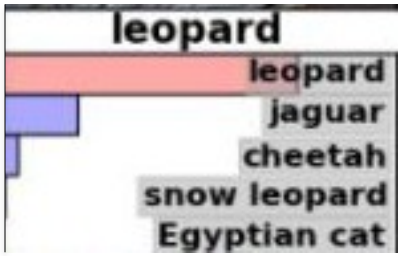
Neural Networks in Computer Vision

I've cut and pasted almost all of the following slides from several Powerpoint files I found on the Internet. The links are too numerous and long to include here and unfortunately the slides don't include the authors' names in the footers. You can find these and lots of other slides on-line by searching for " neural networks slides", "neural network powerpoint", "backpropagation" and so on.

Goal

Classification



ImageNet

- Over 15M labeled high resolution images
- Roughly 22K categories
- Collected from web and labeled by Amazon Mechanical Turk

http://image--
-net.org/

ImageNet

“ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.” http://image--net.org/

“WordNet is a lexical database for the English language. It groups English words into sets of synonyms called synsets, provides short definitions and usage examples, and records a number of relations among these synonym sets or their members. “ Wikipedia

ImageNet Publications

ImageNet: A Large-Scale Hierarchical Image Database
Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fe, IEEE CVPR 2009

ImageNet Large Scale Visual Recognition Challenge
Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei
IJCV 2015

ILSVRC

ImageNet Large Scale Visual Recognition Competition

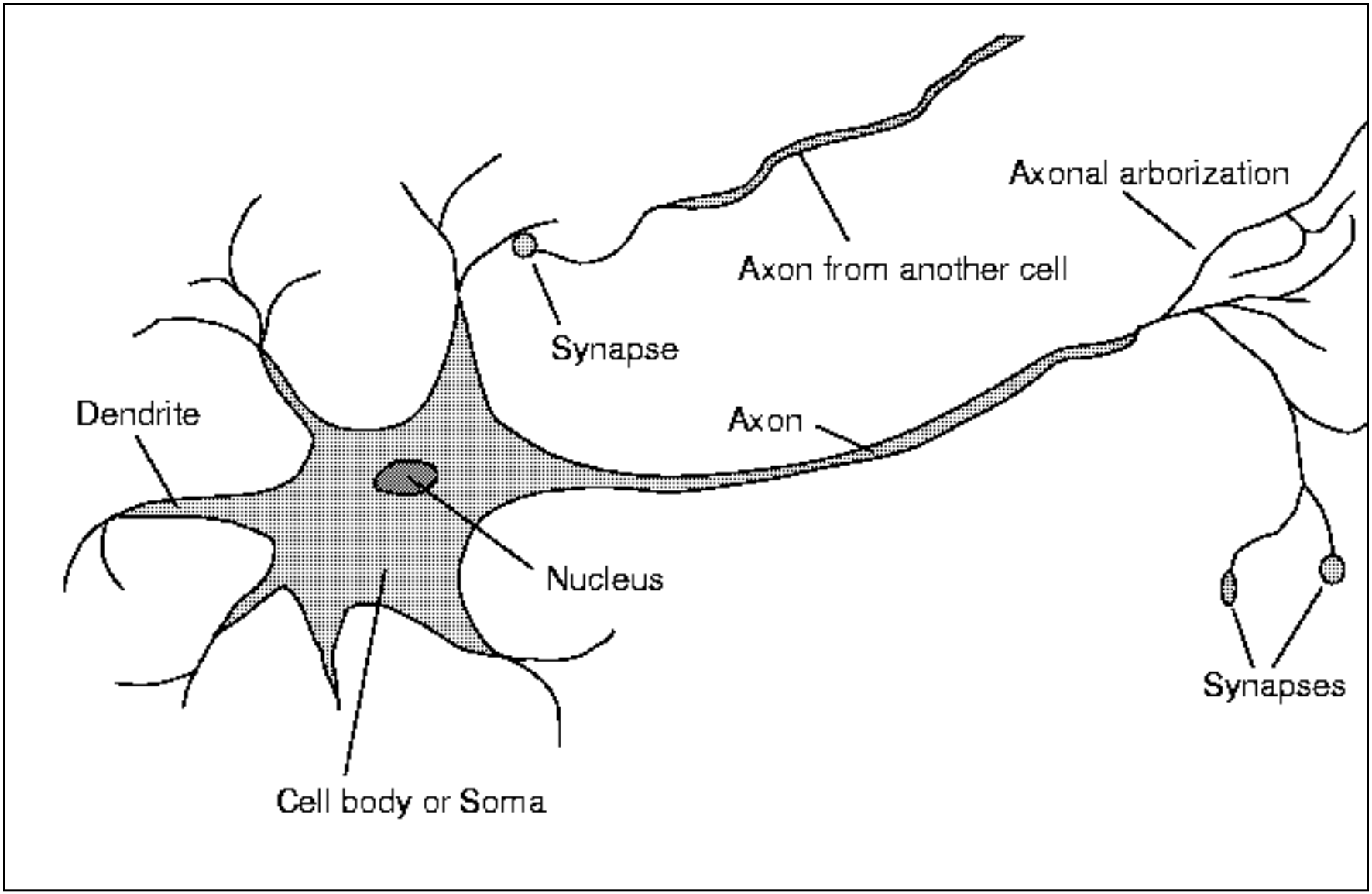
ILSVRC

- Annual competition of image classification at large scale
- 1.2M images in 1K categories
- Classification: make 5 guesses about the image label

EntleBucher

Appenzeller

Neural Networks



Neural Networks

- **We are born with about 86 billion neurons (one estimate) and 200 billion by another**

- **A neuron may connect to 1000s of other neurons.**

Neural Networks

- **Signals "move" via electrochemical signals**
- **The synapses release a chemical transmitter – the sum of which can cause a threshold to be reached – causing the neuron to "fire"**
- **Synapses can be inhibitory or excitatory**

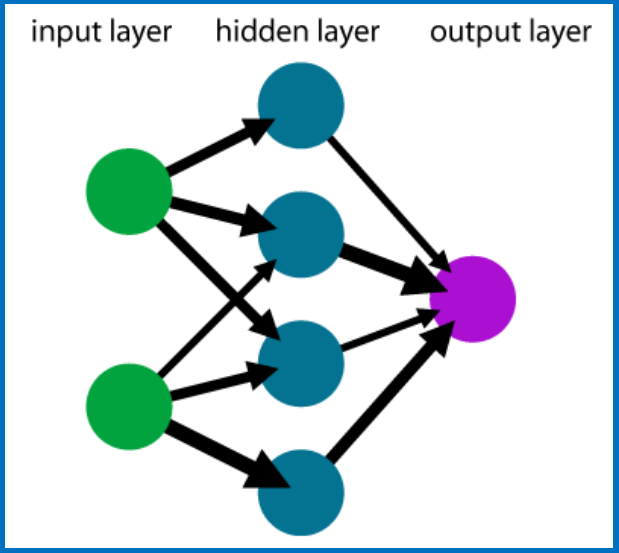
The First Neural Neural Networks

McCulloch and Pitts produced the first neural network in 1943

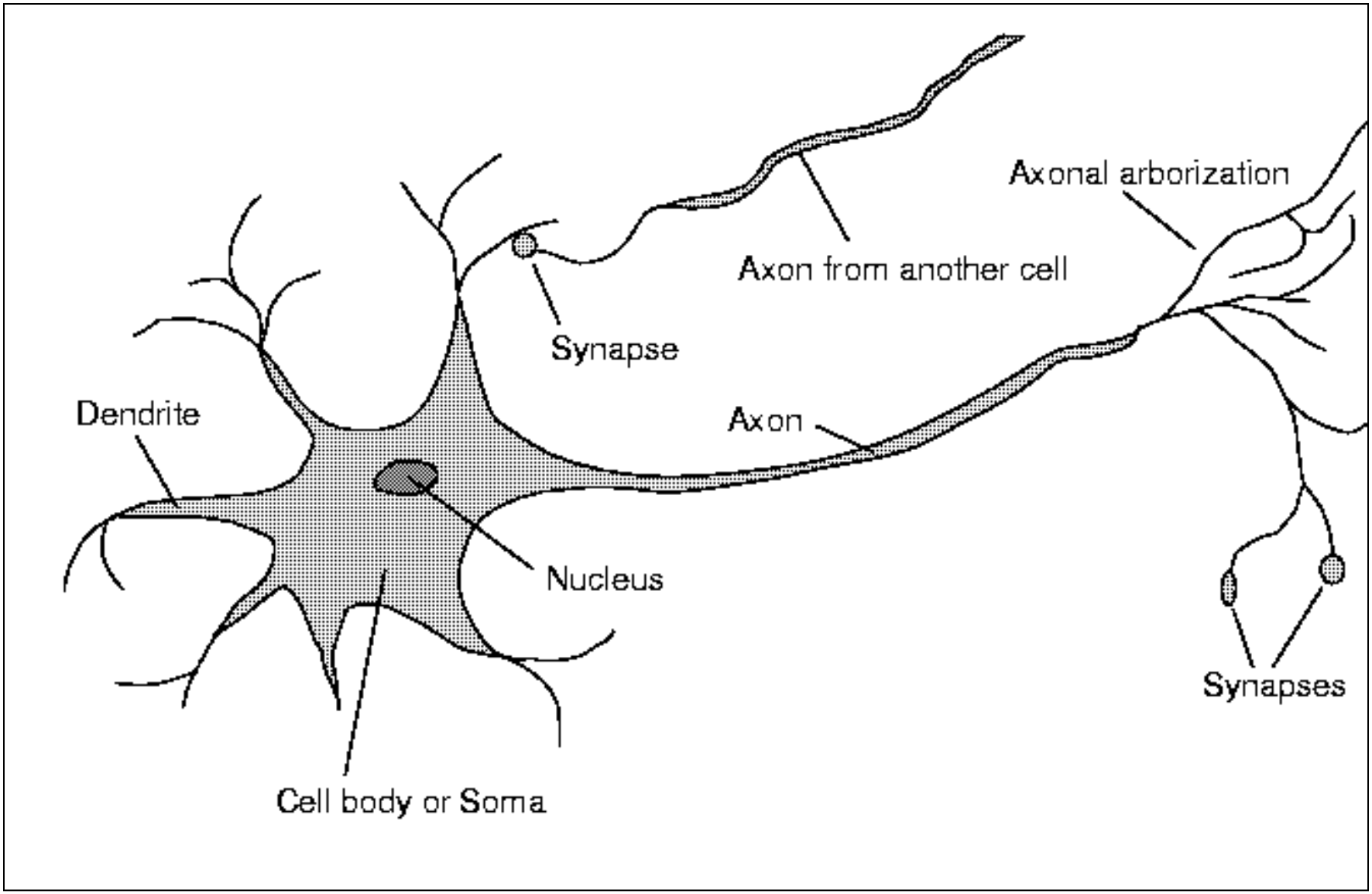
Many of the principles can still be seen in neural networks of today

What is a (an Artificial) Neural Network?

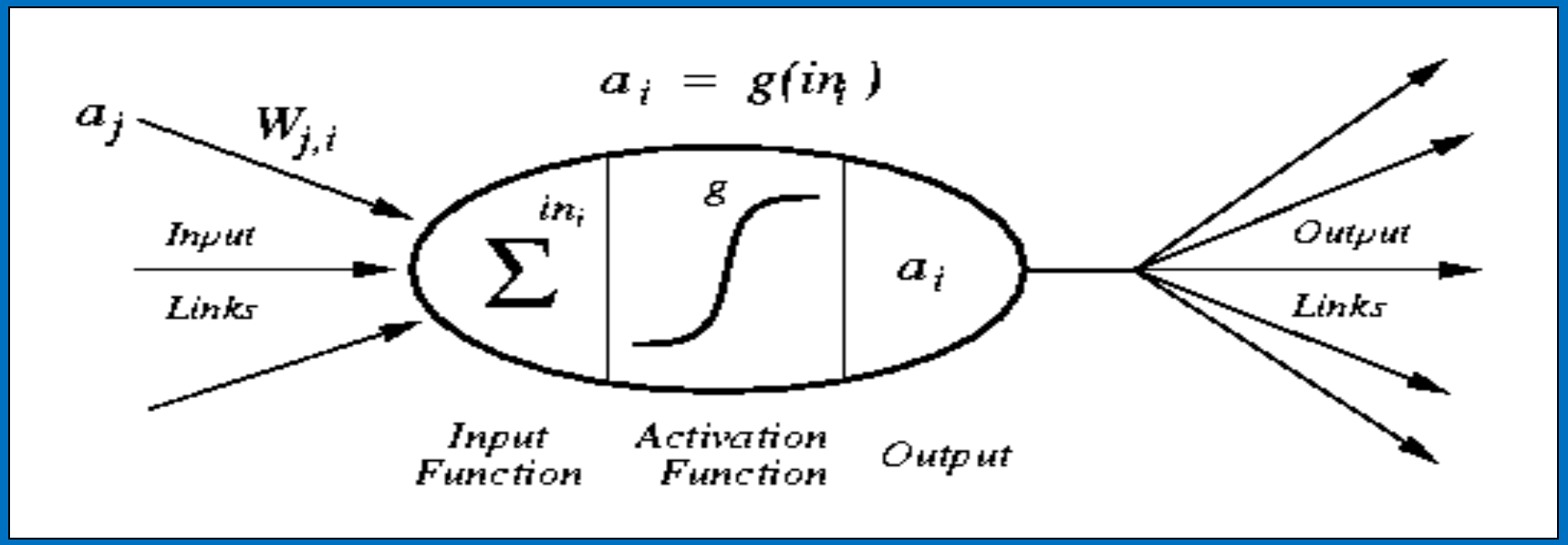
- Inspired from real neural systems
- Having a network structure, consisting of artificial neurons (nodes) and neuronal connections (weights)
- A general methodology for function approximation
- It may either be used to gain an understanding of biological neural networks, or for solving artificial intelligence problems



Neural Networks

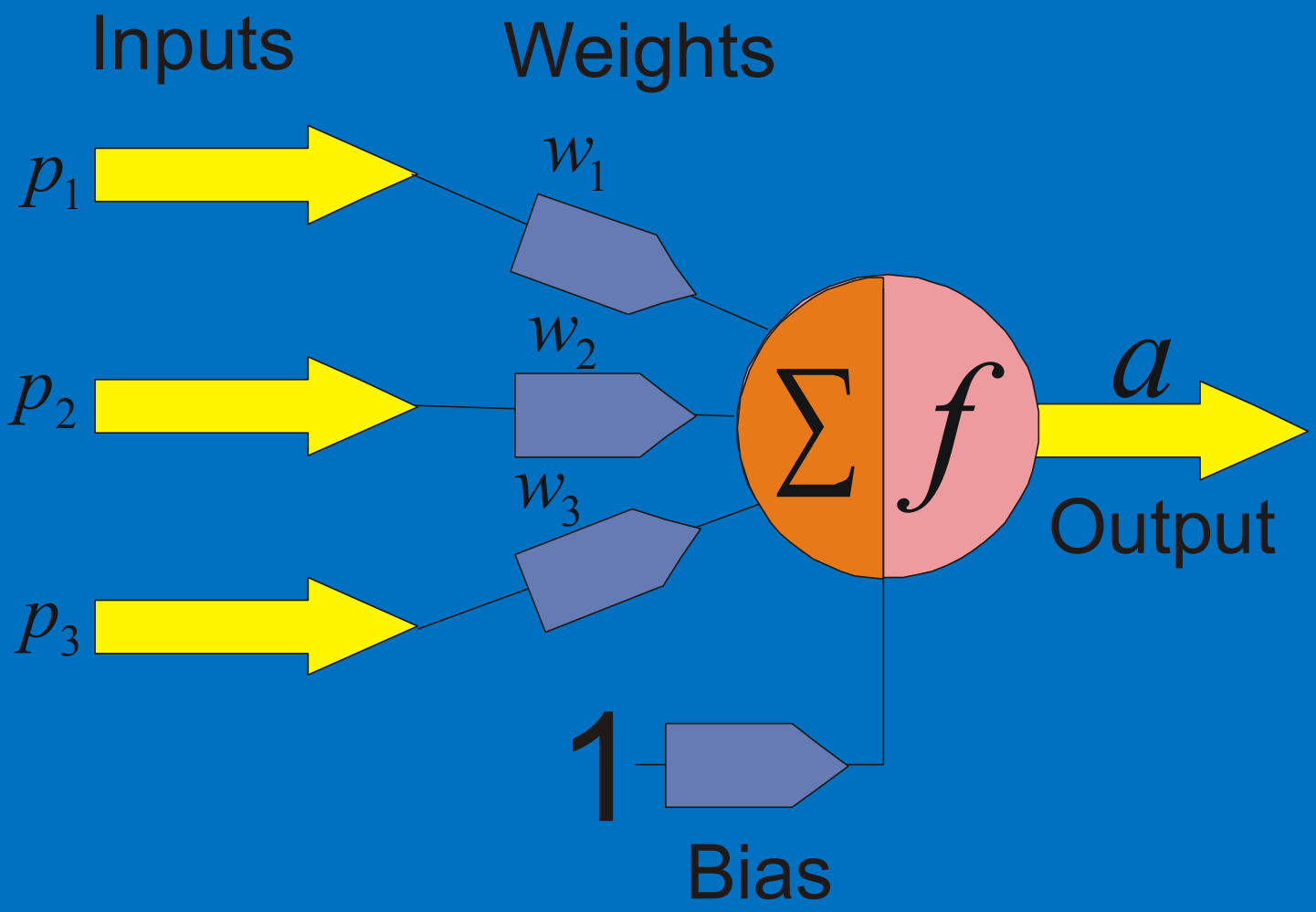


Modelling a Neuron



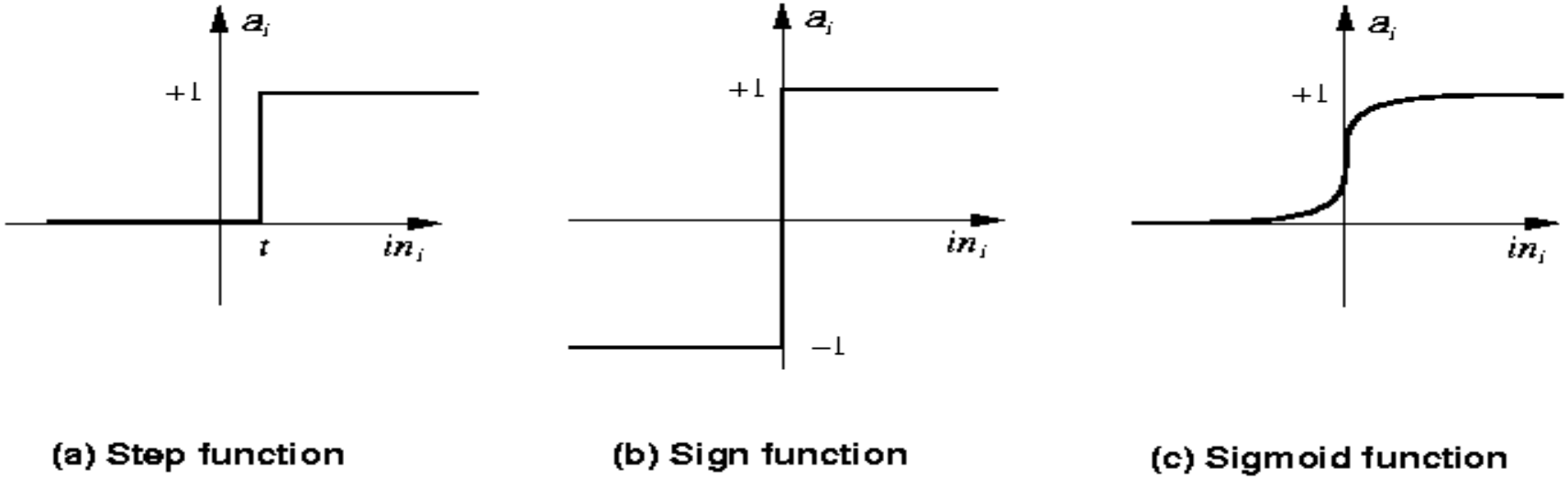
$$in_i = \sum_j W_{j,i} a_j$$

- a_j :Activation value of unit j
- $w_{j,I}$:Weight on the link from unit j to unit i
- in_I :Weighted sum of inputs to unit i
- a_I :Activation value of unit i
- g :Activation function



$$a = f\left(p_1 w_1 + p_2 w_2 + p_3 w_3 + b\right) = f\left(\sum p_i w_i + b\right)$$

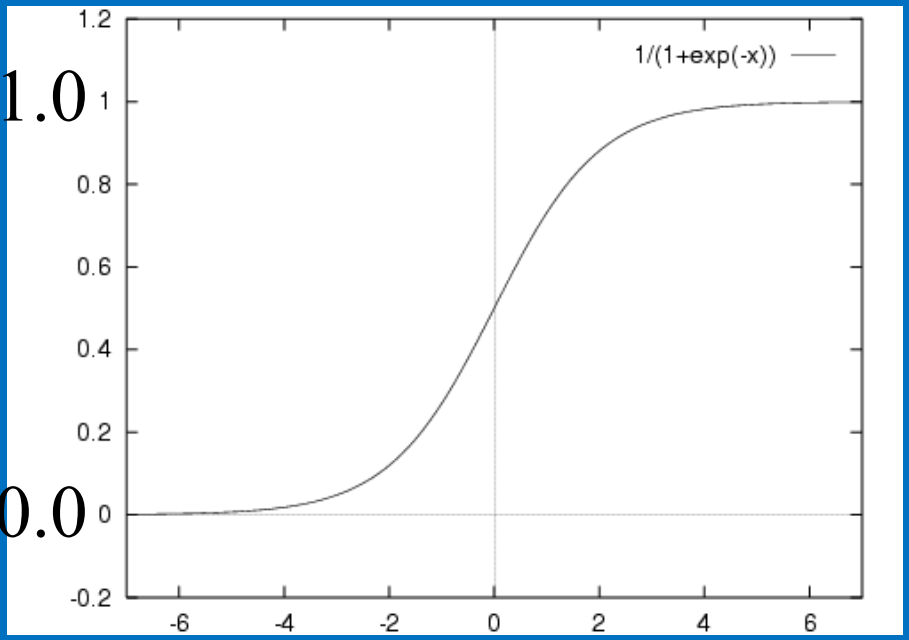
Activation Functions

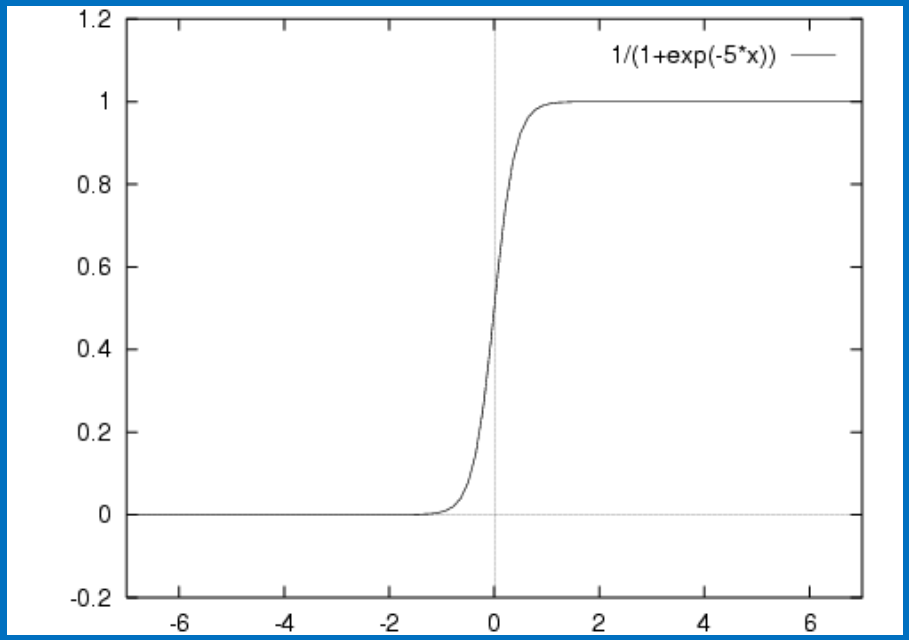


(a) Step function (b) Sign function (c) Sigmoid function

- **$Step_t(x)$ = 1 if x >= t, else 0**
- **Sign(x) = +1 if x >= 0, else –1**
- **Sigmoid(x) = $1/(1+e^{-x})$**
- **Identity Function**

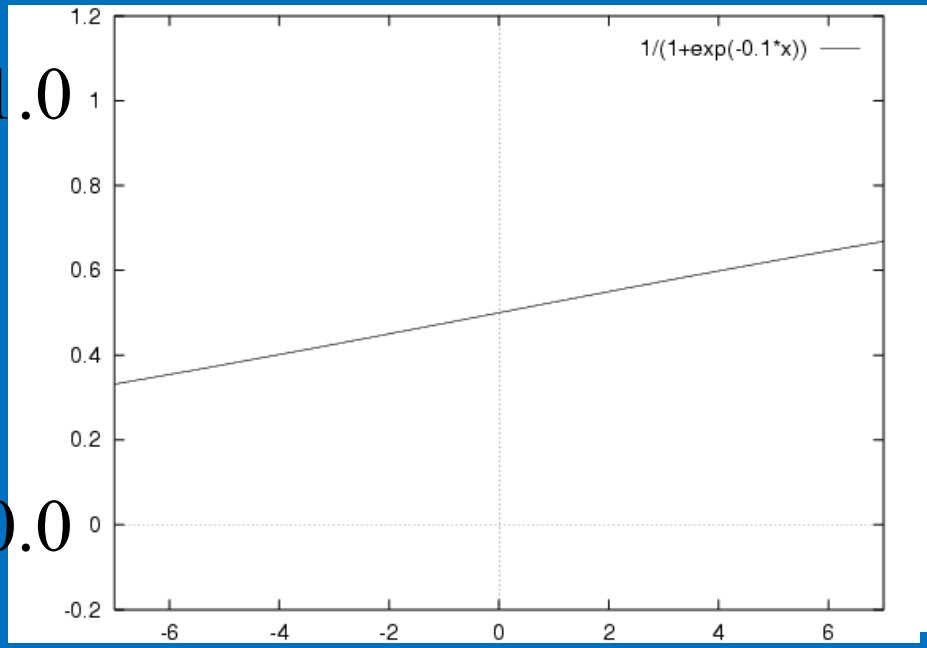
The sigmoid function





$$\frac{1}{1+e^{-\beta V}} \quad \text{with } \beta = 1.0$$

$\beta = 5.0$

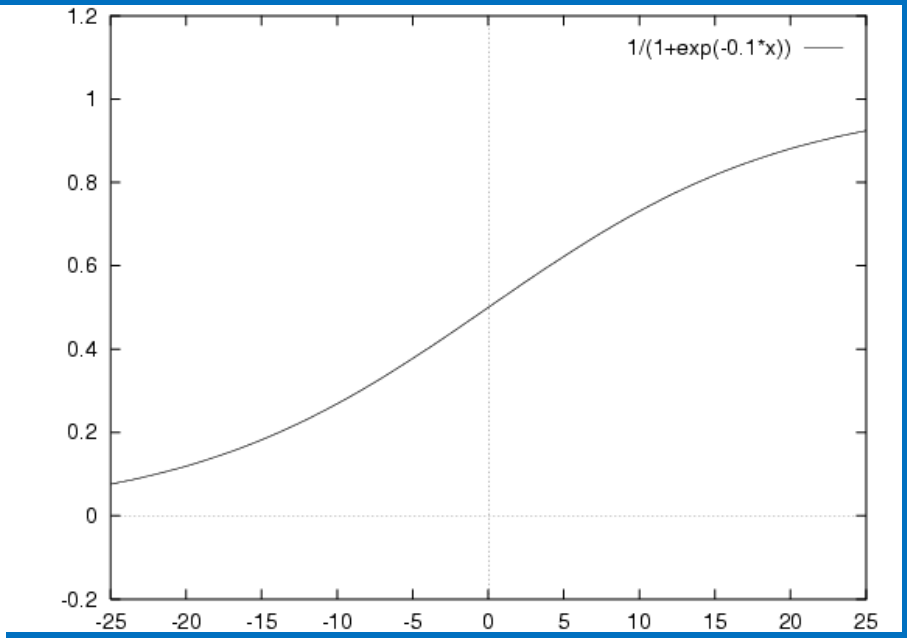


1.0

0.0

-6.0 0.0 6.0

$\beta = 0.1$

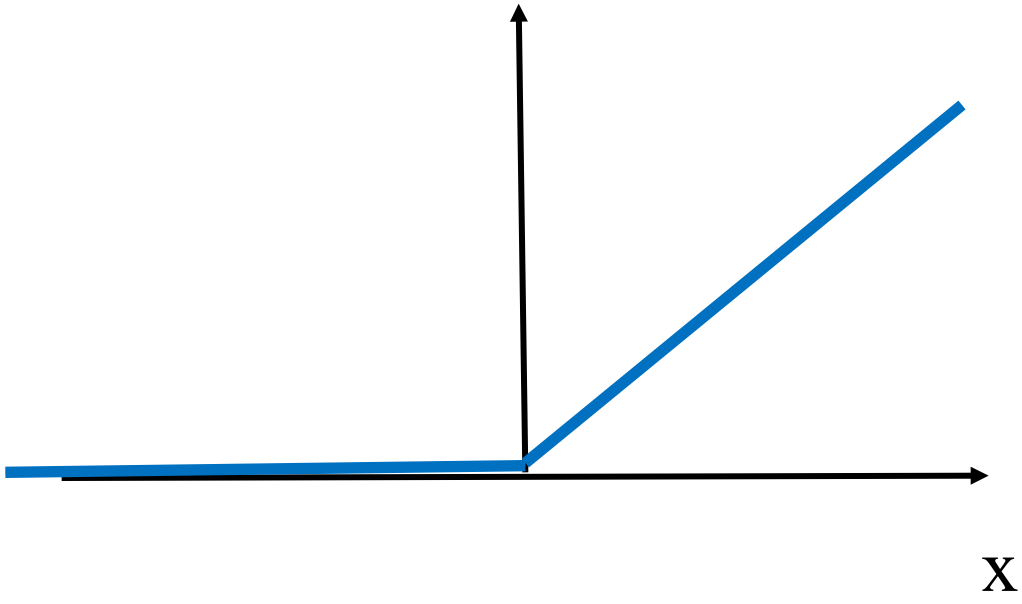


-25.0 0.0 25.0

$\beta = 0.1$

ReLU Activation Function:
Now Most Used One in Computer Vision

$$f(x) = \max(0,x)$$



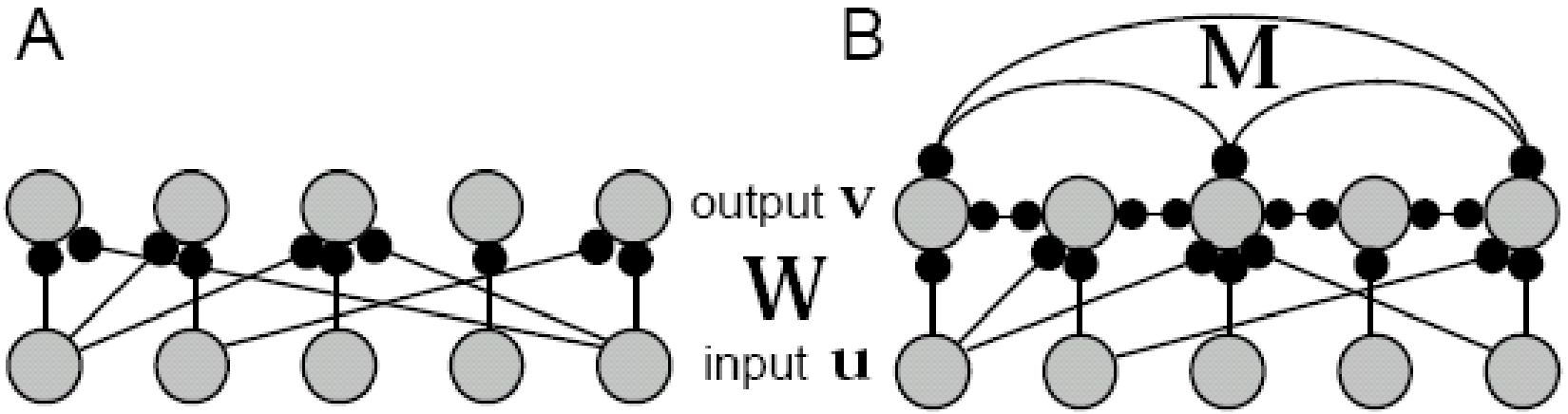
The idea of artificial neural networks

1. **Nodes --- Neurons** (artificial neurons, performing a linear or non-linear mapping)
2. **Weights --- Synapses**
3. **Mimic the network structure of neural systems**

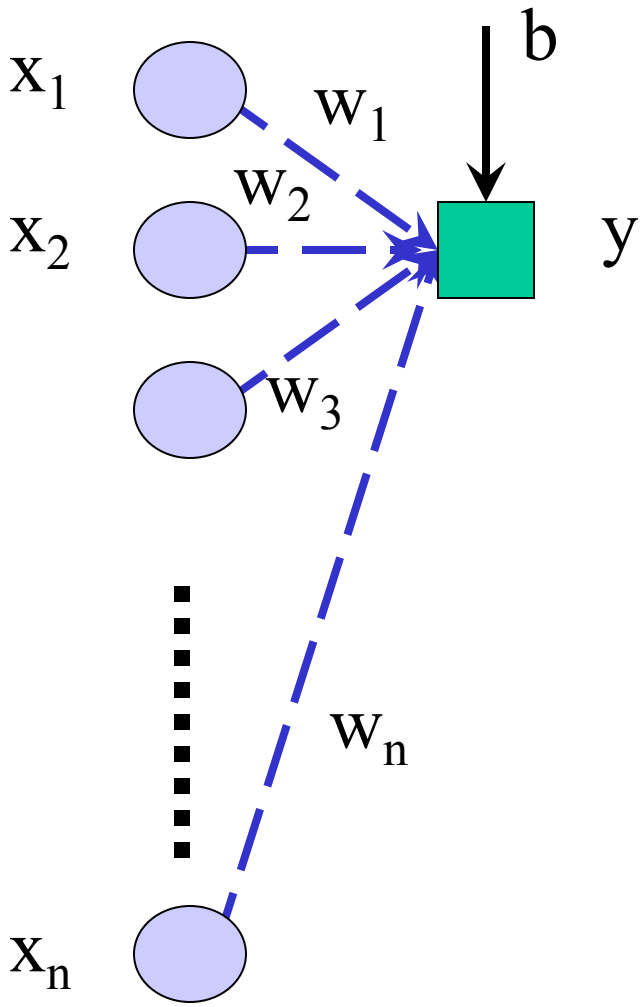
- A single neuron's function is simple. The specific functions come from the network structure

- The **connection style**
  - Feed-forward and recurrent

Feedforward and recurrent networks



An example of one-layer feed-forward neural network

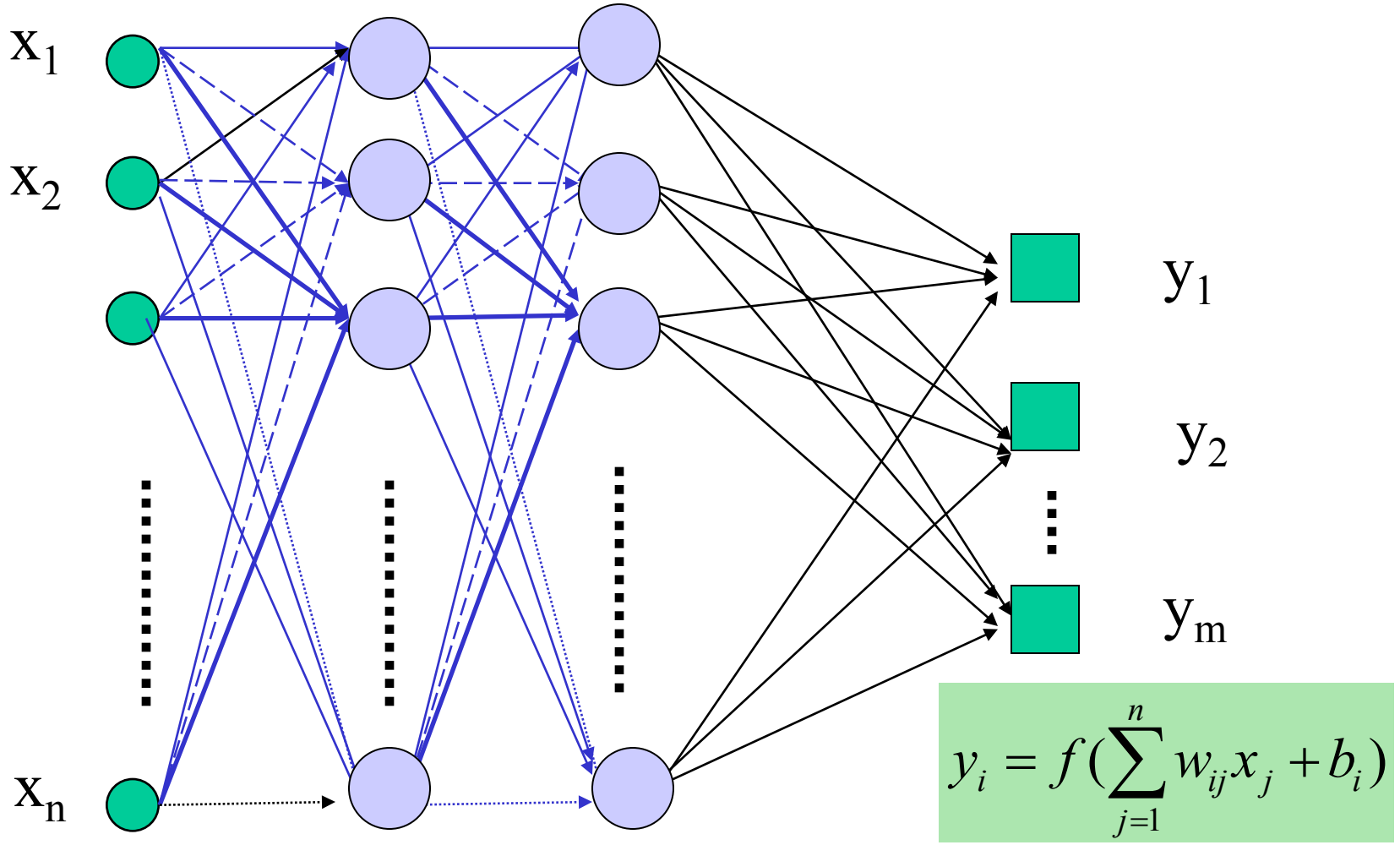


Activation function **weights**

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Bias ‘b’ is an external parameter that can be modeled by adding an extra input

An example of 3-layer feed-forward network



Neural Network (NN) for function approximation

NN transforms input into output. In other words, a NN model implements a function

NN has a set of adjustable parameters (weights, activation threshold, and etc.)

Network parameters (in particular, network weights) are free to be adjusted in order to achieve desired functions

Type of learning used depends on task at hand

- Supervised learning: have a teacher
- Unsupervised learning: no teacher
- Reinforcement learning: no detailed instruction, only the final reward is available

Possible use of artificial neural networks

Pattern recognition (face recognition, radar systems, object recognition),
Sequence recognition (gesture, speech, handwritten text recognition)
Medical diagnosis, Financial applications, Data mining,
Stock market analysis, Weather forecast

- **Classification: given an input, decide its class index**
- **Regression: given an input, decide the corresponding continuous output value**

Training Algorithm: Backpropagation

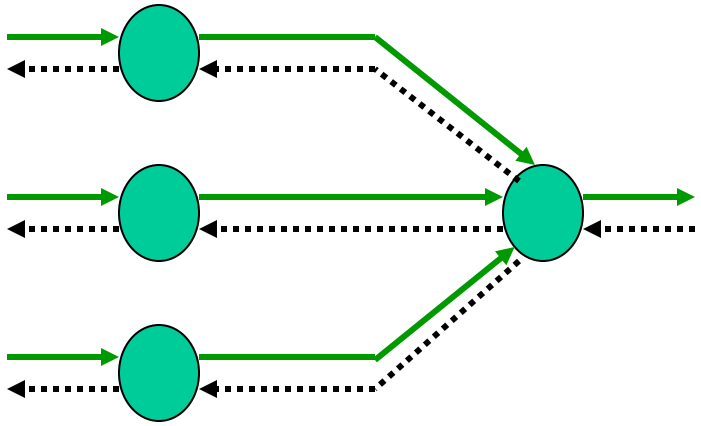
- The Backpropagation searches for weight values that minimize the total error of the network over a set of training examples (i.e., the training set).

- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.

  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

Backpropagation

- Back-propagation training algorithm



Network activation
Forward Step

Error propagation
Backward Step

- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.

Training multi--layer networks: back-propagation

Back-propagation: an effiicient method for computing gradients needed to perform gradient--based optimization of the weights in a multi--layer network

Loop until convergence:

- For each example n
  1) Input $\mathbf{x}^{(n)}$, propagate activity forward ($\mathbf{x}^{(n)} \rightarrow\rightarrow \mathbf{h}^{(n)} \rightarrow\rightarrow \mathbf{o}^{(n)}$)
  2) Propagate gradients backward
  3) Update each weight (via gradient descent)

Given any error function *E*, activation functions *g()* and *f()*, just need to derive gradients

From Uratsun, U of Toronto

Key idea behind backpropagation

We don’t have targets for a hidden unit, but we can compute how fast the error changes as we change its activity

From Uratsun, U of Toronto

Key idea behind backpropagation

We don't have targets for a hidden unit, but we can compute how fast the error changes as we change its activity

– Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.

From Uratsun, U of Toronto

Key idea behind backpropagation

We don't have targets for a hidden unit, but we can compute how fast the error changes as we change its activity

- Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.

- Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.

From Uratsun, U of Toronto

Key idea behind backpropagation

We don't have targets for a hidden unit, but we can compute how fast the error changes as we change its activity

- Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
- Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
- We can compute error derivatives for all the hidden units efficiently.
- Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

From Uratsun, U of Toronto

Total Mean Squared Error

- The error of output neuron *k* after the activation of the network on the *n-th* training example (x(n), d(n)) is:

$$e_k(n) = d_k(n) - y_k(n)$$

- The network error is the sum of the squared errors of the output neurons:

$$\mathbf{E(n)} = \sum \mathbf{e_k^2(n)}$$

- The total mean squared error is the average of the network errors of the training examples.

$$E_{AV} = \frac{1}{N} \sum_{n=1}^{N} E(n)$$

Weight Update Rule

- The Backprop weight update rule is based on the gradient descent method:
  - It takes a step in the direction yielding the maximum decrease of the network error E.
  - This direction is the opposite of the gradient of E.
- Iteration of the Backprop algorithm is usually terminated when the sum of squares of errors of the output values for all training data in an epoch is less than some threshold such as 0.01

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \Delta \mathrm{w}_{\mathrm{ij}} = \text{-}\eta \frac{\partial E}{\partial \mathrm{w}_{\mathrm{ij}}}$$

Backprop learning algorithm (incremental-mode)

n=1;

initialize **weights** randomly;

while (stopping criterion not satisfied or n <max_iterations)

for each example (**x**,*d*)

- run the network with input x and compute the output y
- update the weights in backward order starting from those of the output layer:

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

with Δw_{ji} computed using the (generalized) Delta rule

end-for

n = n+1;

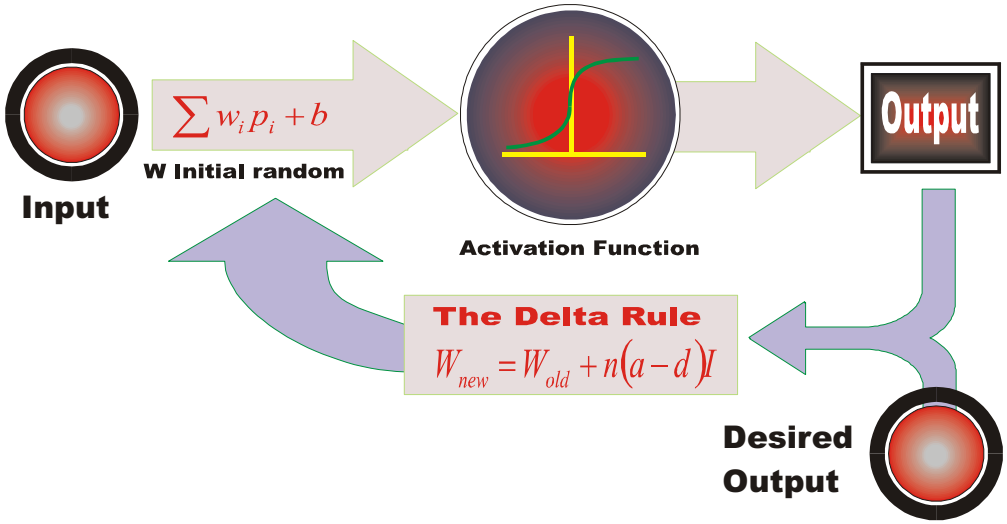
end-while;

Stopping criteria

- Total mean squared error change:
  - Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used then the part of the training set used for validating (the validation set) the network generalization will not used for updating the weights.

The Learning Rule

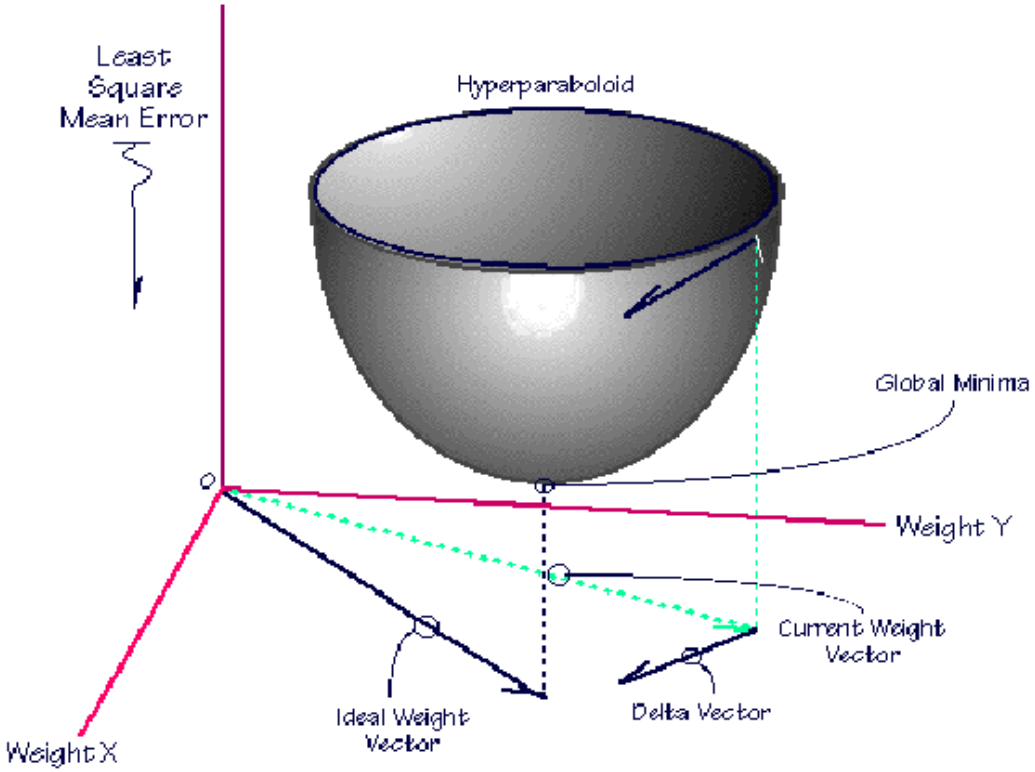
- The delta rule is often utilized by the most common class of ANNs called backpropagational neural networks.



- When a neural network is initially presented with a pattern it makes a random guess as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights.

Delta Rule Intuition

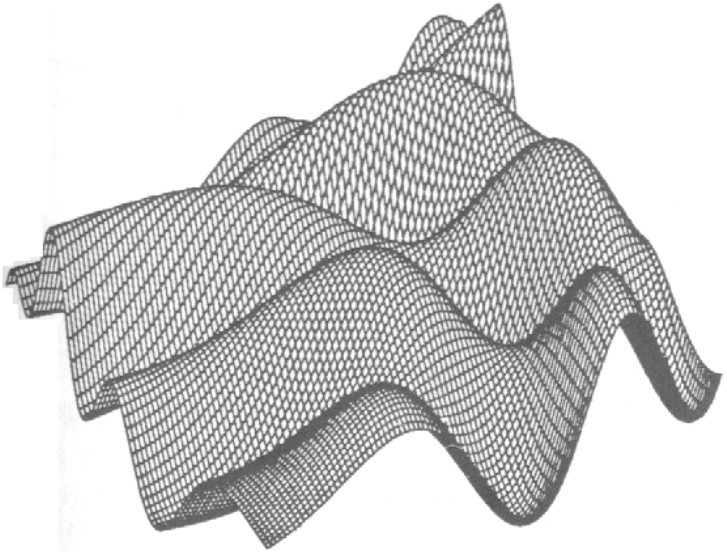
- Backpropagation performs a gradient descent within the solution's vector space towards a global minimum. The error surface itself is a hyperparaboloid but is seldom smooth as is depicted in the graphic below. Indeed, in most problems, the solution space is quite irregular with numerous pits and hills which may cause the network to settle down in a local minimum which is not the best overall solution.



Training the Network - Learning

- Backpropagation
  - Requires training set (input / output pairs)
  - Starts with small random weights
  - Error is used to adjust weights (supervised learning)

  → Gradient descent on error landscape



Backpropagation is Chain Rule

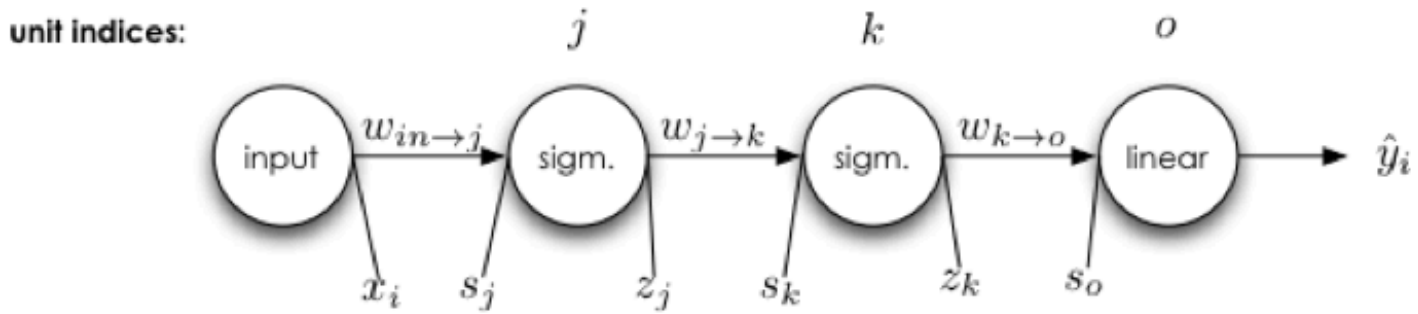
- $F(x) = f(g(x))$
- $F'(x) = f'(g(x))g'(x)$

- We want to know how the output error changes as a function of a change in a given weight.
  - i.e., we want to know the error derivative for that weight.
  - The error is a function of the weight nested via layers and activation functions.

By Brian Dolhansky

Case 1: Single input and single output

Suppose we have the following network:



A simple "one path" network.

We can explicitly write out the values of each of variable in this network:

$$
\begin{aligned}
s_j &= w_1 \cdot x_i \\
z_j &= \sigma(in_j) = \sigma(w_1 \cdot x_i) \\
s_k &= w_2 \cdot z_j \\
z_k &= \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
s_o &= w_3 \cdot z_k \\
\hat{y}_i &= in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
E &= \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2
\end{aligned}
$$

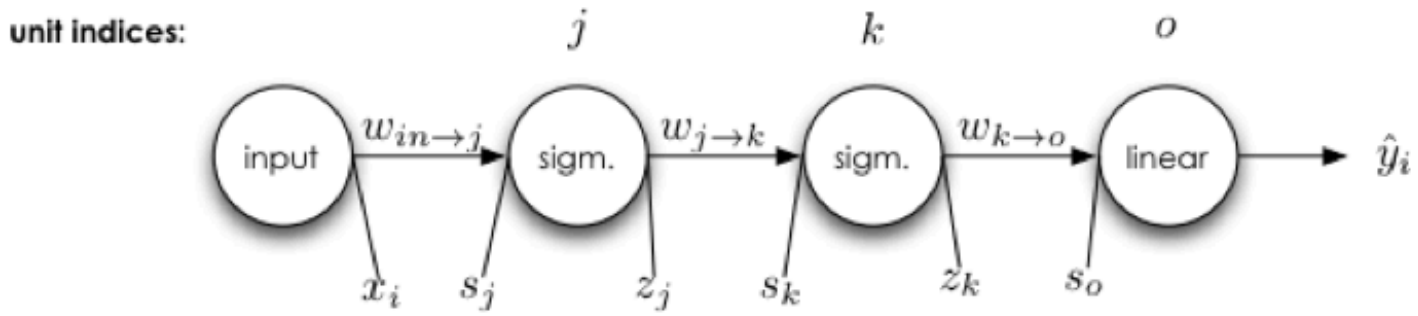
He's using σ for sigmoid activation

http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4

By Brian Dolhansky

Case 1: Single input and single output

Suppose we have the following network:



A simple "one path" network.

We can explicitly write out the values of each of variable in this network:

$$s_j = w_1 \cdot x_i$$
$$z_j = \sigma(in_j) = \sigma(w_1 \cdot x_i)$$
$$s_k = w_2 \cdot z_j$$
$$z_k = \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$
$$s_o = w_3 \cdot z_k$$
$$\hat{y}_i = in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$
$$E = \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2$$

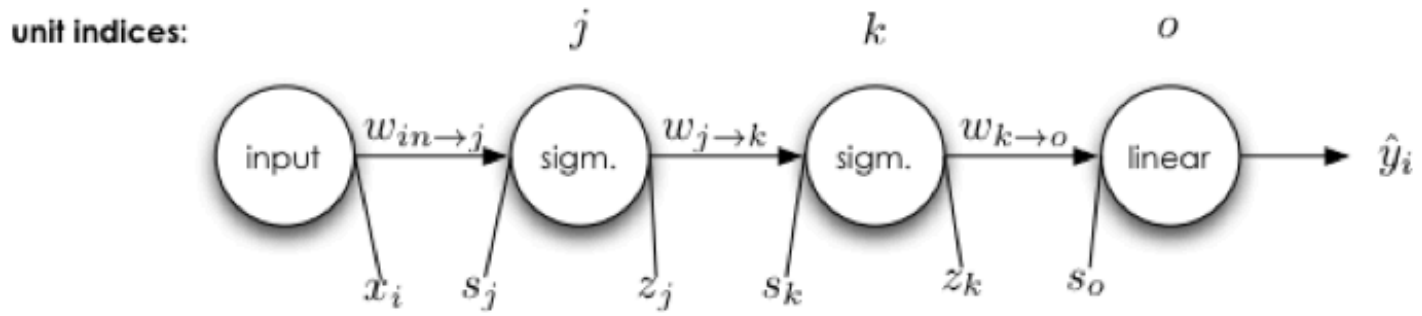
He's using σ for sigmoid activation

http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4

By Brian Dolhansky

Case 1: Single input and single output

Suppose we have the following network:



A simple "one path" network.

We can explicitly write out the values of each of variable in this network:

$$s_j = w_1 \cdot x_i$$
$$z_j = \sigma(in_j) = \sigma(w_1 \cdot x_i)$$
$$s_k = w_2 \cdot z_j$$
$$z_k = \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$
$$s_o = w_3 \cdot z_k$$
$$\hat{y}_i = in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$
$$E = \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2$$

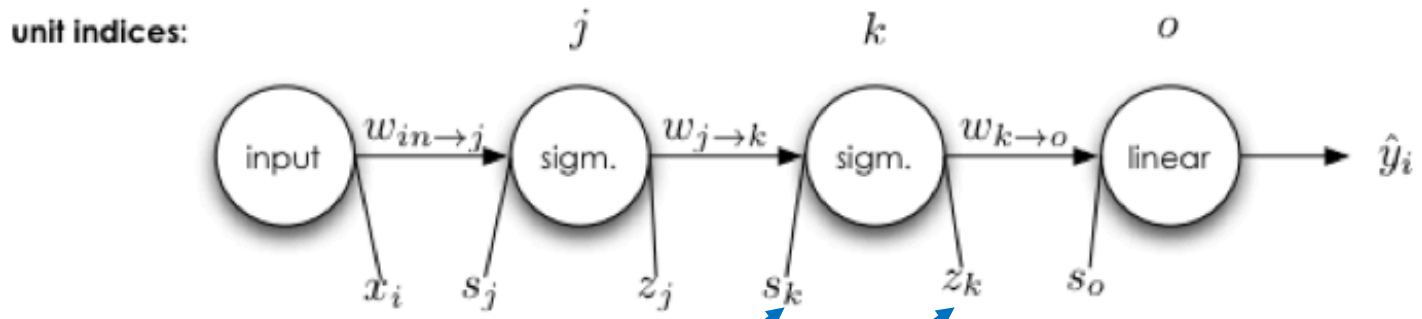
He's using σ for sigmoid activation

http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4

By Brian Dolhansky

Case 1: Single input and single output

Suppose we have the following network:



A simple "one path" network.

We can explicitly write out the values of each of variable in this network:

$$s_j = w_1 \cdot x_i$$
$$z_j = \sigma(in_j) = \sigma(w_1 \cdot x_i)$$
$$s_k = w_2 \cdot z_j$$
$$z_k = \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$
$$s_o = w_3 \cdot z_k$$
$$\hat{y}_i = in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$
$$E = \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2$$

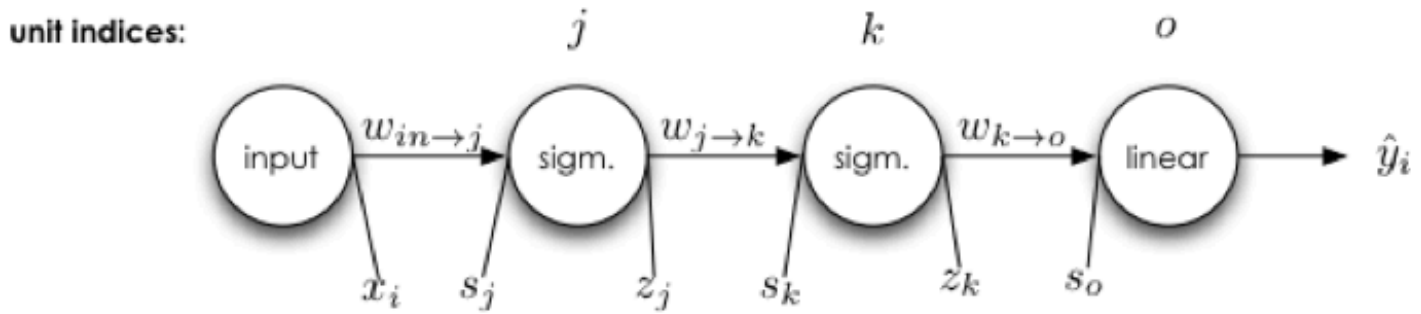
He's using σ for sigmoid activation

http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4

By Brian Dolhansky

Case 1: Single input and single output

Suppose we have the following network:



A simple "one path" network.

We can explicitly write out the values of each of variable in this network:

$$s_j = w_1 \cdot x_i$$
$$z_j = \sigma(in_j) = \sigma(w_1 \cdot x_i)$$
$$s_k = w_2 \cdot z_j$$
$$z_k = \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$
$$s_o = w_3 \cdot z_k$$
$$\hat{y}_i = in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i))$$

$$E = \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2$$

He's using σ for sigmoid activation.

His notation for weights is a bit inconsistent. Some use numbers some use arrows for same weights.

http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4

For this simple example, it's easy to find all of the derivatives by hand. In fact, let's do that now. I am going to color code certain parts of the derivation, and see if you can deduce a pattern that we might exploit in an iterative algorithm. First, let's find the derivative for $w_{k\to o}$ (remember that $\hat{y} = w_{k\to o} z_k$, as our output is a linear unit):

$$\begin{aligned}
\frac{\partial E}{\partial w_{k\to o}} &= \frac{\partial}{\partial w_{k\to o}} \frac{1}{2}(\hat{y}_i - y_i)^2 \\
&= \frac{\partial}{\partial w_{k\to o}} \frac{1}{2}(w_{k\to o} \cdot z_k - y_i)^2 \\
&= (w_{k\to o} \cdot z_k - y_i) \frac{\partial}{\partial w_{k\to o}} (w_{k\to o} \cdot z_k - y_i) \\
&= (\hat{y}_i - y_i)(z_k)
\end{aligned}$$

Derivative of the output error

Output error times input to last layer

Finding the weight update for $w_{i\to k}$ is also relatively simple:

j->k not i->k

$$\begin{aligned}
\frac{\partial E}{\partial w_{j\to k}} &= \frac{\partial}{\partial w_{j\to k}} \frac{1}{2}(\hat{y}_i - y_i)^2 \\
&= (\hat{y}_i - y_i) \left(\frac{\partial}{\partial w_{j\to k}} (w_{k\to o} \cdot \sigma(w_{j\to k} \cdot z_j) - y_i) \right) \\
&= (\hat{y}_i - y_i)(w_{k\to o}) \left(\frac{\partial}{\partial w_{j\to k}} \sigma(w_{j\to k} \cdot z_j) \right) \\
&= (\hat{y}_i - y_i)(w_{k\to o}) \left(\sigma(s_k)(1 - \sigma(s_k)) \frac{\partial}{\partial w_{j\to k}} (w_{j\to k} \cdot z_j) \right) \\
&= (\hat{y}_i - y_i)(w_{k\to o}) \left(\sigma(s_k)(1 - \sigma(s_k)) \right) (z_j)
\end{aligned}$$

Again, finding the weight update for $w_{i \to j}$ consists of some straightforward calculus:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{i \to j}} &= \frac{\partial}{\partial w_{i \to j}} \frac{1}{2} (\hat{y}_i - y_i)^2 \\
&= (\hat{y}_i - y_i) \left(\frac{\partial}{\partial w_{i \to j}} (\hat{y}_i - y_i) \right) \\
&= (\hat{y}_i - y_i)(w_{k \to o}) \left(\frac{\partial}{\partial w_{i \to j}} \cdot \sigma(w_{j \to k} \cdot \sigma(w_{i \to j} \cdot x_i)) \right) \\
&= (\hat{y}_i - y_i)(w_{k \to o})(\sigma(s_k)(1 - \sigma(s_k)))(w_{j \to k}) \left(\frac{\partial}{\partial w_{i \to j}} \sigma(w_{i \to j} \cdot x_i) \right) \\
&= (\hat{y}_i - y_i)(w_{k \to o})(\sigma(s_k)(1 - \sigma(s_k)))(w_{j \to k})(\sigma(s_j)(1 - \sigma(s_j)))(x_i)
\end{aligned}
$$

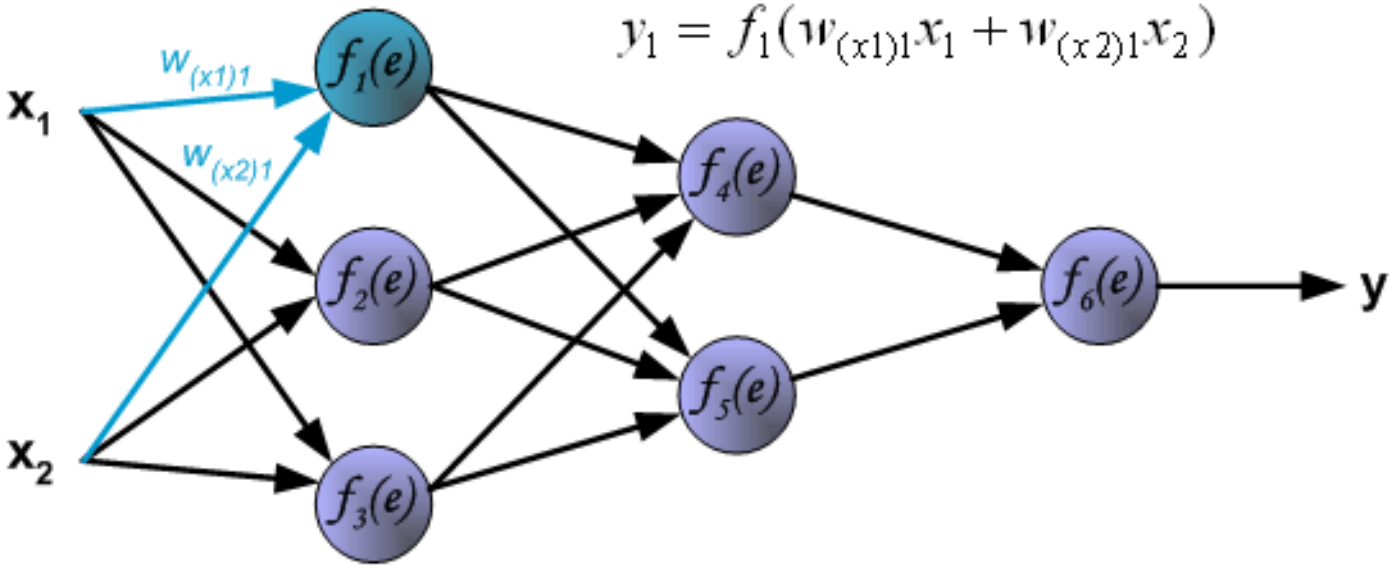
By now, you should be seeing a pattern emerging, a pattern that hopefully we could encode with backpropagation. We are reusing multiple values as we compute the updates for weights that appear earlier and earlier in the network. Specifically, we see the derivative of the network error, the weighted derivative of unit k's output with respect to s_k, and the weighted derivative of unit j's output with respect to s_j.

So, in summary, for this simple network, we have:

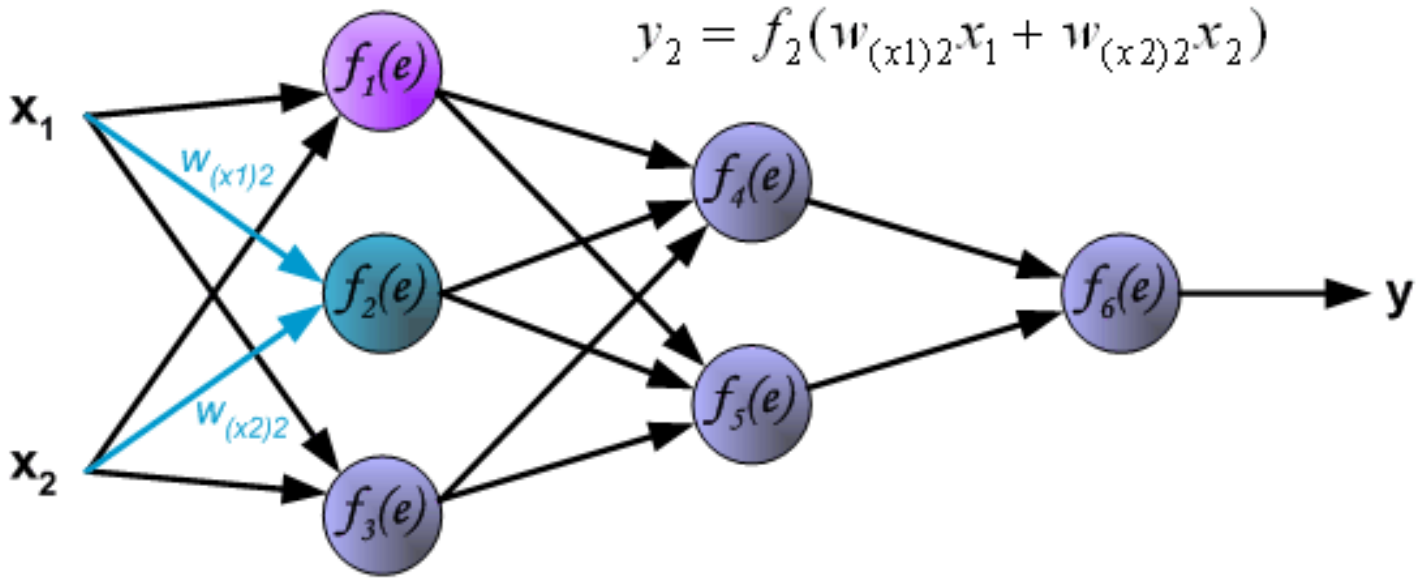
$$
\begin{aligned}
\Delta w_{i \to j} &= -\eta \left[(\hat{y}_i - y_i)(w_{k \to o})(\sigma(s_k)(1 - \sigma(s_k)))(w_{j \to k})(\sigma(s_j)(1 - \sigma(s_j)))(x_i) \right] \\
\Delta w_{j \to k} &= -\eta \left[(\hat{y}_i - y_i)(w_{k \to o}) \left(\sigma(s_k)(1 - \sigma(s_k))\right) (z_j) \right] \\
\Delta w_{k \to o} &= -\eta \left[(\hat{y}_i - y_i)(z_k) \right]
\end{aligned}
$$

Learning Algorithm: Backpropagation

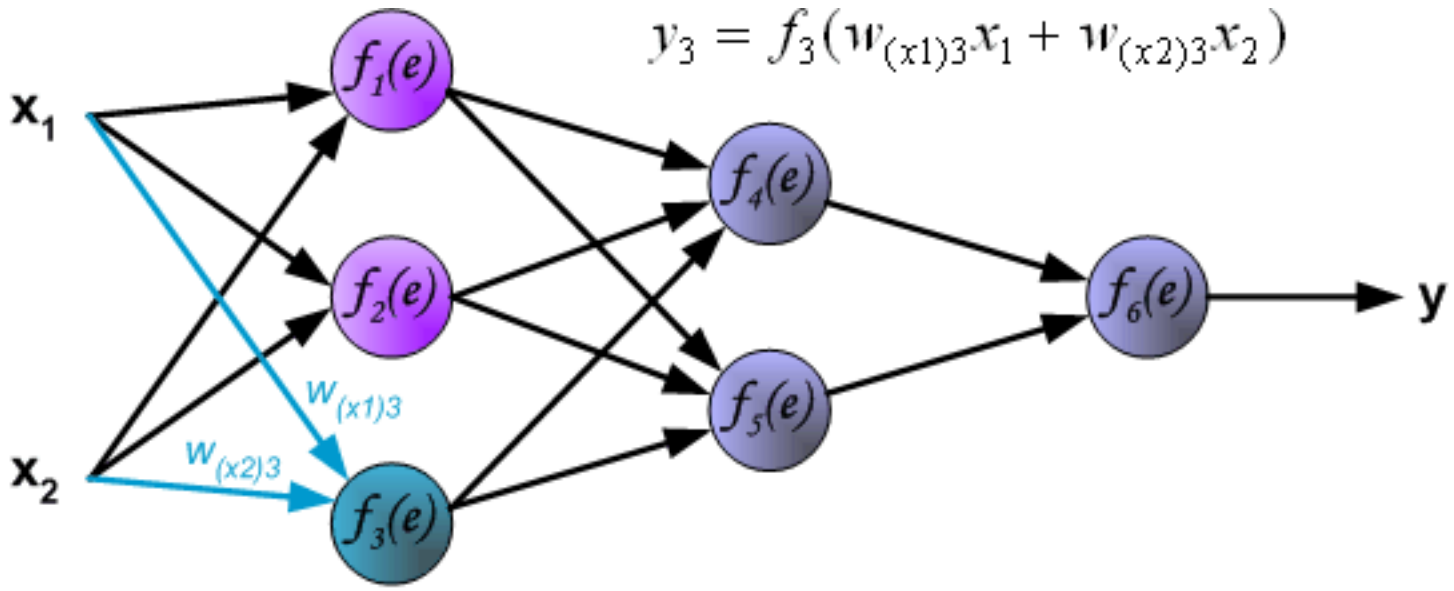
Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



Learning Algorithm: Backpropagation

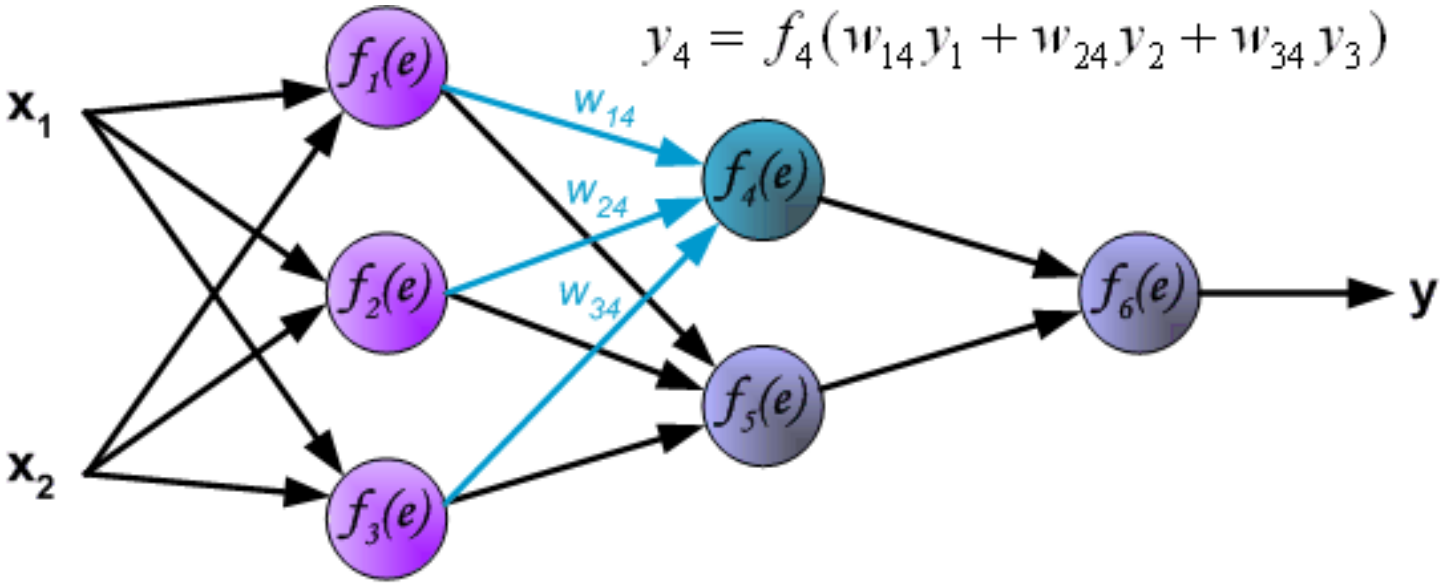


Learning Algorithm: Backpropagation

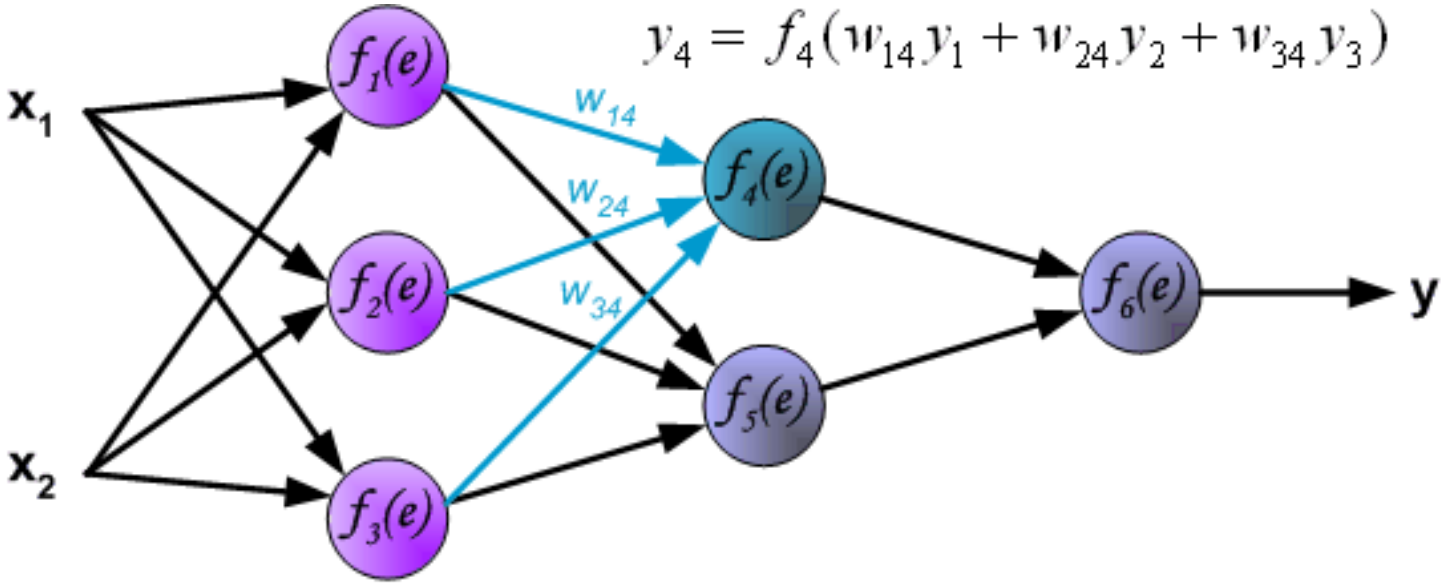


Learning Algorithm: Backpropagation

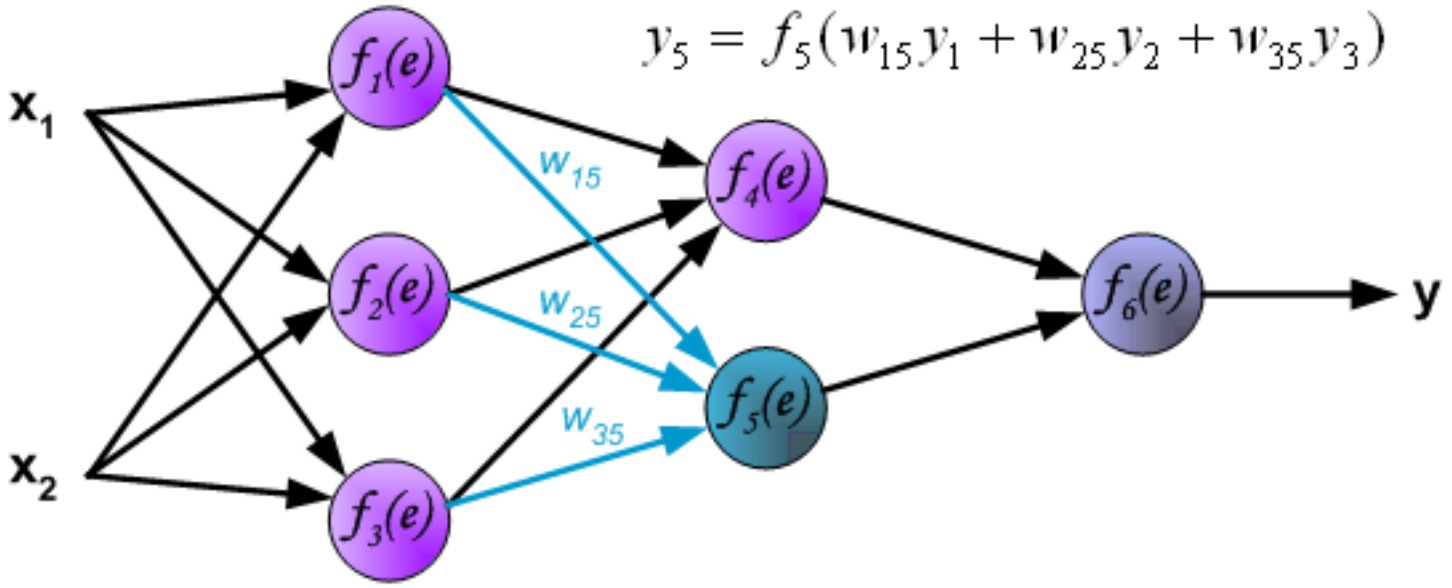
Propagation of signals through the hidden layer.
Symbols w_{mn} represent weights of connections between output of neuron m and input of neuron n in the next layer.



Learning Algorithm: Backpropagation

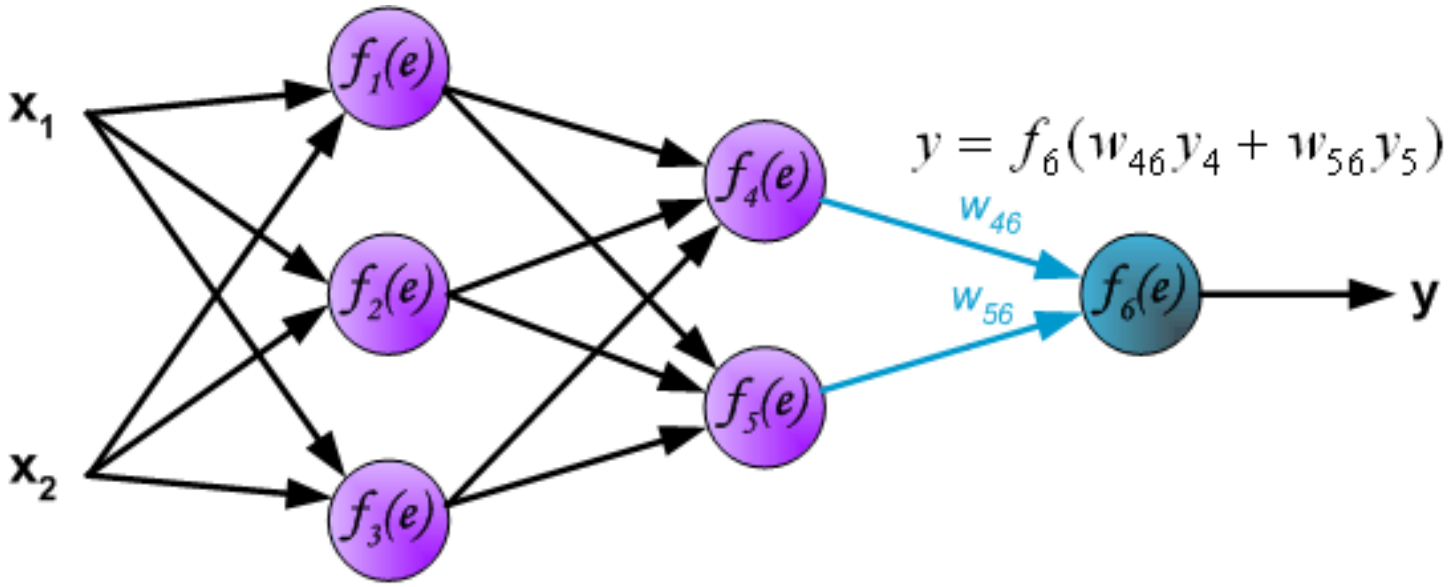


Learning Algorithm: Backpropagation



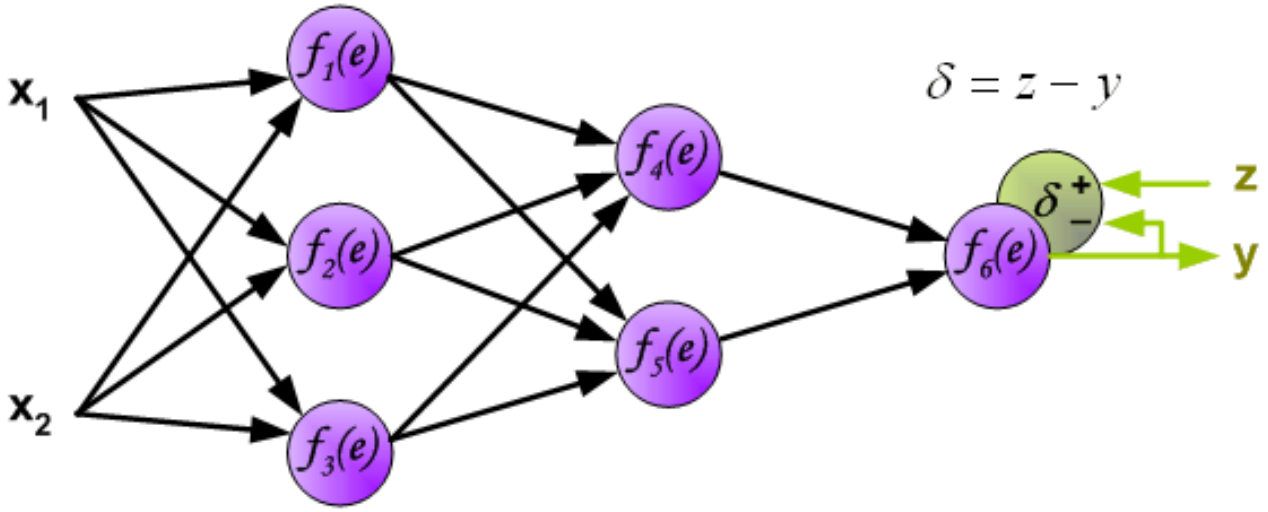
Learning Algorithm: Backpropagation

Propagation of signals through the output layer.



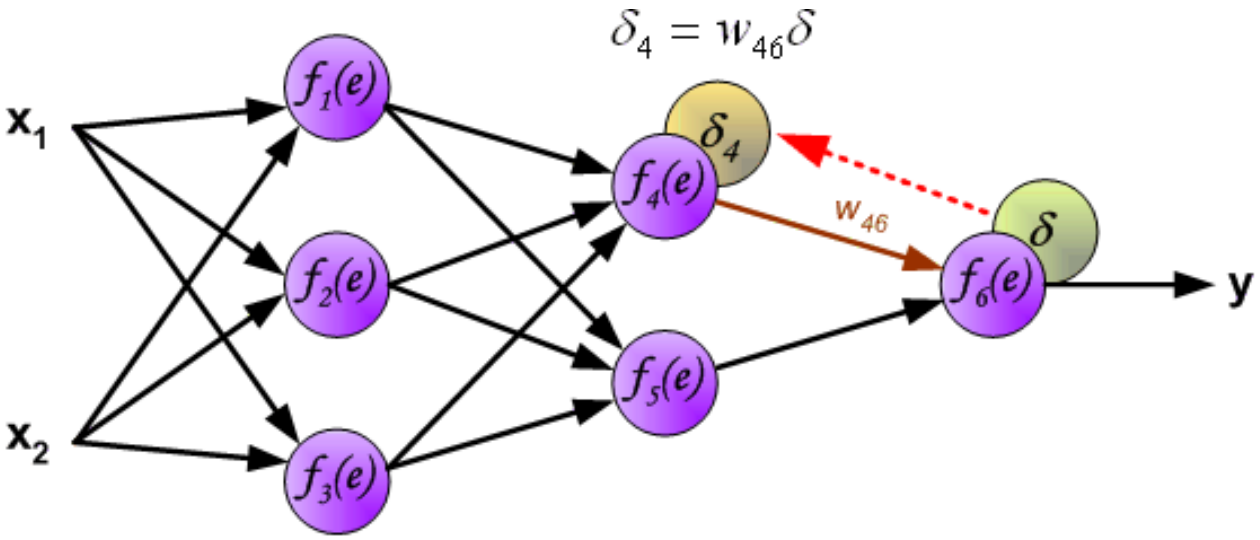
Learning Algorithm: Backpropagation

In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal *delta* of output layer neuron



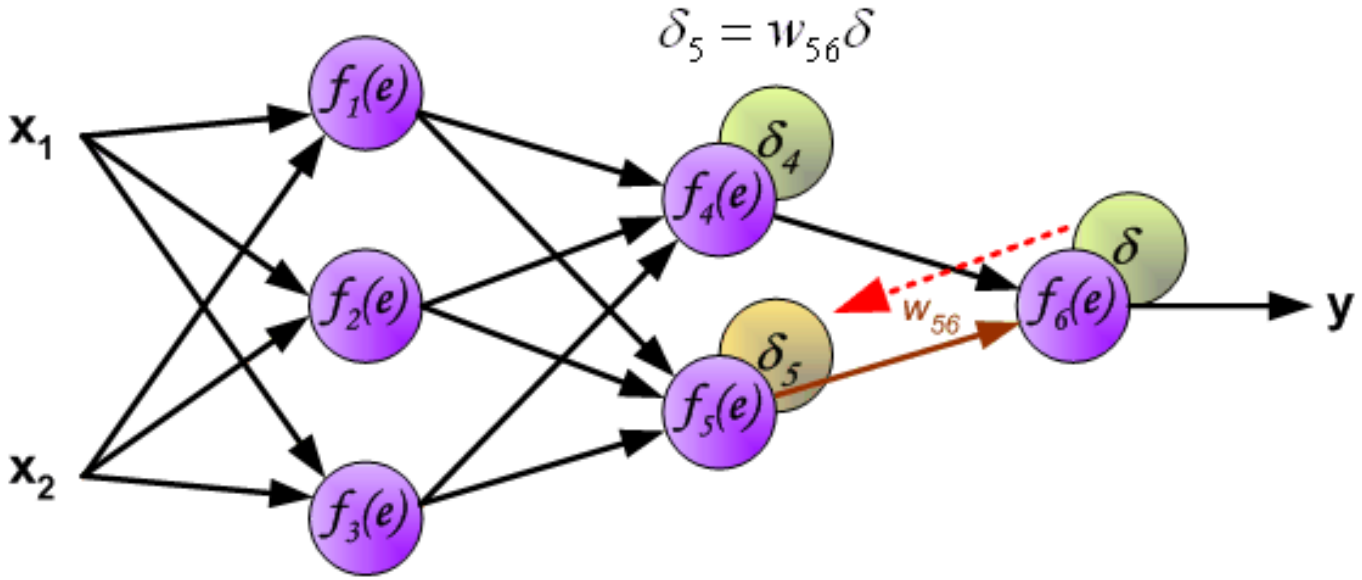
Learning Algorithm: Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



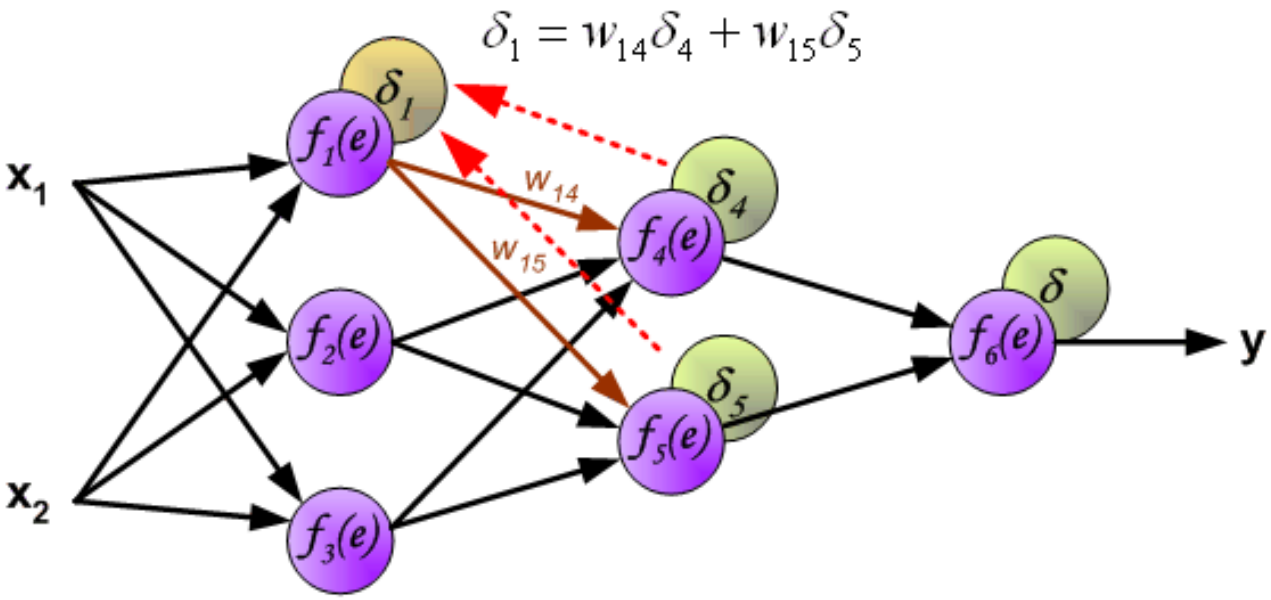
Learning Algorithm: Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



Backpropagation

The weights' coefficients w_{mn} used to propagate errors back are equal to those used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from several neurons they are added. The illustration is below:

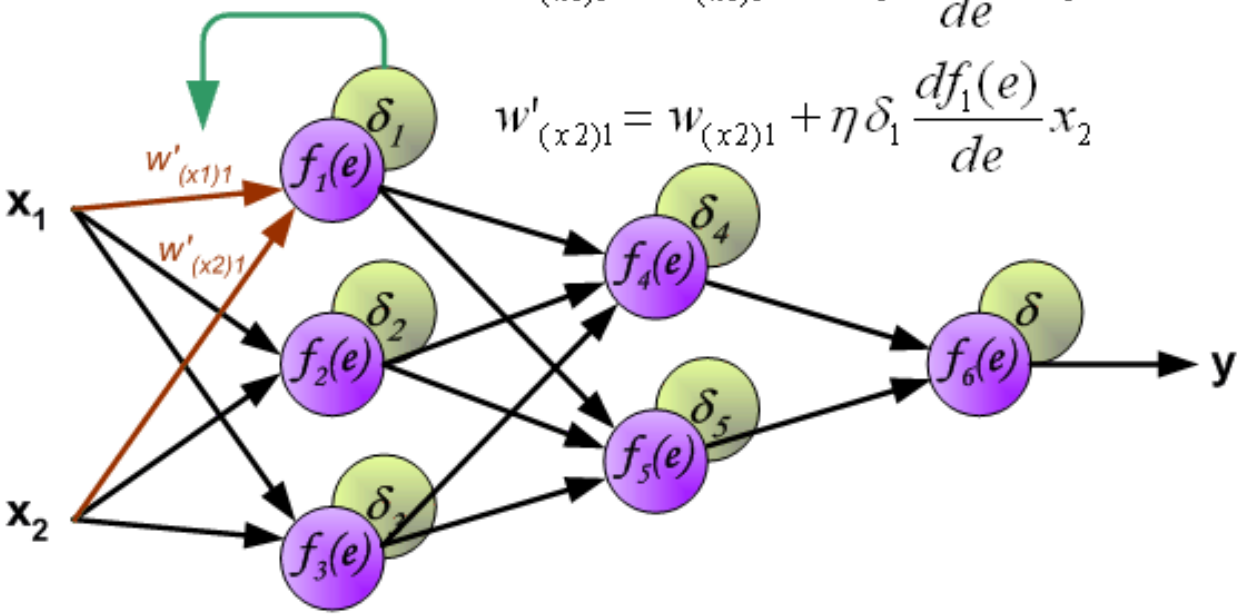


Backpropagation

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In the formulas below *df(e)/de* represents derivative of neuron activation function.

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

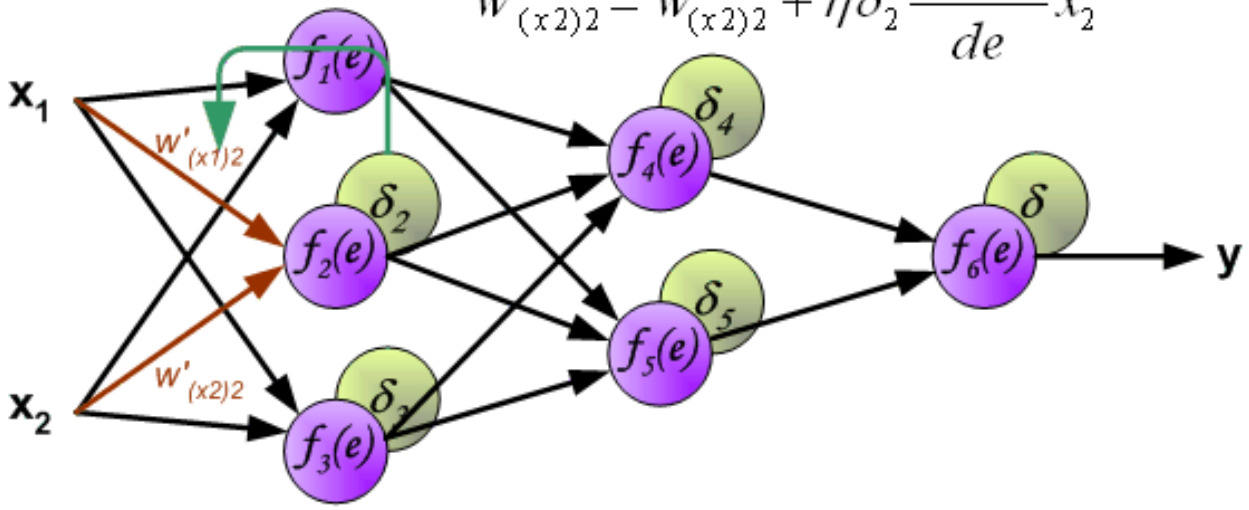
$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Backpropagation

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

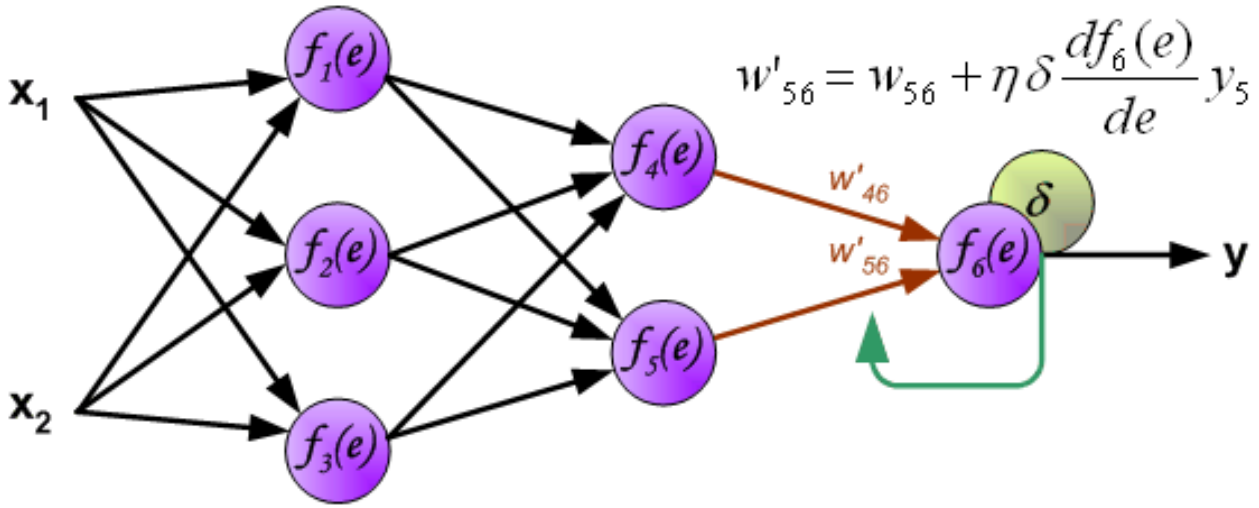
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



Learning Algorithm: Backpropagation

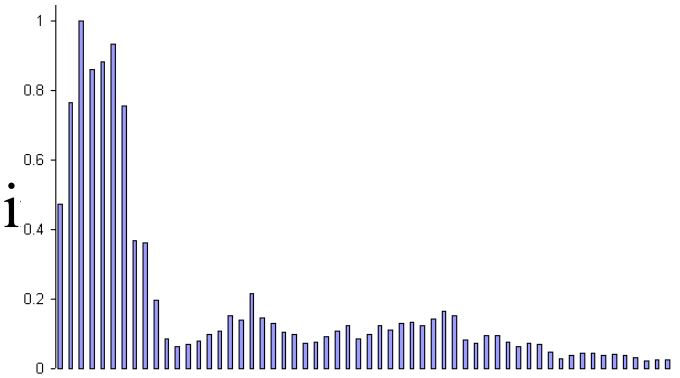
$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$

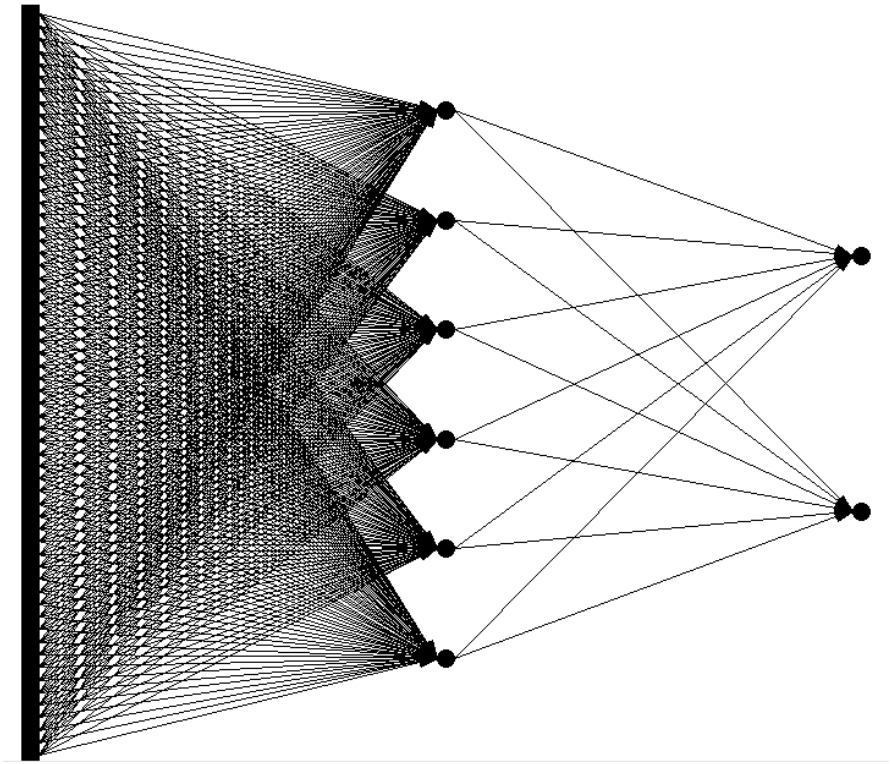


Example: Voice Recognition

- Task: Learn to discriminate between two different voices saying “Hello”
- Data
  - Sources
    - Steve
    - David
  - Format
    - Frequency distribution (60 bi
    - Analogy: cochlea

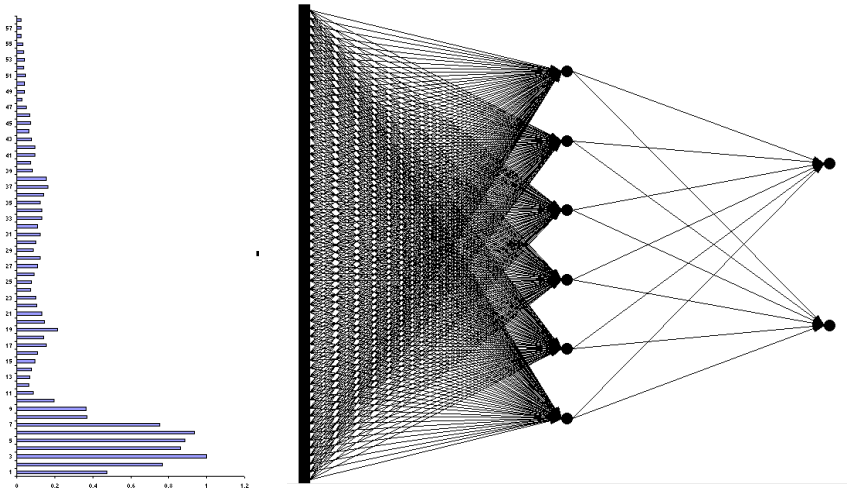


- Network architecture
  - Feed forward network
    - 60 input (one for each frequency bin)
    - 6 hidden
    - 2 output (0-1 for “Steve”, 1-0 for “David”)

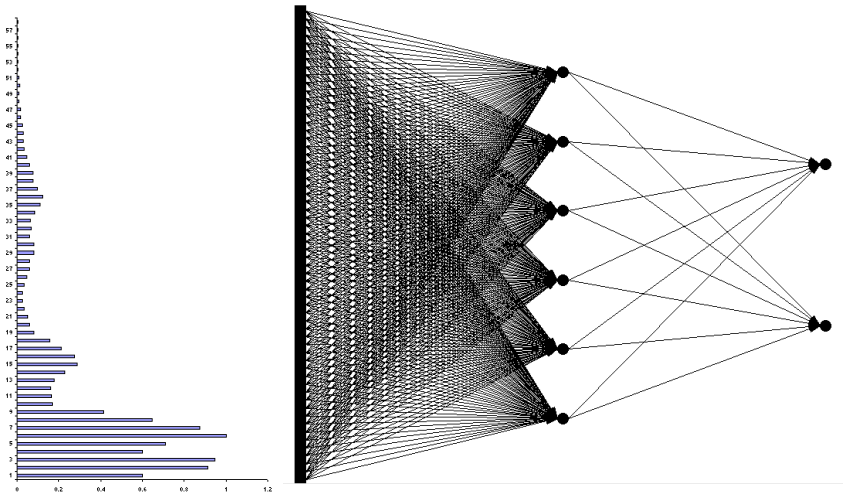


- Presenting the data

Steve

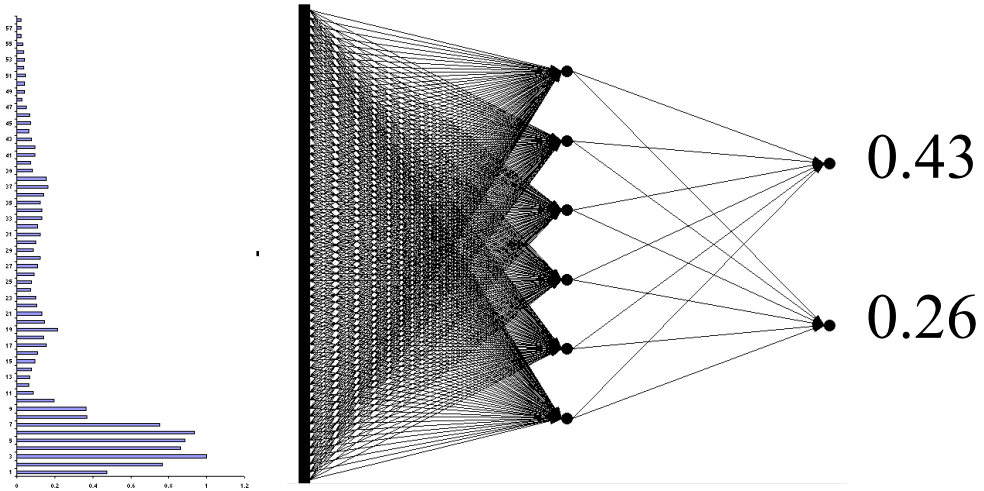


David

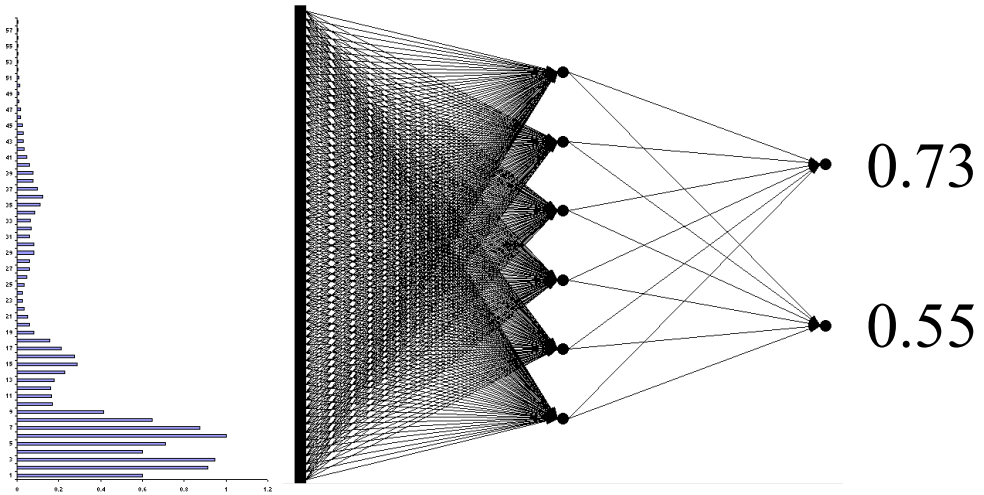


- Presenting the data (untrained network)

Steve

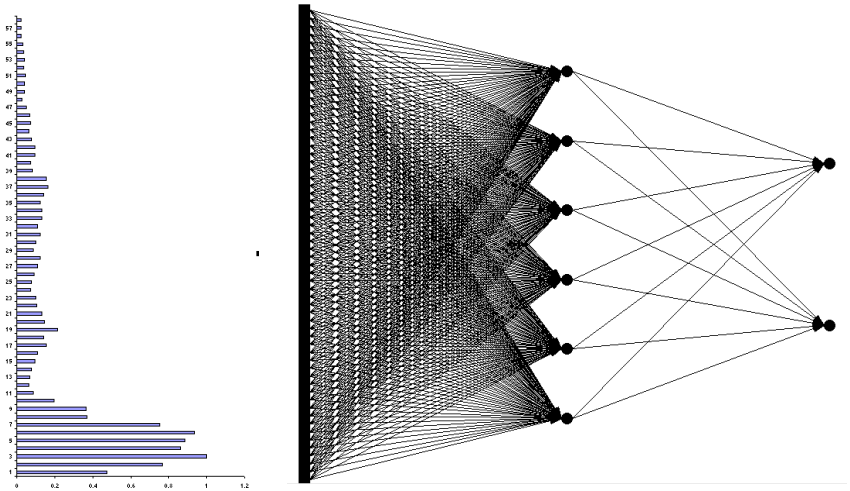


David



- Calculate error

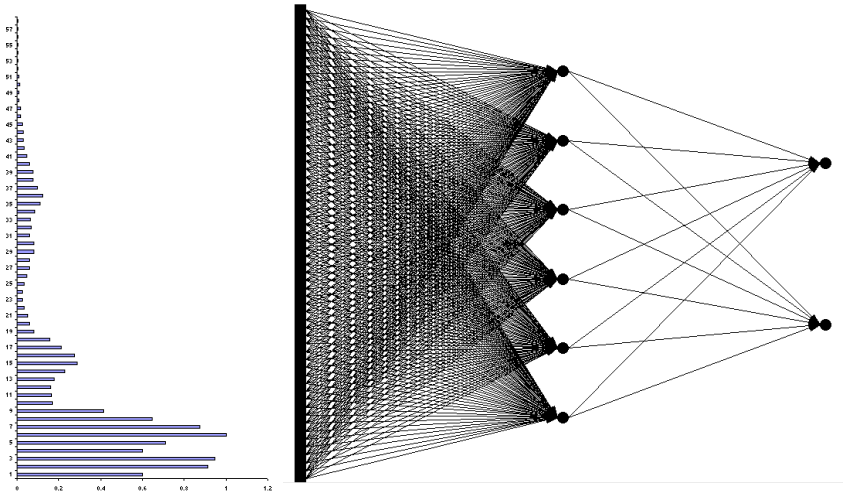
Steve



$0.43 - 0 = 0.43$

$0.26 - 1 = 0.74$

David

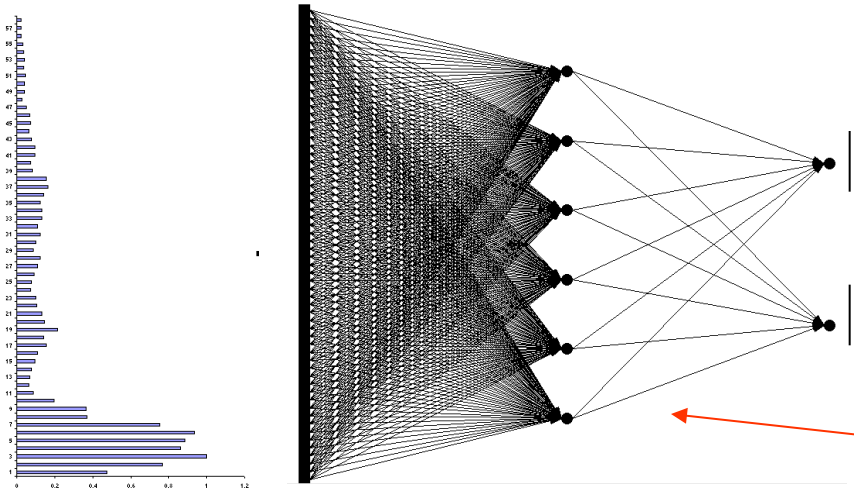


$0.73 - 1 = 0.27$

$0.55 - 0 = 0.55$

- Backprop error and adjust weights

Steve

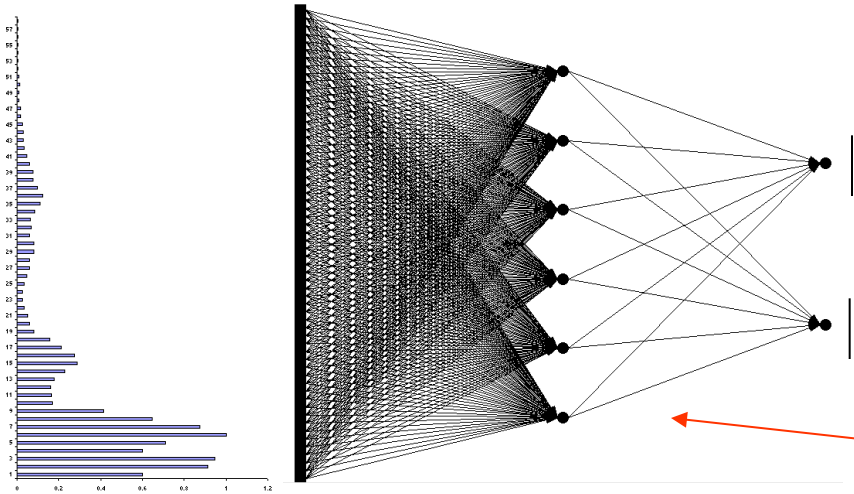


$$|0.43 - 0| = 0.43$$

$$|0.26 - 1| = 0.74$$

$$1.17$$

David



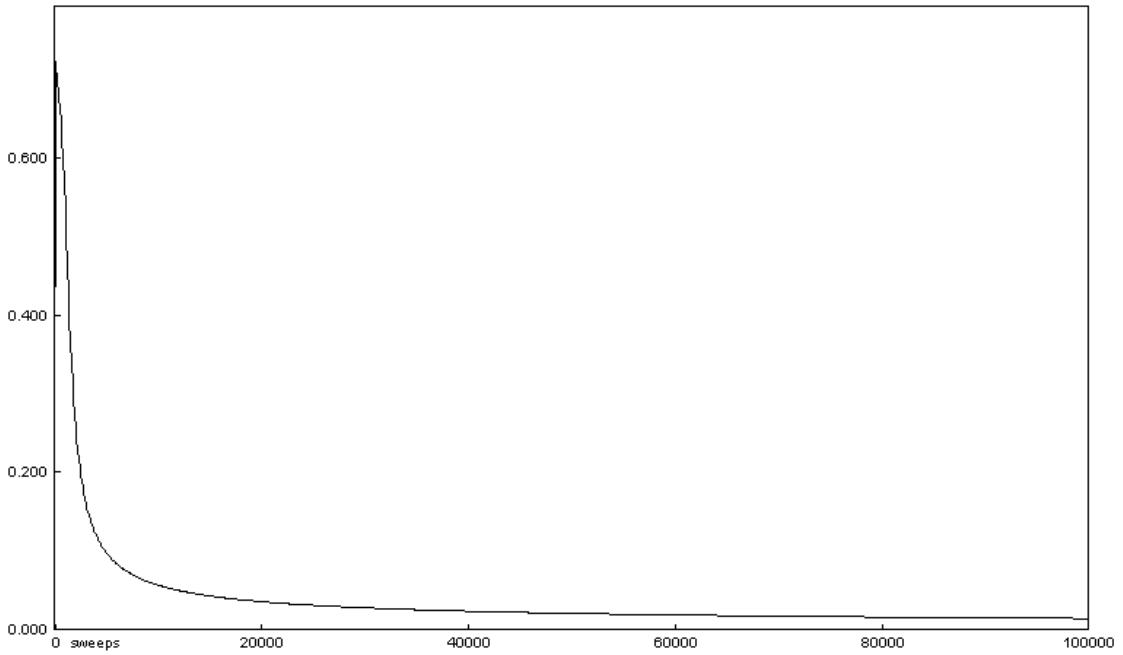
$$|0.73 - 1| = 0.27$$

$$|0.55 - 0| = 0.55$$

$$0.82$$

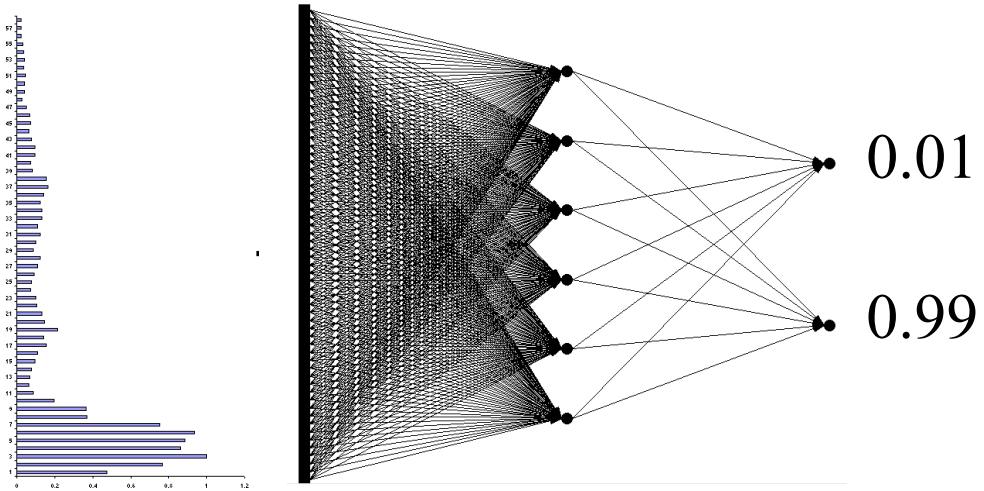
- Repeat process (sweep) for all training pairs
  - Present data
  - Calculate error
  - Backpropagate error
  - Adjust weights

- Repeat process multiple times

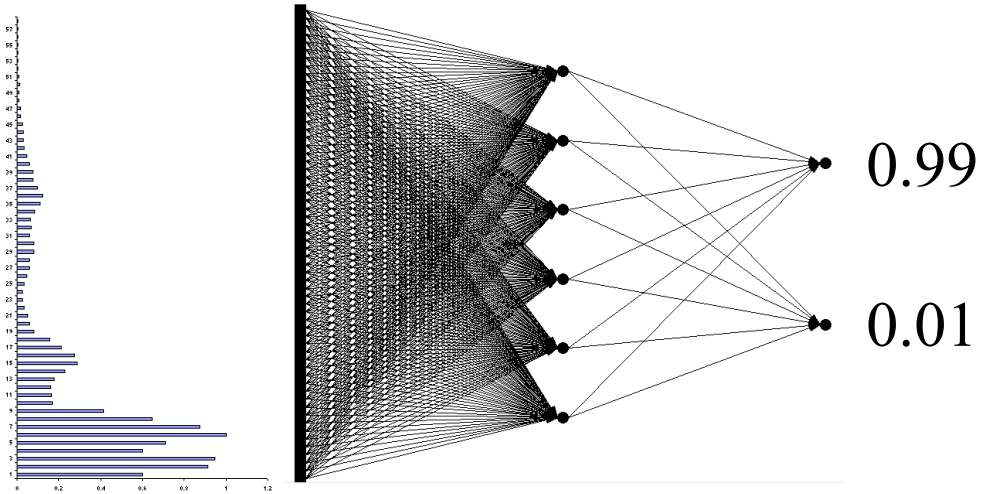


- Presenting the data (trained network)

Steve



David



- Results – Voice Recognition
  - Performance of trained network
    - Discrimination accuracy between known “Hello”s
      - 100%
    - Discrimination accuracy between new “Hello”’s
      - 100%

- Results – Voice Recognition (ctnd.)
  - Network has learned to generalise from original data
  - Networks with different weight settings can have same functionality
  - Trained networks ‘concentrate’ on lower frequencies (In the case of the sound example)
  - Network is robust against non-functioning nodes

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

Presented by

Tugce Tasci, Kyunghee Kim

05/18/2015

Additions/Changes made by Brian Funt March 2016

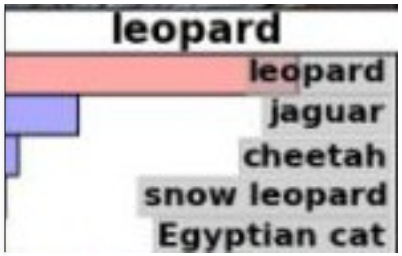
Original slides downloaded from
http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf

Outline

- Goal
- DataSet
- Architecture of the Network
- Reducing overfitting
- Learning
- Results
- Discussion

Goal

Classification



ImageNet

- Over 15M labeled high resolution images
- Roughly 22K categories
- Collected from web and labeled by Amazon Mechanical Turk

Tugce Tasci, Kyunghee

http://image--
-net.org/

ImageNet

“ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.” http://image--net.org/

“WordNet is a lexical database for the English language. It groups English words into sets of synonyms called synsets, provides short definitions and usage examples, and records a number of relations among these synonym sets or their members. “ Wikipedia

ImageNet Publications

ImageNet: A Large-Scale Hierarchical Image Database
Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fe, IEEE CVPR 2009

ImageNet Large Scale Visual Recognition Challenge
Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei
IJCV 2015

ILSVRC

ImageNet Large Scale Visual Recognition Competition

ILSVRC

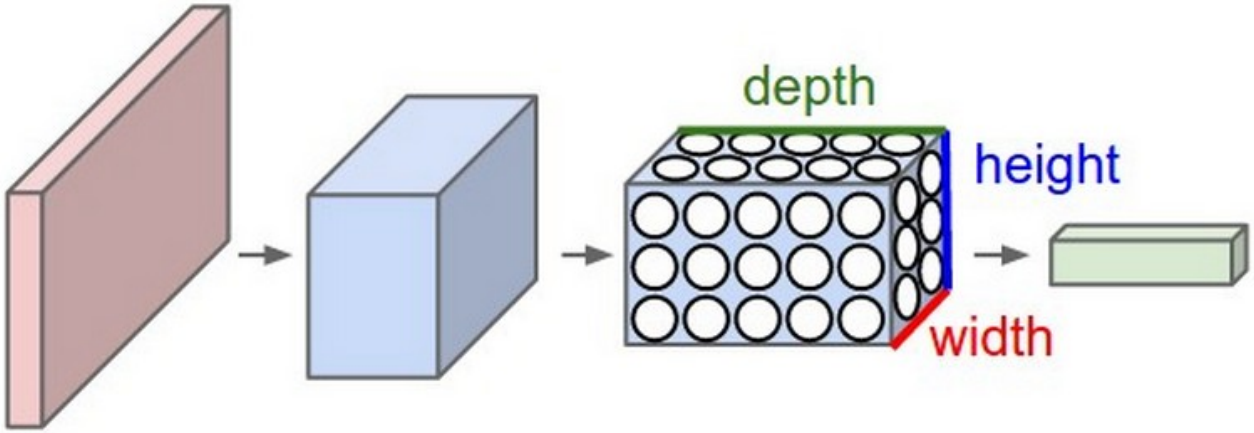
- Annual competition of image classification at large scale
- 1.2M images in 1K categories
- Classification: make 5 guesses about the image label

EntleBucher

Appenzeller

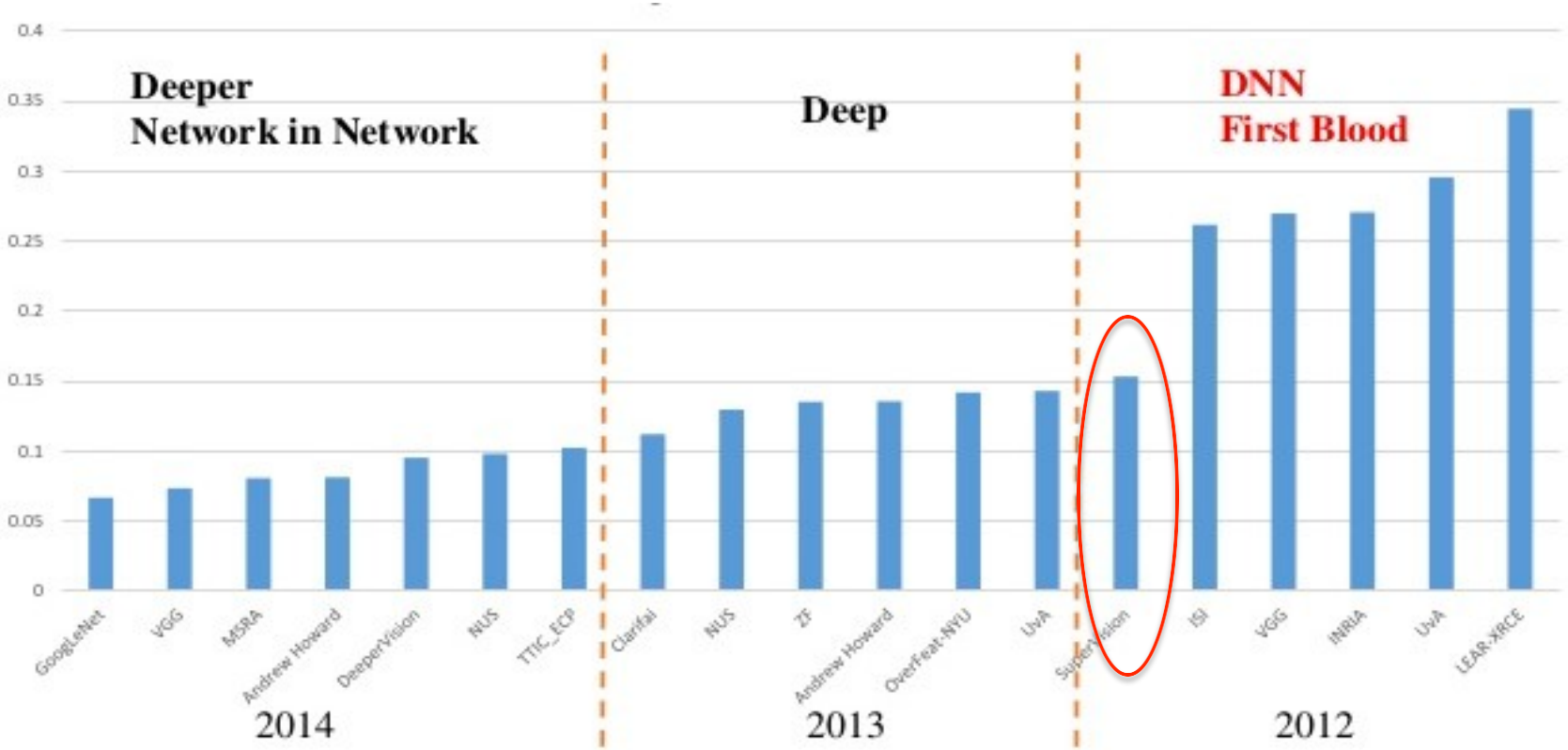
Convolutional Neural Networks

- Model with a large learning capacity
- Prior knowledge to compensate all data we do not have



ILSVRC

ImageNet Classification error throughout years and groups



Li Fei-Fei: ImageNet Large Scale Visual Recognition Challenge, 2014 http://image-net.org/

ILSVRC 2015 New York Times

“The Microsoft team was able to cut the number of errors in half in a task that required their program to classify objects from a set of 1,000 categories. The team also won a second competition by accurately detecting all instances of objects in 200 categories. The contest requires the programs to examine a large number of digital images, and either label or find objects in the images. For example, they may need to distinguish between objects such as bicycles and cars, both of which might appear to have two wheels from a certain perspective.”

MSRA– Microsoft Research Asia (Beijing)

SuperVision (SV)

Image classification with deep convolutional neural networks

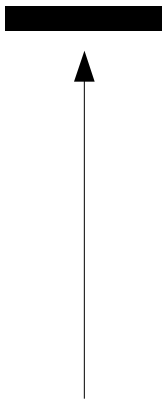
- 7 hidden “weight” layers
- 650K neurons
- 60M parameters
- 630M connections

- Rectified Linear Units, overlapping pooling, dropout trick
- Randomly extracted 224x224 patches for more data

Input representation

- Centered (0-mean) RGB values.

An input image (256x256)

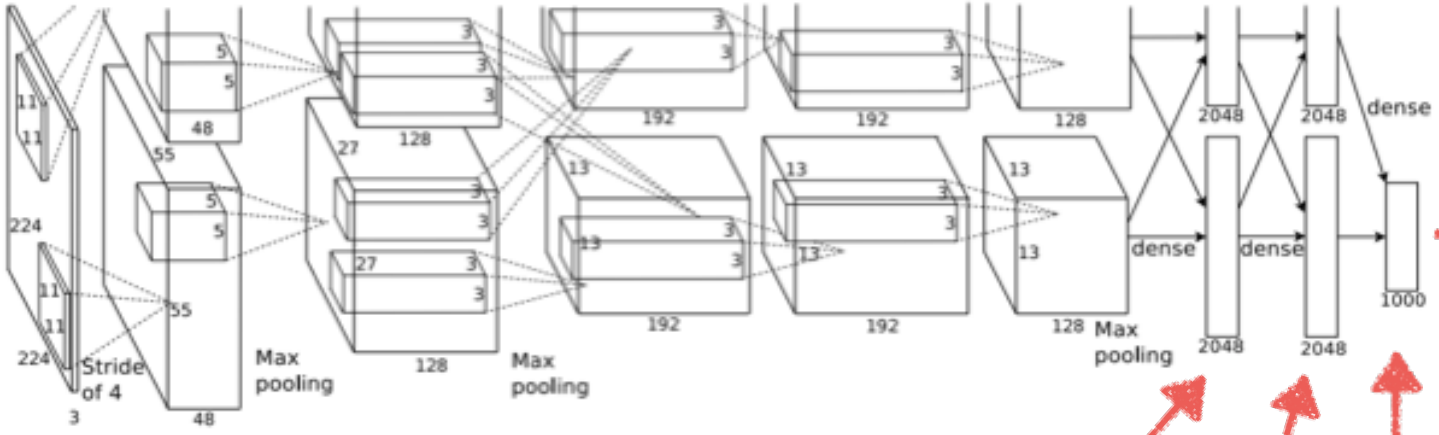


Minus sign

The mean input image

Architecture

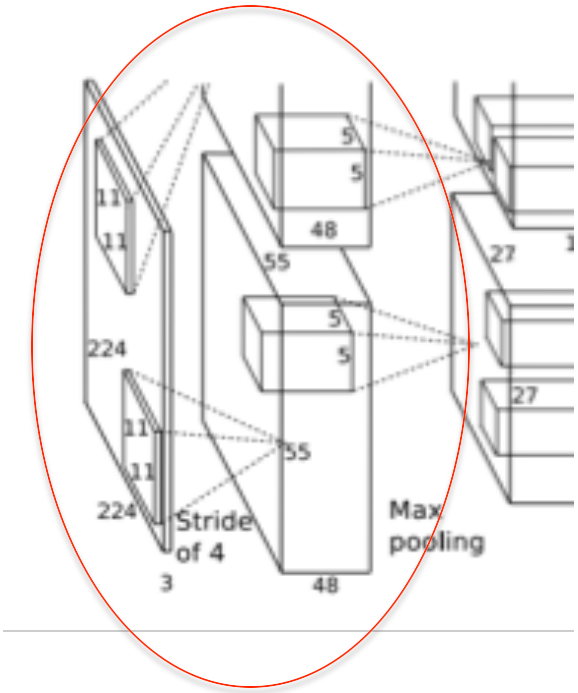
5 Convolutional Layers



1000--
-way
softmax

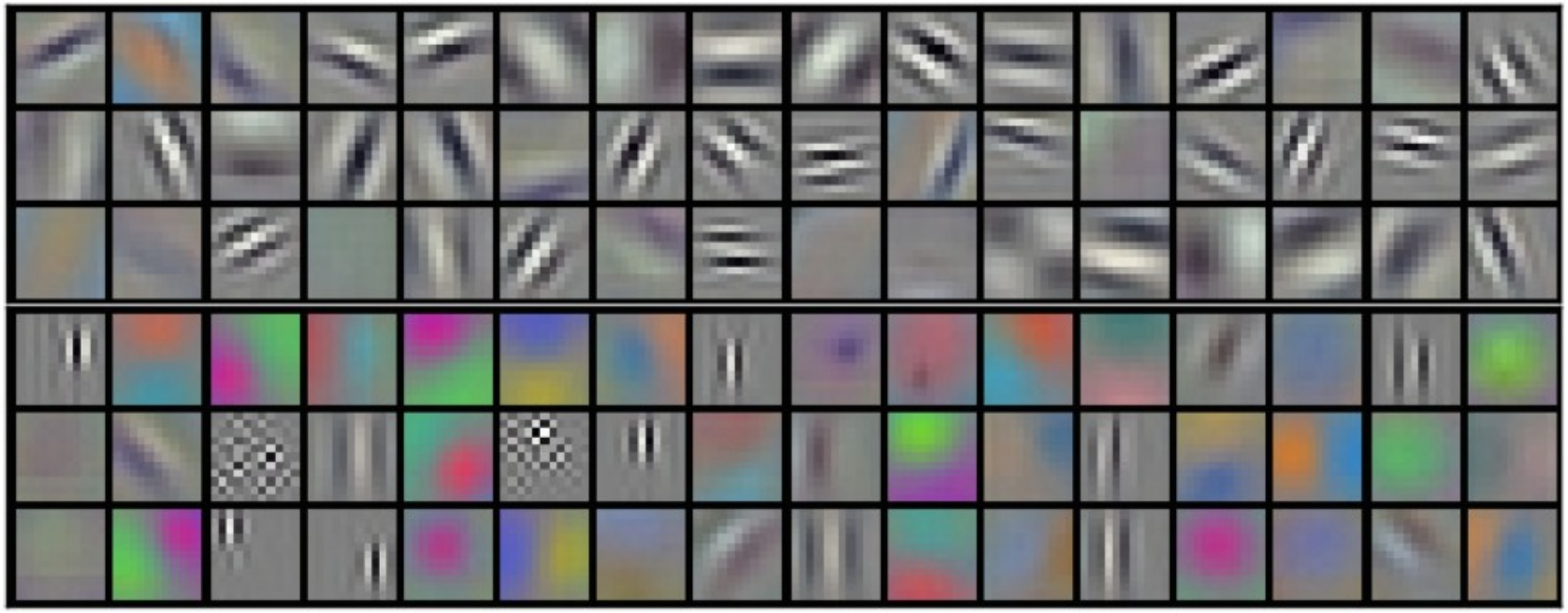
3 Fully Connected Layers

Layer 1 (Convolutional)



- Images: 227x227x3
- F (receptive field size): 11
- S (stride) = 4
- Conv layer output: 55x55x96

96 Learned Convolutional Kernels

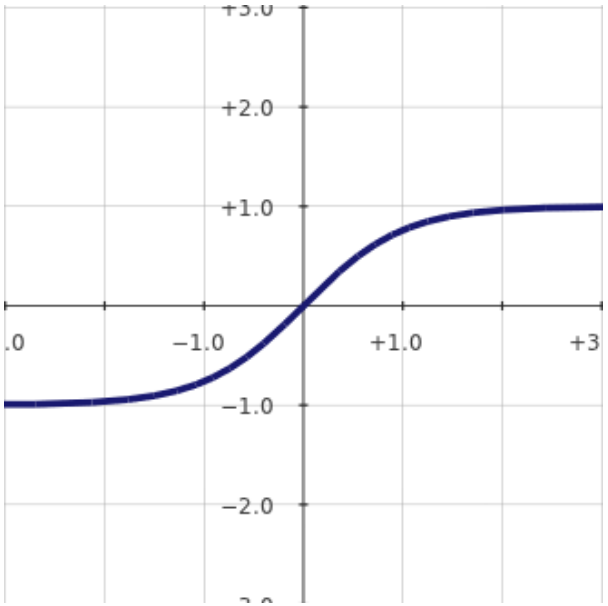


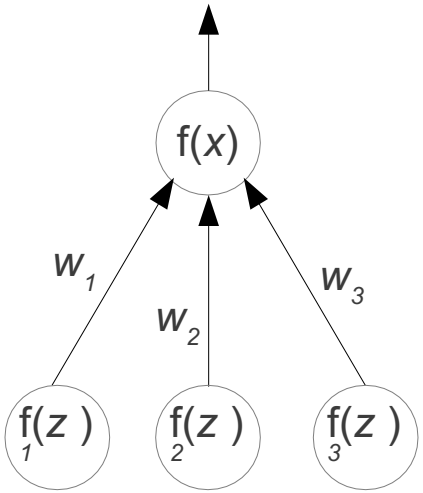
- 11 x 11 x 3 size kernels.

Tugce Tasci, Kyunghee Kim quoting from Krizhevsky et al. NIPS2012

Neurons

f(*x*) = tanh(*x*)

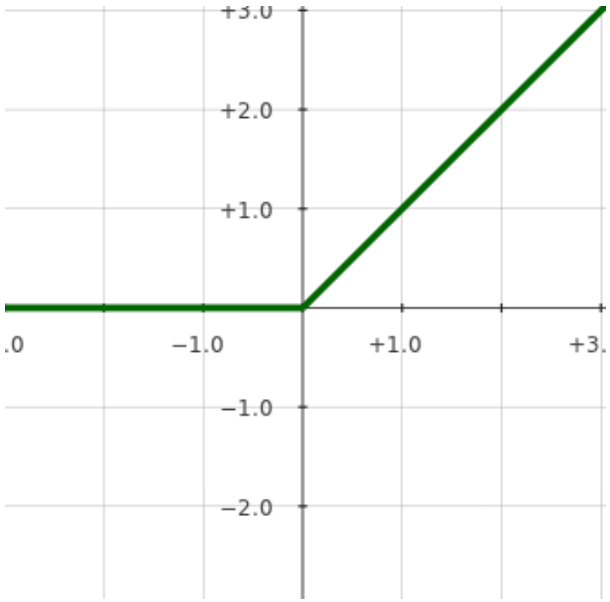




$x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3)$

x is called the total input to the neuron, and f(*x*) is its output

f(*x*) = max(0, *x*)

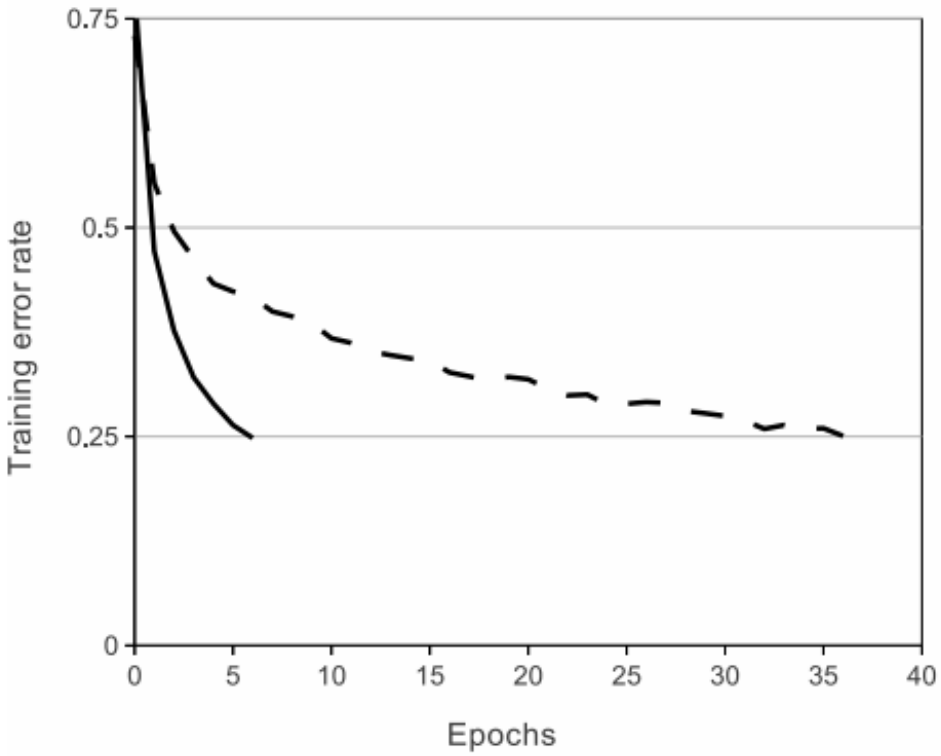


Very bad (slow to train)

Very good (quick to train)

Architecture

RELU Nonlinearity



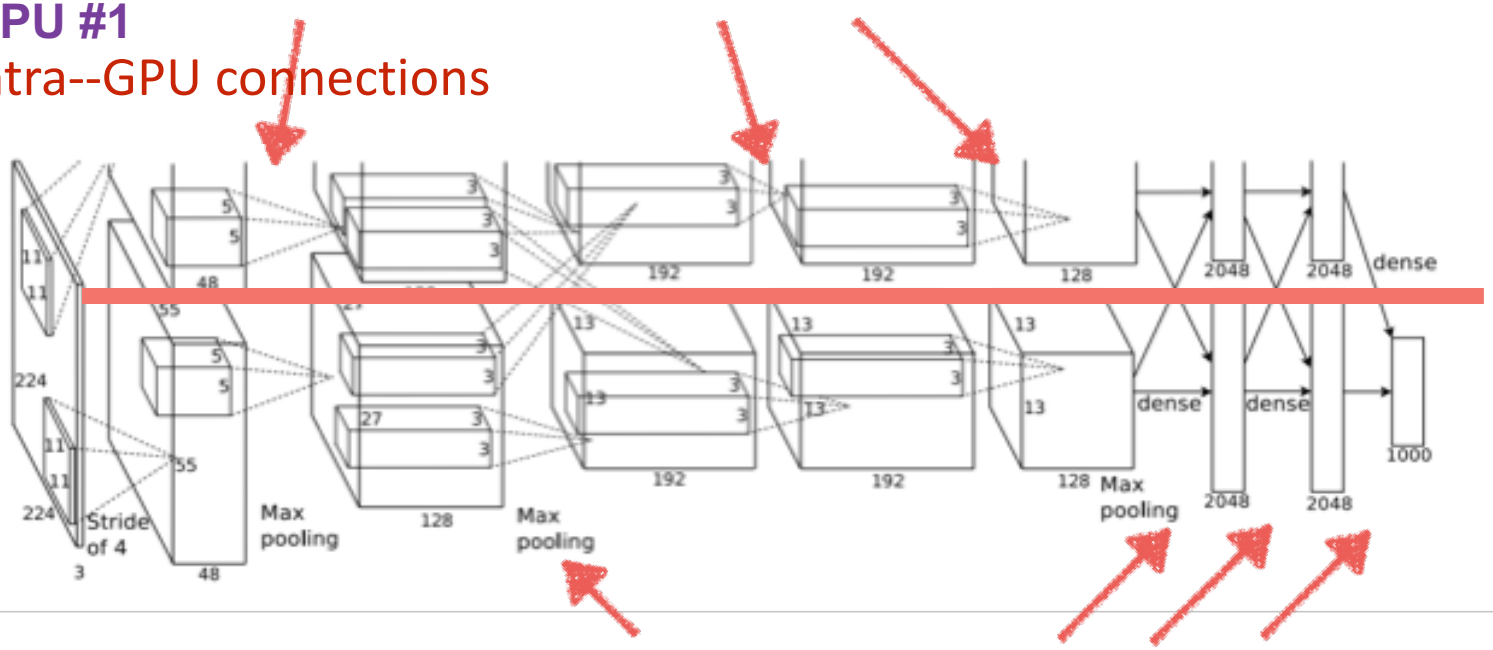
A 4-layer CNN with ReLUs (solid line) converges six times faster than an equivalent network with tanh neurons (dashed line) on CIFAR-10 dataset

Architecture

Training on Multiple GPUs

GPU #1

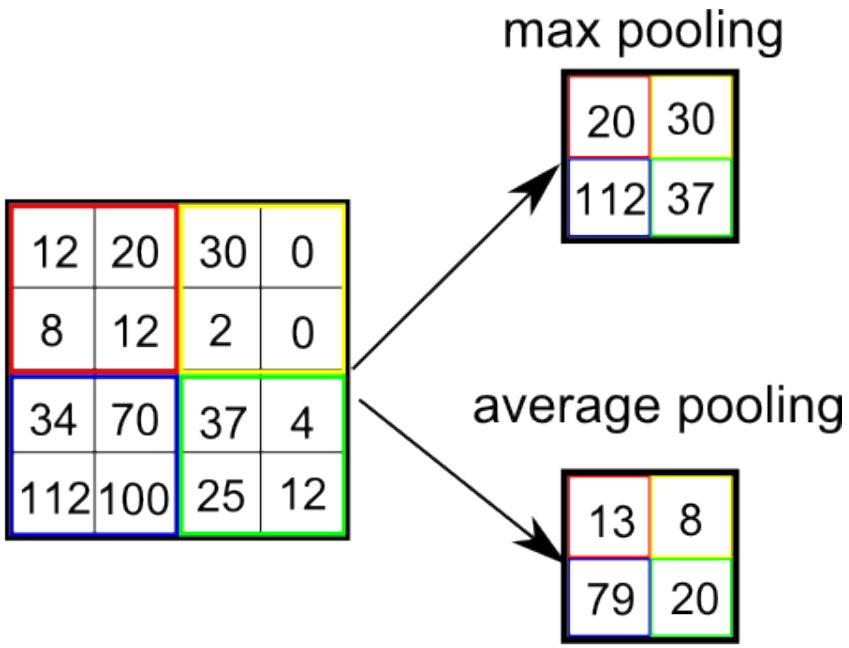
intra--GPU connections



GPU #2

inter--GPU connections

Pooling



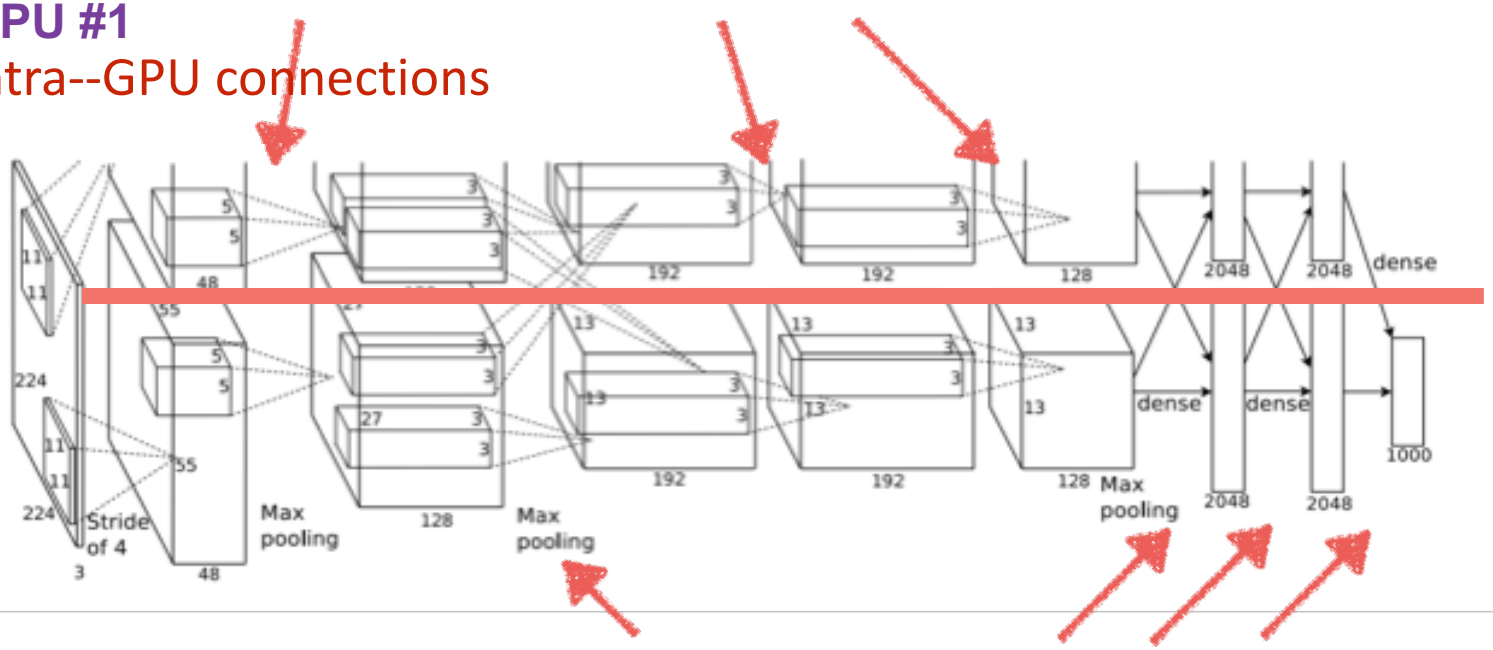
http://vaaaaaanquish.hatenablog.com/entry/2015/01/26/060622

Architecture

Training on Multiple GPUs

GPU #1

intra--GPU connections



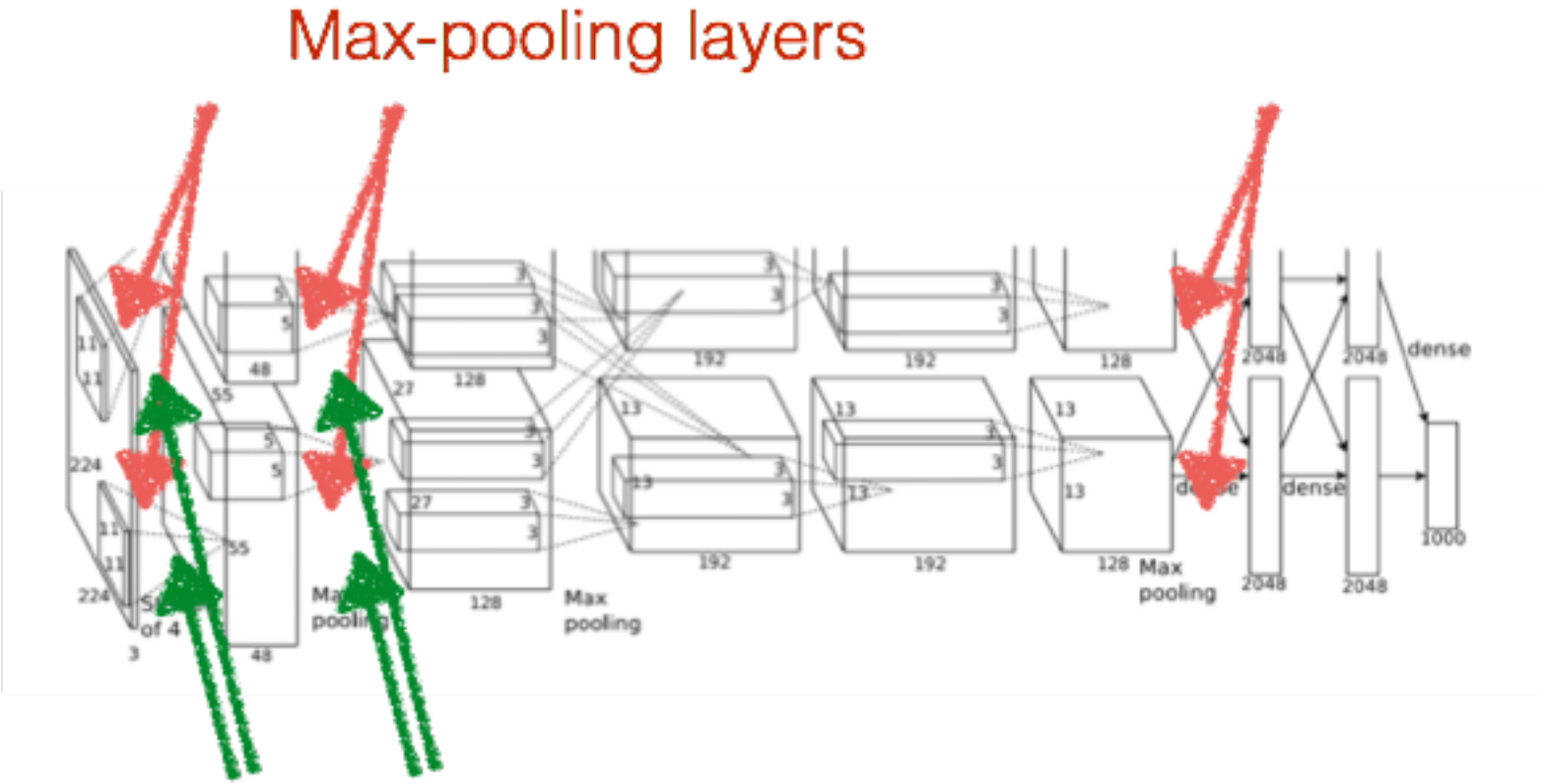
GPU #2

inter--GPU connections

Top-1 and Top-5 error rates decreases by 1.7% & 1.2% respectively, comparing to the net trained with one GPU and half neurons!!

Architecture

Overlapping Pooling



Tugce Tasci, Kyunghee

Architecture

Local Response Normalization

•No need to input normalization with ReLUs.
•But still the following local normalization scheme helps generalization.

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

Response--normalized activity

Activity of a neuron computed by applying kernel at position (x,y) and then applying the ReLU nonlinearity

- Response normalization reduces top-1 and top-5 error rates by 1.4% and 1.2% , respectively.

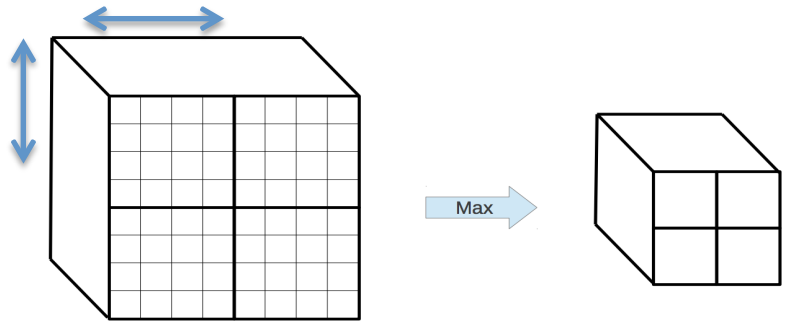
Architecture

Overlapping Pooling

- Traditional pooling (s = z)

s

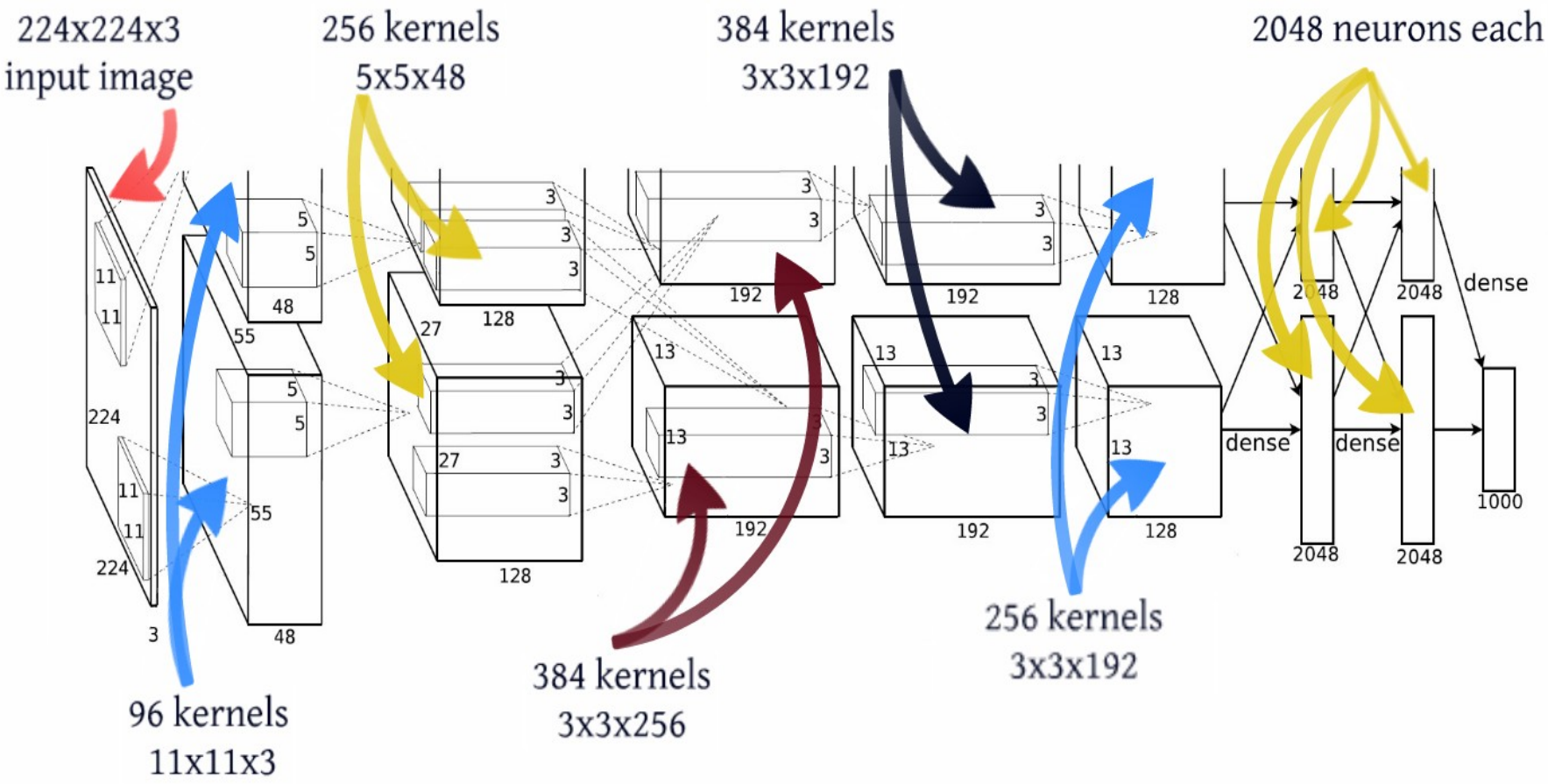
z



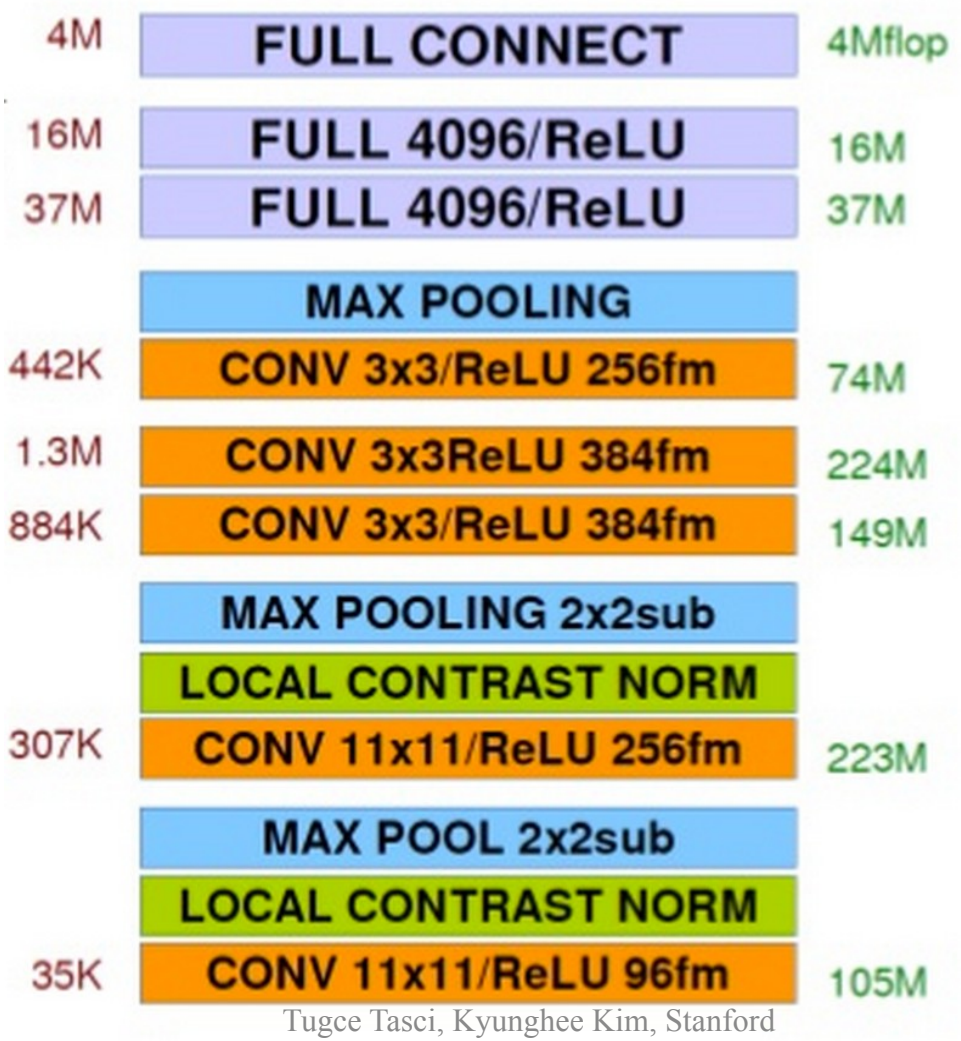
- s < z ➔➔ overlapping pooling
- top-1 and top-5 error rates decrease by 0.4% and 0.3%, respectively, compared to the non-overlapping scheme s = 2, z = 2

Architecture

.



Architecture Overview



Reducing Overfitting

Data Augmentation

•• 60 million parameters, 650,000 neurons

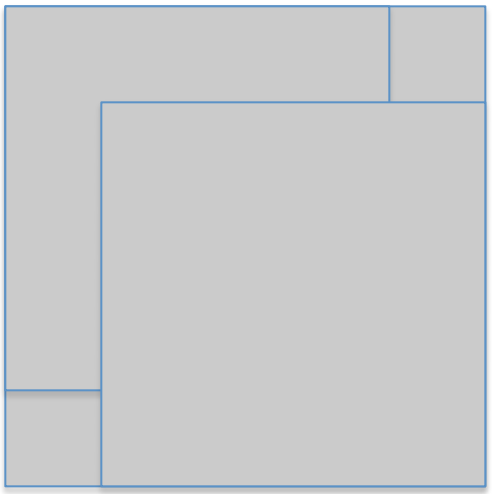
→→ Overfits a lot.

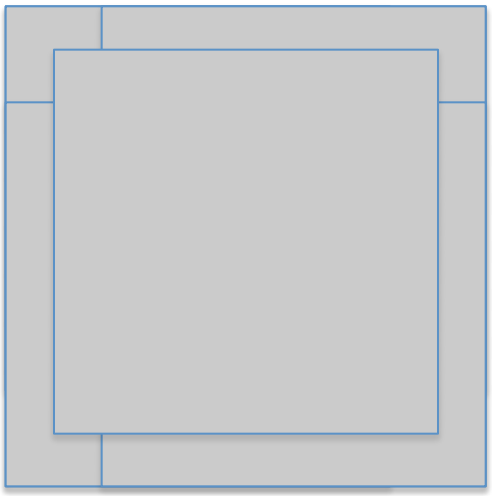
•• Crop 224x224 patches (and their horizontal reflections.)

Reducing Overfitting

Data Augmentation

•• At test time, average the predictions on the 10 patches.





Reducing Overfitting

Data Augmentation

- Change the intensity of RGB channels
-

$$I_{xy} = [I_{xy}^R,\ I_{xy}^G,\ I_{xy}^B]^T$$

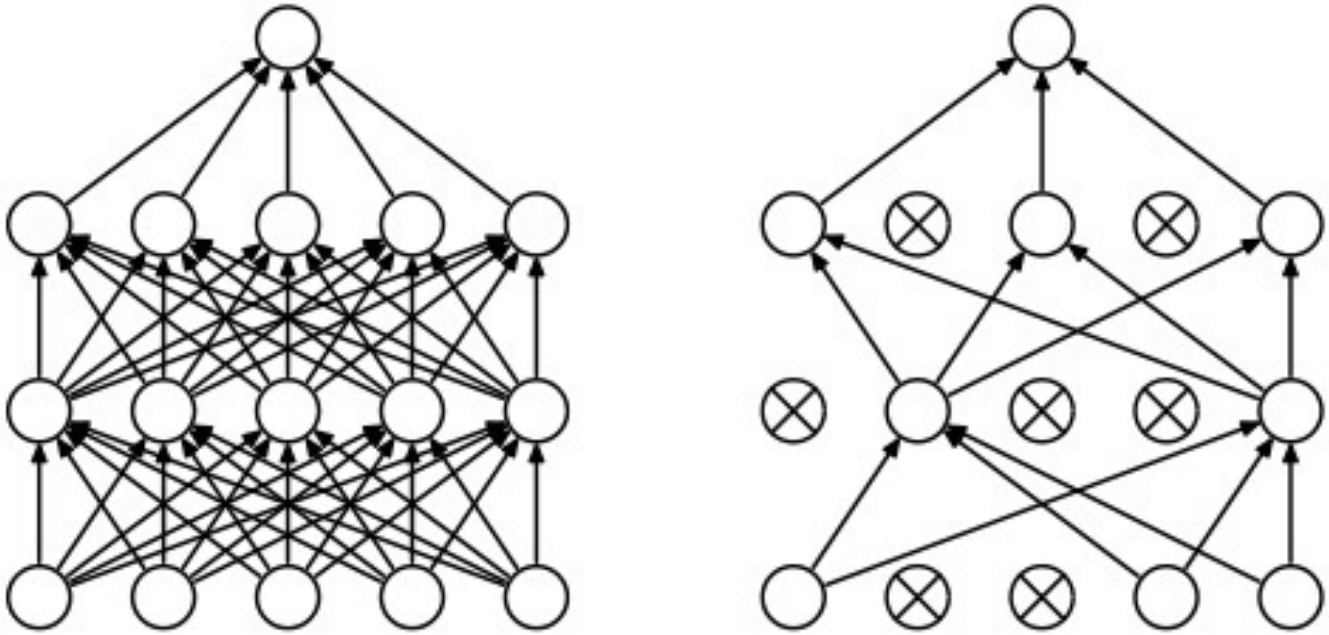
add multiples of principle components

$$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

$$\alpha_i \sim N(0,\ 0.1)$$

Reducing Overfitting

Dropout



Standard Neural Net

After applying dropout.

- With probability 0.5
- last two 4096 fully-connected layers.

Figure credit from Srivastava et al.

Stochastic Gradient Descent Learning

Momentum Update

momentum(damping parameter)

Learning rate (initialized at 0.01)

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

weight decay

Gradient of Loss
w.r.t weight
Averaged over batch

Batch size: 128

- The training took 5 to 6 days on two NVIDIA GTX 580 3GB GPUs.

Results : ILSVRC-2010

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

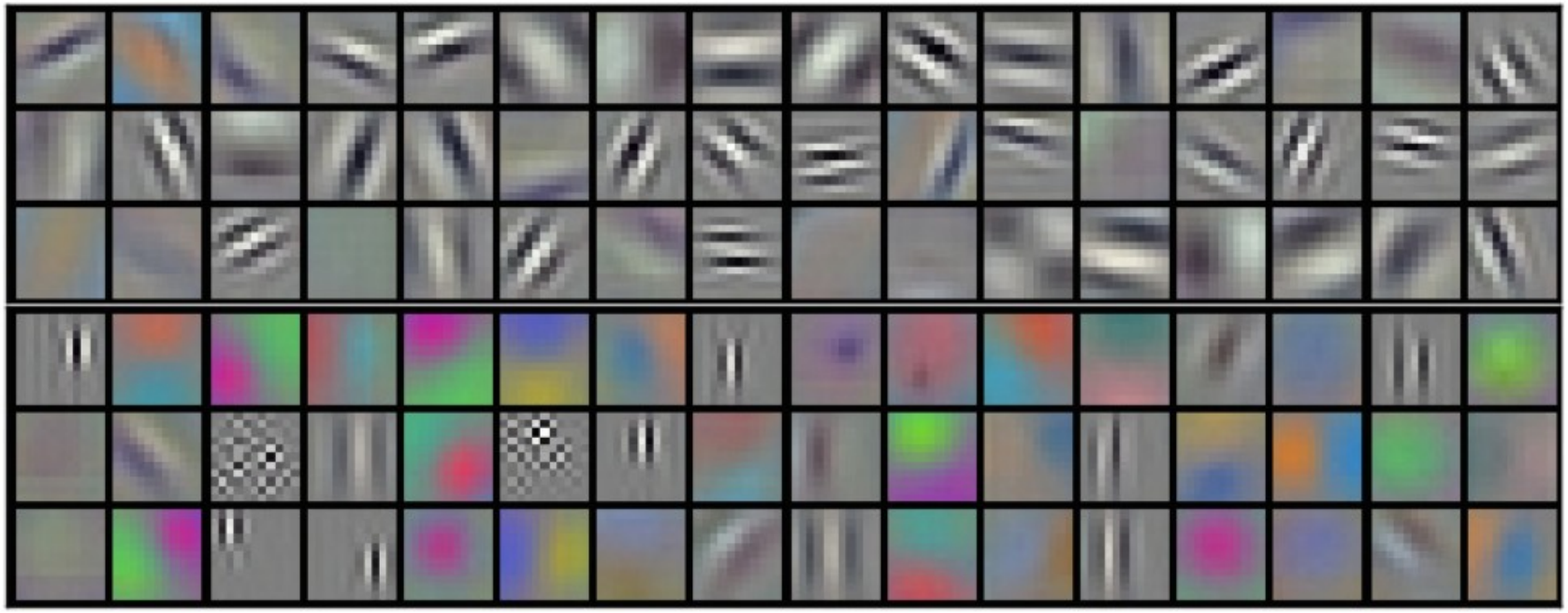
Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

Results : ILSVRC-2012

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

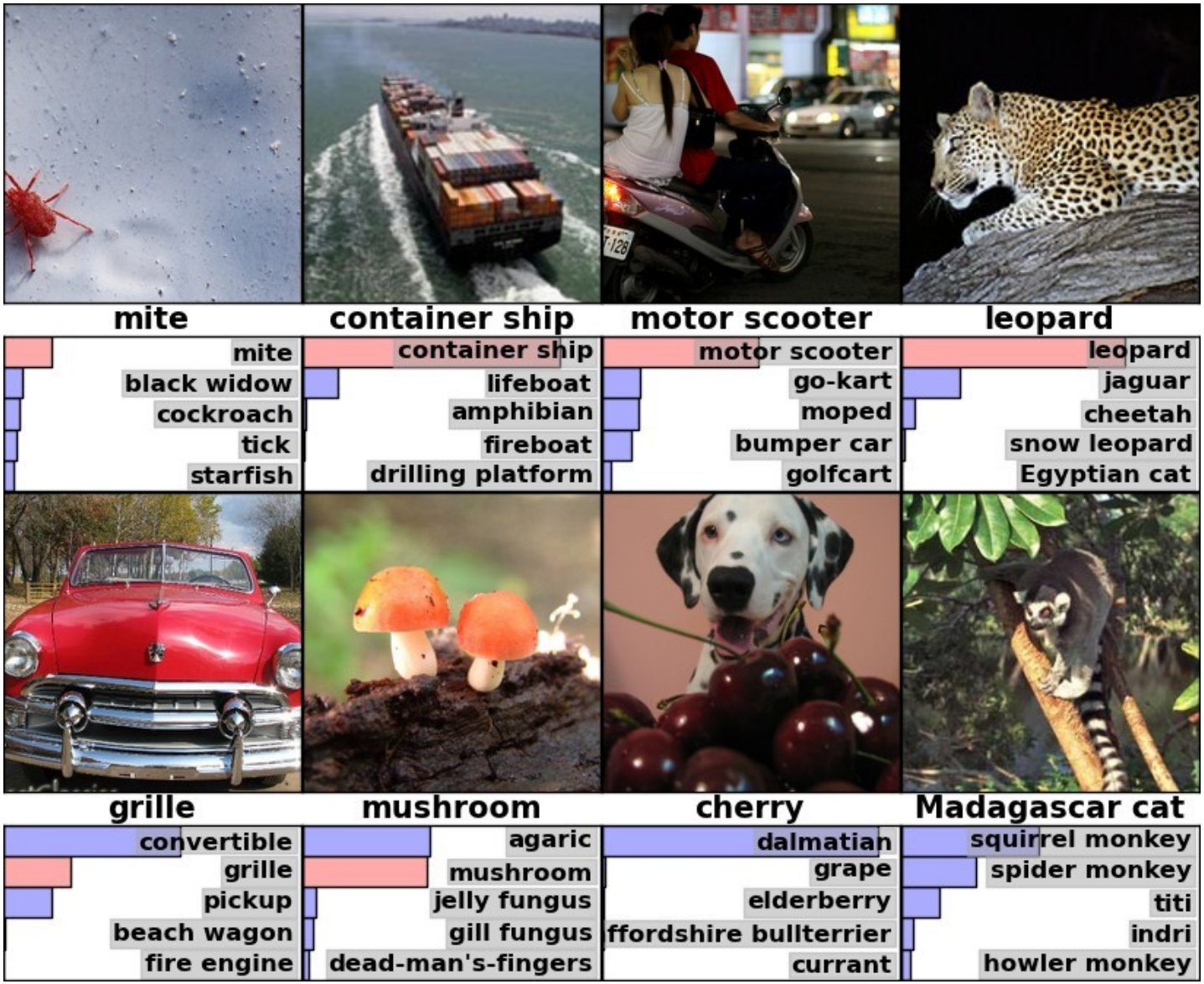
96 Convolutional Kernels



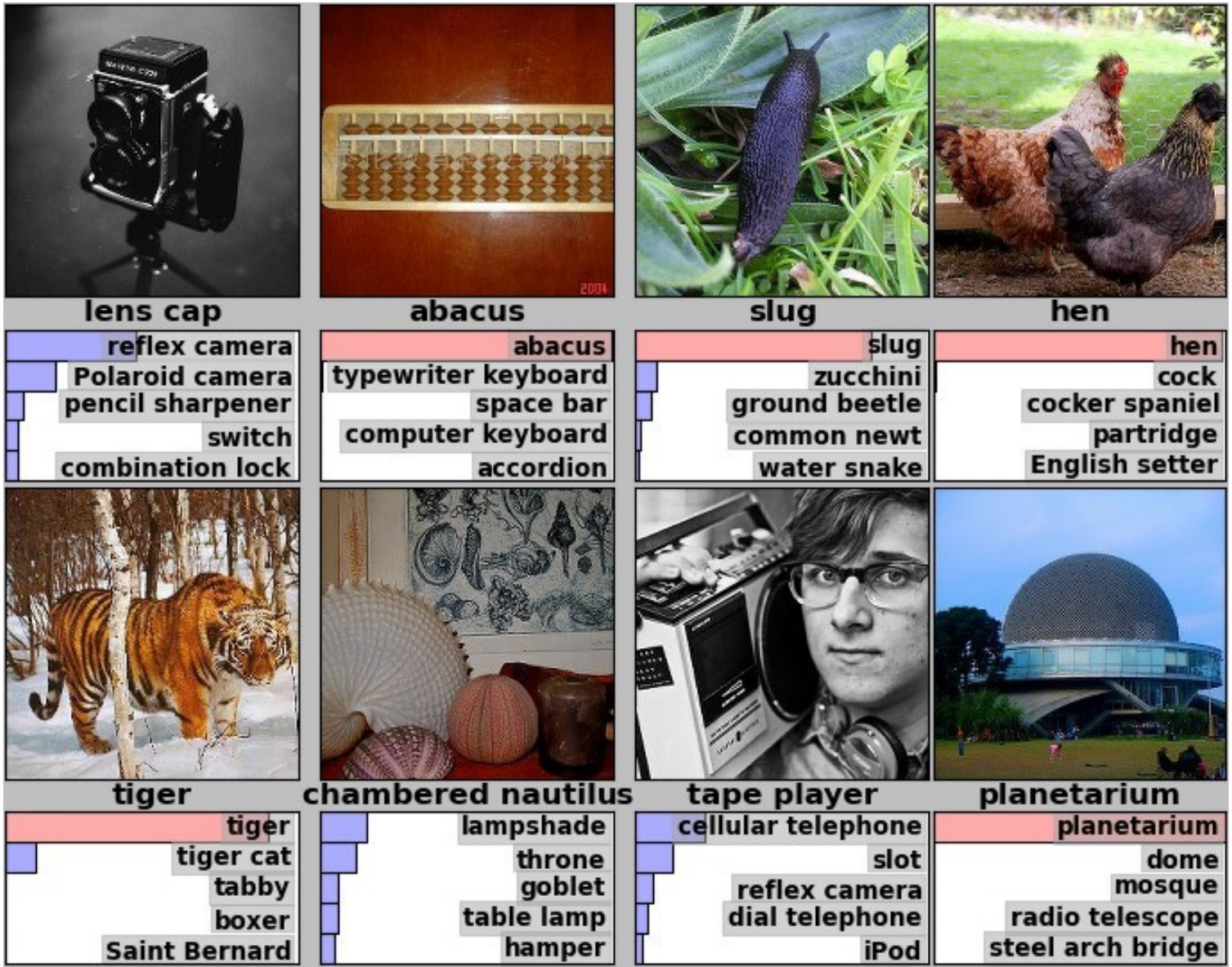
- 11 x 11 x 3 size kernels.
- top 48 kernels on GPU 1 : color-agnostic
- bottom 48 kernels on GPU 2 : color-specific.

Tugce Tasci, Kyunghee Kim quoting from Krizhevsky et al. NIPS2012

Validation


mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
mushroom
cherry
Madagascar cat
convertible
grille
pickup
beach wagon
fire engine
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
dalmatian
grape
elderberry
ffordshire bullterrier
currant
squirrel monkey
spider monkey
titi
indri
howler monkey

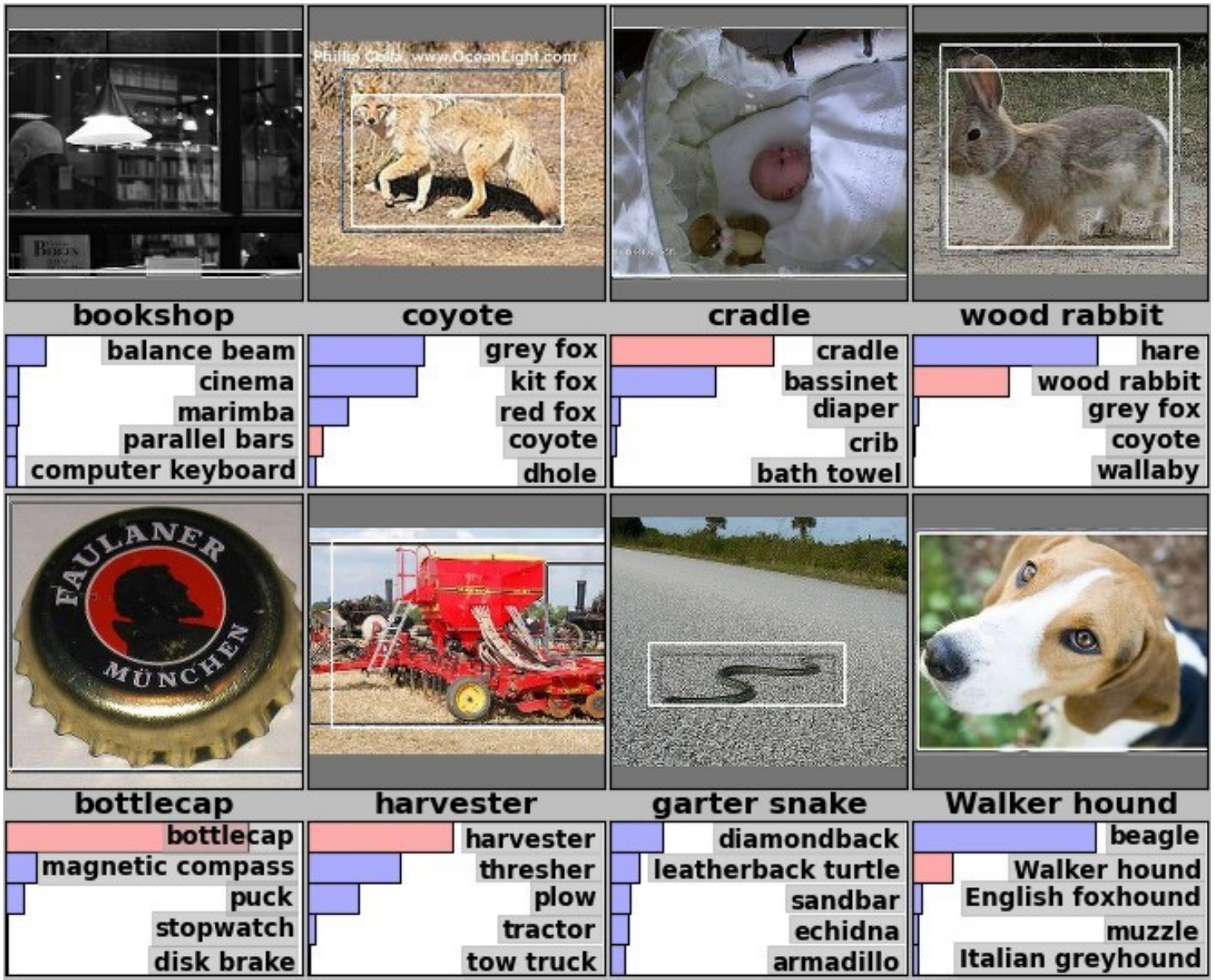
Validation


lens cap
reflex camera
Polaroid camera
pencil sharpener
switch
combination lock
abacus
abacus
typewriter keyboard
space bar
computer keyboard
accordion
slug
slug
zucchini
ground beetle
common newt
water snake
hen
hen
cock
cocker spaniel
partridge
English setter
tiger
tiger
tiger cat
tabby
boxer
Saint Bernard
chambered nautilus
lampshade
throne
goblet
table lamp
hamper
tape player
cellular telephone
slot
reflex camera
dial telephone
iPod
planetarium
planetarium
dome
mosque
radio telescope
steel arch bridge

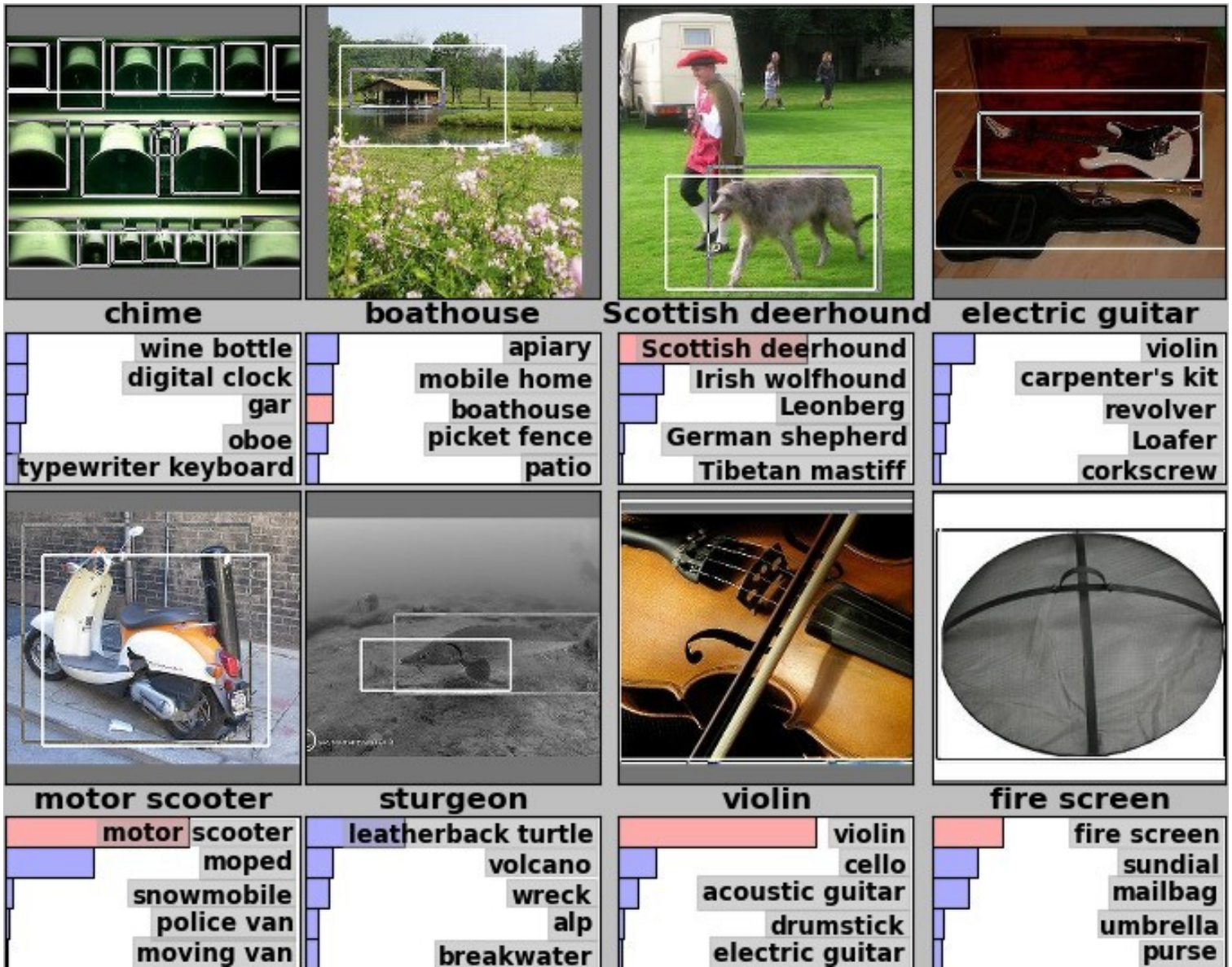
Validation


koala
wombat
Norwegian elkhound
wild boar
wallaby
koala
tiger
tiger
tiger cat
jaguar
lynx
leopard
European fire salamander
European fire salamander
spotted salamander
common newt
long-horned beetle
box turtle
loggerhead
African crocodile
Gila monster
loggerhead
mud turtle
leatherback turtle
seat belt
seat belt
ice lolly
hotdog
burrito
Band Aid
television
television
microwave
monitor
screen
car mirror
sliding door
sliding door
shoji
window shade
window screen
four-poster
wallaby
hare
wallaby
wood rabbit
Lakeland terrier
kit fox

Validation


bookshop
balance beam
cinema
marimba
parallel bars
computer keyboard
coyote
grey fox
kit fox
red fox
coyote
dhole
cradle
cradle
bassinet
diaper
crib
bath towel
wood rabbit
hare
wood rabbit
grey fox
coyote
wallaby
bottlecap
bottlecap
magnetic compass
puck
stopwatch
disk brake
harvester
harvester
thresher
plow
tractor
tow truck
garter snake
diamondback
leatherback turtle
sandbar
echidna
armadillo
Walker hound
beagle
Walker hound
English foxhound
muzzle
Italian greyhound

Validation


chime
wine bottle
digital clock
gar
oboe
typewriter keyboard
boathouse
apiary
mobile home
boathouse
picket fence
patio
Scottish deerhound
Scottish deerhound
Irish wolfhound
Leonberg
German shepherd
Tibetan mastiff
electric guitar
violin
carpenter's kit
revolver
Loafer
corkscrew
motor scooter
motor scooter
moped
snowmobile
police van
moving van
sturgeon
leatherback turtle
volcano
wreck
alp
breakwater
violin
violin
cello
acoustic guitar
drumstick
electric guitar
fire screen
fire screen
sundial
mailbag
umbrella
purse

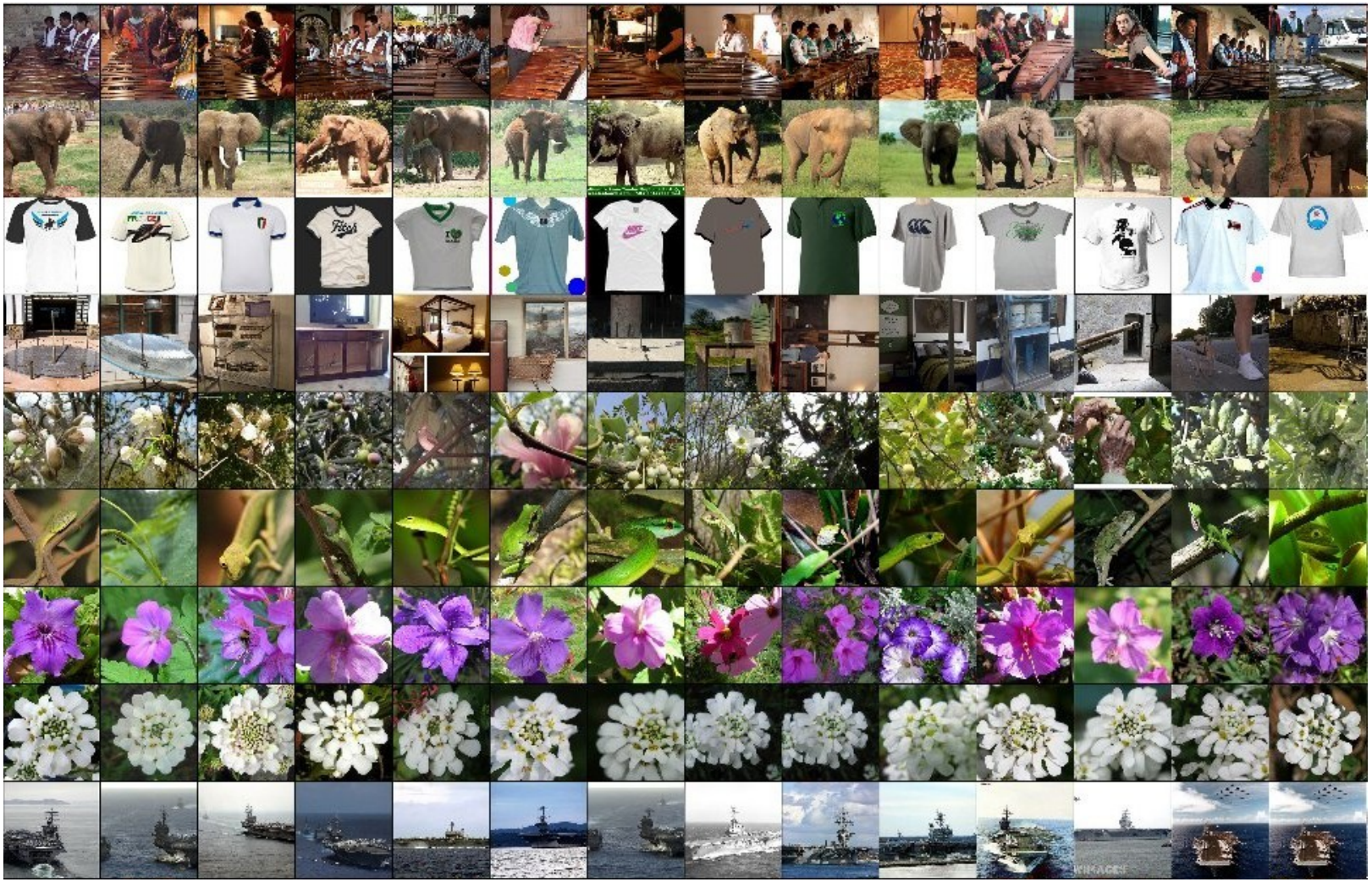
Retrieval Method:
Five ILSVRC-2010 test images in left column

The output from the last 4096 fully-connected layer

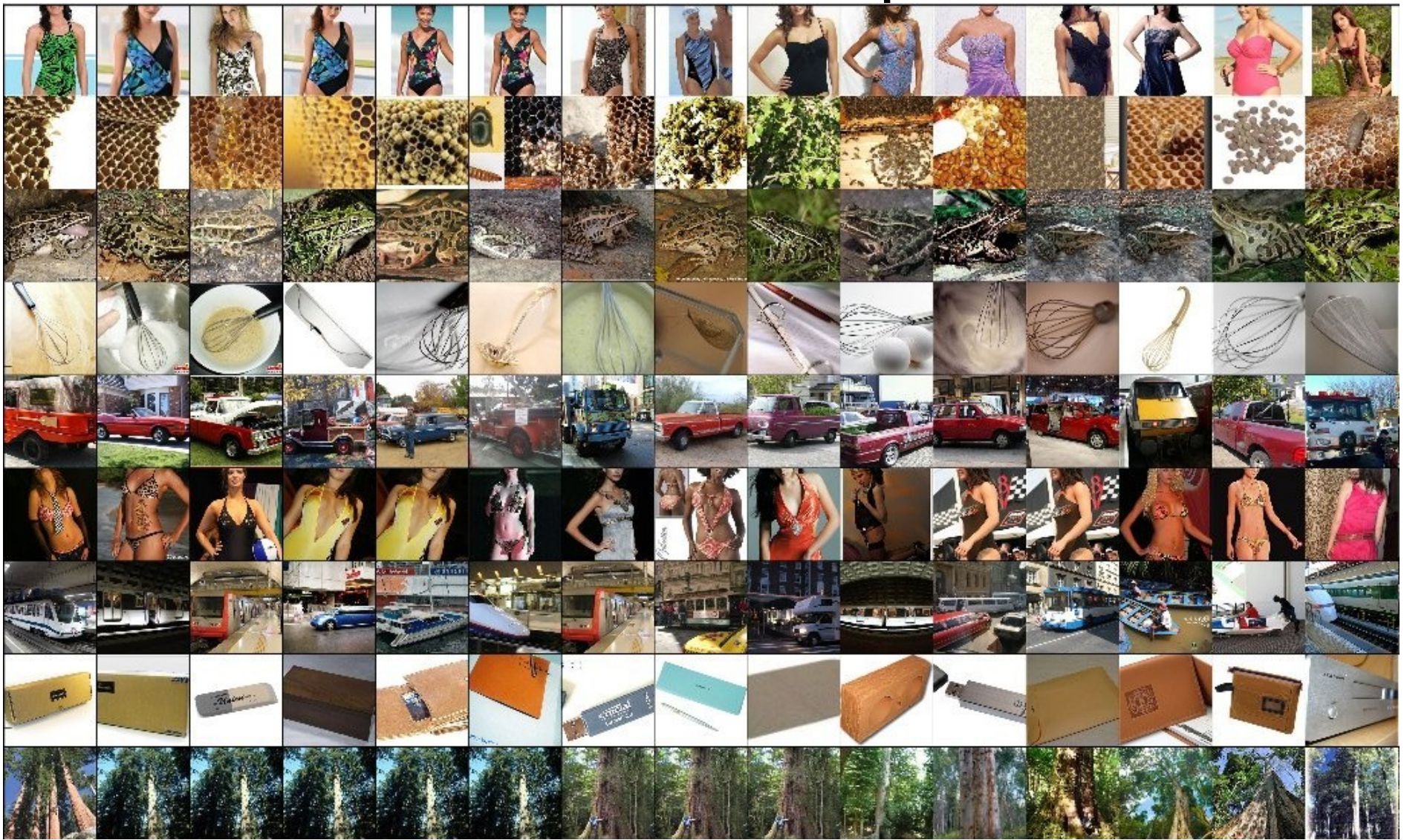
: 4096 dimensional feature.

Retrieval experiments

First column contains query images from ILSVRC-2010 test set, remaining columns contain retrieved images from training set.



Retrieval



Discussion

•• Depth is really important.

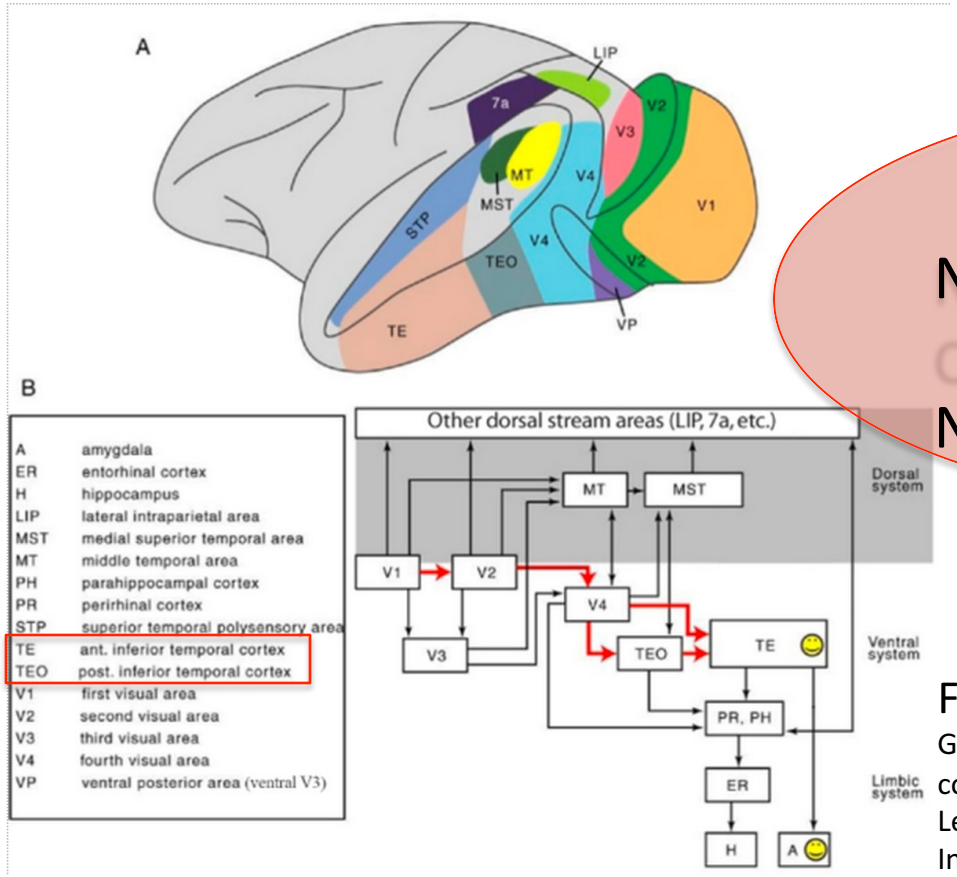
removing a single convolutional layer degrades the performance.

K. Simonyan, A. Zisserman.
Very Deep Convolutional Networks for Large-Scale Image Recognition. Technical report, 2014.

→→ 16-layer model, 19-layer model. 7.3% top-5 test error on ILSVRC-2012

Discussion

- Still have many orders of magnitude to go in order to match the infero-temporal(IT) pathway of the human visual system.



Convolutional Neural Networks? vs. Convolutonal Networks?

Figure adapted from Gross, C. G., Rodman, H. R., Gochin, P. M., and Colombo, M. W. (1993). Inferior temporal cortex as a pattern recognition device. In "Computational Learning and Cognition" (E. Baum, ed.), pp. 44–73. Society for Industrial and Applied Mathematics, Philadelphia.

Discussion

••Classification on video.

video sequences provide temporal structure missing in static images.

K. Simonyan, A. Zisserman.
Two-Stream Convolutional Networks for Action Recognition in Videos. NIPS 2014.

→→ separating two pathways for spatial and temporal networks analogous to the ventral and dorsal pathways.