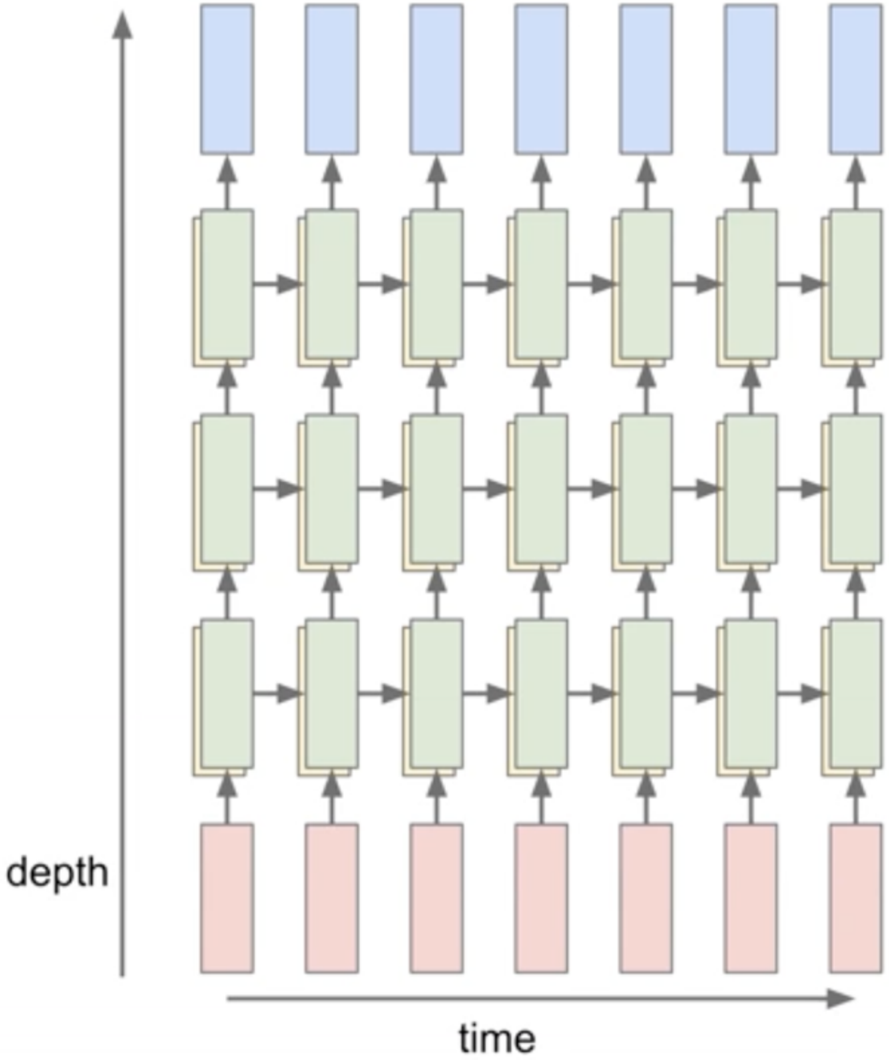
$$\begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} = f_W(W_{hh}\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + W_{xh}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}) \quad (1)$$

$$\begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} = f_W(W_{hh}\begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} + W_{xh}\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}) \quad (2)$$

$$\begin{bmatrix} 0.1 \\ -0.5 \\ -0.3 \end{bmatrix} = f_W(W_{hh}\begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} + W_{xh}\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}) \quad (3)$$

$$\begin{bmatrix} -0.3 \\ 0.9 \\ 0.7 \end{bmatrix} = f_W(W_{hh}\begin{bmatrix} 0.1 \\ -0.5 \\ -0.3 \end{bmatrix} + W_{xh}\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}) \quad (4)$$

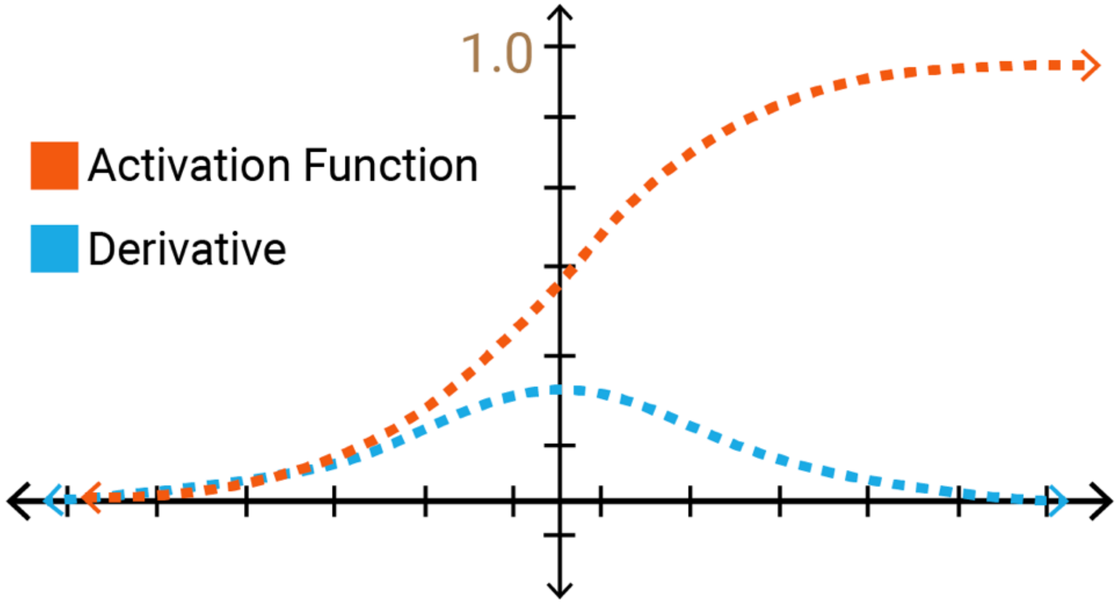
Multi-layer RNNs



Each transformation from input and hidden state to output can involve a deep neural network.

Hello vanishing gradients… again

Vanilla RNNs aren’t used in practice anymore, because they are hard to train.



$$\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W_{hh}}$$

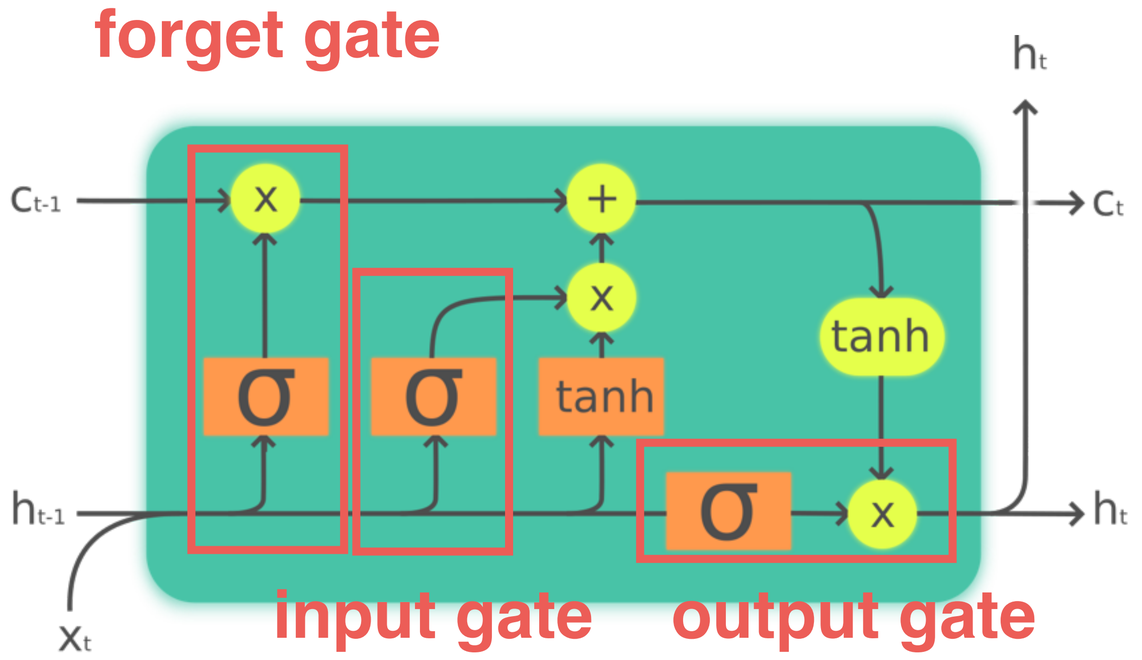
$$= \frac{\partial L_t}{\partial h_t}\left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right)\frac{\partial h_1}{\partial W_{hh}}$$

$$= \frac{\partial L_t}{\partial h_t}\left(\prod_{t=2}^{T} tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}^{T-1}\right)\frac{\partial h_1}{\partial W_{hh}}$$

LSTM

In addition to a hidden state, has a “cell state” to store long term information

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$



LSTM

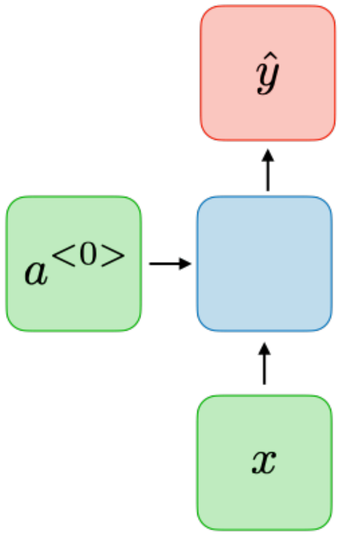
Forget gate controls how much information is forgotten from the previous cell state.

Input gate controls how much information from the previous hidden state and current input are added to the cell state.

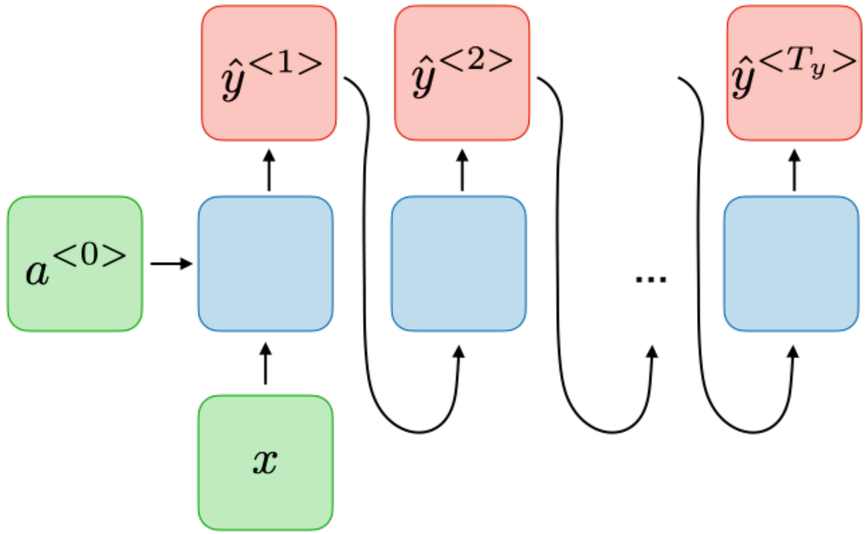
Output gate controls how much information is passed to the next hidden state and output.

The presence of the cell state ameliorates the vanishing gradient problem.

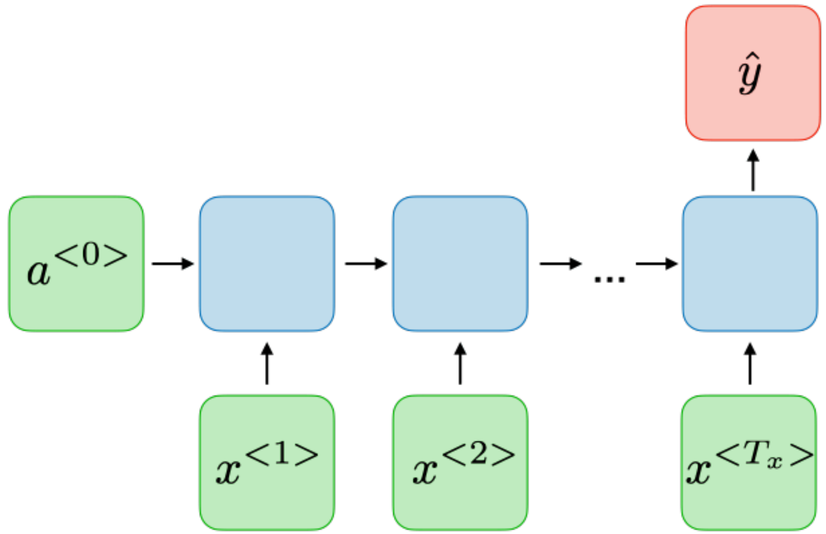
How many outputs?



One-to-one



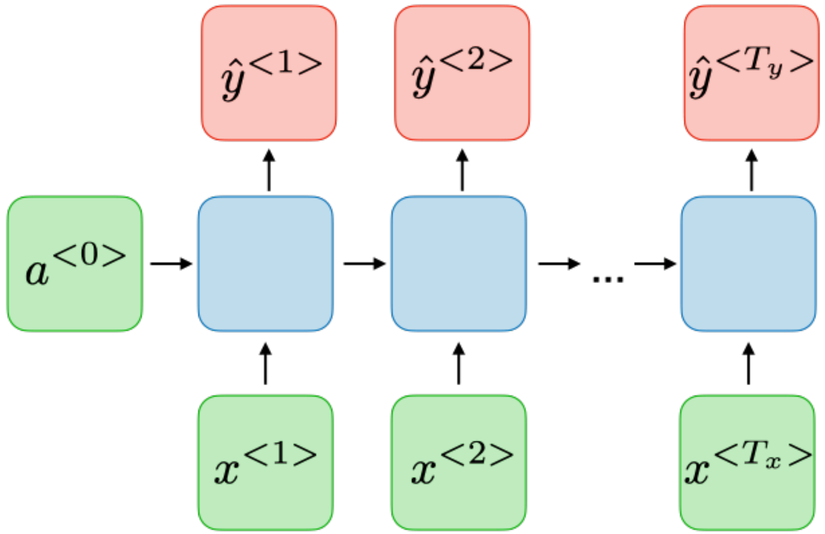
One-to-many



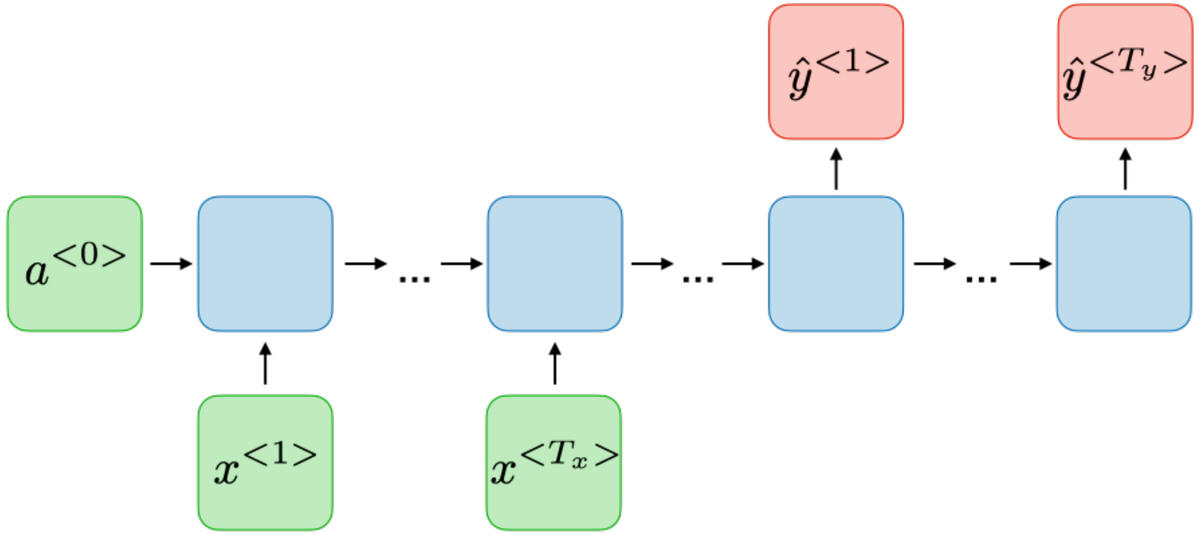
Many-to-one

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

How many outputs?



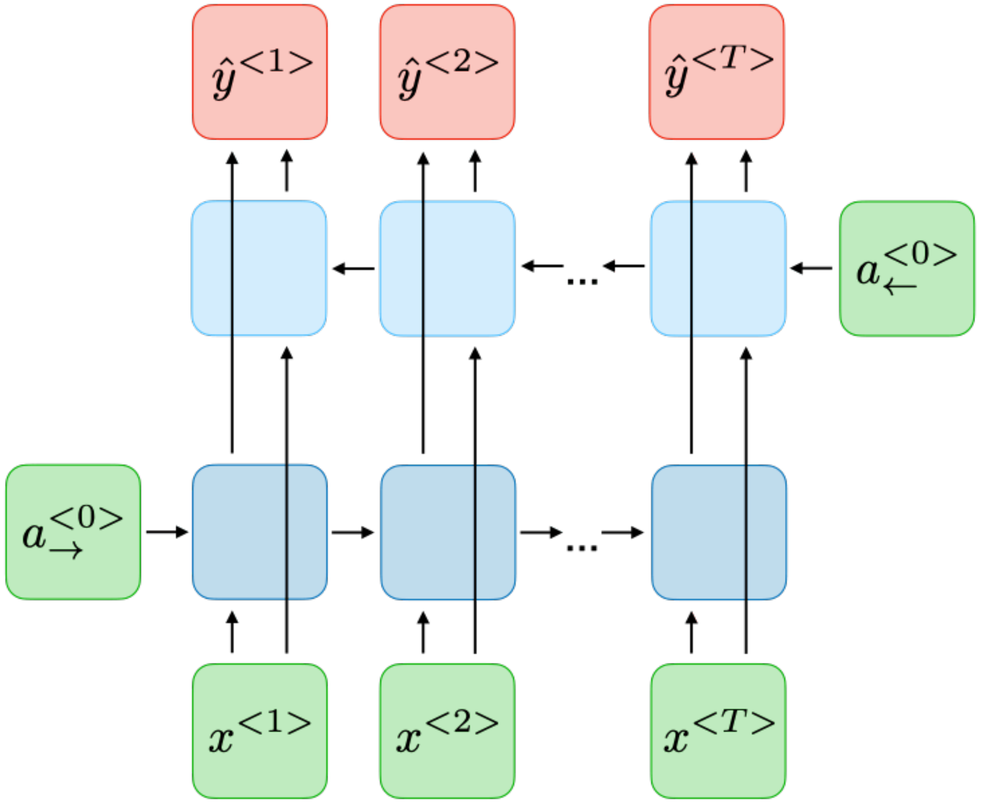
Many-to-many



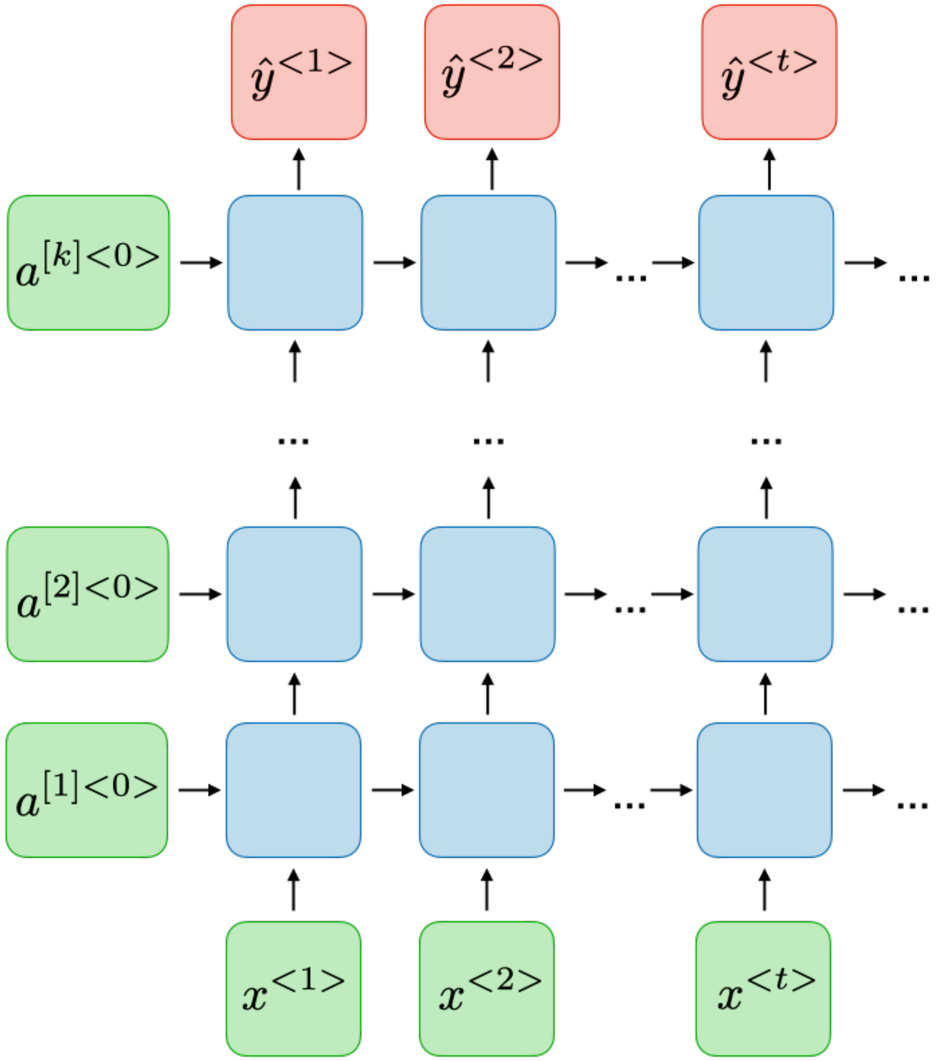
Many-to-many

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

RNN variants



Bidirectional



Deep

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Recap: RNNs

RNNs allow for processing of variable-length inputs

Weights are shared across time steps; model size does not increase with size of input

Computation is often slow, and the network “forgets” information from a long time ago

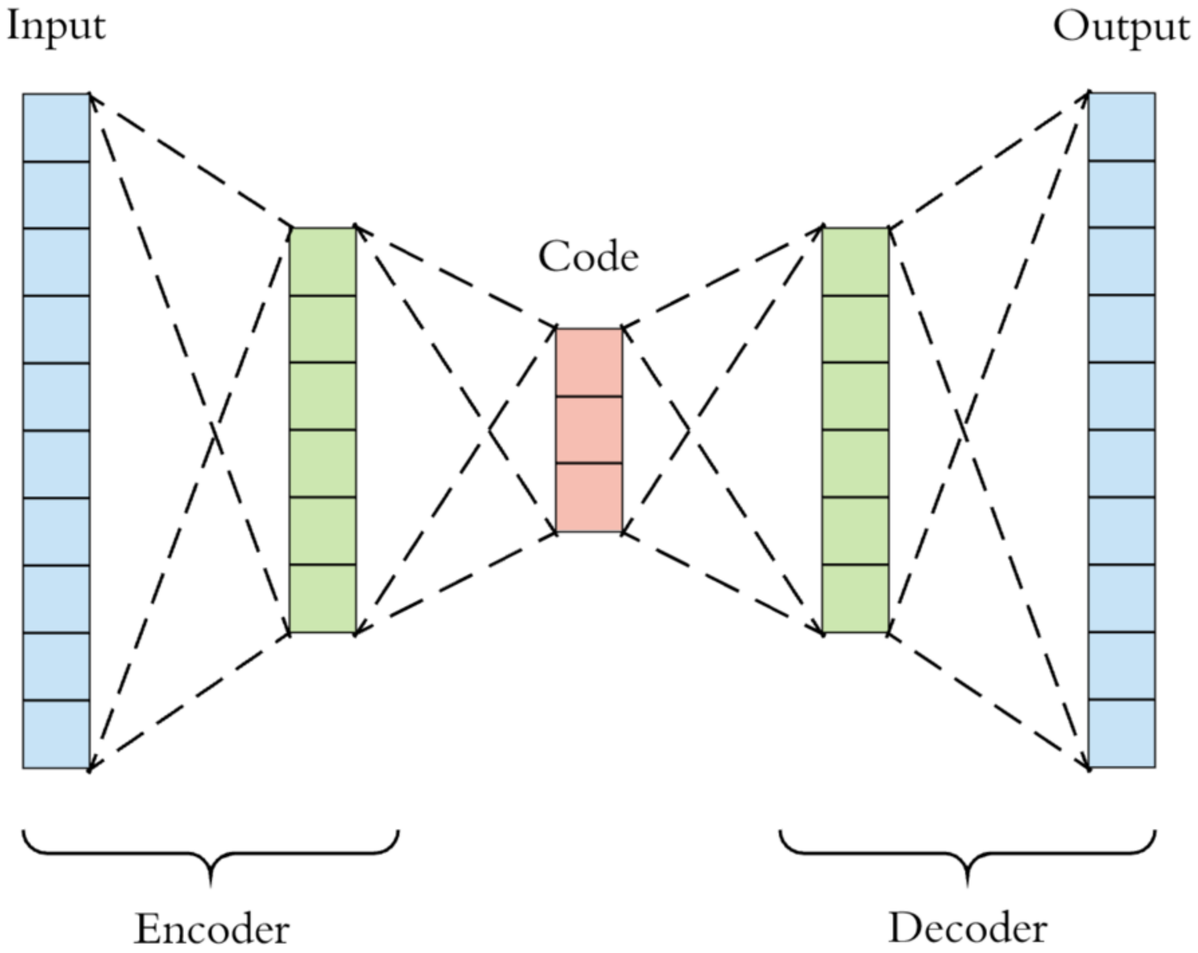
Other Architectures

Autoencoders

A way to do unsupervised deep learning

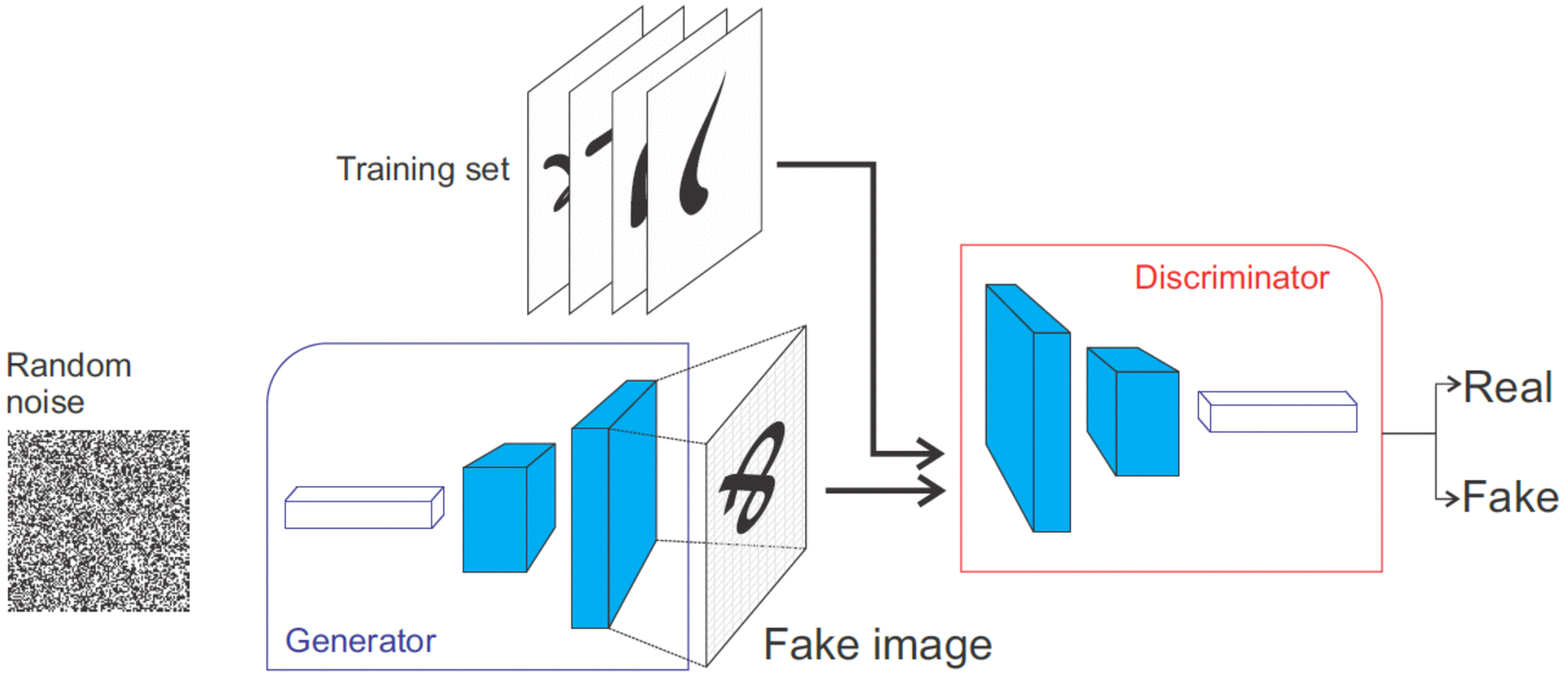
Input and output are the same: the network is tasked with recreating the input but going through a lower-dimensional bottleneck

Bottleneck layer is interpreted as a latent representation for the input



https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

Generative adversarial networks



https://www.oreilly.com/library/view/java-deep-learning/9781788997454/60579068-af4b-4bbf-83f1-e988fbe3b226.xhtml

Generative adversarial networks

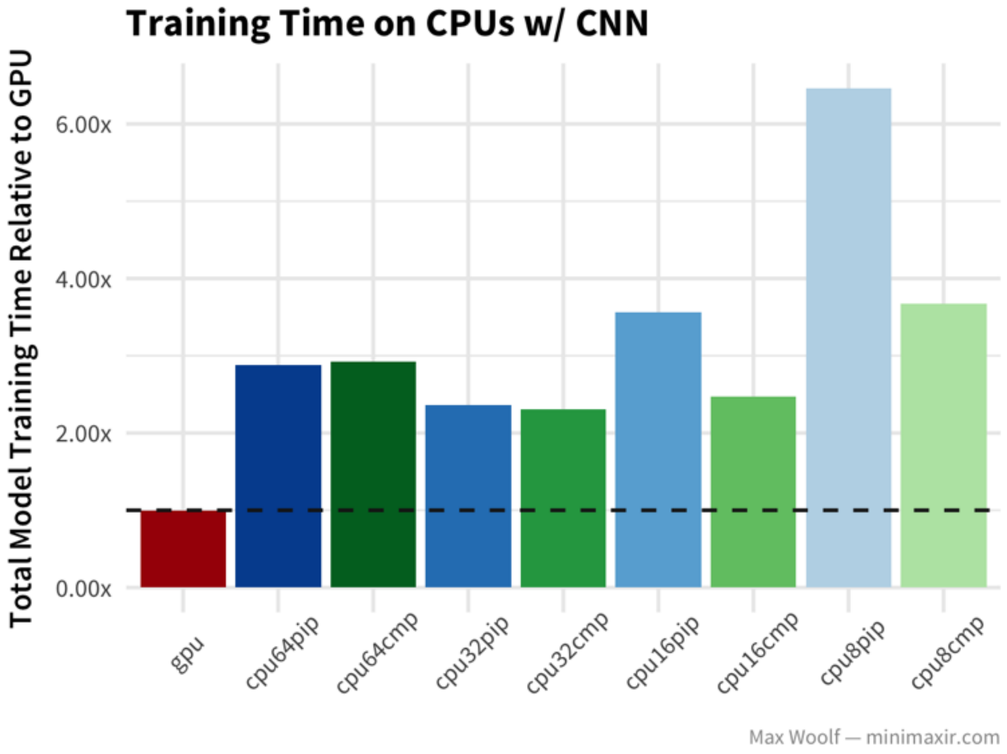
Deep Learning Libraries

What do deep learning packages provide?

1. Ability to run on GPUs
2. Build computation graphs and provide automatic differentiation
3. Useful medium- to high-level components for deep learning already implemented (convolutions, batch norm, VGG, ResNet, etc.)

CPUs vs GPUs

GPUs are optimized to perform matrix multiplications, and speed up neural network training many-fold



Python: The preferred language of DL

1. Simplicity of syntax
2. Extensive libraries
   - Bumpy for scientific computing
   - Pandas for high-level data structures
   - Scikit-learn for ML
3. Open source, good support
4. And now, feedback from TensorFlow, PyTorch, etc.

Deep learning libraries

1. TensorFlow — mainly used in industry
2. Keras — easy-to-use API for TensorFlow
3. PyTorch — mainly used in research
4. Others include Caffe, Theano, MXNET…

Caffe

Worth mentioning because it was the first mainstream library, developed by Berkeley AI Research (BAIR).

Designed for CNNs, amassed a large model zoo.

Not good for RNNs, verbose.

Caffe

TensorFlow

Developed by Google Brain.

Production-grade deep learning library. Used frequently for development and production.

First version had a steep learning curve and was difficult to debug. TensorFlow 2.0 released in 2019 made the library more user-friendly and intuitive (or at least, more familiar to Python users).

Keras is a high-level API integrated into TensorFlow in 2017.

Great visualization tools in TensorBoard.

PyTorch

Developed at Facebook.

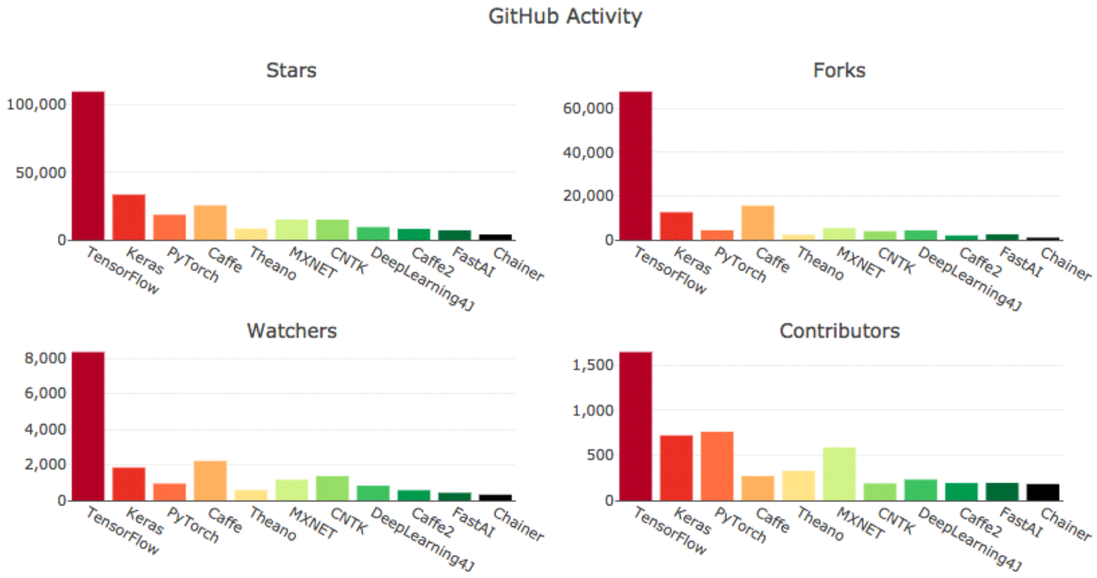
Very natural for Python programmers to learn and use; if you already use Python, learning curve will be relatively shallow.

Favored by researchers, also used by professional developers now.

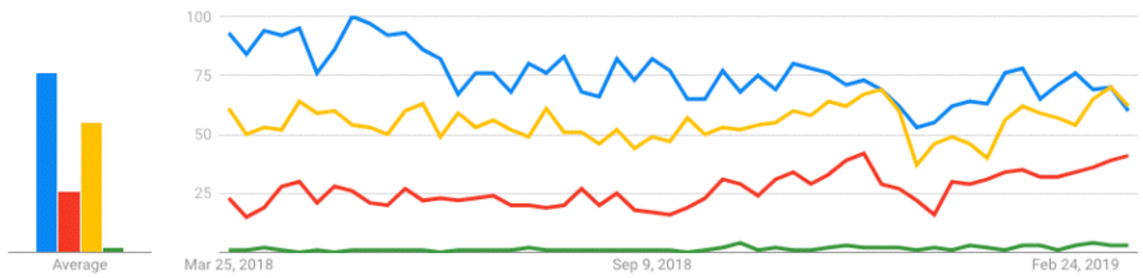
Inspired TensorFlow 2.0.

Visualization done in Visdom.

Which one?



Google search interest



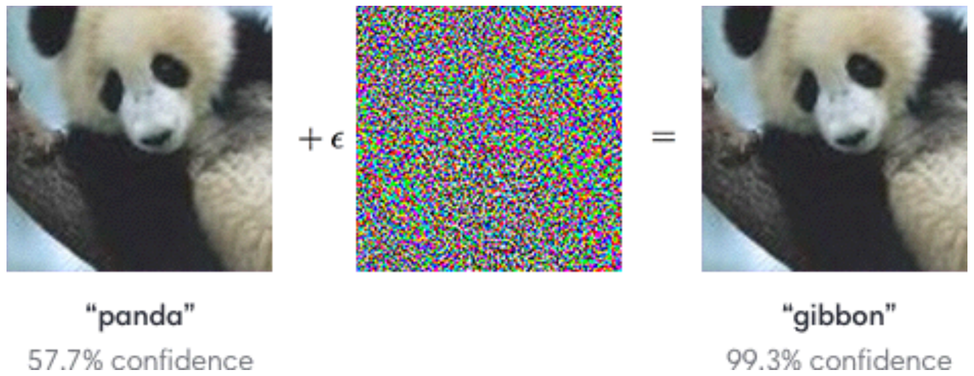
TensorFlow in blue; Keras in yellow, PyTorch in red, fastai in green

Best fit will depend on programming language preferences, model zoo, deployment, …

Failures and Limits

Adversarial attacks

- Noise that is imperceptible to humans has been shown to dramatically change neural network output



Real-world adversarial attacks

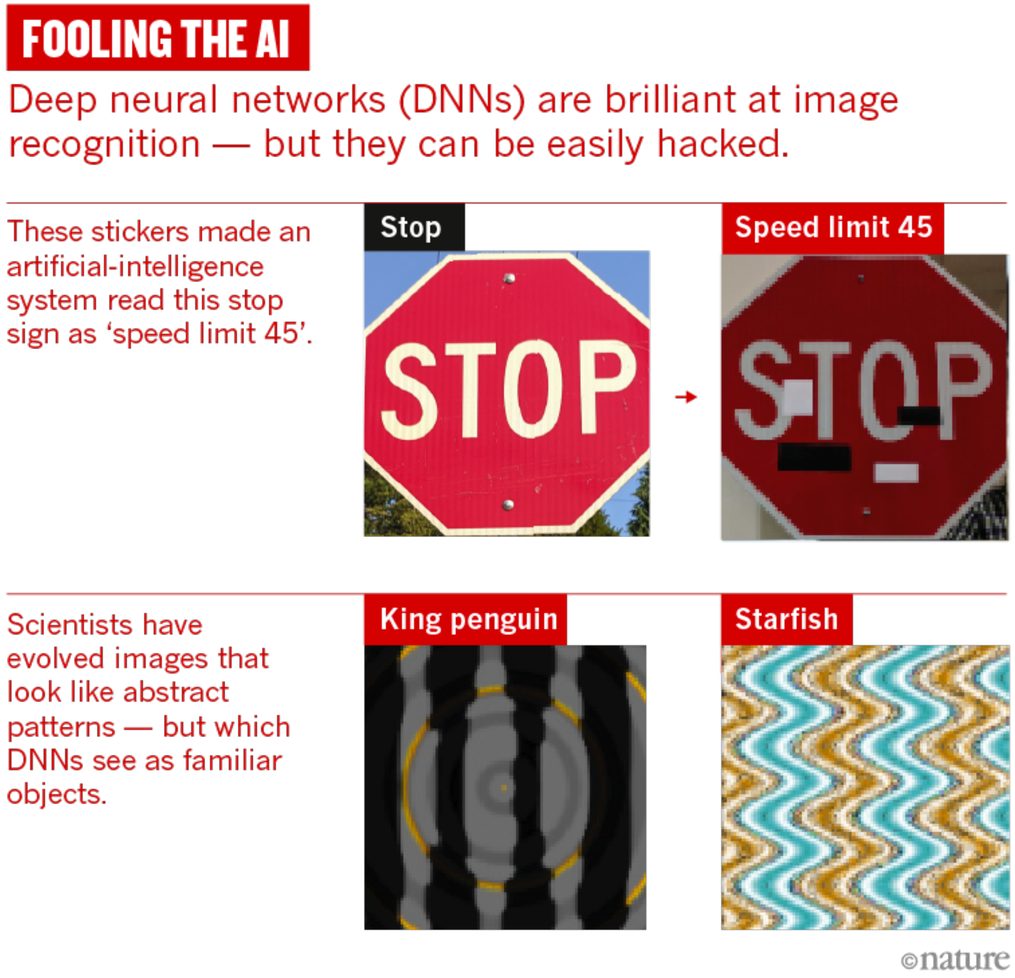
- Some self-driving cars rely on convolutional neural networks…

Table 1: Sample of physical adversarial examples against LISA-CNN and GTSRB-CNN.

Distance/Angle	Subtle Poster	Subtle Poster Right Turn	Camouflage Graffiti	Camouflage Art (LISA-CNN)	Camouflage Art (GTSRB-CNN)
5′ 0°					
5′ 15°					
10′ 0°					
10′ 30°					
40′ 0°					
Targeted-Attack Success	100%	73.33%	66.67%	100%	80%

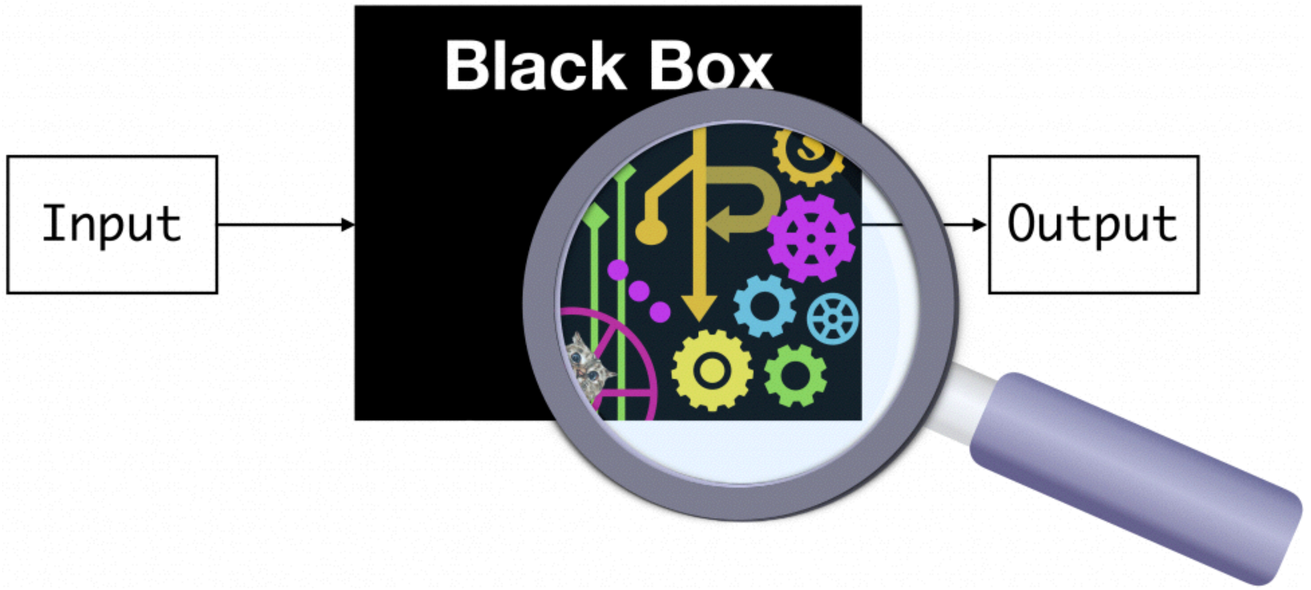
Eykholt *et al.* “Robust Physical-World Attacks on Deep Learning Visual Classification” (2018)

Adversarial attacks



“Brittle, greedy, opaque, and shallow”

“People can rationalize what’s going on in their thought processes. Deep learning can’t: These systems have no idea how they’re thinking or how they’re categorizing themselves.” —Rodney Brooks



https://www.flagshippioneering.com/stories/brittle-greedy-opaque-and-shallow-the-promise-and-limits-of-todays-artificial-intelligence

“Brittle, greedy, opaque, and shallow”

“True intelligence is being able to approach a new problem you haven’t had a lot of direct experience with. A human being can play a game that they’ve never played before and in a matter of minutes figure out something about what’s going on. Machines still can’t do that.”

—Gary Marcus



??

Deep learning can be notoriously brittle

https://www.flagshippioneering.com/stories/brittle-greedy-opaque-and-shallow-the-promise-and-limits-of-todays-artificial-intelligence

Next week: Intermediate ML

Topics covered:

- Transfer learning
- Representation learning
- Weakly supervised learning
- Self-supervised learning
- And more!

Thank you!