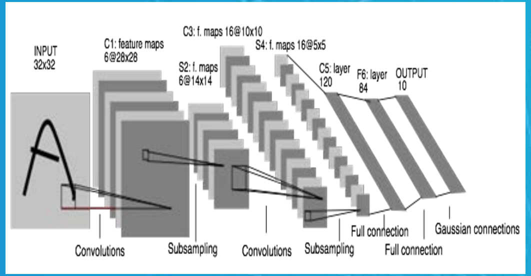
Carnegie Mellon University
School of Computer Science

10701: Introduction to Machine Learning

Neural Networks and Deep Learning (1)

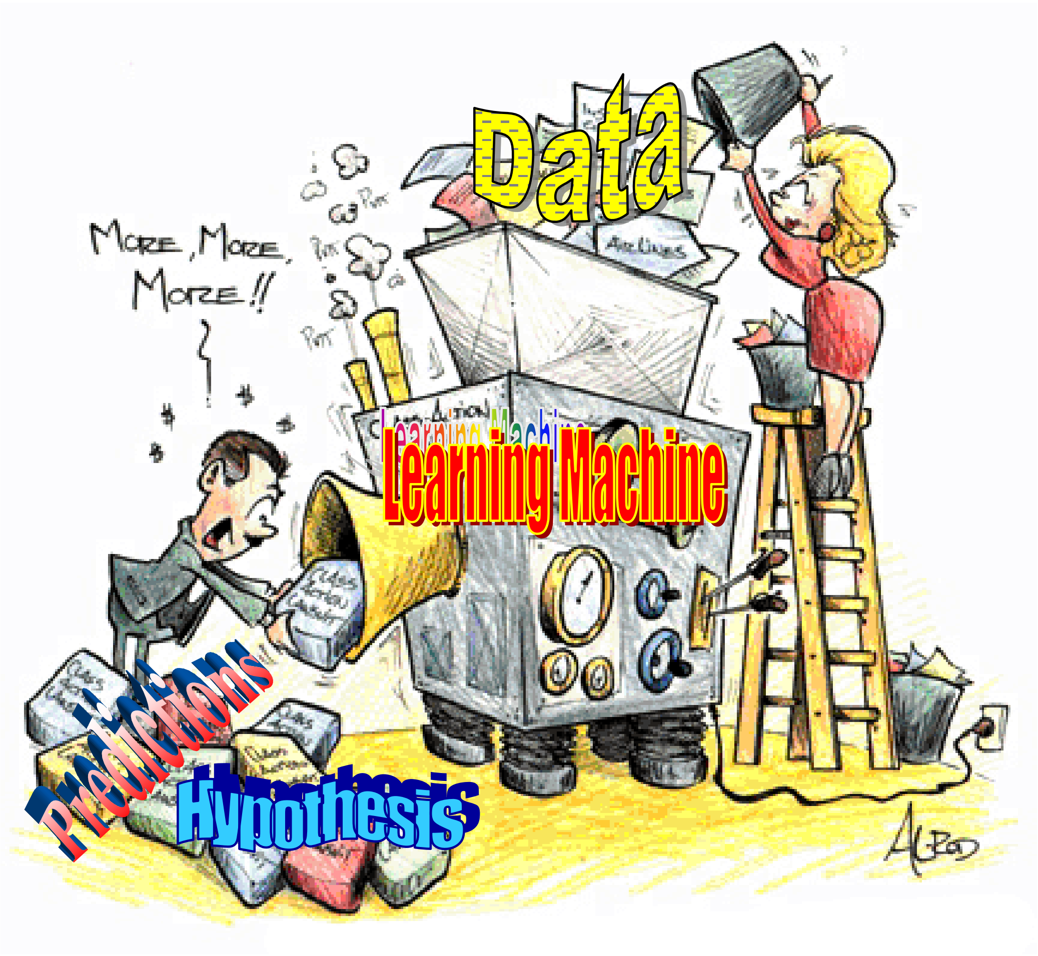
- Basics in artificial neural networks

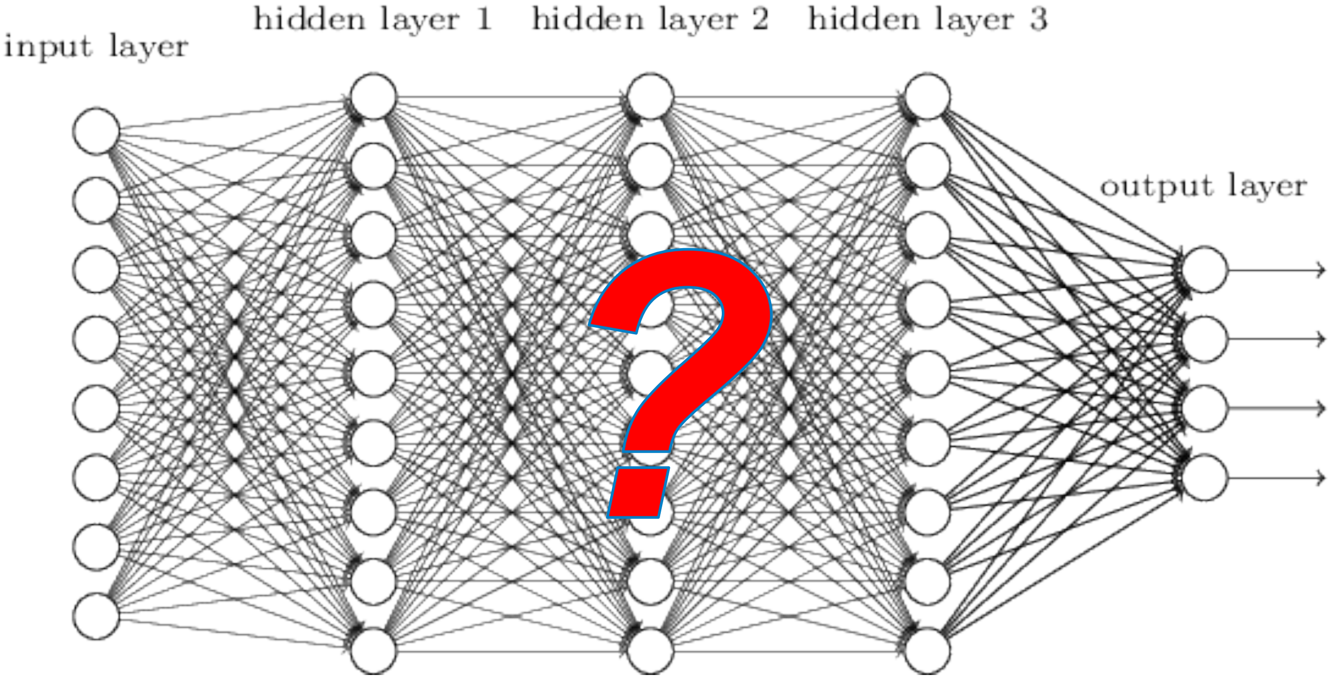


Eric Xing
Lecture 10, October 7, 2020

Reading: see class homepage

ML vs DL


Data
More, More, More!!
Learning Machine
Predictions
Hypothesis


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer
?

Outline

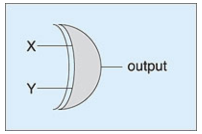
- An overview of DL components
  - Historical remarks: early days of neural networks
    - Perception
    - ANN
    - Reverse-mode automatic differentiation (aka backpropagation)
    - Pretrain
    - CNN
  - Modern building blocks: units, layers, activations functions, loss functions, etc. (next lecture)

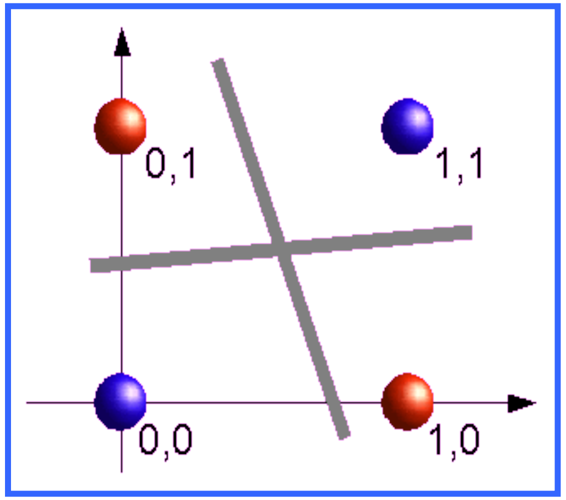
Learning highly non-linear functions

f: X → Y

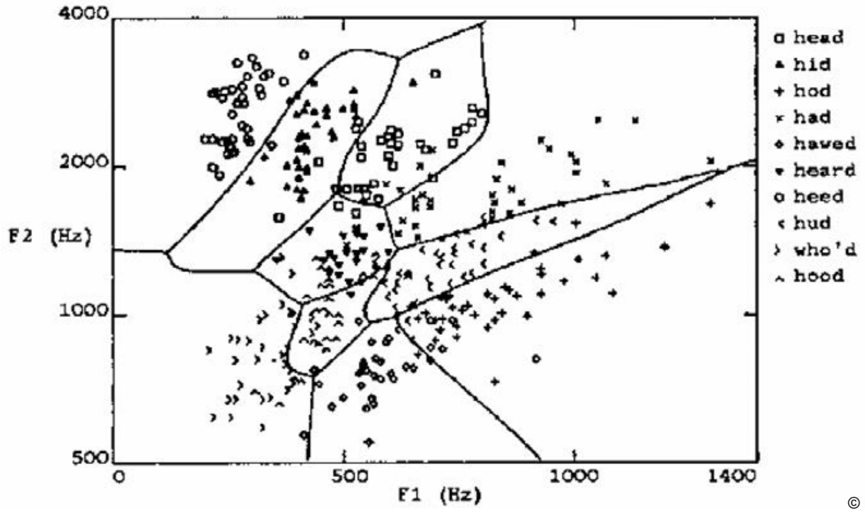
- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

The XOR gate



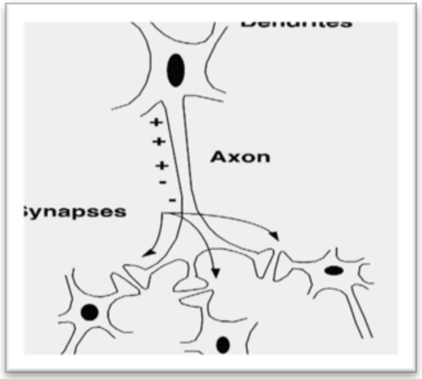


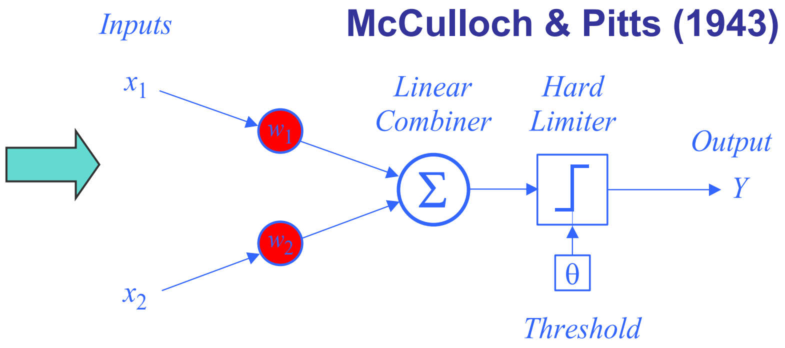
Speech recognition



Perceptron and Neural Nets

- From biological neuron to artificial neuron (perceptron)

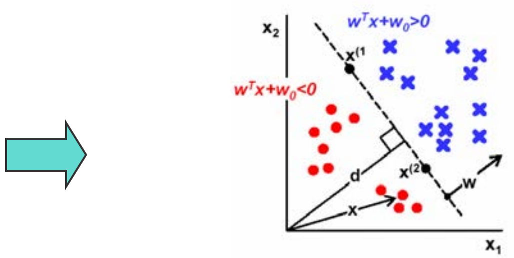




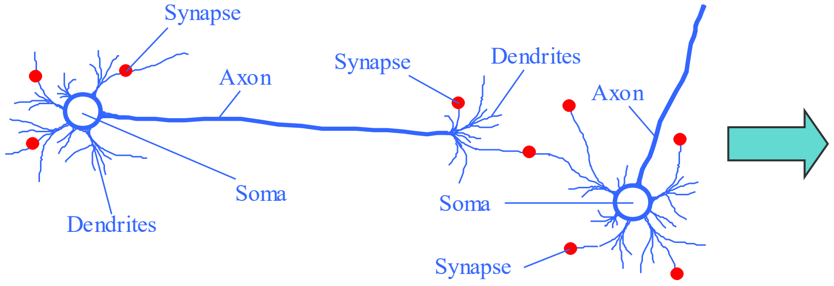
- Activation function

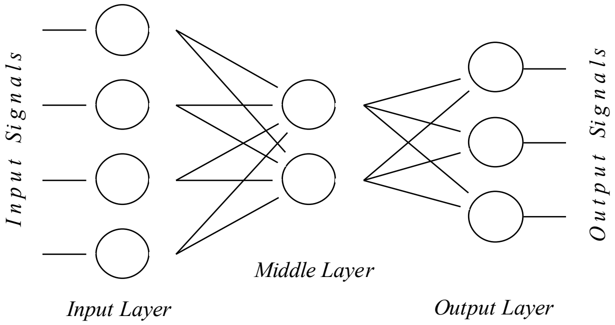
$$X = \sum_{i=1}^{n} x_i w_i$$

$$Y = \begin{cases} +1, & \text{if } X \ge \omega_0 \\ -1, & \text{if } X < \omega_0 \end{cases}$$



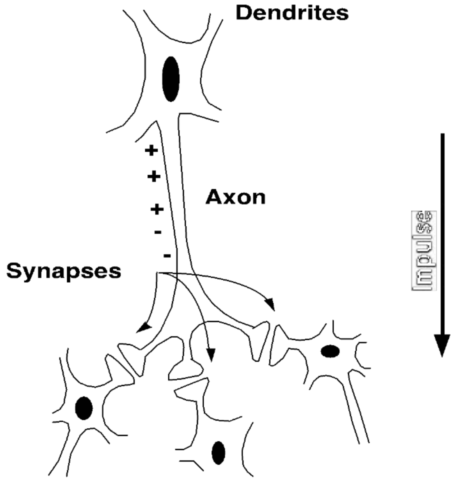
- From biological neuron network to artificial neuron networks

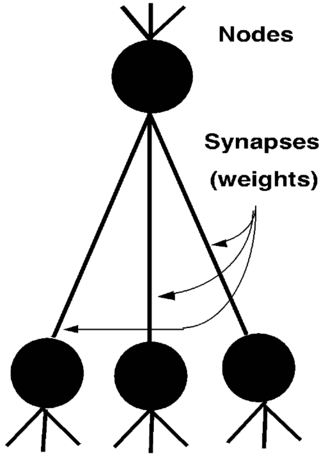




Connectionist Models

- Consider humans:
  - Neuron switching time
    ~ 0.001 second
  - Number of neurons
    ~ 10^{10}
  - Connections per neuron
    ~ 10^{4-5}
  - Scene recognition time
    ~ 0.1 second
  - 100 inference steps doesn't seem like enough
    → much parallel computation
- Properties of artificial neural nets (ANN)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed processes

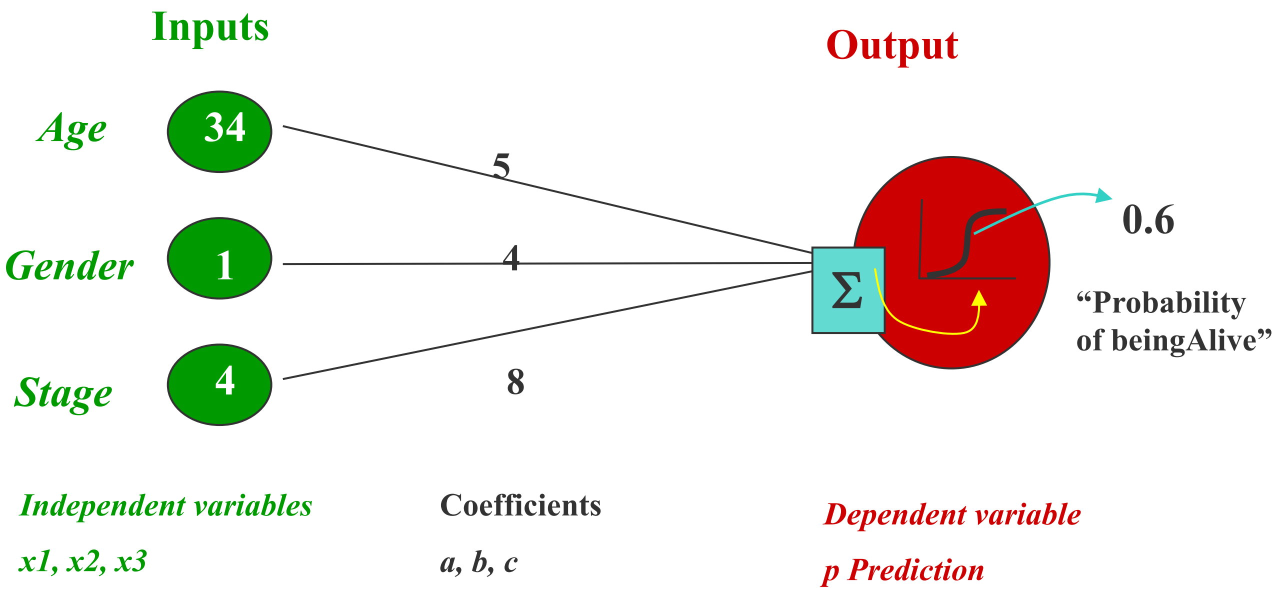




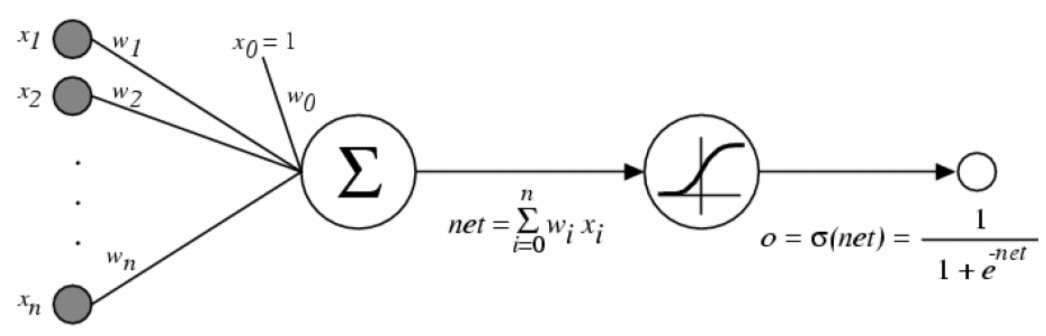
Jargon Pseudo-Correspondence

- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”
- Estimates = “targets”

Logistic Regression Model (the sigmoid unit)



The perceptron learning algorithm



- Recall the nice property of sigmoid function
- Consider regression problem f: X→Y, for scalar Y:

$$\frac{d\sigma}{dt} = \sigma(1-\sigma)$$

- We used to maximize the conditional data likelihood

$$y = f(x) + \epsilon$$

$$\vec{w} = \arg\max_{\vec{w}} \ln \prod_i P(y_i | x_i; \vec{w})$$

- Here …

$$\vec{w} = \arg\min_{\vec{w}} \sum_i \frac{1}{2}(y_i - \hat{f}(x_i; \vec{w}))^2$$

The perceptron learning algorithm

$$
\begin{aligned}
\frac{\partial E_D[\vec{w}])}{\partial w_j} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d}(t_d - o_d)\Big(-\frac{\partial o_d}{\partial w_i}\Big) \\
&= -\sum_{d}(t_d - o_d)\frac{\partial o_d}{\partial net_i}\frac{\partial net_d}{\partial w_i} \\
&= -\sum_{d}(t_d - o_d)o_d(1 - o_d)x_d^i
\end{aligned}
$$

x_d = input

t_d = target output

o_d = observed output

w_i = weight i

Batch mode:

Do until converge:

1. compute gradient $\nabla E_D[w]$
2. $\vec{w} = \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode:

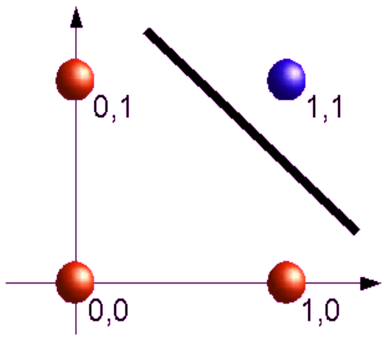
Do until converge:

- For each training example *d* in *D*
  1. compute gradient $\nabla E_d[w]$
  2. $\vec{w} = \vec{w} - \eta \nabla E_d[\vec{w}]$

  where

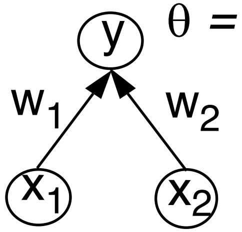
  $\nabla E_d[\vec{w}] = -(t_d - o_d)o_d(1 - o_d)\vec{x}_d$

What decision surface does a perceptron define?



NAND

x	y	Z (color)
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**



$\theta = 0.5$

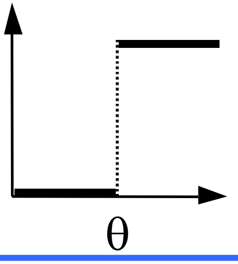
$f(x_1 w_1 + x_2 w_2) = y$

$f(0w_1 + 0w_2) = 1$

$f(0w_1 + 1w_2) = 1$

$f(1w_1 + 0w_2) = 1$

$f(1w_1 + 1w_2) = 0$

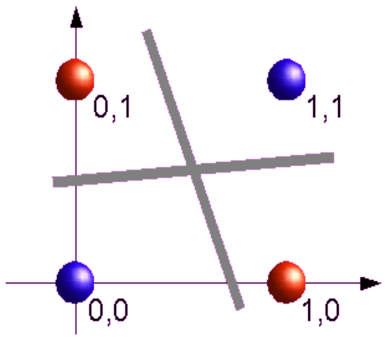


$$f(a) = \begin{cases} 1, \text{ for } a > \theta \\ 0, \text{ for } a \leq \theta \end{cases}$$

some possible values for w_1 and w_2

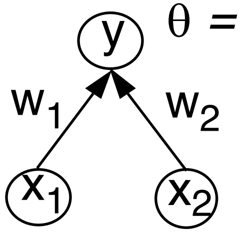
w_1	w_2
0.20	0.35
0.20	0.40
0.25	0.30
0.40	0.20

What decision surface does a perceptron define?



NAND

x	y	Z (color)
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**



$\theta = 0.5$

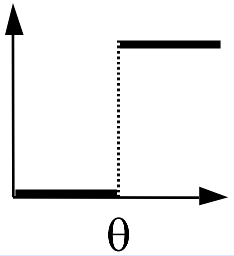
$$f(x_1 w_1 + x_2 w_2) = y$$

$$f(0w_1 + 0w_2) = 0$$

$$f(0w_1 + 1w_2) = 1$$

$$f(1w_1 + 0w_2) = 1$$

$$f(1w_1 + 1w_2) = 0$$

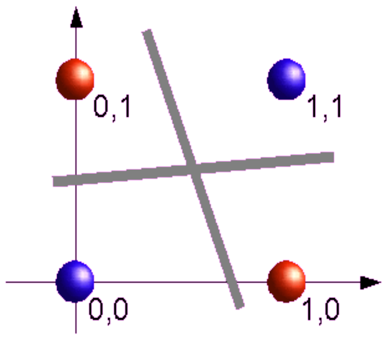


$$f(a) = \begin{cases} 1, \text{ for } a > \theta \\ 0, \text{ for } a \leq \theta \end{cases}$$

some possible values for w_1 and w_2

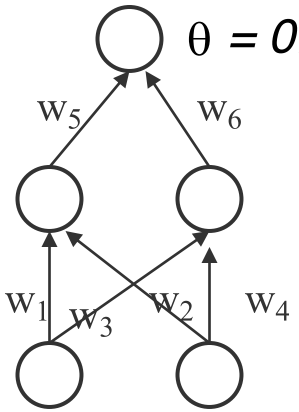
w_1	w_2

What decision surface does a perceptron define?

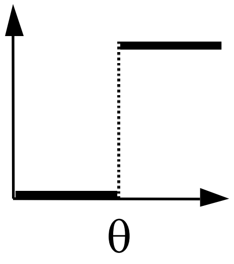


NAND

x	y	Z (color)
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**



θ *= 0.5 for all units*



$$f(a) = \begin{cases} 1, \text{ for } a > \theta \\ 0, \text{ for } a \le \theta \end{cases}$$

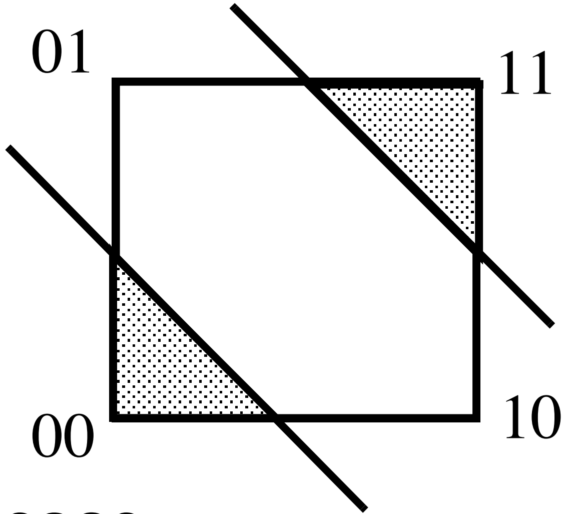
a possible set of values for $(w_1, w_2, w_3, w_4, w_5, w_6)$:

$(0.6,-0.6,-0.7,0.8,1,1)$

Non Linear Separation

Meningitis
No cough
Headache

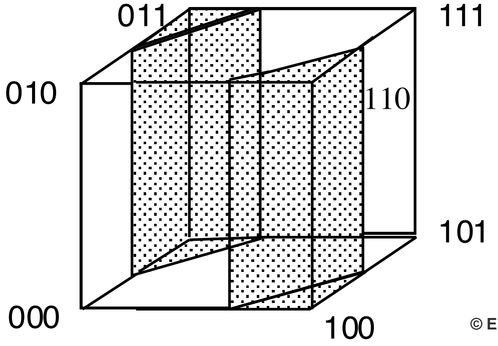
Flu
Cough
Headache



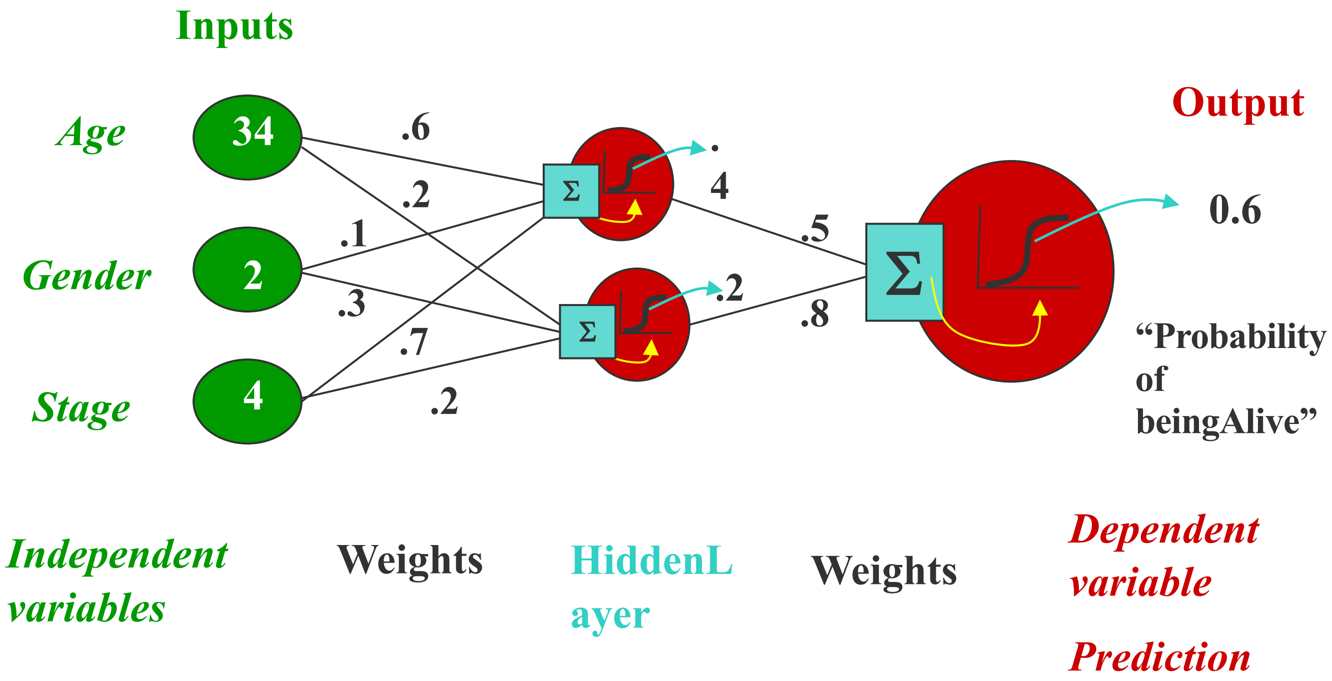
▨ **No treatment**
☐ **Treatment**

No disease
No cough
No headache

Pneumonia
Cough
No headache



Neural Network Model



Inputs

Age 34

Gender 2

Stage 4

Output

0.6

"Probability of beingAlive"

Independent variables

Weights

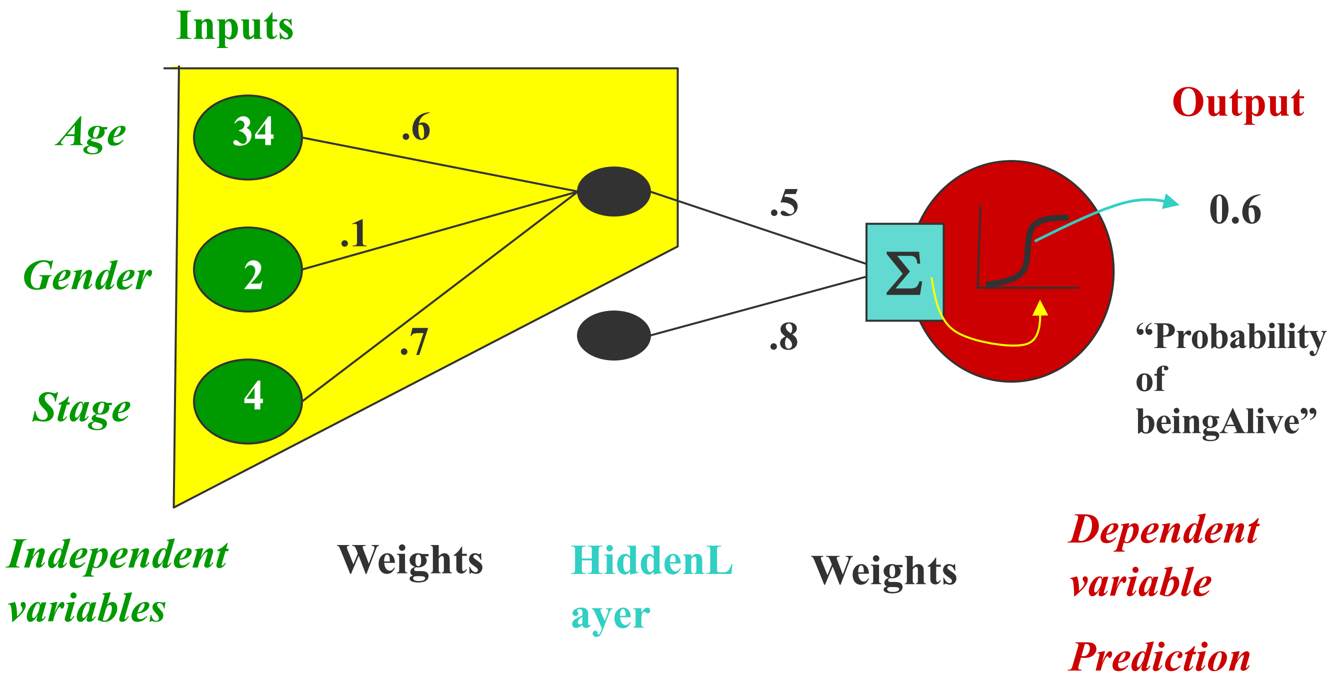
HiddenLayer

Weights

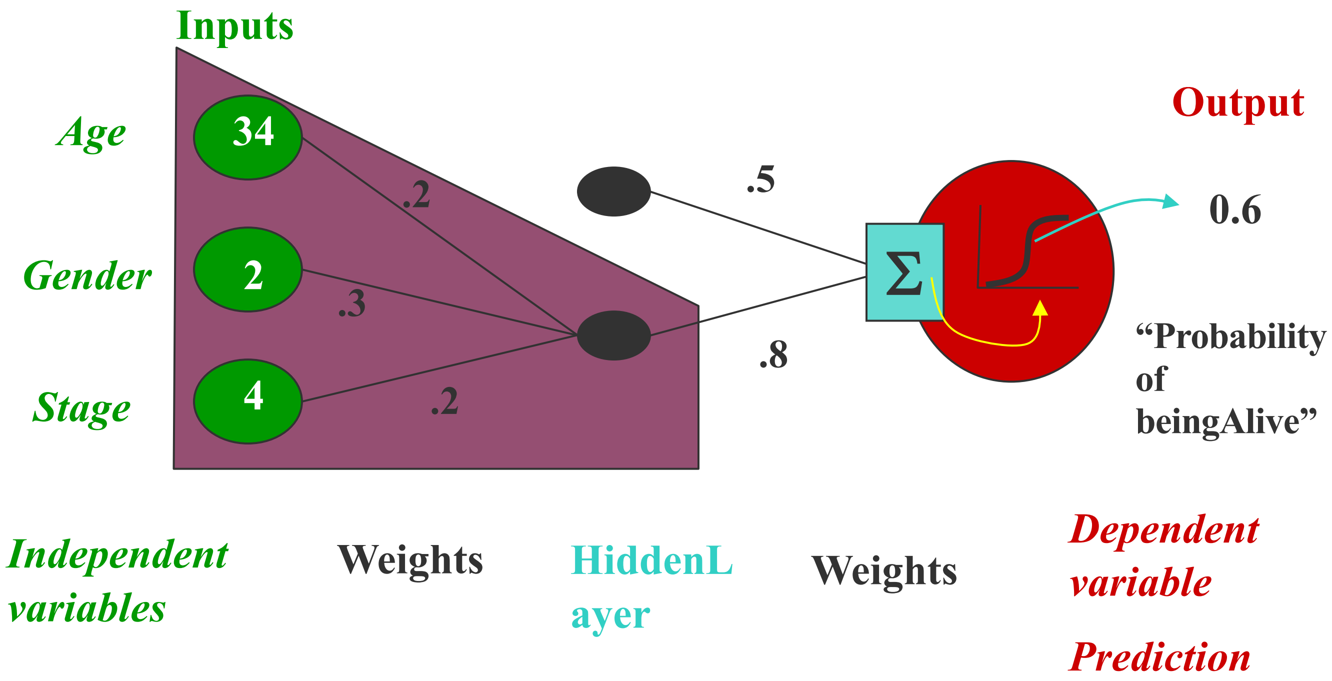
Dependent variable

Prediction

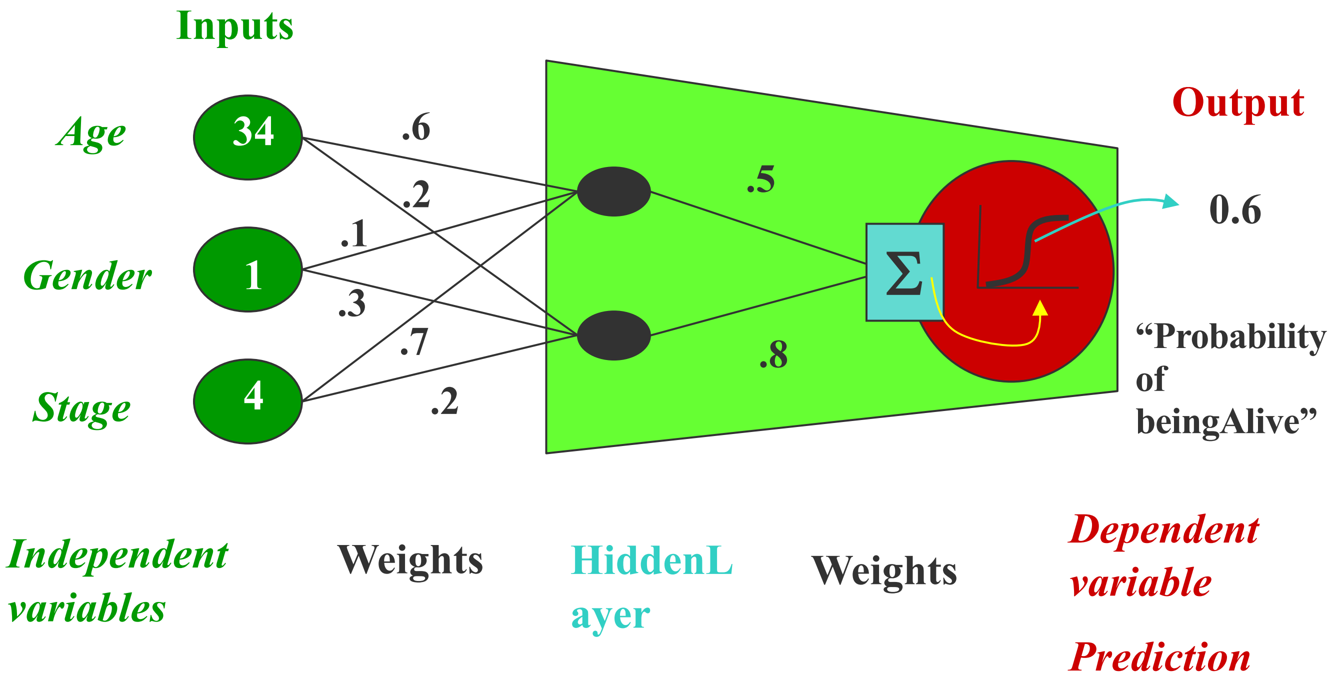
"Combined logistic models"



“Combined logistic models”



"Combined logistic models"



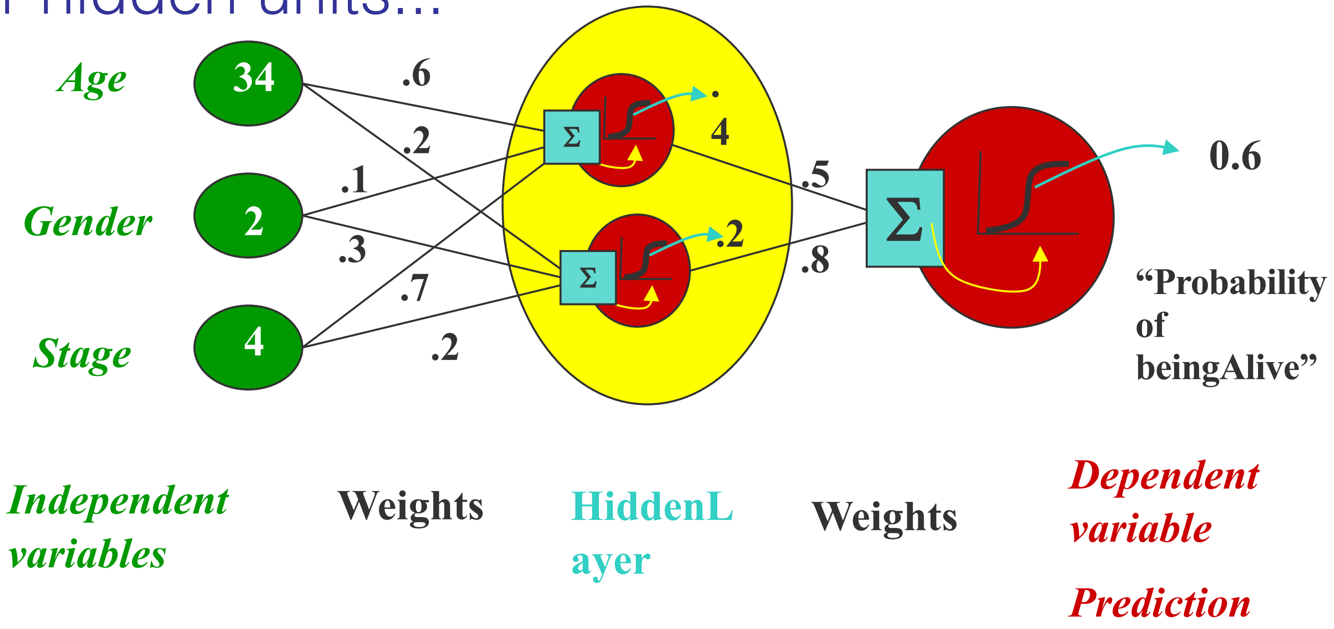
Neural Network Training

- Gradient descent
- No target for hidden units...

$$\frac{\partial L}{\partial w_{ij}}$$

← Training loss

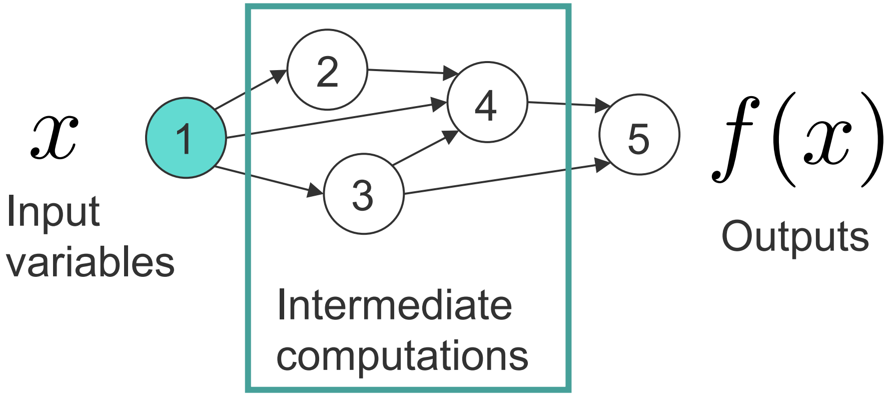
← Weight between unit i and j



- Back-Propagation (BP)
  - A routine to compute gradient
  - Use chain rule of derivative

Backpropagation: Reverse-mode differentiation

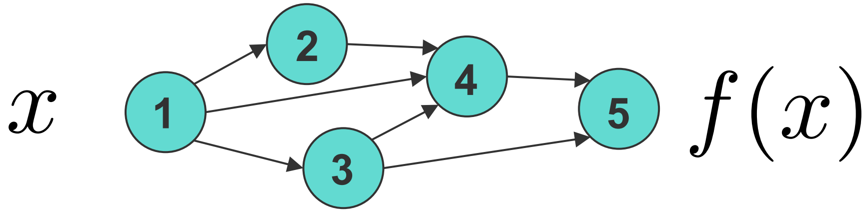
- Artificial neural networks are nothing more than complex functional compositions that can be represented by **computation graphs**:



$$\frac{\partial f_n}{\partial x} =$$

Backpropagation: Reverse-mode differentiation

- Artificial neural networks are nothing more than complex functional compositions that can be represented by **computation graphs**:



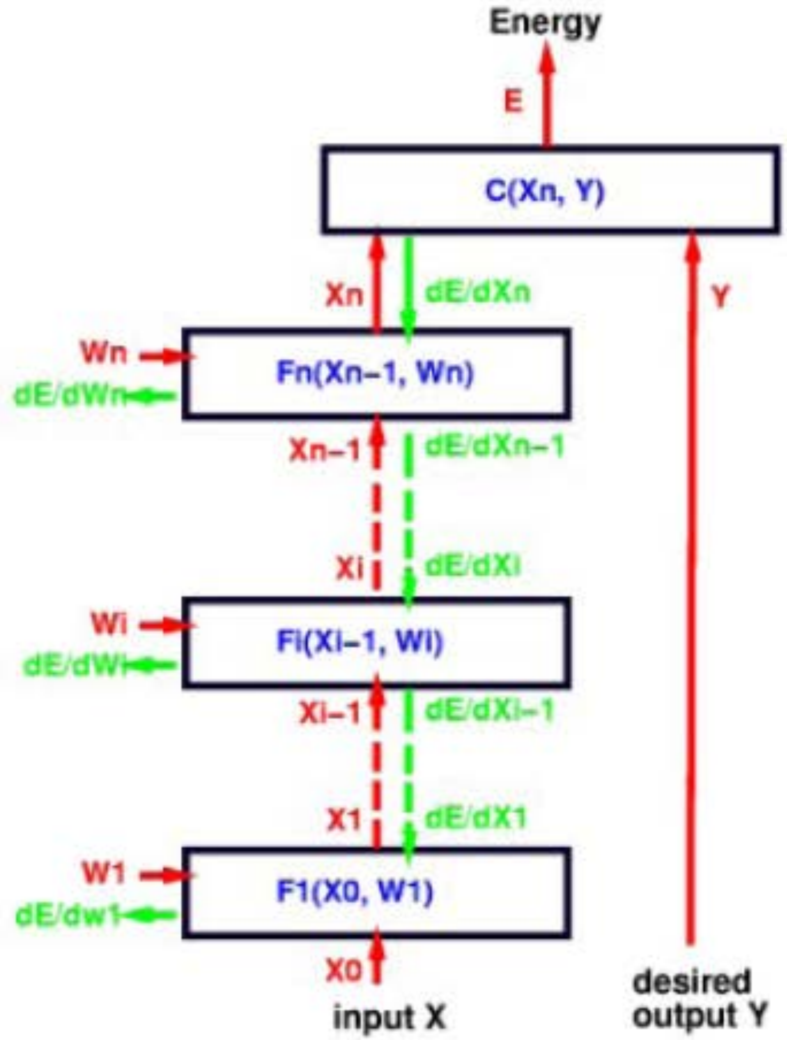
$$\frac{\partial f_n}{\partial x} =$$

- By applying the chain rule and using reverse accumulation, we get

$$\frac{\partial f_n}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \frac{\partial f_{i_1}}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \sum_{i_2 \in \pi(i_1)} \frac{\partial f_{i_1}}{\partial f_{i_2}} \frac{\partial f_{i_1}}{\partial x} = \dots$$

- The algorithm is commonly known as backpropagation
- What if some of the functions are stochastic?
- Then use stochastic backpropagation!
  (to be covered in the next part)
- Modern packages can do this *automatically* (more later)

Backpropagation (continue)



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \boxed{\frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}}$
- $\boxed{\frac{\partial E}{\partial W_n} = \boxed{\frac{\partial E}{\partial X_n}} \boxed{\frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}}}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

$\frac{\partial \sigma}{\partial X_{n-1}} \sigma(1-\sigma) W_n$

Say: $F_n = \sigma$

$\frac{\partial \sigma}{\partial W_n} \sigma(1-\sigma) x_{n-1}$

x_d = input
t_d = target output
o_d = observed output
w_i = weight i

Backpropagation (continue)

- Say, $E = (t - o)^2$, $\boldsymbol{F_n = \sigma}$

- Initialize all weights to small random numbers
  Until convergence, Do

1. Input the training example to the network and compute the network outputs

1. For each output unit k

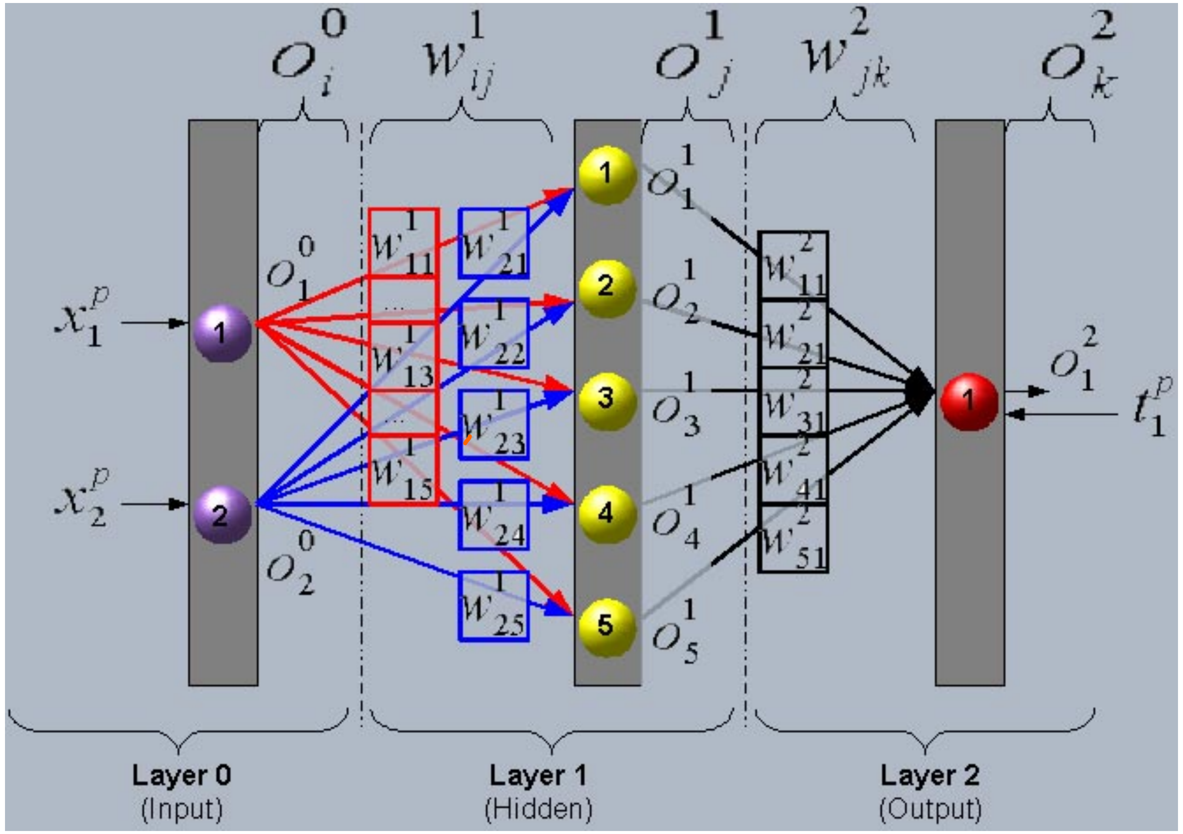
$$\delta_k \leftarrow o_k^2(1 - o_k^2)(t - o_k^2)$$

2. For each hidden unit h

$$\delta_h \leftarrow o_h^1(1 - o_h^1) \sum_{k \in outputs} w_{h,k}\delta_k$$

3. Undate each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \quad \text{where} \quad \Delta w_{i,j} = \eta \delta_j x^j$$



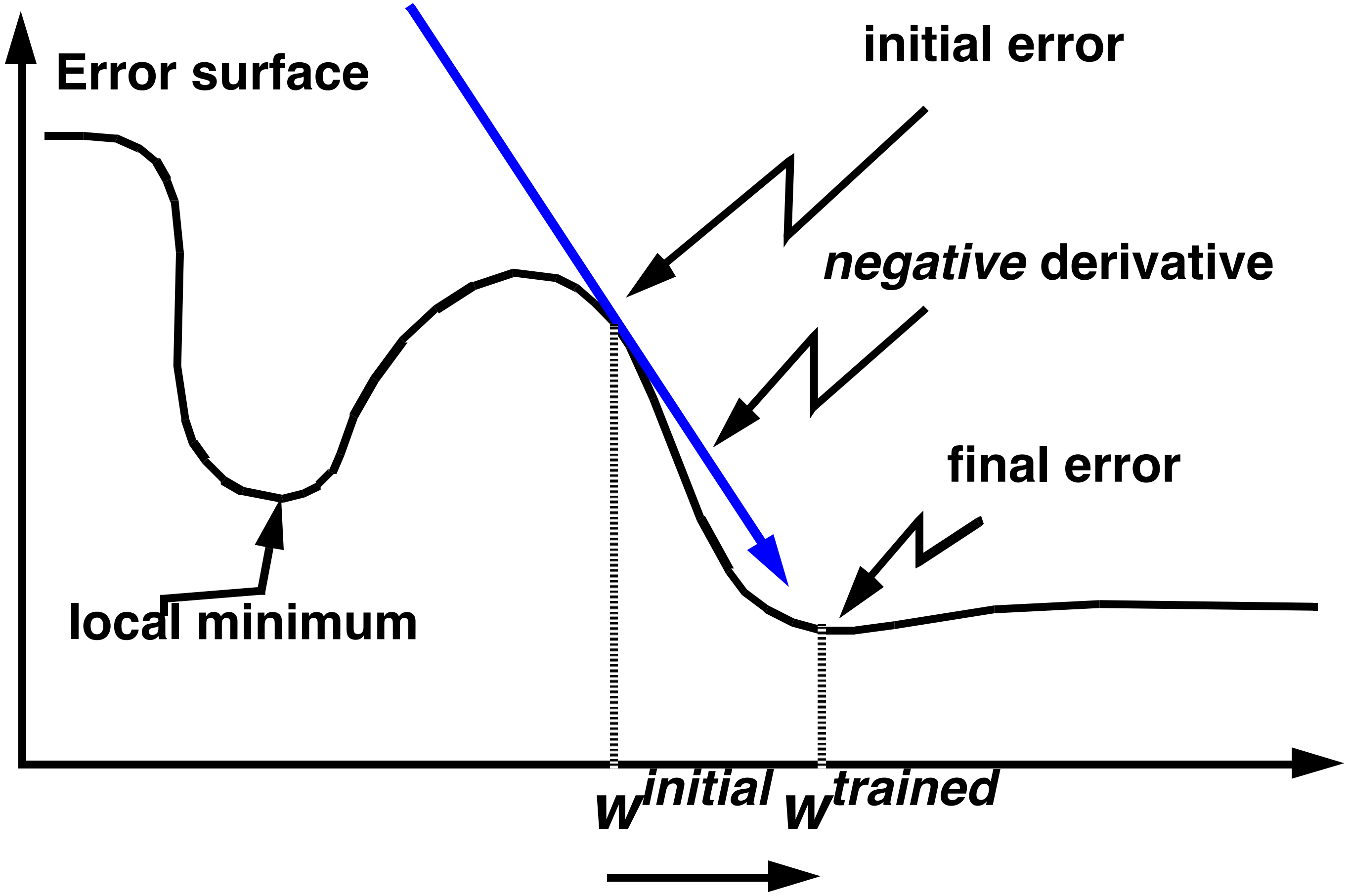
More on Backpropatation

- It is doing gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{i,j}(t) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(t-1)$$

- Minimizes error over *training* examples
  - Will it generalize well to subsequent testing examples?
- Training can take thousands of iterations, → very slow!
- Using network after training is very fast

Minimizing the Error



Error surface

initial error

negative derivative

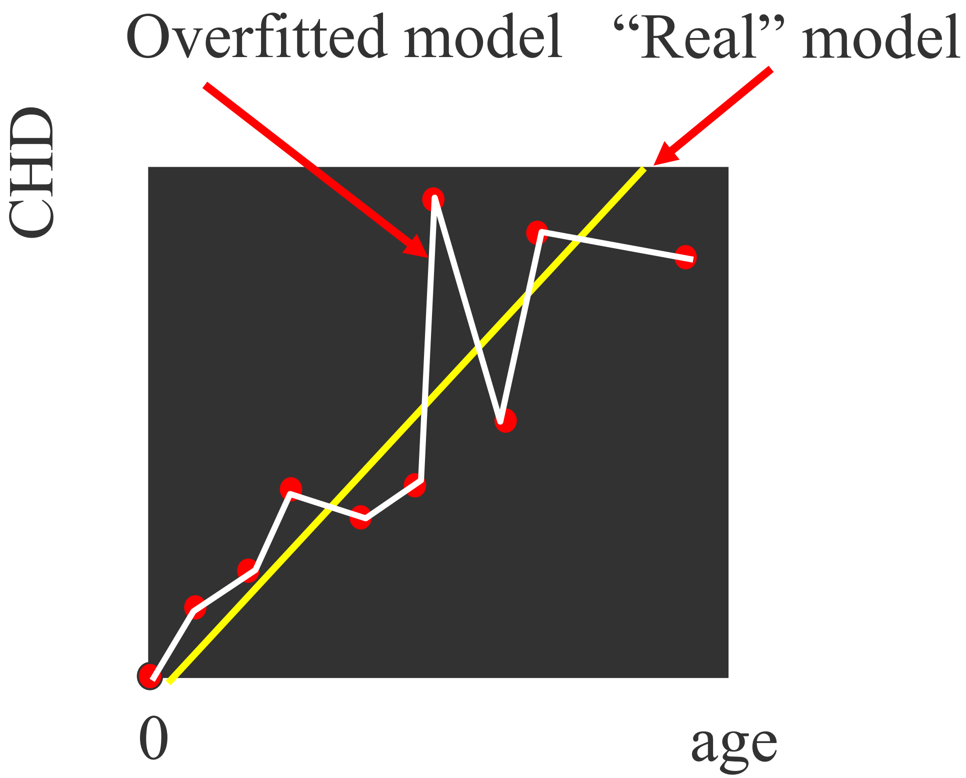
final error

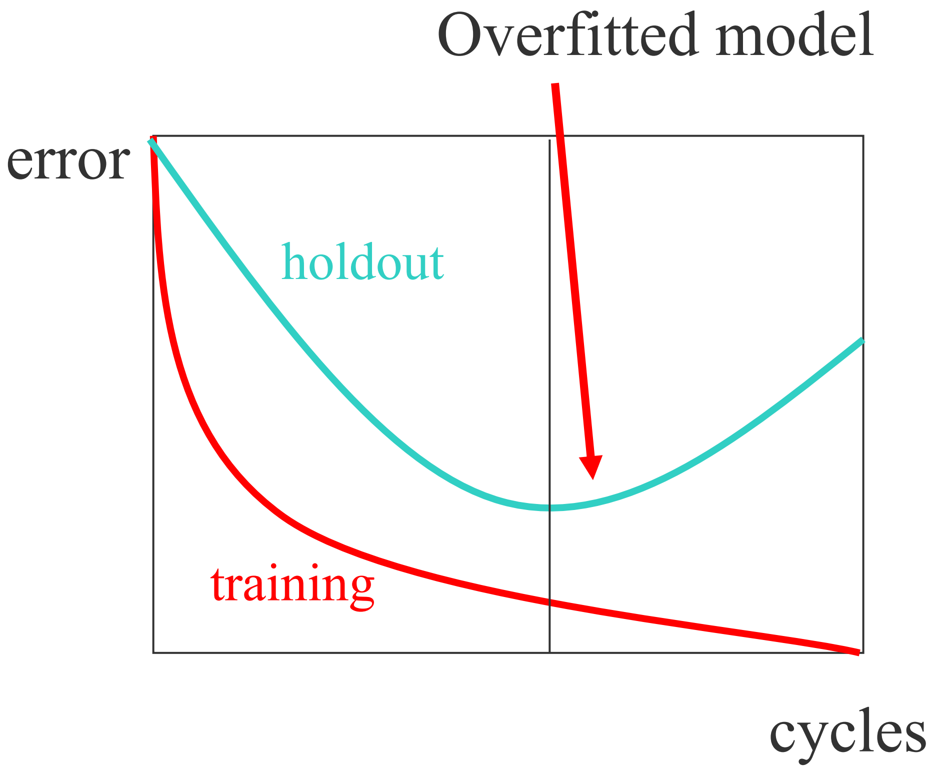
local minimum

$w^{initial}$ $w^{trained}$

positive change

Overfitting in Neural Nets





Alternative Error Functions

- Penalize large weights:

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{k,d} - o_{k,d})^2 + \gamma \sum_{i,j} w_{j,i}^2$$

- Training on target slopes as well as values

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{k,d} - o_{k,d})^2 + \mu \sum_{,j \in inputs} \left(\frac{\partial t_{k,d}}{\partial x_d^j} - \frac{\partial o_{k,d}}{\partial x_d^j}\right)$$

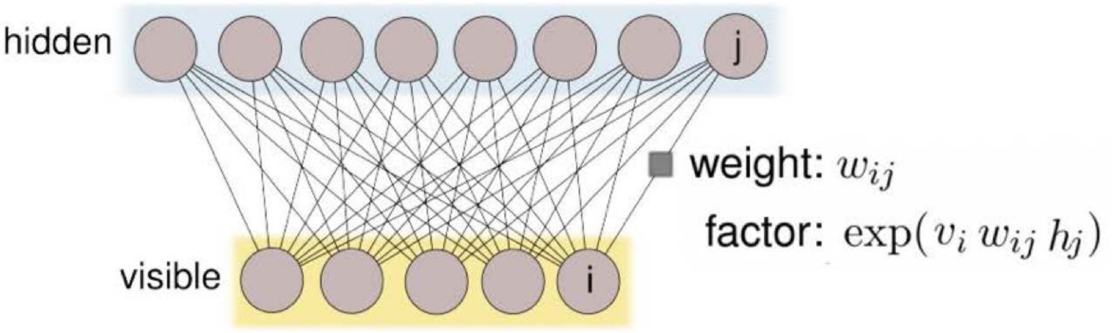
- Tie together weights

Pretraining

- A better initialization strategy of weight parameters
  - Based on Restricted Boltzmann Machine
  - An auto-encoder model
  - Unsupervised
  - Layer-wise, greedy
- Useful when training data is limited
- Not necessary when training data is rich

Restricted Boltzmann Machines

- RBM is a Markov random field represented with a bi-partite graph
- All nodes in one layer/part of the graph are connected to all in the other; no inter-layer connections



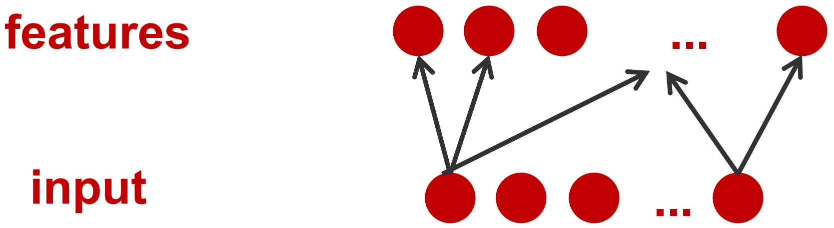
- Joint distribution:

$$P(v,h) = \frac{1}{Z}\exp\left\{\sum_{i,j} w_{ij} v_i h_i + \sum_i b_i v_i + \sum_j c_j h_j\right\}$$

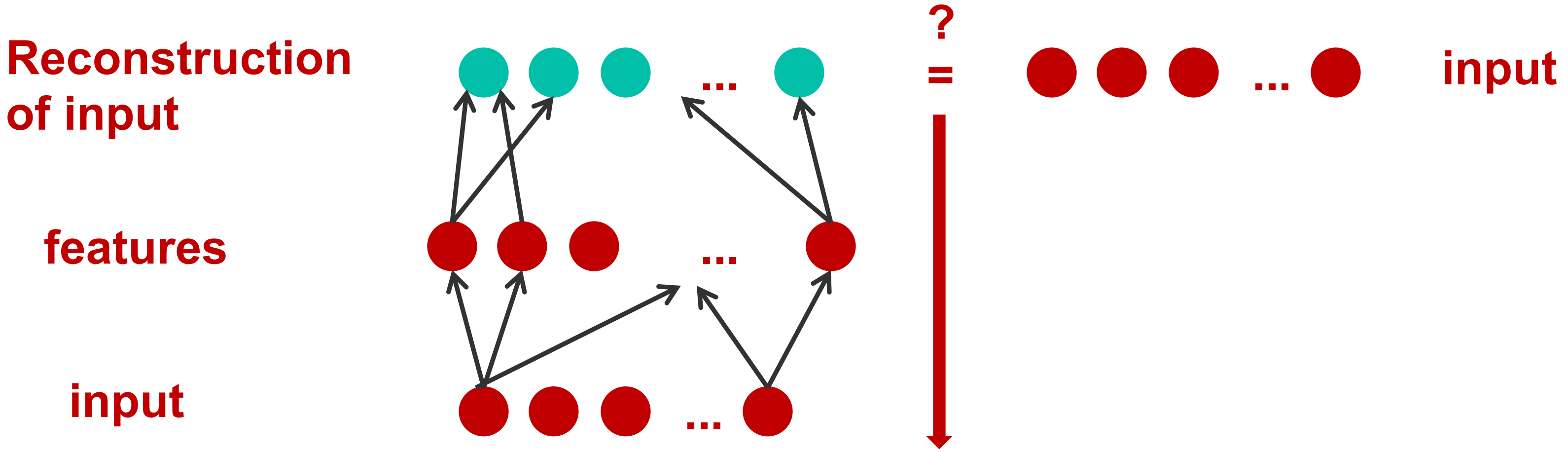
Images from Marcus Frean, MLSS Tutorial 2010

Layer-wise Unsupervised Pre-training

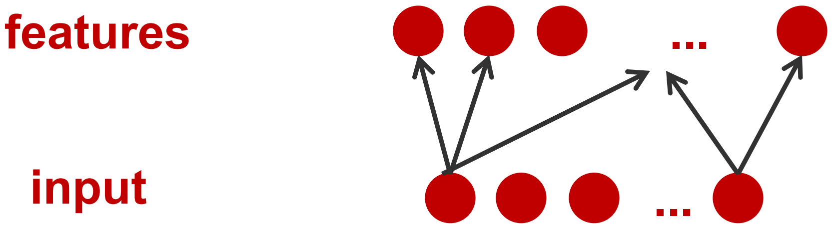
Layer-wise Unsupervised Pre-training



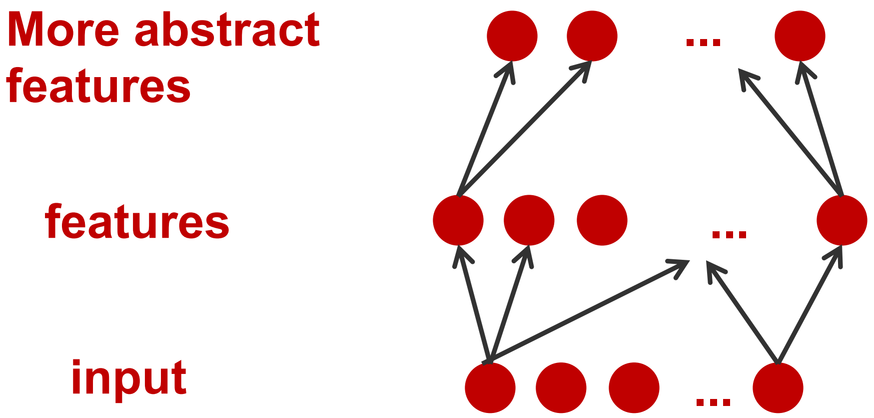
Layer-wise Unsupervised Pre-training



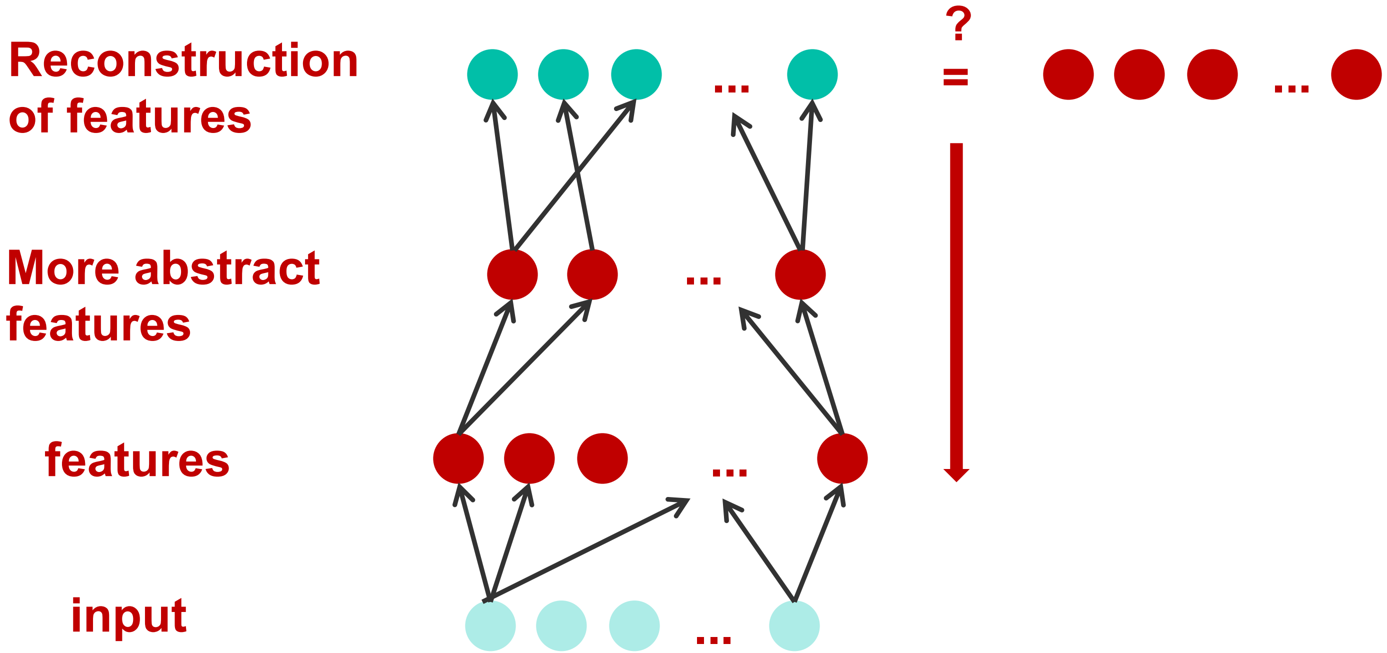
Layer-wise Unsupervised Pre-training



Layer-wise Unsupervised Pre-training



Layer-wise Unsupervised Pre-training



Reconstruction of features

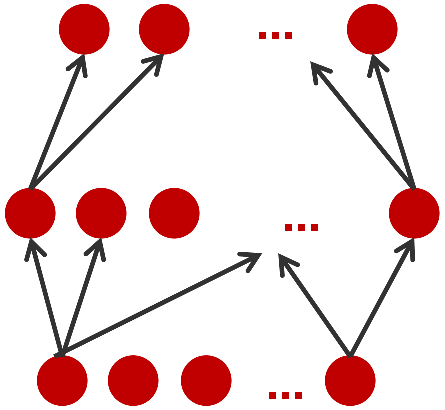
?
=

More abstract features

features

input

Layer-wise Unsupervised Pre-training

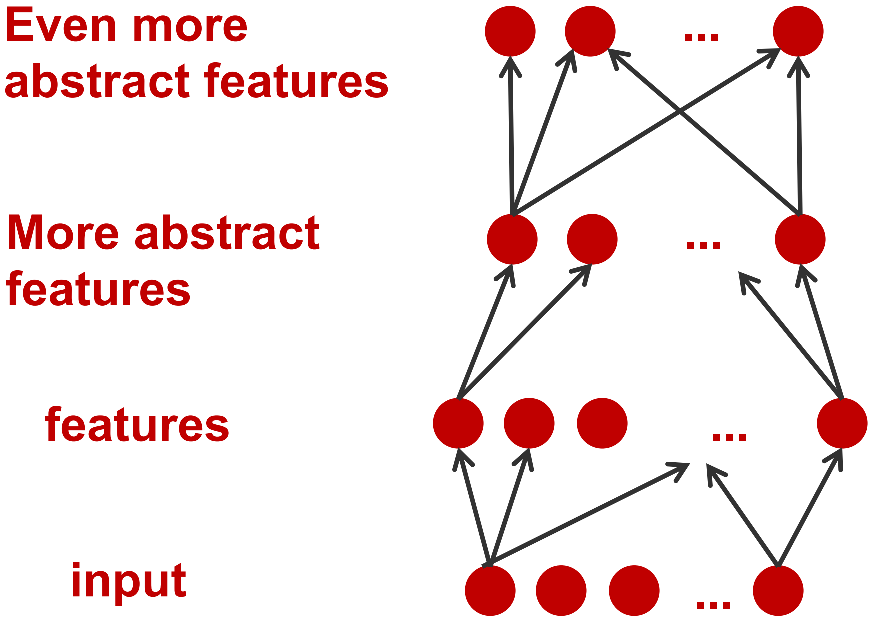


More abstract features

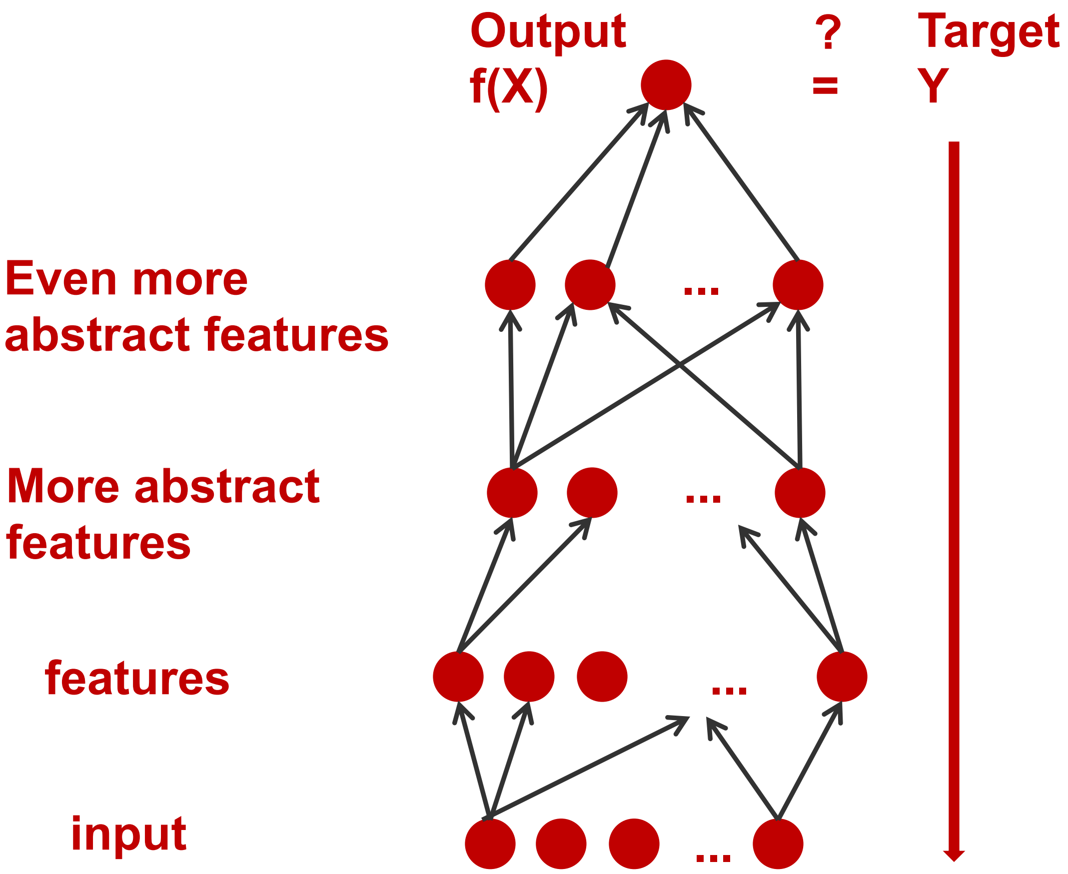
features

input

Layer-wise Unsupervised Pre-training

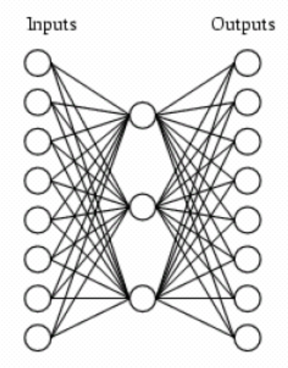


Layer-wise Unsupervised Pre-training

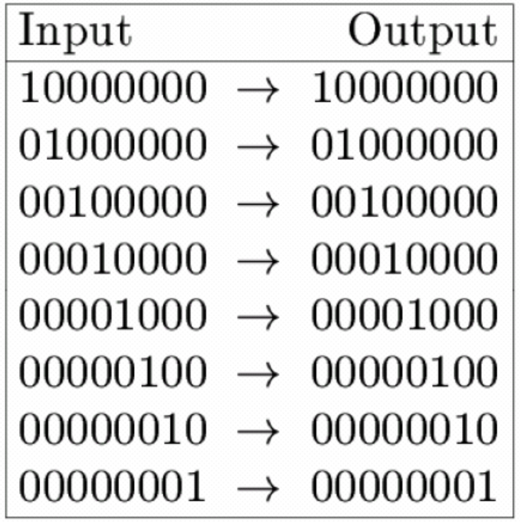


Learning Hidden Layer Representation

- A network:



- A target function:

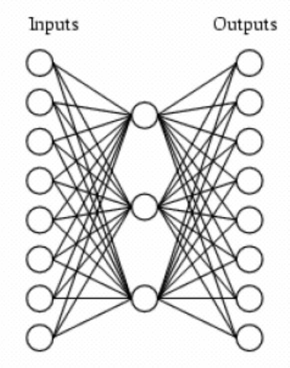


Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

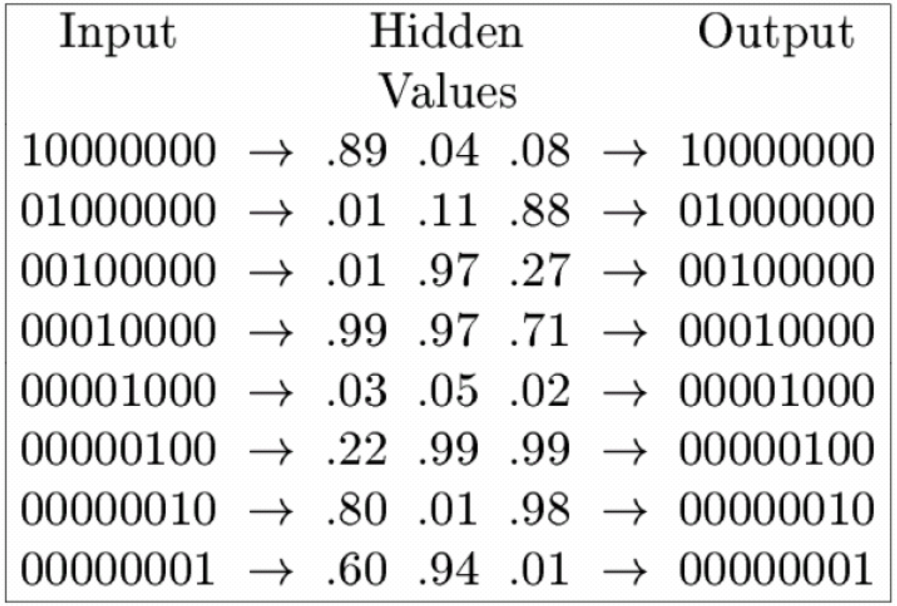
- Can this be learned?

Learning Hidden Layer Representation

- A network:

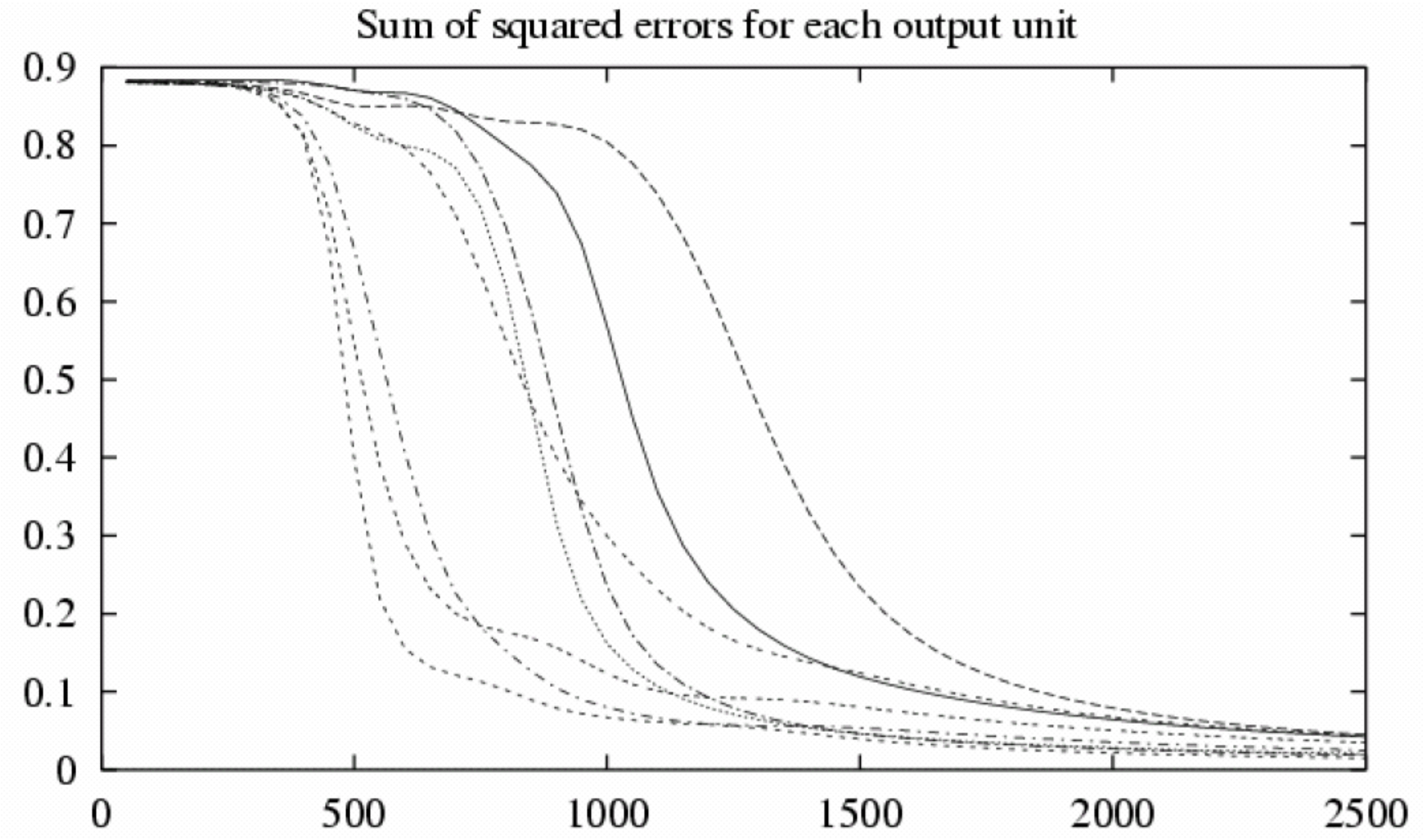


- Learned hidden layer representation:

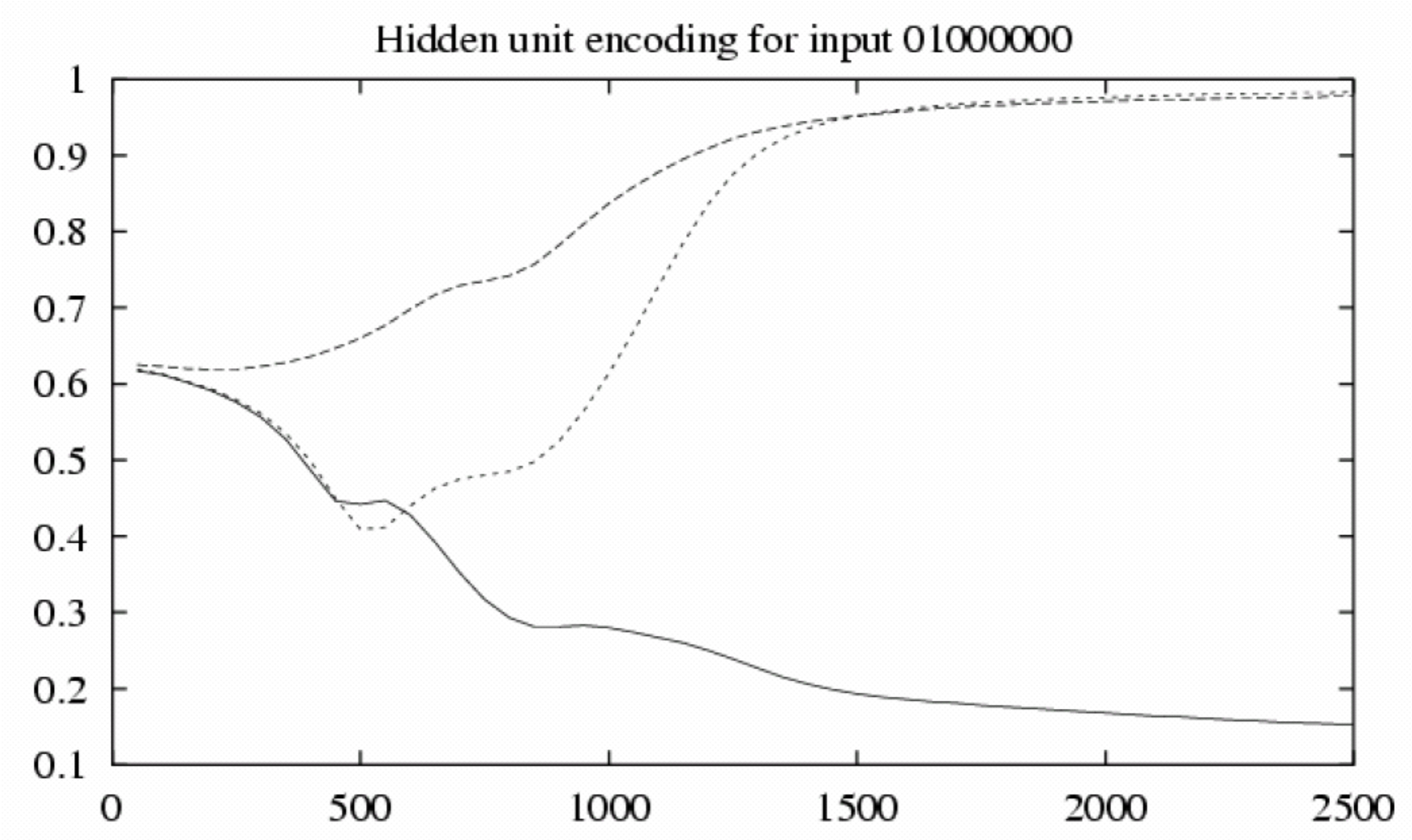


Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Training

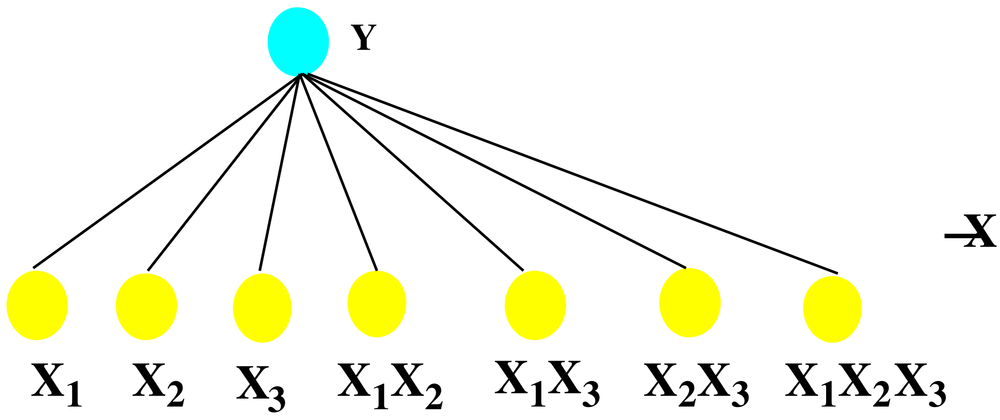

Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500

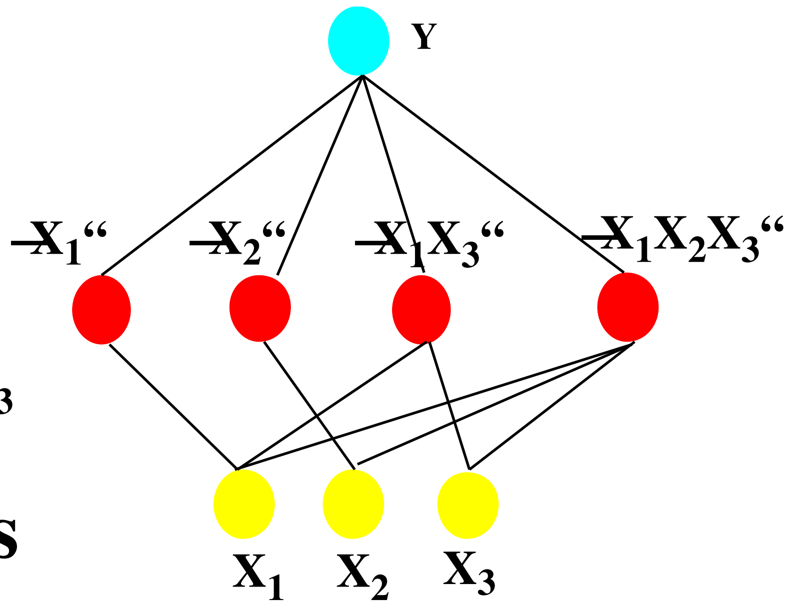
Training



Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

Non-linear LR vs. ANN





$(2^3\text{-}1)$ possible combinations

$$Y = a(X_1) + b(X_2) + c(X_3) + d(X_1X_2) + \ldots$$

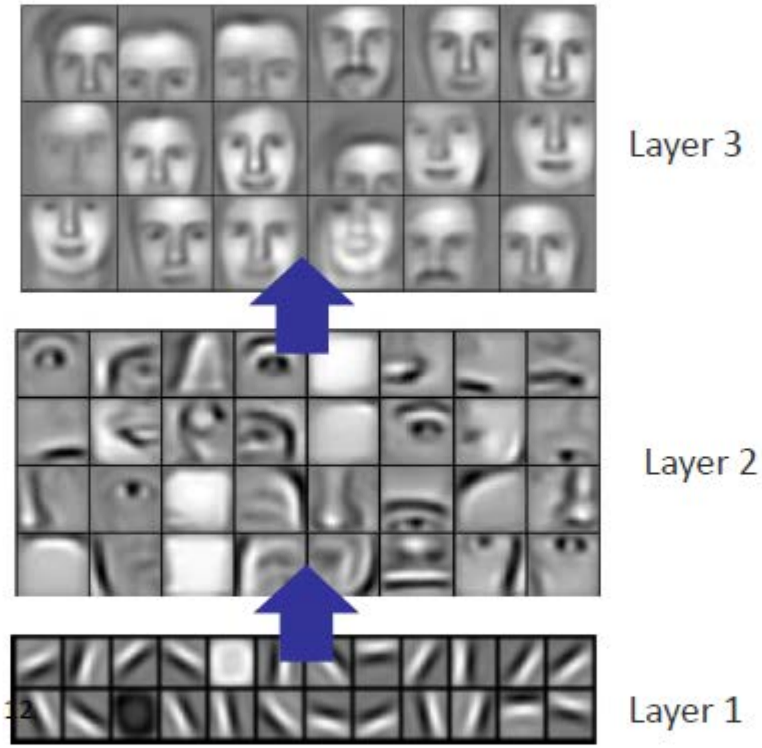
Expressive Capabilities of ANNs

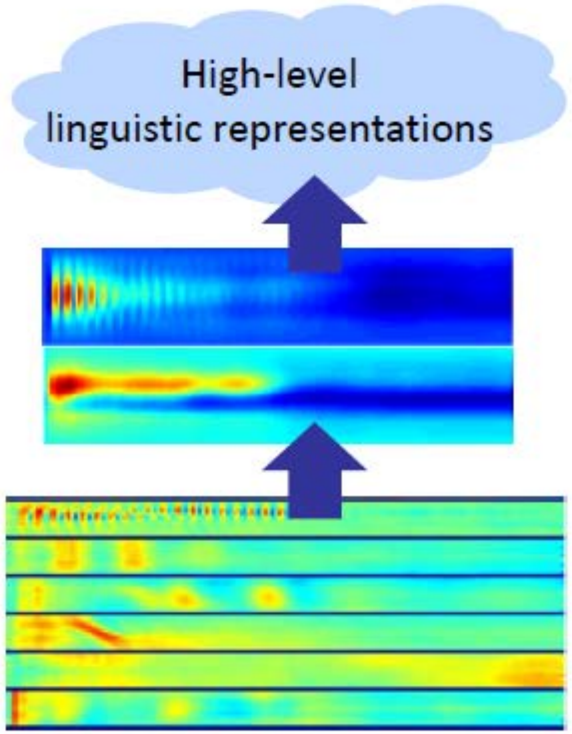
- Boolean functions:
  - Every Boolean function can be represented by network with single hidden layer
  - But might require exponential (in number of inputs) hidden units

- Continuous functions:
  - Every bounded continuous function can be approximated with arbitrary small error, by network with one hidden layer [Cybenko 1989; Hornik et al 1989]
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

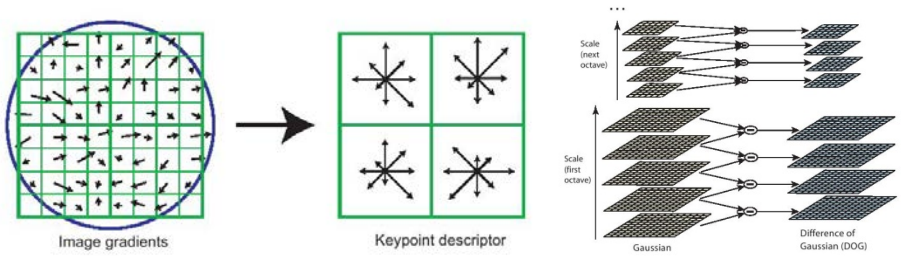
Feature learning

- Successful learning of intermediate representations
  [Lee et al ICML 2009, Lee et al NIPS 2009]

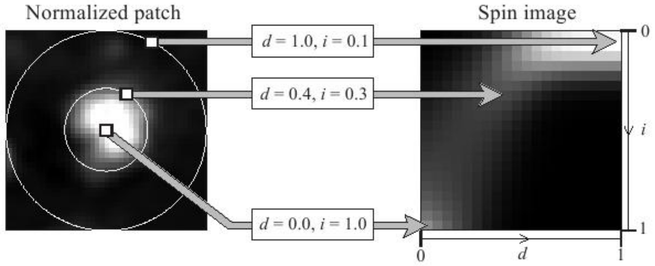




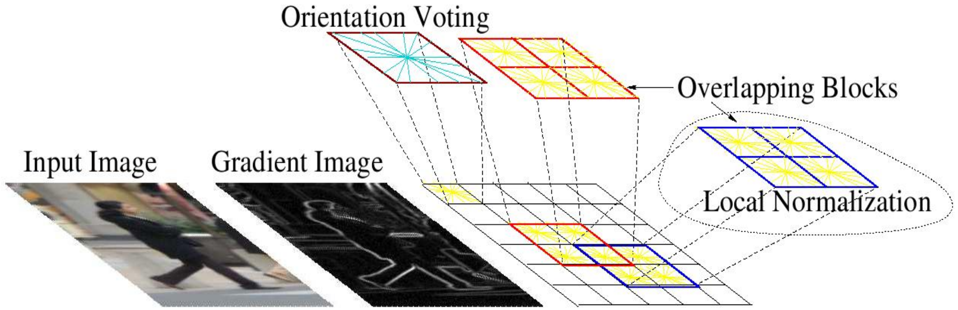
Computer vision features

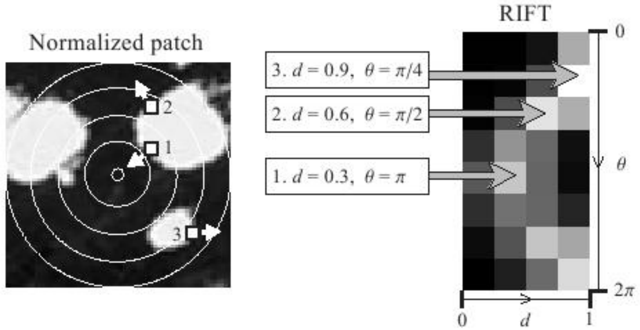


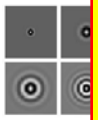
SIFT



Spin image

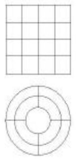






Drawbacks of feature engineering

1. Needs expert knowledge
2. Time consuming hand-tuning



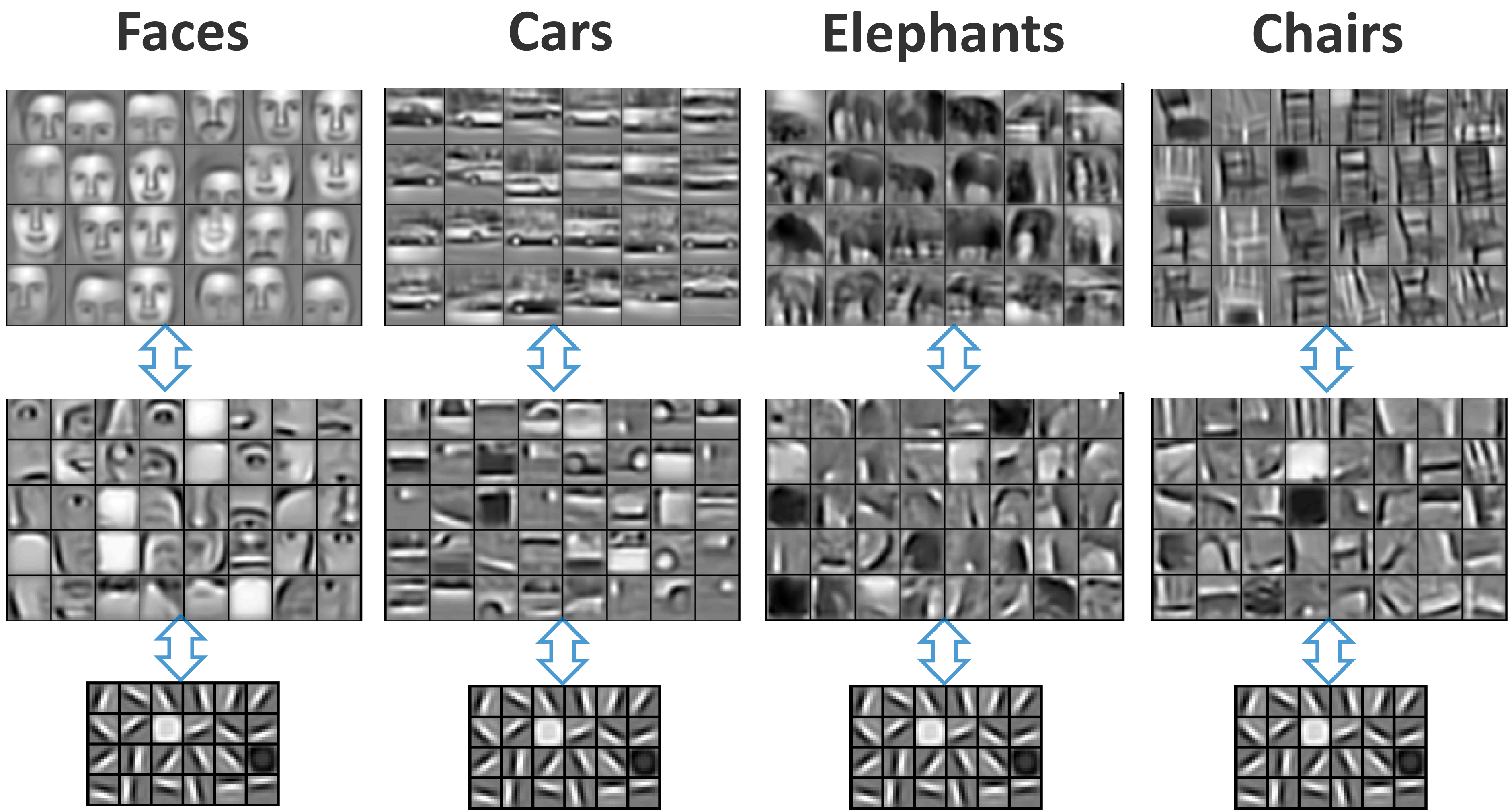
(e)

Textons

GLOH

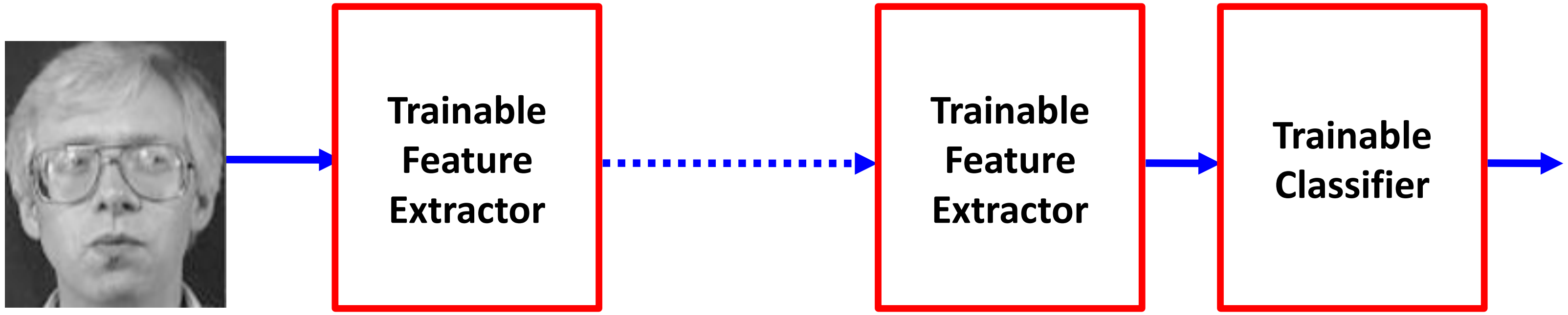
Courtesy: Lee and Ng

Unsupervised learning of object-parts



Courtesy: Lee and Ng

Using ANN to learn hierarchical representation



Trainable Feature Extractor

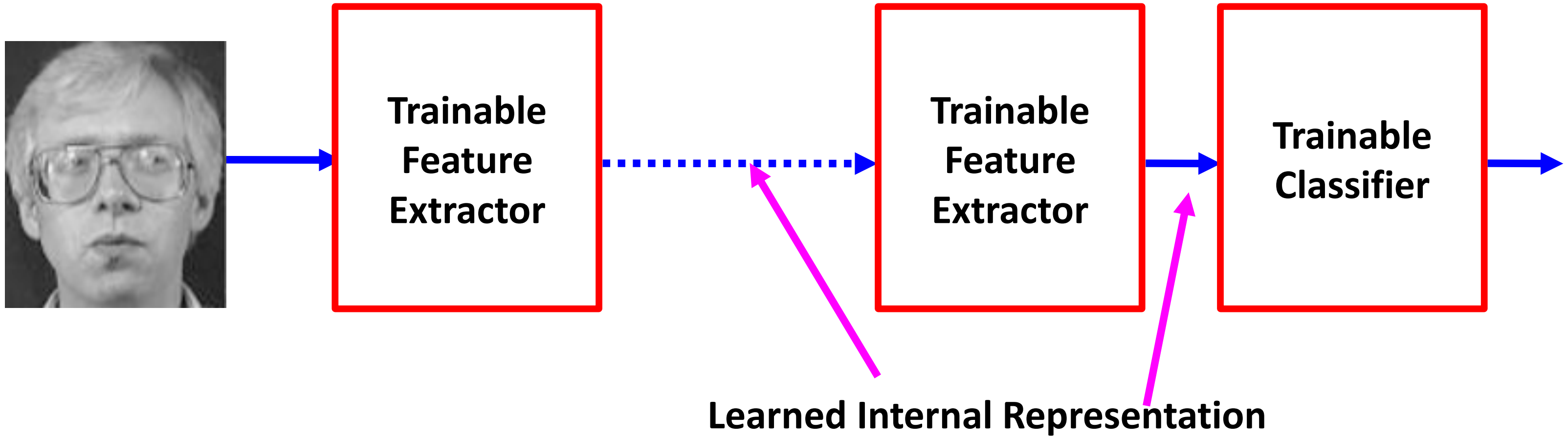
Trainable Feature Extractor

Trainable Classifier

Good Representations are hierarchical

- In Language: hierarchy in syntax and semantics
  - Words->Parts of Speech->Sentences->Text
  - Objects,Actions,Attributes...-> Phrases -> Statements -> Stories
- In Vision: part-whole hierarchy
  - Pixels->Edges->Textons->Parts->Objects->Scenes

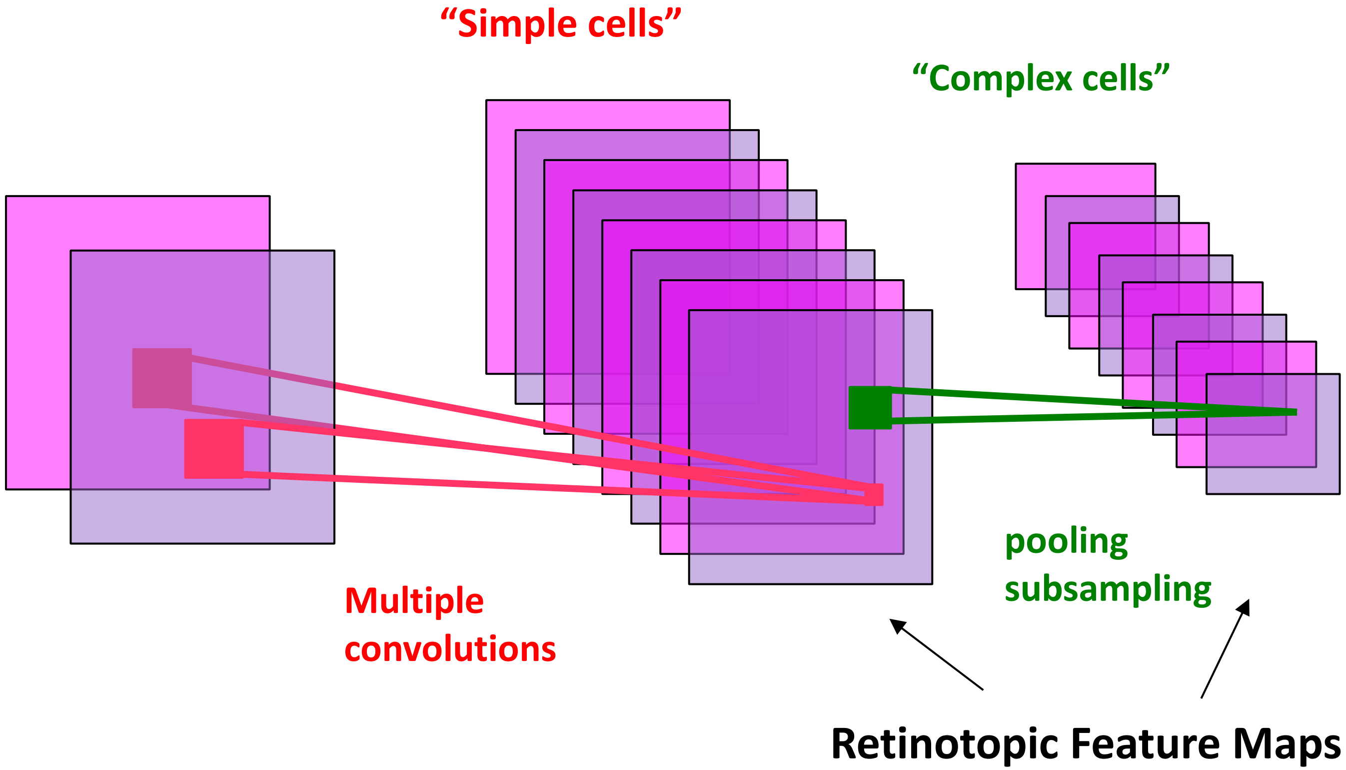
“Deep” learning: learning hierarchical representations



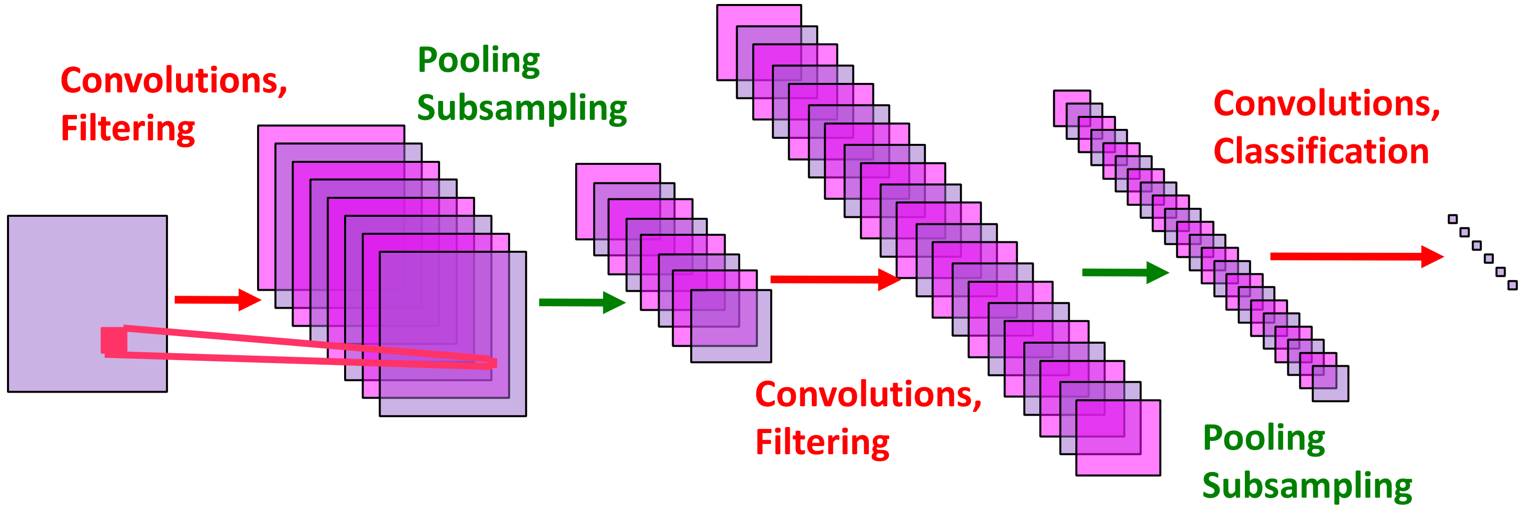
- Deep Learning: learning a hierarchy of internal representations
- From low-level features to mid-level invariant representations, to object identities
- Representations are increasingly invariant as we go up the layers
- Using multiple stages gets around the specificity/invariance dilemma

Filtering+NonLinearity+Pooling = 1 stage of a Convolutional Net

- [Hubel & Wiesel 1962]:
  - simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.

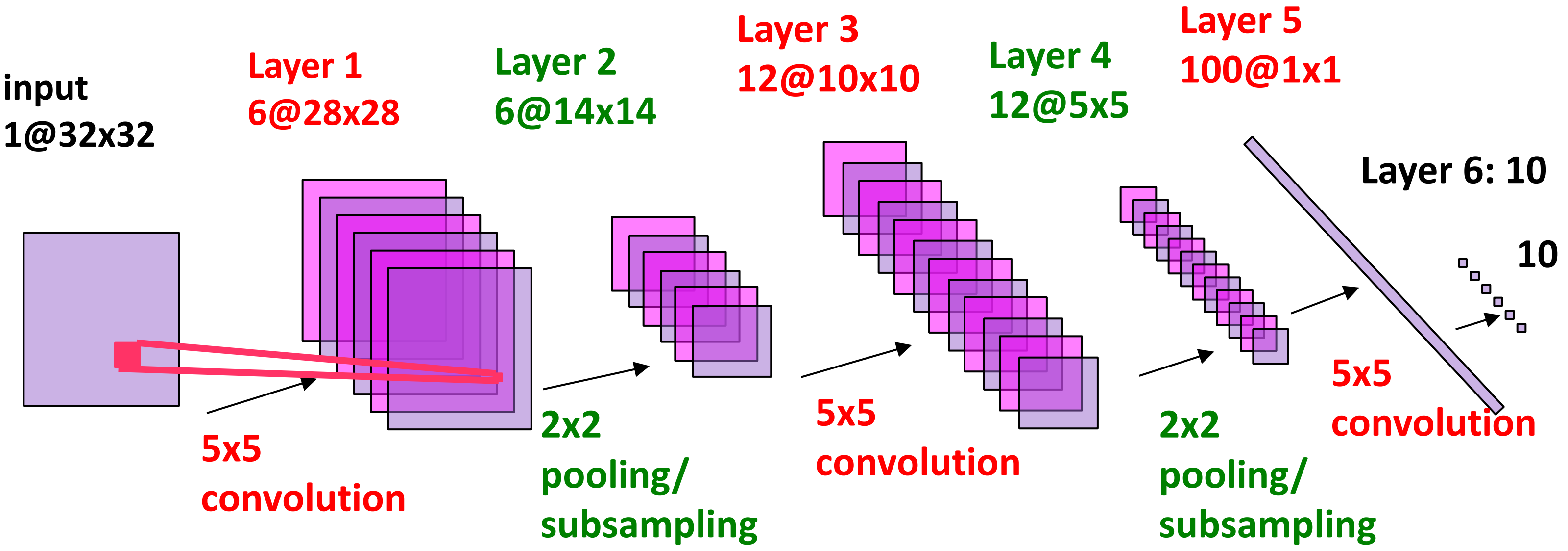


Convolutional Network: Multi-Stage Trainable Architecture



- **Hierarchical Architecture**
  - Representations are more global, more invariant, and more abstract as we go up the layers
- **Alternated Layers of Filtering and Spatial Pooling**
  - Filtering detects conjunctions of features
  - Pooling computes local disjunctions of features
- **Fully Trainable**
  - All the layers are trainable

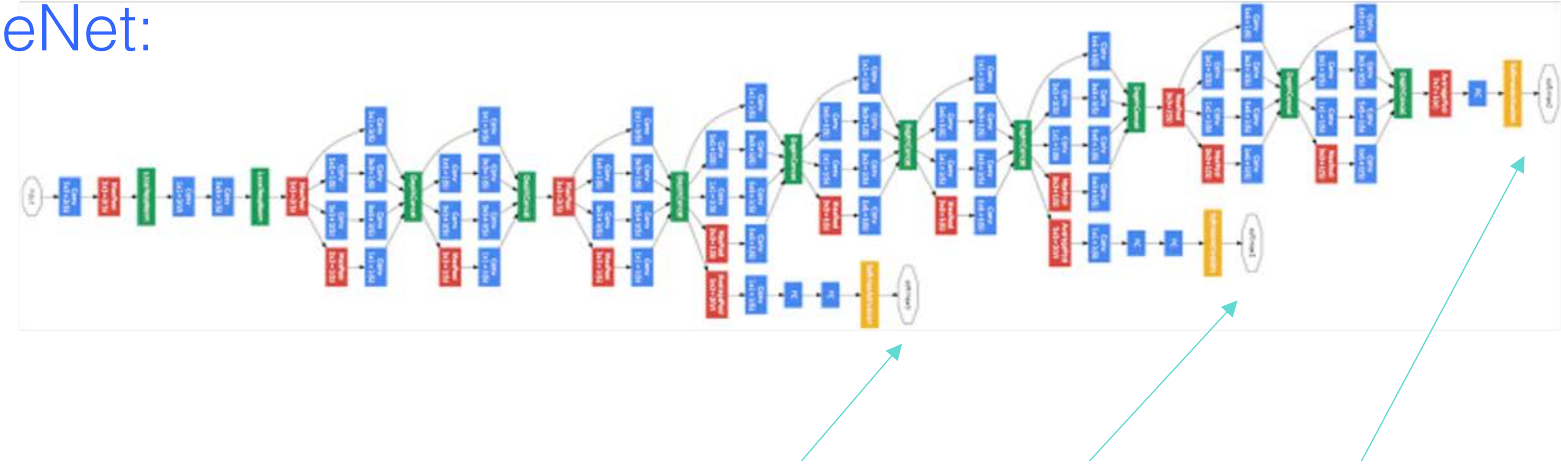
Convolutional Net Architecture for Hand-writing recognition



- Convolutional net for handwriting recognition (400,000 synapses)
  - Convolutional layers (simple cells): all units in a feature plane share the same weights
  - Pooling/subsampling layers (complex cells): for invariance to small distortions.
  - Supervised gradient-descent learning using back-propagation
  - The entire network is trained end-to-end. All the layers are trained simultaneously.
  - [LeCun et al. Proc IEEE, 1998]

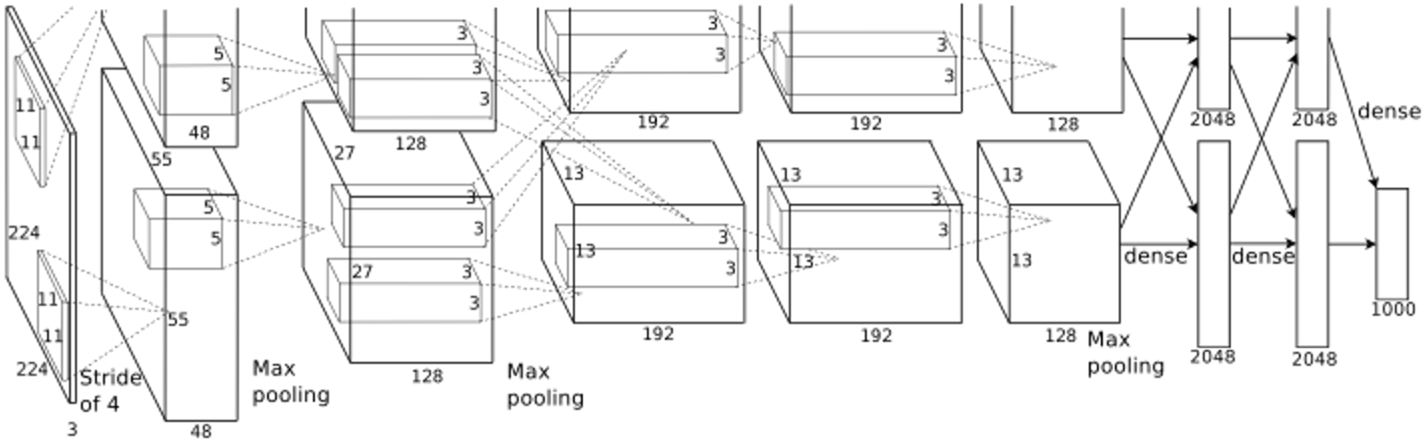
Training CNN: depth matters!

- GoogLeNet:



- 21 Layers!
- Gradient vanishes when the network is too deep: Lazy to learn!
  - Add intermediate loss layers to produce error signals!
  - Do contrast normalization after each conv layer!
  - Use ReLU to avoid saturation!

Training CNN: huge model, more data!



IMAGENET

- Only 7 layers, 60M parameters!
- Need more labeled data to train!
- Data augmentation: crop, translate, rotate, add noise!

Training CNN: highly nonconvex objective

- Demand more advanced optimization techniques
  - Add momentum as we have done for NN

  - Learning rate policy
    - decrease learning rate regularly!
    - different layers use different learning rate!
    - observe the trend of objective curve more often!

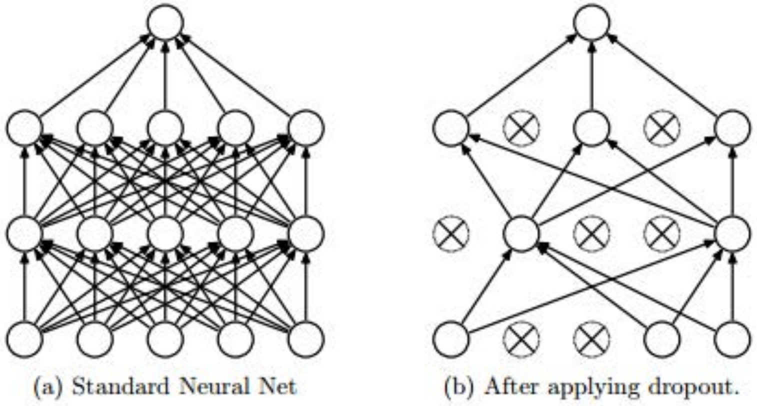
  - Initialization really cares!
    - Supervised pretraining
    - Unsupervised pretraining

Training CNN: avoid overfitting

- More data are always the best way to avoid overfitting
  - data augmentation

- Add regualizations: recall what we have done for linear regression

- Dropout



Summary: artificial neural networks – what you should know

- Highly expressive non-linear functions
- Highly parallel network of logistic function units
- Minimizing sum of squared training errors
  - Gives MLE estimates of network weights if we assume zero mean Gaussian noise on output values
- Minimizing sum of sq errors plus weight squared (regularization)
  - MAP estimates assuming weight priors are zero mean Gaussian
- Gradient descent as training procedure
  - How to derive your own gradient descent procedure
- Discover useful representations at hidden units
- Local minima is greatest problem
- Overfitting, regularization, early stopping

Limitations

- Supervised Training
  - Need huge amount of labeled data, but label is scarce!
  - Pre-training, self-supervised training …

- Slow Training
  - Train an AlexNet on a single machine need one week!

- Optimization
  - Highly nonconvex objective

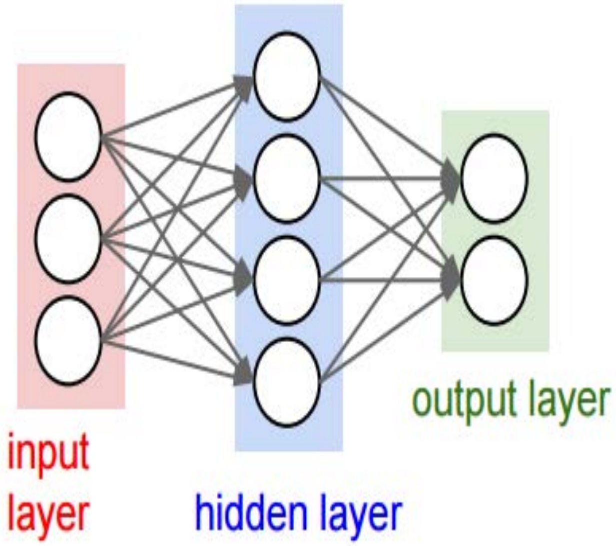
- Parameter tuning is hard
  - The parameter space is so large…

Supplementary

Detailed Tutorial on Convolutional Neural Network

Some contents are borrowed from Rob Fergus, Yan Lecun and Stanford's course

Ordinary Neural Network



Now

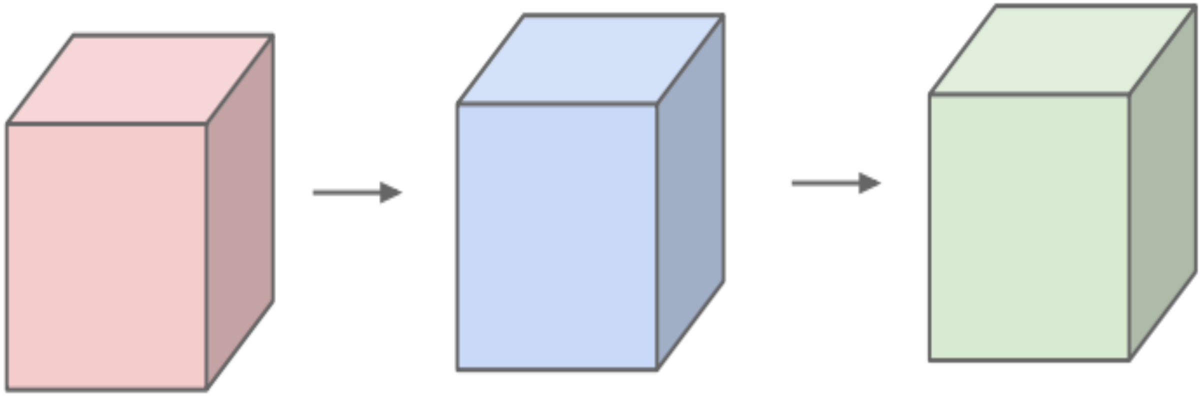
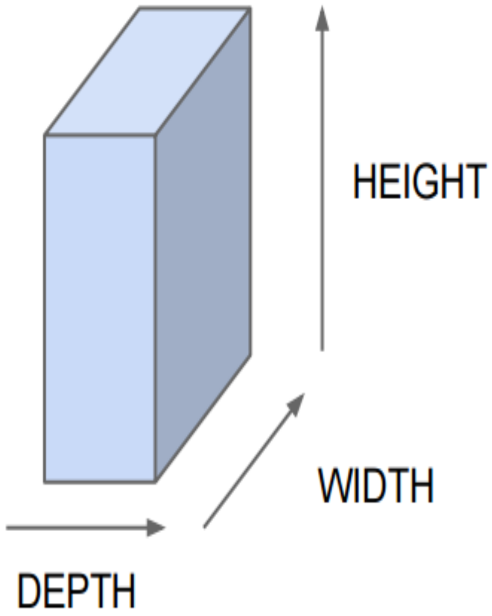


Figure courtesy, Fei-Fei, Andrej Karpathy

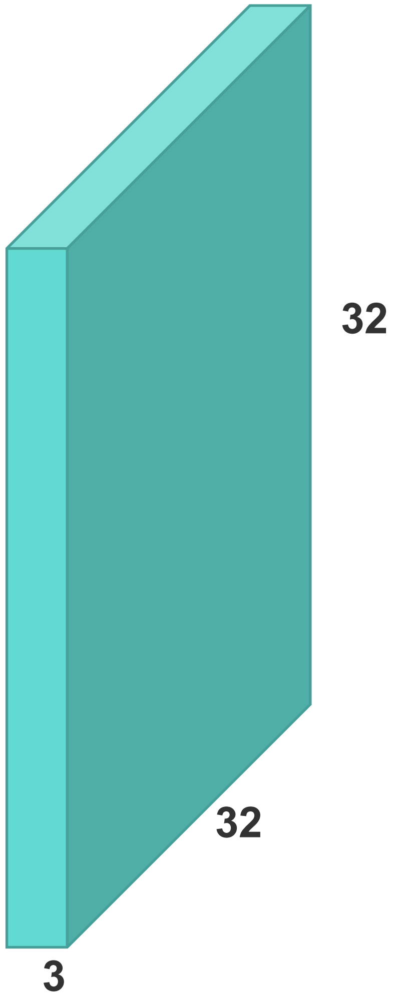
All Neural Net activations arranged in 3 dimensions

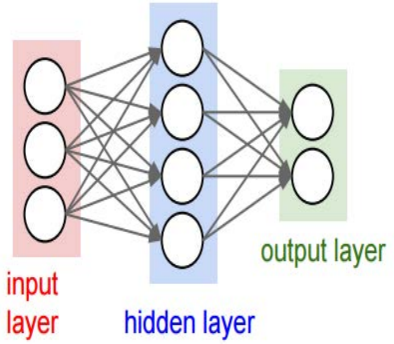


For example, a CIFAR-10 image is a 32*32*3 volume: 32 width, 32 height, 3 depth (RGB)

Figure courtesy, Fei-Fei, Andrej Karpathy

Local connectivity





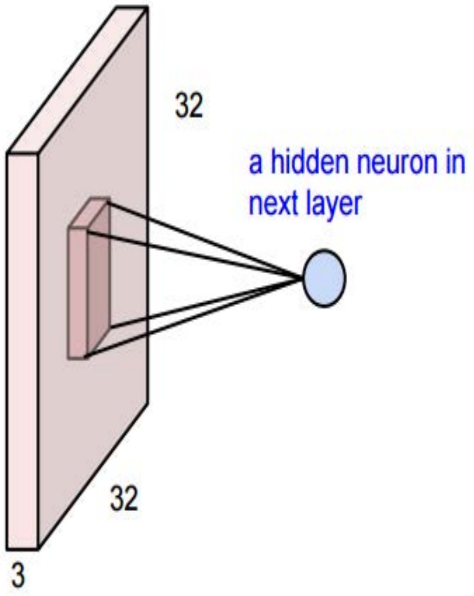


image: 32 * 32 * 3 volume

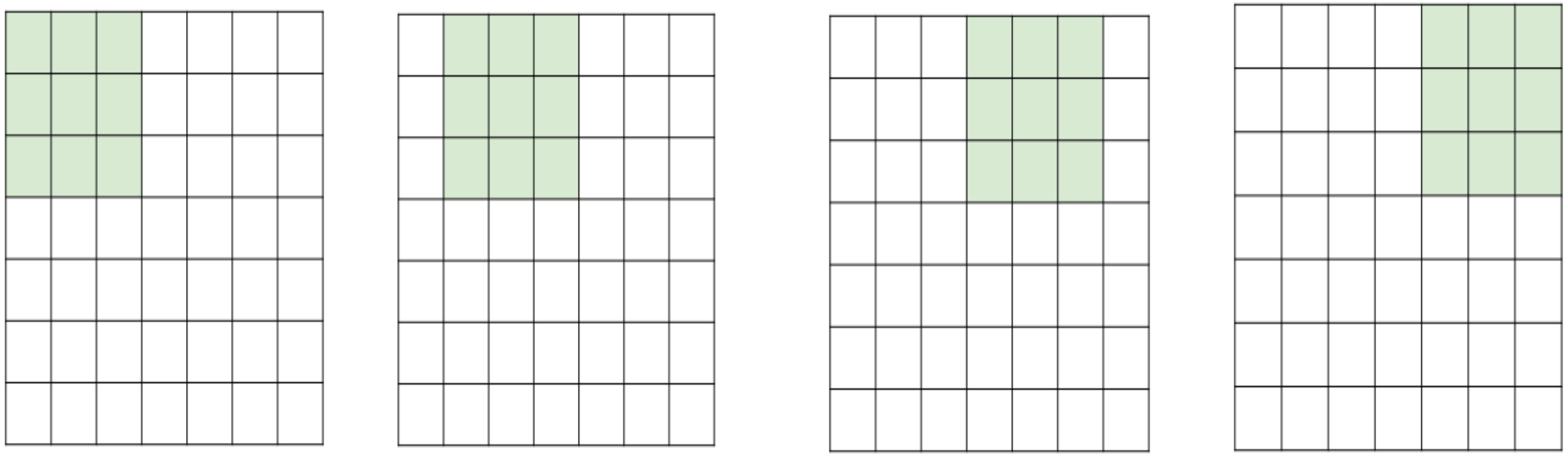
before: full connectivity:
32 * 32 * 3 weights for each neuron

now: one unit will connect to, e.g. 5*5*3 chunk and only have 5*5*3 weights

Note the connectivity is:
- local in space
- full in depth

Convolution

- One local region only gives one output
- Convolution: Replicate the column of hidden units across space, with some stride



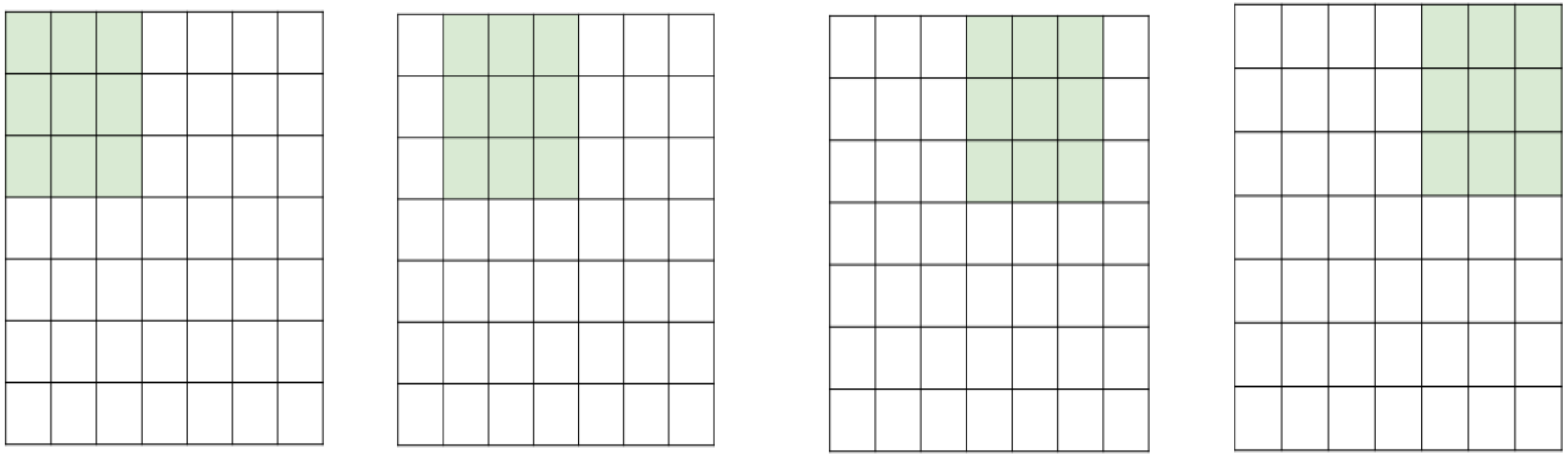
- 7 * 7 Input
- Assume 3*3 connectivity, stride = 1

→

- Produce a map
- What's the size of the map? 5 * 5

Convolution

- One local region only gives one output
- Convolution: Replicate the column of hidden units across space, with some stride

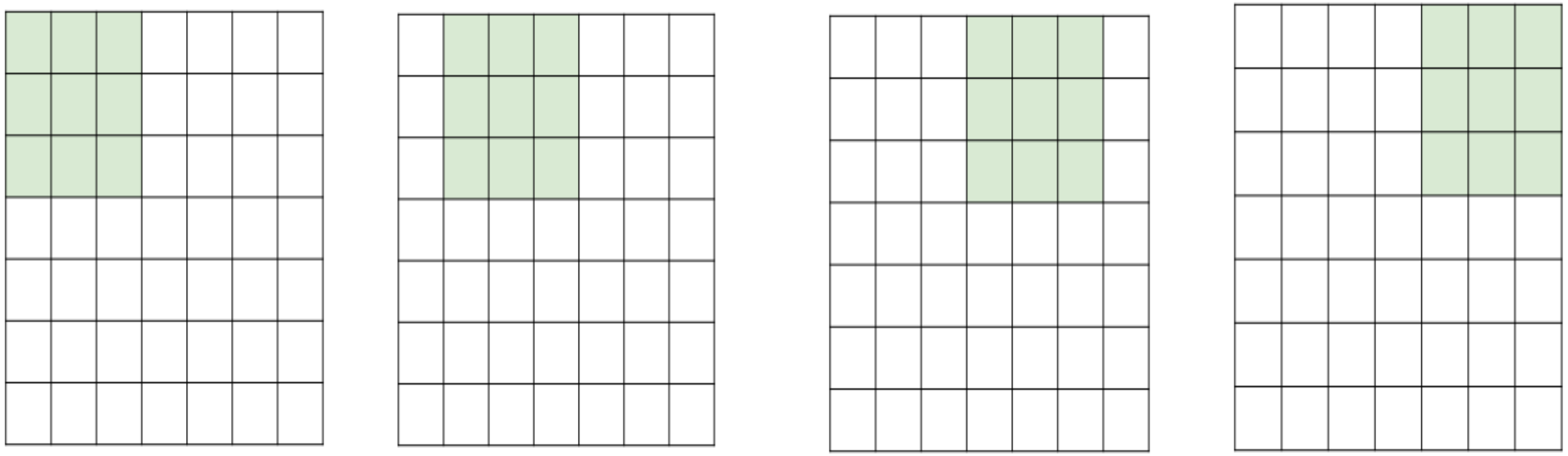


- 7 * 7 Input
- Assume 3*3 connectivity, stride = 1

- What if stride = 2?

Convolution

- ❑ One local region only gives one output
- ❑ Convolution: Replicate the column of hidden units across space, with some stride



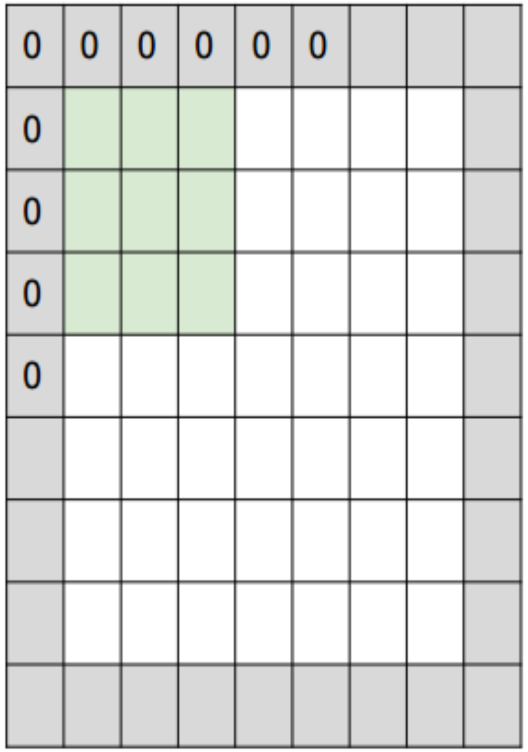
- 7 * 7 Input
- Assume 3*3 connectivity, stride = 1

→

- What if stride = 3?

Convolution: In Practice

- Zero Padding
  - Input size: 7 * 7
  - Filter Size: 3*3, stride 1
  - Pad with 1 pixel border
  - Output size?
    - 7 * 7 => preserved size!



Slide courtesy, Fei-Fei, Andrej Karpathy

Convolution: Summary

- Zero Padding
  - Input volume of size [W1 * H1 * D1]
  - Using K units with receptive fields F x F and applying them at strides of S gives

Output volume: [W2, H2, D2]

- $W2 = (W1 - F)/S + 1$
- $H2 = (H1 - F) / S + 1$
- $D2 = k$

Slide courtesy, Fei-Fei, Andrej Karpathy

Convolution: Problem

- Assume input [32 * 32 * 3]
- 30 units with receptive field 5 * 5, applied at stride 1/pad 1

  => Output volume: [30 * 30 * 30]

At each position of the output volume, we need 5 * 5 * 3 weights

  => Number of weights in such layer: 27000 * 75 = 2 million ☹

Idea:
 Weight sharing!

Learn one unit, let the unit convolve across all local receptive fields!

Convolution: Problem

- ☐ Assume input [32 * 32 * 3]
- ☐ 30 units with receptive field 5 * 5, applied at stride 1/pad 1

  => Output volume: [30 * 30 * 30] = 27000 units

Weight sharing

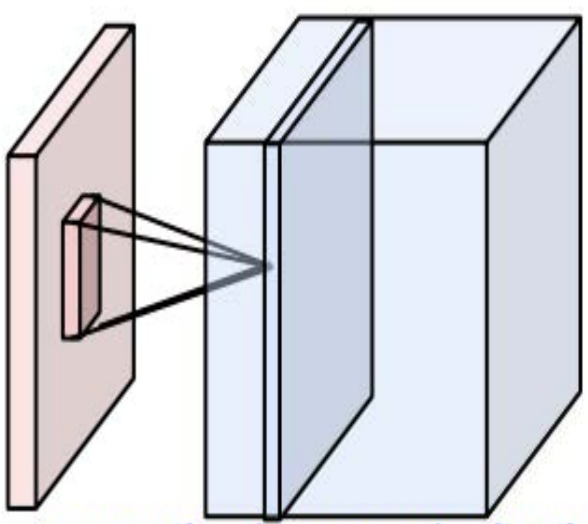
  => Before: Number of weights in such layer: 27000 * 75 = 2 million ☹

  => After: weight sharing: 30 * 75 = 2250 ☺

But also note that sometimes it's not a good idea to do weight sharing! When?

Convolutional Layers

- Connect units only to local receptive fields
- Use the same unit weight parameters for units in each "depth slice" (i.e. across spatial positions)



one activation map (a depth slice), computed with one set of weights

Can call the units "**filters**"

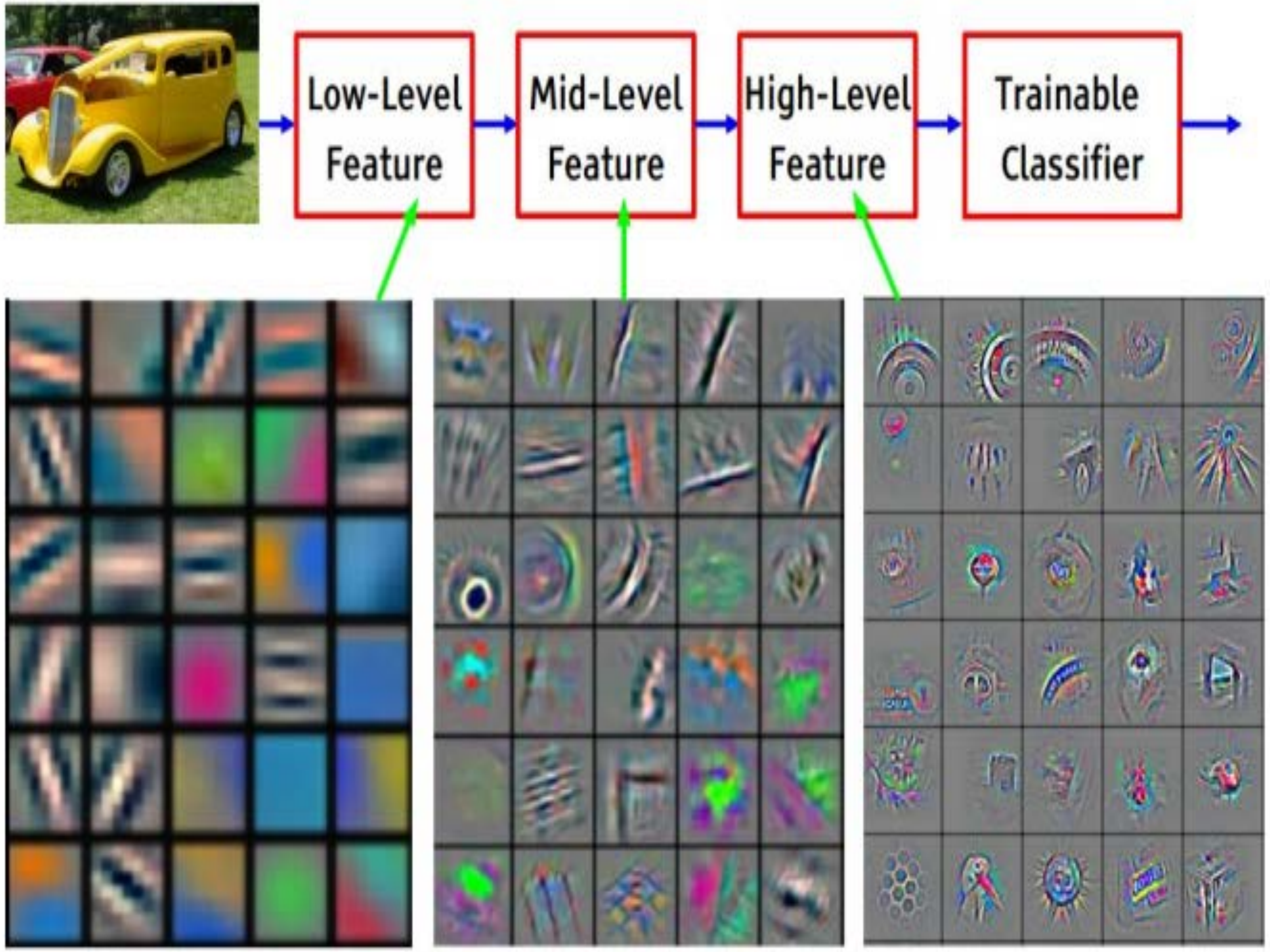
We call the layer convolutional because it is related to convolution of two signals

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

Sometimes we also add a bias term b, y = Wx + b, like what we have done for ordinary NN

Short question: Will convolution layers introduce nonlinearity?

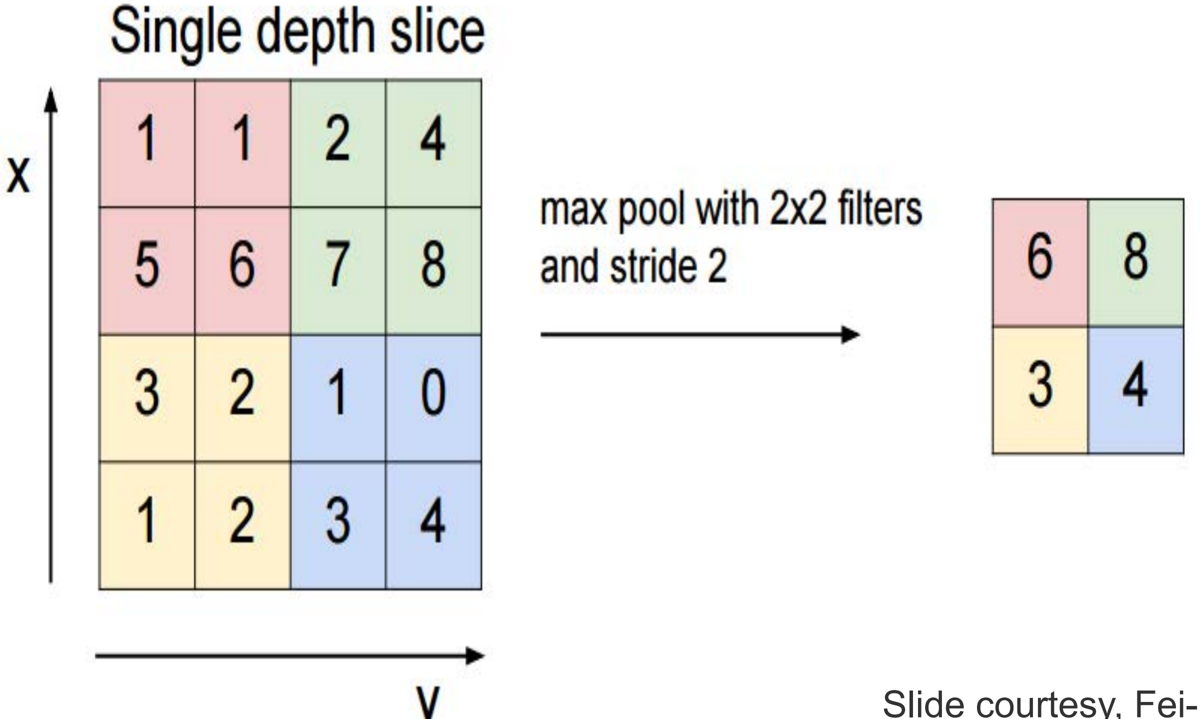
Stacking Convolutional Layers



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

Pooling Layers

- In ConvNet architectures, Conv layers are often followed by Pool layers
  - makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)



Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

6	8
3	4

Slide courtesy, Fei-Fei, Andrej Karpathy

Pooling Layers

- In ConvNet architectures, Conv layers are often followed by Pool layers
  - makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)
- Input volume of size [W1 x H1 x D1]
- Pooling unit receptive fields F x F and applying them at strides of S gives
- Output volume: [W2, H2, D1]: depth unchanged!

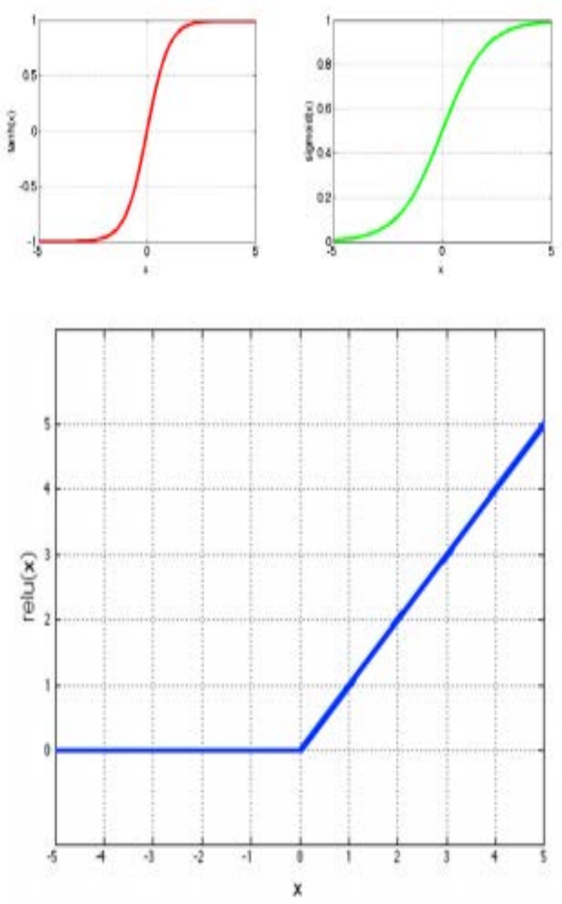
$$W2 = (W1-F)/S+1,$$

$$H2 = (H1-F)/S+1$$

Short question: Will pooling layer introduce nonlinearity?

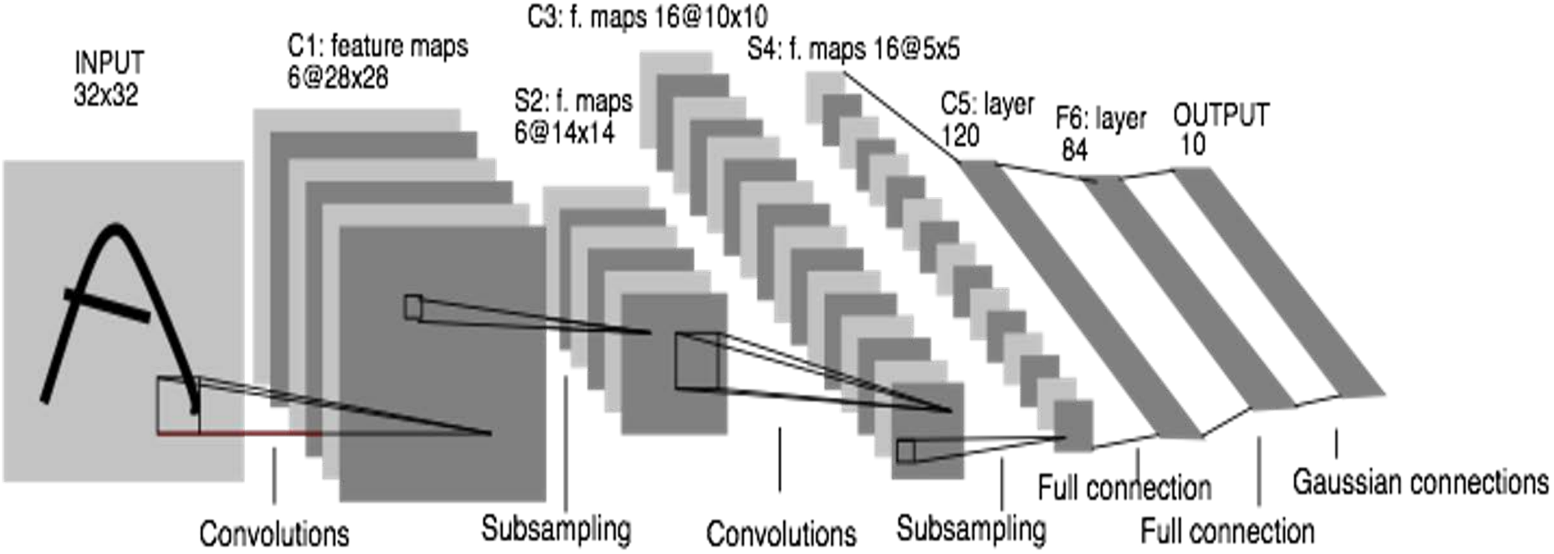
Nonlinerity

- Similar to NN, we need to introduce nonlinearity in CNN
  - Sigmoid
  - Tanh
  - RELU: Rectified Linear Units -> preferred
    - Simplifies backpropagation
    - Makes learning faster
    - Avoids saturation issues



Slide courtesy, Yan Lecun

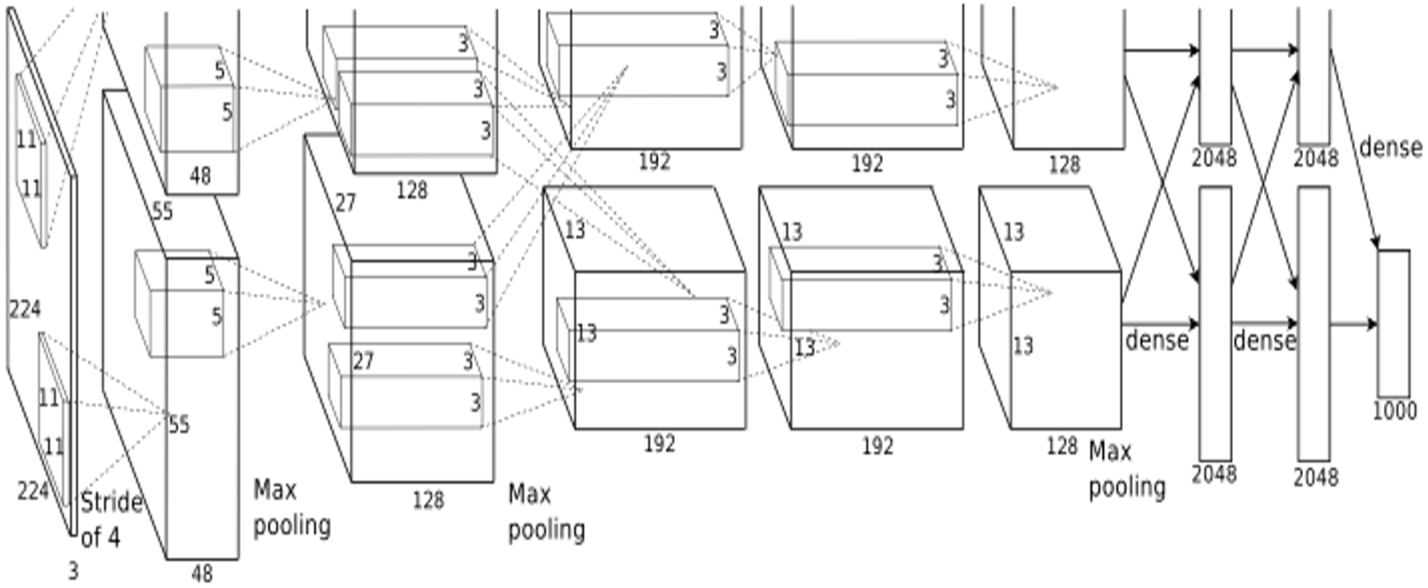
Convolutional Networks: 1989



- LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [LeNet]

Slide courtesy, Yangqing Jia

Convolutional Nets: 2012

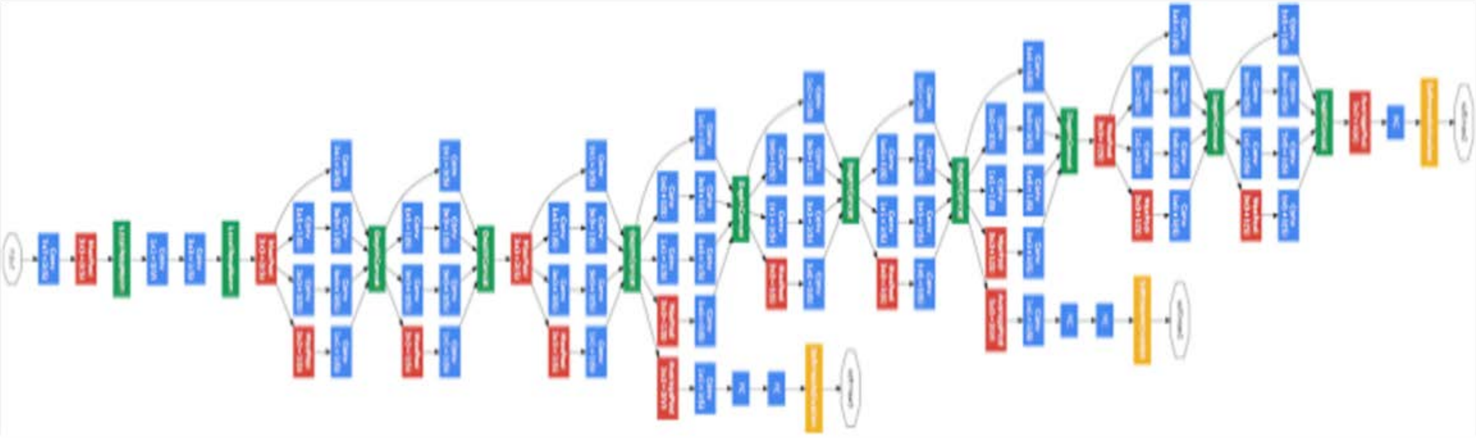


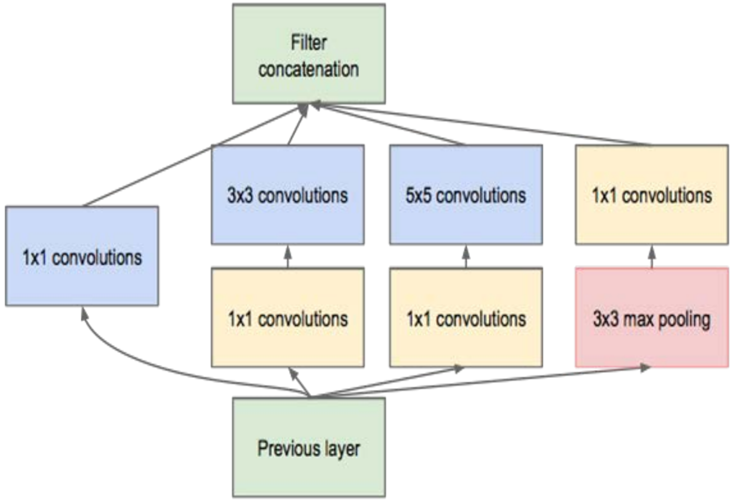
+ data
+ gpu
+ non-saturating nonlinearity
+ regularization

- AlexNet: a layered model composed of convolution, subsampling, and further operations followed by a holistic representation and all-in-all a landmark classifier on
- ILSVRC12. [AlexNet]

Slide courtesy, Yangqing Jia

Convolutional Nets: 2014





+ depth
+ data
+ dimensionality reduction

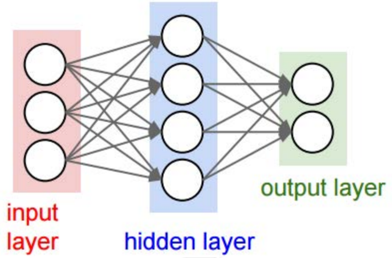
- ILSVRC14 Winners: ~6.6% Top-5 error
  - GoogLeNet: composition of multi-scale dimension-reduced modules (pictured)
  - VGG: 16 layers of 3x3 convolution interleaved with max pooling + 3 fully-connected layers

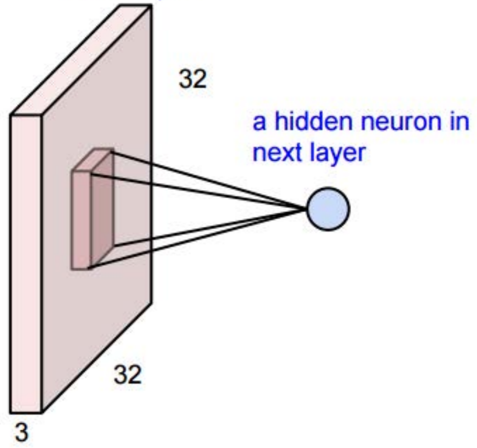
Slide courtesy, Yangqing Jia

Training CNN: Use GPU

- Convolutional layers
  - Reduce parameters BUT Increase computations

- FC layers
  - each neuron has more weights
  - but less computations





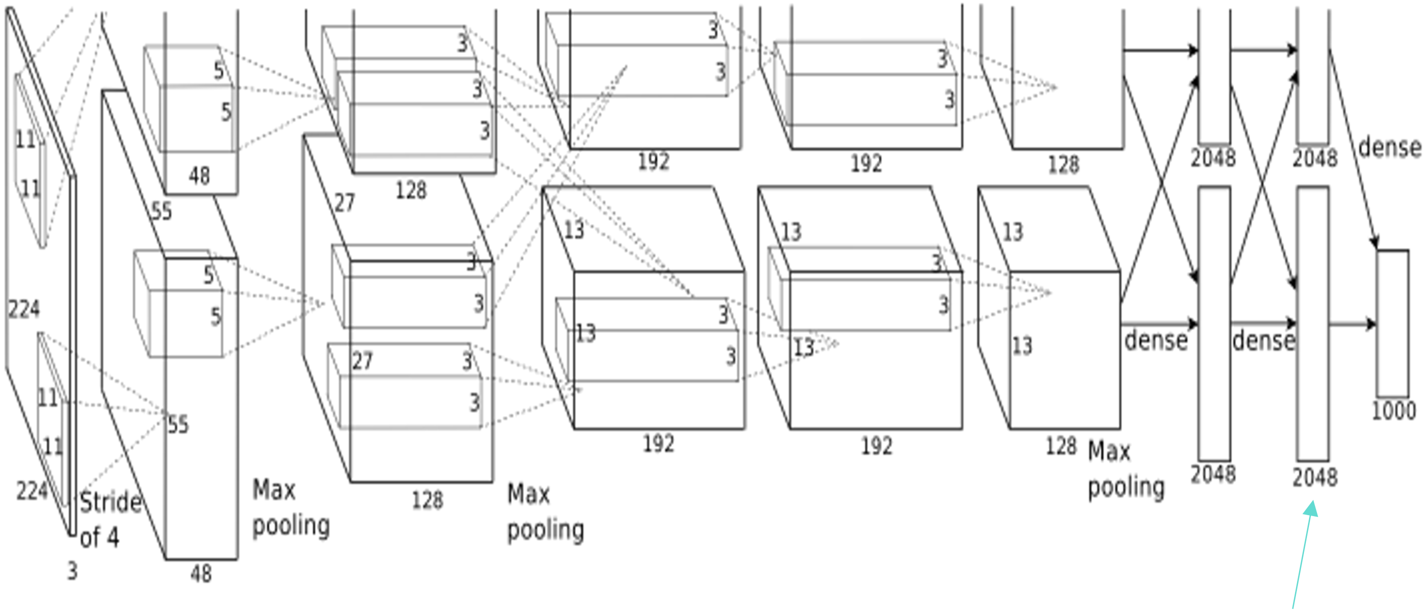
- Conv layers
  - each neuron has less weights
  - but more computations. Why?

because c

it will co ion!

GPU is good at convolution!

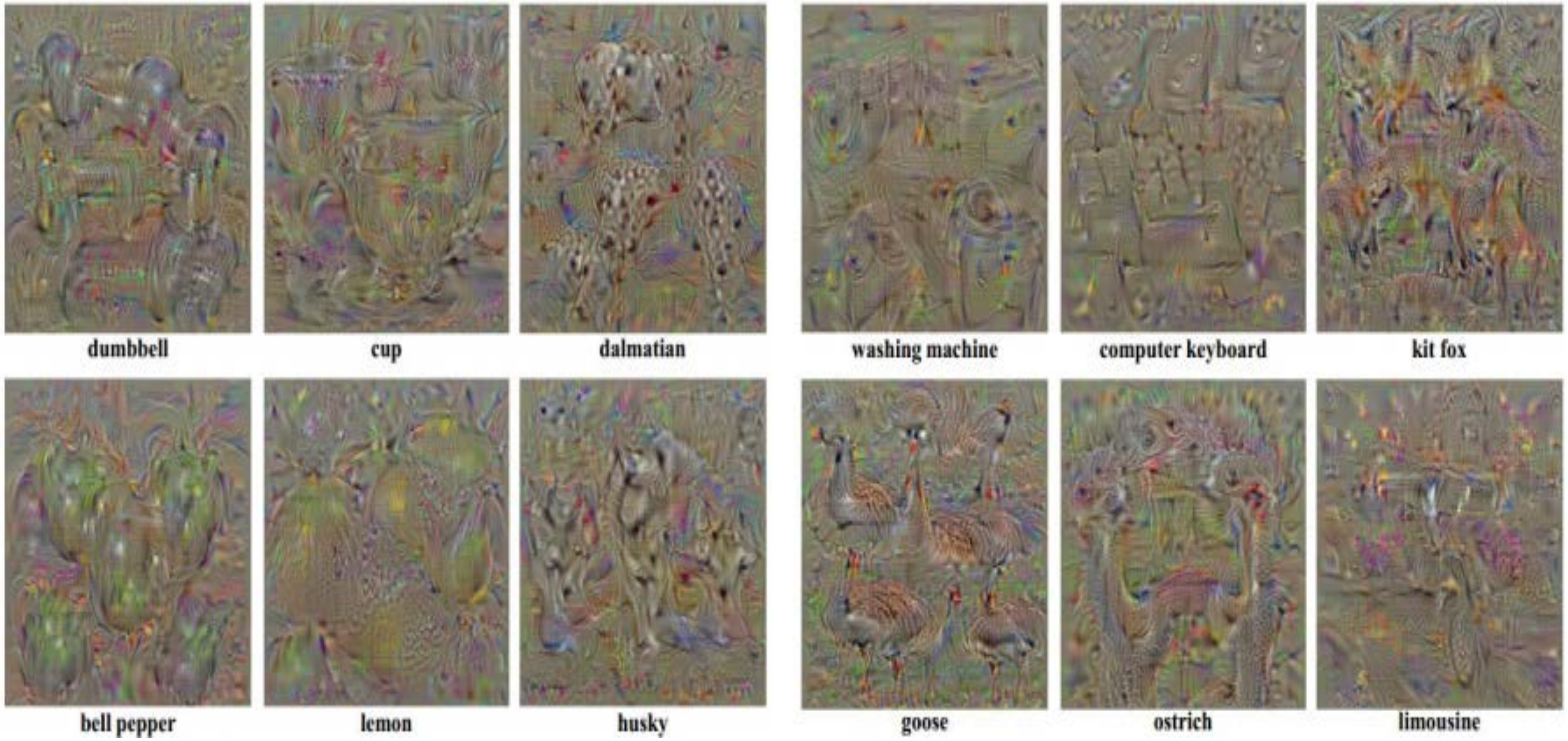
Visualize and Understand CNN



A CNN transforms the image to 4096 numbers that are then linearly classified.

Visualize and Understand CNN

- Find images that maximize some class score:



Yes, Google Inceptionism!