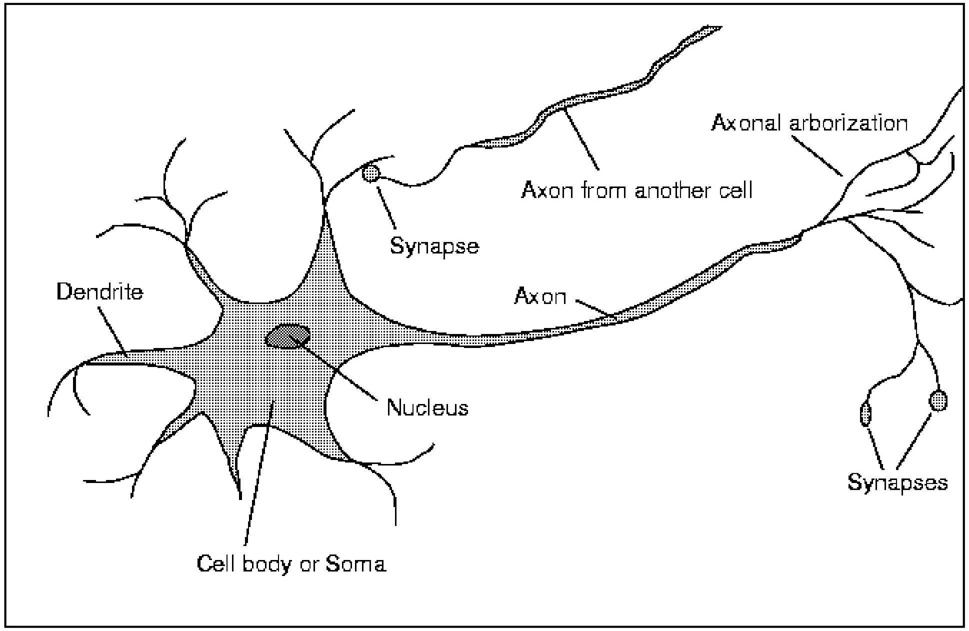
Introduction to Data Analytics

Xin Gao
Xin.gao@kaust.edu.sa
July 29, 2022
SDU

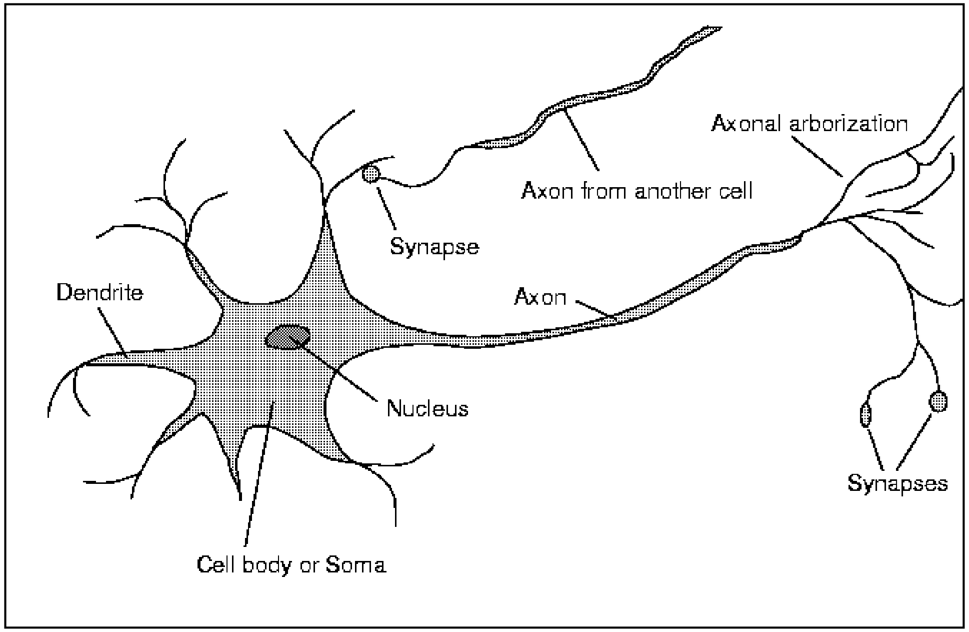
Neurons

- A **neuron** is an electrically excitable cell (threshold switching unit) that processes and transmits information by electrical and chemical signaling
  - Dendrites, axon, synapses
- The cell body of a neuron frequently gives rise to multiple dendrites, but never to more than one axon, although the axon may branch hundreds of times before it terminates



Neurons

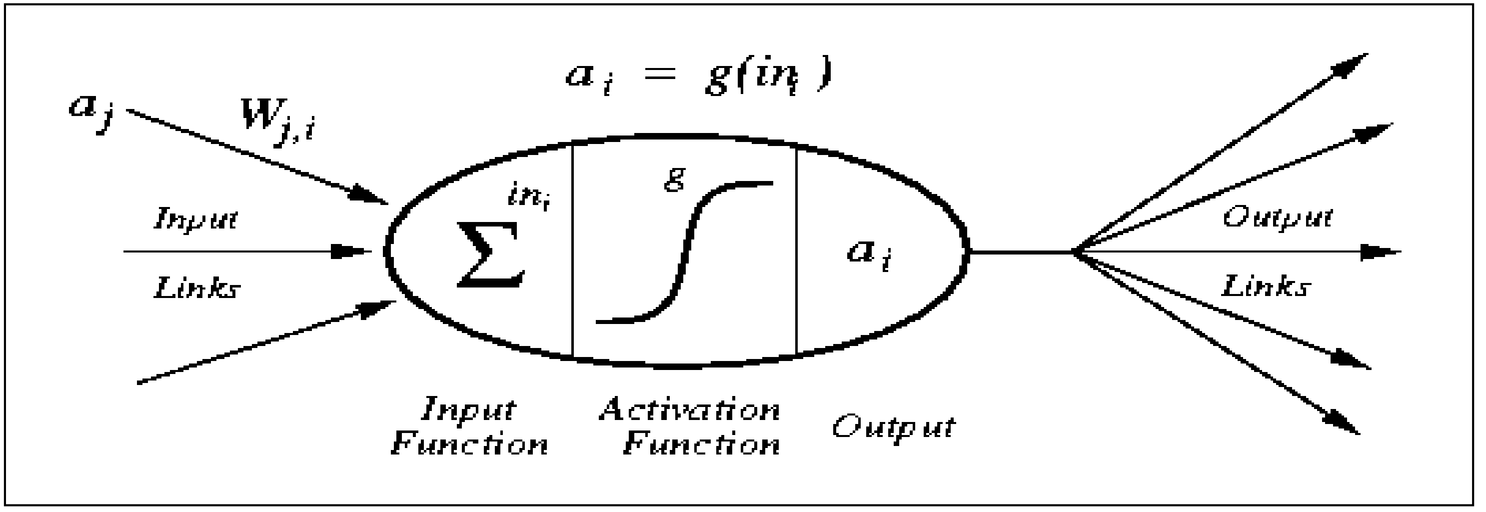
- **Dendrites:** filaments that arise from the cell body, often extending for hundreds of micrometers and branching multiple times, giving rise to a complex "dendritic tree“
- **Axon:** a special cellular filament that arises from the cell body at a site called the axon hillock and travels for a distance, as far as 1m in humans or even more in other species
- **Synapses:** send signals from the axon of one neuron to a dendrite of another

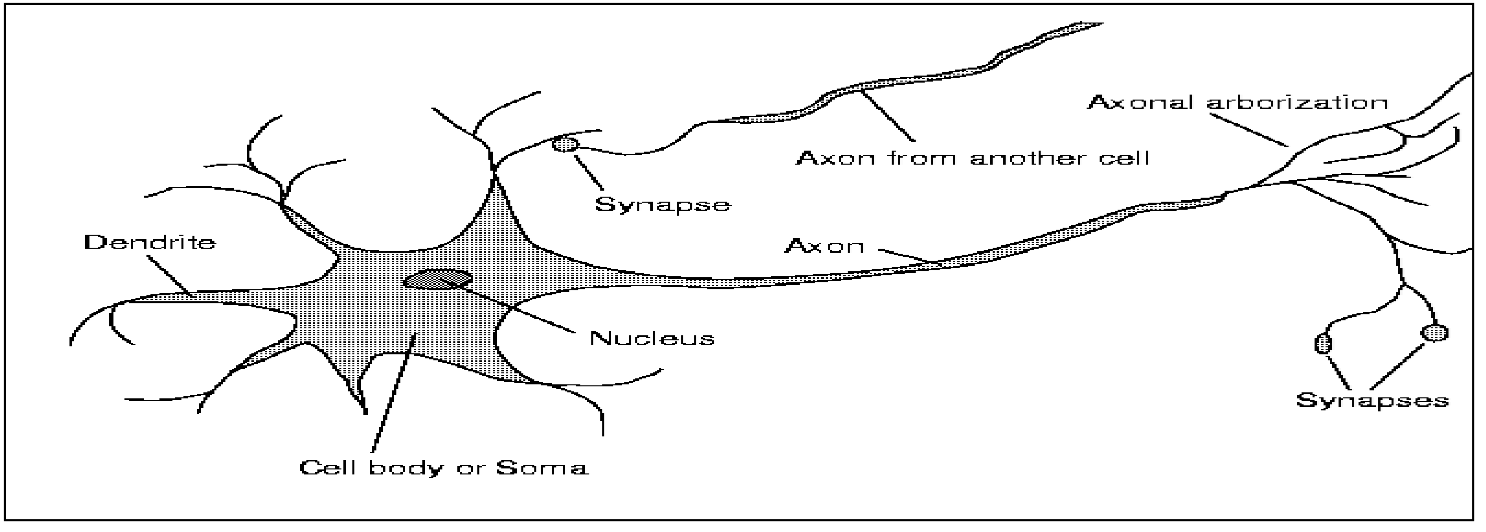


Neurons

- We are born with about 100 billion neurons
- Computers are at least 10^6 times faster in raw switching speed
- But the brain is faster and reliable at computationally intensive tasks, such as computer vision, speech recognition, etc
- The brain is also fault-tolerant, and exhibits graceful degradation with damage
  - A neuron may connect to as many as 100,000 other neurons
  - Even if you break 50% of the connections, the brain can still function properly
    - Very strong and robust connection construction

Artificial Neurons





Artificial Neural Networks

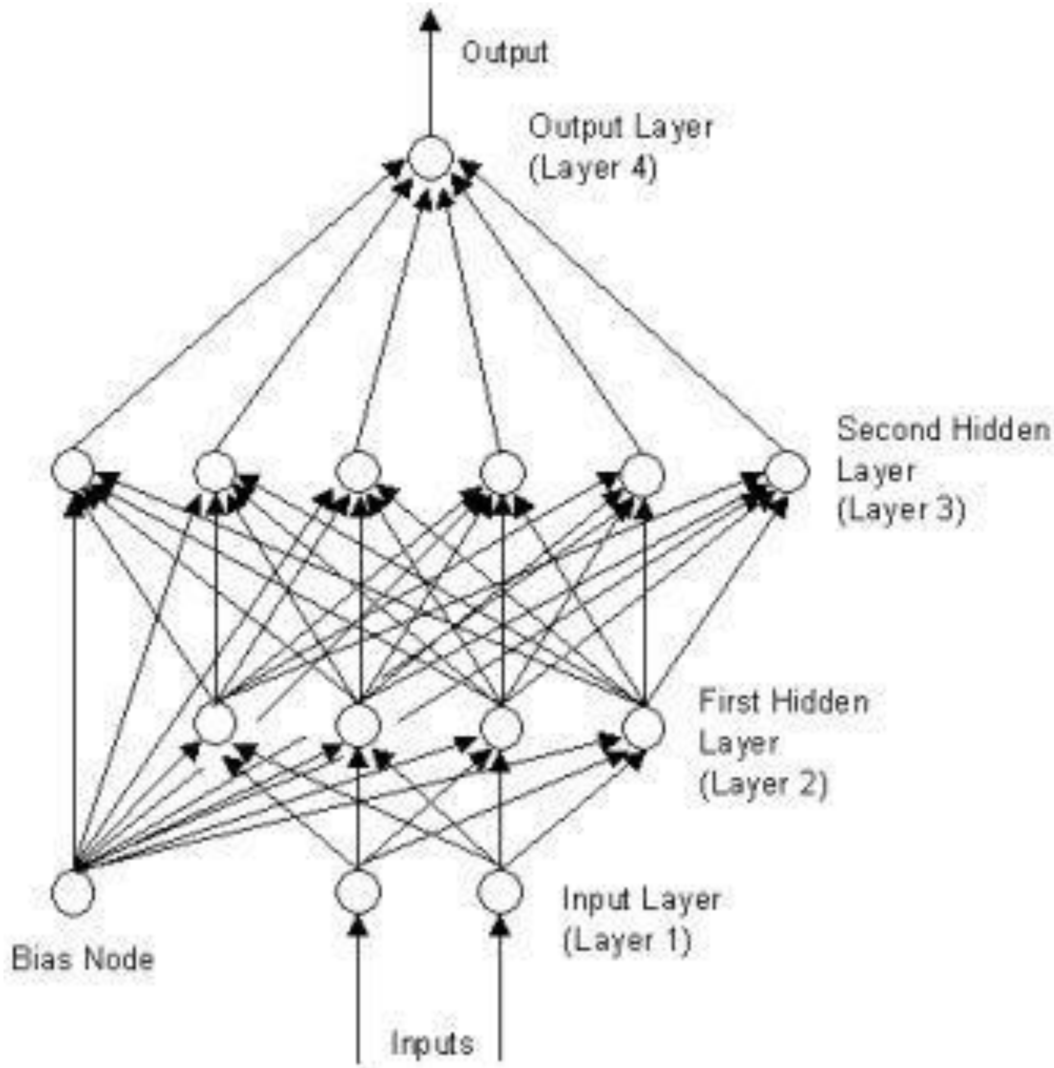
- f might be non-linear function
- X (vector of) continuous and/or discrete variables
- Y (vector of) continuous and/or discrete variables

- Represent f by network of logistic units
- Each unit is a logistic function

$$\text{unit output} = \frac{1}{1 + \exp(\mathrm{w}_0 + \sum_{\mathrm{i}} \mathrm{w}_{\mathrm{i}} \mathrm{x}_{\mathrm{i}})}$$

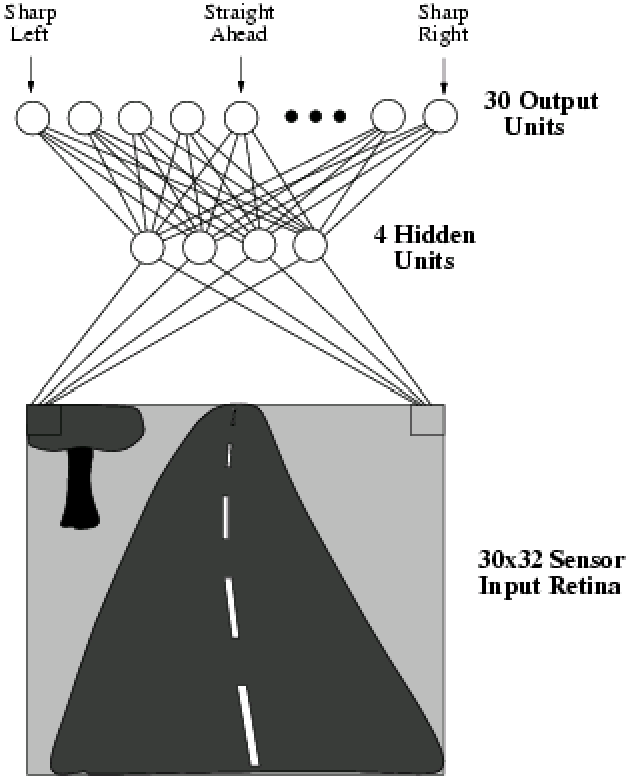
- Goal: train weights of all units to minimize the errors of predicted network outputs

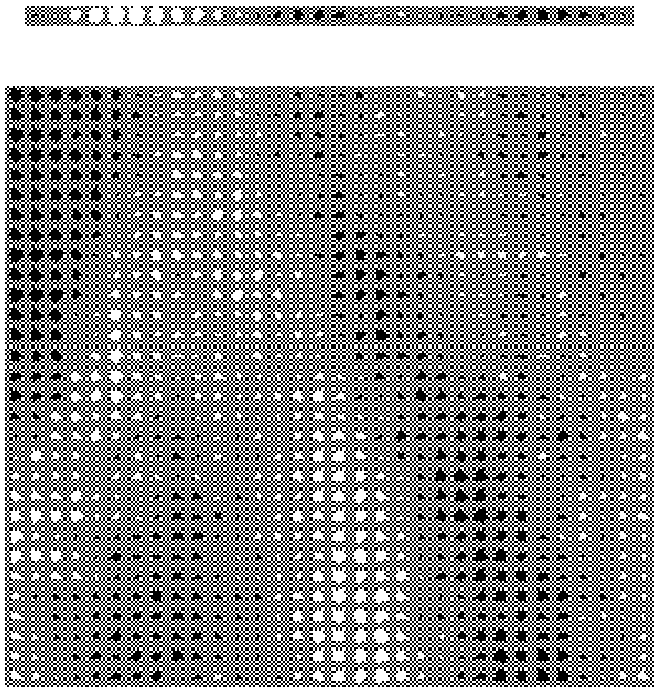
Artificial Neural Networks



Example

- ALVINN: an autonomous land vehicle in a neural network – Pomerleau 1993

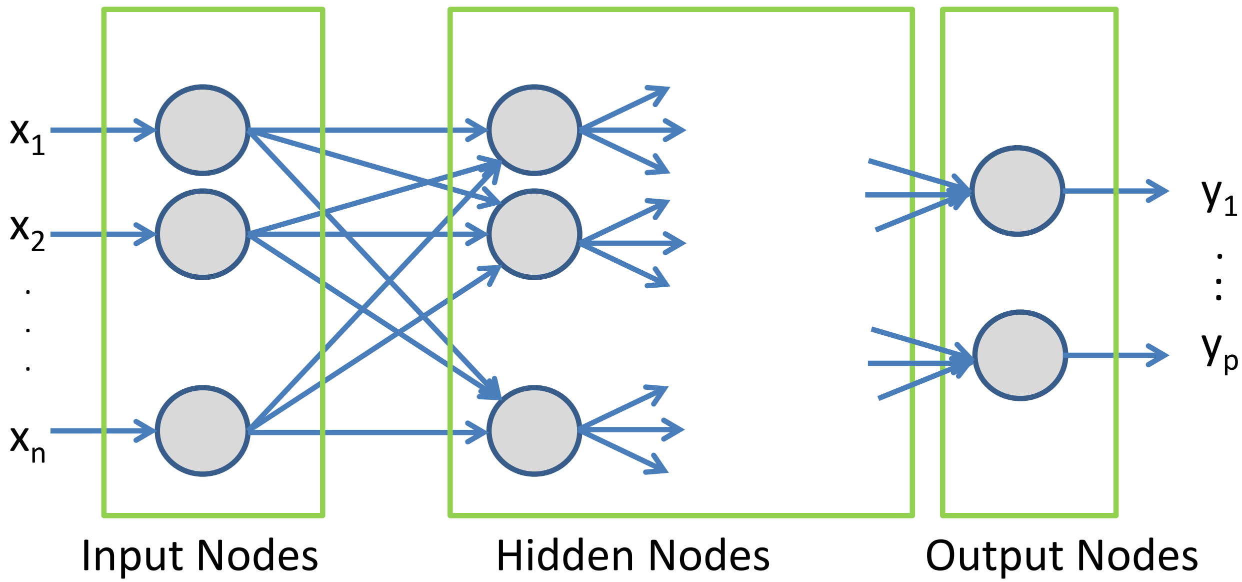




Connection Models

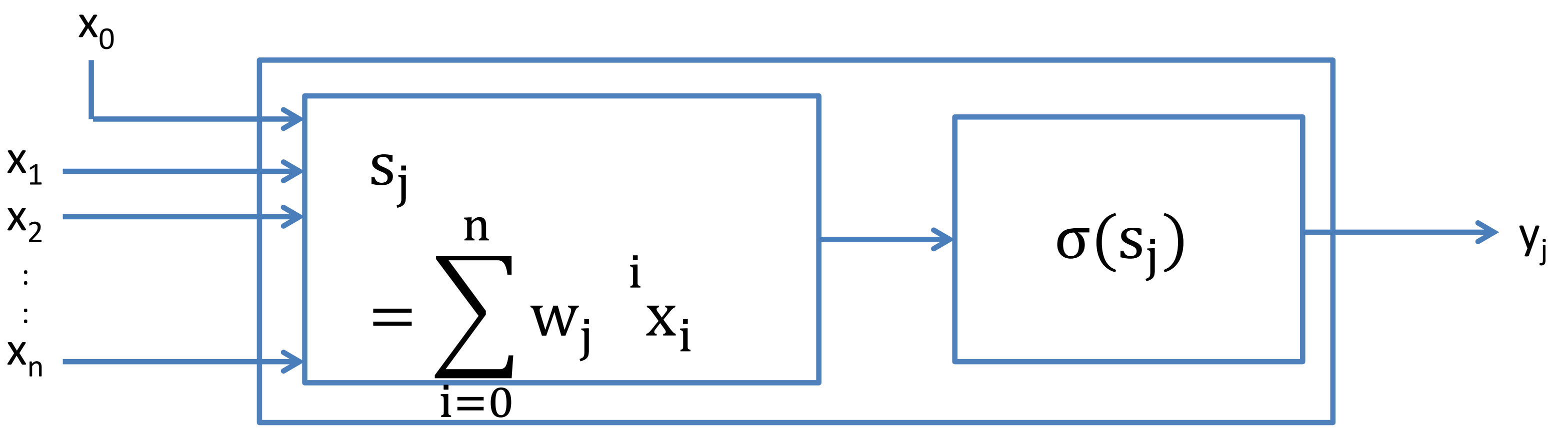
- Humans
  - Neuron switching time ~ 0.001 second
  - Number of neurons ~ 10^{11}
  - Connections per neuron ~ 10^{5}
  - Scene recognition time ~ 0.1 second
  - 100 inference steps doesn't seem enough
  - > much parallel computation
- Properties of artificial neural networks (ANN's)
  - Many neuron-like threshold witching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process

Artificial Neural Networks



- All nodes are involved in computation except for the input nodes, which simply send the input values to all nodes in the next layer

Artificial Neural Networks

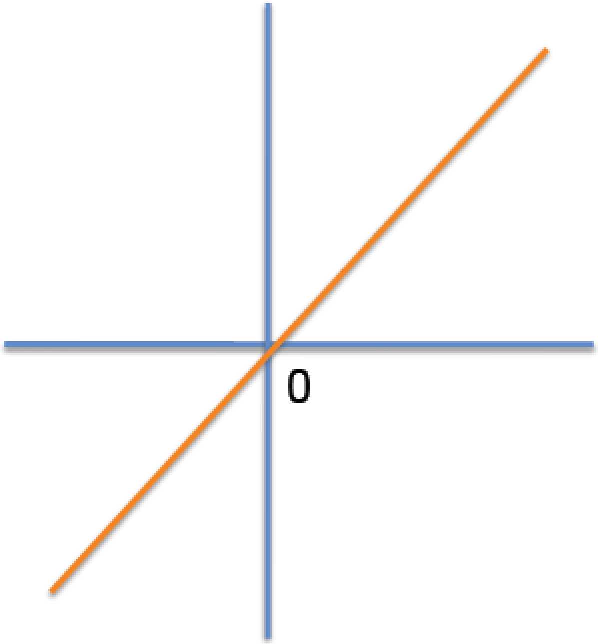


Node y_j

- $s_j = \sum_{i=0}^{n} w_j^i x_i = w_j^0 x_0 + w_j^1 x_1 + \cdots + w_j^n x_n$
- $x_0 = 1$, which is called "bias"
- $\sigma(s)$ is called the transfer function. There are various possibilities for $\sigma(s)$

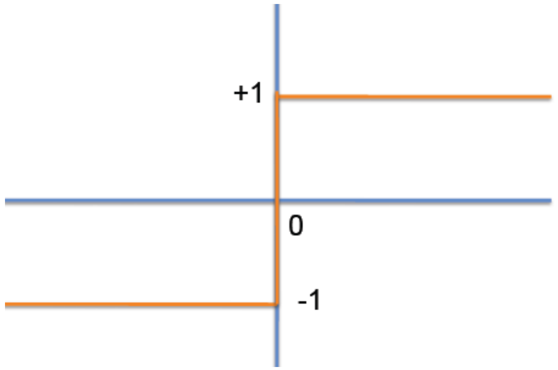
Transfer Function

- Linear function
  - $\sigma(s) = ks$, where k is a real number
  - y_j is simply a linear function of the input x_i
  - A form of linear regression

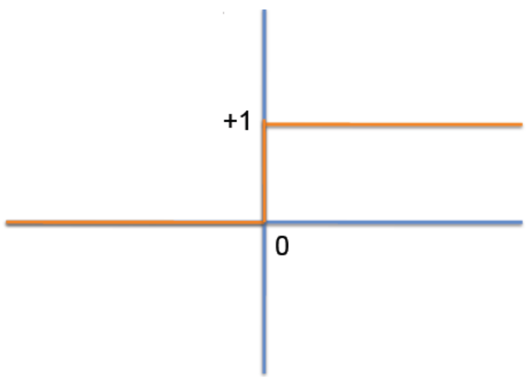


Transfer Function

- Step function (threshold function)
  - $\sigma(s) = \begin{cases} 1, & s > 0 \\ -1, & s \leq 0 \end{cases}$

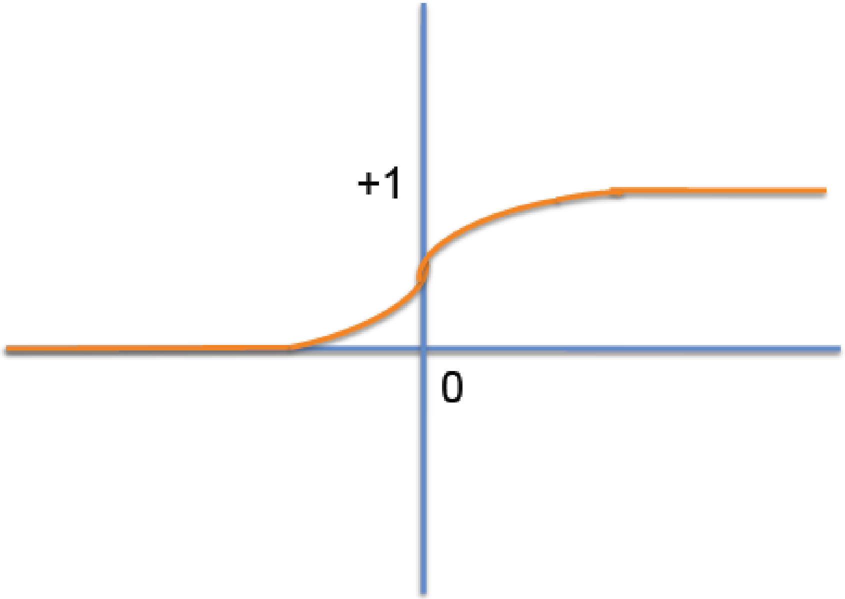


  - $\sigma(s) = \begin{cases} 1, & s > 0 \\ 0, & s \leq 0 \end{cases}$

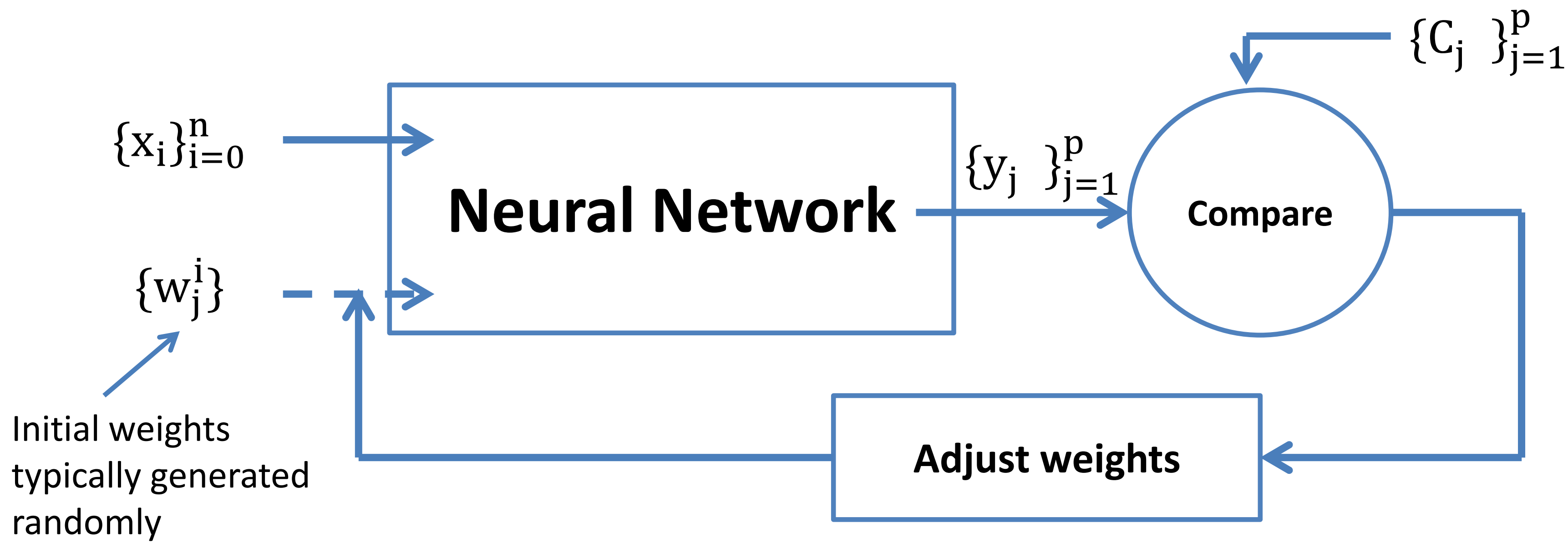


Transfer Function

- Sigmoid function
  - $\sigma(s) = \frac{1}{1+e^{-s}}$
  - Properties
    - **Differentiable function:** a function whose derivative exists at each point in its domain
    - $\sigma'(s) = \frac{d\sigma}{ds} = \frac{e^{-s}}{(1+e^{-s})^2} = \sigma(s)(1 - \sigma(s))$

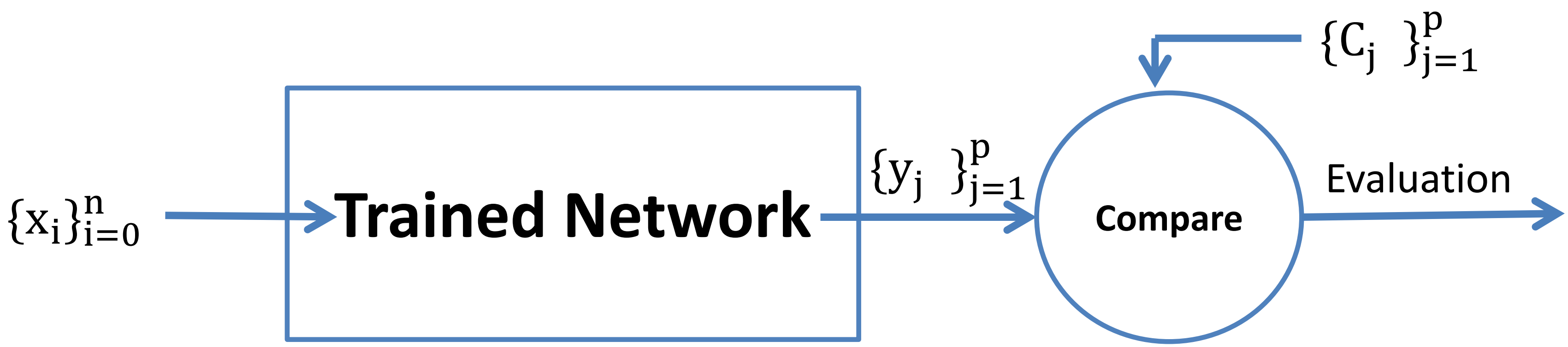


Training ANN



- Training involves a "training set", each member of the training set is a vector $\{x_i\}_{i=0}^{n}$ and an output $\{C_j\}_{j=1}^{p}$

Testing ANN



- Training involves a "test set", each member of the test set is a vector $\{x_i\}_{i=0}^{n}$
- Training data and test data are separate data sets. However, they should be drawn from the same distribution

Training ANN

- We want to minimize the error
  - Least square error: $\mathrm{E} = \frac{1}{2}\sum_{\mathrm{k}=1}^{\mathrm{p}}(\mathrm{y_k} \; - \mathrm{C_k}\;)^2$
- So E has to be minimized with respect to the weights $\{\mathrm{w_j^i}\}$
- We need $\frac{\partial E}{\partial \mathrm{w_j^i}}$ to discover how the error E depends on the $\{\mathrm{w_j^i}\}$

Back-propagation Algorithm

- Gradient descent over entire network weight vector
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Minimizes error over training examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations. Slow!
- Using network after training is fast

Overfitting

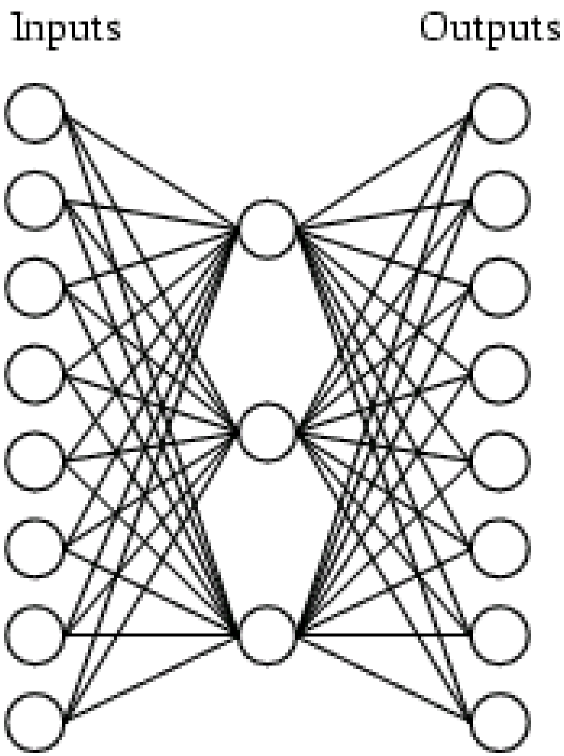
- ANNs are supervised learning
  - Every supervised learning has risks of overfitting

- Training involves iterative weight updating. The number of iterations, n, is important
  - How do we choose n to minimize the error rate over future data?
  - We use cross validation

Expressive Capability 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 nodes
- Continuous functions
  - Every bounded continuous function can be approximated with arbitrarily 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]

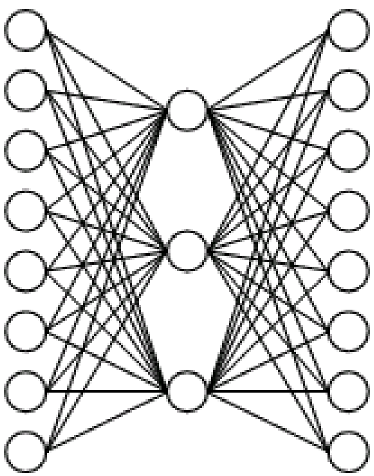
Example



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

Can this be learned?

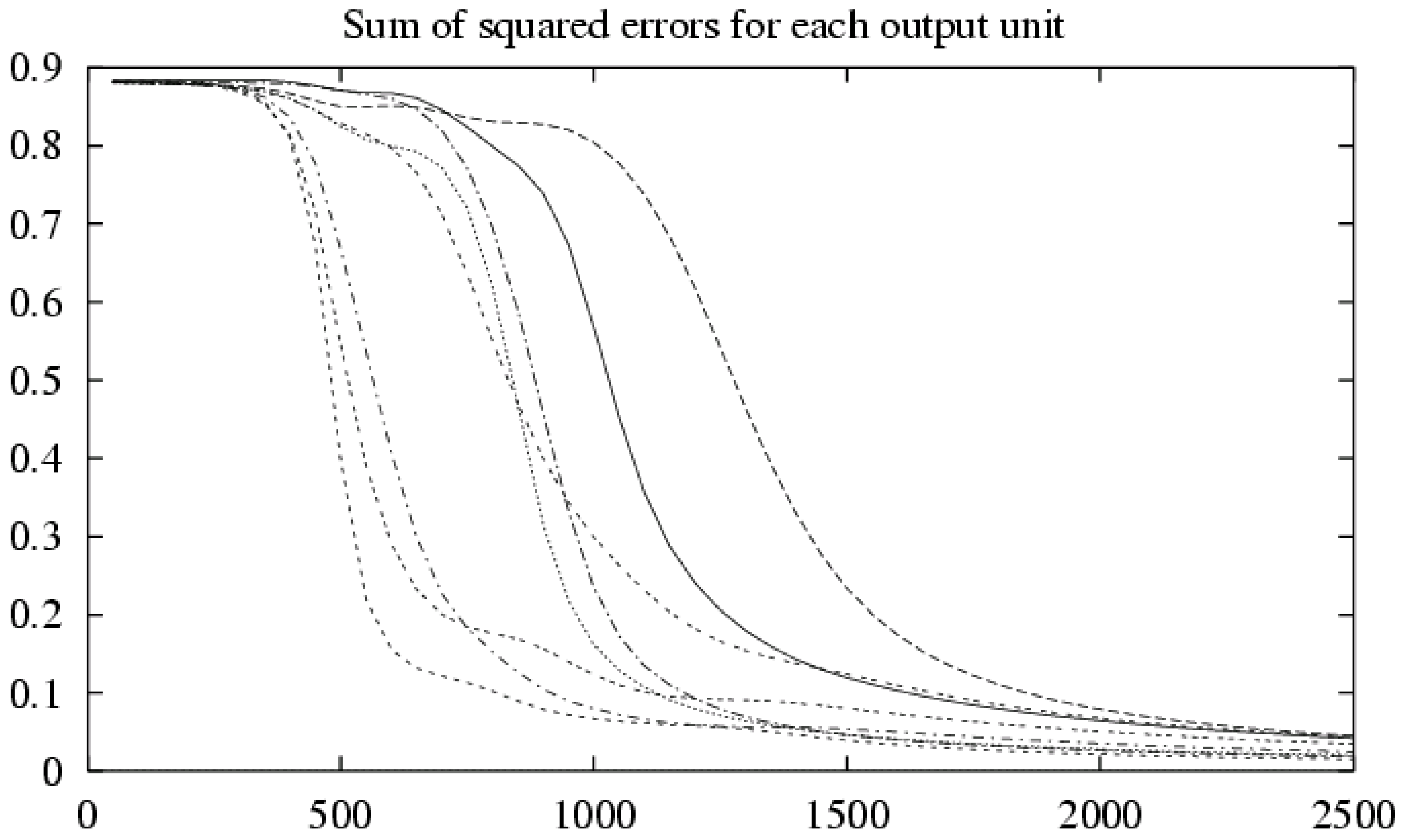
Example



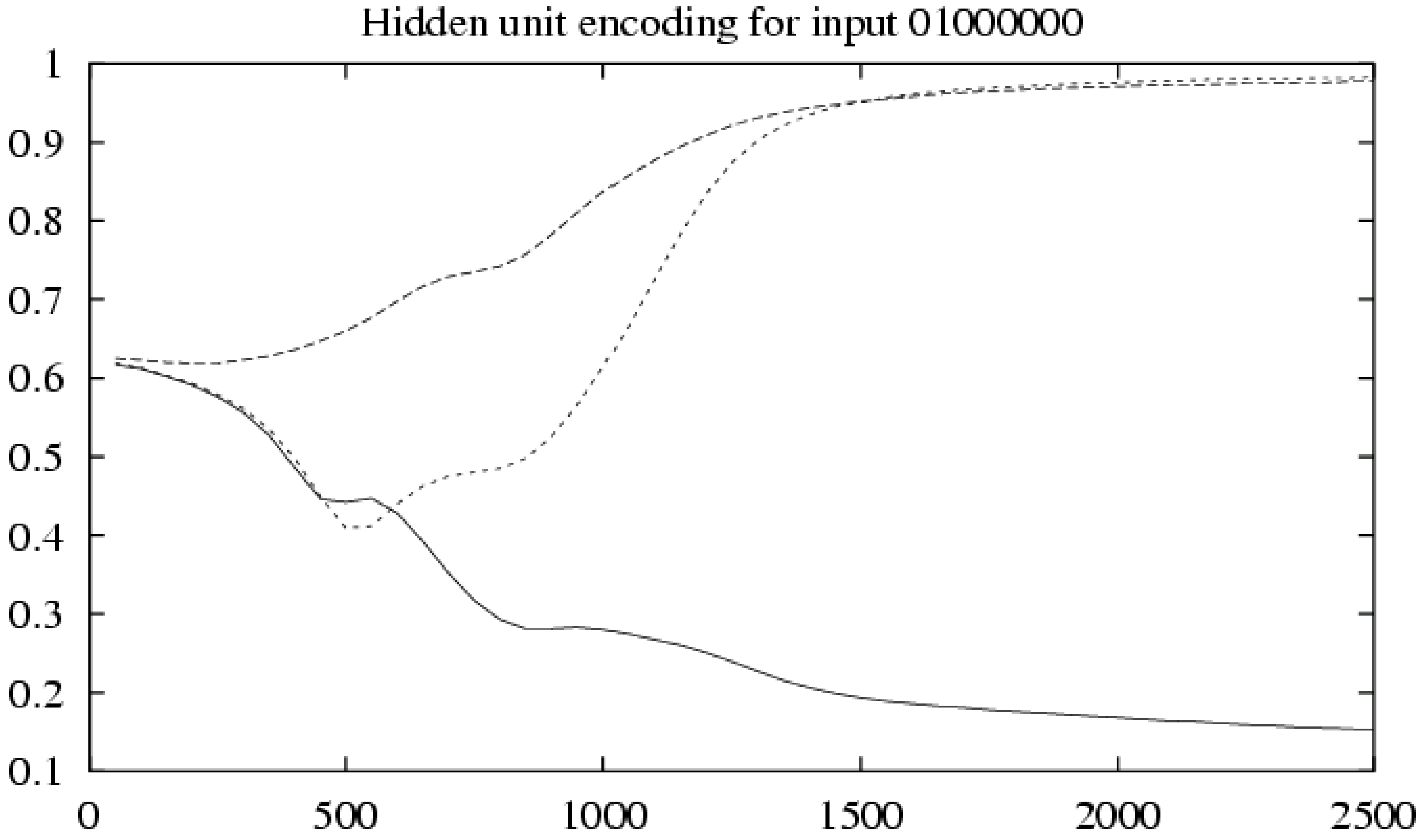
Learned hidden layer representation

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

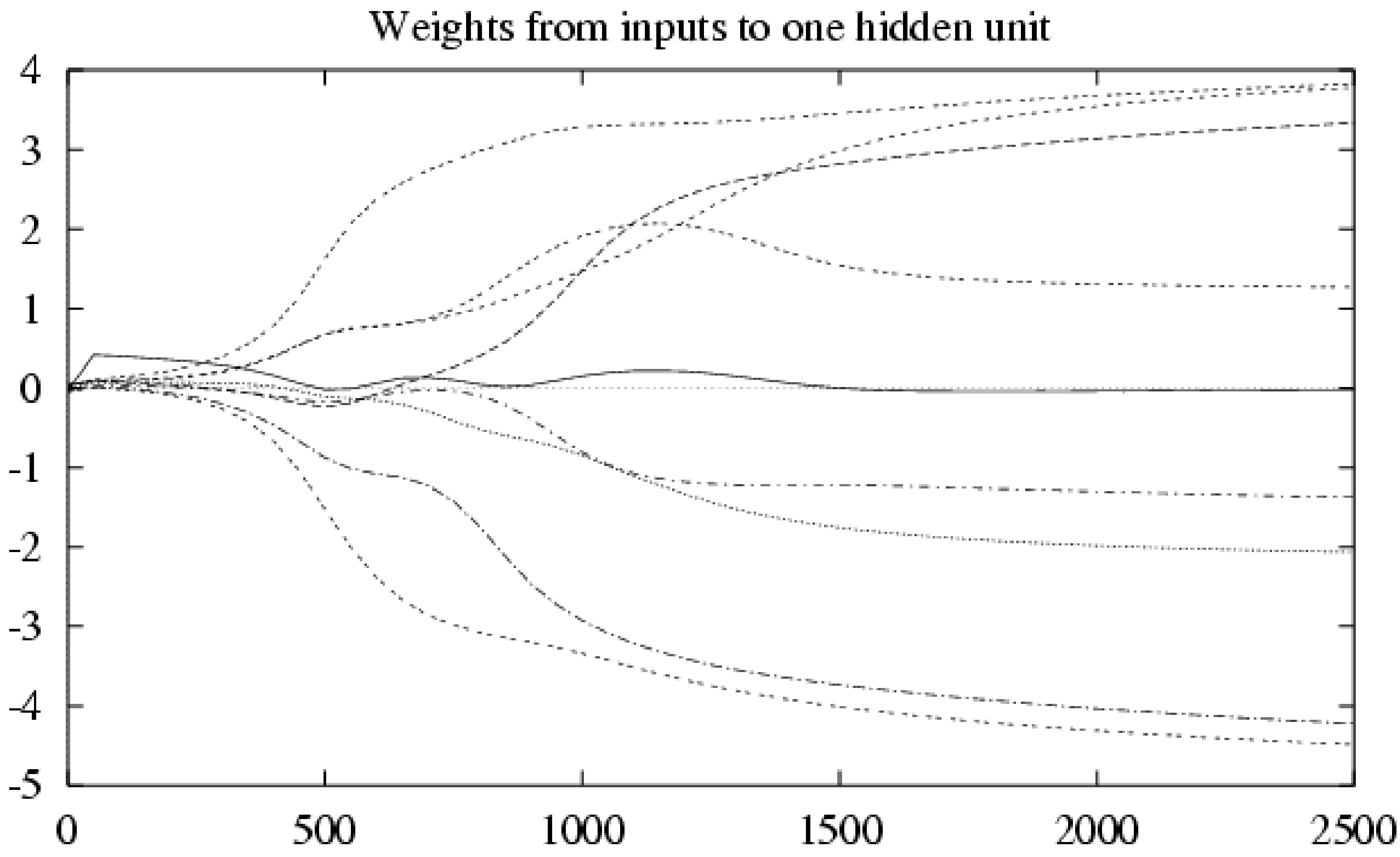
Example


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

Example


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

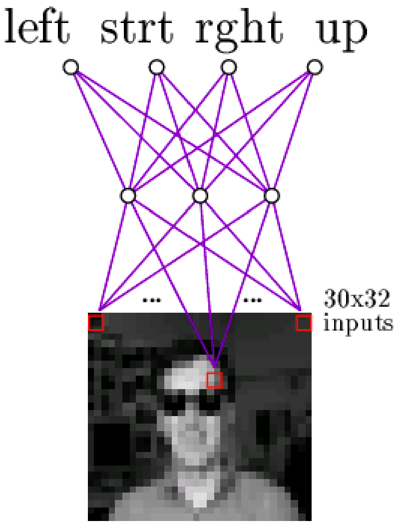
Example



Weights from inputs to one hidden unit
4
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500

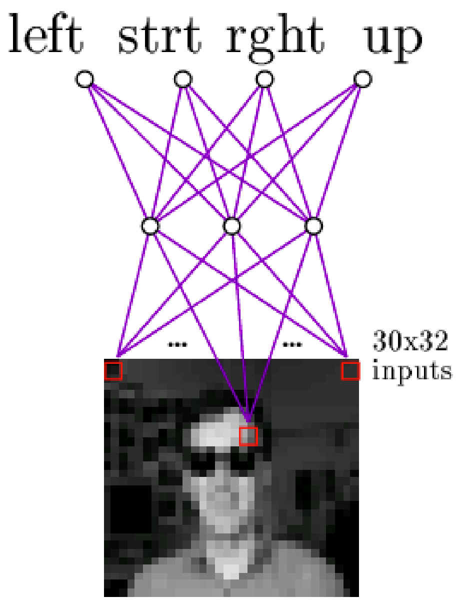
Another Example

- Neural network based face recognition

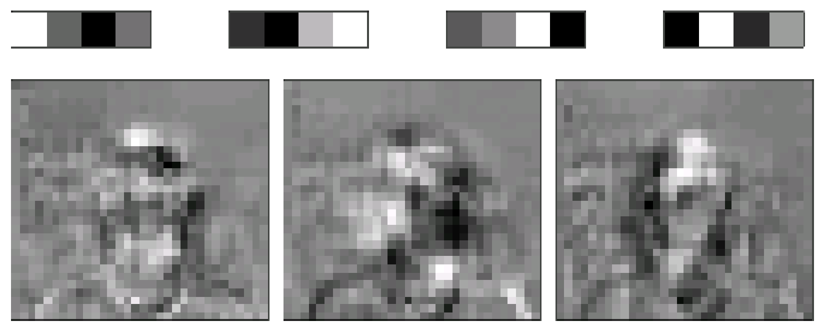


Typical input images

Another Example

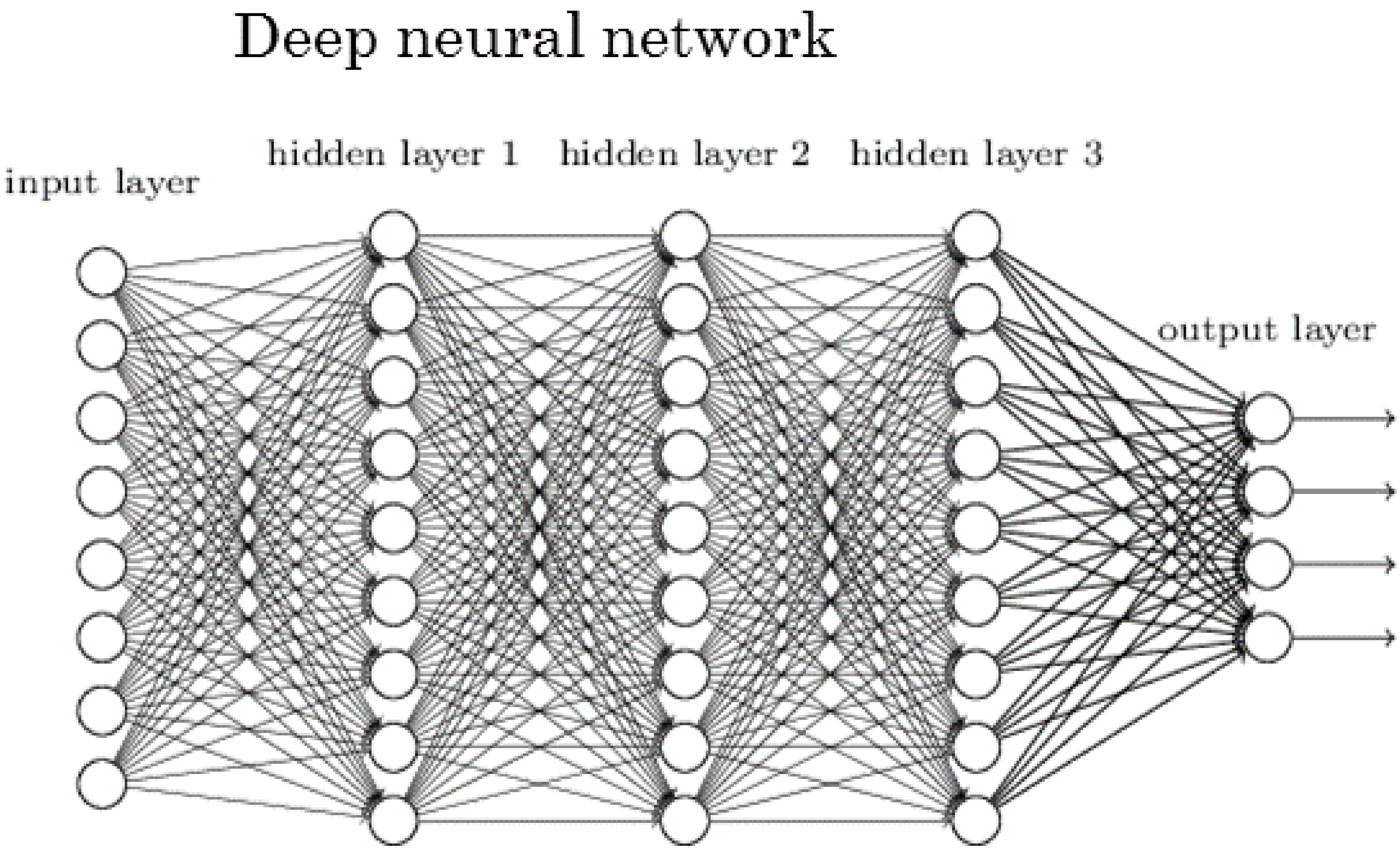


Learned Weights



Typical input images

Detour to Deep Neural Network



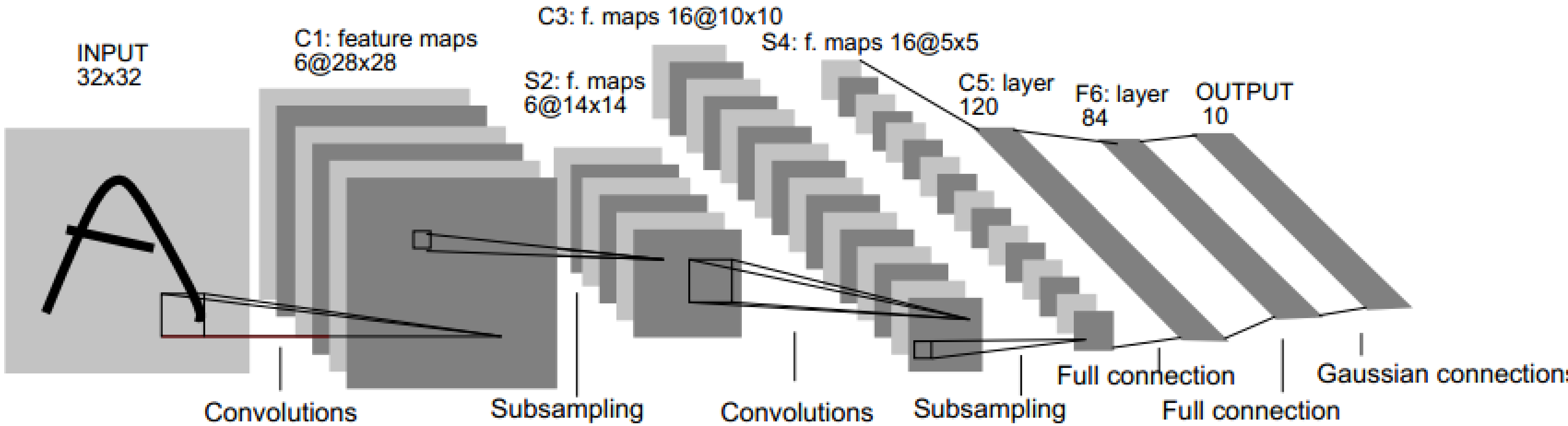
What’s the Problem?

What’s the Problem?

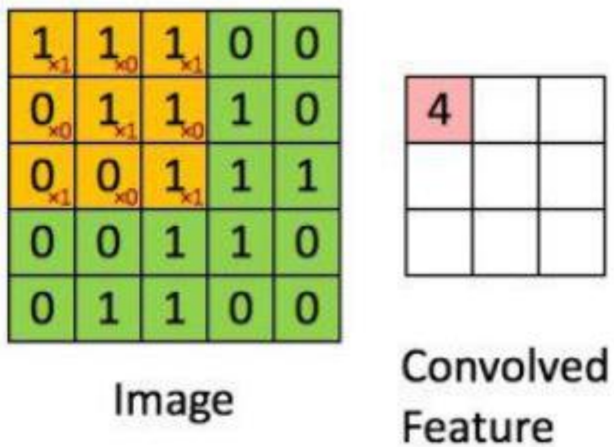
Two Approaches in Supervised Learning

- Do we use the prediction performance to guide the search?
  - NO → **Filter**
  - Yes → **Wrapper**

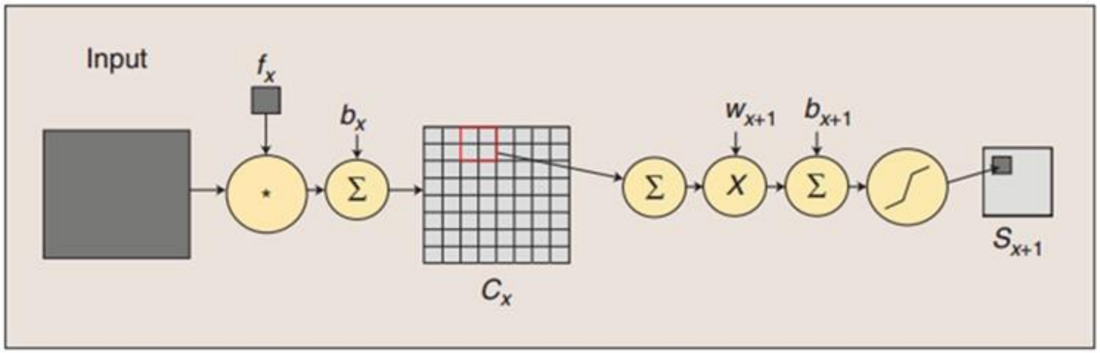
Deep Learning – Convolutional Neural Network



LeNet-5



Convolution

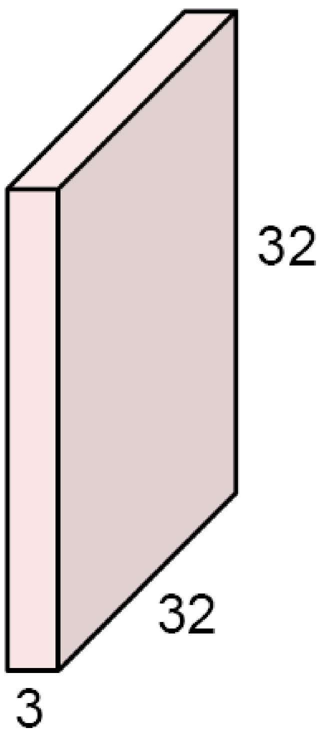


Pooling

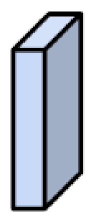
Convolutional Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter



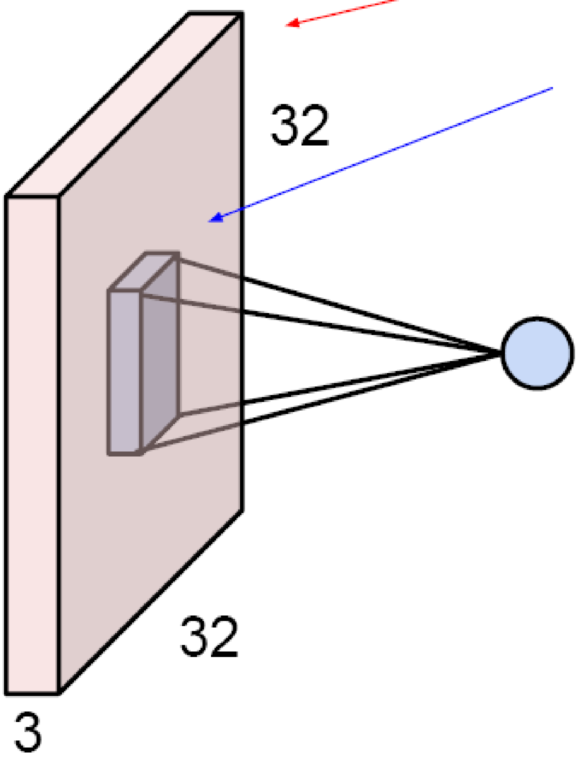
Convolve the filter with the image
i.e. "slide over the image spatially, computing dot products"

Courtesy from Feifei Li

Convolutional Layer

32x32x3 image

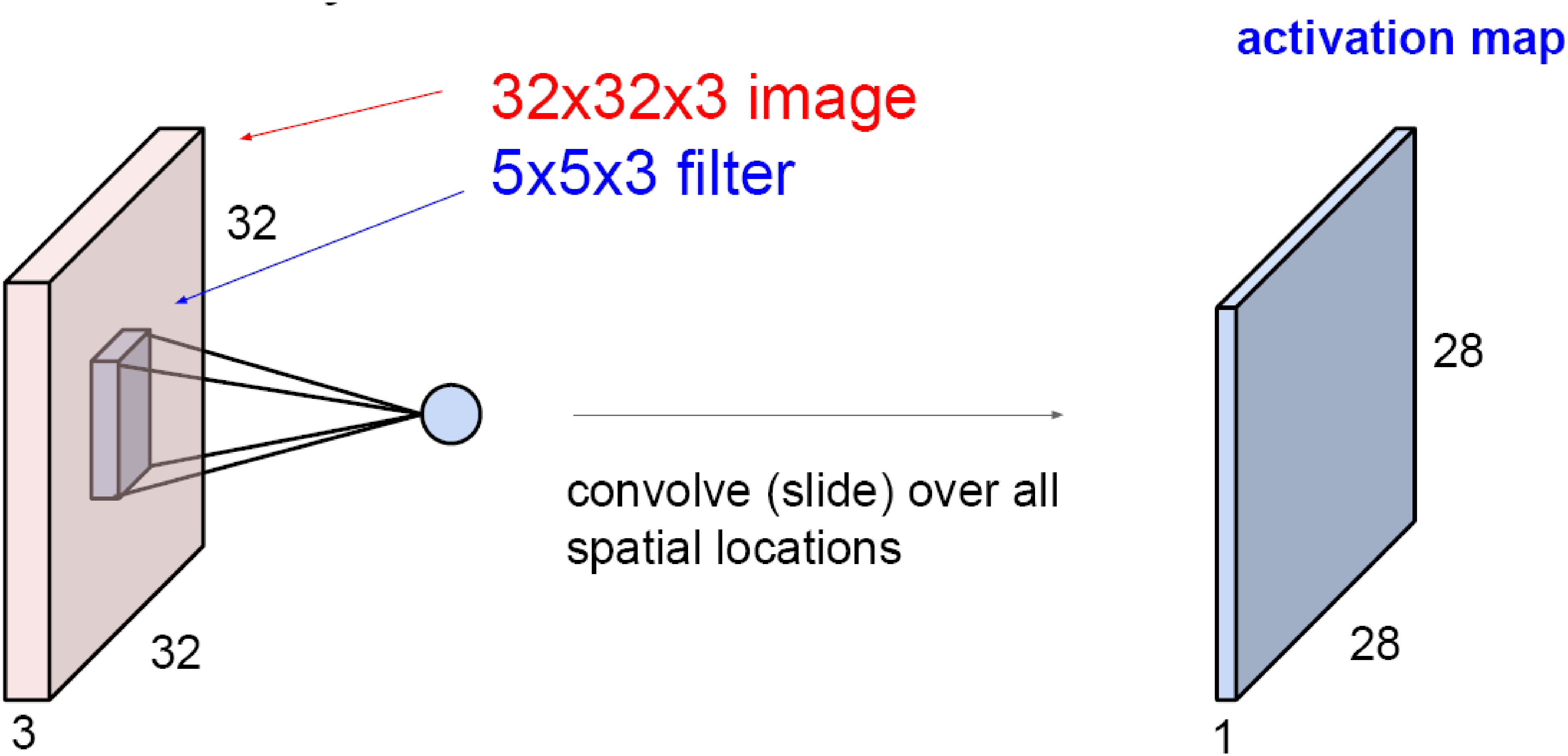
5x5x3 filter w



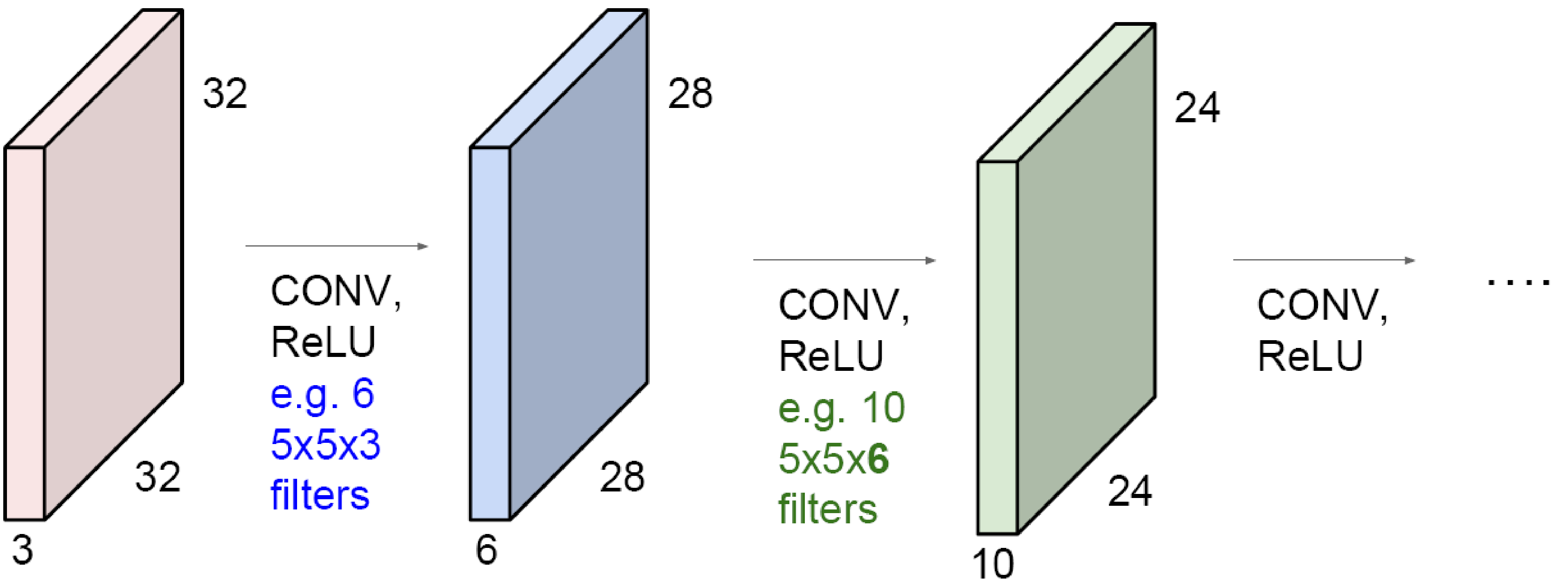
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

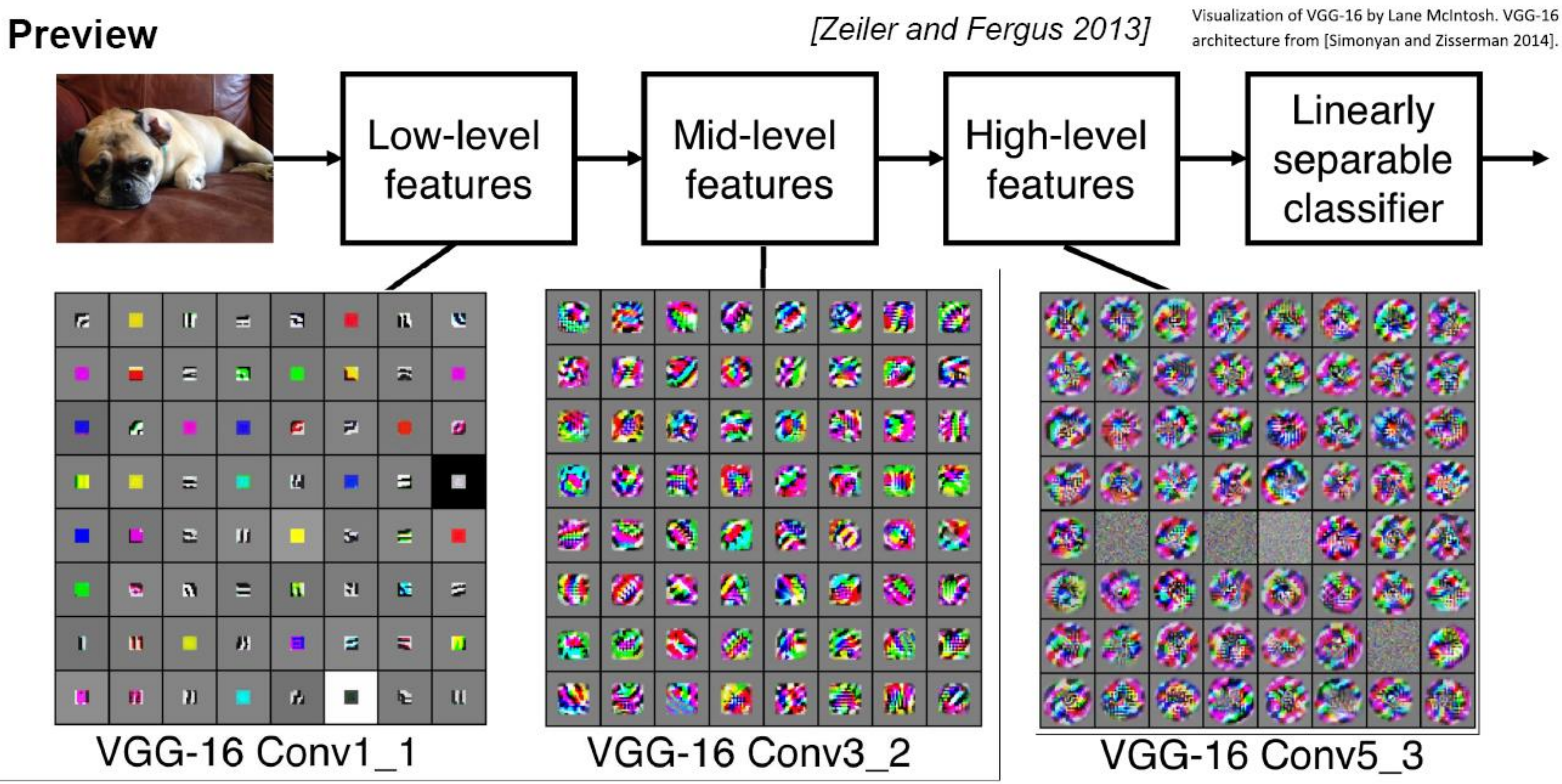
Convolutional Layer



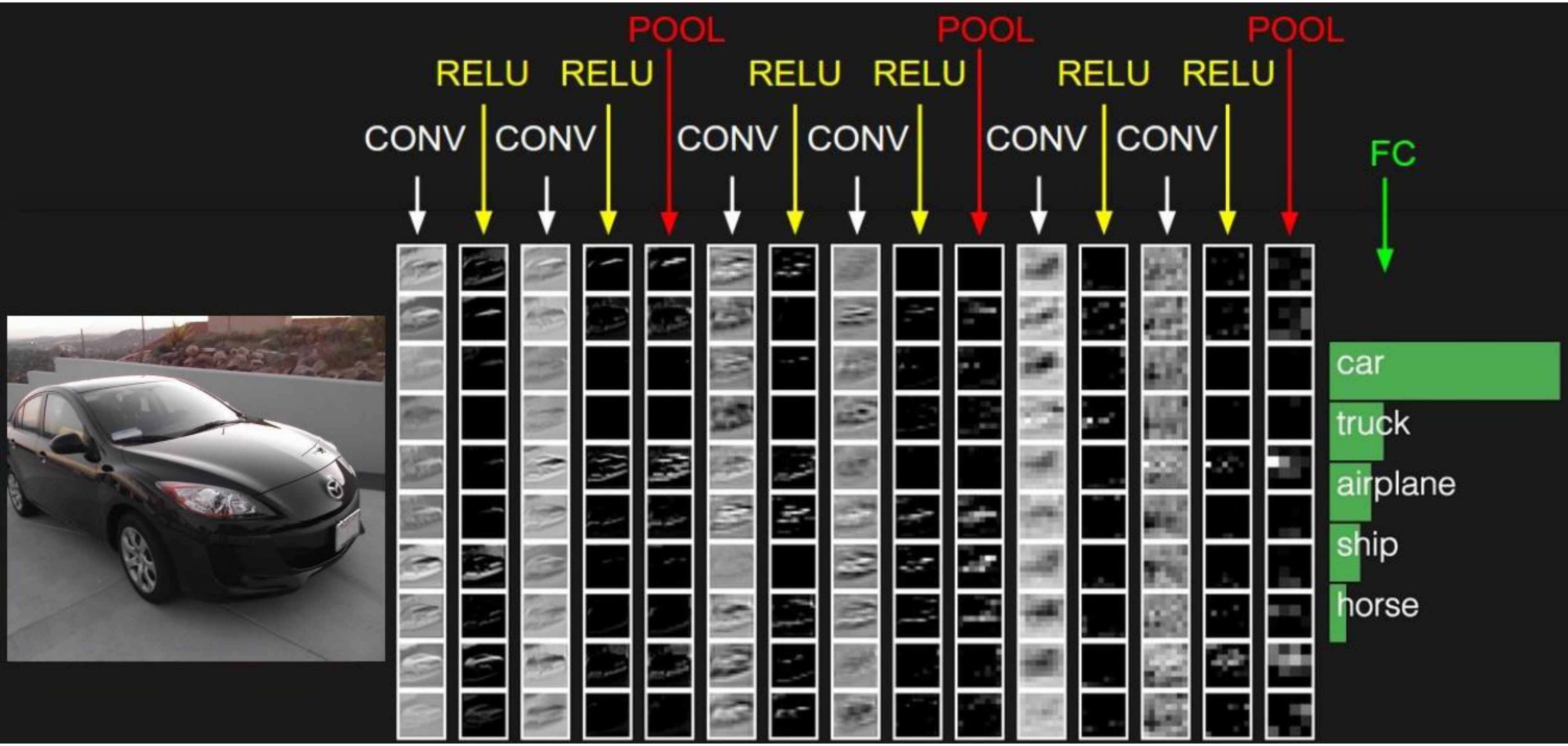
Convolutional Layer



Convolutional Neural Network



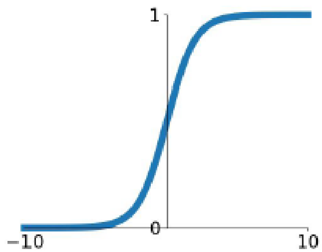
Convolutional Neural Network



Activation Function

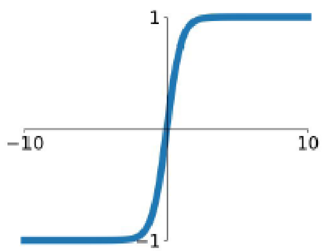
Sigmoid

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



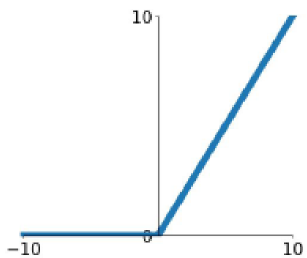
tanh

$\tanh(x)$



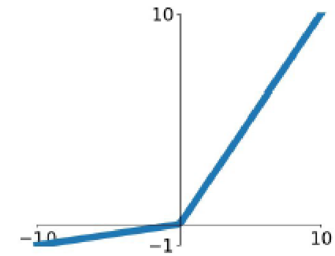
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

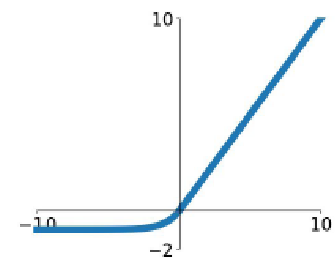


Maxout

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

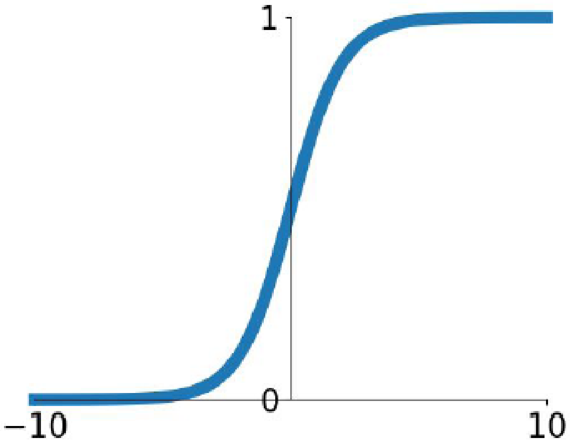
ELU

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



Activation Function

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



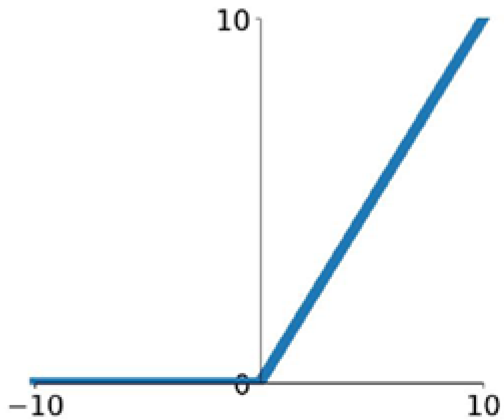
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

3 problems:

1. Saturated neurons "kill" the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

Activation Function

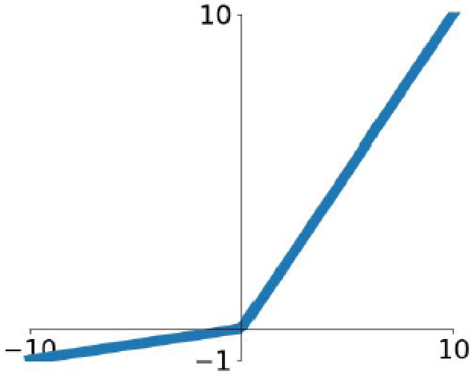


ReLU
(Rectified Linear Unit)

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

- Not zero-centered output

Activation Function

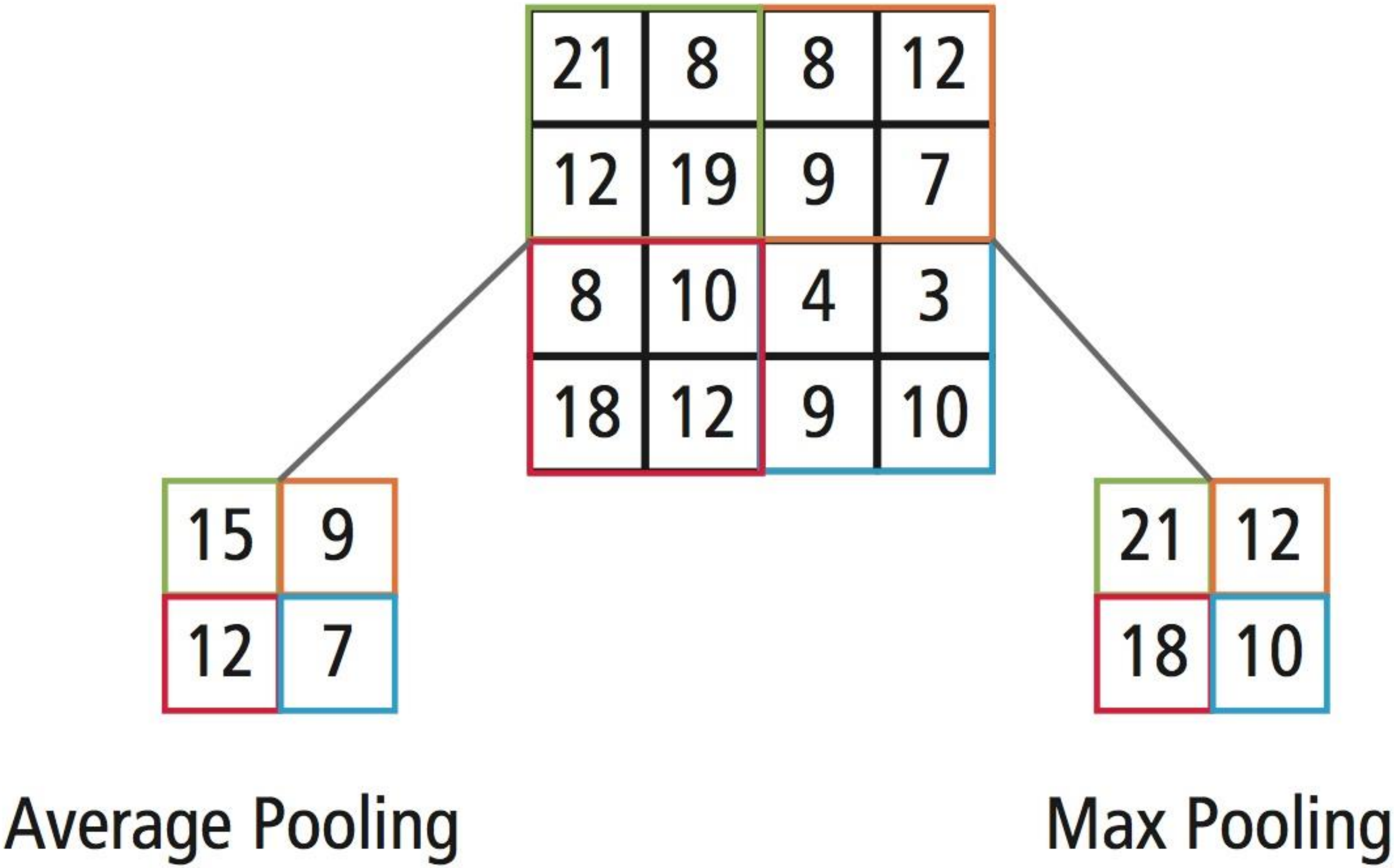


- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

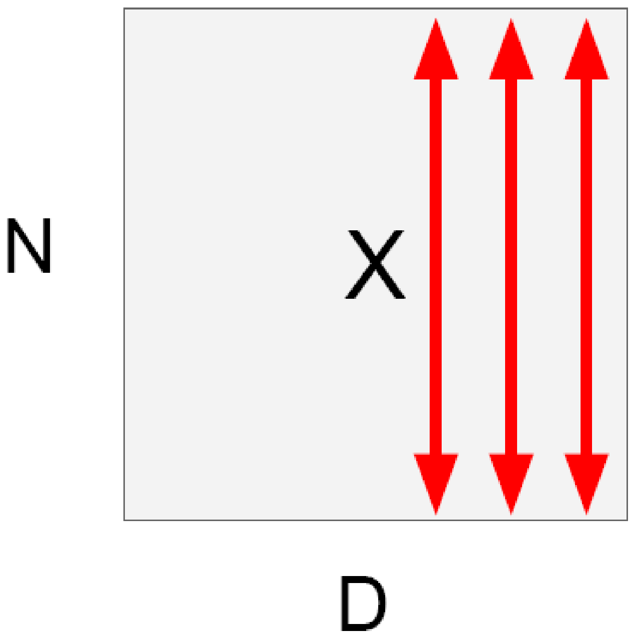
Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Pooling


21
8
8
12
12
19
9
7
8
10
4
3
18
12
9
10
15
9
12
7
21
12
18
10
Average Pooling
Max Pooling

Batch Normalization



1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Softmax

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

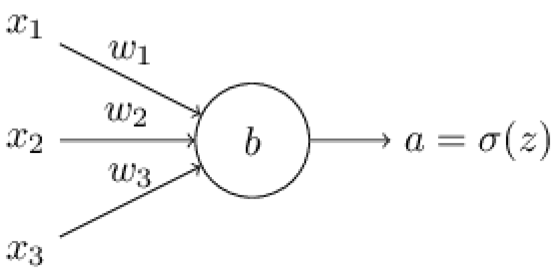
$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Loss Function – Cross-entropy Loss

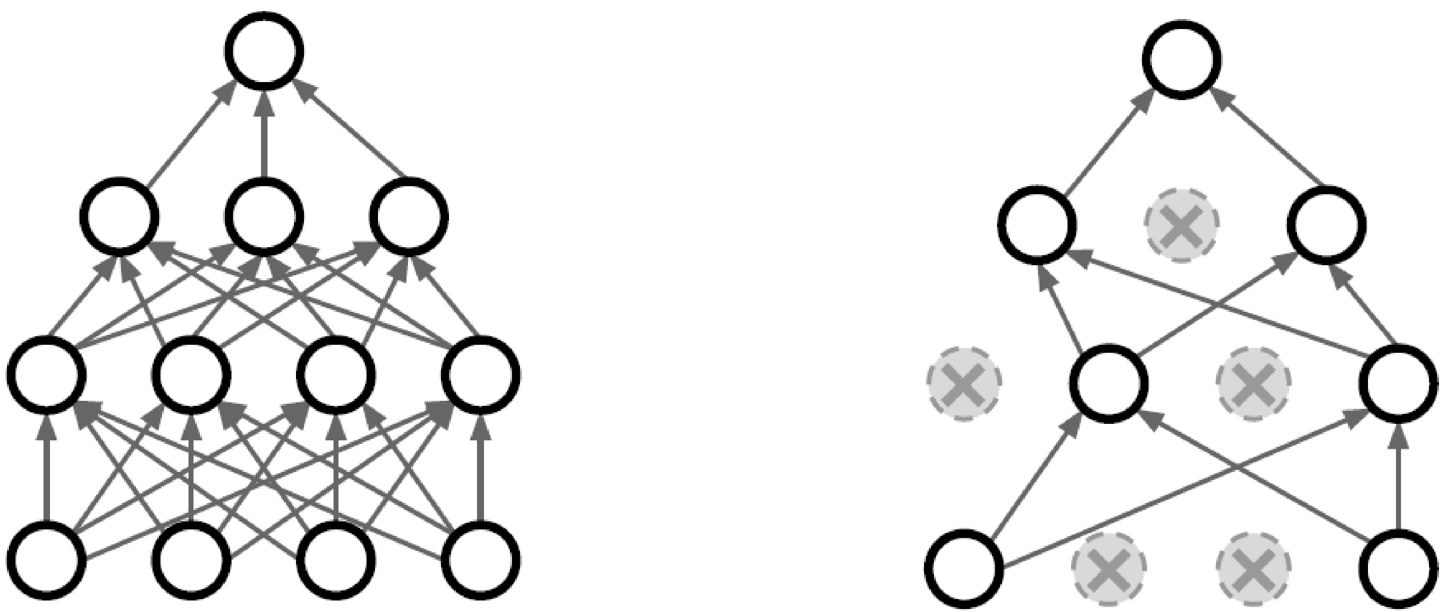


$$C = -\frac{1}{n} \sum_x \left[y \ln a + (1 - y) \ln(1 - a) \right]$$

Cross entropy is always larger than entropy; encoding symbols according to the wrong distribution will always make us use more bits.

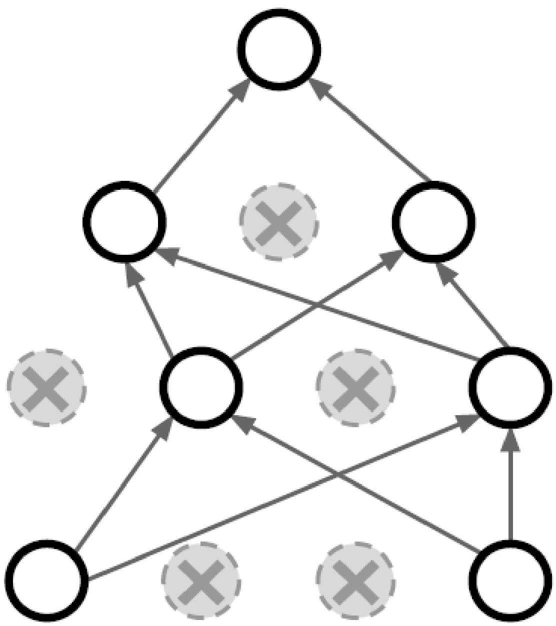
Regularization - Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



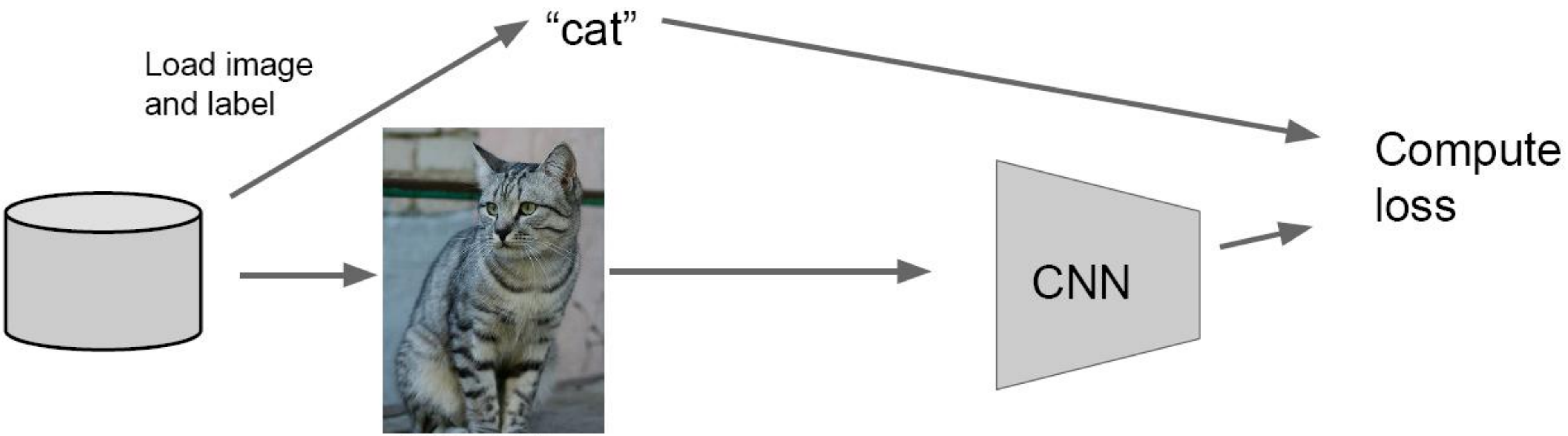
Regularization - Dropout

How can this possibly be a good idea?

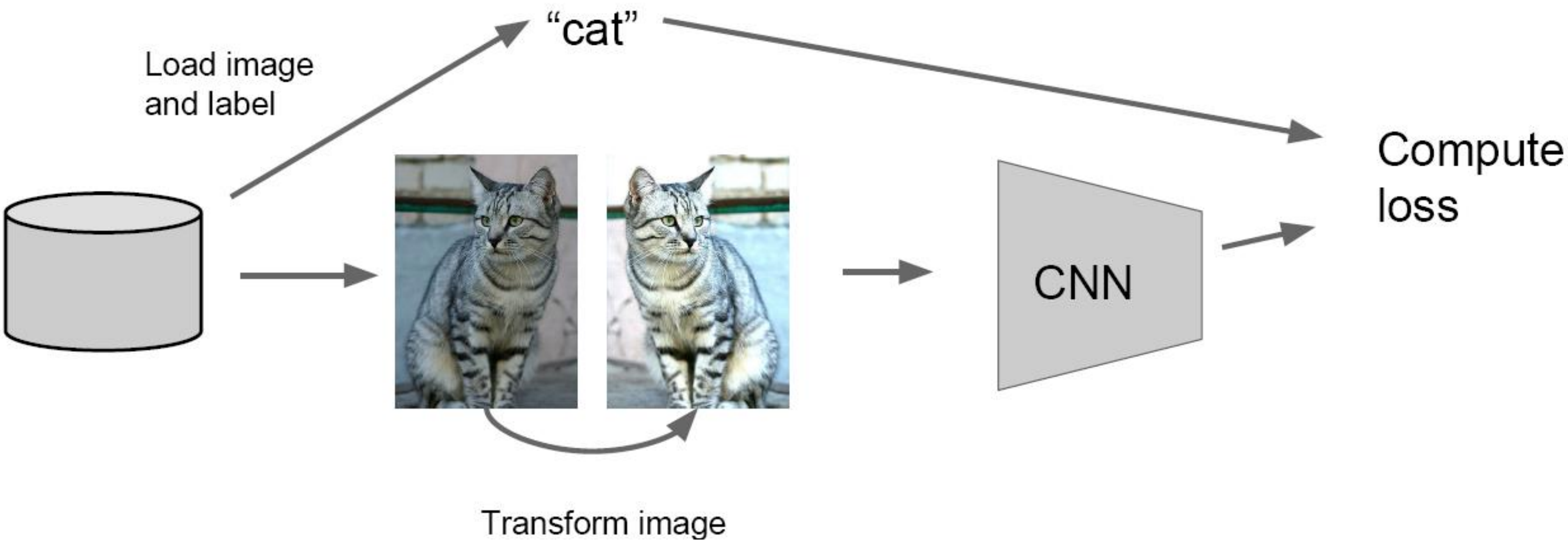


Forces the network to have a redundant representation;
Prevents co-adaptation of features

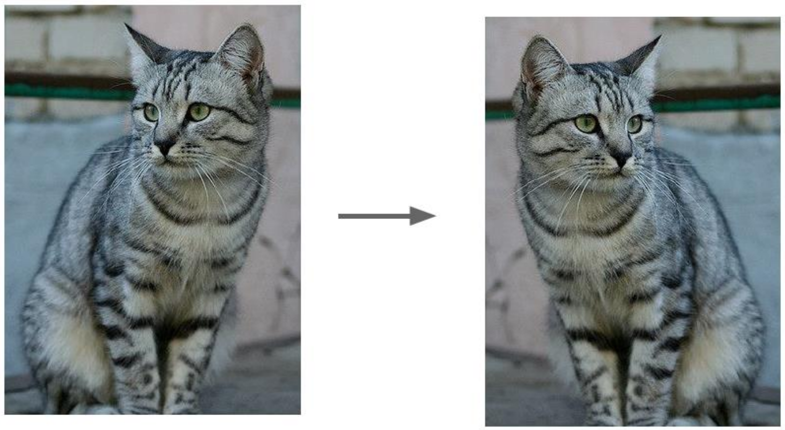
Regularization – Data Augmentation



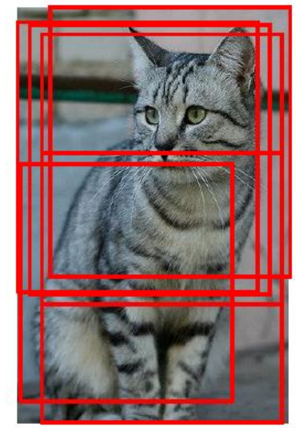
Regularization – Data Augmentation



Regularization – Data Augmentation

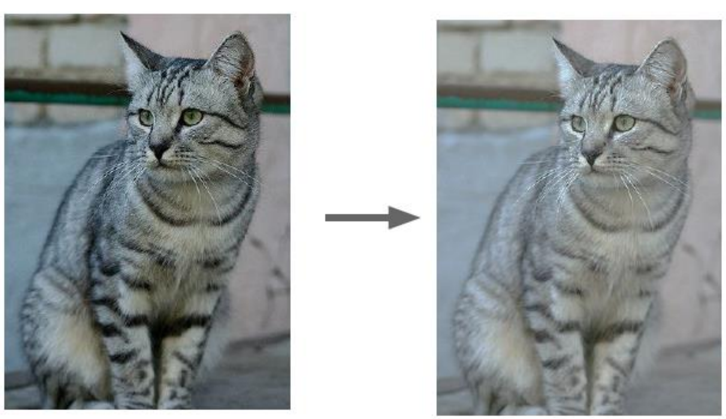


Horizontal flips



Random crops and scale

Simple: Randomize contrast and brightness



Color jitter

Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Stochastic Gradient Descent

In stochastic (or "on-line") gradient descent, the true gradient is approximated by a gradient at a single example.

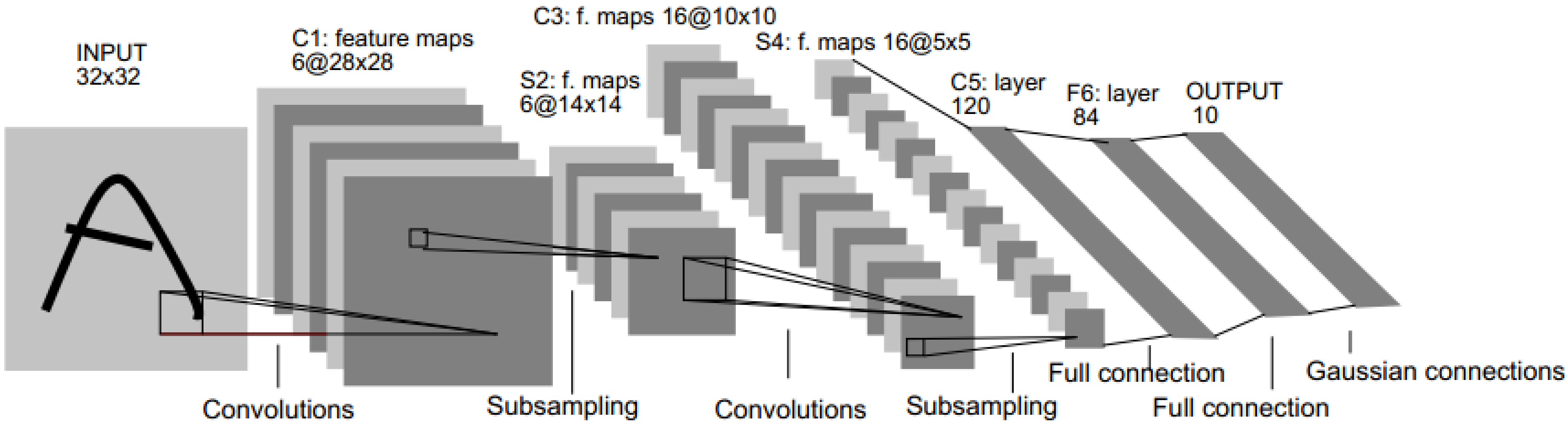
- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta \nabla Q_i(w)$.

Gradient descent: use all examples in each iteration
Stochastic gradient descent: use 1 example in each iteration
Mini-batch gradient descent: use b examples in each iteration

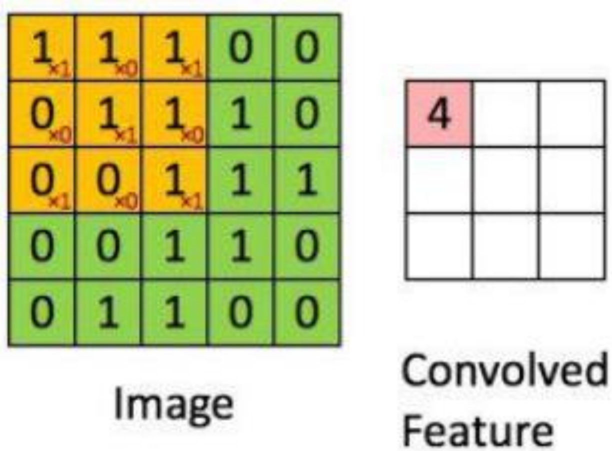
Representative CNN Networks

- LeNet-5
- AlexNet
- VGG
- Autoencoder
- ResNet
- GAN

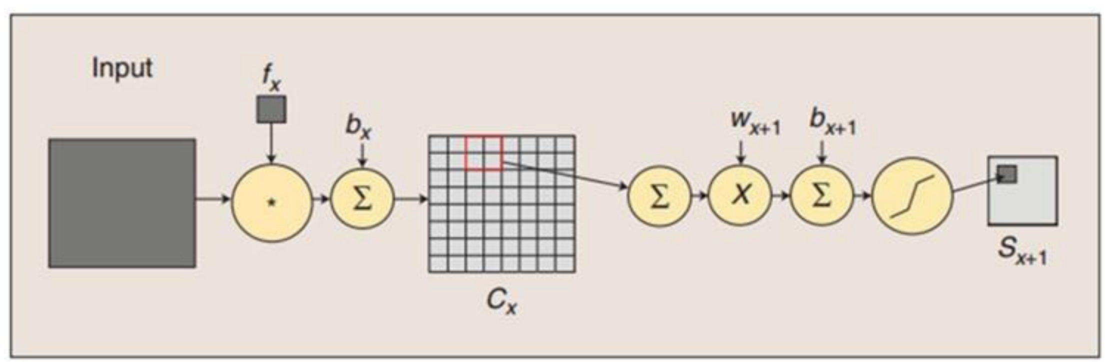
LeNet-5



LeNet-5



Convolution

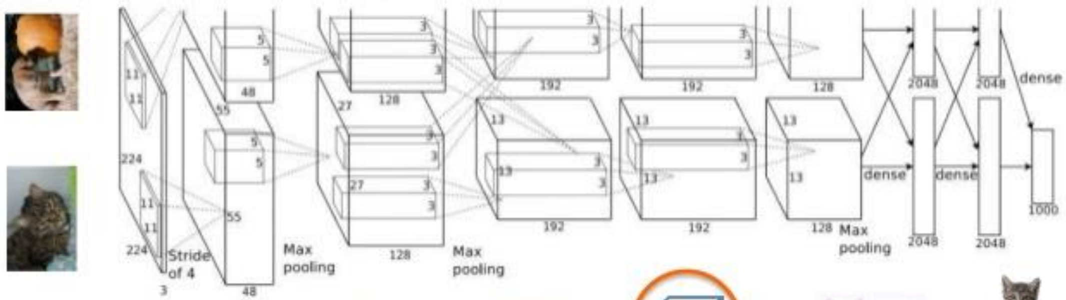


Pooling

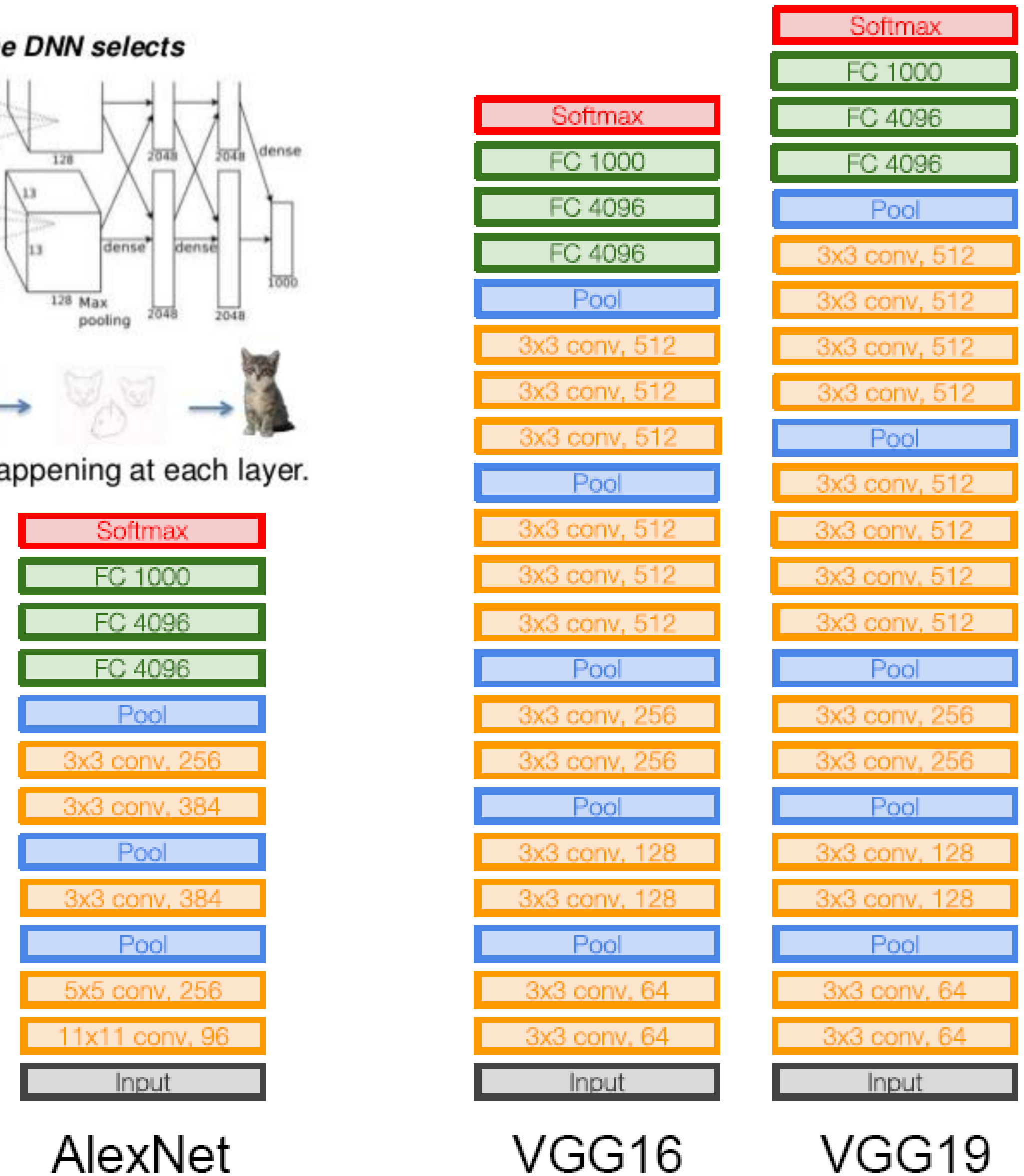
AlexNet and VGG

AlexNet (Krizhevsky et al. 2012)

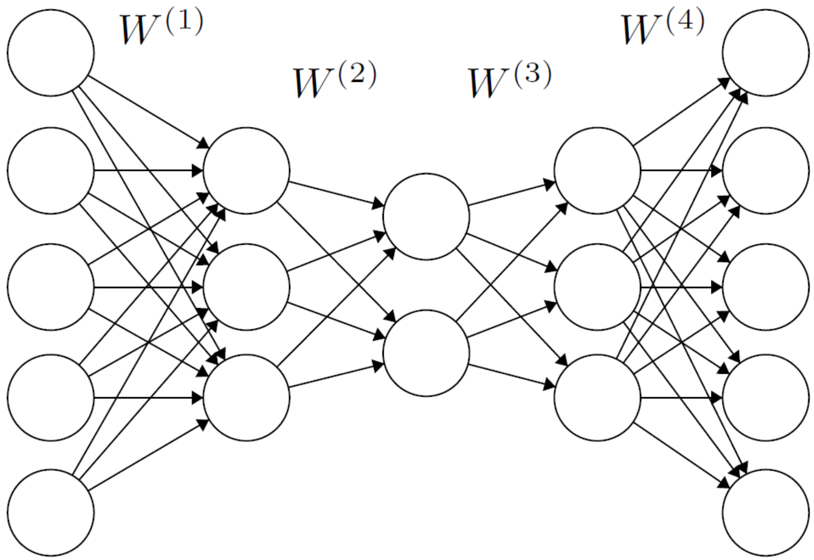
The class with the highest likelihood is the one the DNN selects

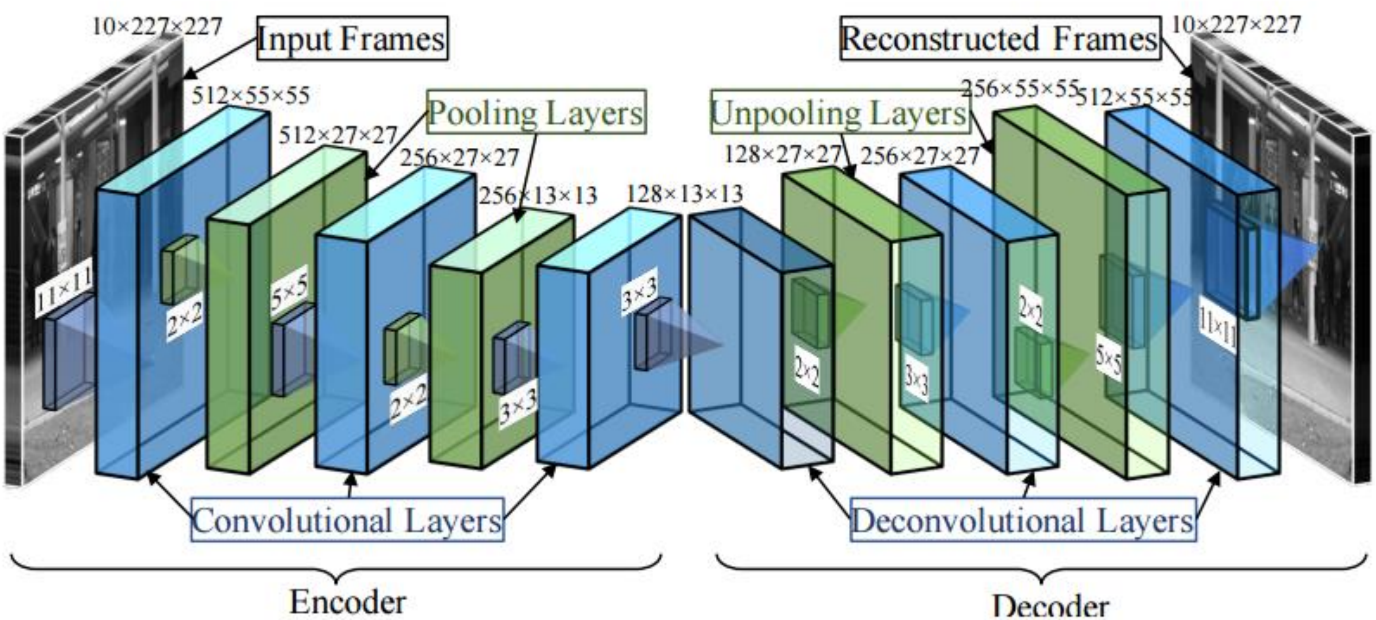


When AlexNet is processing an image, this is what is happening at each layer.



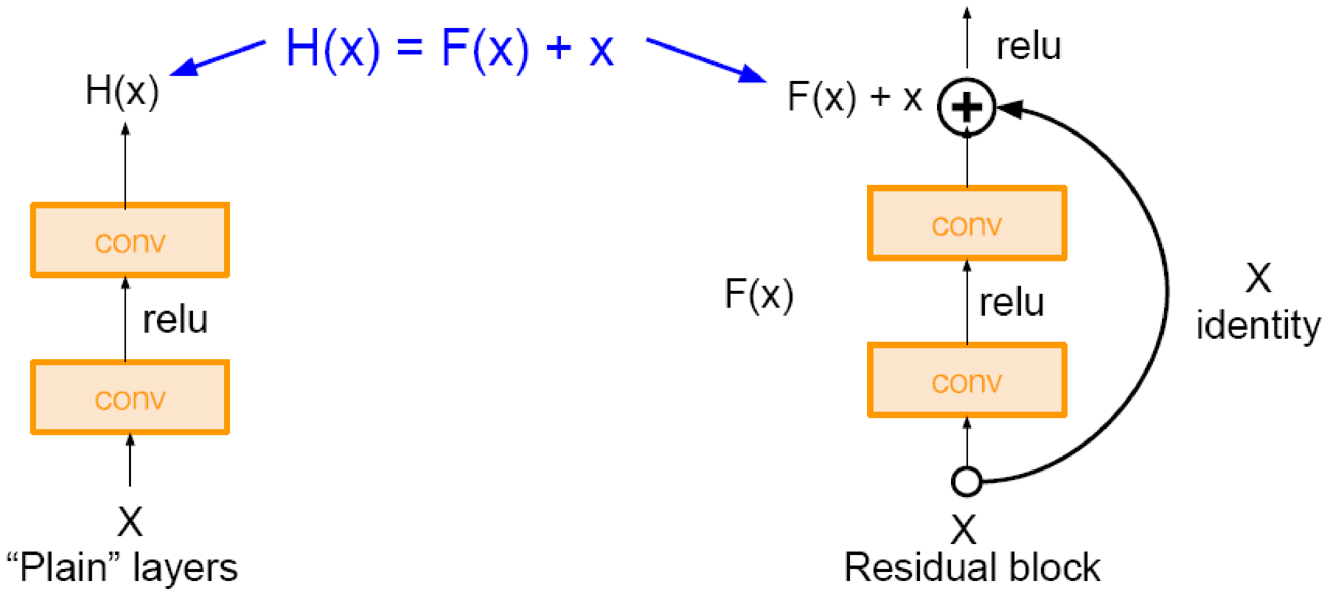
Autoencoder





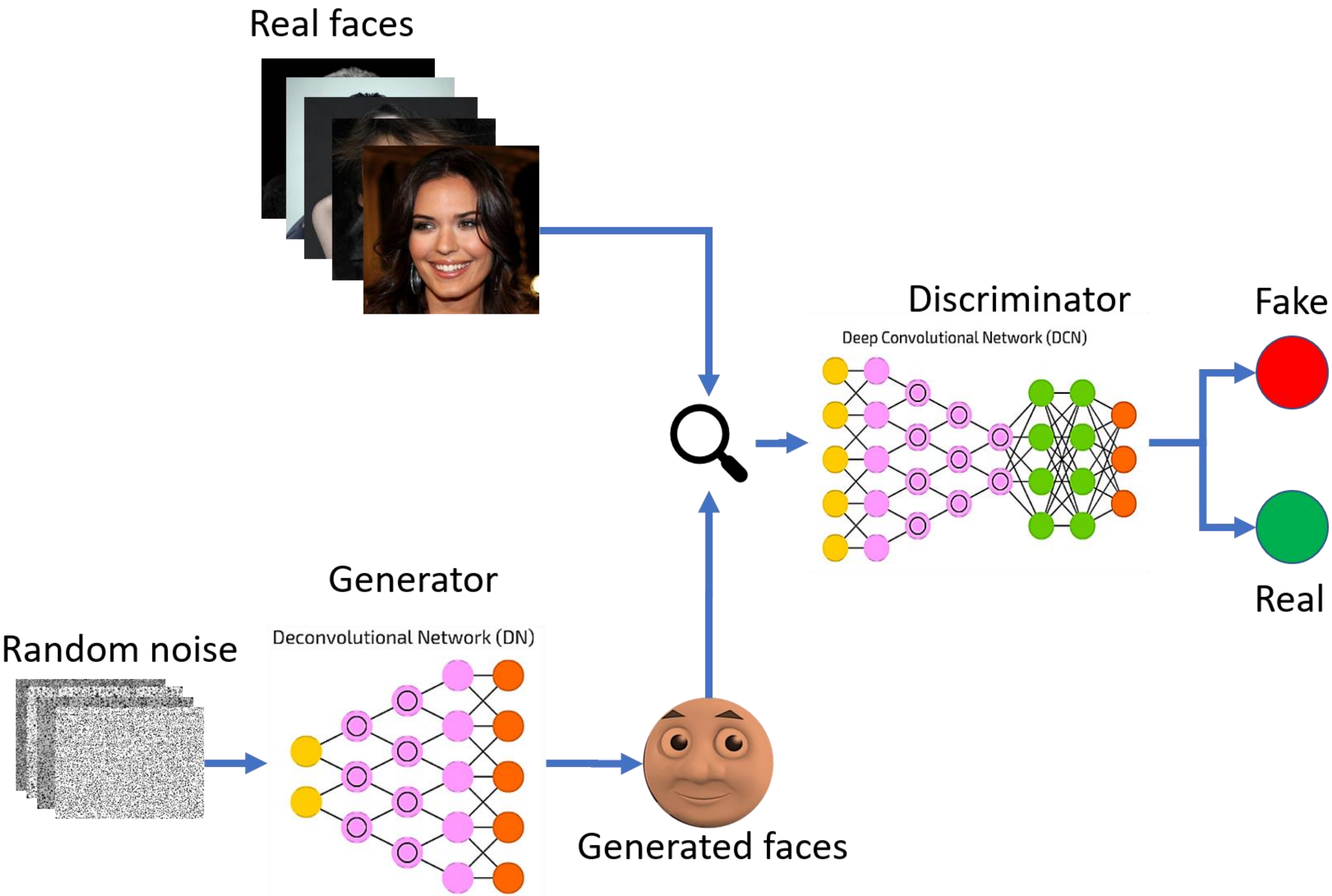
ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

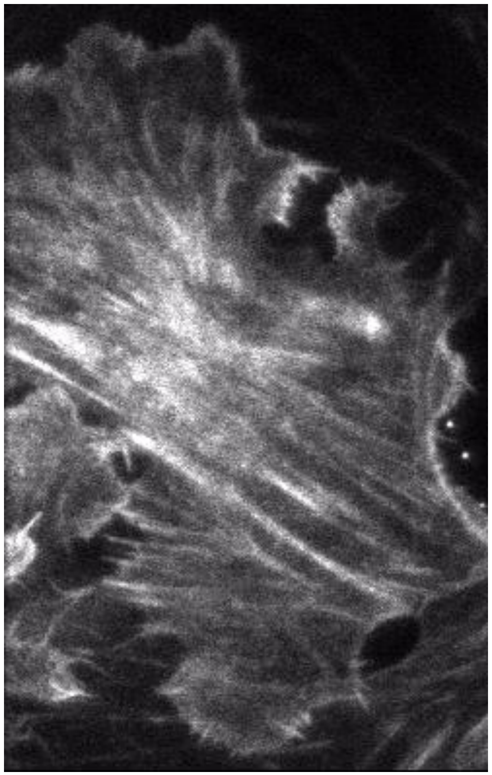


Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

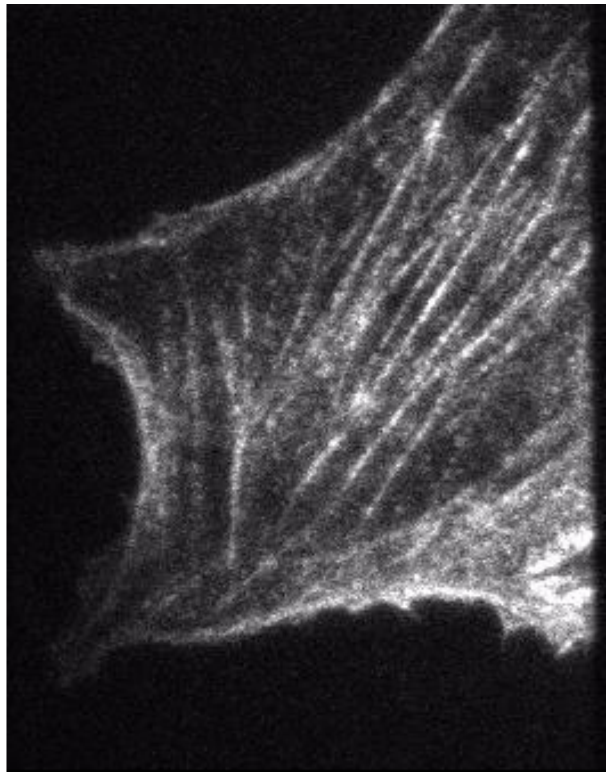
Generative Adversarial Network



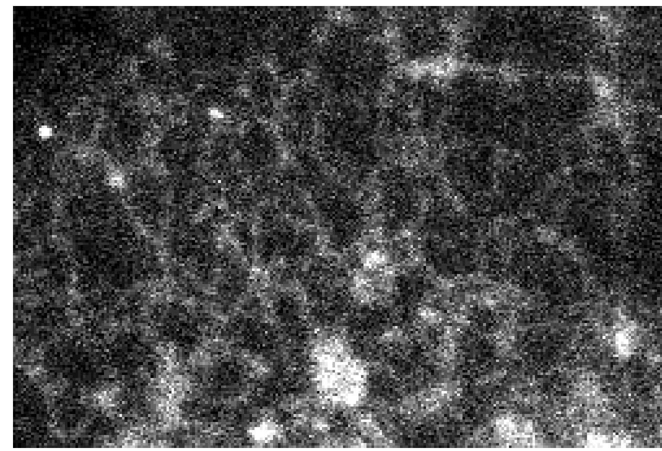
Fluorescence Microscopy



U2OS

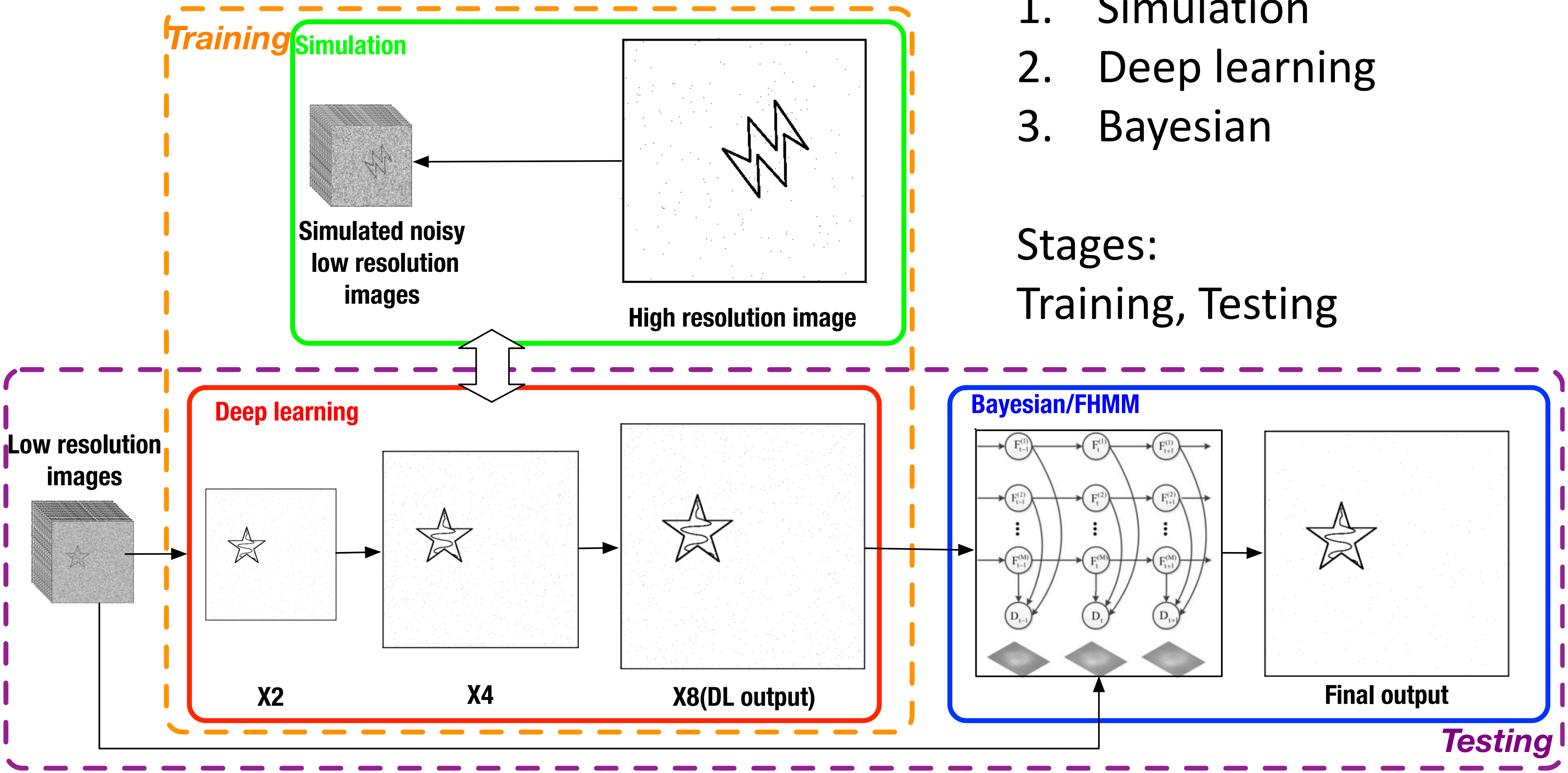


Actin



Endo

Overview of DLBI



Components:

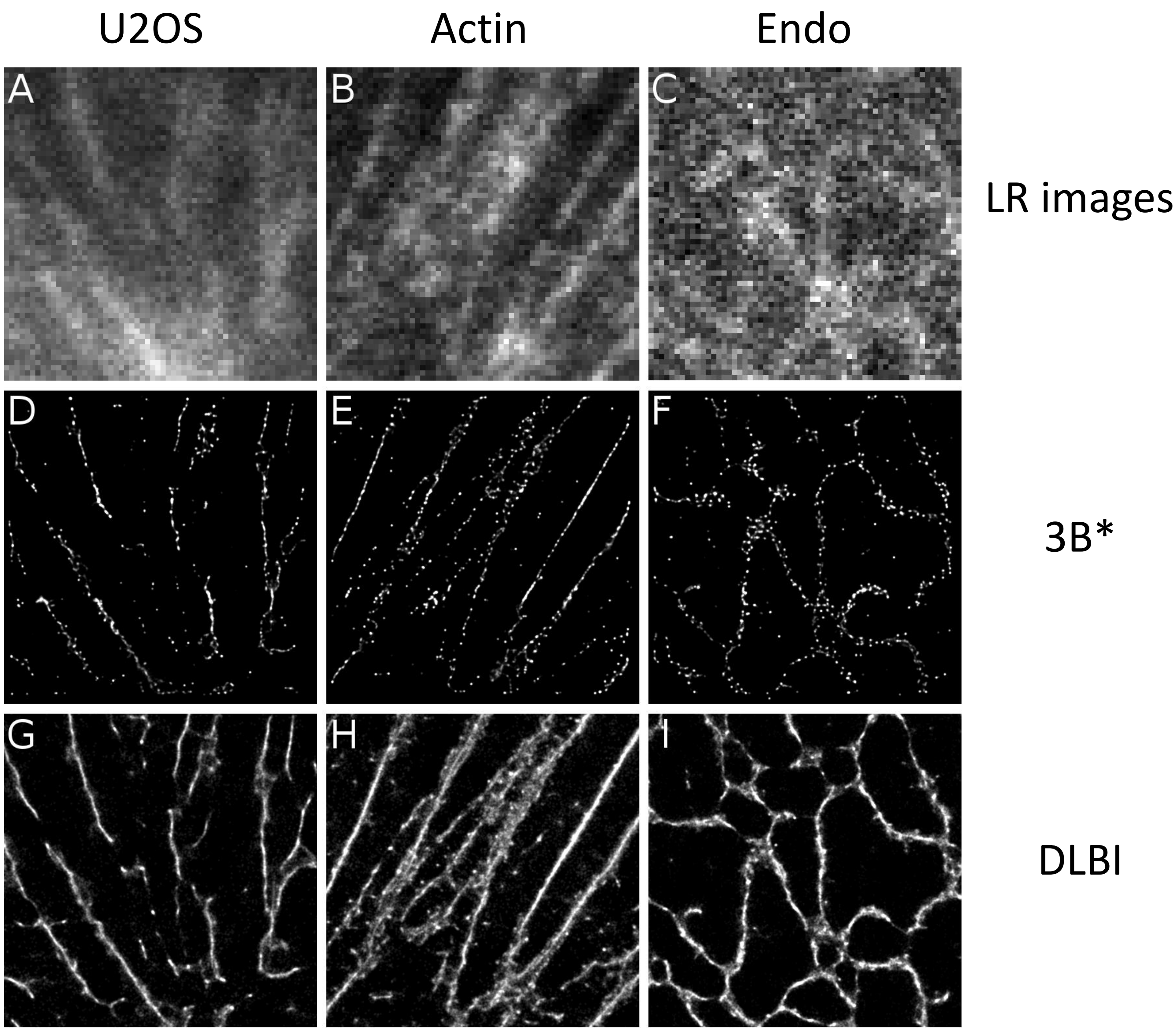
1. Simulation
2. Deep learning
3. Bayesian

Stages:

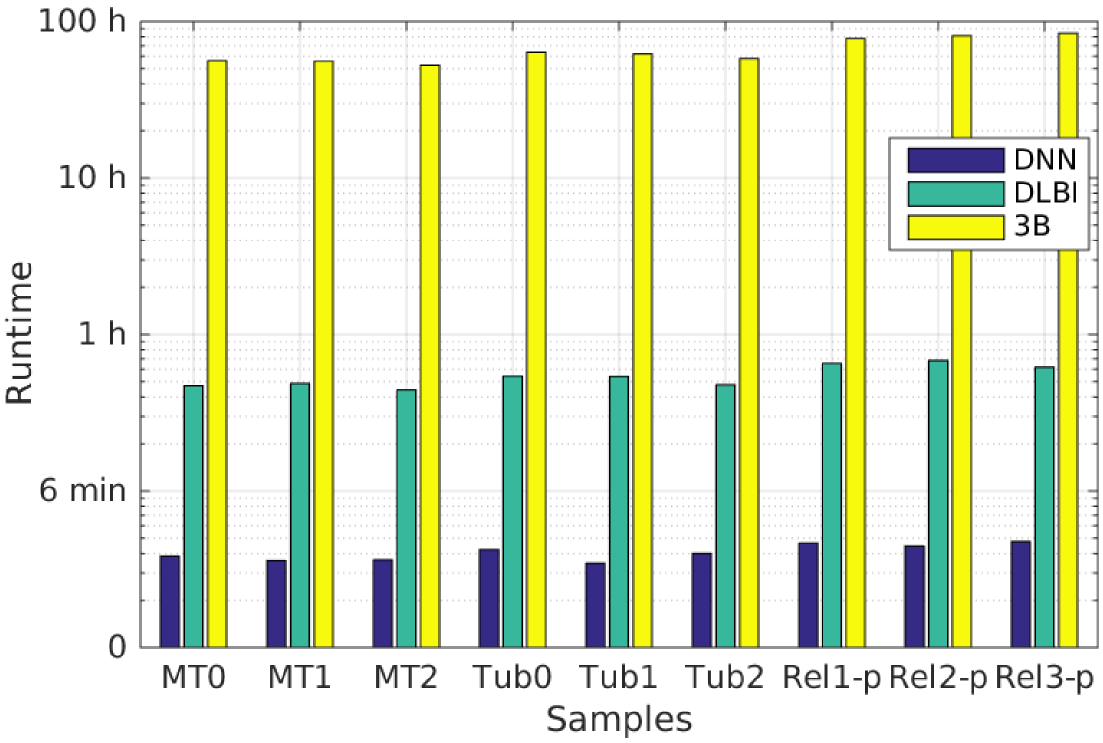
Training, Testing

Li, Xu, Zhang, Xu, Zhang, Fan, Li, Gao, and Han. ***Bioinformatics***, 2018

Performance on Real Data



Runtime



Compared to 3B:

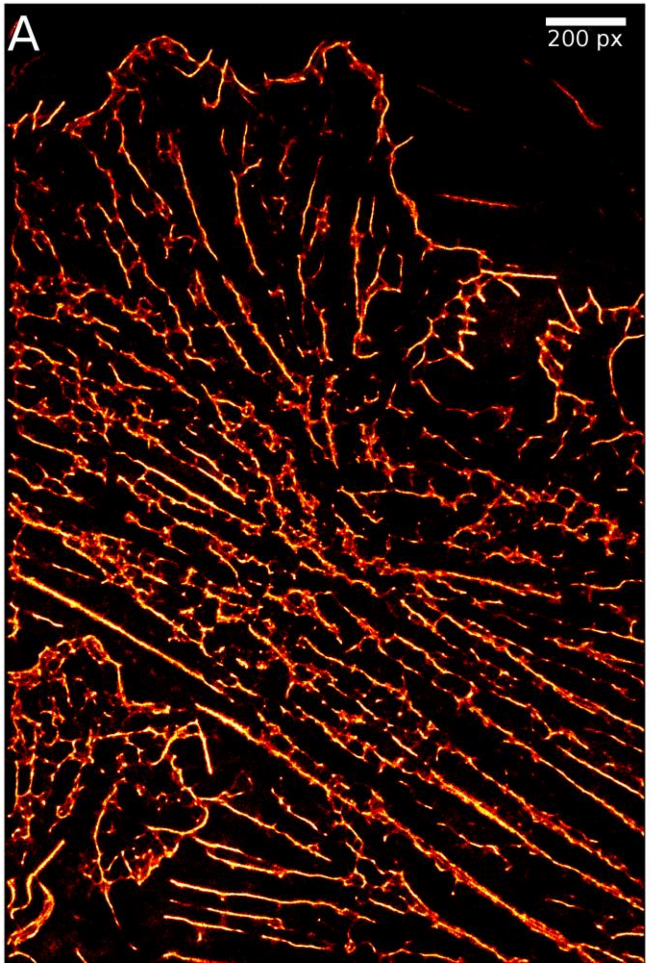
DNN: 1500X speed up

DLBI: 150X speed up

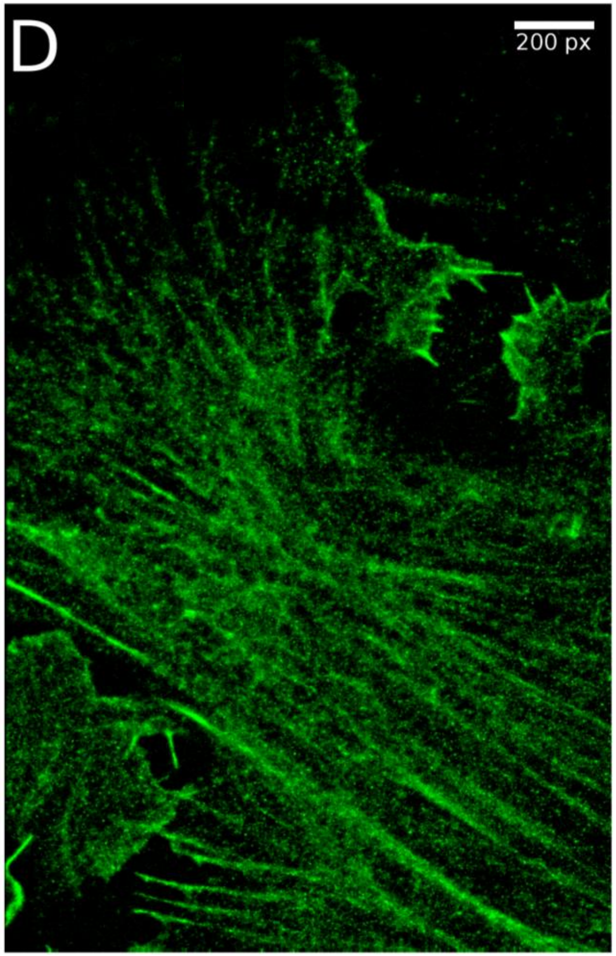
- Large field reconstruction
- Real-time reconstruction

Large-field Reconstruction

Actin in U2OS (249*395)



DLBI: 200 frames (2K*3.2K)



PALM*: 20,000 frames

Backpropagation – Example

Backpropagation: a simple example

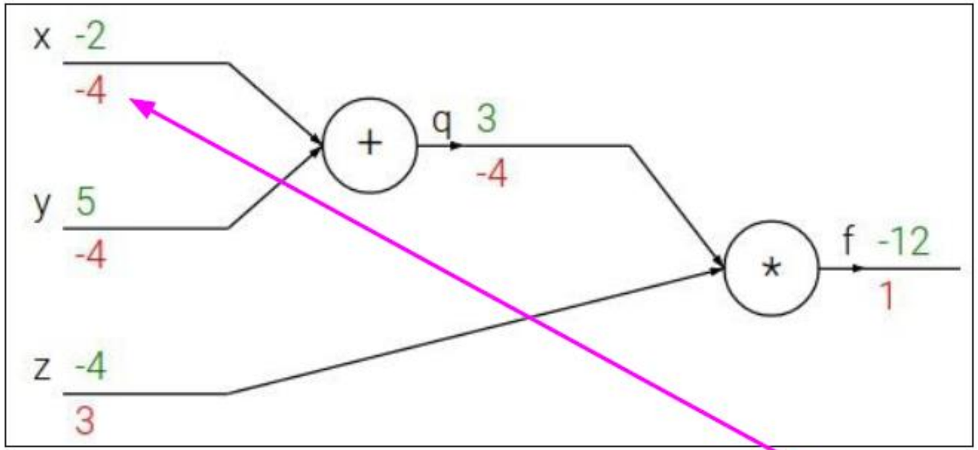
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial x}}$$

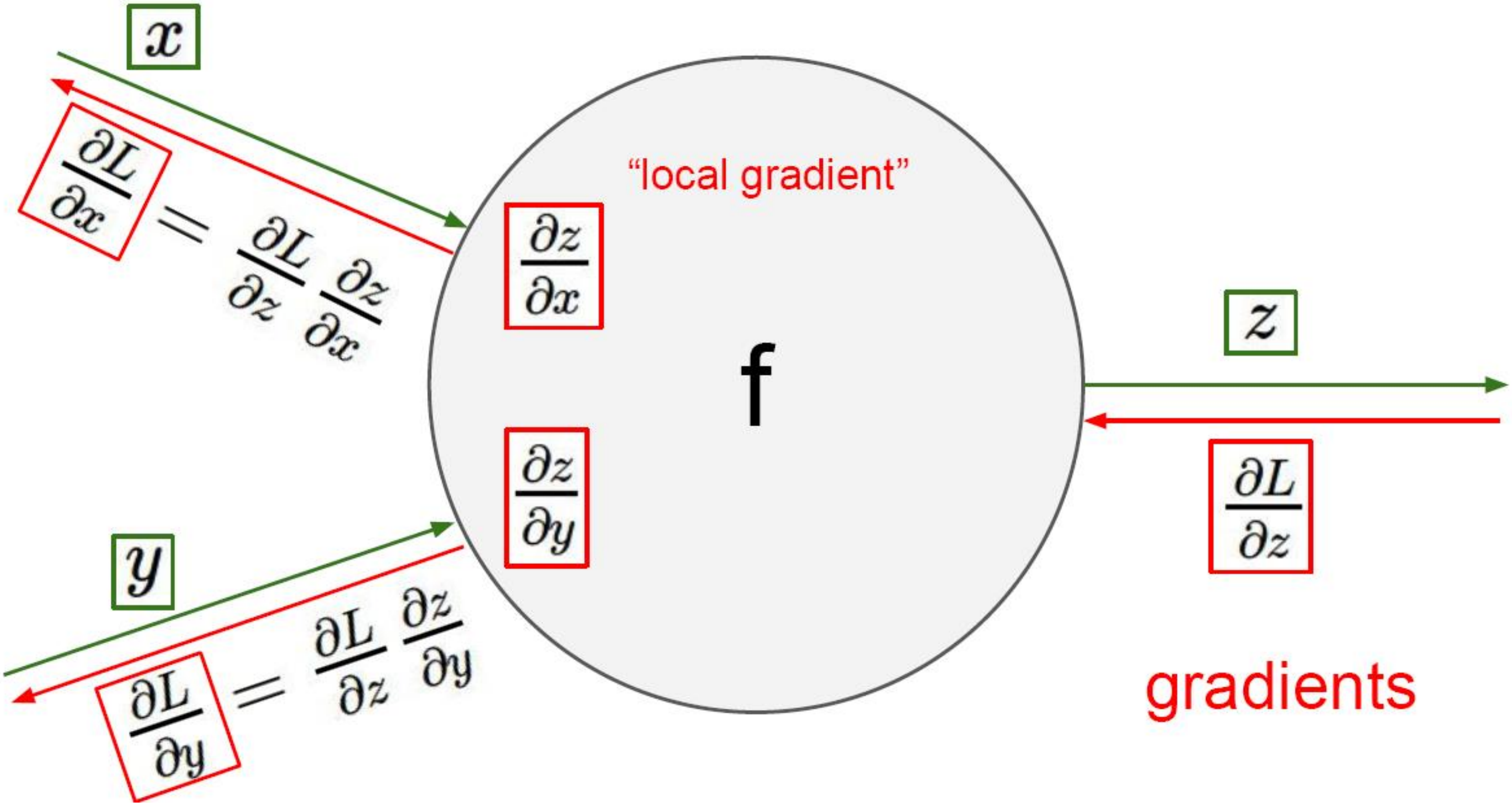
Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}}$$

Upstream gradient

Local gradient

Backpropagation – Example



Training ANN

- $\frac{\partial E}{\partial w_j^i} = \frac{\partial}{\partial w_j^i}\left[\frac{1}{2}\sum_{k=1}^{p}\left(y_k - C_k\right)^2\right] = \frac{\partial}{\partial w_j^i}\left[\frac{1}{2}\left(y_j - C_j\right)^2\right]$. That is, only the term where k = j do we have any contribution made by w_j^i
- Recall $y_j = \sigma\left(s_j\right)$ and $s_j = \sum_{i=0}^{n} w_j^{\ i} x_i$, thus

$$\frac{\partial E}{\partial w_j^i} = \frac{\partial}{\partial w_j^i}\left[\frac{1}{2}\left(y_j - C_j\right)^2\right] = \left(y_j - C_j\right)\frac{\partial y_j}{\partial w_j^i}$$

$$= \left(y_j - C_j\right)\frac{\partial y_j}{\partial s_j}\cdot\frac{\partial s_j}{\partial w_j^i} = \left(y_j - C_j\right)\frac{\partial y_j}{\partial s_j}\cdot\frac{\partial s_j}{\partial w_j^i}$$

$$= \left(y_j - C_j\right)y_j\left(1 - y_j\right)x_i$$

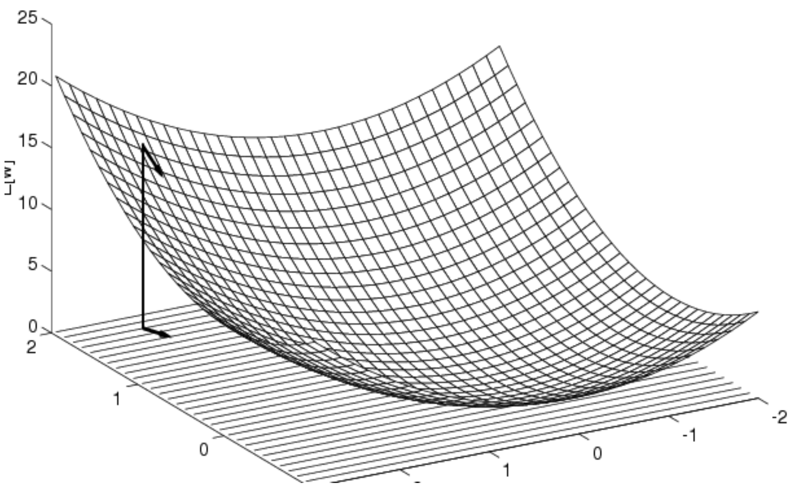
Training ANN

- $\frac{\partial E}{\partial w_j^i} = \left(y_j - C_j\right) y_j \left(1 - y_j\right) x_i$
- Define $\delta_j = \left(y_j - C_j\right) y_j \left(1 - y_j\right)$
- Thus $\frac{\partial E}{\partial w_j^i} = \delta_j x_i$
- More generally, $\delta_j = \left(y_j - C_j\right) \sigma'(s_j)$

Training ANN

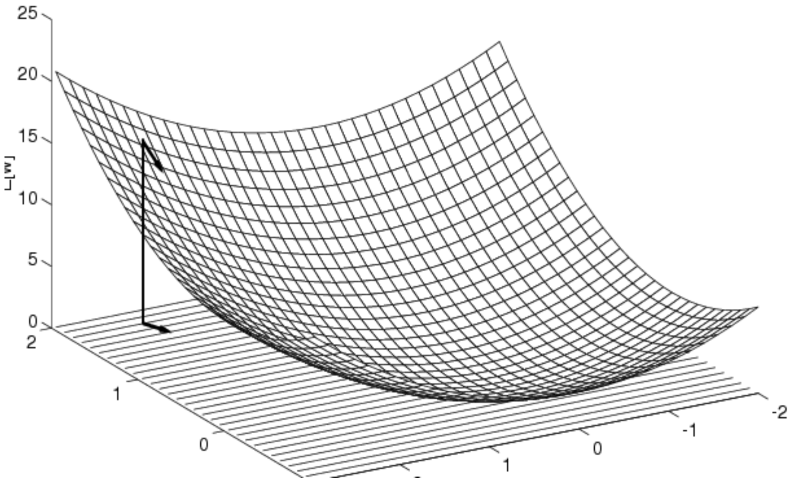
- Now how do we use $\frac{\partial \mathrm{E}}{\partial \mathrm{w}_{\mathrm{j}}^{\mathrm{i}}}$?
  - It is the gradient!

$$\frac{\partial \mathrm{E}}{\partial w} = [\frac{\partial \mathrm{E}}{\partial \mathrm{w}^0}, \frac{\partial \mathrm{E}}{\partial \mathrm{w}_{\mathrm{j}}^1}, \dots, \frac{\partial \mathrm{E}}{\partial \mathrm{w}_{\mathrm{j}}^{\mathrm{n}}}]$$



Training ANN

- Training rule: $\triangle\, w_i = -\eta \frac{\partial E}{\partial w_i}$
- A small η means slow convergence, a big η means risks of jumping over global minimum
- Why "-"?
  - $\frac{\partial E}{\partial w_i}$ positive means $\triangle\, w_i$ should be negative



Back-propagation Algorithm

- Initialize all weights to small random numbers.
- Until satisfied, Do
  - For each training example, Do
    - Input the training example to the network and compute the network outputs
    - For each output unit k: $\delta_k \leftarrow \left(z_k - C_k\right) z_k\,(1 - z_k)$
    - For each hidden unit h:

      $$\delta_h \leftarrow z_h\,(1 - z_h) \sum_k \delta_k w_k^h$$

    - Update each network weight w_j^i: $w_j^i \leftarrow w_j^i + \triangle w_j^i$

      Where $\triangle w_j^i = -\eta \delta_j\, x_i$