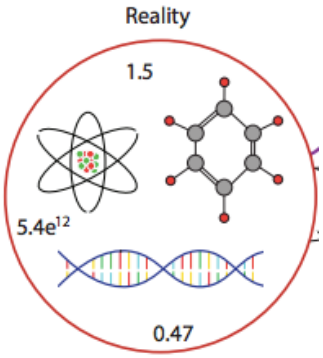
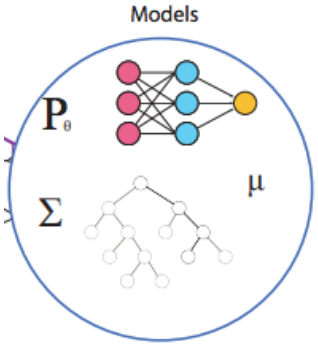


Statistics 154, Spring 2019



Modern Statistical Prediction and Machine Learning

Lecture 25: Neural Networks

Yuansi Chen

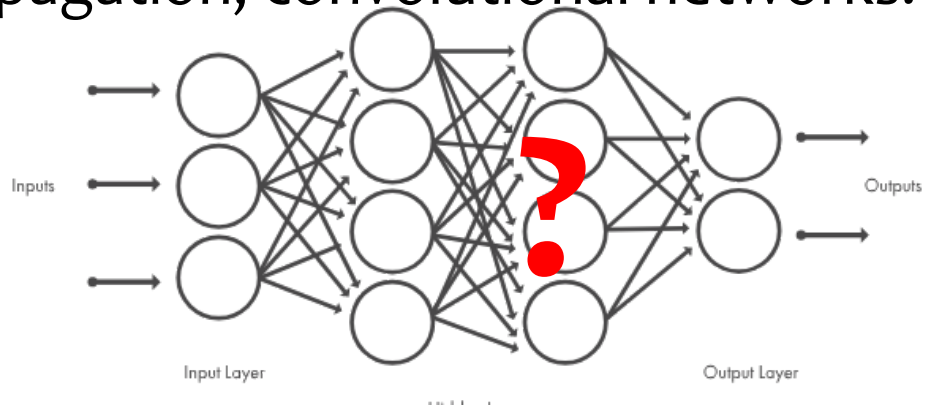
Instructor: Bin Yu

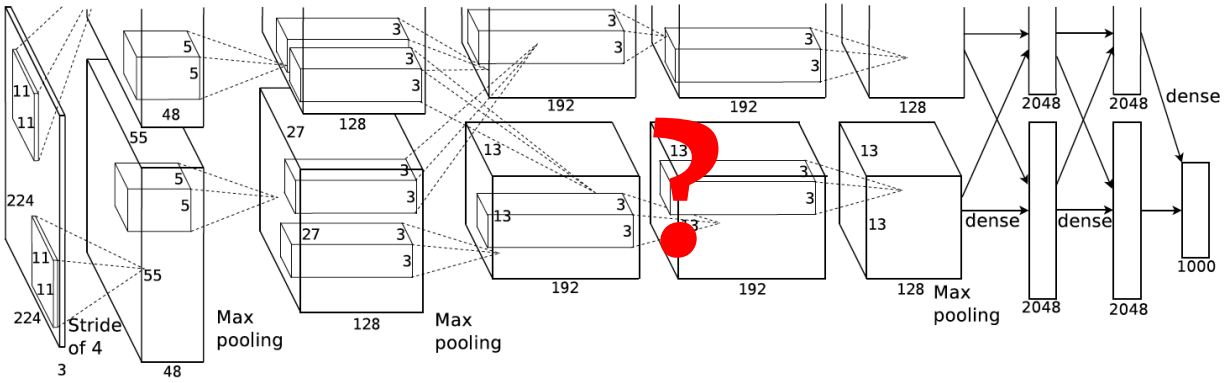
Logistics

- This is the last lecture. There will be review sessions during the lecture time.
- See Piazza post about Final exam and RRR week schedule
- Final Exam May 16th: 7-10 PM, Li Ka Sching 245
- For students with conflicts: 4-7 PM Evans 344. Mark your name on the relevant Piazza post.

What is this lecture about?

- Demystify neural networks, deep learning, backpropagation, convolutional networks.





From Krizhevsky et al. 2012

What is this lecture about?

- Demystify neural networks, deep learning, backpropagation, convolutional networks.

- Master neural network implementation, optimization and interpretation.

What is this lecture about?

- Demystify neural networks, deep learning, backpropagation, convolutional networks.

- Master neural network implementation, optimization and interpretation.

- Course suggestions:
  - STAT 157 Introduction to Deep Learning: Alex Smola and Mu Li
  - Graduate program in Statistics or computer science

In the search of complicated models

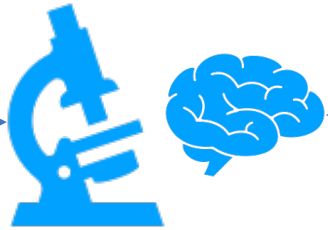
- Recall OLS and Kernel regression

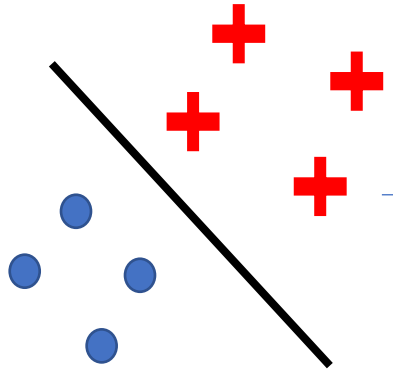
$$f(\mathbf{x}) = \beta^\top \mathbf{x} = \sum_{j=1}^{d} \beta_j x_j,$$

$$f(\mathbf{x}) = \beta^\top \phi(\mathbf{x}) = \sum_{j=1}^{p} \beta_j \phi_j(\mathbf{x})$$

- What if we can also learn the features $\phi(\mathbf{x})$?

More classical machine learning pipeline





Cat

Not cat

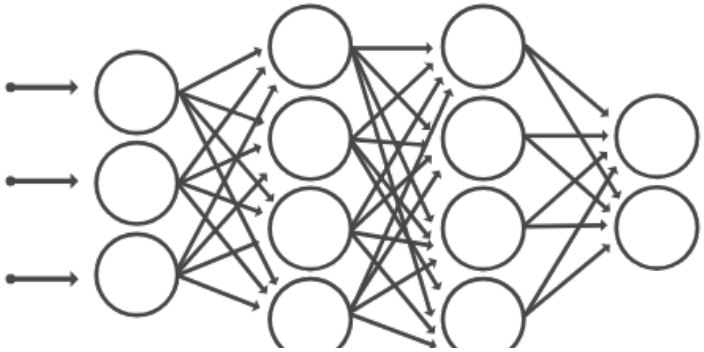
Input

Feature extraction

Classification

Output

Deep learning



Cat

Not cat

Input

Feature extraction + classification

Output

Is it hard to learn the features?

- Our loss minimization framework:

$$\min_{\beta} \sum_{i=1}^{n} \mathcal{L}(y_i, f(\mathbf{x}_i; \beta)) + \mathcal{R}(\beta)$$

- Ridge regression
- LASSO
- SVM
- Logistic regression

Is it hard to learn the features?

- Our loss minimization framework:

$$\min_{\beta} \sum_{i=1}^{n} \mathcal{L}(y_i, f(\mathbf{x}_i; \beta)) + \mathcal{R}(\beta)$$

- Make the feature learnable:

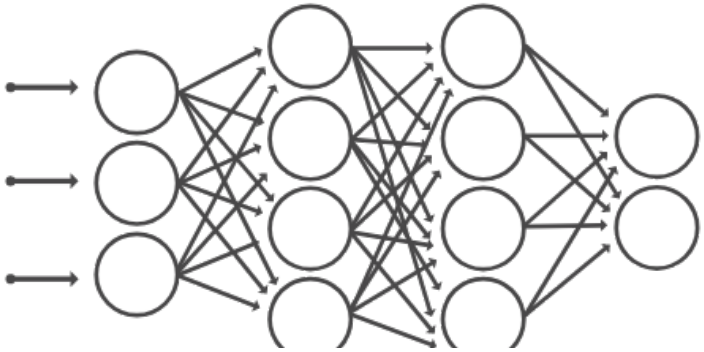
$$\min_{\phi,\beta} \sum_{i=1}^{n} \mathcal{L}(y_i, f(\phi(\mathbf{x}_i); \beta)) + \mathcal{R}(\phi, \beta)$$

Outline

- From OLS to neural network: basic concepts
- Neural Network Optimization: Backpropagation
- Convolutional networks and the effectiveness of big data

Neural network gives a generic way to parametrize the features

Input

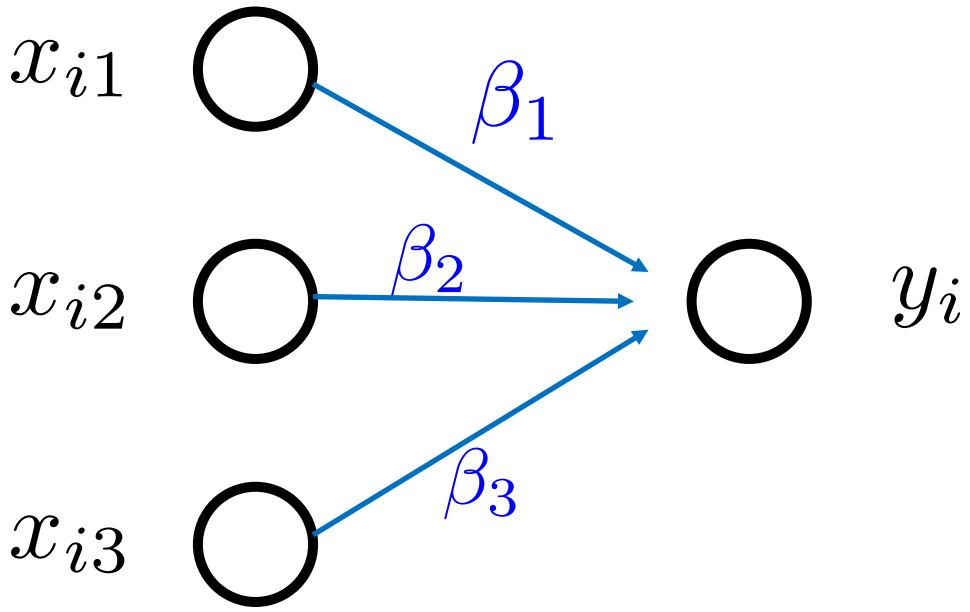


Output

$$\phi, \beta$$

What does each node mean?

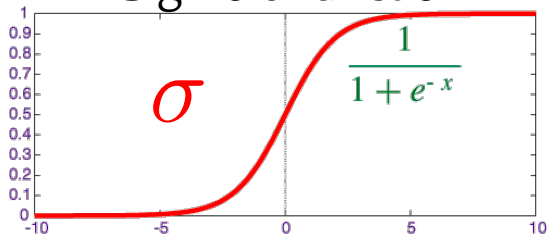
OLS as a simple neural network

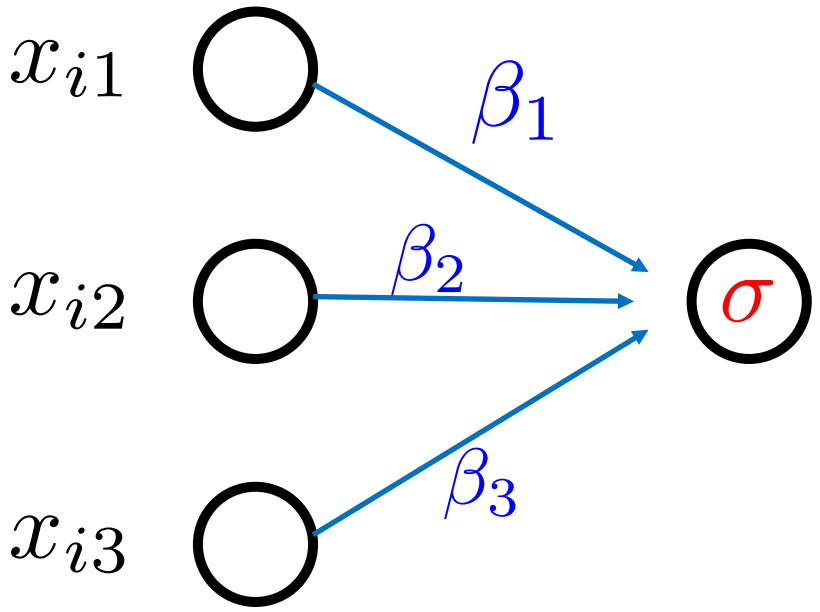


$$y = \mathbf{x}^\top \beta$$

Logistic regression as a simple neural network

Sigmoid function

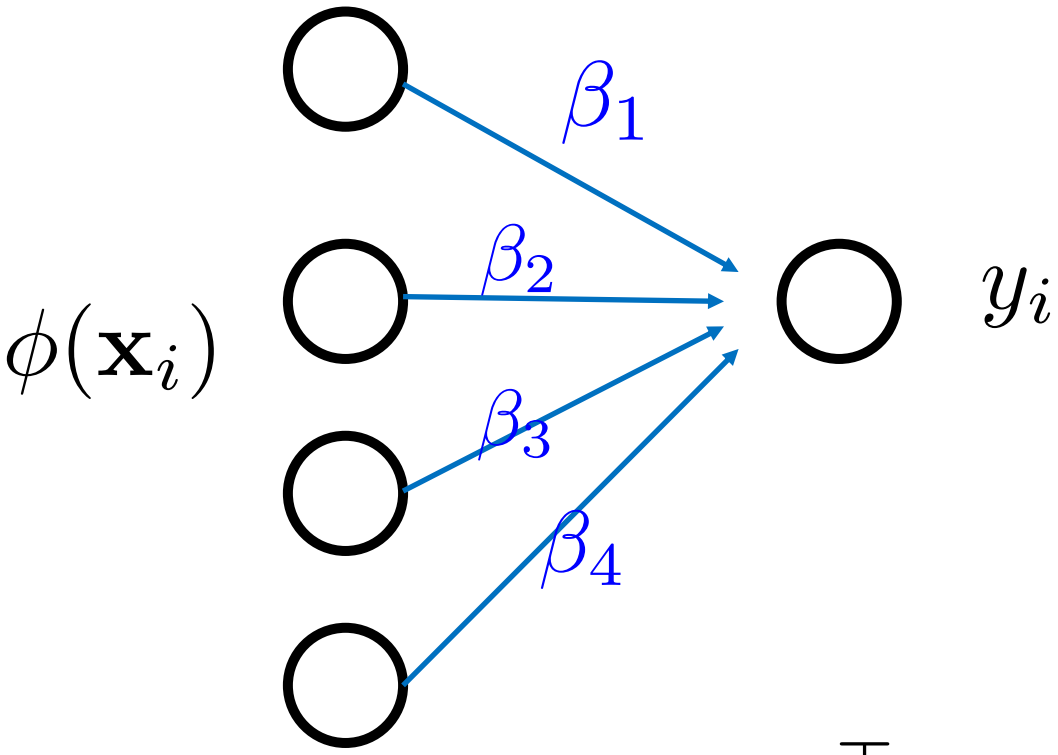




$$P(y_i = 1 \mid \mathbf{x}_i)$$

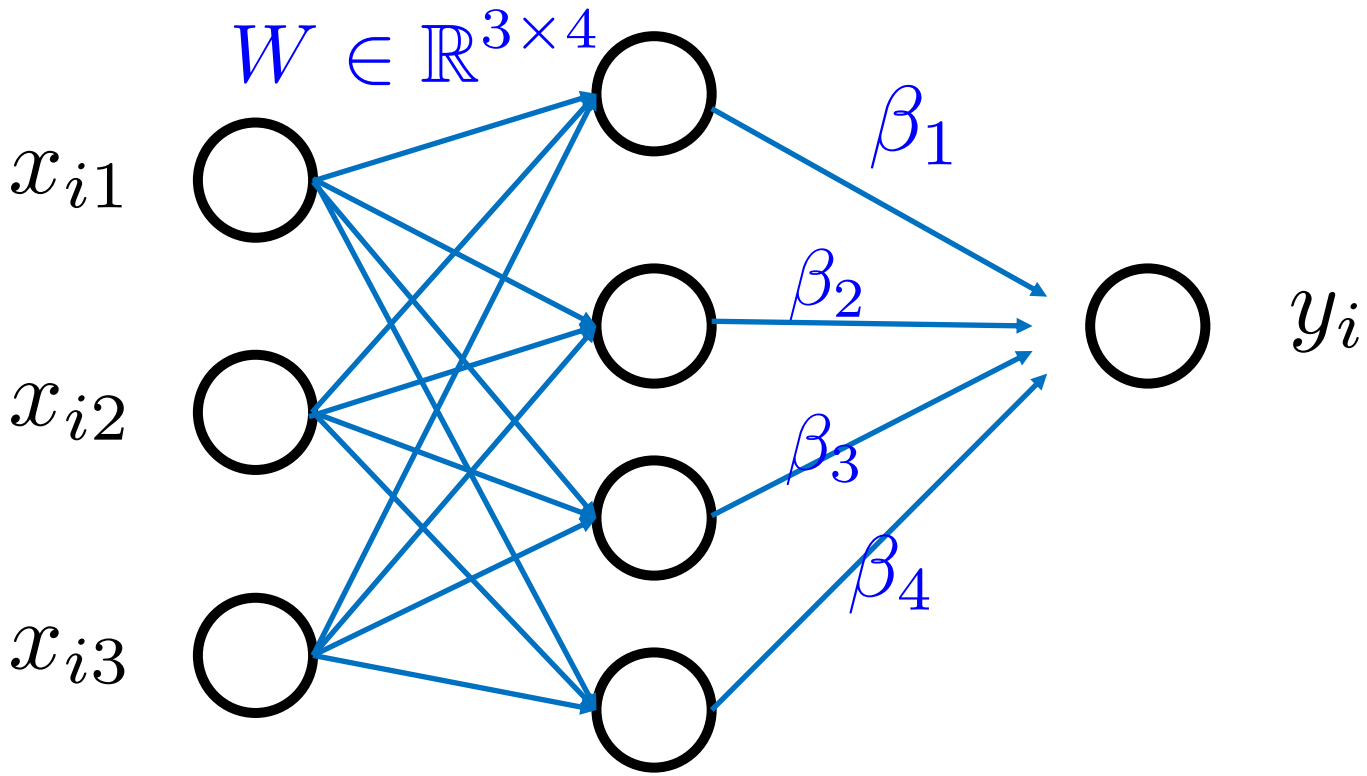
$$P(y_i = 1 \mid \mathbf{x}_i) = \sigma(\mathbf{x}^\top \beta)$$

Need a feature extraction layer



$$y = \phi(\mathbf{x})^\top \beta$$

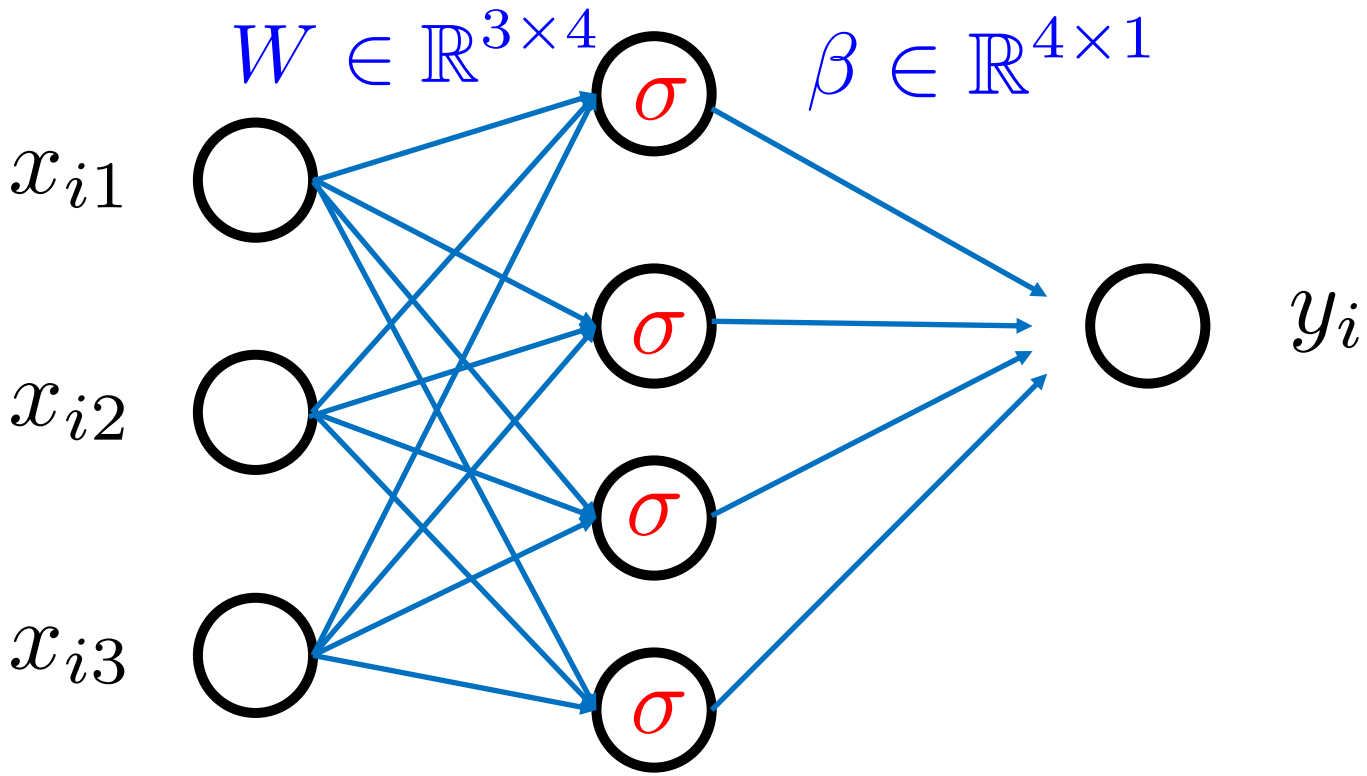
Need a feature extraction layer: Will a linear layer work?



$$y = (Wx)^\top \beta = x^\top \left(W^\top \beta\right)$$

Still linear

Break the linearity via activation function!

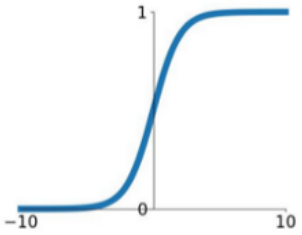


$$y = h^\top \beta = [\sigma\,(Wx)]^\top \beta$$

Typical activation functions

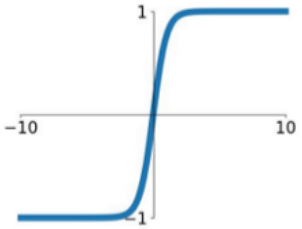
Sigmoid

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



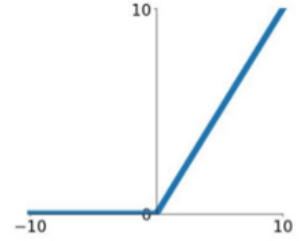
tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



ReLU = Rectified Linear Unit

Does it work?

- Try it out at Tensorflow playground https://playground.tensorflow.org

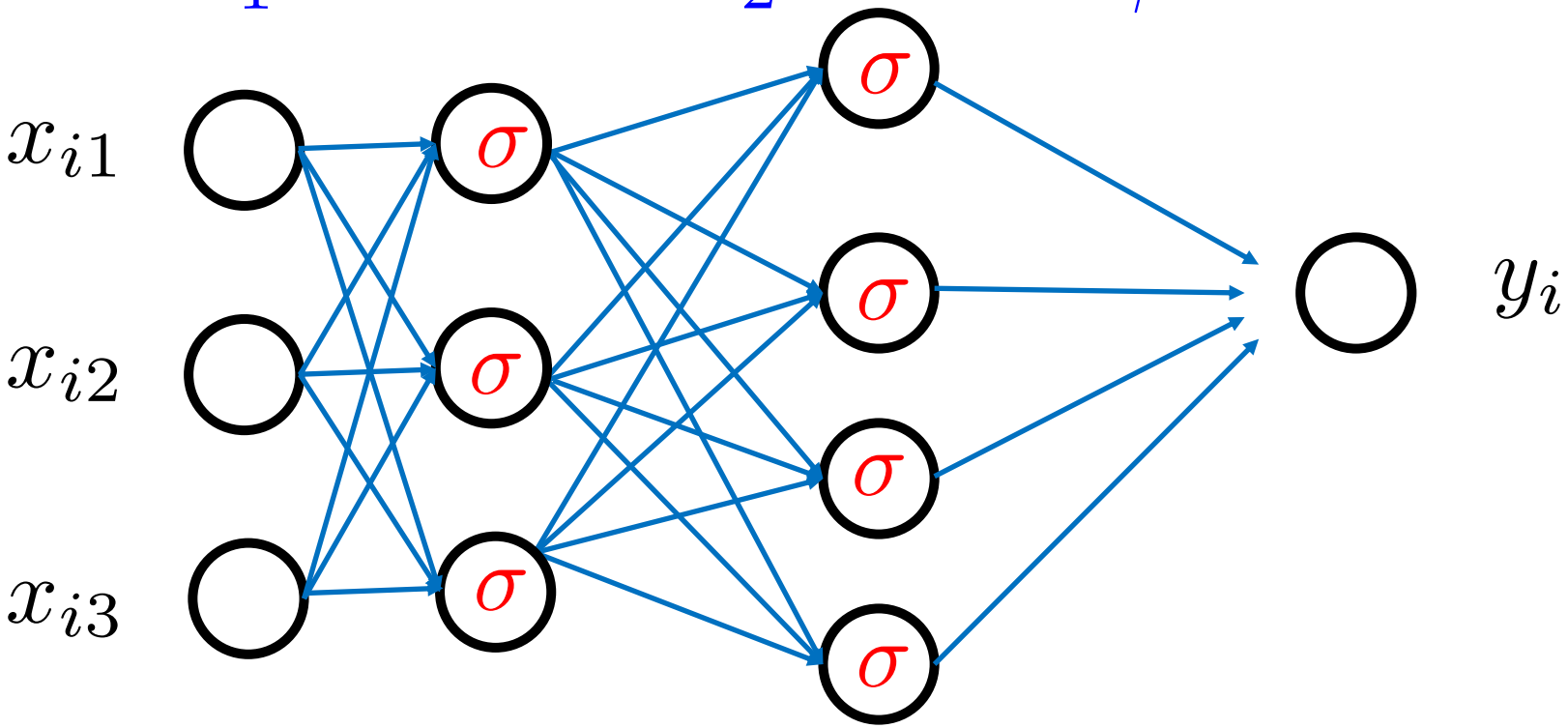
- Linear separable case:
  - 0 hidden layer + 1 node = logistic regression

- Circle case:
  - 0 hidden layer NN no longer works
  - 1 hidden layer : increase the number of neurons to observe the decision boundary. Things start to work from 3 middle neurons.

- Spiral:
  - 1 hidden layer roughly works but not perfect
  - 2 hidden layer: observe the decision boundary
- Try regression

Deeper model is always available

$$W_1 \in \mathbb{R}^{3\times 3} \quad W_2 \in \mathbb{R}^{3\times 3} \quad \beta \in \mathbb{R}^{4\times 1}$$



$$y = h_2^\top \beta = \left[\sigma\left(W_2 h_1\right)\right]^\top \beta$$

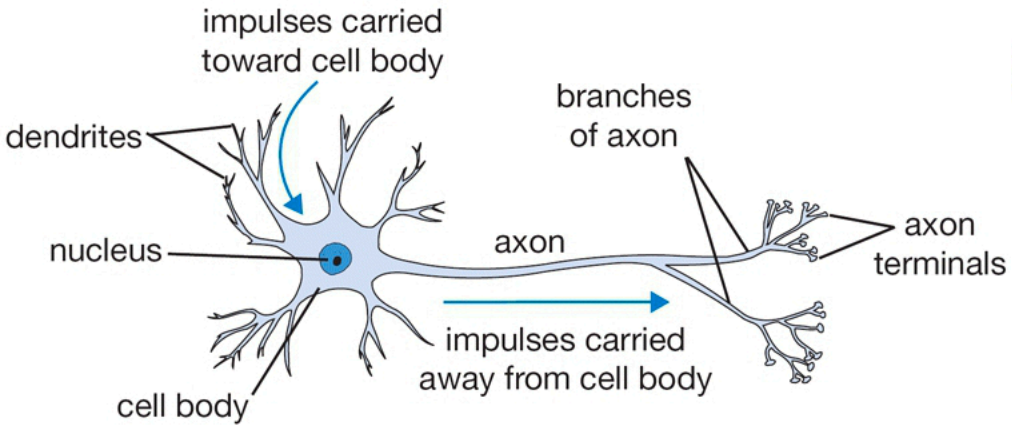
$$= \left[\sigma\left(W_2 \sigma\left(W_1 x\right)\right)\right]^\top \beta$$

Biological inspiration

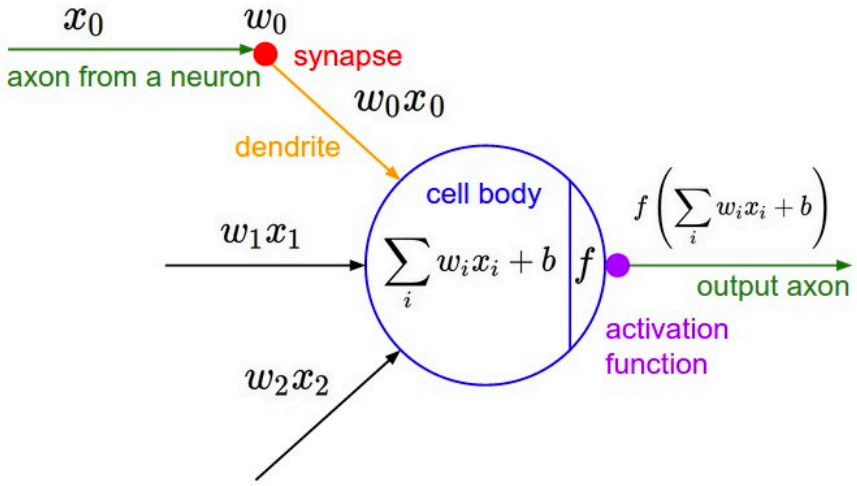
- Basic computational unit in human brain: a neuron
- Human nervous system contains 86 billion neurons connected with 10^15 synapses

http://med.stanford.edu/content/dam/sm-news/images/2016/12/brain-synapses-stock.jpg

Biological inspiration



Cartoon biological neuron



Mathematical model

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

Universal approximation theorem

- (Cybenko 1989): a feed–forward neural network with a single hidden layer containing a finite number of neurons (with sigmoid activation function) can approximate any continuous functions on compact subsets of R^d

- The theorem states that simple neural networks can represent a wide variety of interesting functions when given enough number of neurons

- Series of work to cover general activation functions: Hornik 1991, Csaji 2001, Lu 2017…

However

- The universal approximation theorem does not touch upon the learnability of the parameters:
  - What is an efficient optimization algorithm that finds the good parameters?
  - How much data samples are needed to learn such a neural network?

Outline

- From OLS to neural network: basic concepts
- Neural Network Optimization: Backpropagation
- Convolutional networks and the effectiveness of big data

Neural network optimization

- Loss minimization problem in 1-hidden layer NN:

$$\min_{\phi,\beta} \sum_{i=1}^{n} \mathcal{L}(y_i, [\sigma(W x_i)]^\top \beta))$$

Neural network optimization

- Loss minimization problem in 1-hidden layer NN:

$$\min_{\phi,\beta} \sum_{i=1}^{n} \mathcal{L}(y_i, [\sigma(Wx_i)]^\top \beta))$$

- A composition of four functions:

$$f_W : x \mapsto Wx$$

$$\sigma : z \mapsto \sigma(z)$$

$$g_\beta : z \mapsto z^\top \beta$$

$$L : z \mapsto \mathcal{L}(y_i, z)$$

$$\min_{W,\beta} L(g_\beta(\sigma(f_W(x))))$$

Backpropagation: chain rule for NN

- Chain rule for a simple 1D composition:

$$\frac{\partial g(f_w(x))}{\partial w} = g'(f_w(x)) \cdot \frac{\partial f_w(x)}{\partial w}$$

Backpropagation: chain rule for NN

- Chain rule for a simple 1D composition:

$$\frac{\partial g(f_w(x))}{\partial w} = g'(f_w(x)) \cdot \frac{\partial f_w(x)}{\partial w}$$

- Chain rule for a more complicated 1D composition:

$$\frac{\partial h(g_v(f_w(x)))}{\partial v} = h'(g_v(f_w(x))) \cdot \frac{\partial g_v}{\partial v}(f_w(x))$$

$$\frac{\partial h(g_v(f_w(x)))}{\partial w} = h'(g_v(f_w(x))) \cdot g_v'(f_w(x)) \cdot \frac{\partial f_w}{\partial w}(x)$$

Chain rule for 1D feed-forward NN:

One forward pass and one backward pass is sufficient to compute all the partial derivatives

$$\frac{\partial h(g_v(f_w(x)))}{\partial v} = h'(g_v(f_w(x))) \cdot \frac{\partial g_v}{\partial v}(f_w(x))$$

$$\frac{\partial h(g_v(f_w(x)))}{\partial w} = h'(g_v(f_w(x))) \cdot g_v'(f_w(x)) \cdot \frac{\partial f_w}{\partial w}(x)$$

Reusable

Backpropagation

- General backprop is done is a staged fashion
  - One forward pass to compute all layer-wise function values
  - One backward pass to compute all layer-wise partial derivatives

- More complicated when the 1D case, because derivative with respect to matrix is need.
  Specifically, the derivative of matrix-matrix multiplication

$$\frac{\partial(WX)}{\partial W}$$

This will give a tensor: http://cs231n.stanford.edu/vecDerivs.pdf

Back to 1-hidden layer NN optimization

- Loss minimization problem in 1-hidden layer NN:

$$\min_{W,\beta} \sum_{i=1}^{n} \mathcal{L}(y_i, [\sigma(Wx)]^\top \beta))$$

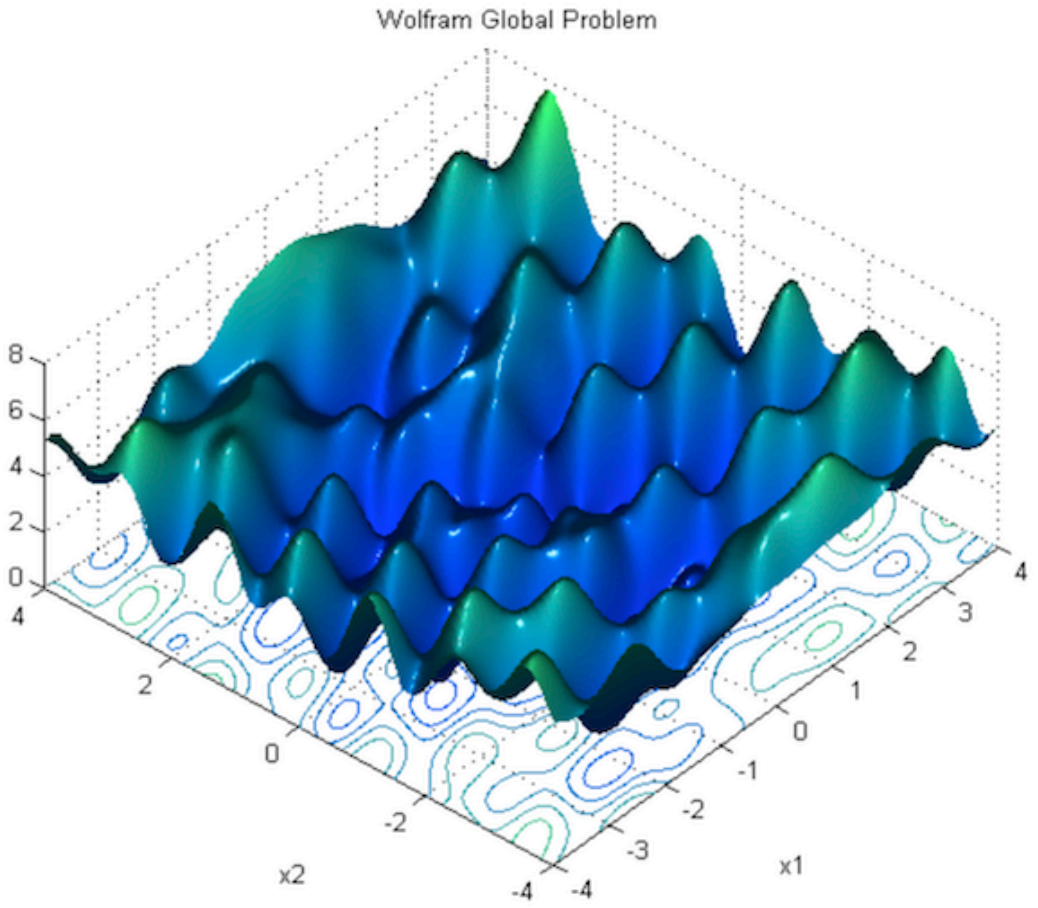
- Backprop => gradient for each layer's parameters

- We can use gradient descent or stochastic gradient descent to optimize

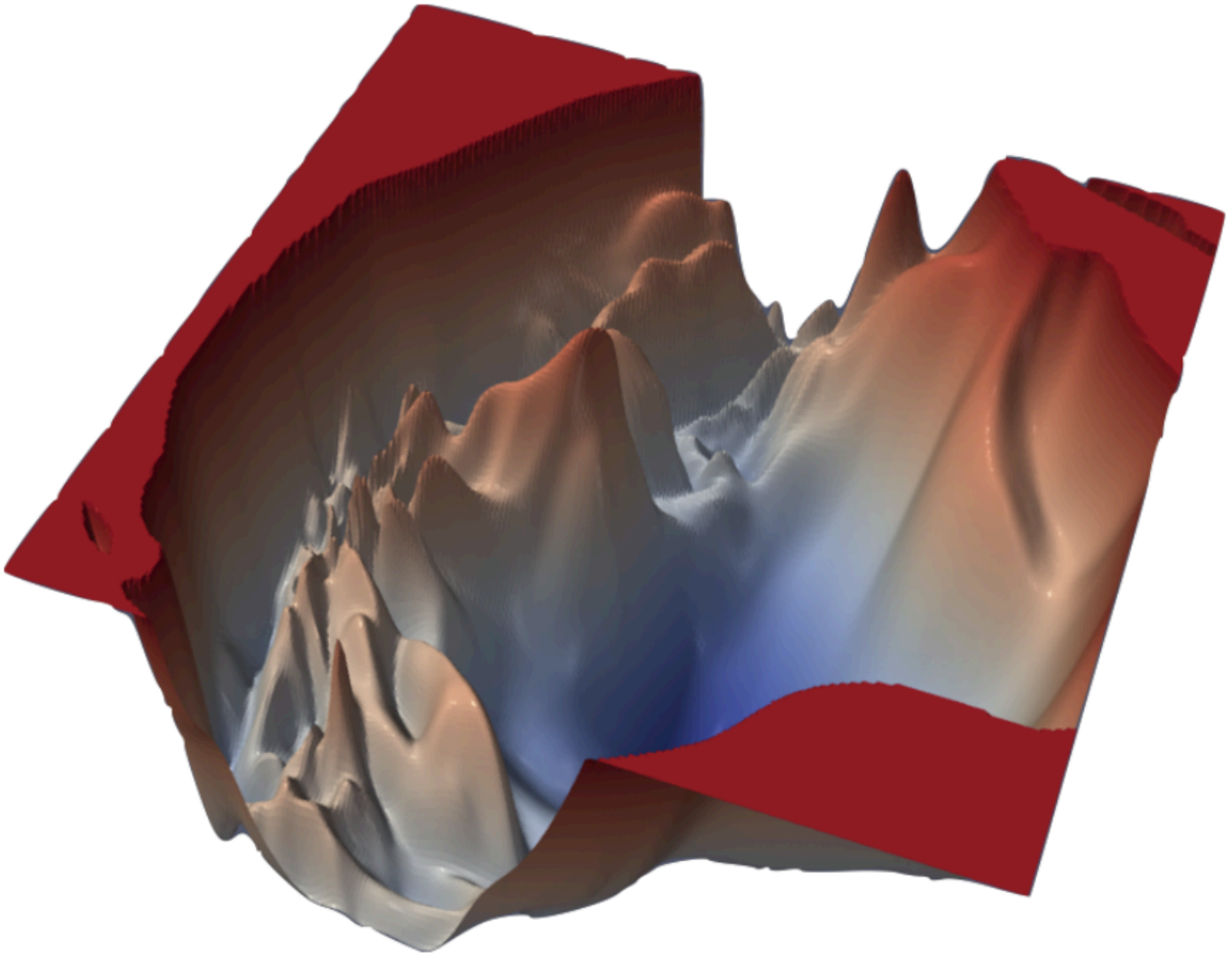
Is the neural network optimization problem convex?

- No!

- With two nodes in the hidden layer, we can have this:



Pictures from Yoshua Bengio



(a) ResNet-110, no skip connections

(Li et al. 2018)
2D slide of a practical neural network that has good classification performance

Are we stuck?

- No generic solution to the non-convex optimization problem so far

- Tons of heuristics (need many experience)

- Be brave with stochastic gradient descent!
- Be patient in your step-size parameter tuning on your validation set!

Outline

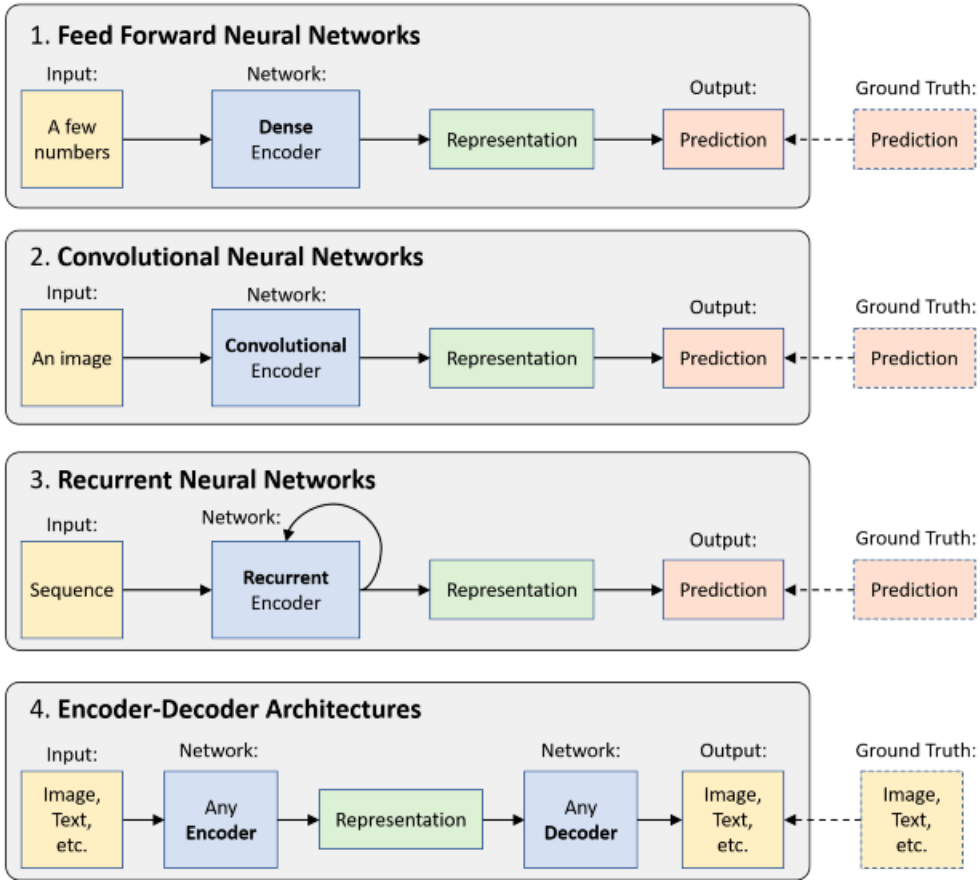
- From OLS to neural network: basic concepts
- Neural Network Optimization: Backpropagation
- Convolutional networks and the effectiveness of big data

Neural networks are highly engineerable!

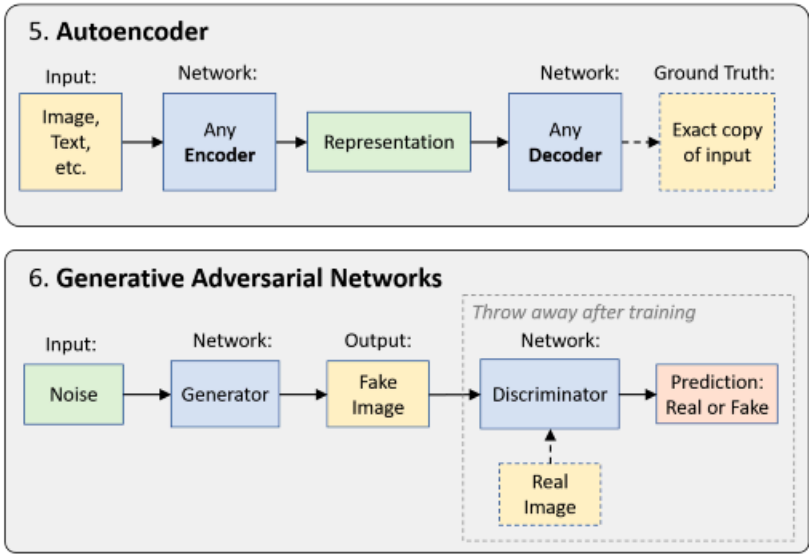
- Regularizations
- Constraints
- Activation functions

Neural network architectures from Tensorflow tutorials

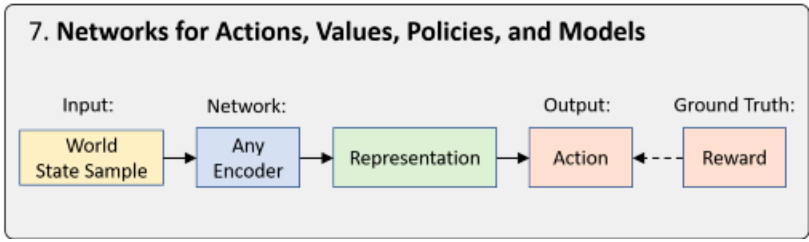
Supervised Learning



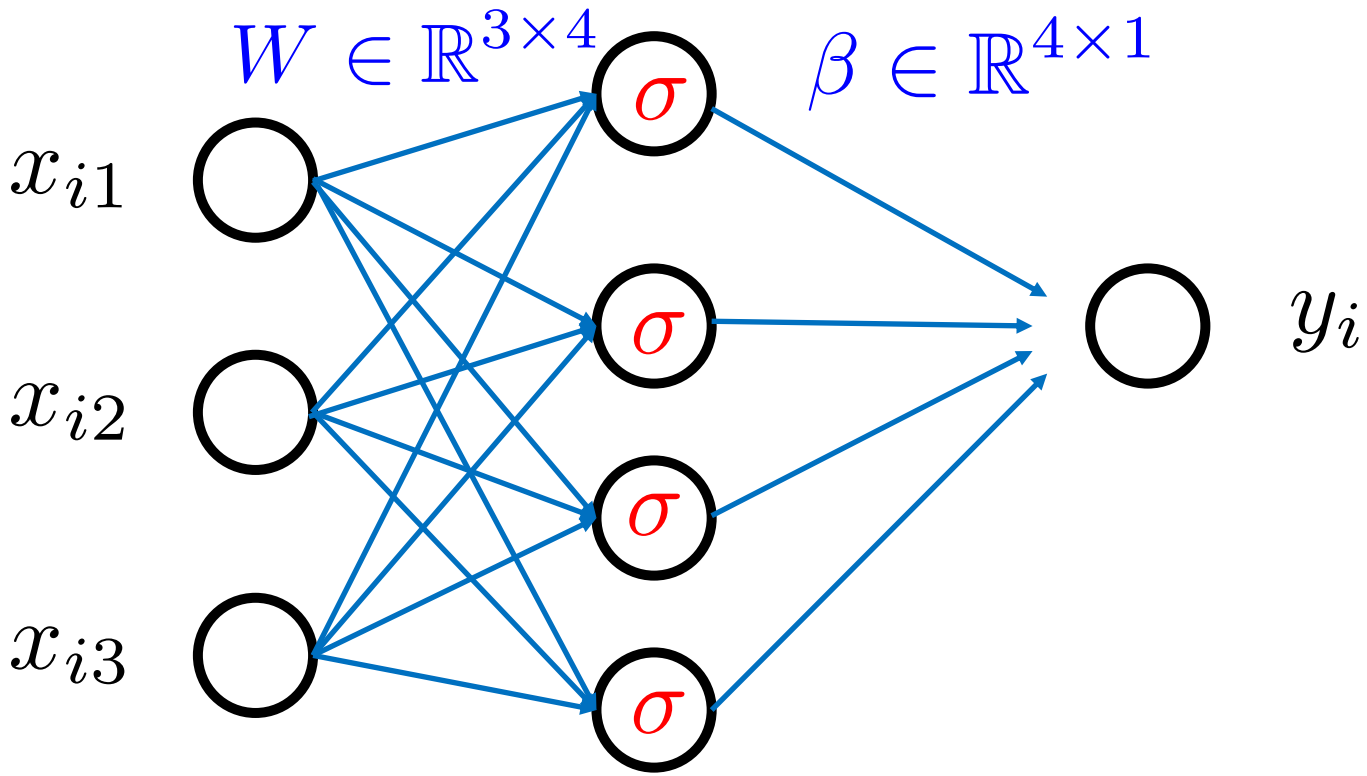
Unsupervised Learning



Reinforcement Learning

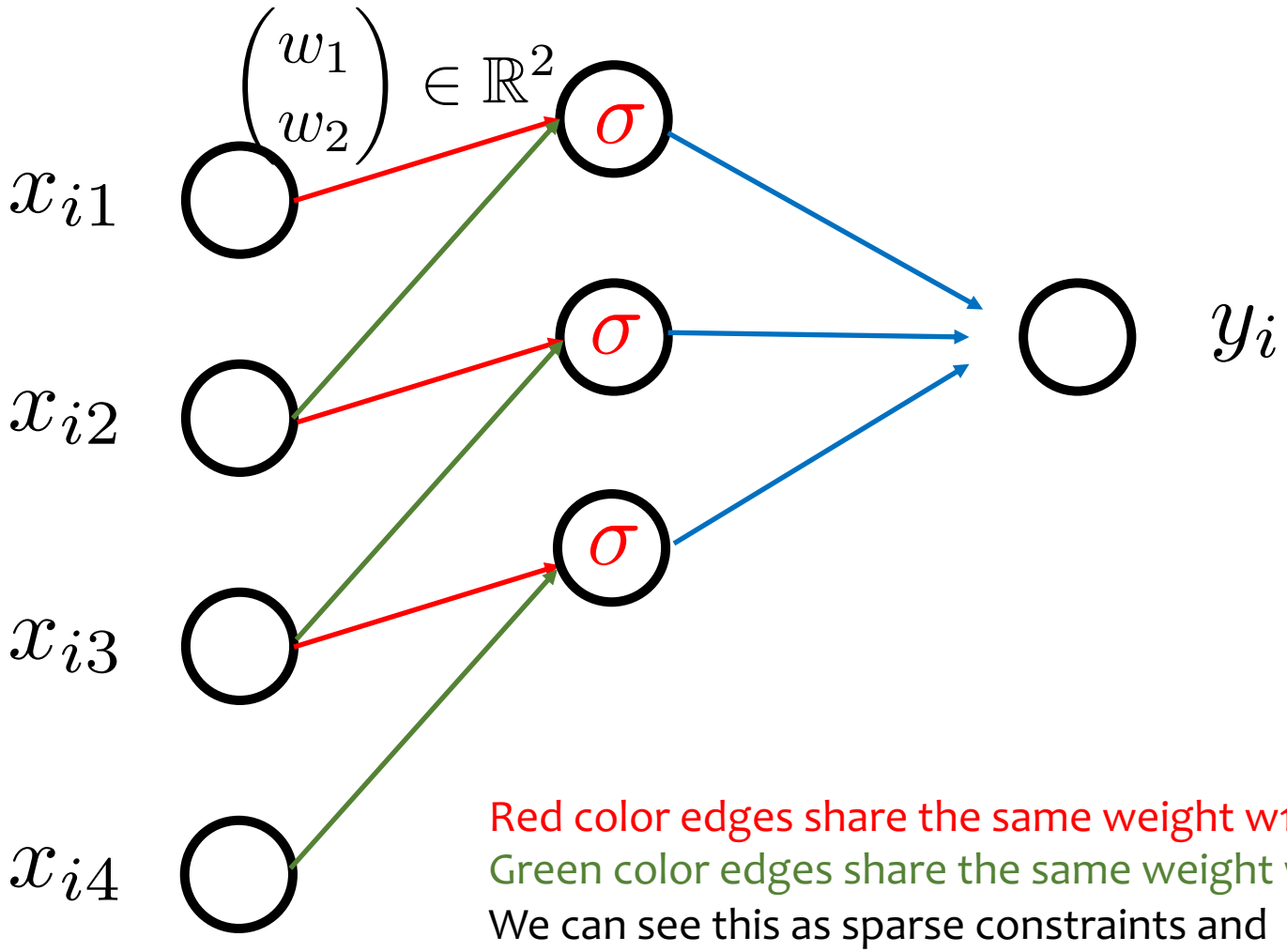


A single-hidden-layer dense network



$$y = h^\top \beta = [\sigma\,(Wx)]^\top \beta$$

A convolutional neural network (CNN): with one convolutional filter

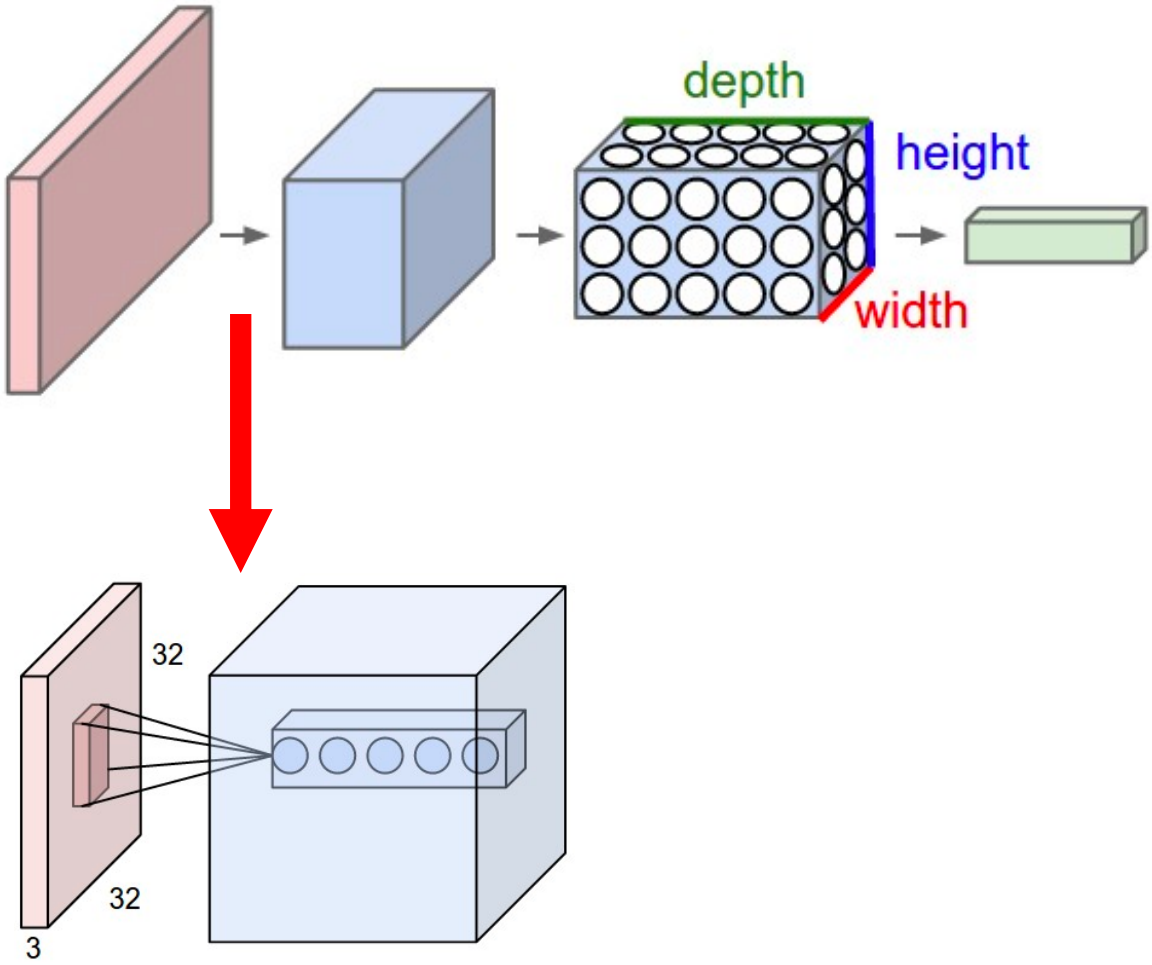


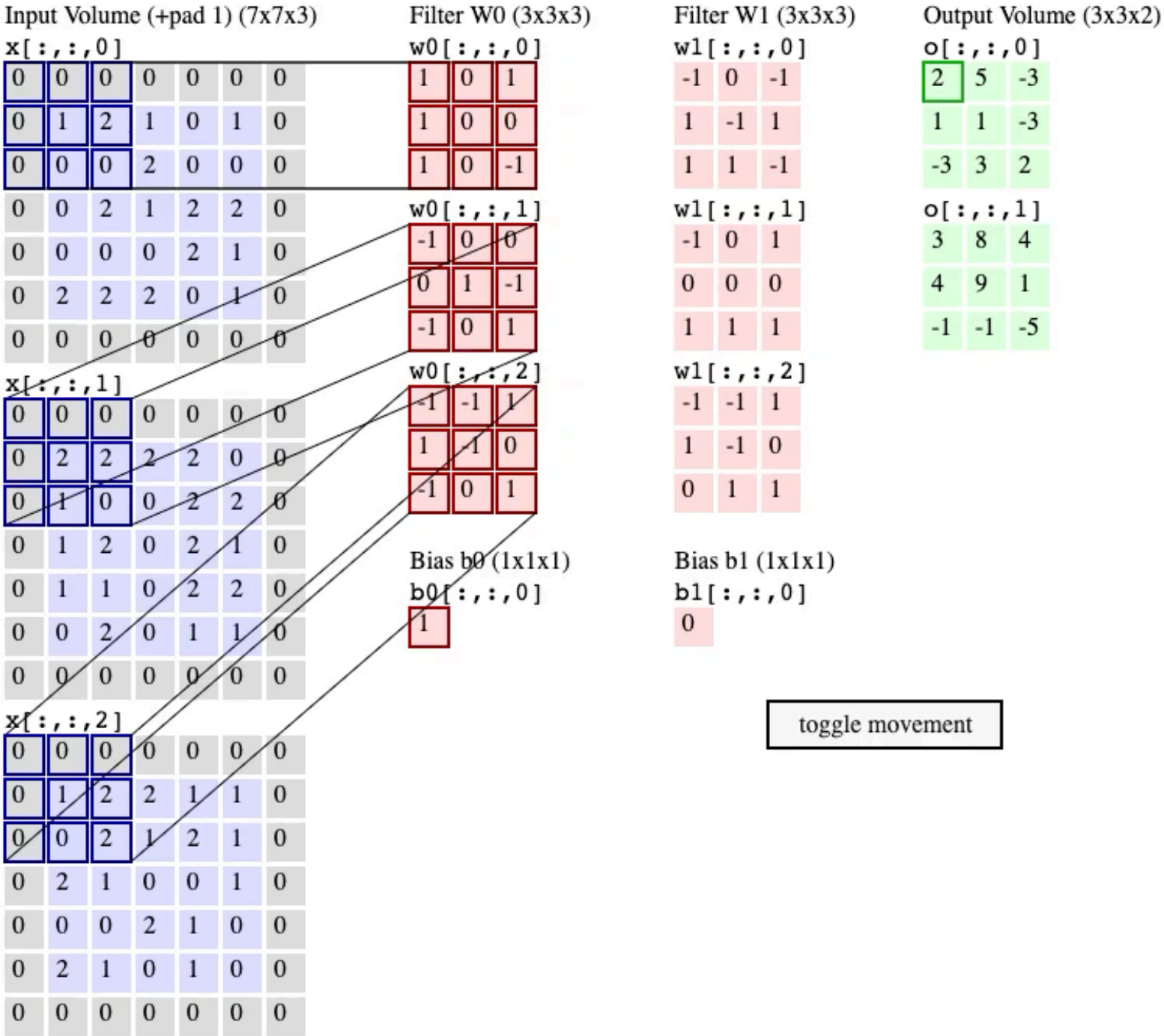
Red color edges share the same weight w1

Green color edges share the same weight w2

We can see this as sparse constraints and equality constraints

CNNs on 2D color image data

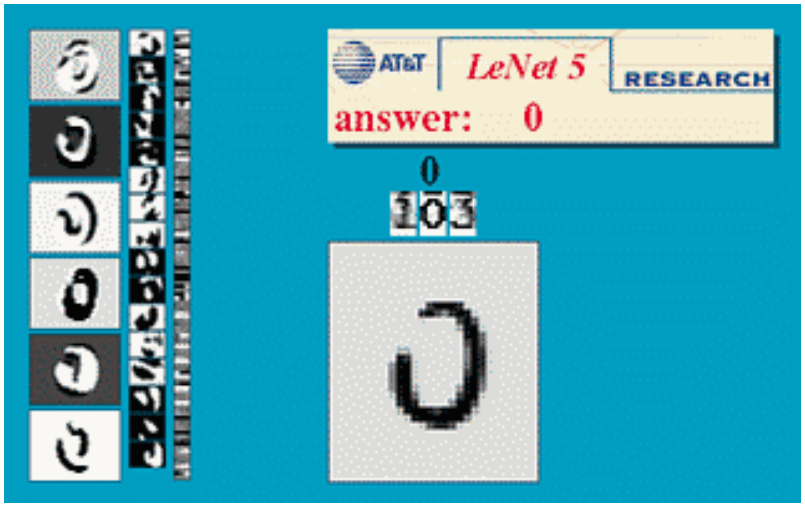


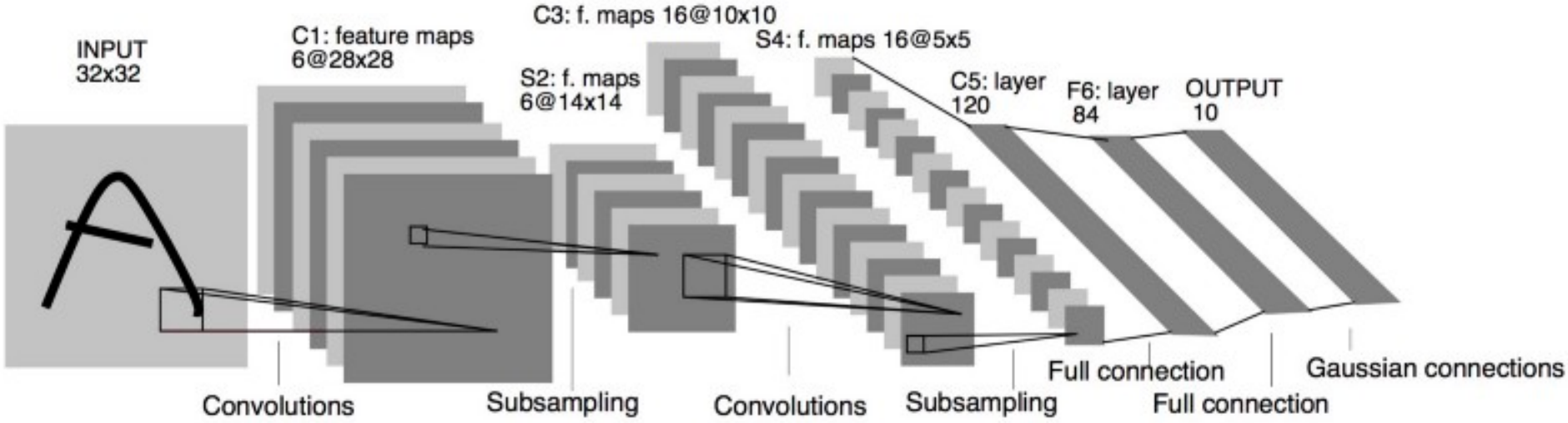

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 1 0 1 0
0 0 0 2 0 0 0
0 0 2 1 2 2 0
0 0 0 0 2 1 0
0 2 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 2 2 0 0
0 1 0 0 2 2 0
0 1 2 0 2 1 0
0 1 1 0 2 2 0
0 0 2 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 2 2 1 1 0
0 0 2 1 2 1 0
0 2 1 0 0 1 0
0 0 0 2 1 0 0
0 2 1 0 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 0 1
1 0 0
1 0 -1
w0[:,:,1]
-1 0 0
0 1 -1
-1 0 1
w0[:,:,2]
-1 -1 1
1 -1 0
-1 0 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 -1
1 -1 1
1 1 -1
w1[:,:,1]
-1 0 1
0 0 0
1 1 1
w1[:,:,2]
-1 -1 1
1 -1 0
0 1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 -3
1 1 -3
-3 3 2
o[:,:,1]
3 8 4
4 9 1
-1 -1 -5
toggle movement


Input

CNN for digit recognition

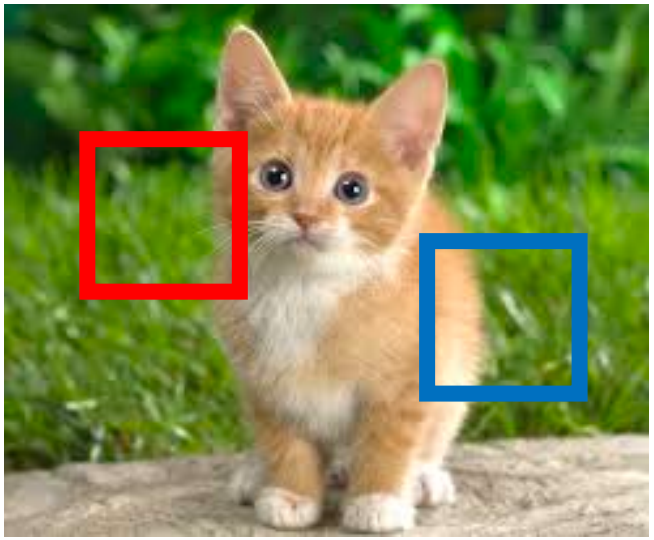
LeCun et al. 1989





Intuitions for CNN

- Regular Dense neural nets don't scale well to full images. Full connectivity is observed to wasteful and the huge number of parameters would quickly lead to overfitting.
- **Convolution**: Take advantage of stationarity of small image patches!



If one crops patches at red and blue locations for millions of images and compute the statistics,

how close do you think are the red patch distribution vs blue patch distribution?

- **Subsampling/pooling**: Local translation invariance.

How about the other part of learnability?

- How many data samples we need to train neural network?

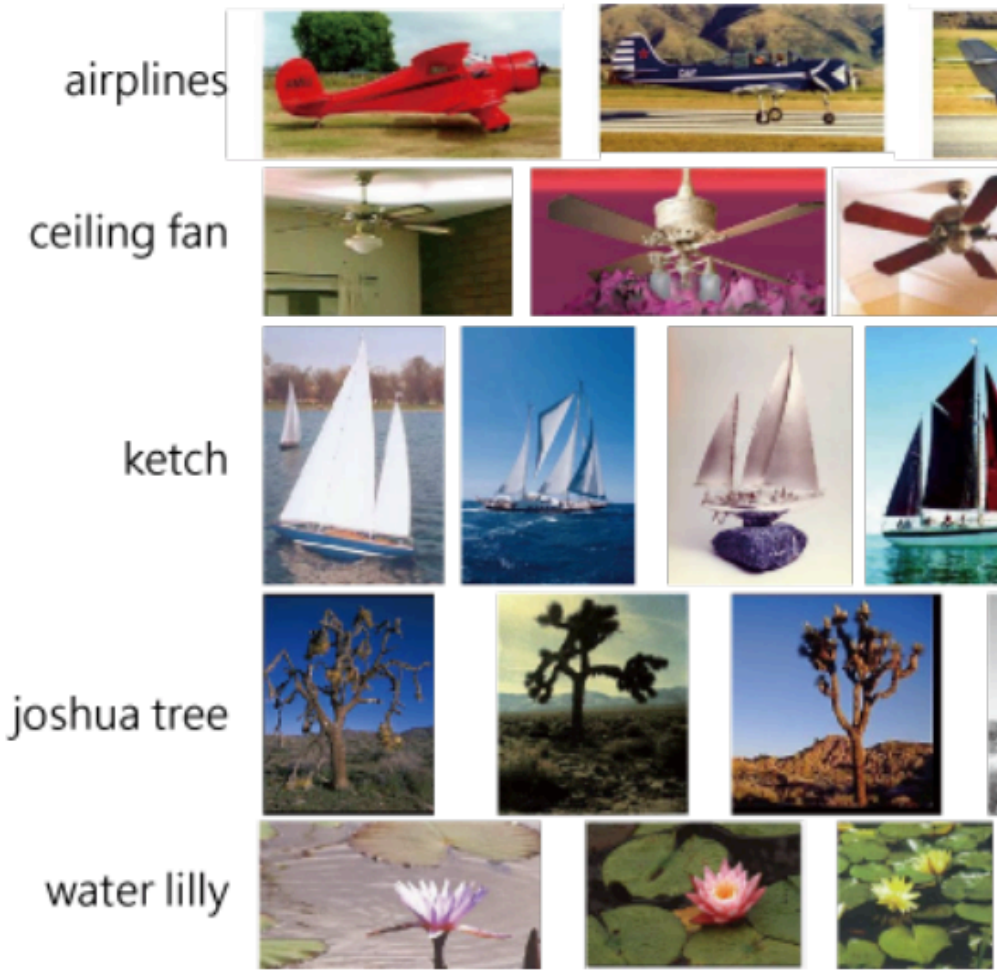
- Neural network architecture engineering and regularization is important. But still be careful of overfitting.

  The Caltech 101 and ImageNet lesson

Two classification datasets in computer vision

Dataset	Caltech101	ImageNet
Year collected	2003 By Fei-Fei Li et al.	2010 By Fei-Fei Li et al.
#classes	101	1000
#training image /class	15/30	~1000

Caltech 101 samples



ImageNet samples

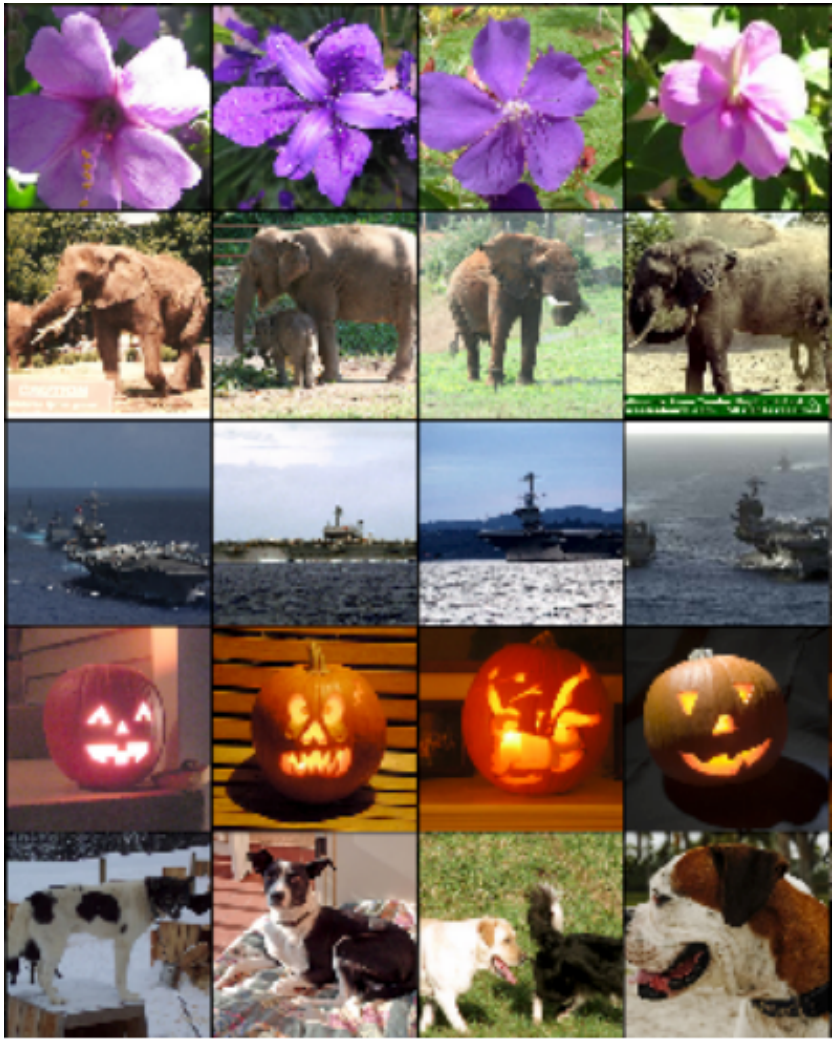


Image recognition before 2012

- Feature extraction and classification are mostly done separately.

- Does the CNN exist?
  Yes. LeCun et al had successful CNN for digit recognition in 1989.

- Does CNN work on Caltech 101?
  No.

A glimpse on the best model on Caltech101 in 2010

Ashish et al. 2010

Table 2 Recognition accuracy on the Caltech-101 using 15 labeled points per class

Method	GP 1-vs-All	SVM 1-vs-All	1-NN
Dense PMK	52.13 ± 0.69	48.77 ± 0.95	24.20 ± 0.48
Spatial PMK	51.90 ± 0.78	54.26 ± 0.65	41.10 ± 0.78
GB	64.15 ± 0.76	65.87 ± 0.92	45.58 ± 0.79
GBDist	58.81 ± 0.58	65.91 ± 0.66	50.23 ± 0.66
Combination	73.95 ± 1.13	-NA-	

Table 3 Accuracy reported with kernel combination on the Caltech-101

Method	Accuracy
Ours (4 Kernels)	**73.95 ± 1.13**
Boiman et al. (2008)	72.80 ± 0.39
Varma and Ray (2007) (4 Kernels)	68.82 ± 1.00
Frome et al. (2007)	60.30 ± 0.70
Lin et al. (2007)	59.80 ± NA
Zhang et al. (2006)	59.08 ± 0.37
Kumar and Sminchisescu (2007)	57.83 ± NA

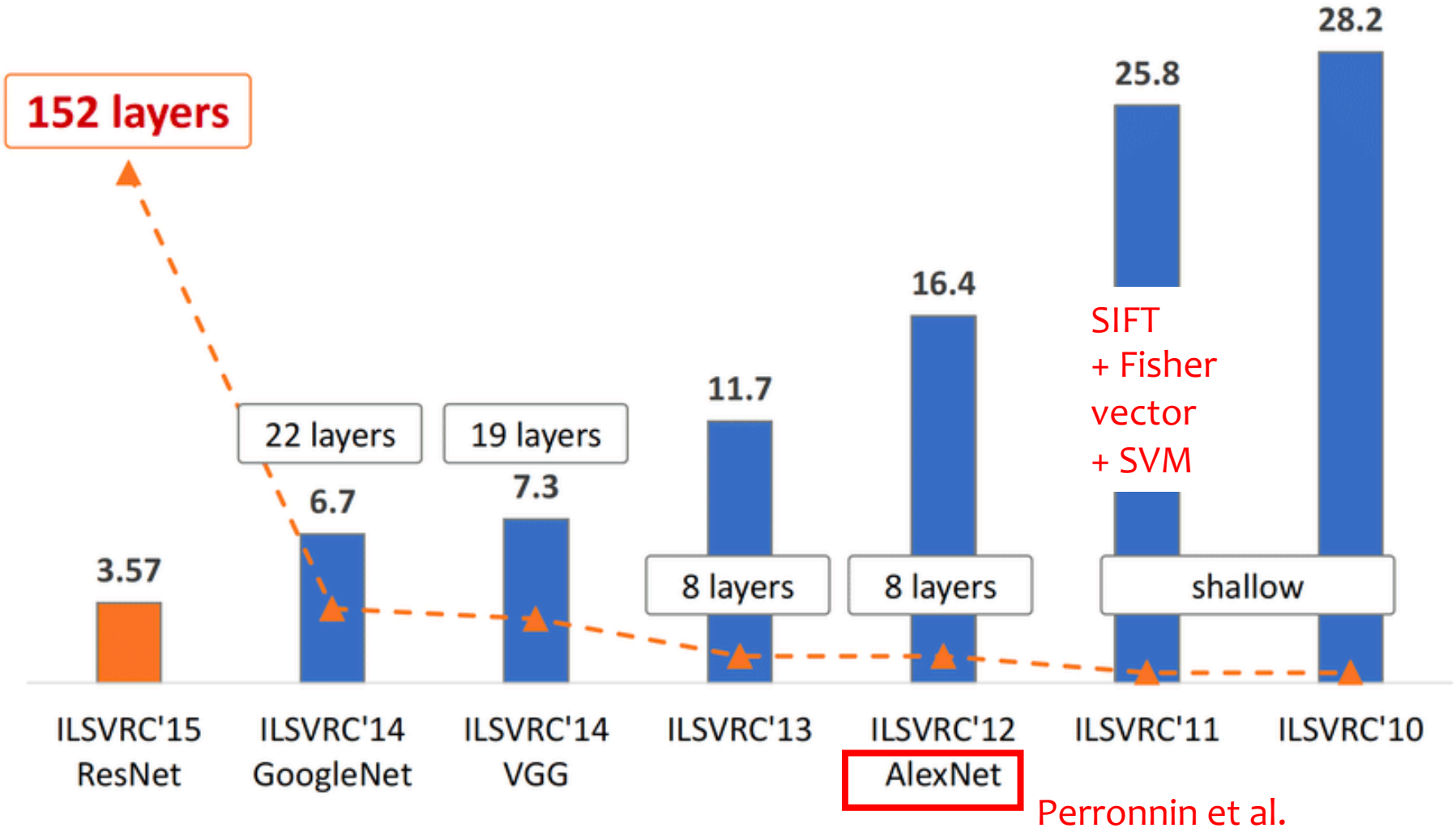
A glimpse on the best model on Caltech256 (variant of Caltech 101) in 2010

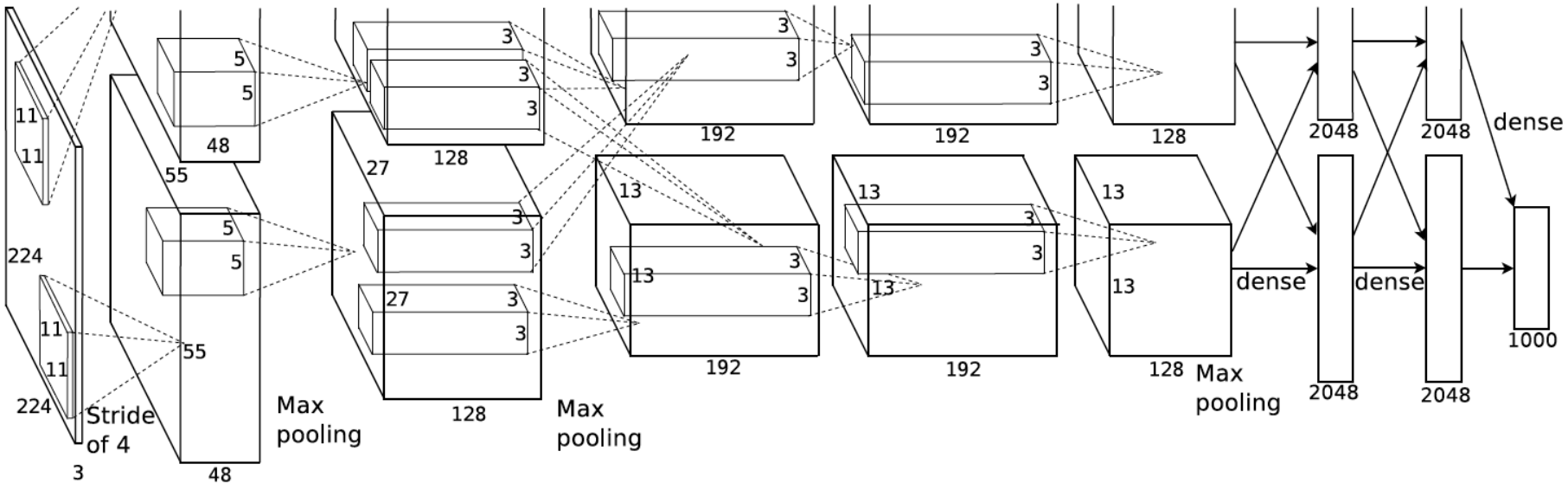
Perronnin et al. 2010 SIFT + Fisher vector + SVM

Table 3. Comparison of the proposed Improved Fisher Kernel (IFK) with the state-of-the-art on CalTech 256. Please see the text for details about each of the systems.

Method	ntrain=15	ntrain=30	ntrain=45	ntrain=60
Kernel Codebook [3]	-	27.2 (0.4)	-	-
EMK (SIFT) [20]	23.2 (0.6)	30.5 (0.4)	34.4 (0.4)	37.6 (0.5)
Standard FK (SIFT) [22]	25.6 (0.6)	29.0 (0.5)	34.9 (0.2)	38.5 (0.5)
Sparse Coding (SIFT) [18]	27.7 (0.5)	34.0 (0.4)	37.5 (0.6)	40.1 (0.9)
Baseline (SIFT) [26]	-	34.1 (0.2)	-	-
NN (SIFT) [19]	-	38.0 (-)	-	-
NN [19]	-	42.7 (-)	-	-
IFK (SIFT)	34.7 (0.2)	40.8 (0.1)	45.0 (0.2)	47.9 (0.4)

ImageNet challenge classification error across years





AlexNet architecture (Krizhevsky et al. 2012)
5 conv layers + 3 dense layers

Summary

- The strength and weakness of deep neural networks
  - Very powerful method
  - Highly engineerable
  - Hard to optimize (a lot of heuristics)
  - Need large samples

- Further readings
  - Chapter 11 in ESL book
  - Stanford cs231n lecture notes
  - Popular packages and the official tutorials: Keras, tensorflow, Pytorch