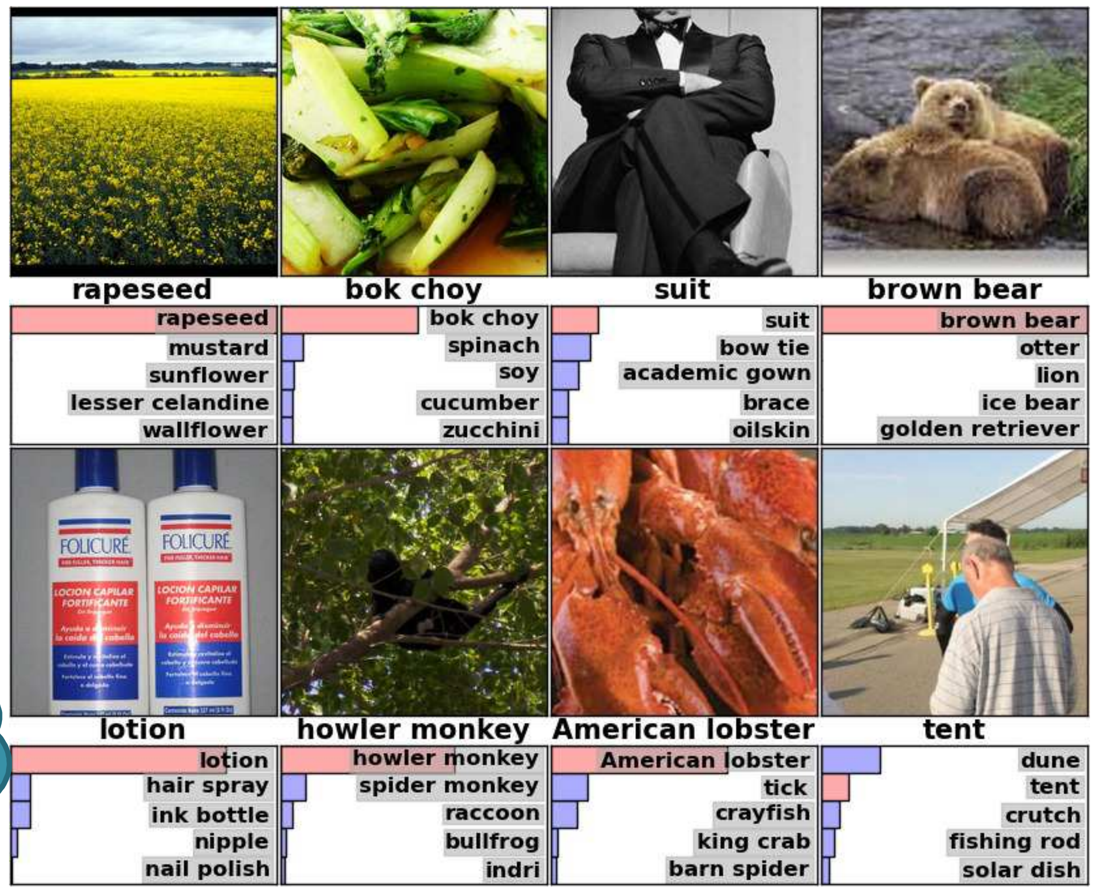
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

Motivation

Classification goals:

- •Make 1 guess about the label (Top-1 error)
- •Make 5 guesses about the label (Top-5 error)



No Bounding Box

Database

ImageNet

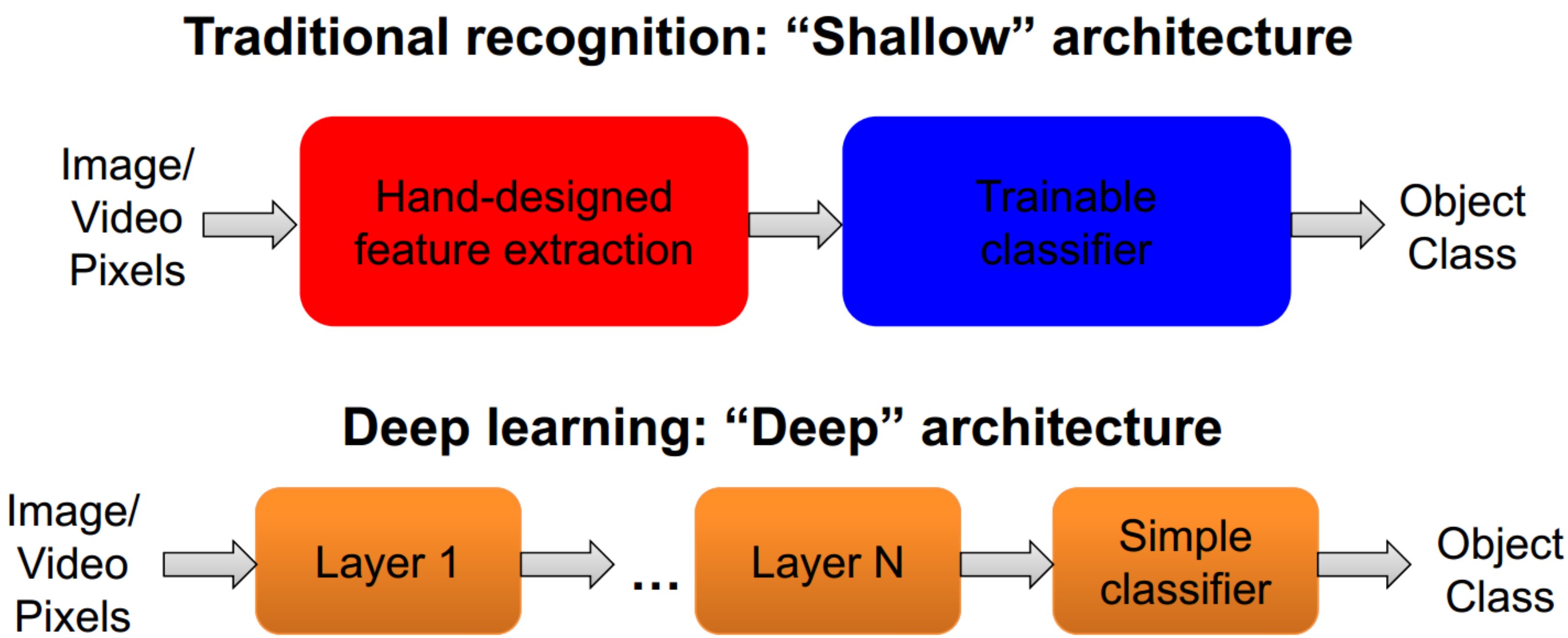
- 15M images
- 22K categories
- Images collected from Web
- RGB Images
- Variable-resolution
- Human labelers (Amazon's Mechanical Turk crowd-sourcing)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)

- 1K categories
- 1.2M training images (~1000 per category)
- 50,000 validation images
- 150,000 testing images

Strategy – Deep Learning

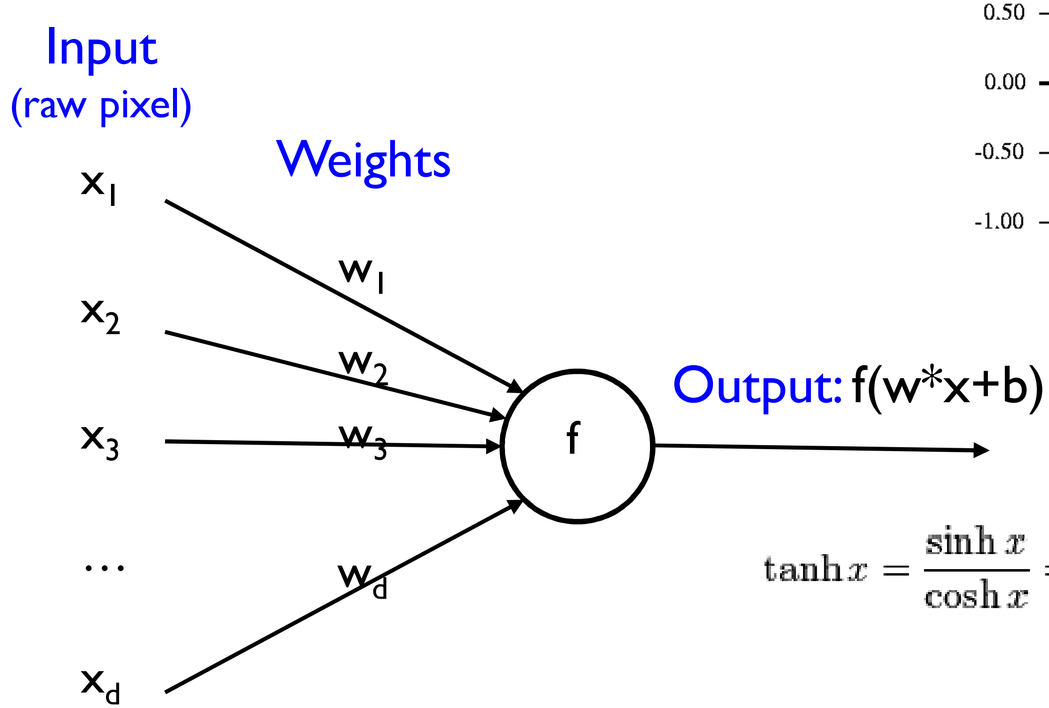
“Shallow” vs.“deep” architectures

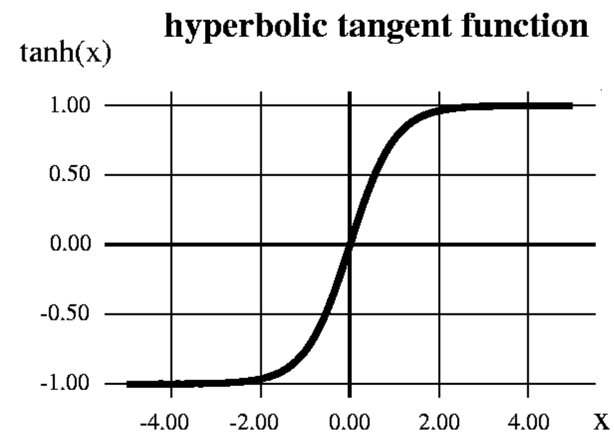


Learn a *feature hierarchy* all the way from pixels to classifier

reference : http://web.engr.illinois.edu/~slazebni/spring14/lec24_cnn.pdf

Neuron - Perceptron



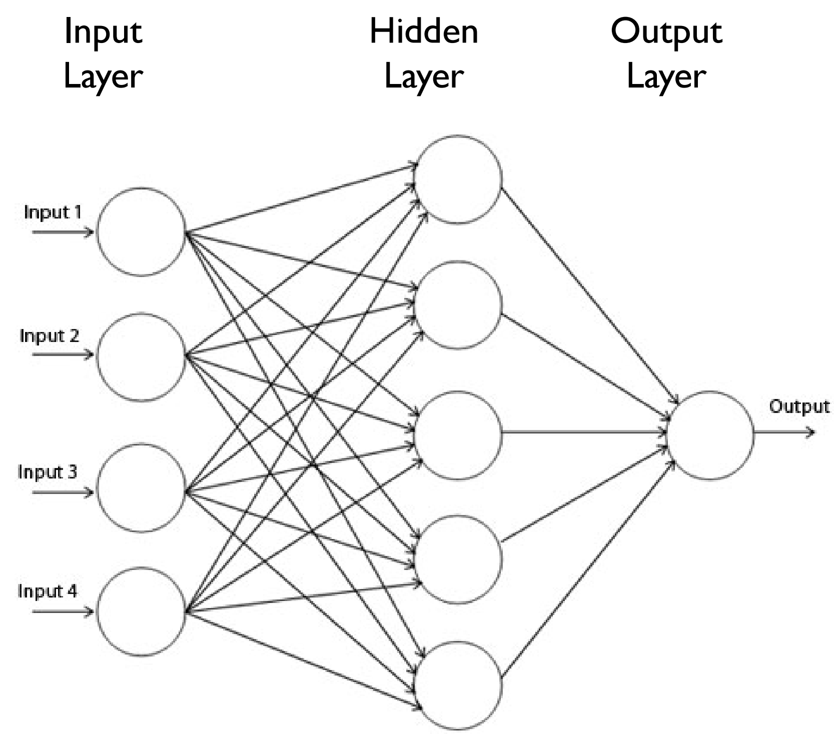


$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

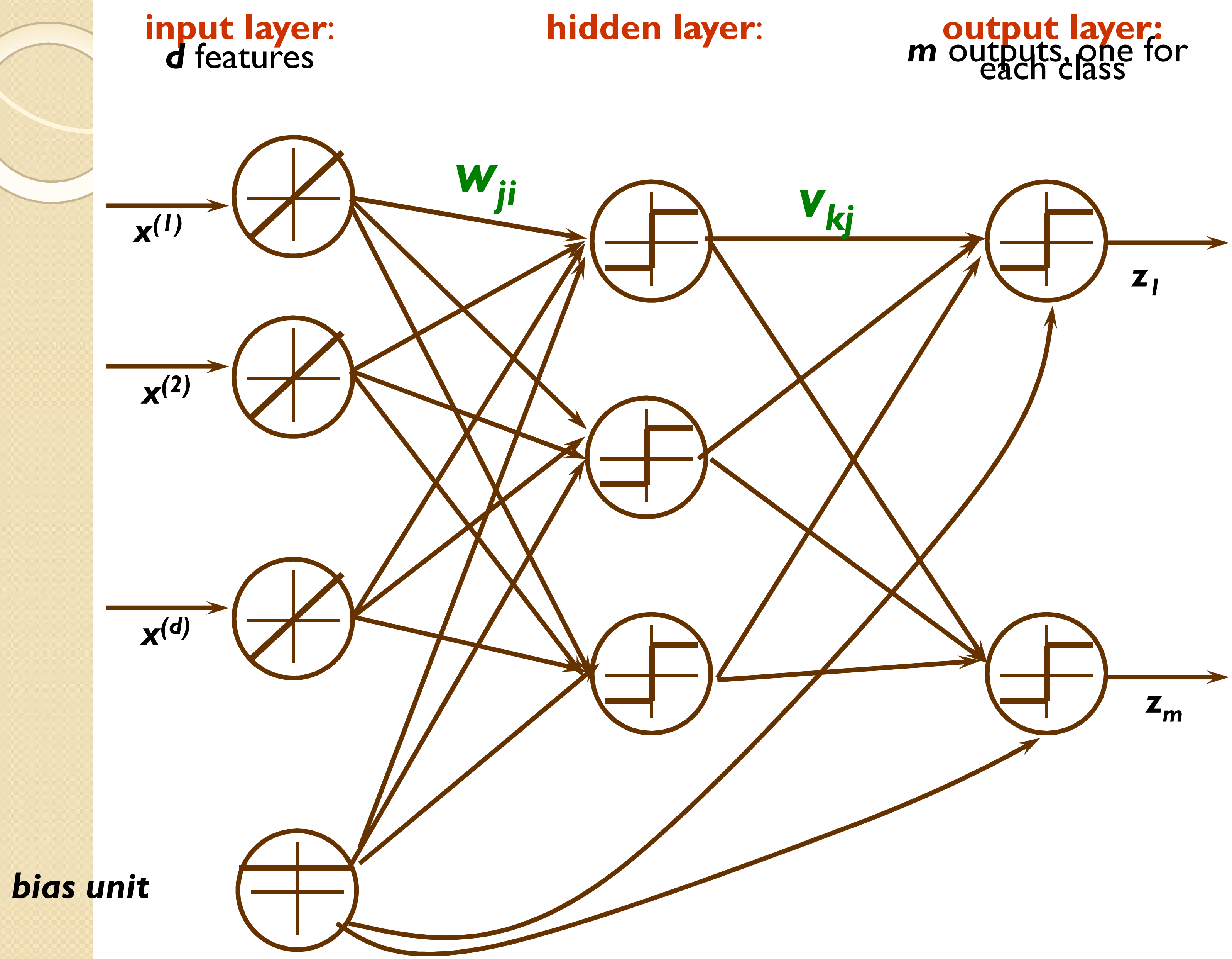
reference : http://en.wikipedia.org/wiki/Sigmoid_function#mediaviewer/File:Gjl-t(x).svg

Multi-Layer Neural Networks

- Nonlinear classifier
- Learning can be done by *gradient descent* → Back-Propagation algorithm

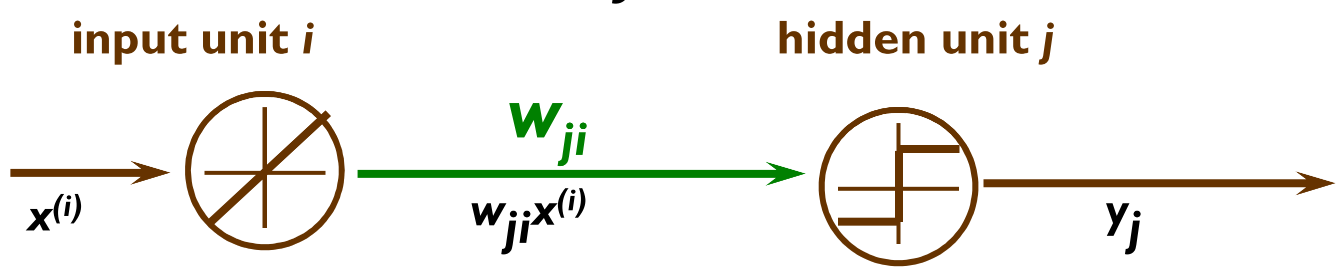


Feed Forward Operation

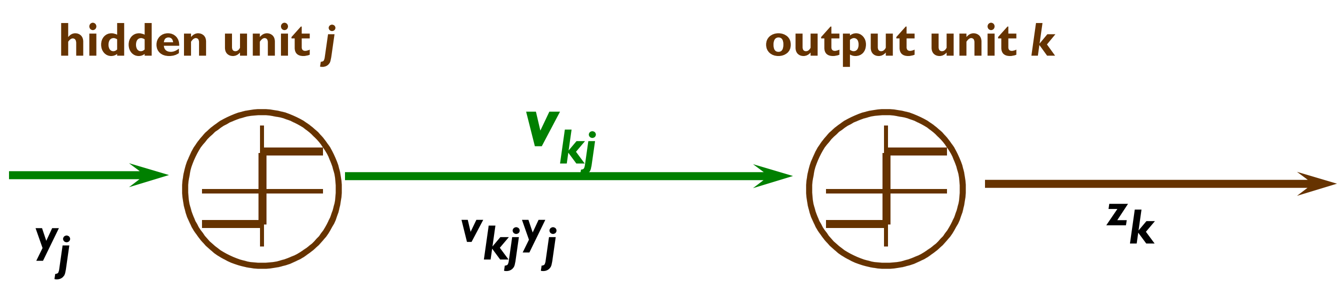


Notation for Weights

- Use $\boldsymbol{w_{ji}}$ to denote the weight between input unit $\boldsymbol{i}$ and hidden unit $\boldsymbol{j}$



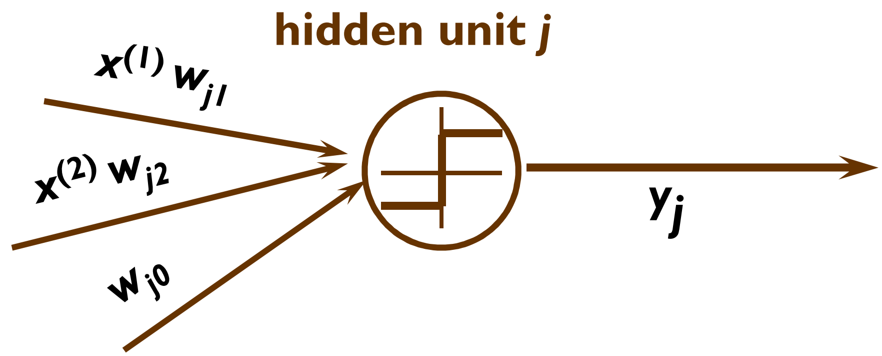
- Use $\boldsymbol{v_{kj}}$ to denote the weight between hidden unit $\boldsymbol{j}$ and output unit $\boldsymbol{k}$



Notation for Activation

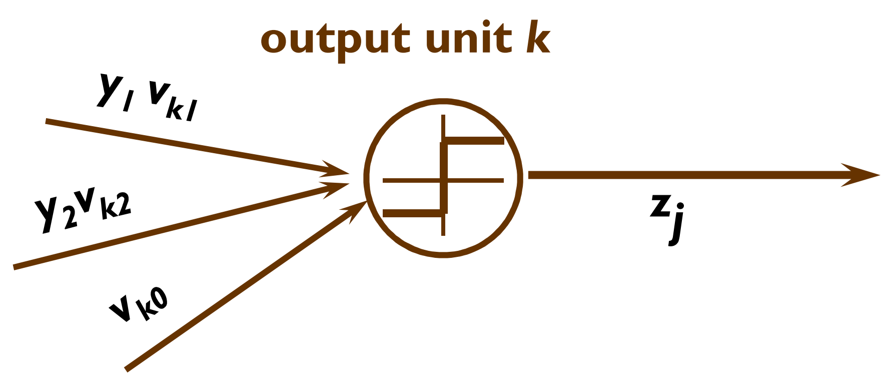
- Use $\boldsymbol{net_j}$ to denote the activation and hidden unit $\boldsymbol{j}$

$$net_j = \sum_{i=1}^{d} x^{(i)} w_{ji} + w_{j0}$$



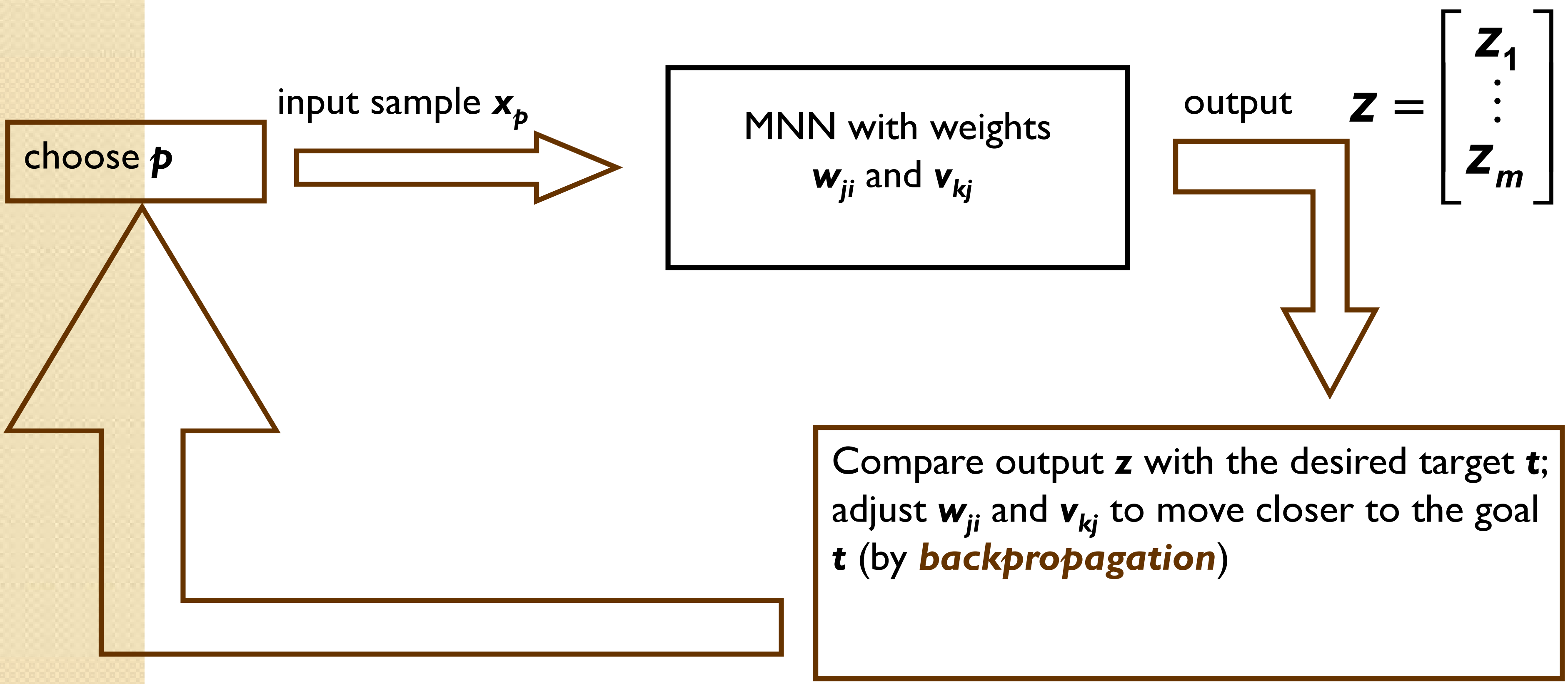
- Use $\boldsymbol{net^*_k}$ to denote the activation at output unit $\boldsymbol{k}$

$$net^*_k = \sum_{j=1}^{N_H} y_j v_{kj} + v_{k0}$$



Network Training

1. Initialize weights $\boldsymbol{w}_{ji}$ and $\boldsymbol{v}_{kj}$ randomly **but not to 0**
2. Iterate until a stopping criterion is reached



BackPropagation

- Learn w_{ji} and v_{kj} by minimizing the training error
- What is the training error?
- Suppose the output of MNN for sample $\boldsymbol{x}$ is $\boldsymbol{z}$ and the target (desired output for $\boldsymbol{x}$) is $\boldsymbol{t}$

- Error on one sample: $$J(w,v) = \frac{1}{2}\sum_{c=1}^{m}(t_c - z_c)^2$$
- Training error: $$J(w,v) = \frac{1}{2}\sum_{i=1}^{n}\sum_{c=1}^{m}\left(t_c^{(i)} - z_c^{(i)}\right)^2$$
- Use gradient descent:

$$v^{(0)}, w^{(0)} = \text{random}$$

repeat until convergence:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w J\left(w^{(t)}\right)$$
$$v^{(t+1)} = v^{(t)} - \eta \nabla_v J\left(v^{(t)}\right)$$

BackPropagation: Layered Model

activation at hidden unit ***j***

$$net_j = \sum_{i=1}^{d} x^{(i)} w_{ji} + w_{j0}$$

$$\Downarrow$$

output at hidden unit ***j***

$$y_j = f\left(net_j\right)$$

$$\Downarrow$$

activation at output unit ***k***

$$net_k^* = \sum_{j=1}^{N_H} y_j v_{kj} + v_{k0}$$

$$\Downarrow$$

activation at output unit ***k***

$$z_k = f\left(net_k^*\right)$$

$$\Downarrow$$

objective function

$$J(w,v) = \frac{1}{2}\sum_{c=1}^{m}\left(t_c - z_c\right)^2$$

chain rule

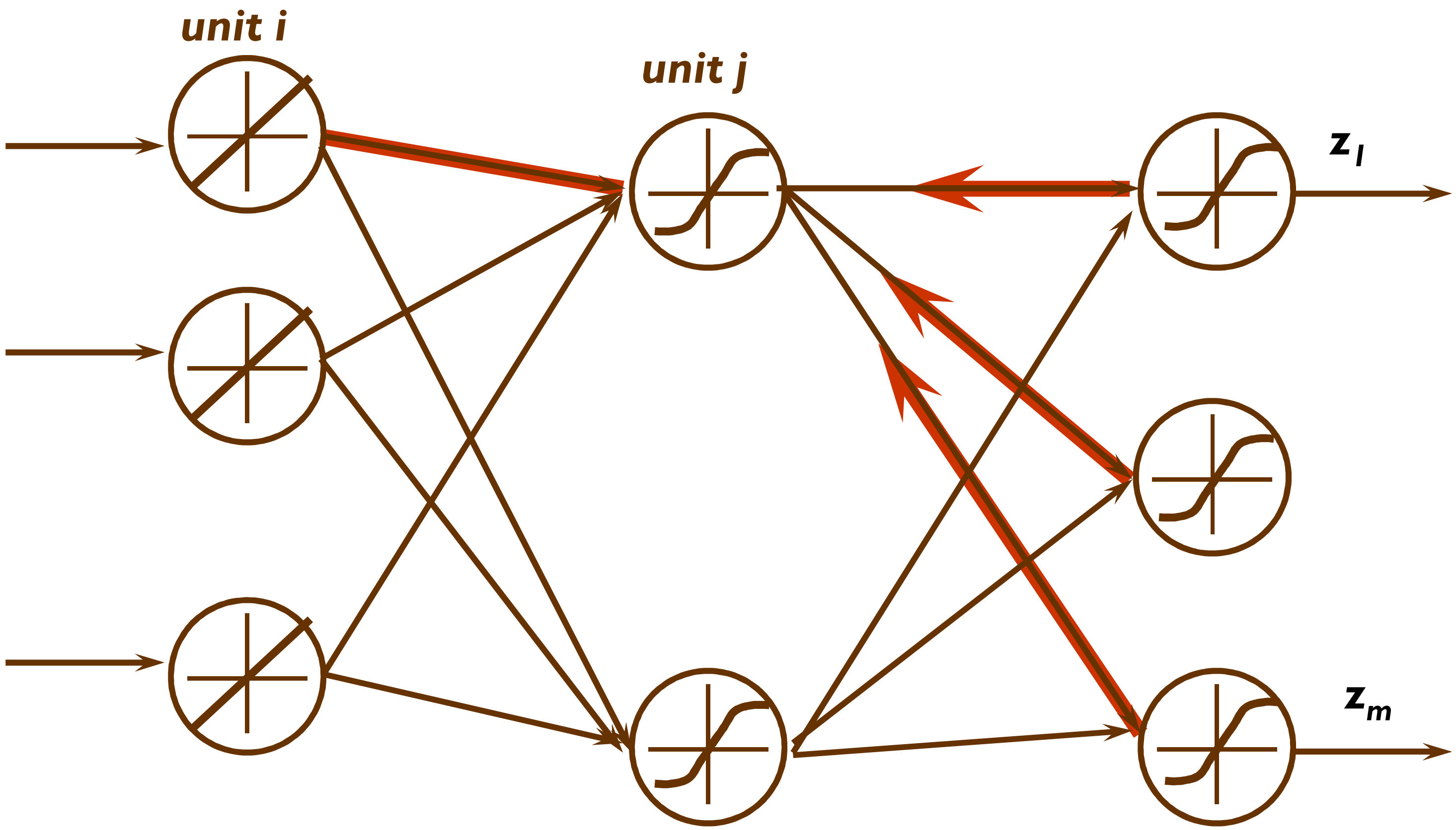
$$\frac{\partial J}{\partial v_{kj}}$$

chain rule

$$\frac{\partial J}{\partial w_{ji}}$$

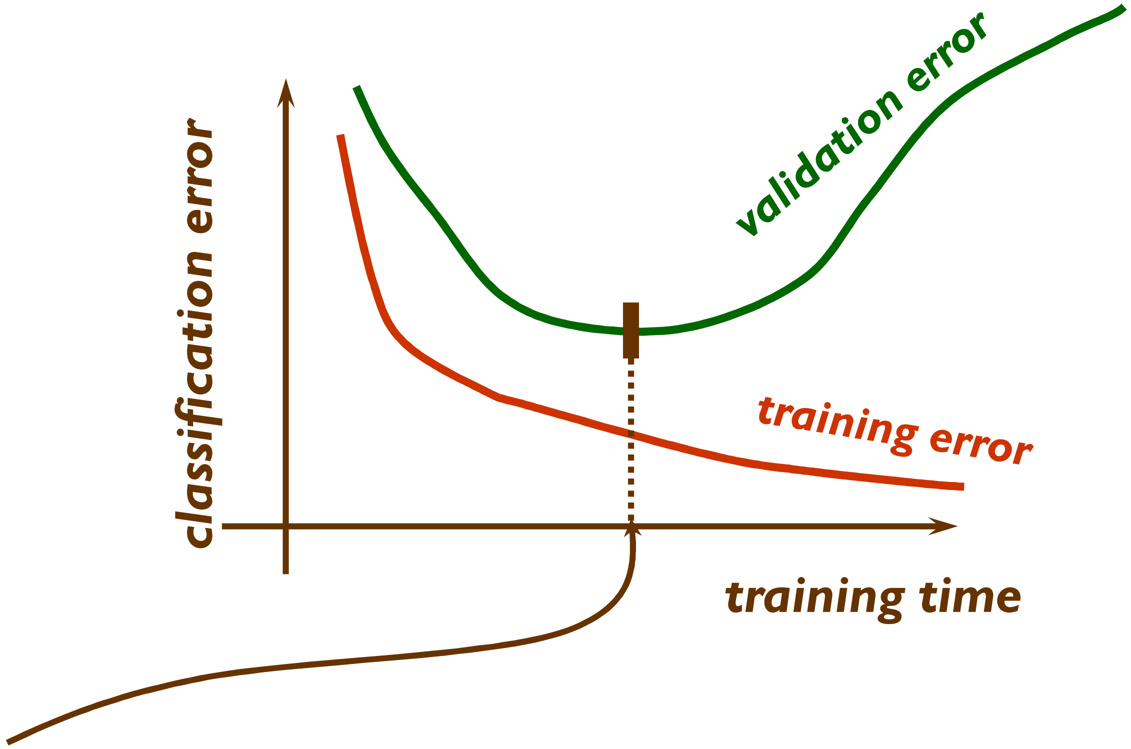
BackPropagation of Errors

$$\frac{\partial J}{\partial w_{ji}} = -f'(net_j)\,x^{(i)}\sum_{k=1}^{m}(t_k - z_k)\,f'(net_k^*)\,v_{kj} \qquad \frac{\partial J}{\partial v_{kj}} = -\underbrace{(t_k - z_k)}_{\text{error}}f'(net_k^*)\,y_j$$



- Name "backpropagation" because during training, errors propagated back from output to hidden layer

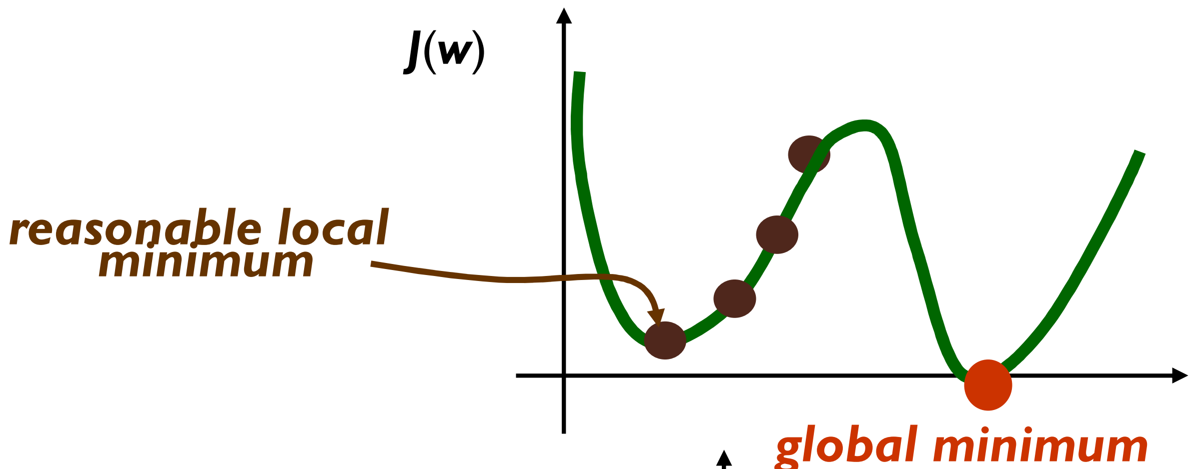
Learning Curves



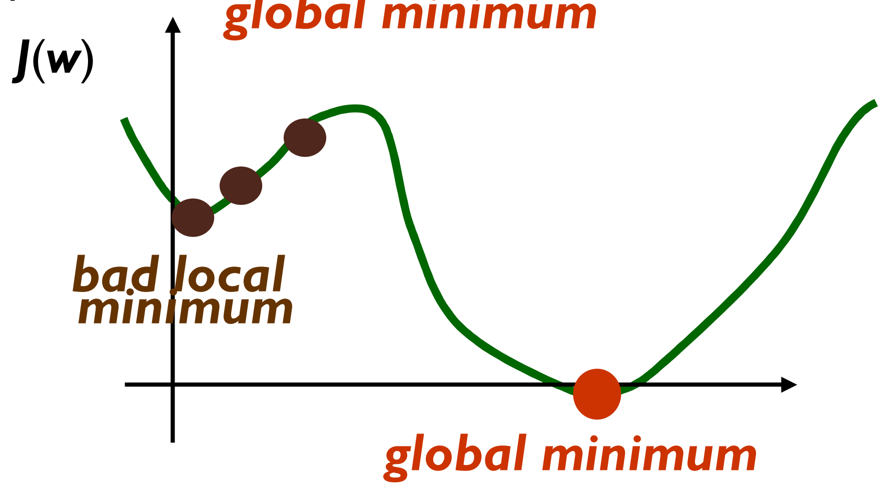
- this is a good time to stop training, since after this time we start to overfit
- Stopping criterion is part of training phase, thus validation data is part of the training data
- To assess how the network will work on the unseen examples, we still need test data

Momentum

- Gradient descent finds only a local minima
  - not a problem if $J(\boldsymbol{w})$ is small at a local minima. Indeed, we do not wish to find $\boldsymbol{w}$ s.t. $J(\boldsymbol{w}) = 0$ due to overfitting



  - problem if $J(\boldsymbol{w})$ is large at a local minimum $\boldsymbol{w}$



Momentum

- Momentum: popular method to avoid local minima and also speeds up descent in plateau regions
  - weight update at time $\boldsymbol{t}$ is $\Delta \boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)}$
  - add temporal average direction in which weights have been moving recently

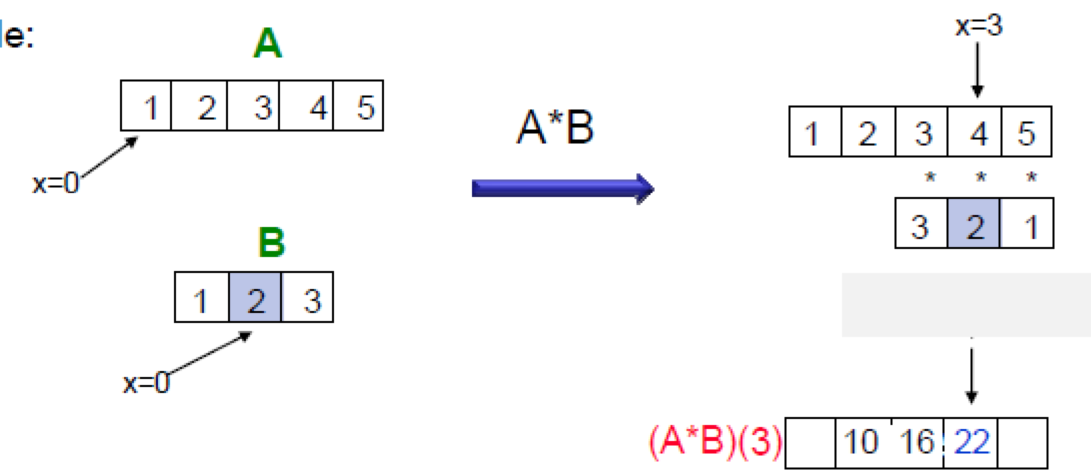
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + (1-\alpha)\underbrace{\left[\eta \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{w}}\right]}_{\text{steepest descent direction}} + \alpha \underbrace{\Delta \boldsymbol{w}^{(t-1)}}_{\text{previous direction}}$$

  - at $\alpha = 0$, equivalent to gradient descent
  - at $\alpha = 1$, gradient descent is ignored, weight update continues in the direction in which it was moving previously (momentum)
  - usually, α is around 0.9

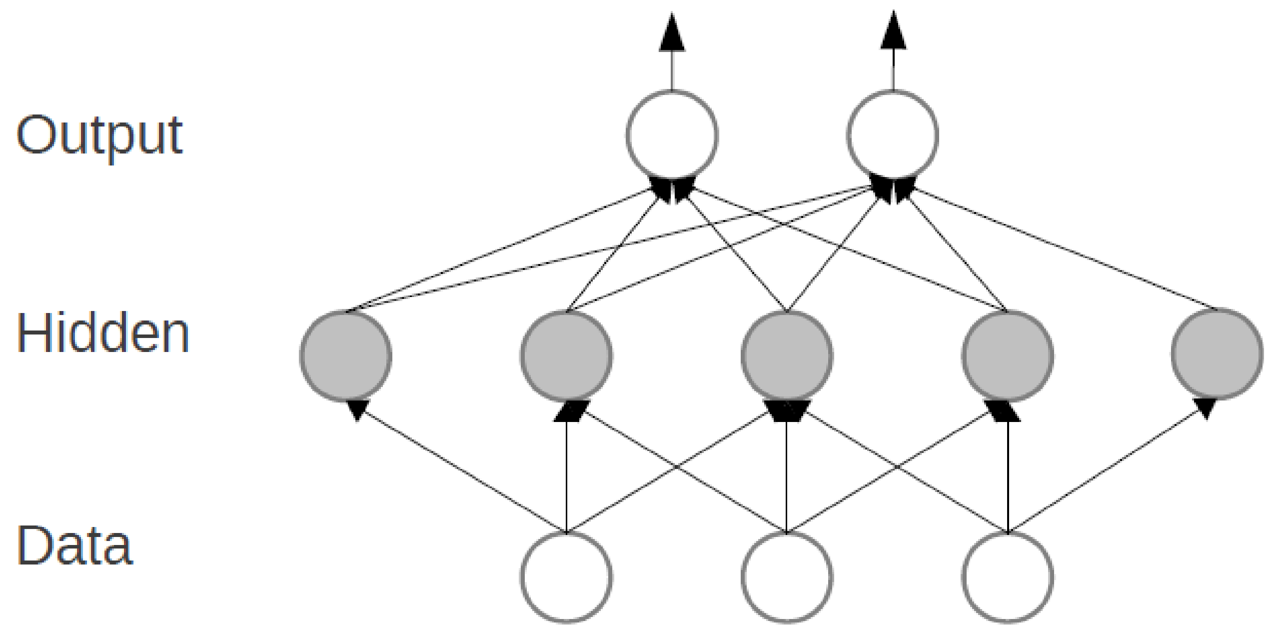
1D Convolution

$$(A * B)(x) = \sum_i A(i)B(x-i)$$

Example:



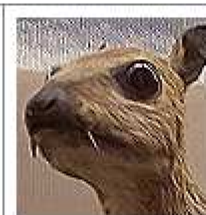
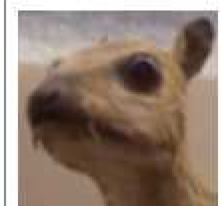
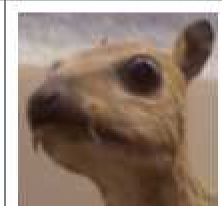
Neural 1D Convolution Implementation



2D Convolution Matrix

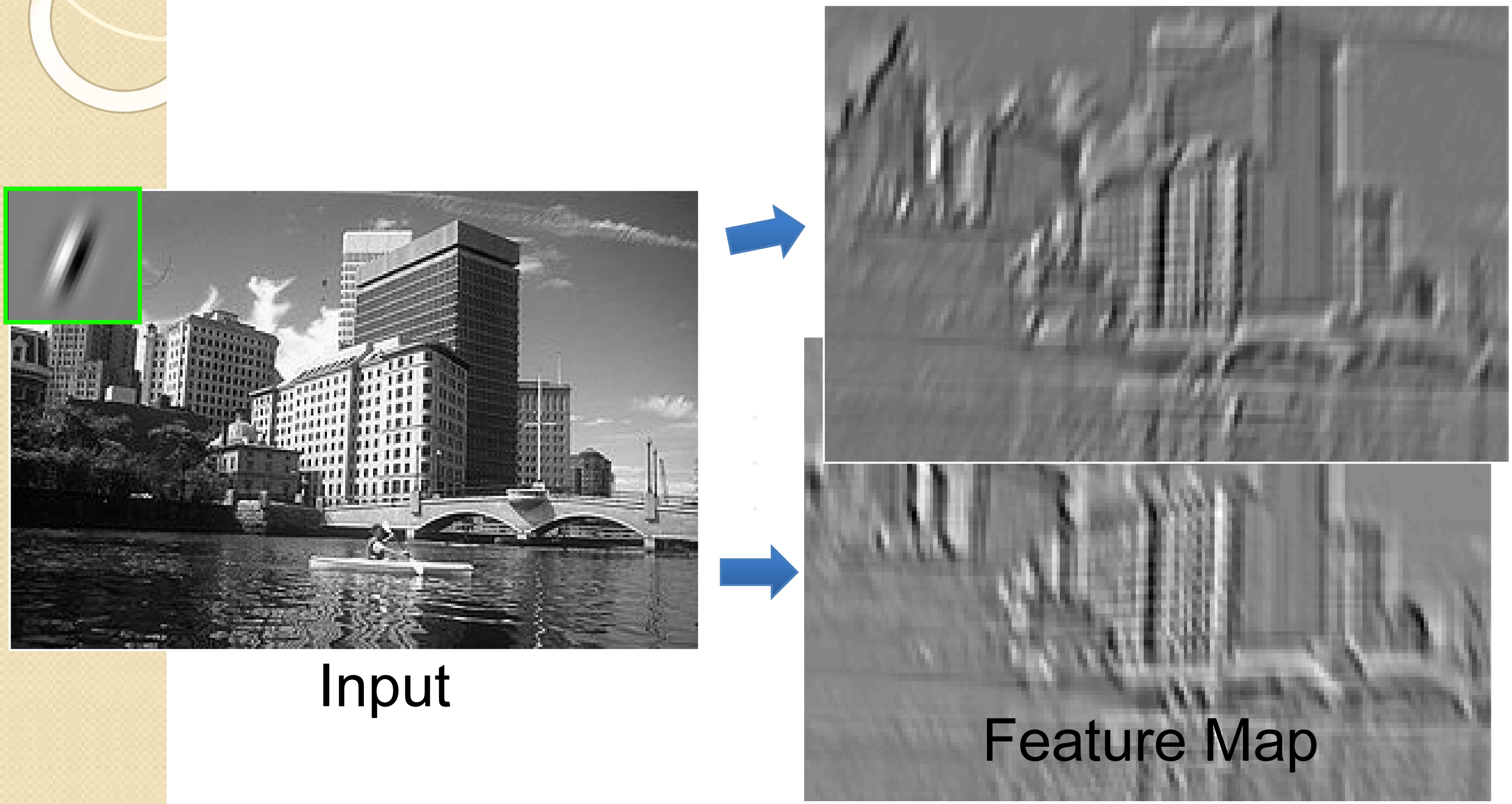
Identity

Edge detection

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

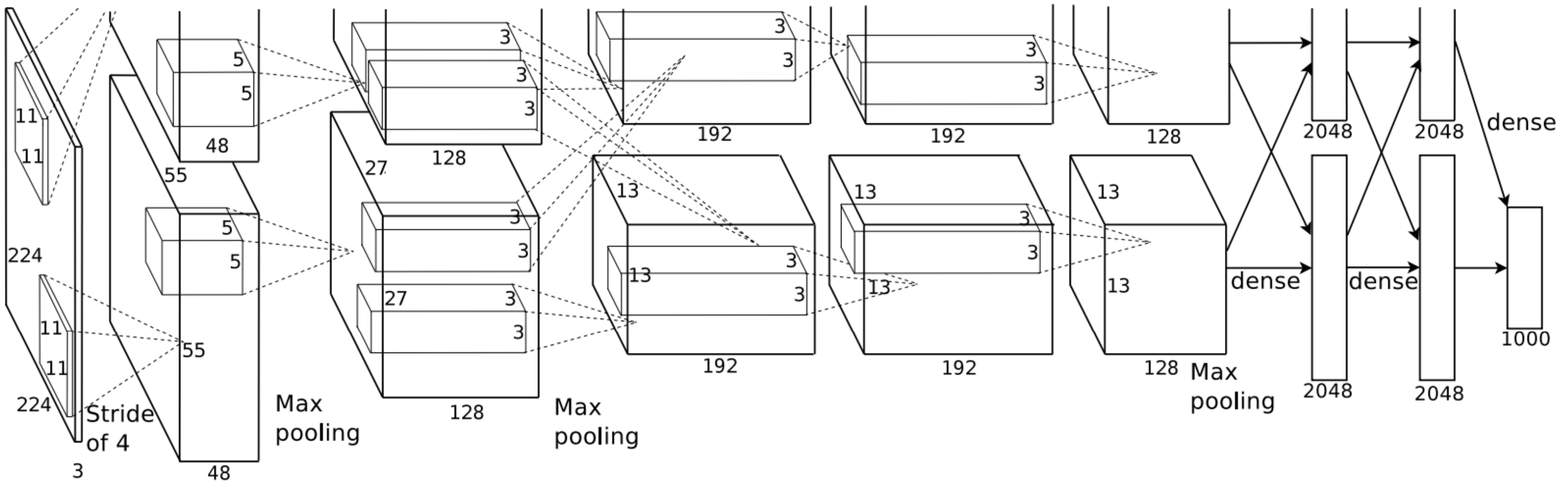
reference : http://en.wikipedia.org/wiki/Kernel_(image_processing)

Convolutional Filter



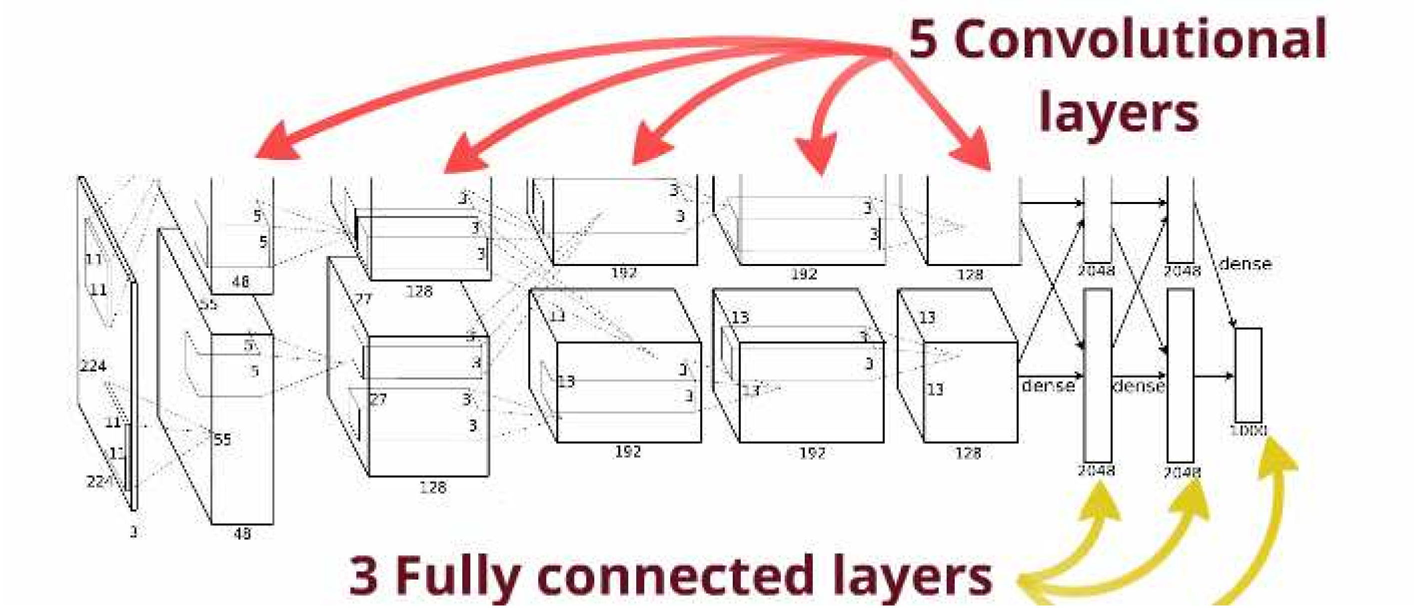
reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

Architecture

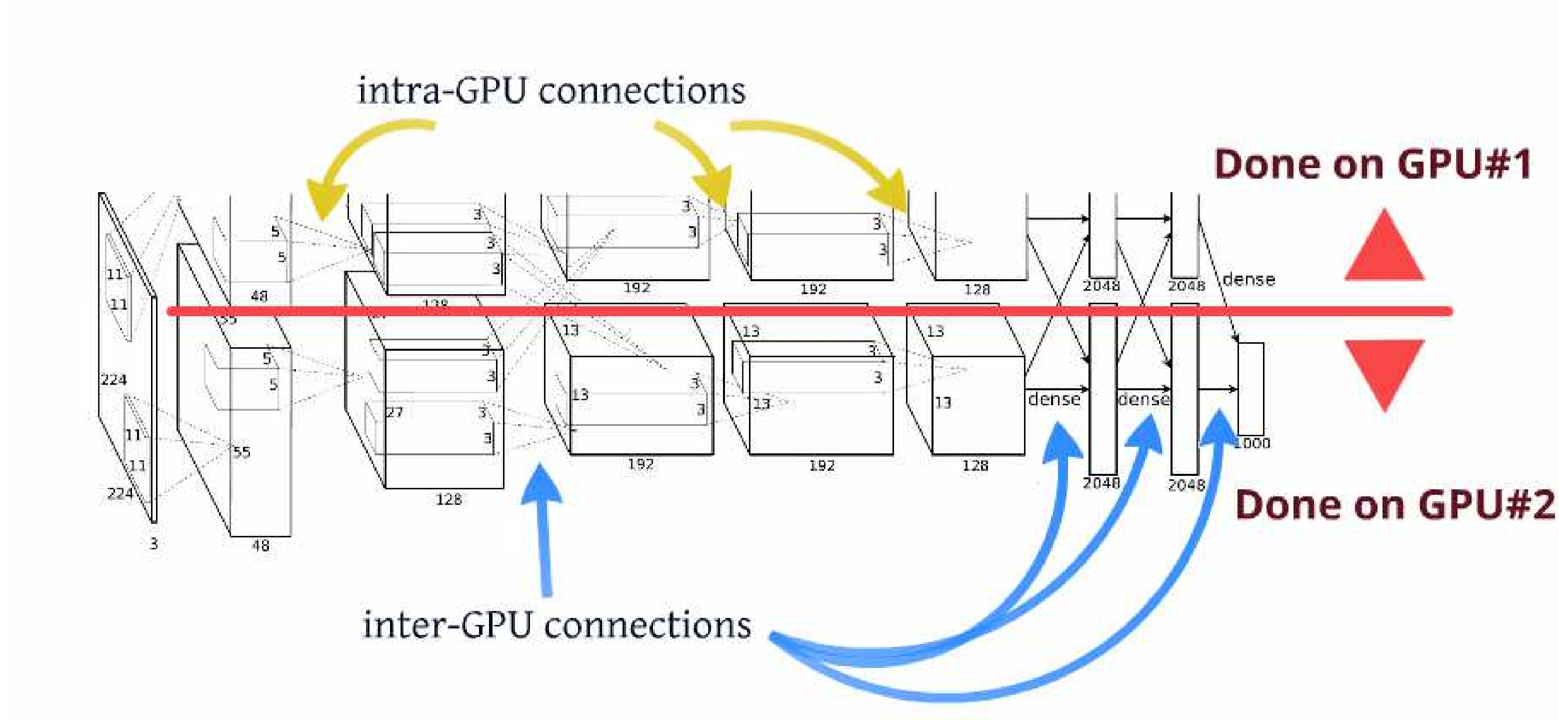


•Trained with stochastic gradient descent on two NVIDIA GPUs for about a week (5~6 days)
•650,000 neurons, 60 million parameters, 630 million connections
•The last layer contains 1,000 neurons which produces a distribution over the 1,000 class labels.

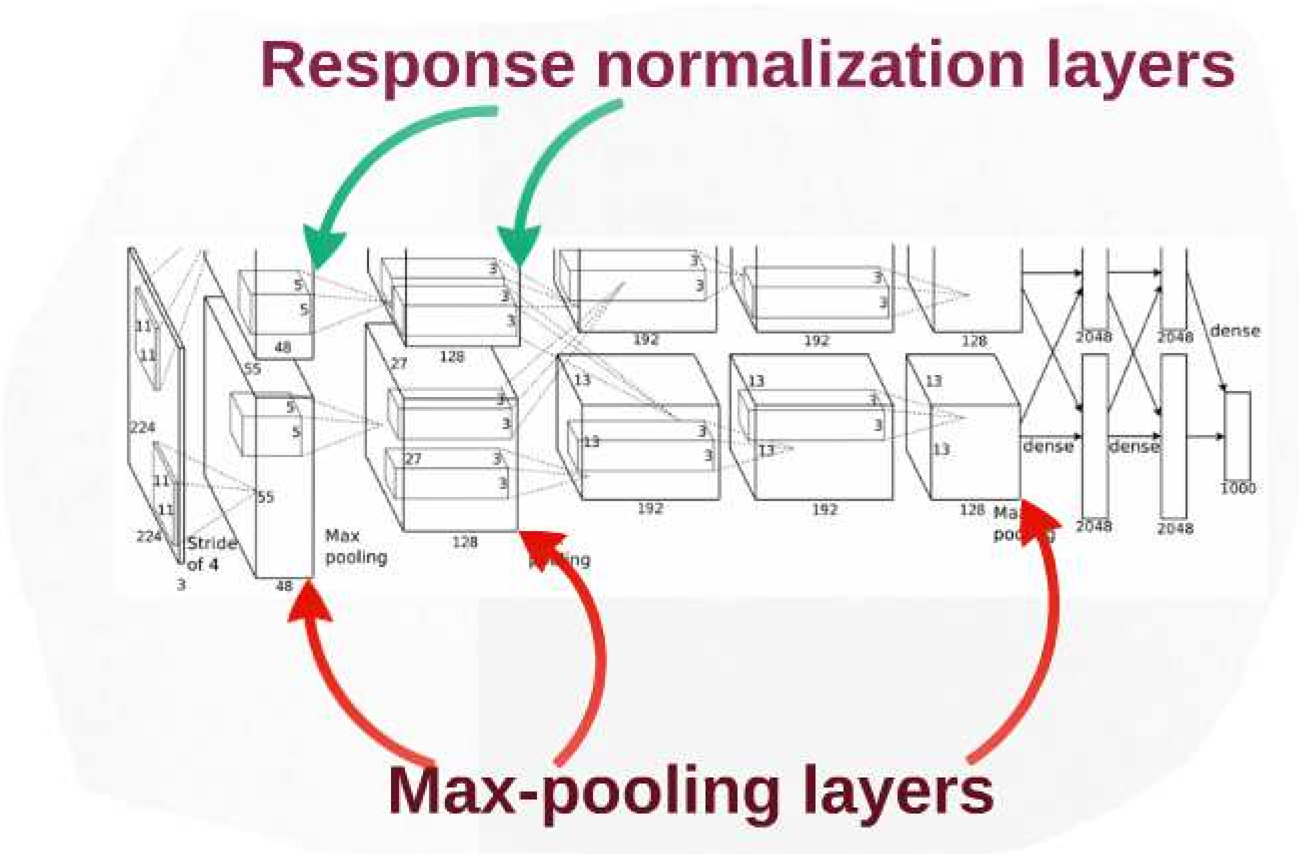
Architecture



Architecture



Architecture



Response-Normalization Layer

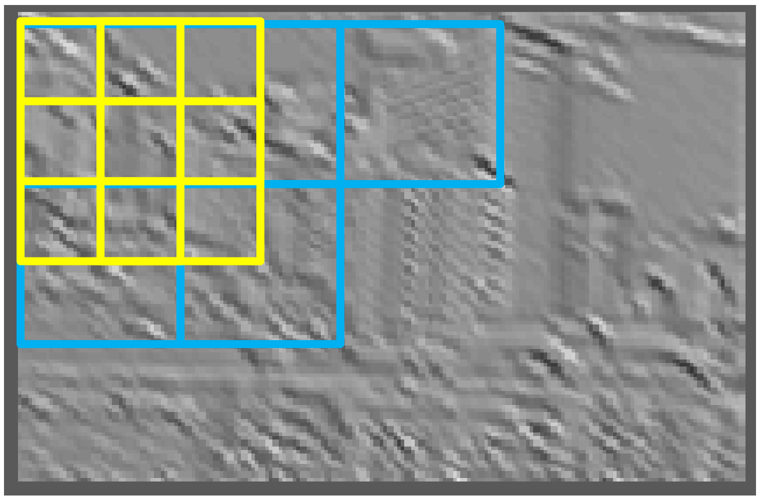
- $a^i_{x,y}$: the activity of a neuron computed by applying kernel i at position (x, y)
- The response-normalized activity is given by

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

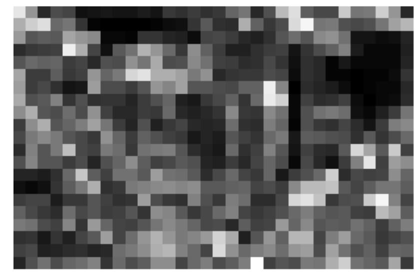
- N : the total # of kernels in the layer
- n : hyper-parameter, n=5
- k : hyper-parameter, k=2
- α : hyper-parameter, α=10^(-4)
- β : hyper-parameter, β =0.75
- This aids generalization even though ReLU don't require it.
- This reduces top-1 error by 1.4 , top-5 error rate by 1.2%

Pooling Layer

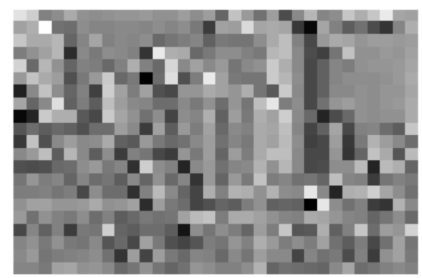
- Non-overlapping / overlapping regions
- Sum or max



Max



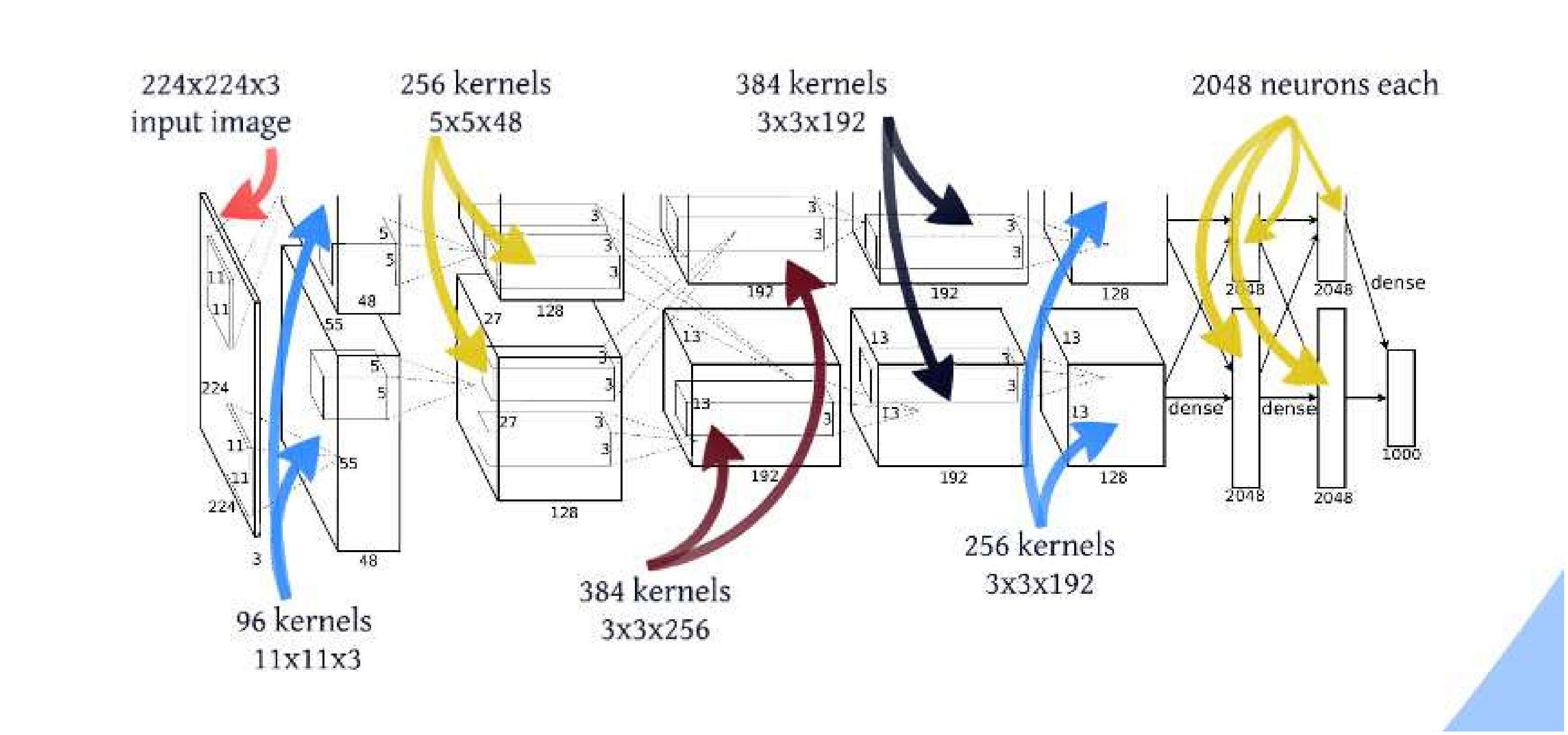
Sum



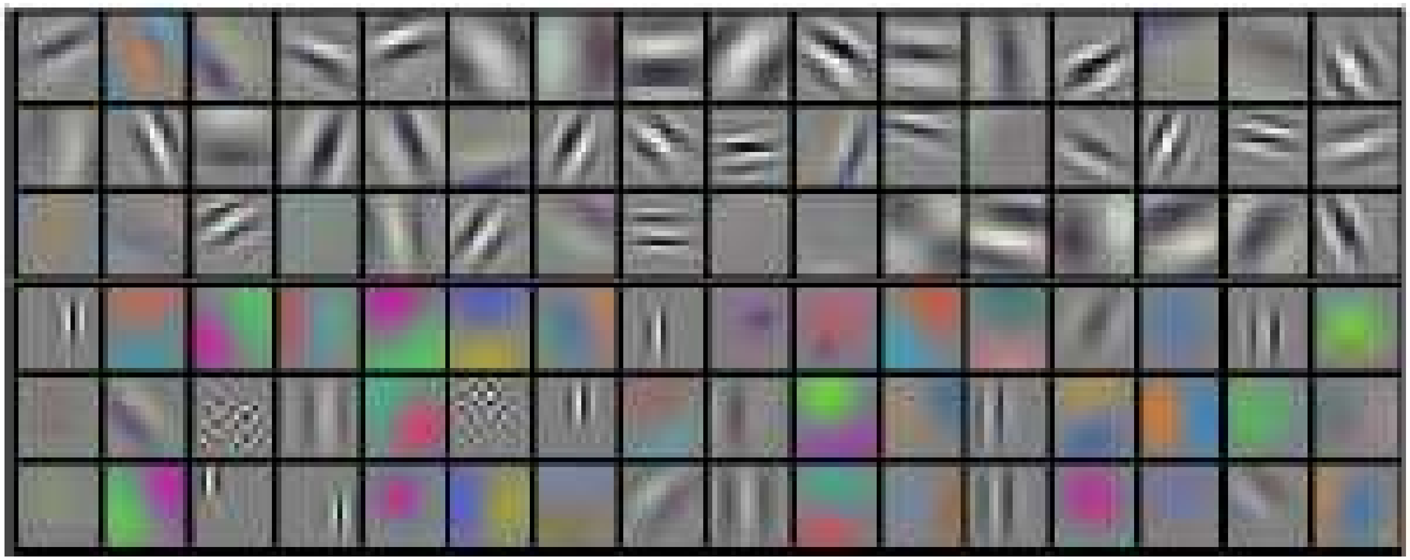
Reduces the error rate of top-1 by 0.4% and top-5 by 0.3%

reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

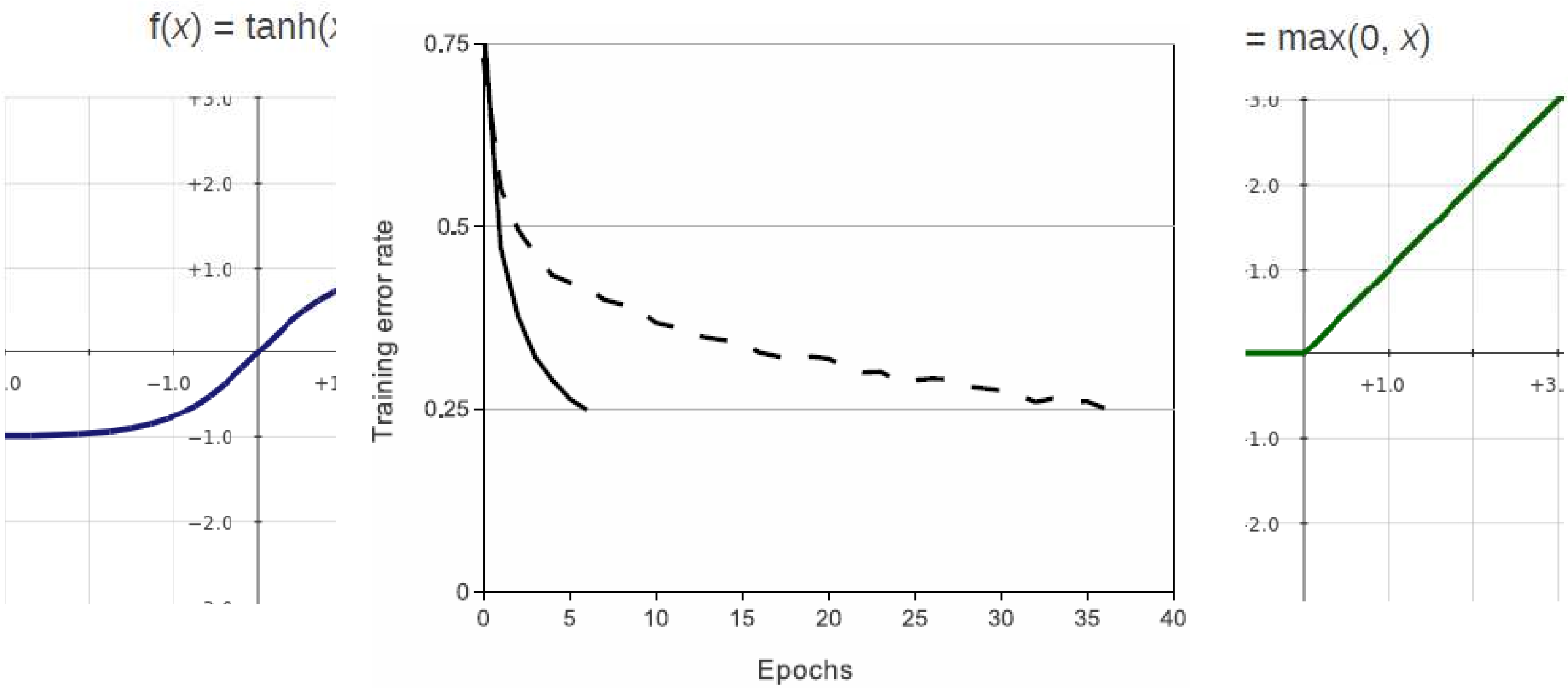
Architecture



First Layer Visualization



ReLU



Very bad (slow to train)

Very good (quick to train)

Learning rule

- Use stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weigh decay of 0.0005
- The update rule for weight w was

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

- i : the iteration index
- ϵ : the learning rate, initialized at 0.01 and reduced three times prior to termination
- $\left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$ the average over the i-th batch D_i of the derivative of the objective with respect to w
- Train for 90 cycles through the training set of 1.2 million images

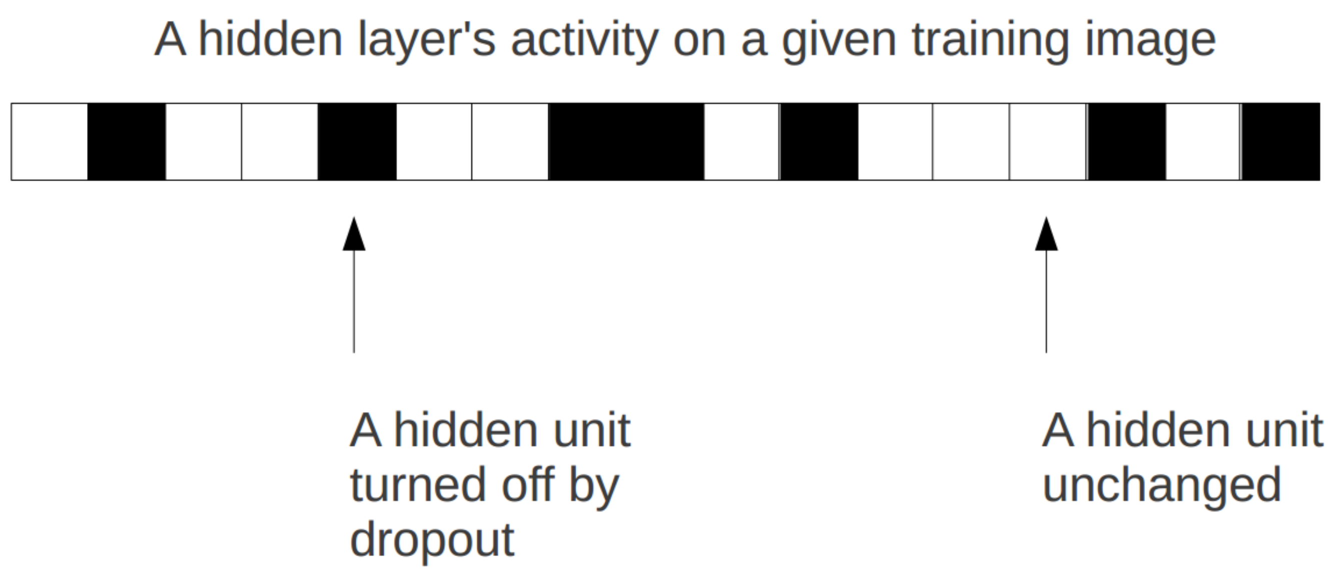
Fighting overfitting - input

- This neural net has 60M real-valued parameters and 650,000 neurons

- It overfils a lot therefore train on five 224x224 patches extracted randomly from 256x256 images, and also their horizontal reflections

Fighting overfitting - Dropout

- Independently set each hidden unit activity to zero with 0.5 probability
- Used in the two globally-connected hidden layers at the net's output
- Doubles the number of iterations required to converge



reference : http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf

Results - Classification

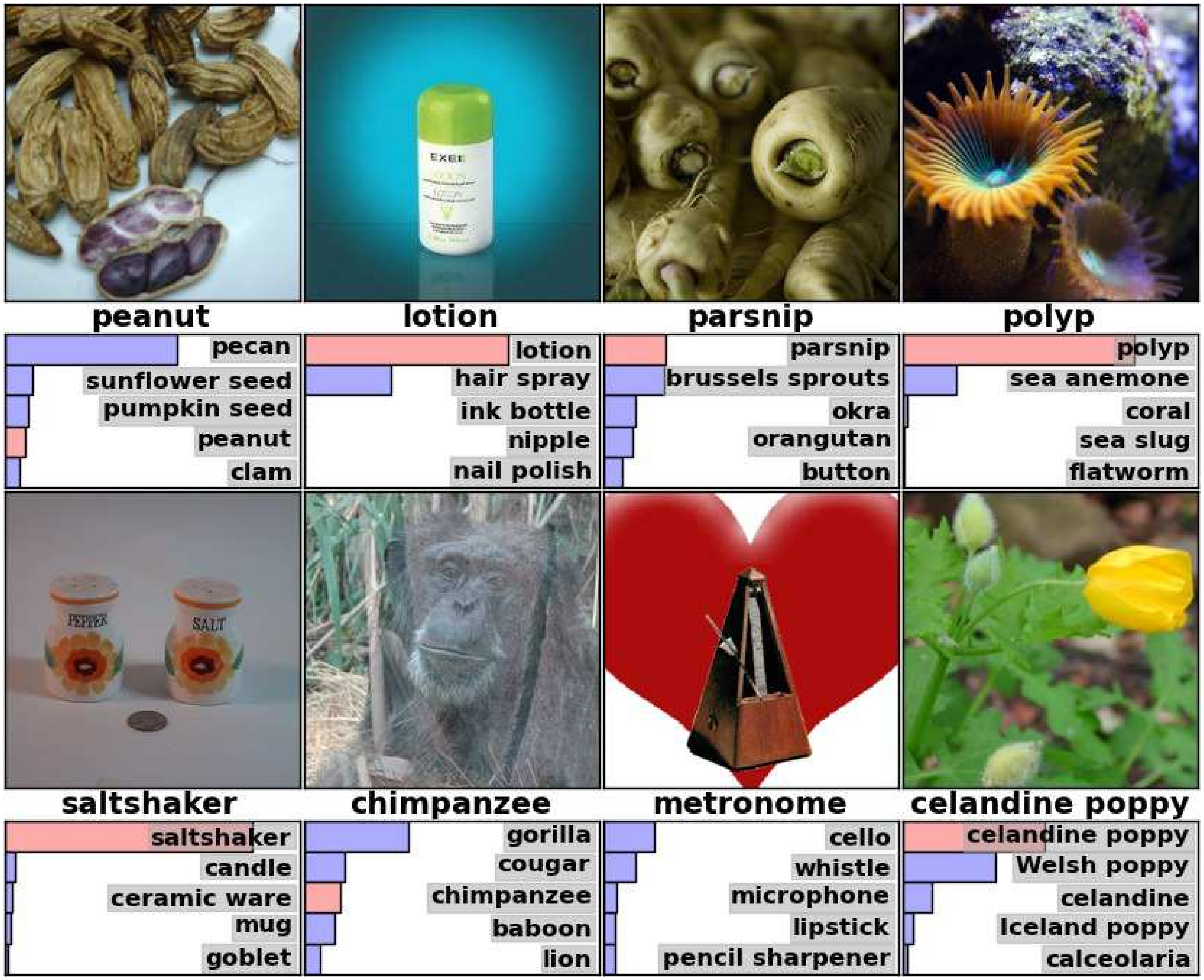
- ILSVRC-2010 test set

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

- ILSVRC-2012 test set

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%**

Results Classification



Results Retrival

The End

Thank you for your attention

Refernces

- *www.cs.toronto.edu/~fritz/absps/**imagenet**.pd*
- https://prezi.com/jiilm_br8uef/imagenet-classification-with-deep-convolutional-neural-networks/
- *sglab.kaist.ac.kr/~sungeui/IR/.../second/20145481오은수.pptx*
- *http://alex.smola.org/teaching/cmu2013-10-701/slides/14_PrincipalComp.pdf*
- *Hagit Hel-or (Convolution Slide)*
- *http://www.cs.haifa.ac.il/~rita/ml_course/lectures/NN.pdf*