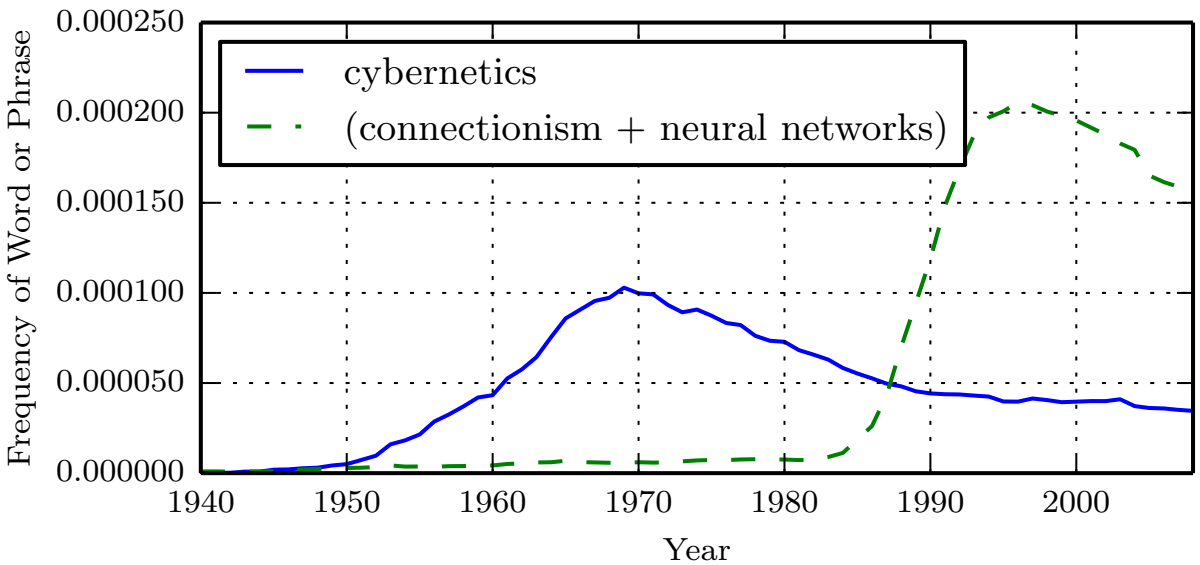
CSI 436/536
Introduction to Machine Learning

Deep Learning

Professor Siwei Lyu
Computer Science
University at Albany, State University of New York

the up and downs of NN

- first high wave 1960s: simple one layer perceptron
- first down wave 1970s: show of limitations of one layer perception
- second high wave 1980s: development of BP and many uses (and abuses)
- second down wave late 1990s to 2006: overfitting problem and vanishing gradient



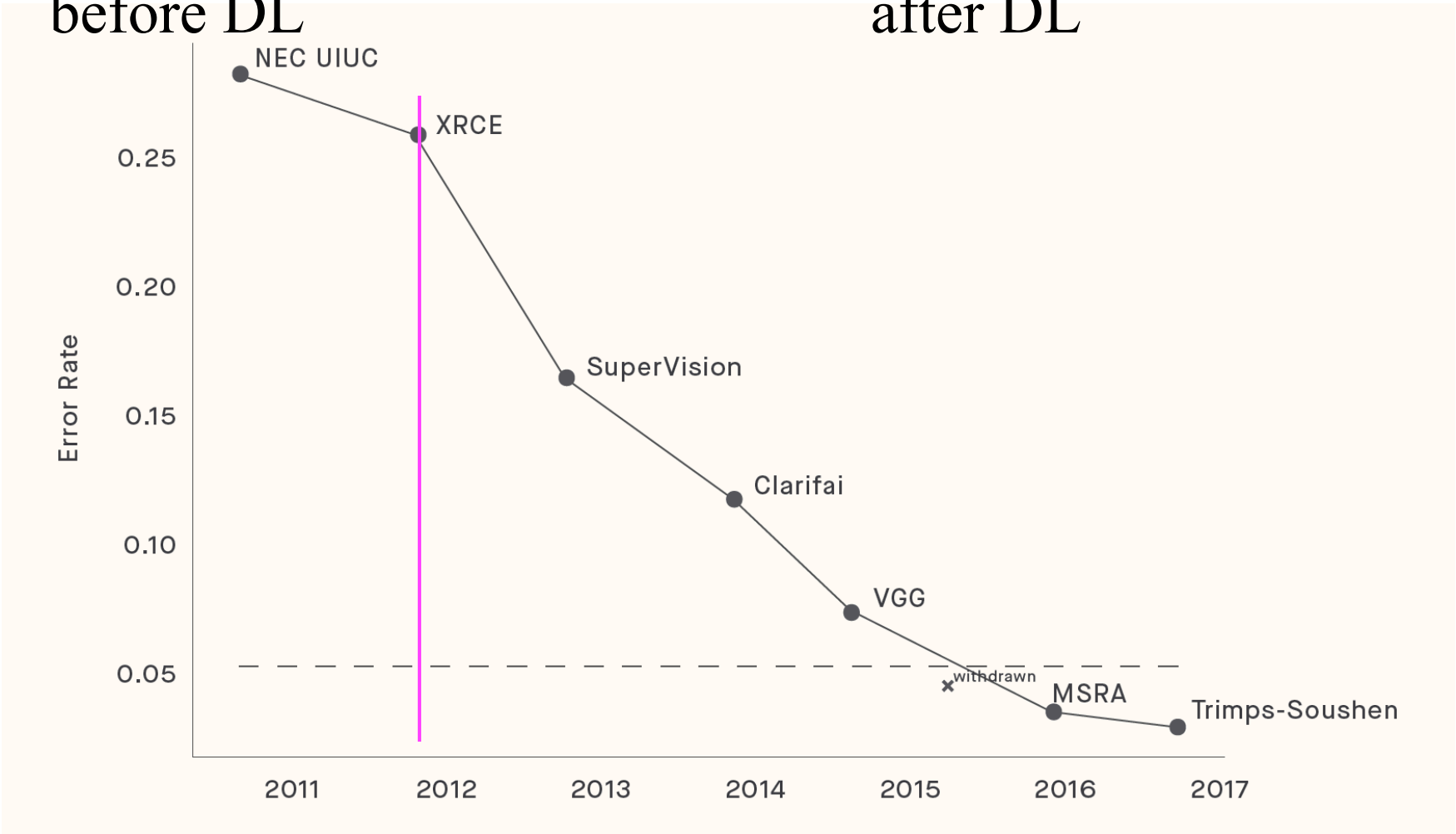
the up and downs of NN

- first high wave 1960s: simple one layer perceptron
- first NN winter 1970s: show of limitations of one layer perception
- second high wave 1980s: development of BP and many uses (and abuses)
- second NN winter late 1990s to 2006: overfitting problem and vanishing gradient
- third high wave 2006 — now: deep learning (learning with deep neural networks)
- third NN winter?

object recognition on ILSVC

before DL

after DL

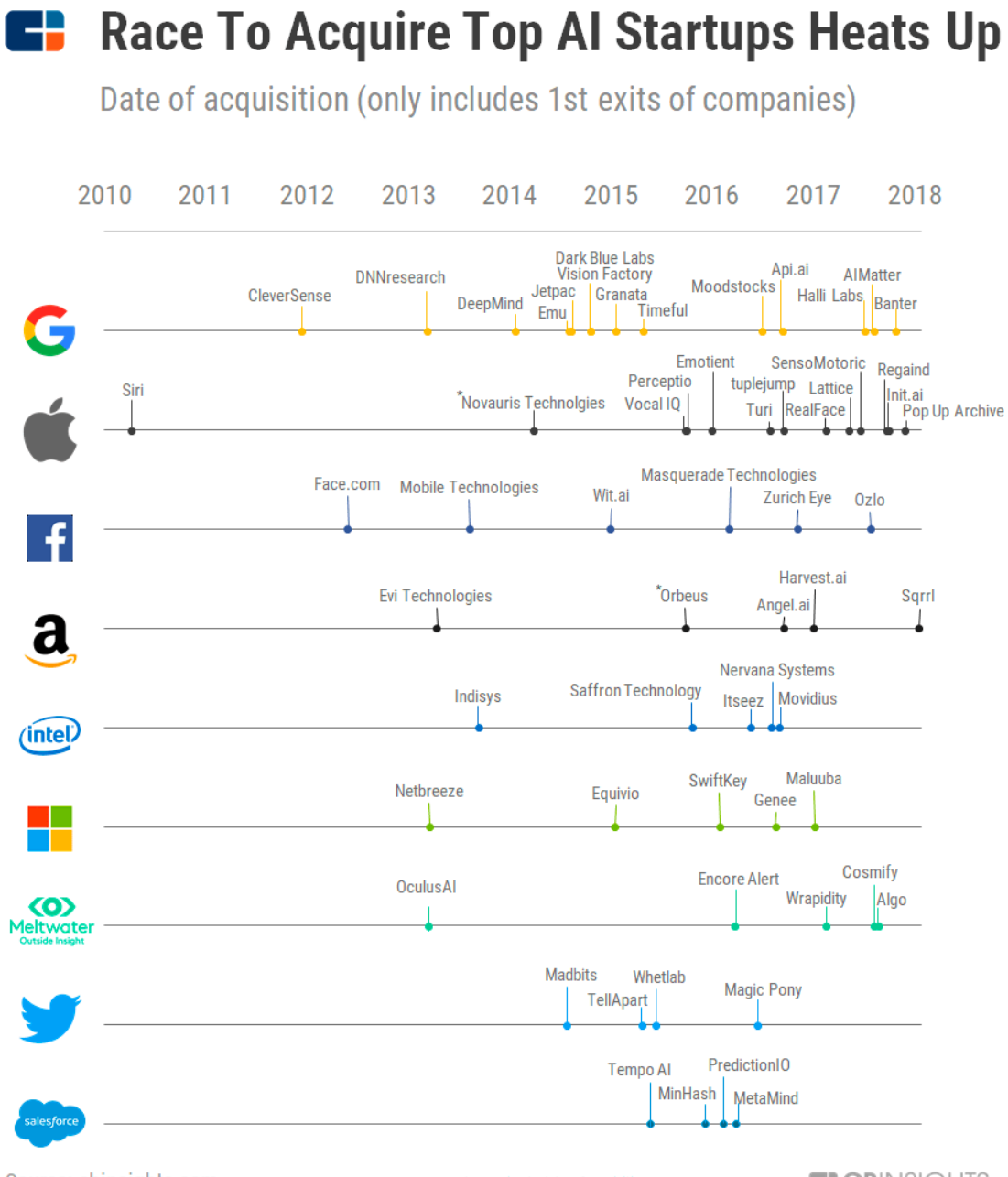


2012: convolutional NN (Alex Net)

2014: ensemble MLP

2015: res-net

Deep learning players



many successes of DL

2014

2016

2017

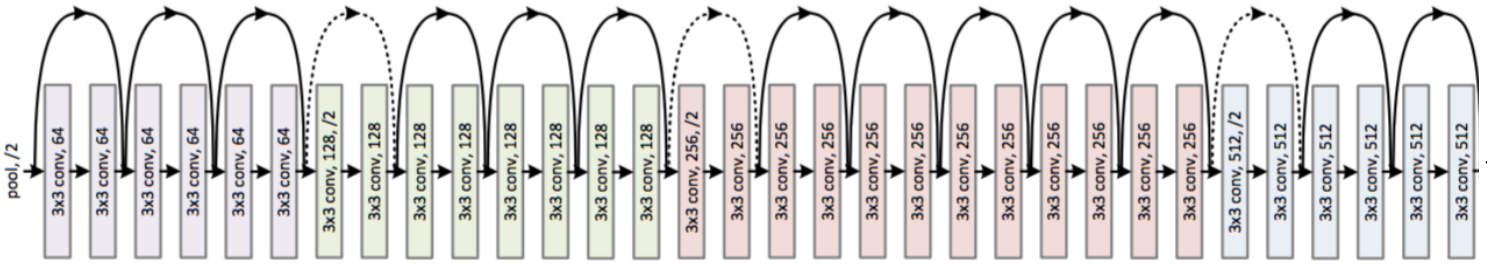
2016

Turing award 2019

- Joshua Bengio, Geoff Hinton & Yann LeCun for their contributions in deep learning

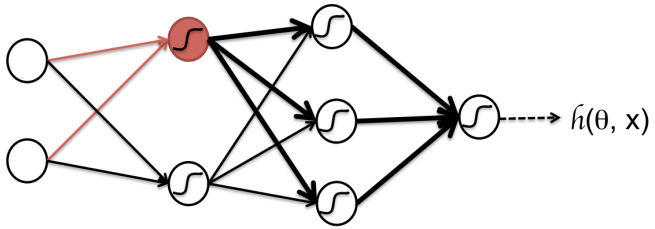
deep learning

- DL refers to an ensemble of learning methods, including deep neural networks and other hierarchical models
  - it has many layers of basic computing units
    - Known as artificial neurons
  - how deep is deep?
    - recent work by Microsoft Research uses 150 layer neural network for image recognition
    - the handwritten digit recognition based on NN in the 1980s had only 3 layers

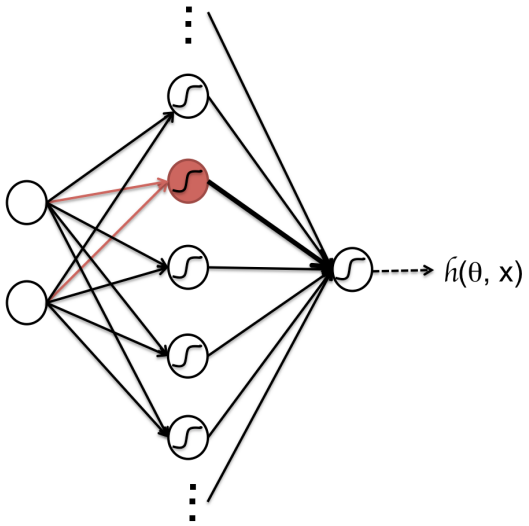


deep vs. shallow

- example of deep learning models:
  - deep NN, stack RBM, stack AE, RNN, LSTM RNN, etc…
- examples of shallow learning models:
  - logistic regression, SVM, RBF network, MRF, etc…



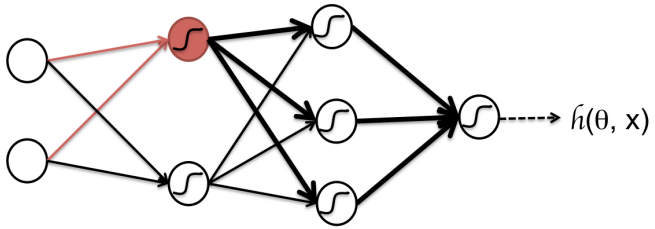
Deep Network



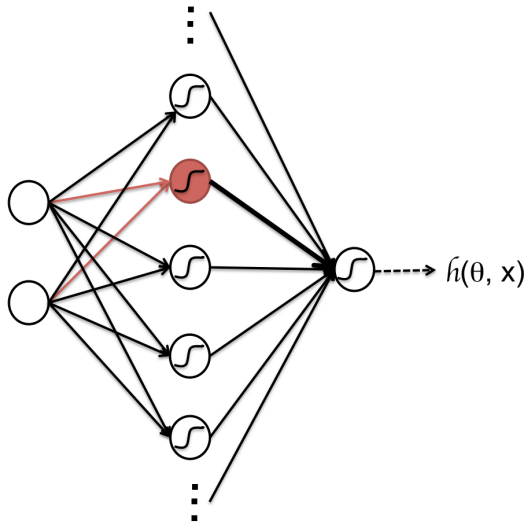
Shallow Network

why deep learning

- they are more efficient than “shallow networks”
  - get the same level of performance a shallow network will need many more parameters
- intuition: deep network has many reusable components
  - example: red neuron below used by many more other neurons in deep network



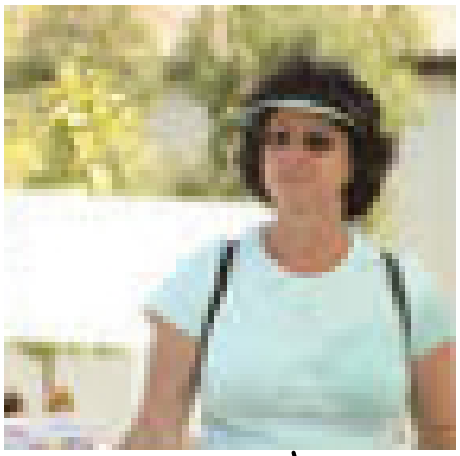
Deep Network

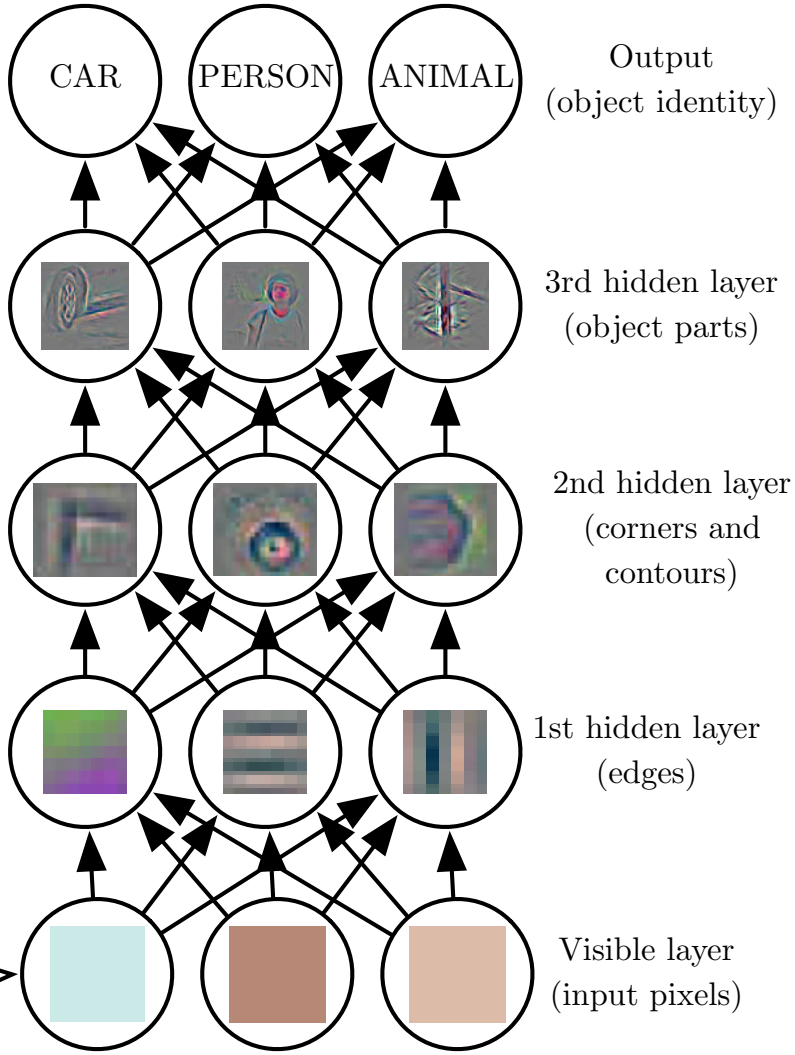


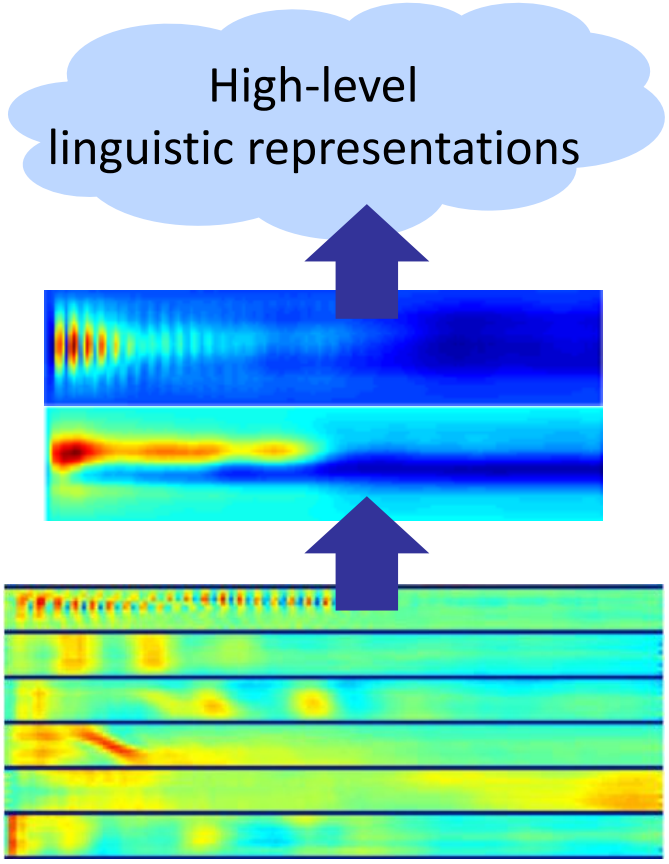
Shallow Network

Why deep learning

- Deep learning finds better data representations automatically





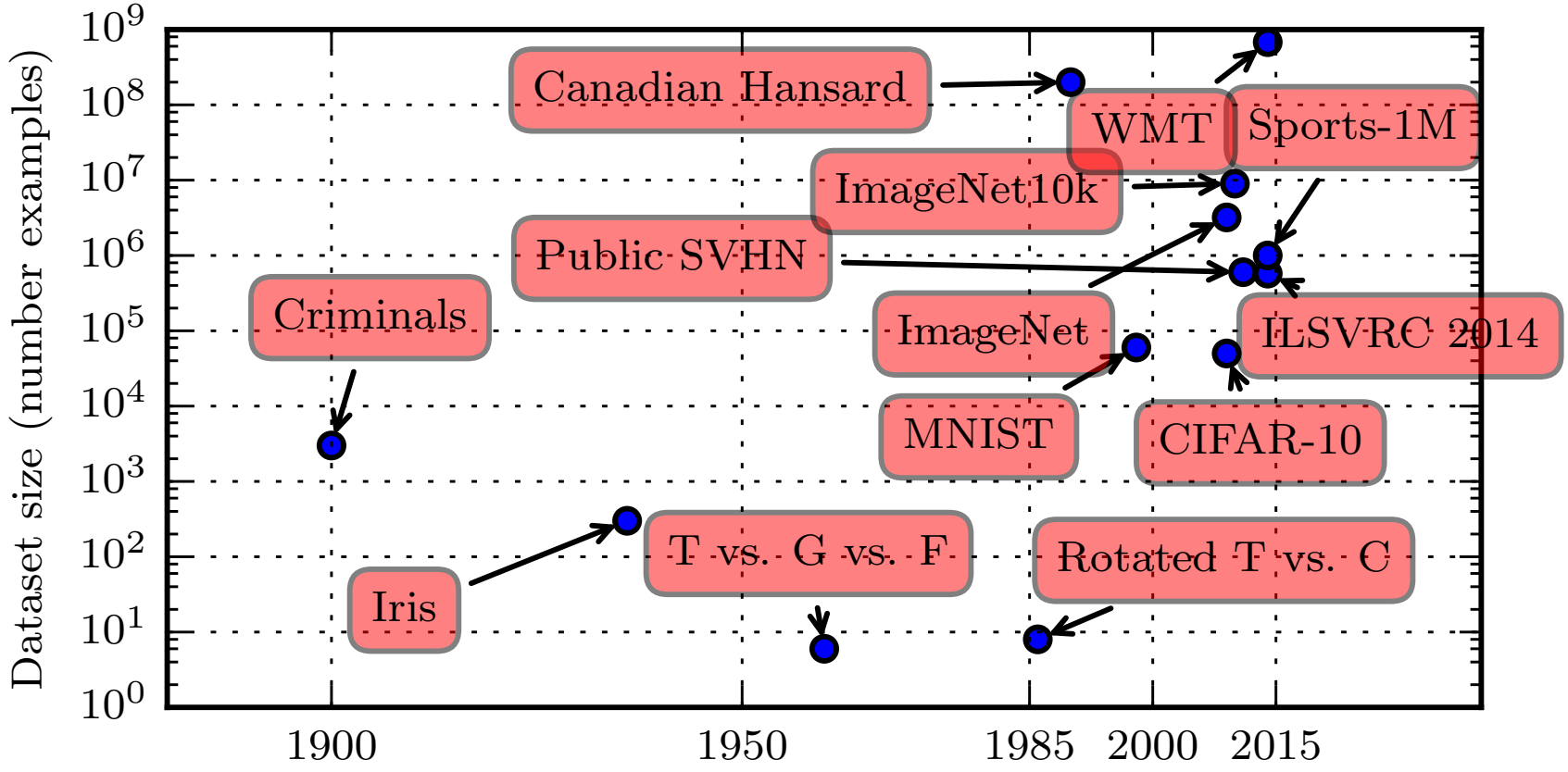


deep learning: why now?

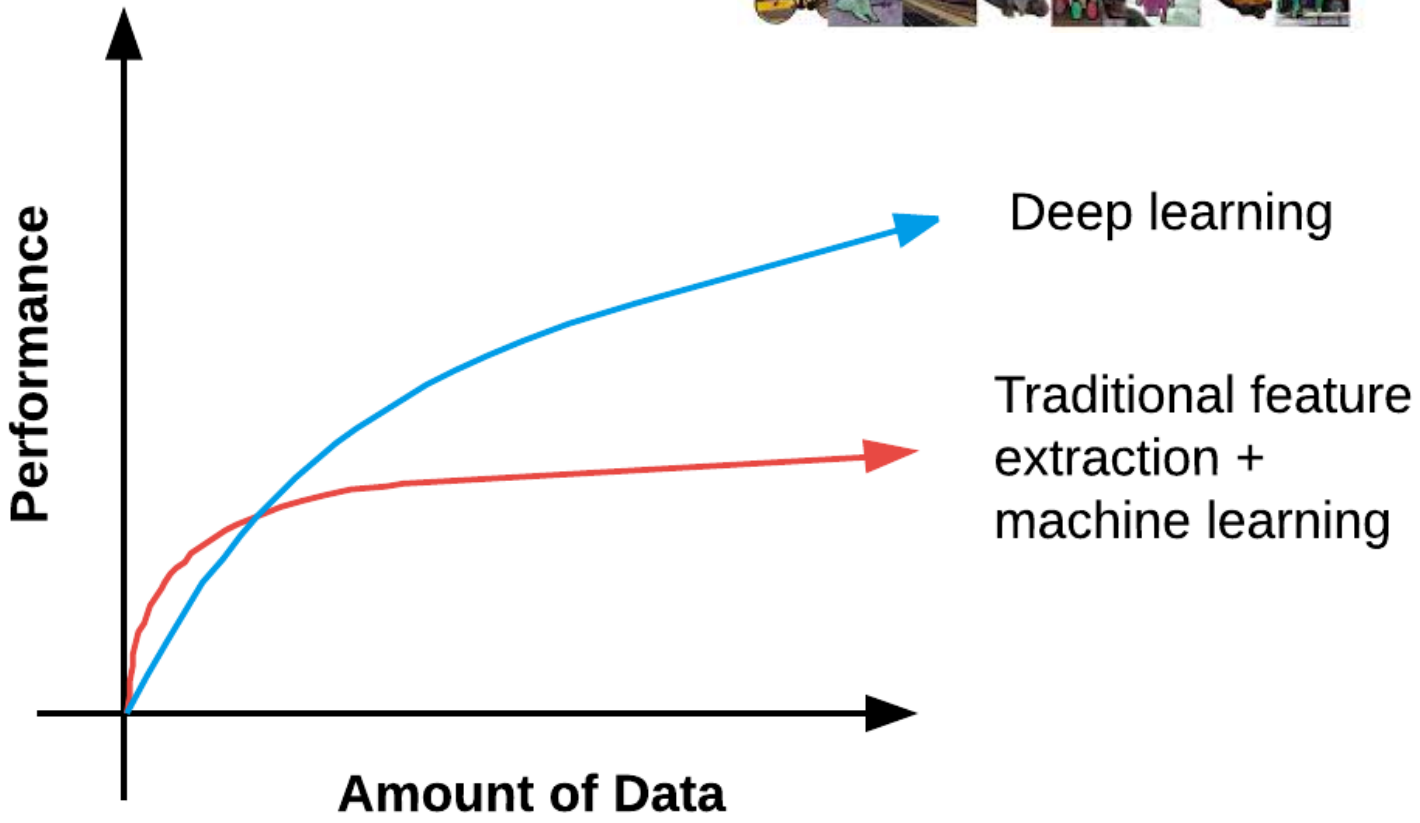
- Powerful hardware
  - GPU-based parallel computing
  - Cloud data/computing service
- Availability of large datasets
  - ImageNet, MS COCO, etc
- Powerful software frameworks
  - TF, PyTorch, CAFFE, Theano
- Improved algorithms:
  - resNet, highway network, U-net for better BP
  - dropout, ReLU activation, regularization to reduce overfitting

Deep learning & big data

- We get bigger and bigger datasets



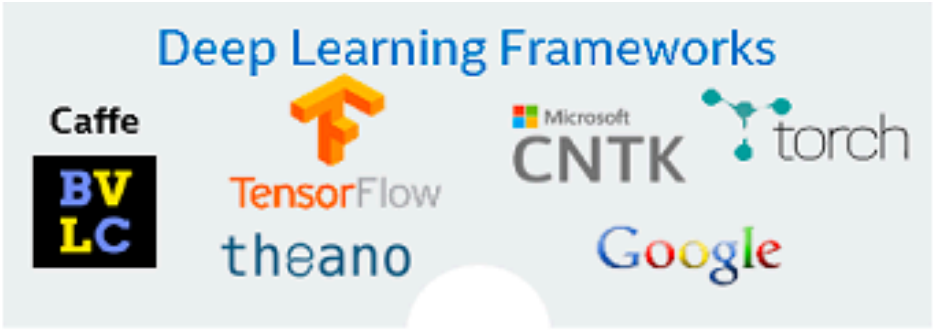
Deep learning & big data



Who's afraid of overfitting?

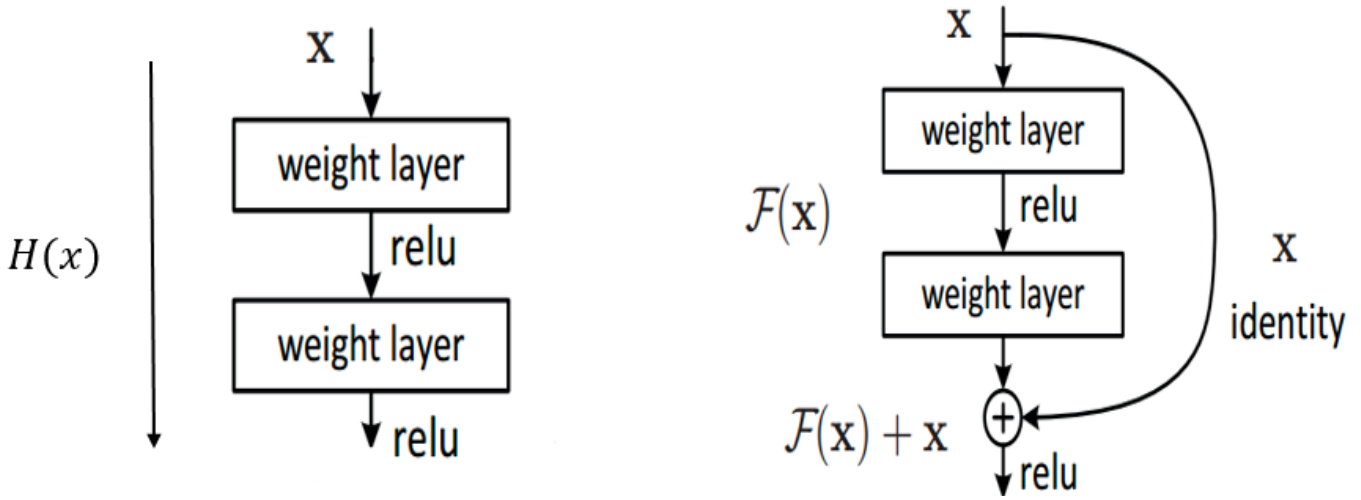
Deep learning frameworks

- CAFFE/CAFFE2: written in C++
  - good for CNN for GPU
  - http://caffe.cs.berkeley.edu
- TensorFlow: python
  - http://openAI.google.com/tensorflow
  - supported by Google & OpenAI, C-based
  - Google TPU (tensor processing unit)
- Torch: written in LUA
  - http://torch.org
- Theano (Python CPU/GPU)
  - http://deeplearning.net/software/theano
  - can do automatic, symbolic differentiation

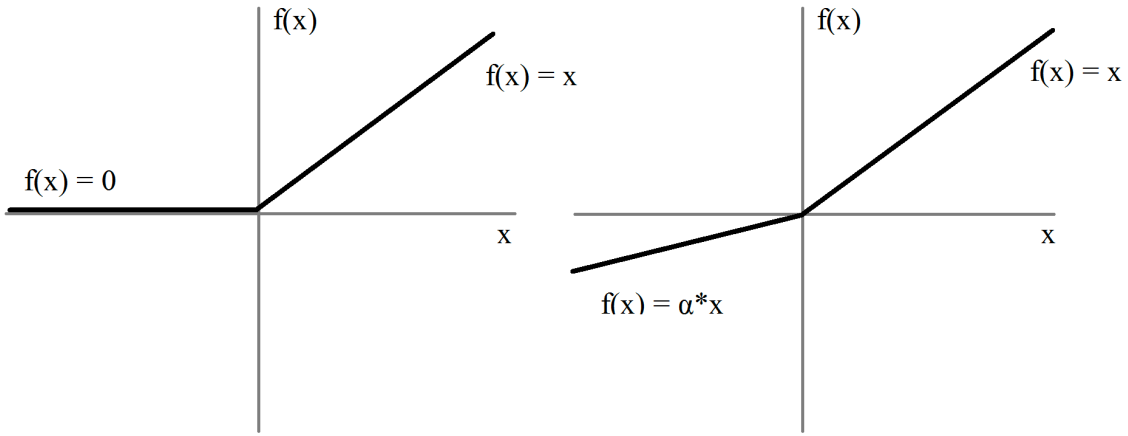


deep learning: why now

- Better network structure
  - Residual network (ResNet)



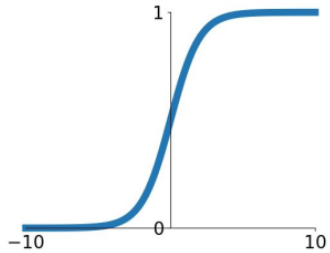
  - Better activation function



Activation Function

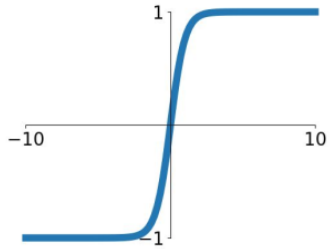
Sigmoid

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



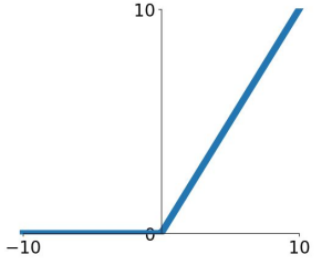
tanh

$$\tanh(x)$$



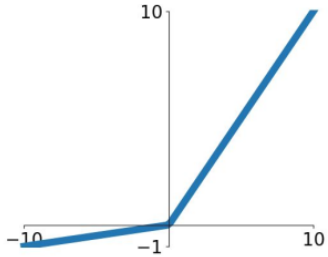
ReLU

$$\max(0, x)$$



Leaky ReLU

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

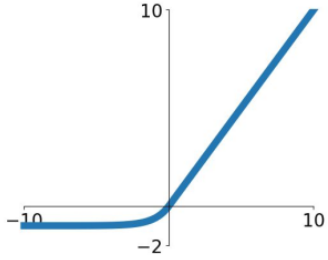


Maxout

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

ELU

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

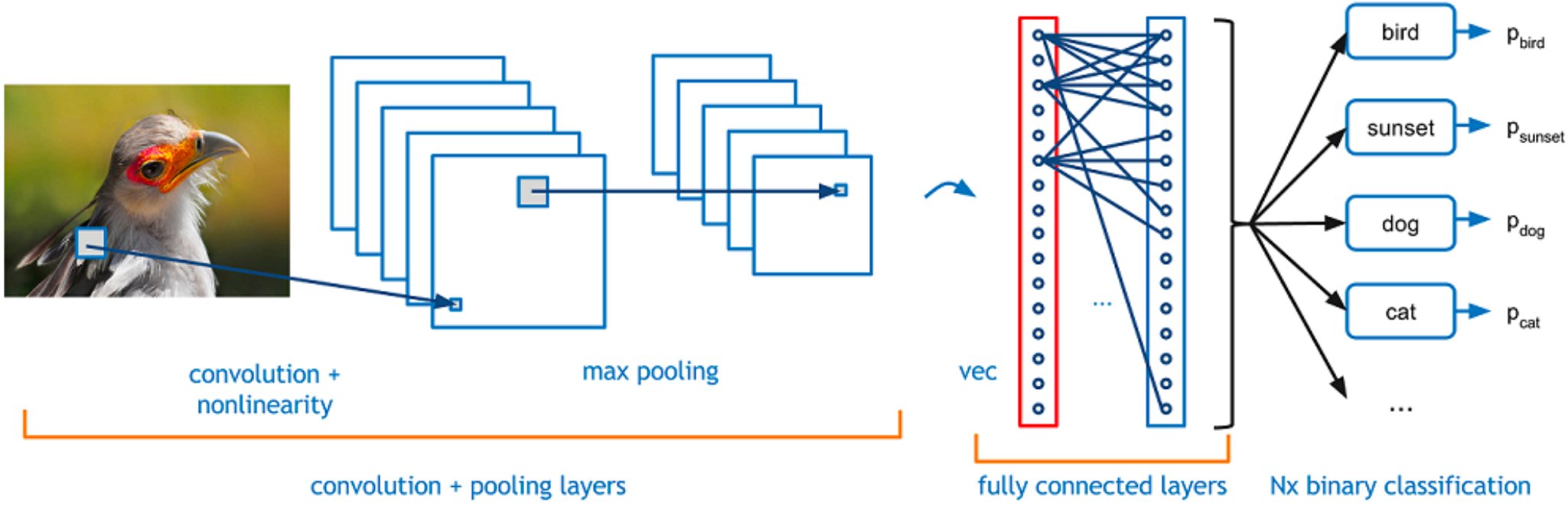


Major DNN models

- Supervised learning:
  - Convolutional neural networks (CNN)
- Sequence learning
  - Recurrent neural networks (RNN)
- Unsupervised learning
  - Generative adversary networks (GAN)
- Reinforcement learning
  - Deep reinforcement learning (DRL)

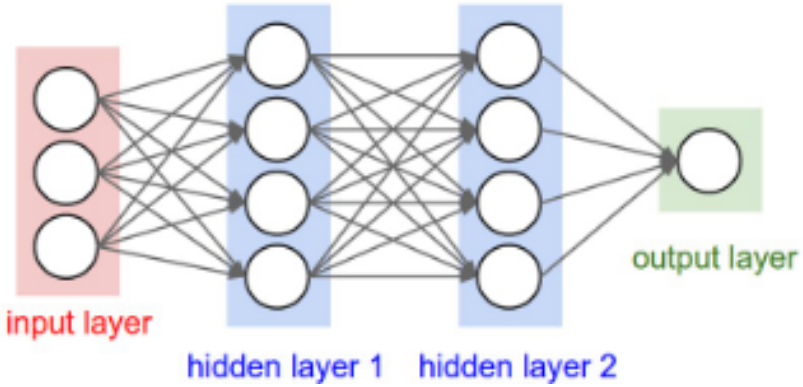
CNN

- Convolutional neural networks
  - Popular NN architecture pioneered by Y. LeCun
  - Using weight sharing (convolution) to reduce the number of parameters
  - #1 choice for computer vision applications

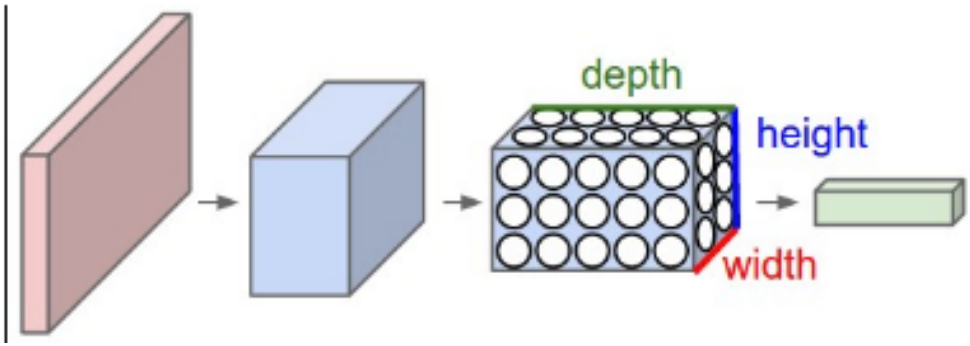


Convolutional Neural Networks

Neural Network



Convolutional Neural Network



LeNet

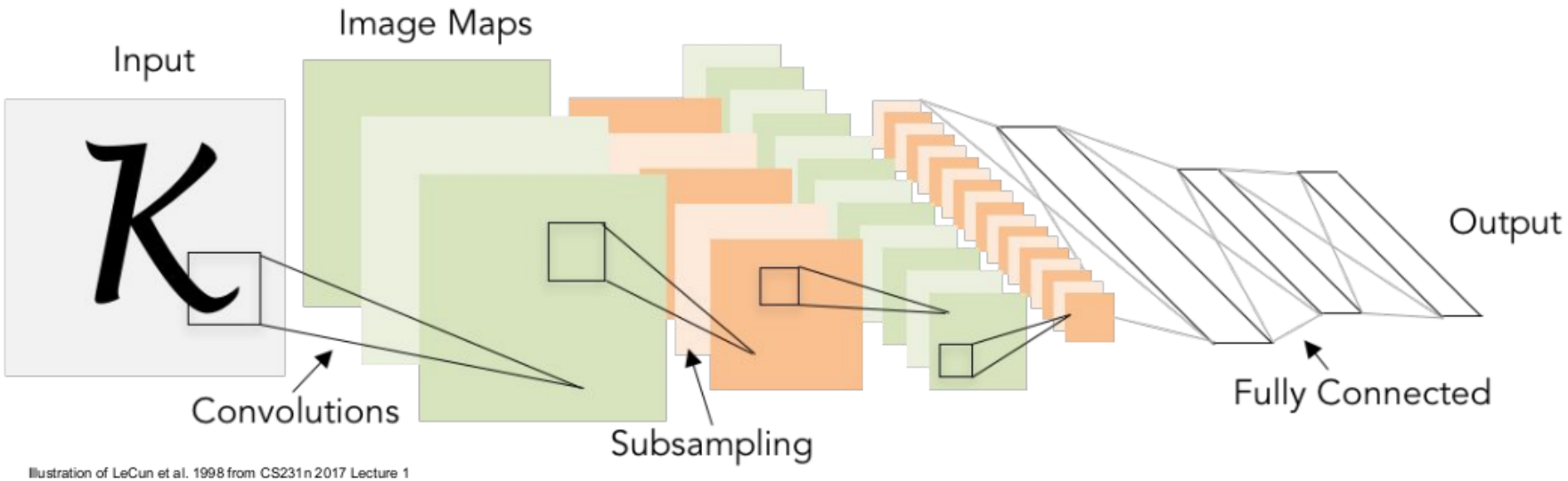
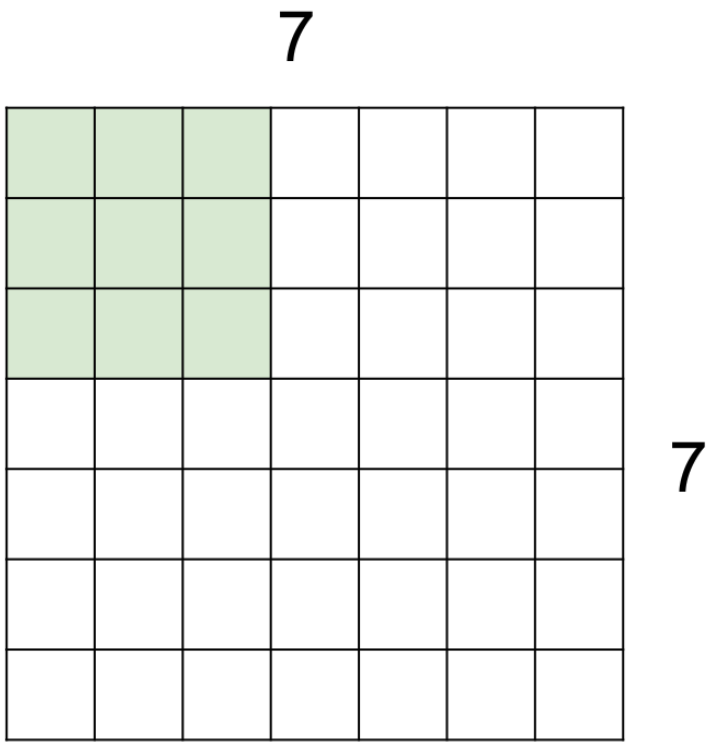


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Convolution Details

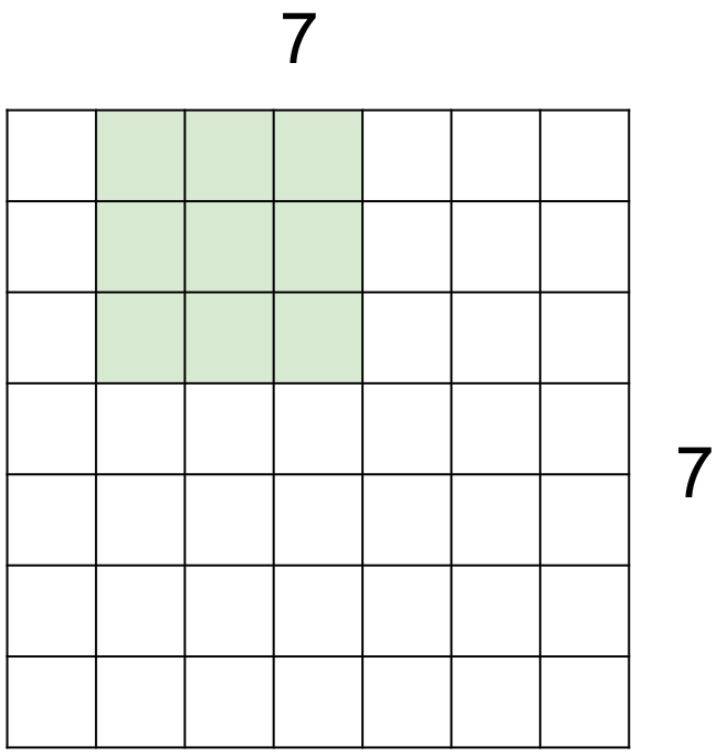
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Details

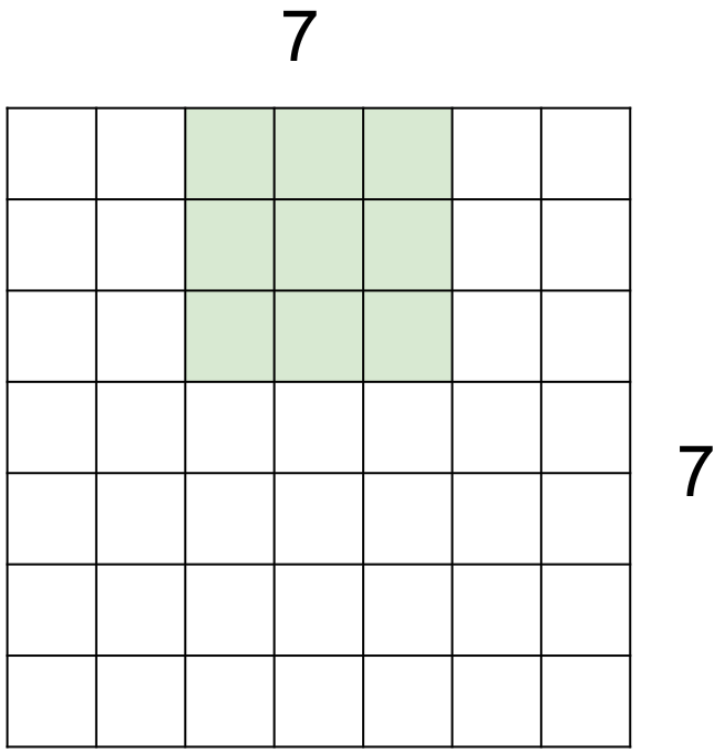
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Details

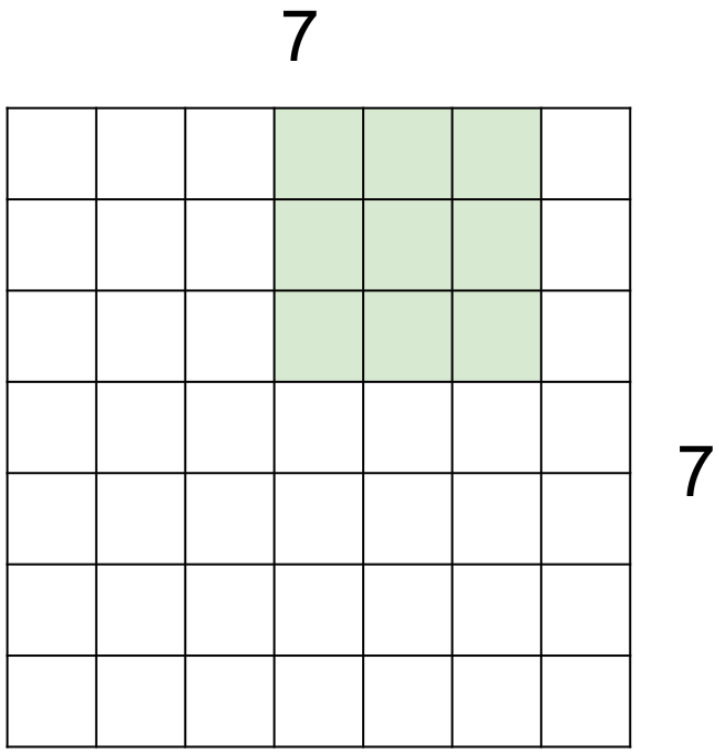
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Details

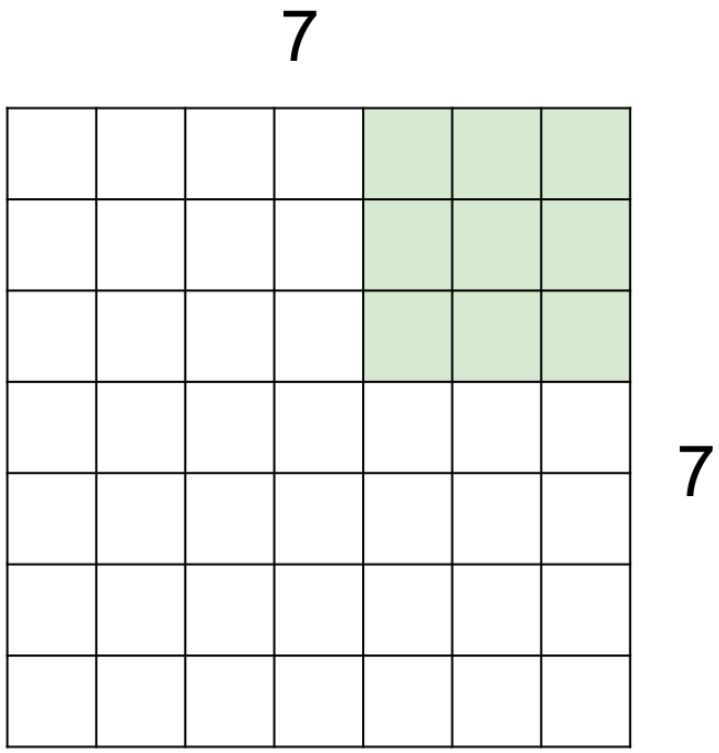
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolution Details

A closer look at spatial dimensions:

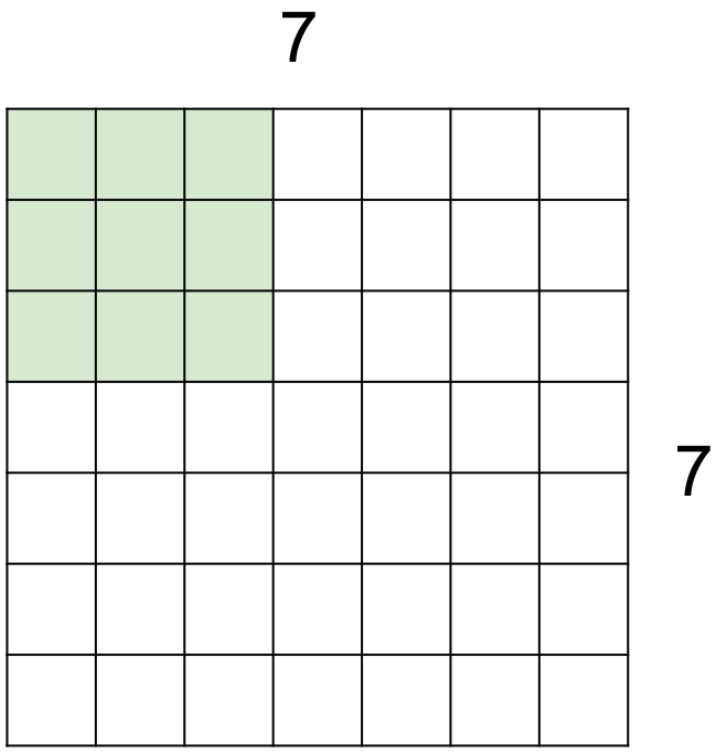


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Convolution Details

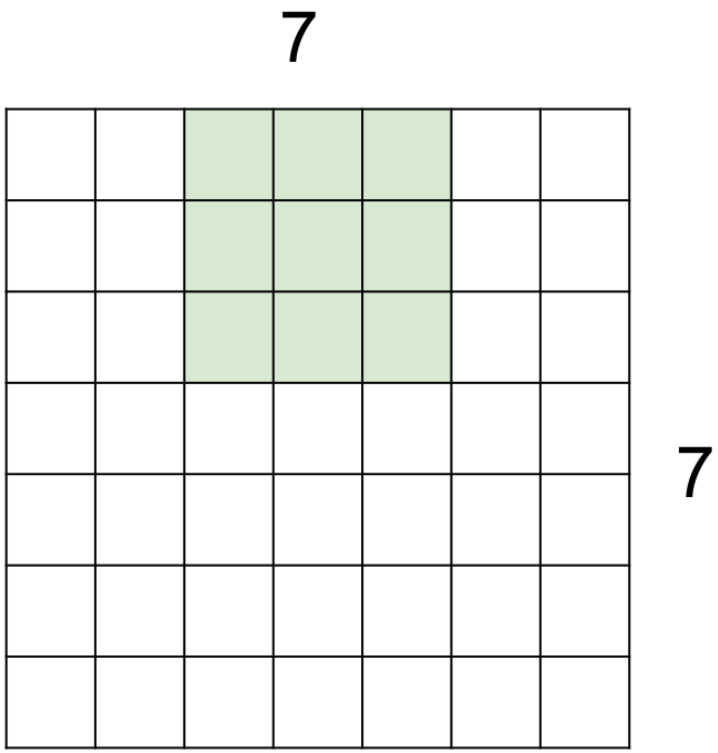
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution Details

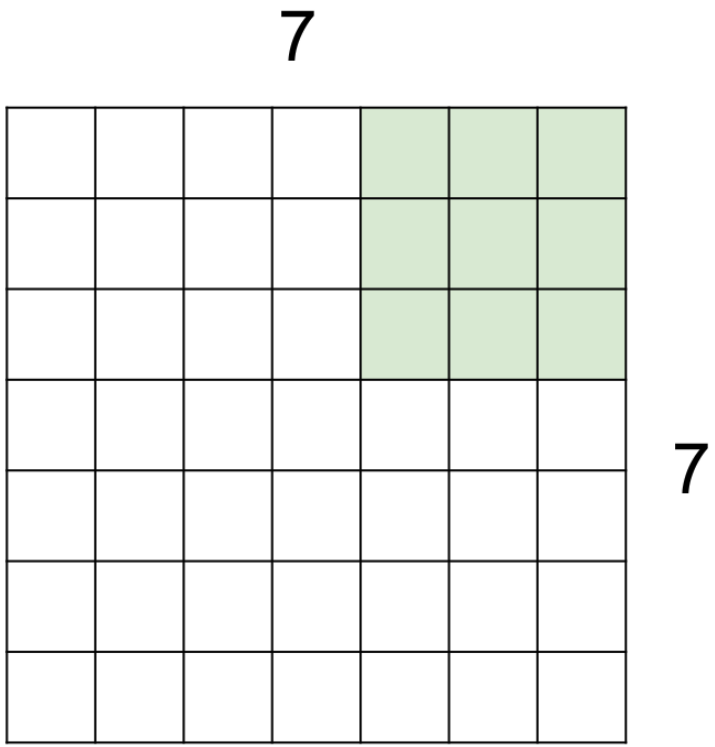
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution Details

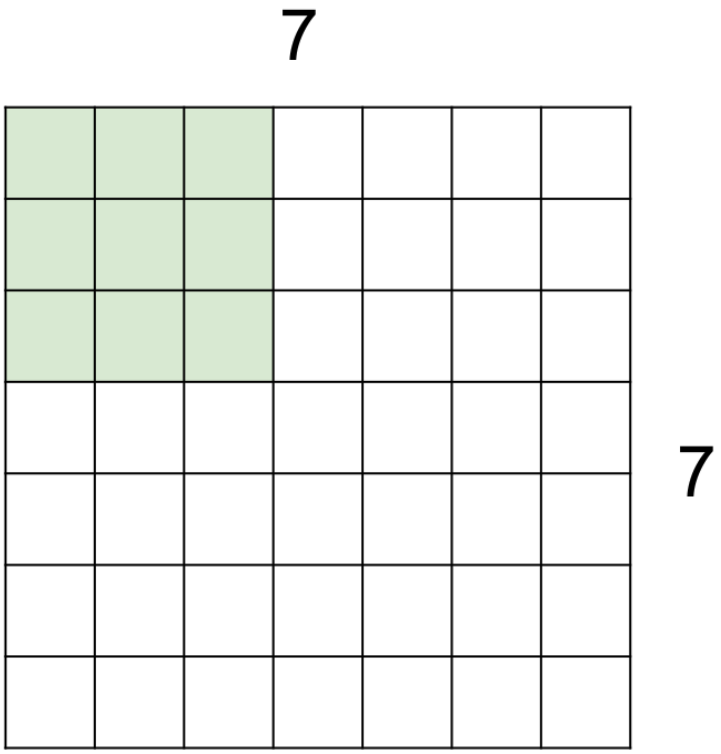
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Convolution Details

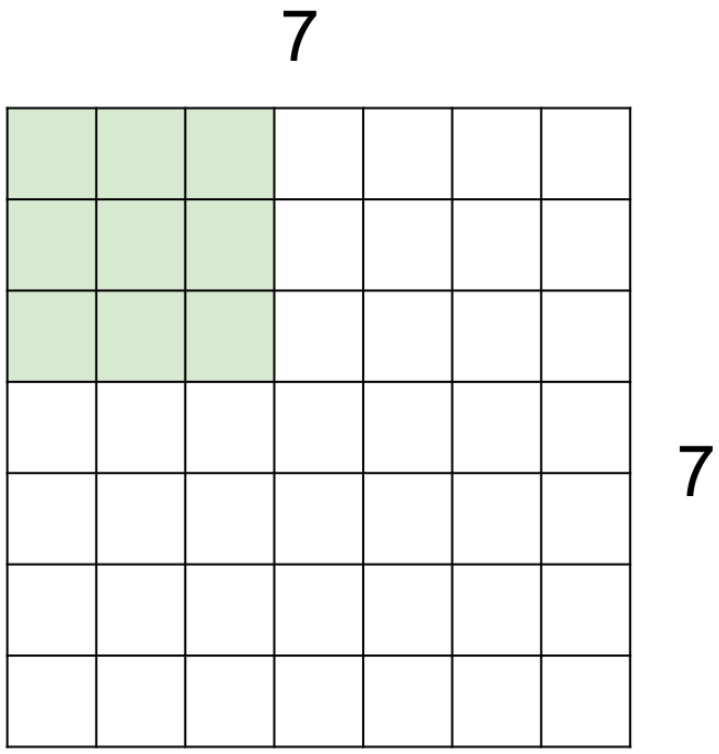
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Convolution Details

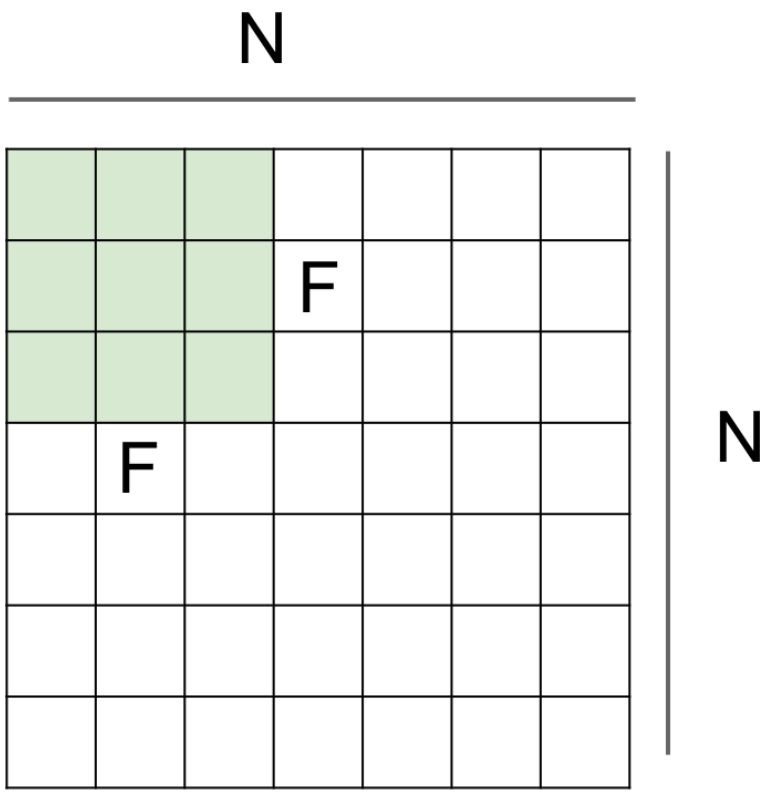
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolution Details

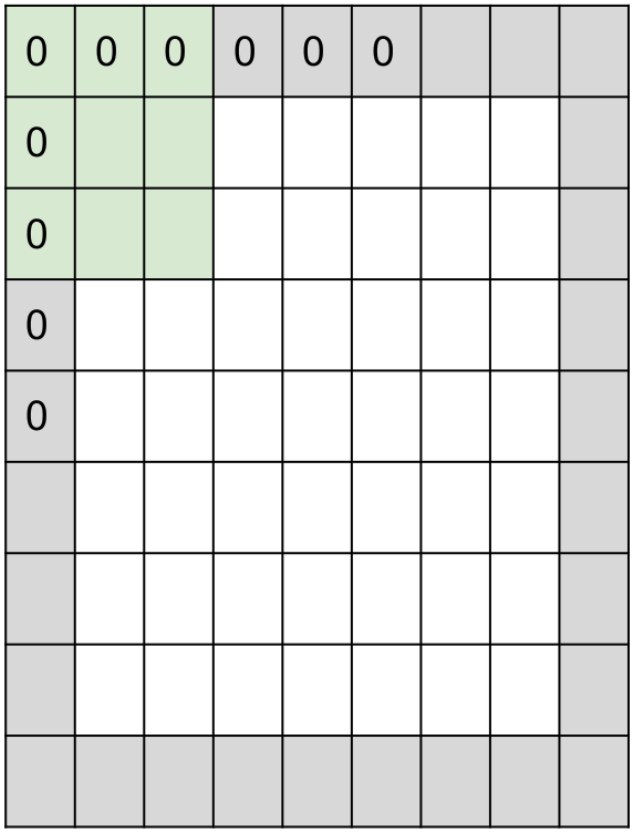


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Padding in Convolution

In practice: Common to zero pad the border

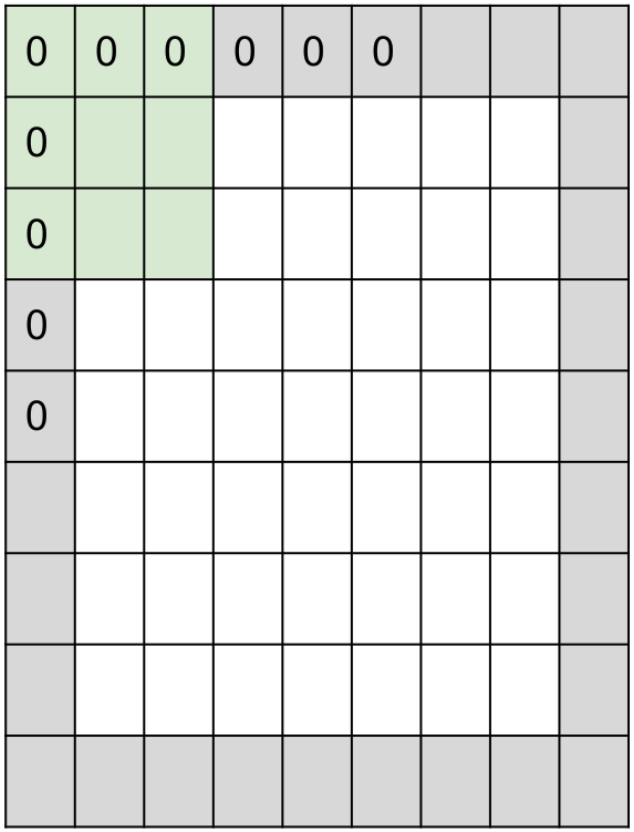


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Padding in Convolution

In practice: Common to zero pad the border

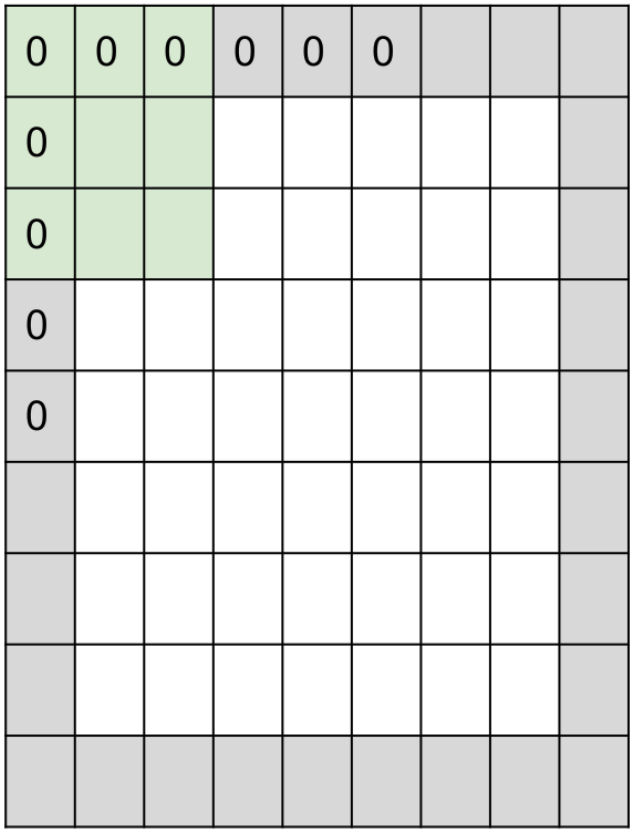


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Padding in Convolution

In practice: Common to zero pad the border

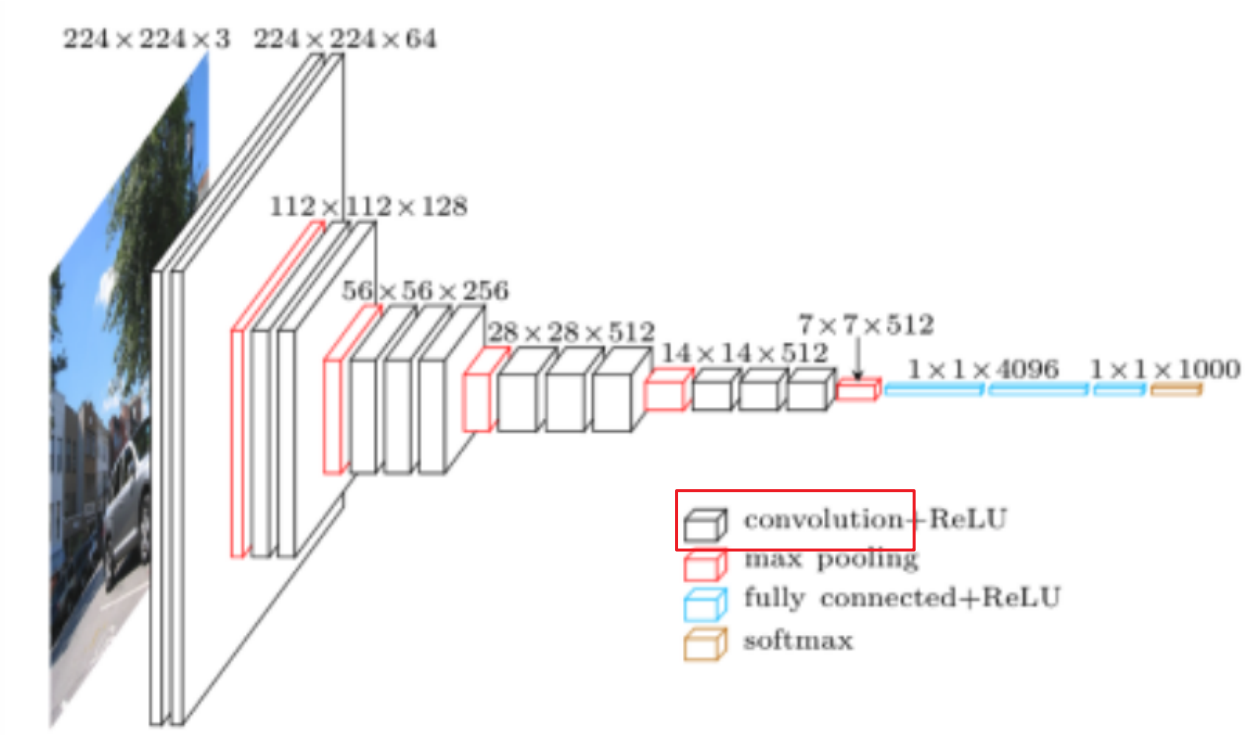


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

CNN for classification

VGG 16 for image classification



Dog: 3%

Cat: 1%

Bus: 15%

⋮

Tree: 2%

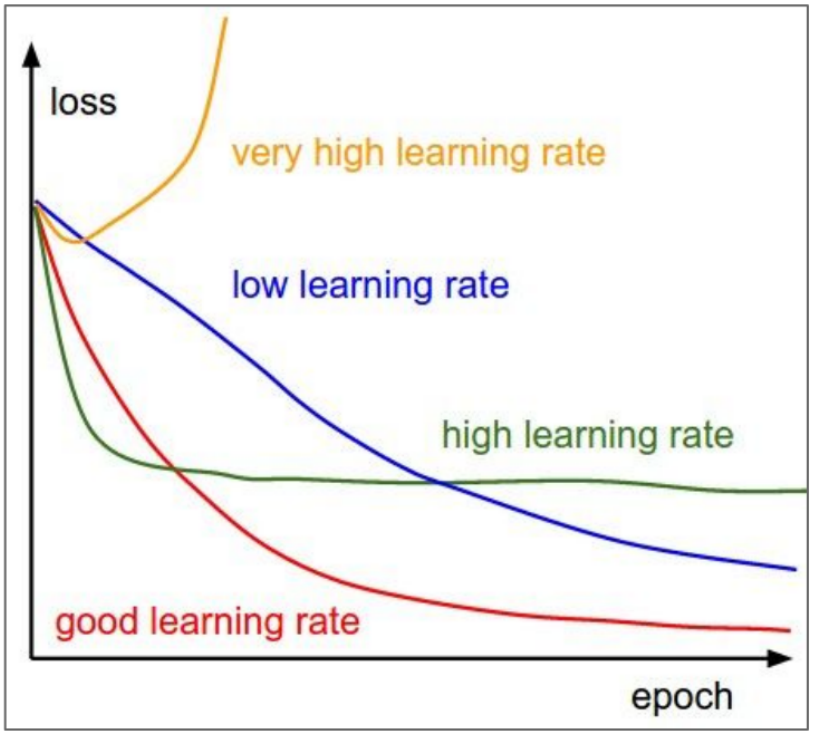
Input image

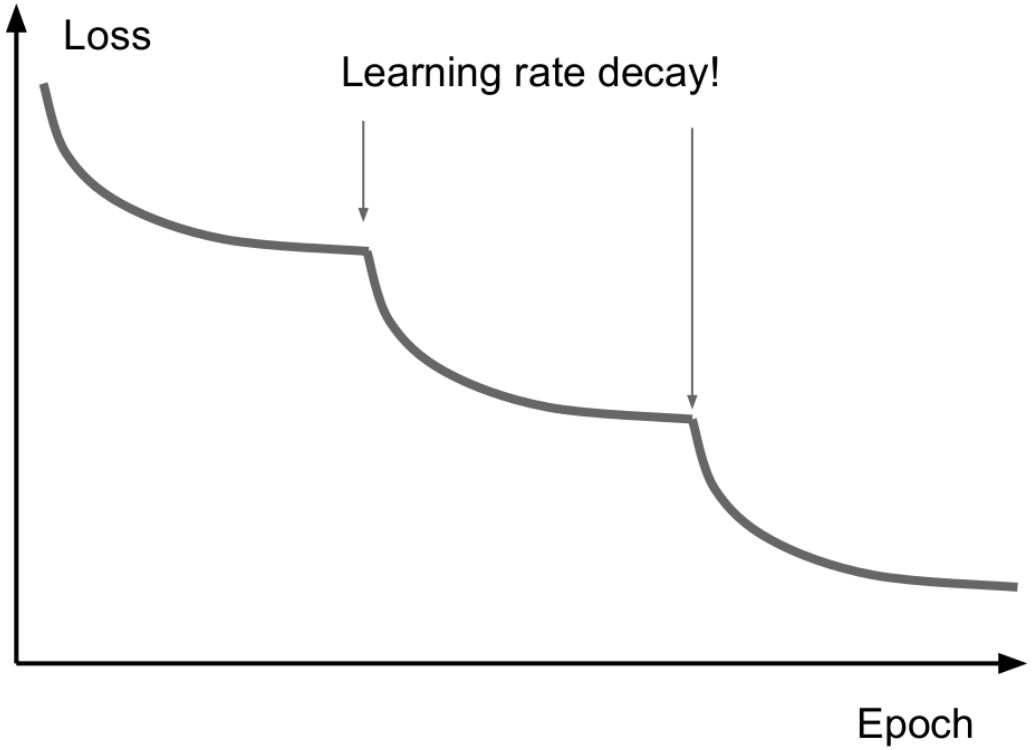
Model

prediction

Optimizer

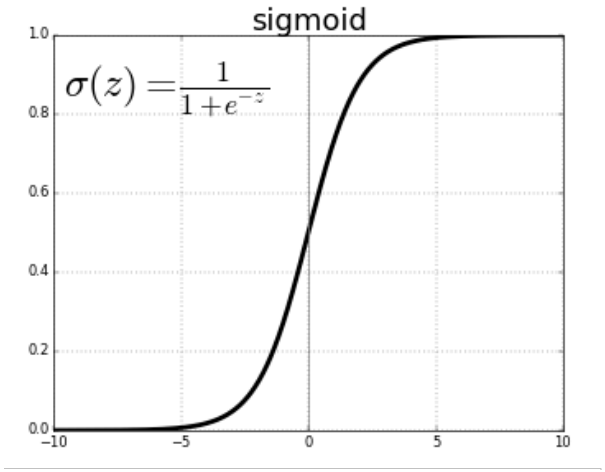
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.

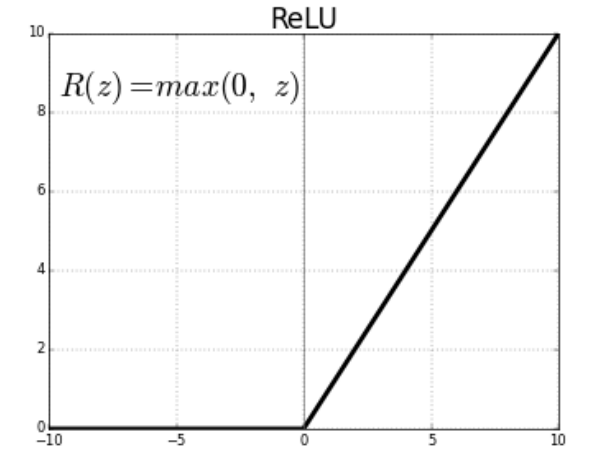


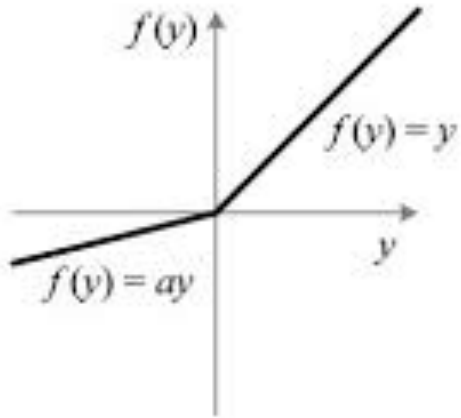


Combating vanishing gradient

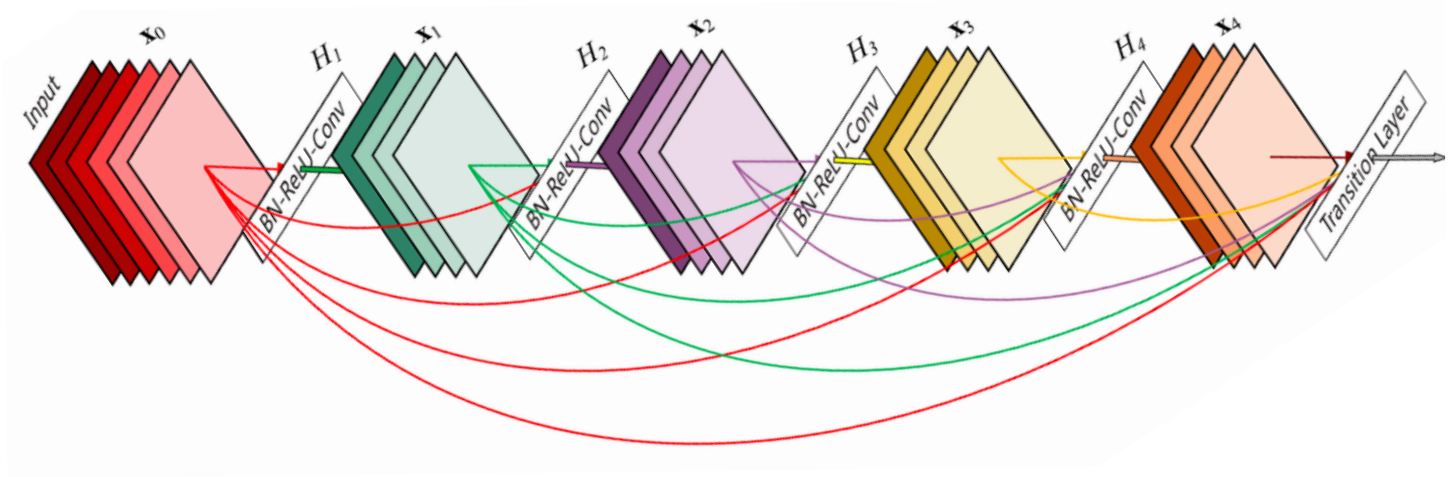
- Non-saturating activation function (sparsity)







- Highway connection reduce the exact depth



regularization

- early stopping: use parameters with best validation error
  - segment a part of training data as validation set (not used in training)
  - during training, monitor error on the validation set
    - keep the lowest validation error
  - stop training if validation error not improve after a fixed number of iterations
- standard L1 or L2 regularization on weights
- sparsity constraints on hidden activations
- weight sharing to reduce number of free parameters

parameter initialization

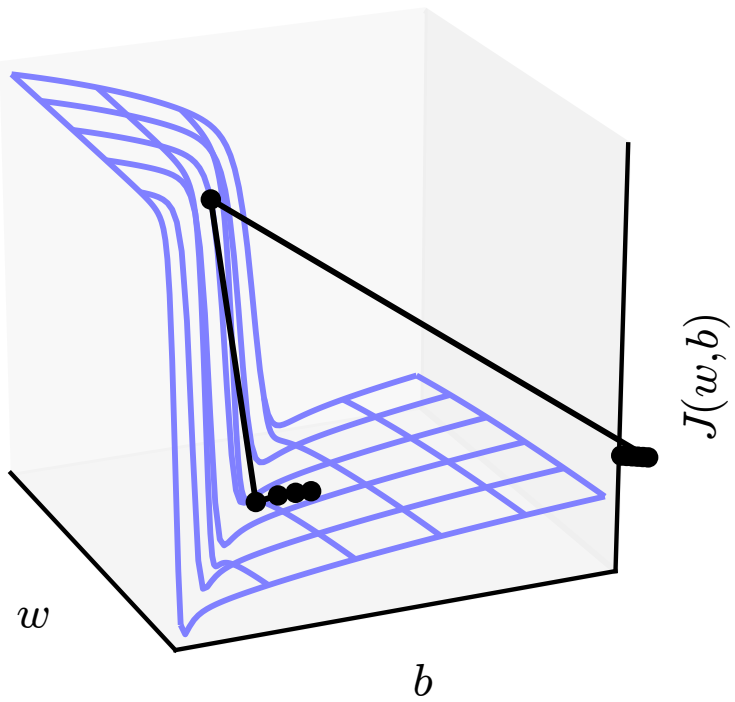
- initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target)
  - initialize weights ~ Uniform(−r, r),
    $$r = (6/(\text{fan-in} + \text{fan-out}))^{1/2}$$
    for tanh units, and
    $$r = 4*(6/(\text{fan-in} + \text{fan-out}))^{1/2}$$
    for sigmoid units [Xavier initialization]
- pre-training to set initial weight values
  - using auto-encoder
  - using RBM
- improve network stability

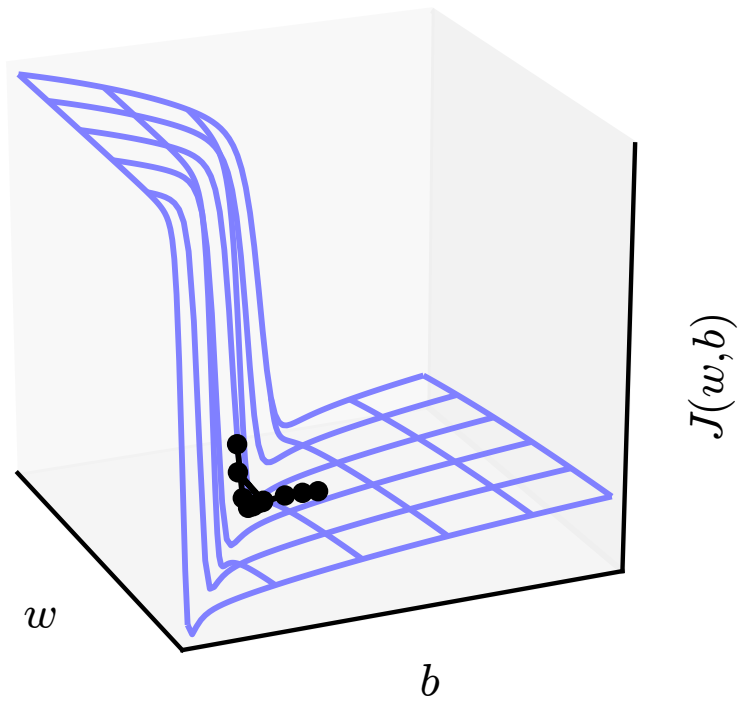
setting learning rate

- simple recipe: keep it fixed and use the same for all parameters
- better results obtained by allowing learning rates to decrease, typically in O(1/t) because of theoretical convergence guarantees
- no learning rate setting if use L-BFGS or AdaGrad
  - AdaGrad: scale gradient by the square root of sum of past gradient magnitudes

adaptive learning rate

- changing learning rates using previous gradient values
  - AdaGrad, RMSprop, ADAM





stochastic gradient update

- (sub) gradient method uses total gradient over all examples per update, stochastic gradient updates after only 1 or few examples

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta}$$

- L = loss function, z_t = current example, θ = parameter vector, and ε_t = learning rate
- ordinary gradient method as a batch method is very slow, should never be used
  - On smaller datasets use 2nd order batch method such as L-BFGS or conjugate gradients
  - On large datasets, SGD usually wins over all batch methods

training schemes

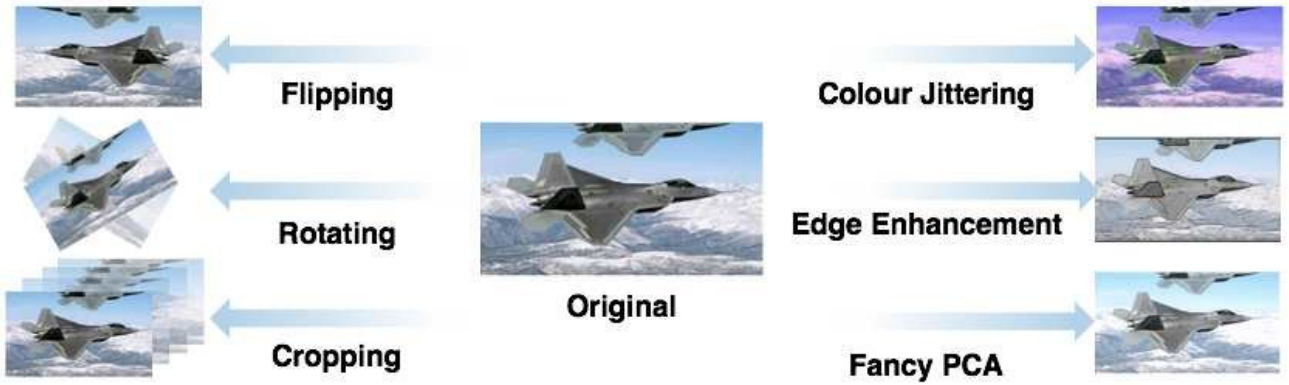
- mini-batches in SGD
  - balance between efficiency & variance
  - bigger batch -> small variance
  - smaller batch -> faster algorithm
- typical SGD convergence graph (vibrating)

dropout training

- training time: at each instance of evaluation (in online SGD- training), randomly set 50% of the inputs to each neuron to 0
- test time: halve the model weights (now twice as many)
- prevents feature co-adaptation:
  - a feature cannot only be useful in the presence of particular other features
- as a form of model averaging as a strong regularizer

Data Augmentation

Input data augmentation, normalization



Followed by normalize image by dividing each pixel by 255.

RNN

- Recurrent neural networks
  - NN with feedback links
  - Unrolling in time leads to a network with (conceptually) infinite number of layers
  - Applications: sequential data analysis
    text, speech, DNA, video, etc

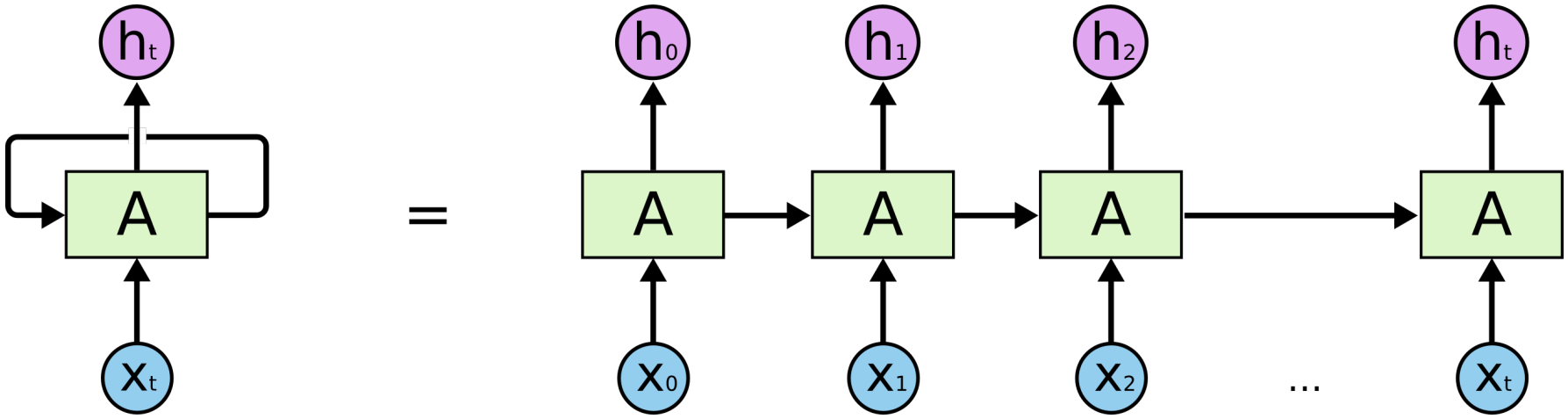


Image from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN

- Model structure
  - Sequential link of RNN cells

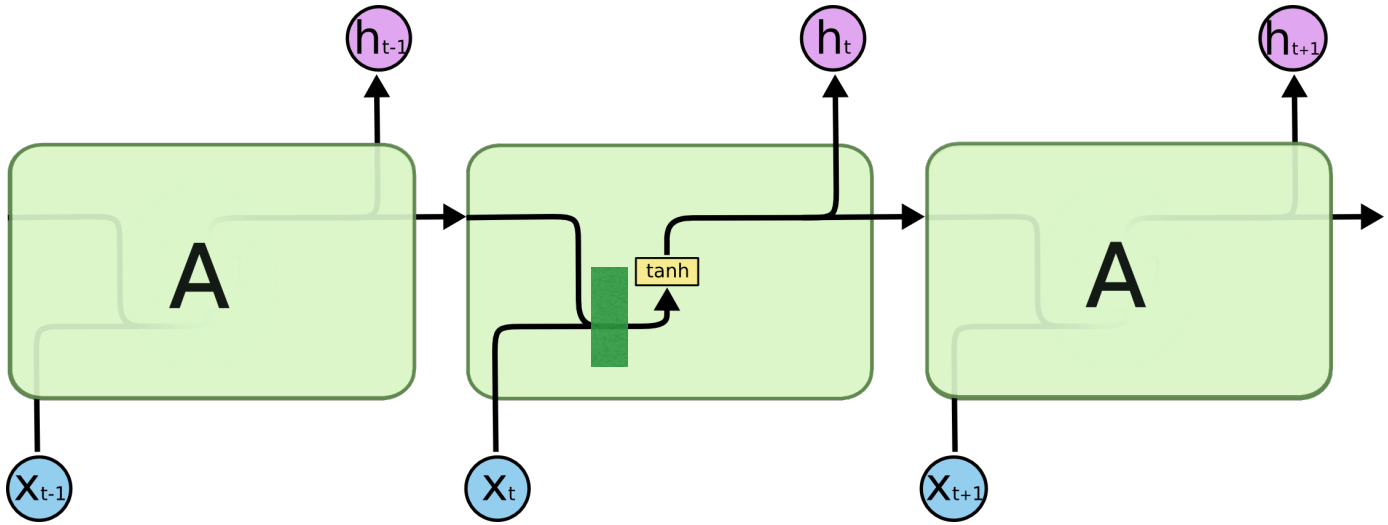
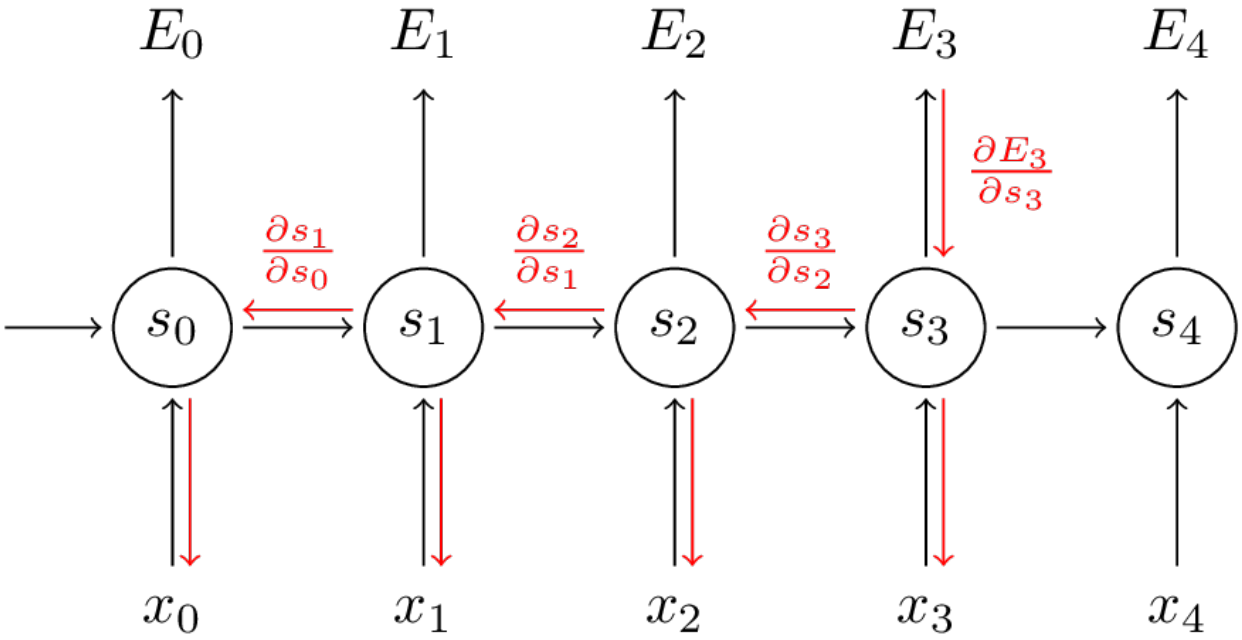


Image from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Input from previous time step carries "historic" information from all previous input
  - h is known as the "latent" or "hidden" state
  - From hidden state output can be generated

Training RNN

- BPTT (back propagation through time) with fixed window
  - All weights are tied together



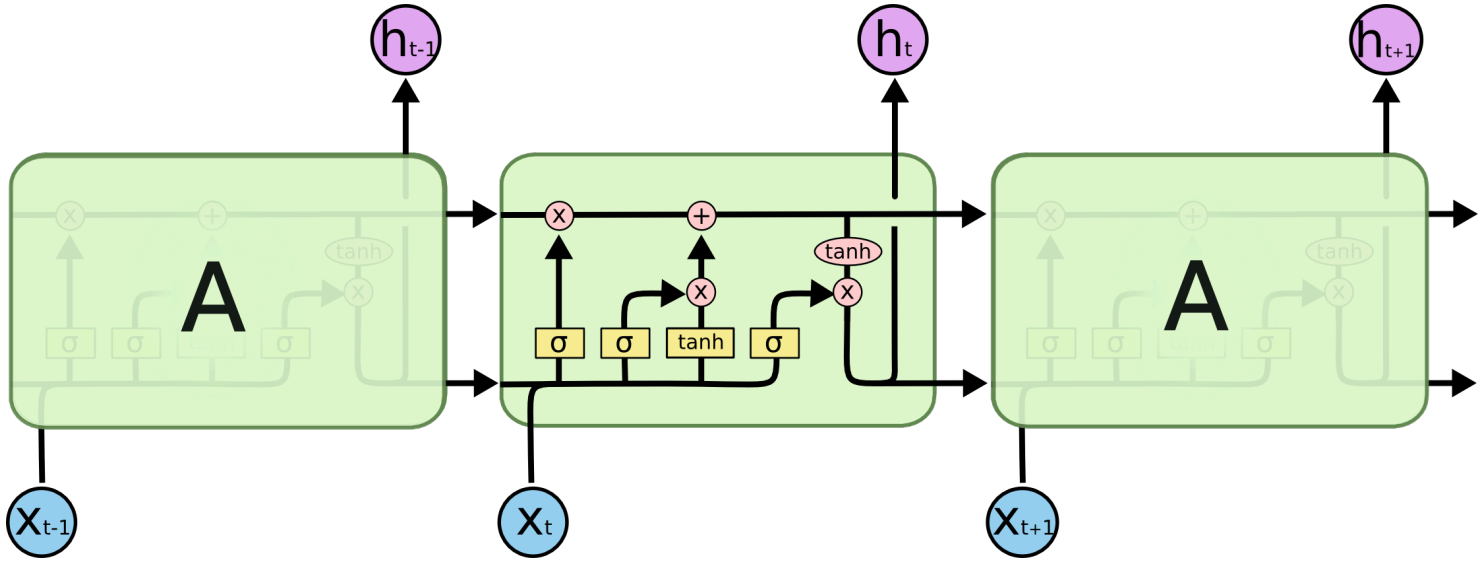
Window size = 4

- Vanishing gradient as in feedforward NN
  - "Future" loss has small effect on "present" cell
  - Squashing nonlinearity of hidden state is one problem

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

LSTM (Hochreiter & Schmidhuber ’94)

- Stands for long short-term memory cell

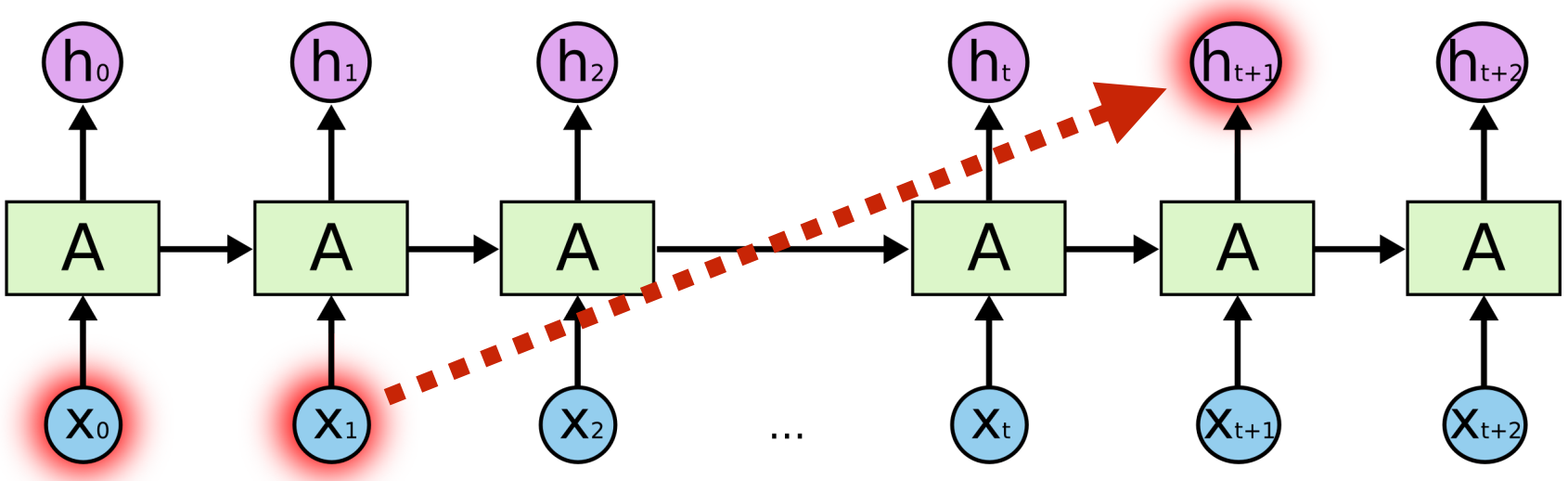


  - LSTM-RNN cell has
    - A short term “public” memory channel (bottom)
    - a long term “private” memory channel (top)
    - Private channel is used to carry long term history through the use of “gates”

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

Why LSTM

- The public hidden state in vanilla RNN cell “forgets” too fast

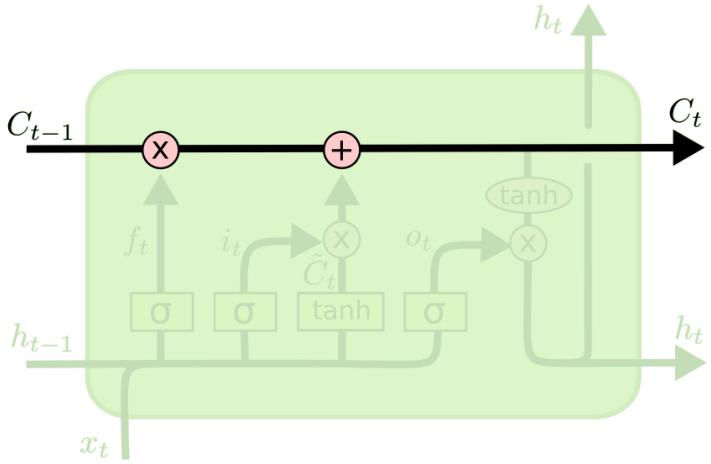


- The memory channel carries information that is strongly affected by current input
  - There are too many blocks in the way

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

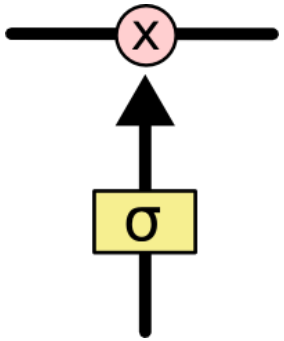
How LSTM solves this

- Providing a more persistent information channel



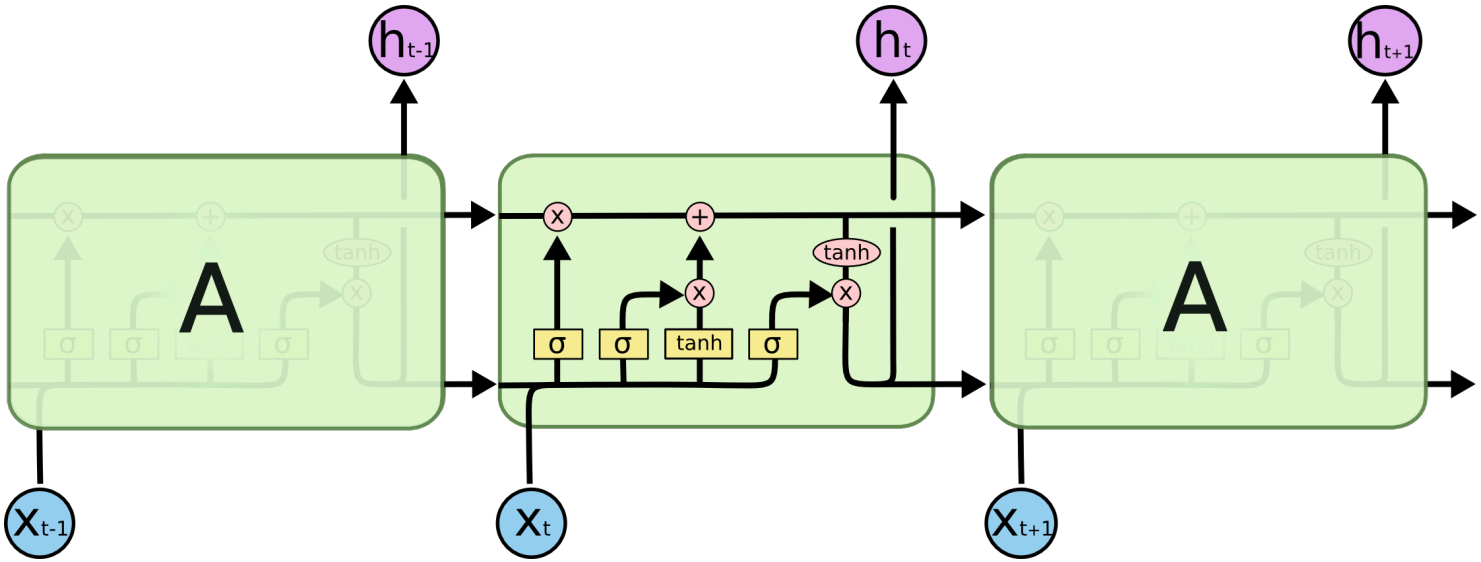
only simple multiplication and addition on the way

- Modulate this channel with gates: sigmoid function followed by point-wise multiplication



How LSTM solves this

- LSTM-RNN is “stateful” while CNN and RNN are stateless
  - Think about the long-term memory as a global variable in a function



  - Three gates: forget gate, input gate, output gate

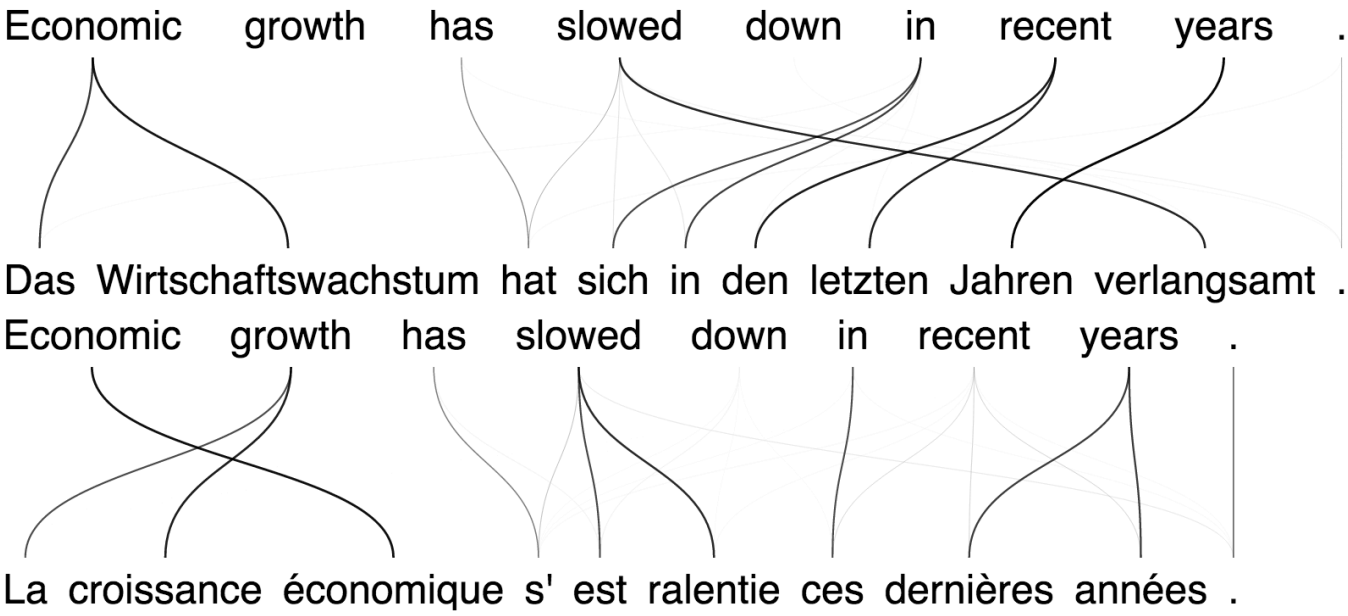
Image from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Summary

- RNN solves sequential learning problem through recursion
- One problem with RNN is the fast fading of history, which also leads to vanishing gradients in BPTT
- One solution to this problem is to introduce long term memory as in LSTM or GRU
- Training of LSTM-RNN is much complex and slow so recent trend is to replace RNN completely with CNN of many layers and skip connections

Seq2seq

- Sequence to sequence learning
  - Unlike CNN, we need to map one sequence to another sequence directly
  - Unlike RNN, it is hard to put symbols in order and handle different # of symbols
- Example machine translation

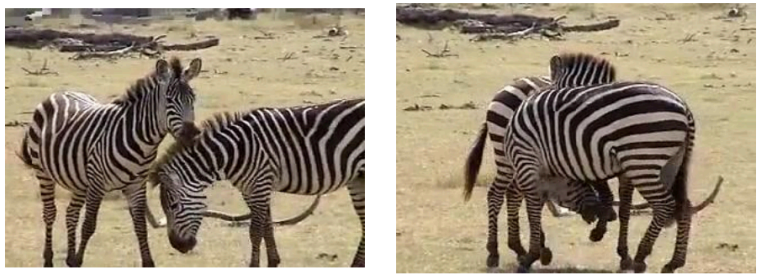


Seq2seq

- Applications
  - Neural machine translation

  - Neural speech recognition

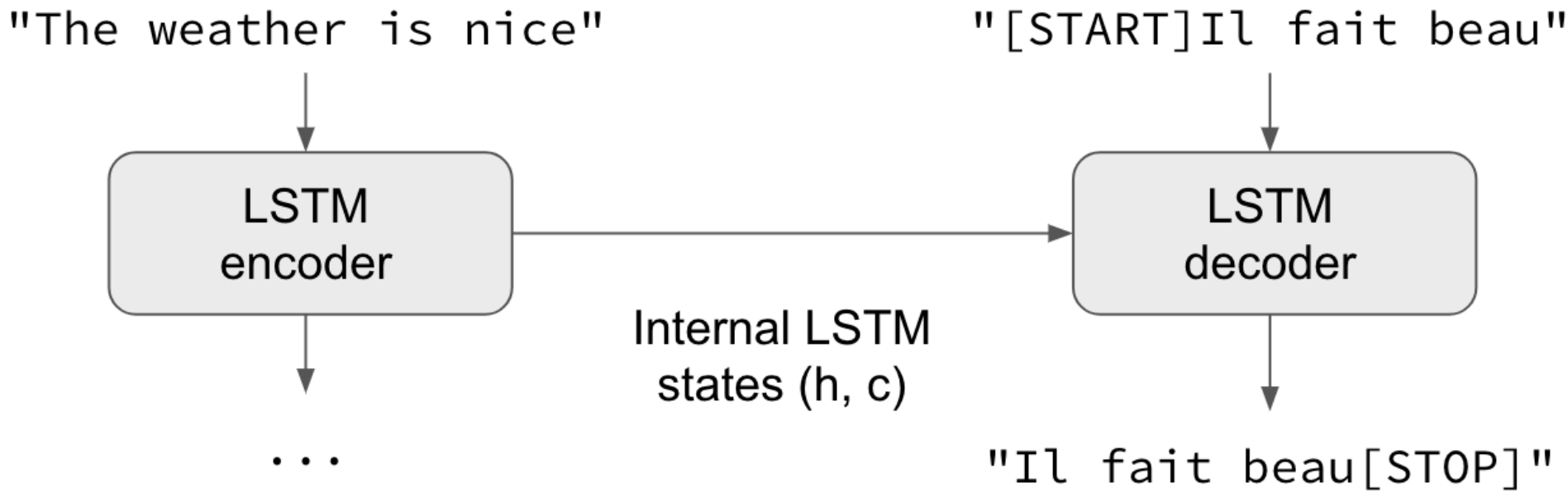
  - Video dubbing and captioning



S2VT: A herd of zebras are walking in a field.

Encoder-decoder model [Suskever eval 2014]

- Consider a simple machine translation task

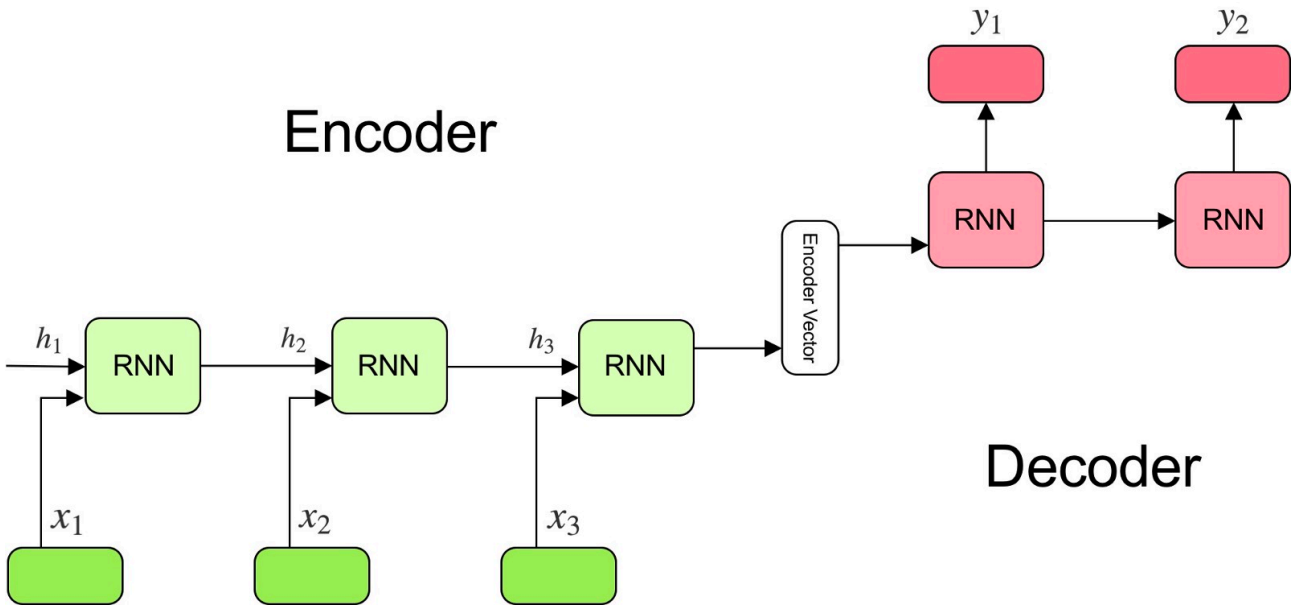


- The model has two modules
  - Encoder: an LSTM-RNN convert input into states (private & public)
  - Decoder: an LSTM-RNN generate output taking encoder's states

https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

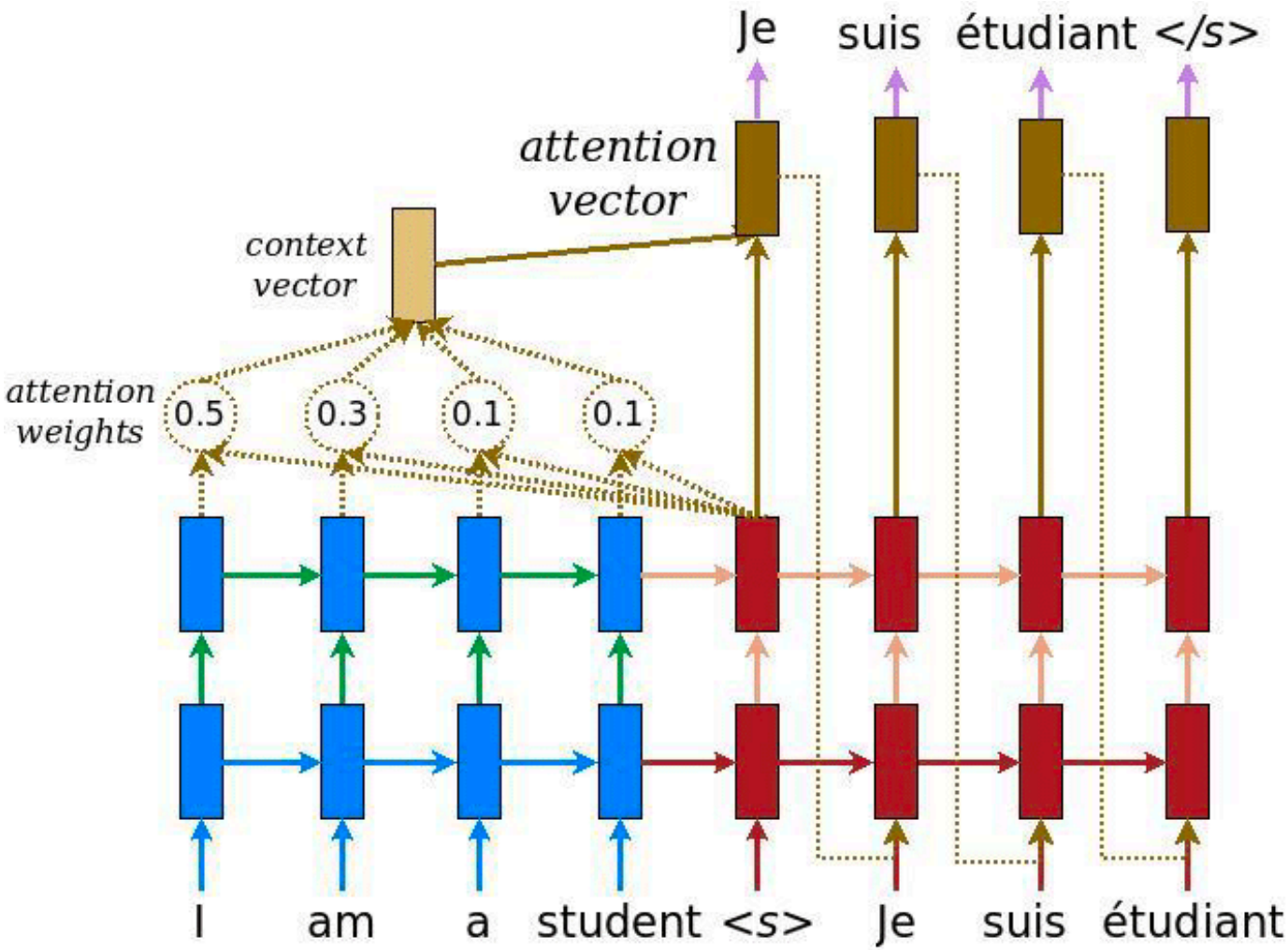
Encoder-decoder in seq2seq

- Many variants about how to transfer LSTM-RNN states from encoder to decoder
  - Last state or the average of all states
- Encoder and decoder can be trained individually



Attention in seq2seq

- "No symbols are created equal", so using only the summary state is too crude



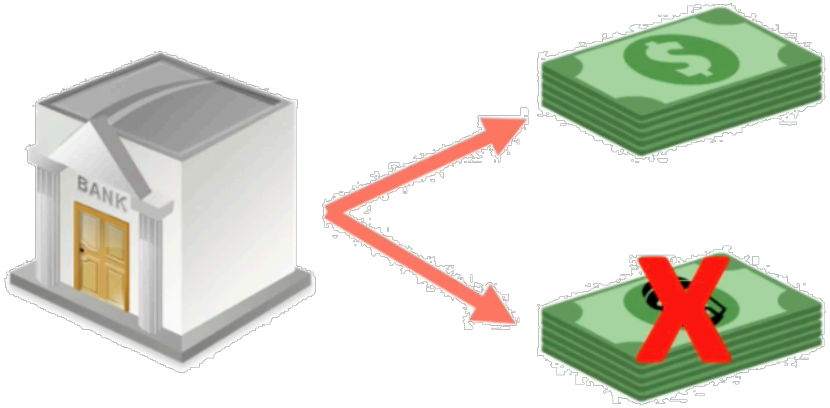
Summary

- Seq2seq learning uses RNN as building blocks but handles locally out of sequential order and different length of input and output
- Two new aspects in seq2seq become independently important in ML
  - Attention mechanism
  - Encoder-decoder architecture

GANs

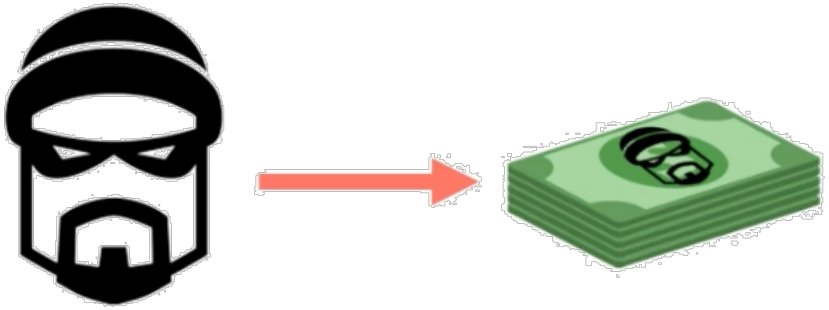
- Generative Adversary Network (GAN)
  - Goodfellow et.al., 2015
  - Discriminative training of generative models

Discriminator



Goal: produce counterfeit money that is as similar as real money

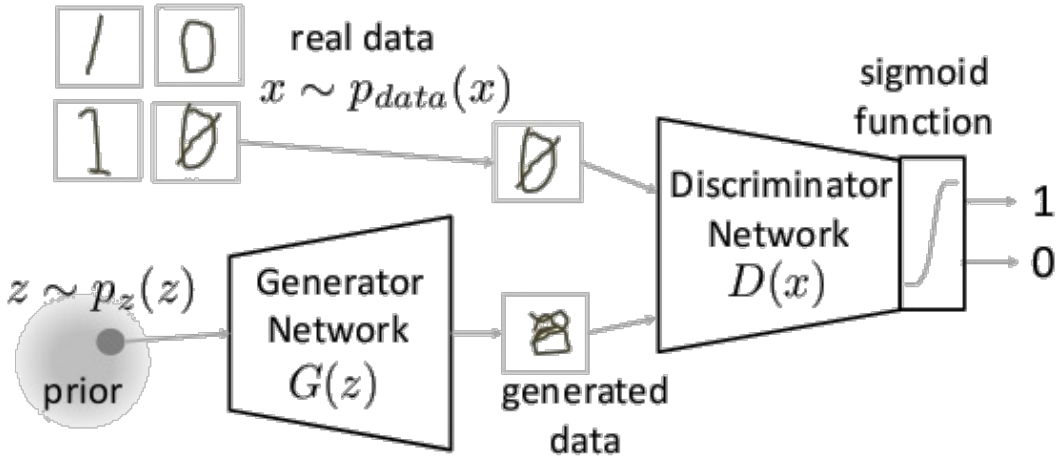
Generator

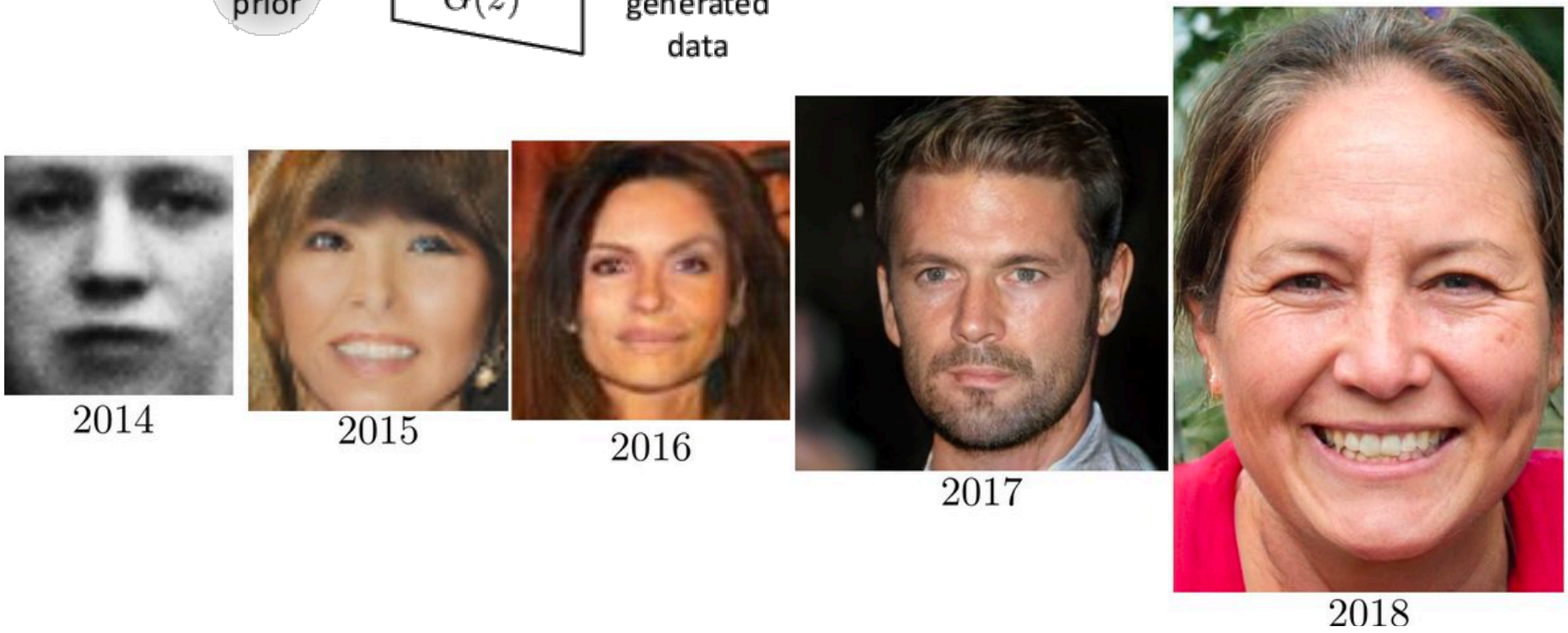


Goal: distinguish between real and counterfeit money

GAN

- The goal of GAN is to learn a generative model for data but using a discriminative NN as helper





Issues with DL

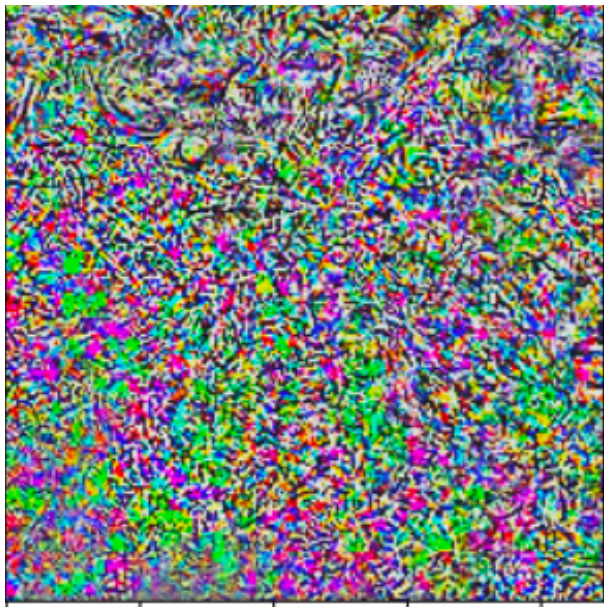
- DL systems are good tools, but have not lead to knowledge and insights
- DL cannot take advantage of prior knowledge effectively, and put too heavy reliance on data
- DL are only suitable for certain type of problems, but are not flexible and adaptive to general decision making process
- DL systems are brittle and can be broken with adversarial examples
- DL systems are potentially biased, and difficult to interpret and explain for the final results

Adversarial perturbations

- Deep neural network models are brittle to adversarial perturbations
- We can use gradient information of DNN based model to change classification results arbitrarily

Pig

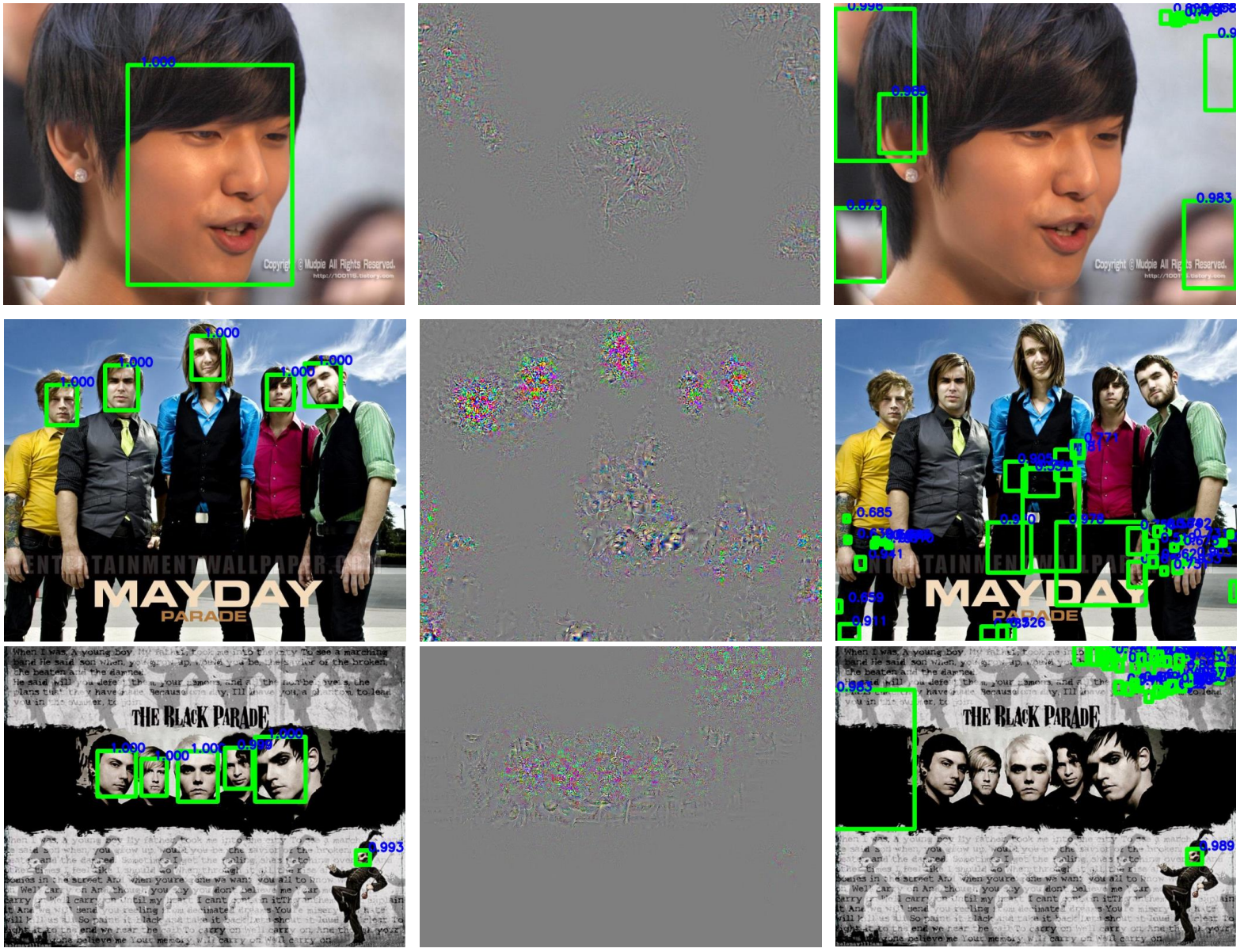
+



=

Airplane

Adversarial perturbations



Before perturbation **Noise added** **After perturbation**

Bias in deep learning model

- A DL algorithm is as good as the data we feed to it for training
  - Data can be contaminated or biased and can be manipulated (a procedure known as data poisoning)

(a) Three samples in criminal ID photo set S_c.

(b) Three samples in non-criminal ID photo set S_n

Explainability of DL models

- DL models are not explainable — how the conclusion is made, based on what?
  - So the trustworthiness of these algorithms are questionable