MICCAI 15 MUNICH
18th International Conference 2015 · October 5th to 9th · Munich, Germany
on Medical Image Computing and Computer Assisted Interventions

Introduction to Neural Networks and Deep Learning

Heung-Il Suk

hisuk@korea.ac.kr

(Contributors: Dr. S. Kevin Zhou and Dr. Hien Van Nguyen)

MiL
Machine Intelligence Laboratory

Department of Brain and Cognitive Engineering,
Korea University

October 5, 2015

Popularity of Deep Learning

MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2013

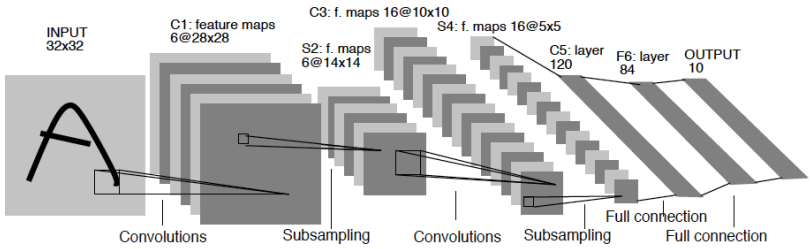
Introduction The 10 Technologies Past Years

Deep Learning

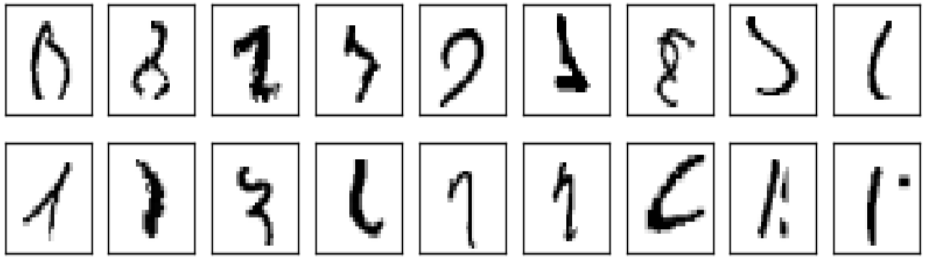
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.

Outperforming or equivalent to many state-of-the-art techniques in various applications

- Digit recognition: MNIST benchmark
  - 21 errors out of 10,000 images [Li *et al.*, 2013]

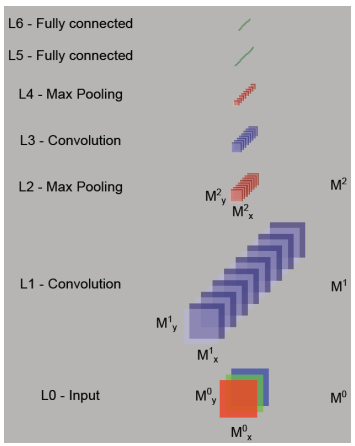


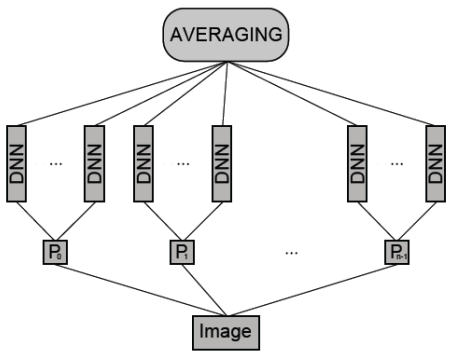
LeNet-5 [LeCun *et al.*, 1998]



Misclassified samples

- Traffic sign recognition
  - First superhuman visual pattern recognition
  - 0.56% error rate [Ciresan *et al.*, 2011]

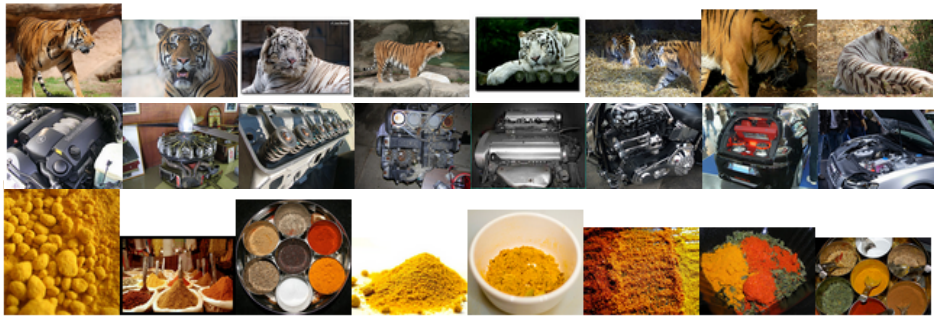




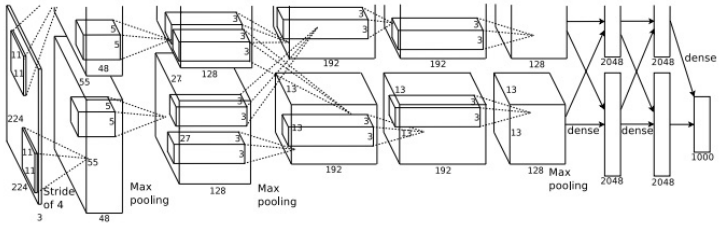
Multi-column DNN

Misclassified samples

- Object recognition
  - Accuracy close to or better than human performance: 5.1%



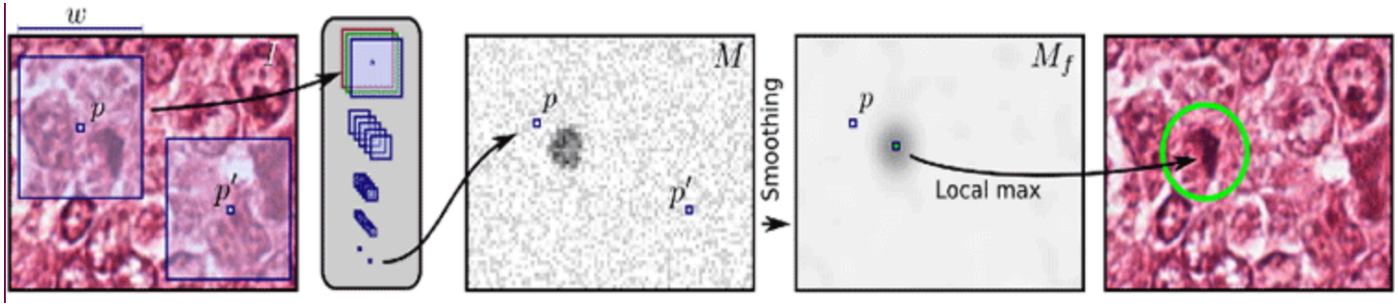
IMAGENET



AlexNet [Krizhevsky *et al.*, 2012]

GoogleNet [Szegedy *et al.*, 2015]

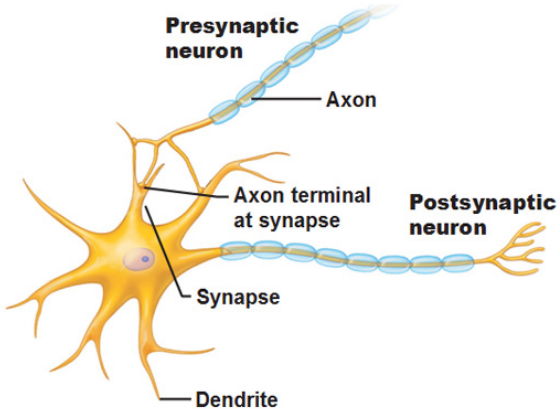
- Mitosis detection in Breast Cancer Histology Images
  - Win the MICCAI Grand Challenge 2013

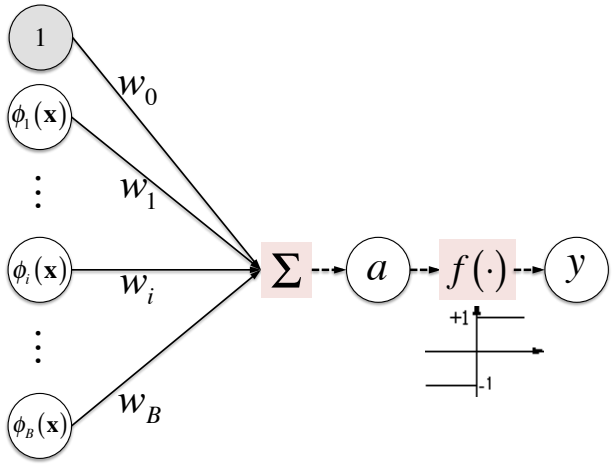


[Ciresan *et al.*, 2013]

(Artificial) Neural Network

Perceptron [Rosenblatt, 1962]





$$y(\mathbf{x}) = f\left(\mathbf{w}^T\mathbf{x}\right)$$

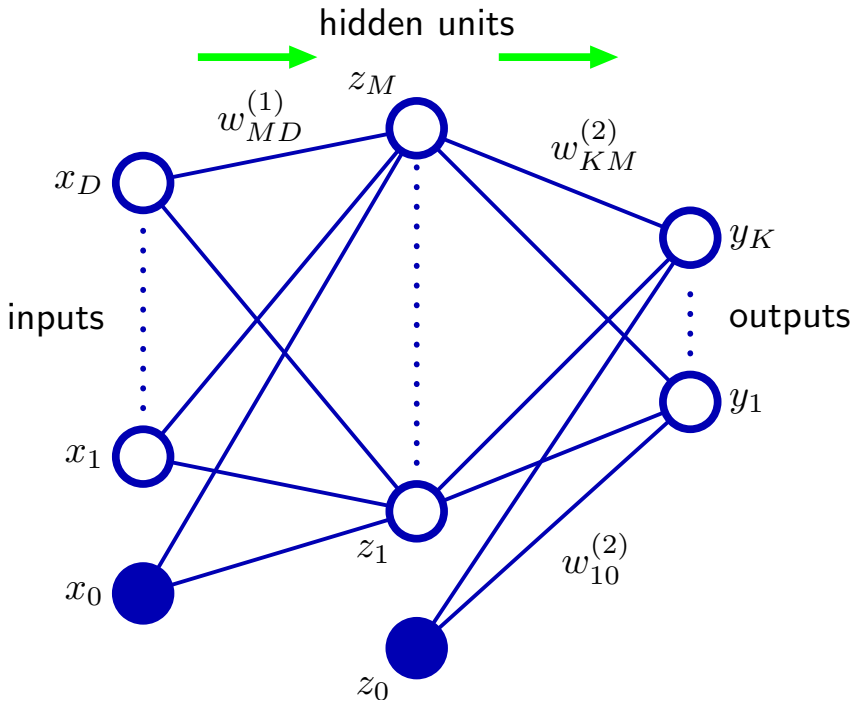
where $f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$

If an exact solution exists, it is guaranteed to find it.

BUT

- Not converge if classes are not linearly separable
- Not readily generalized to $K > 2$ classes

Multi-Layer Perceptron (MLP)

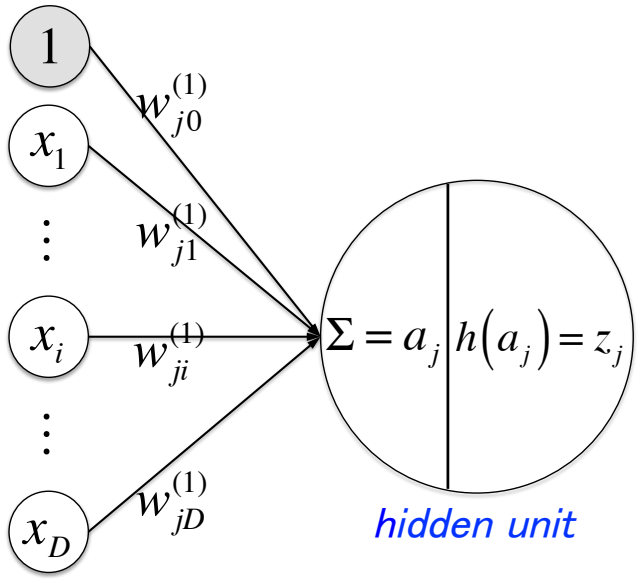


- Multiple layers
- Fully connected
- Feed-forward

Network diagram for the two-layer neural network

- Activations: $\{a_j\}_{j=1}^{M}$

$$a_j = \sum_{i=1}^{D} \underbrace{w_{ji}^{(1)}}_{\text{weights}} x_i + \underbrace{w_{j0}^{(1)}}_{\text{biases}}$$



Connection between input and hidden units

- Outputs of the *hidden units*

$$z_j = h(a_j)$$

  - $h(\cdot)$: differentiable, nonlinear function
  - sigmoidal functions

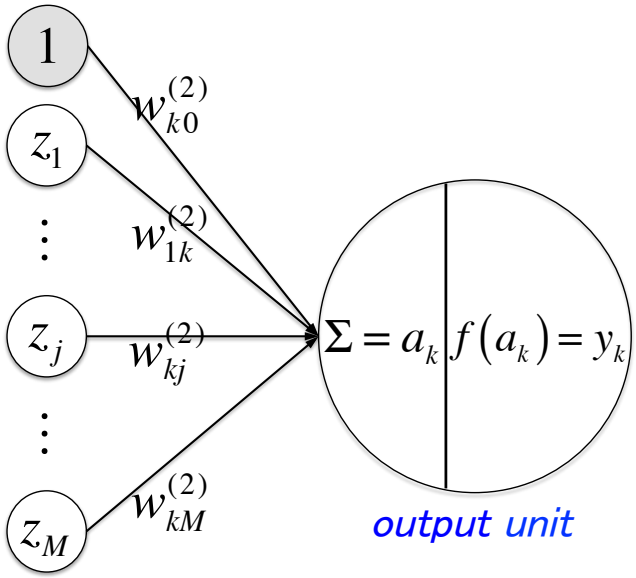
$$\sigma(a) = \frac{1}{1+e^{-a}} \text{ or } \tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$$

- Activations: $\{a_k\}_{k=1}^{K}$

$$a_k = \sum_{j=1}^{M} \underbrace{w_{kj}^{(2)}}_{\text{weights}} z_j + \underbrace{w_{k0}^{(2)}}_{\text{biases}}$$

- Outputs of the '*output units*'

$$y_k = f(a_k)$$



Connection between hidden and output units

- Choice of the output activation function $f(\cdot)$ is determined by the nature of the data and the assumed distribution of target variables
  - Regression: identity

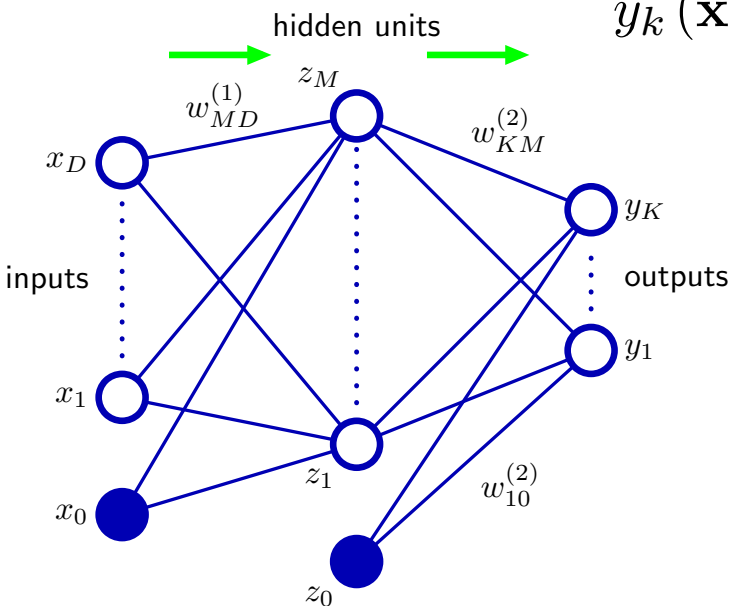
$$y_k = a_k$$

  - Binary classification: logistic sigmoid

$$y_k = \sigma(a_k) = \frac{1}{1+\exp(-a_k)}$$

  - Multiclass classification: softmax

$$y_k = s(a_k) = \frac{\exp(a_k)}{\sum_{l=1}^{K} \exp(a_l)}$$



$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

equivalently

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)}_{\phi_j(\mathbf{x})}\right)$$

$$\mathbf{w} = [w_{j0}, w_{j1}, \ldots, w_{jD}, w_{k0}, w_{k1}, \ldots, w_{kM}]^T$$

Network Training

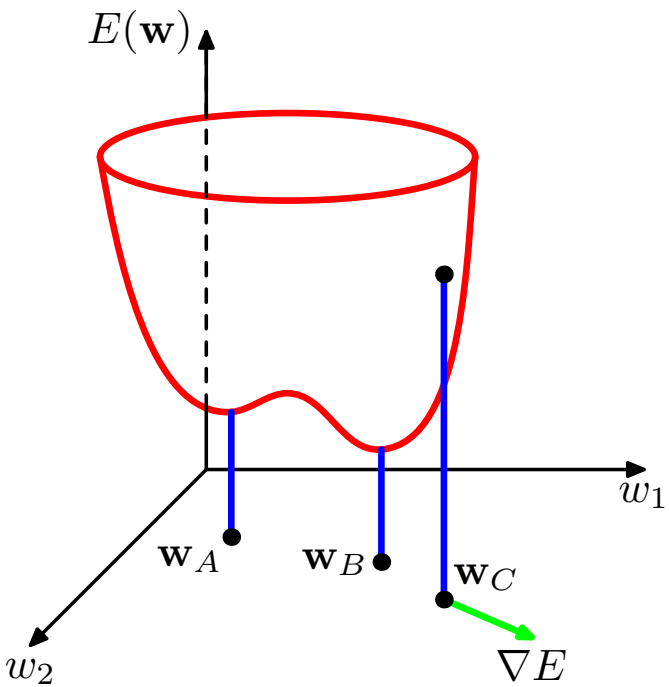
Error Functions

Given a set of input vectors $\{\mathbf{x}_n\}_{n=1}^N$ and target vectors $\{\mathbf{t}_n\}_{n=1}^N$

- Regression
  - Identity activation function
  - Sum-of-squares error: $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^N \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2$
- Binary classification
  - Logistic sigmoid activation function
  - Cross-entropy error function:
    $E(\mathbf{w}) = -\sum_{n=1}^N \{t_n \ln y_n + (1-t_n)\ln(1-t_n)\}$
- Multiclass classification
  - Softmax function
  - Cross-entropy error function: $E(\mathbf{w}) = -\sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_{kn}$

Parameter Optimization

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, E(\mathbf{w})$$



- Geometrical picture of error function
  - a surface sitting over the weight space
- Small step from $\mathbf{w}$ to $\mathbf{w}+\delta\mathbf{w}$ leads to change in error function

$$\delta E \approx \delta\mathbf{w}^T \nabla E(\mathbf{w})$$

  - At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E

$$\nabla E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \cdots, \frac{\partial E}{\partial w_d},\right]^T$$

[Gradient descent optimization]

- Simplest approach to using gradient information
- Take a small step in the direction of the negative gradient

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right)$$

  - η: learning rate

[Different versions of gradient descent optimization]

- Batch: the entire training set to be processes to evaluate ∇E (*gradient descent* or *steepest descent*)
  - More efficient (robust and fast) methods: *conjugate gradients*, *quasi-Newton*
- On-line: one sample at a time (*stochastic gradient descent* or *sequential gradient descent*)
  - Repeated by cycling through the data either in sequence or by selecting at random with replacement
  - Possibility of escaping from local minima
    - Stationary point w.r.t. the error function for the whole data set will generally not be a stationary point for each data point individually
- Mini-batch stochastic gradient descent: a bunch of samples at a time
  - good tradeoff between batch and on-line

Error Backpropagation

To find an efficient technique for evaluating the gradient of an error function $E(\mathbf{w})$ for a feed-forward neural network

1. Evaluation of the derivatives of the error function $\nabla E(\mathbf{w})$ w.r.t. the weights
   - Alternative information passing: forwards and backwards through the network
   - *Local message passing* scheme
2. Computing adjustments to the weights by using the derivatives

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right)$$

Evaluation of Error-Function Derivatives

- Error functions of practical interest comprise a sum of terms

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

- In the following, we consider the problem of evaluating $\nabla E_n(\mathbf{w})$
  - Straightforward for stochastic gradient descent
  - Accumulate over whole training set for batch gradient descent

[*Forward Propagation* in a General Feed-Forward Network]

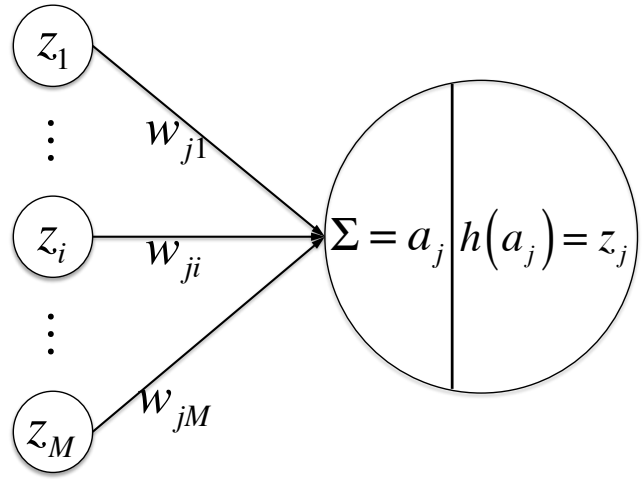
1. Each unit computes a weighted sum of its inputs

$$a_j = \sum_i w_{ji} z_i$$

   - z_i: activation of a unit (or input) that send a connection to unit j
   - w_{ji}: weight associated with that connection

2. Transformation by a nonlinear activation function $h(\cdot)$ to give the activation z_j of unit j

$$z_j = h(a_j)$$



Forward flow of information through the network

[*Backward Propagation* in a General Feed-Forward Network]

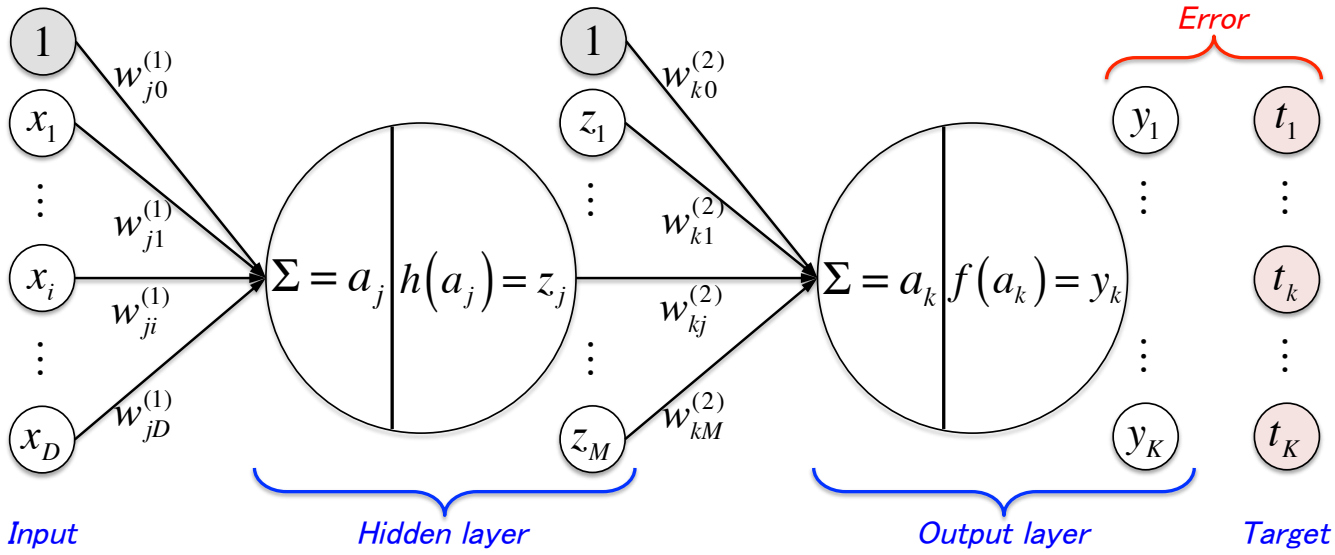
Evaluation of the derivative of E_n w.r.t. w_{ji}

$$\frac{\partial E_n}{\partial w_{ji}} = \underbrace{\frac{\partial E_n}{\partial a_j}}_{\equiv \delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i} \quad \text{(by chain rule)}$$

$$= \delta_j z_i$$

δ_j: gradient of the error function w.r.t. the activation a_j

- Required derivative $\frac{\partial E_n}{\partial w_{ji}}$ is obtained by multiplying the value of δ_j for the unit at the output end of the weight by the value of z_i for the unit at the input end of the weight



- For output unit k,

$$E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2, \quad y_{nk} = f(a_k)$$

$$\frac{\partial E_n}{\partial a_k} \equiv \delta_k = f'(a_k)(y_{nk} - t_{nk}) \qquad f'(a_k) = \begin{cases} 1 & \text{identity} \\ y_{nk}(1 - y_{nk}) & \text{logistic sigmoid} \\ 1 - y_{nk}^2 & \tanh \\ \vdots & \vdots \end{cases}$$

- For hidden unit j,

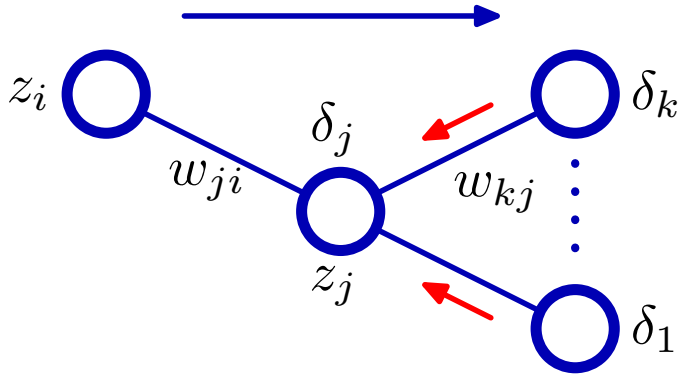
$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

  - We are making use of the fact that variations in a_j give rise to variations in the error function only through variations in the variable a_k
  - By definition

$$\frac{\partial E_n}{\partial a_k} \equiv \delta_k$$

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial \left(\sum_l w_{kl} z_l \right)}{\partial a_j} = \frac{\partial \left(\sum_l w_{kl} h\left(a_l\right) \right)}{\partial a_j} = h'\left(a_j\right) w_{kj}$$

$$\delta_j = h'\left(a_j\right) \sum_k w_{kj} \delta_k$$



Blue arrow: information flow during forward propagation

Red arrow: backward propagation of error information

The value of δ for a particular hidden unit can be obtained by propagating the δ's backwards from units higher up in the network

- Forward propagation

$$
\begin{aligned}
a_j &= \sum_i w_{ji} z_i \\
z_j &= h(a_j)
\end{aligned}
$$

- Backward propagation

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

[Error Backpropagation Algorithm]

1. Apply an input vector $\mathbf{x}_n$ to the network and forward propagate through the network to find the activations of all the hidden and output units

$$a_j = \sum_i w_{ji} z_i \qquad z_j = h\left(a_j\right)$$

2. Evaluate the δ_k for all the output units

$$\delta_k = f'\left(a_k\right)\left(y_k - t_k\right) \qquad f'\left(a_k\right) = \begin{cases} 1 & \text{identity} \\ y_k\left(1 - y_k\right) & \text{logistic sigmoid} \\ 1 - y_k^2 & \tanh \end{cases}$$

3. Backpropagate the δ's to obtain δ_j for each hidden unit in the network

$$\delta_j = h'\left(a_j\right) \sum_k w_{kj} \delta_k$$

4. Evaluate the required derivative

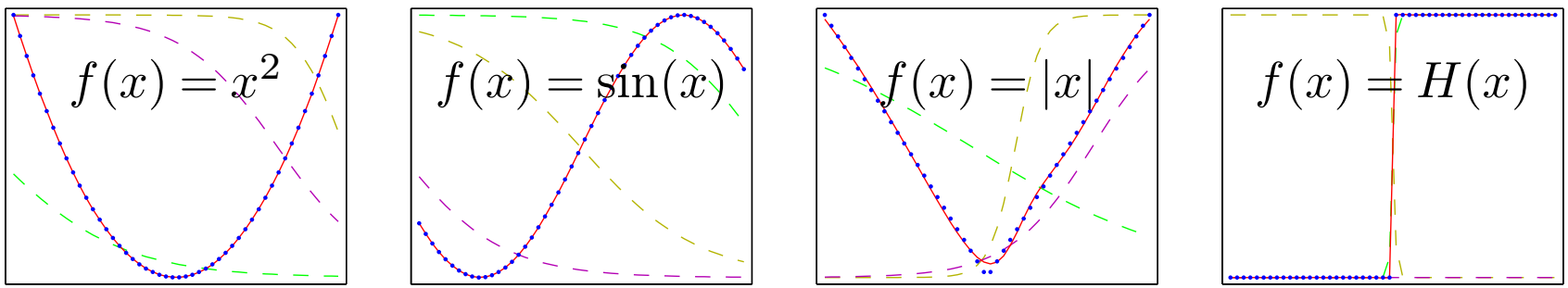
$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

5. Update weight values

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right)$$

MLP: *Universal Approximator*

Universal approximate theorem: A feed-forward network with a single hidden layer containing a finite number of neurons can approximate any continuous function (under mild assumptions on the activation function).



- Blue dots: 50 data points uniformly $(-1, 1)$; 3 hidden units (tanh), linear output units
- Network output (red curve), outputs of thee hidden units

THEN, no need to use a deep architecture?

NO!!!

Deep Learning

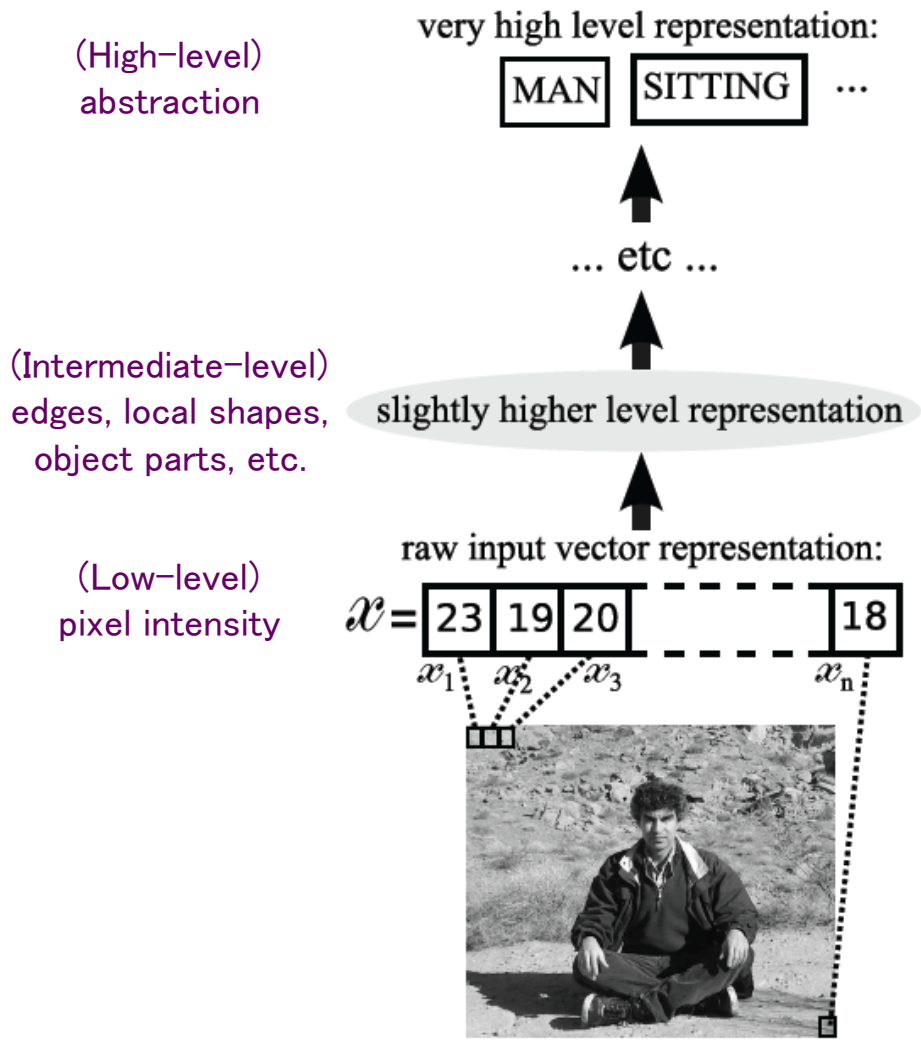
Deep Neural Network

- Possible to approximate complex functions to the same accuracy using a deeper network with much fewer total neurons (model efficiency)
- Smaller number of parameters, requiring a smaller dataset to train

  [Schwarz *et al.*, 1978]
- Hierarchical feature representation (fine-to-abstract)

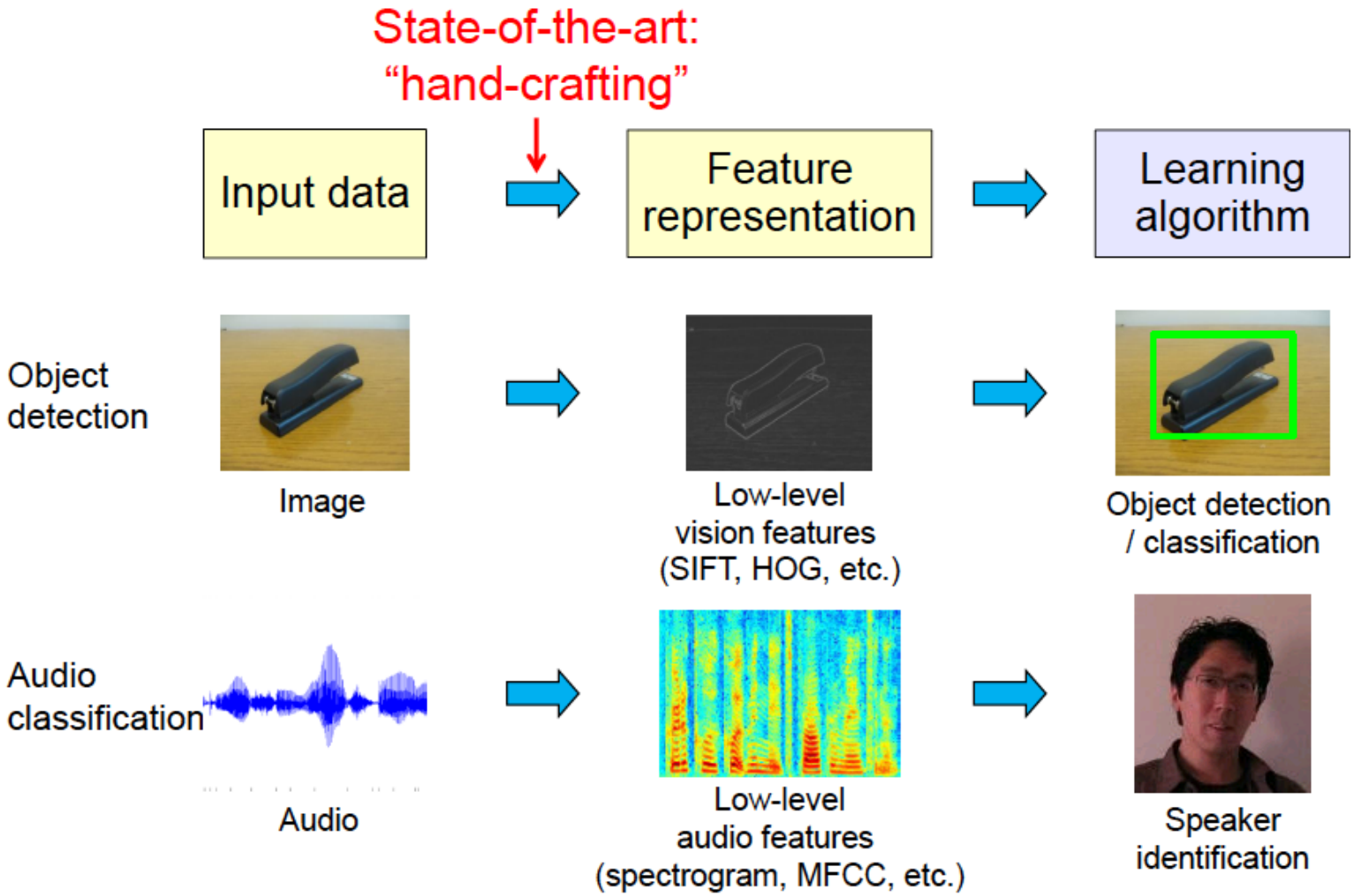
$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)}_{\phi_j(\mathbf{x})}\right) \quad \text{(in 2-layer MLP)}$$

Goal of deep architectures: *to learn feature hierarchies*



(Image credit: Bengio, 2009)

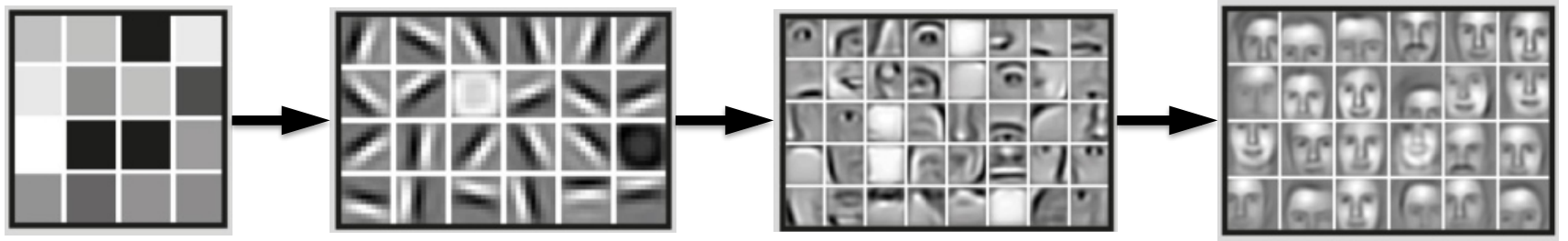
Hand-crafted features in different applications



(Image credit: Prof. H. Lee)

Let the computer learn the feature representations from data autonomously!!!

Deep-learning neural networks use layers of increasingly complex rules to categorize complicated shapes such as faces.



Layer 1: The computer identifies pixels of light and dark.

Layer 2: The computer learns to identify edges and simple shapes.

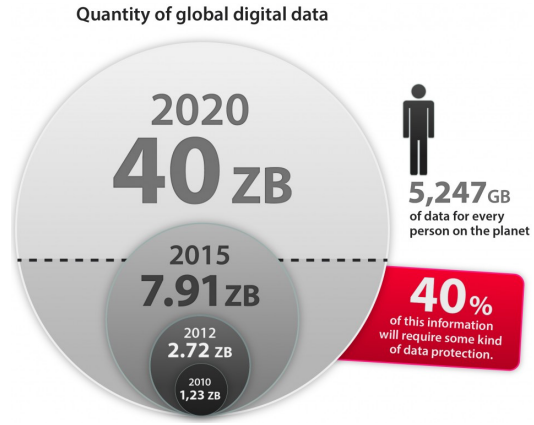
Layer 3: The computer learns to identify more complex shapes and objects.

Layer 4: The computer learns which shapes and objects can be used to define a human face.

(Modified, N. Jones "The Learning Machines," Nature, 2014)

Difficulties in Deep Learning

1. Lack of training samples
   - Huge amount of data available



2. Lack of computational power in HW
   - Multi-core CPUs
   - Graphics Processing Unit (GPU)

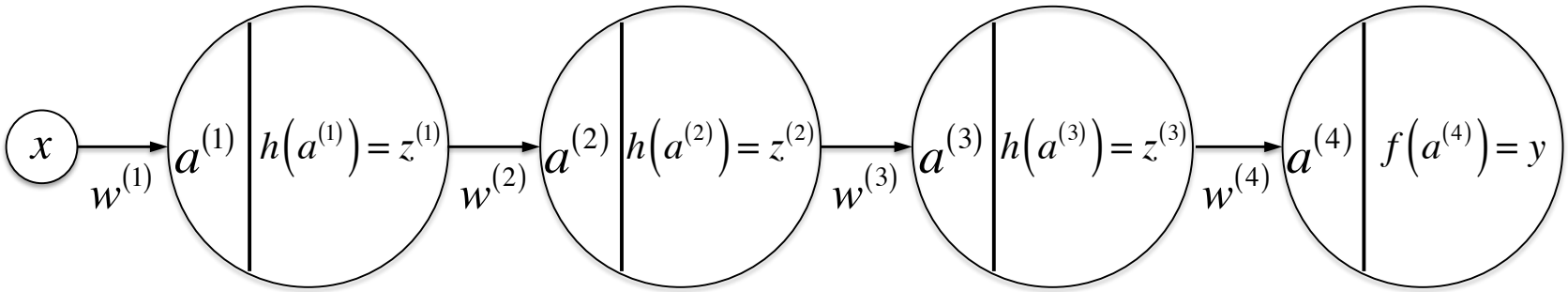
Image sources: (top) http://www.datacenterjournal.com/birth-death-big-data/

(bottom) https://gigaom.com/2014/09/08/nvidia-stakes-its-claim-in-deep-learning-by-making-its-gpus-easier-to-program/

3. Vanishing Gradient Problem

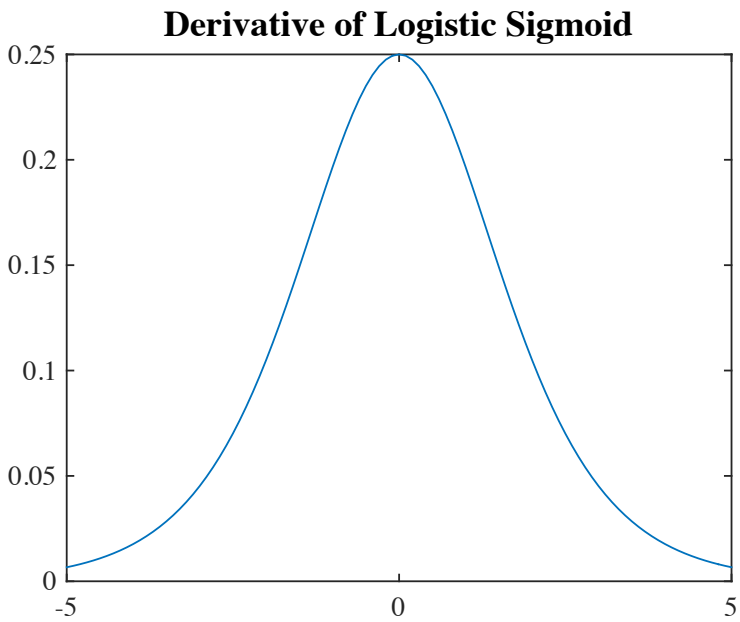
- Backpropagation becomes ineffective due to vanishing gradients after repeated multiplication.
- The gradient tends to get smaller as we propagate backward through the hidden layers.
- Neurons in the earlier layers learn much more slowly then neurons in later laters.

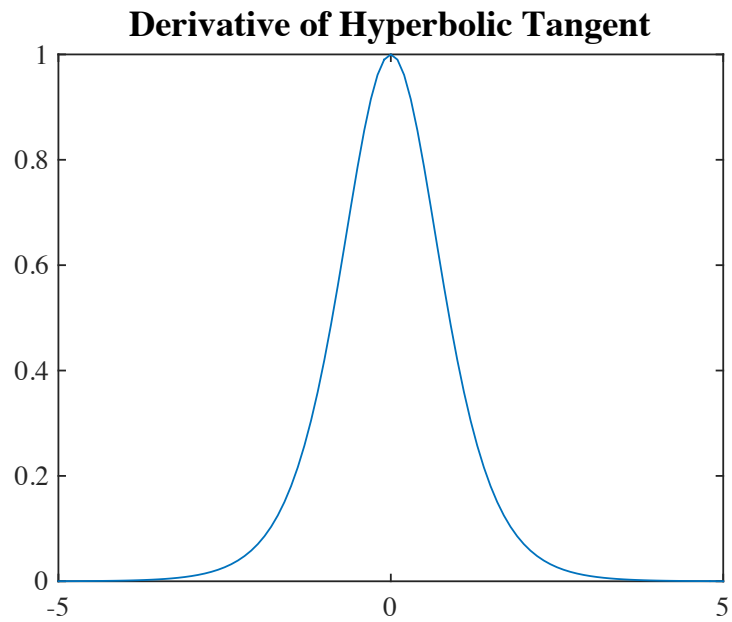
[Cause of Vanishing Gradient Problem]



$$\frac{\partial E_n}{\partial w^{(1)}} = \overbrace{x\, h'\left(a^{(1)}\right) w^{(2)}\, h'\left(a^{(2)}\right) w^{(3)}}^{\delta^{(1)}} \overbrace{h'\left(a^{(3)}\right) w^{(4)}}^{\delta^{(3)}} \underbrace{f'\left(a^{(4)}\right)\left(y_n - t_n\right)}_{\delta^{(4)}=\frac{\partial E_n}{\partial a^{(4)}}}$$

$$\delta^{(2)}$$





- In general, the weights are initialized by using a Gaussian with mean 0 and standard deviation 1.

$$|w_j^{(l)}| < 1$$

Greedy Layer-wise Pre-training [Hinton *et al.*, 2006]

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

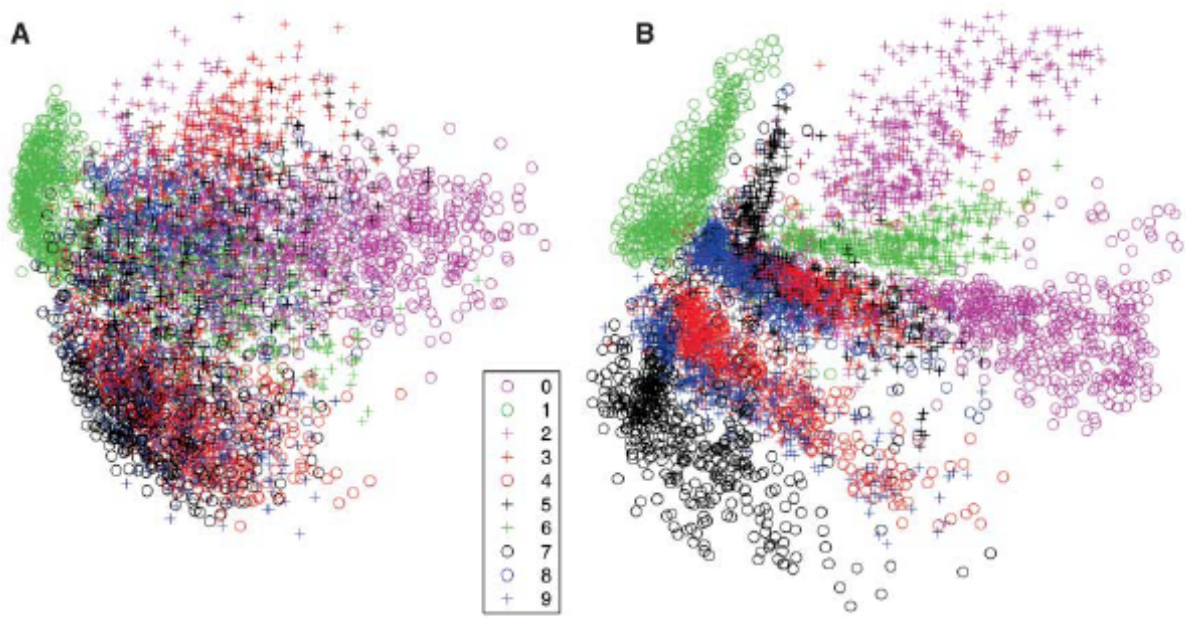
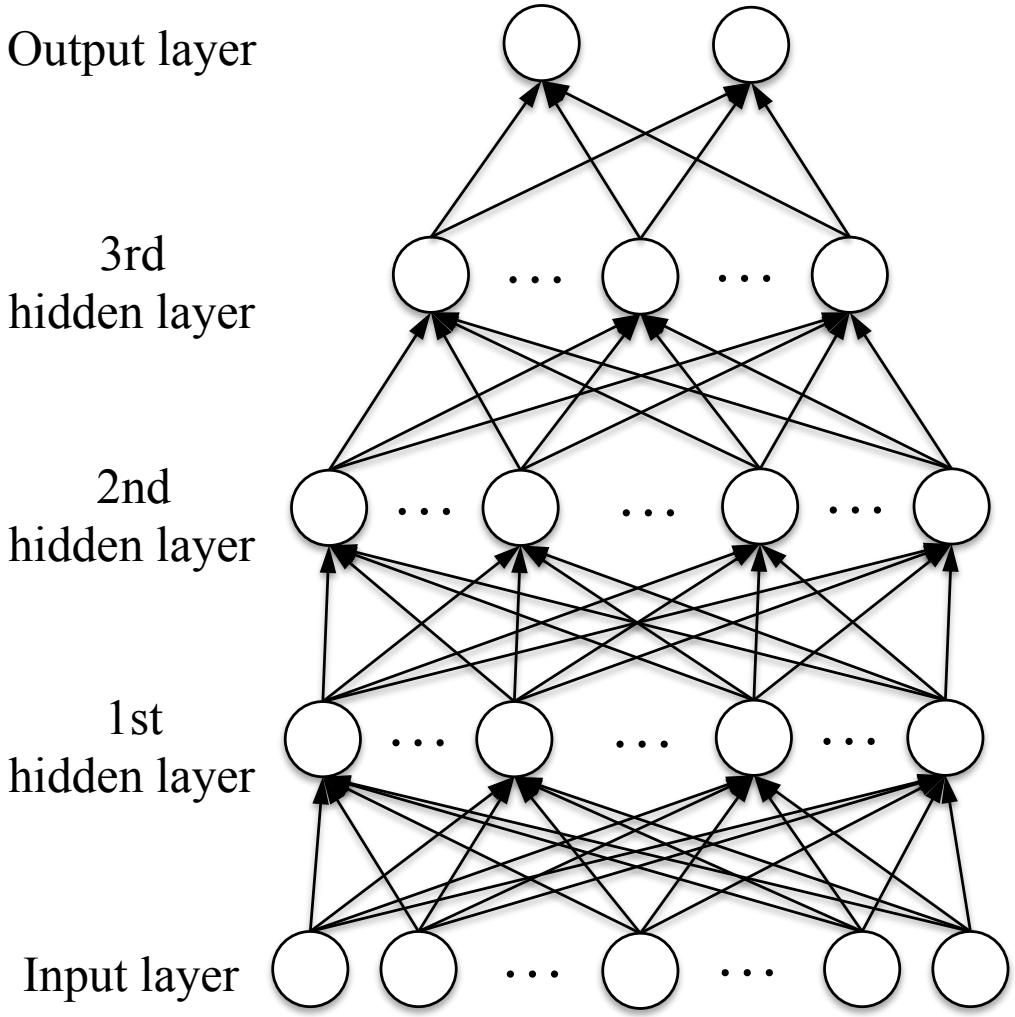
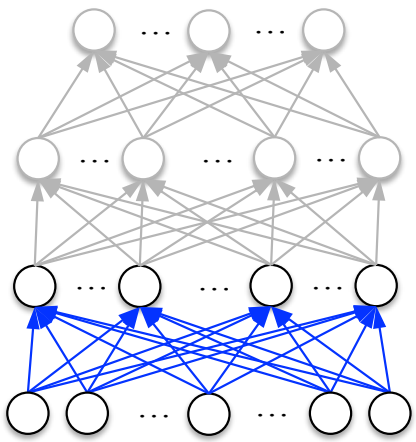


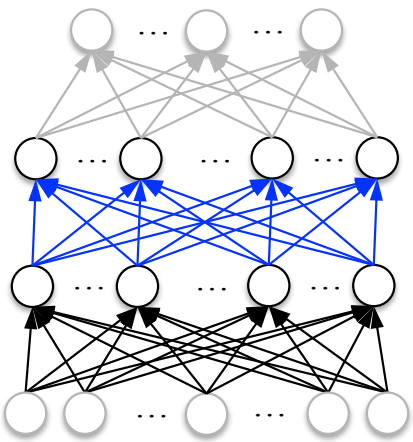
Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).


Output layer
3rd
hidden layer
2nd
hidden layer
1st
hidden layer
Input layer

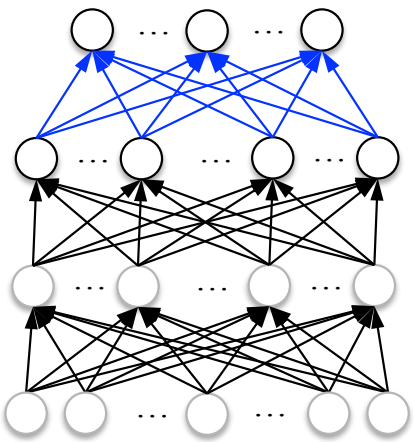
- Train layer 1 in an unsupervised manner
- Train layer 2 while keeping layer 1 fixed in an unsupervised manner
- Train layer 3 while keeping layer 1 & 2 fixed in an unsupervised manner
- Fine-tune the whole network in a supervised manner



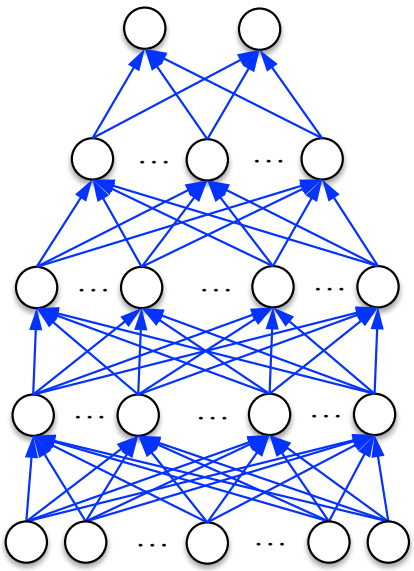
(Pre-train 1st hidden layer)



(Pre-train 2nd hidden layer)

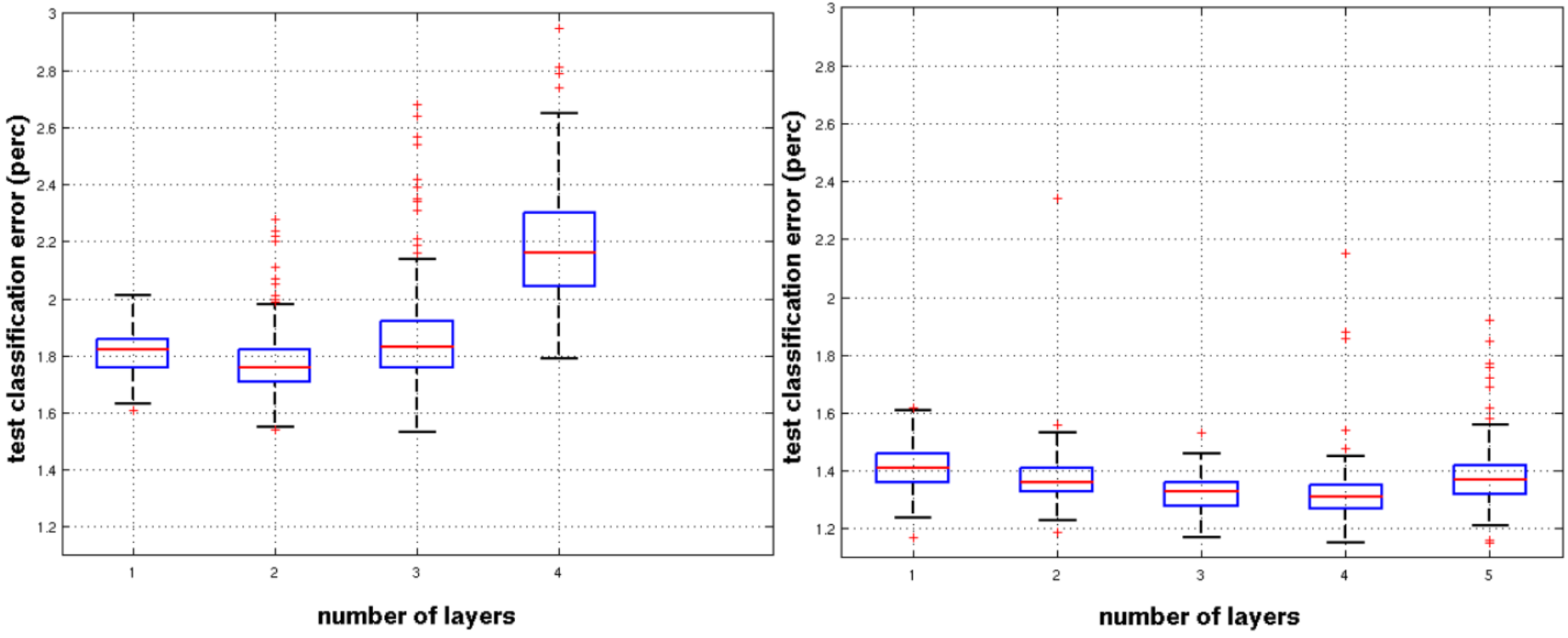


(Pre-train 3rd hidden layer)



(Fine-tuning whole network)

Effects of Pre-Training: Empirical Results



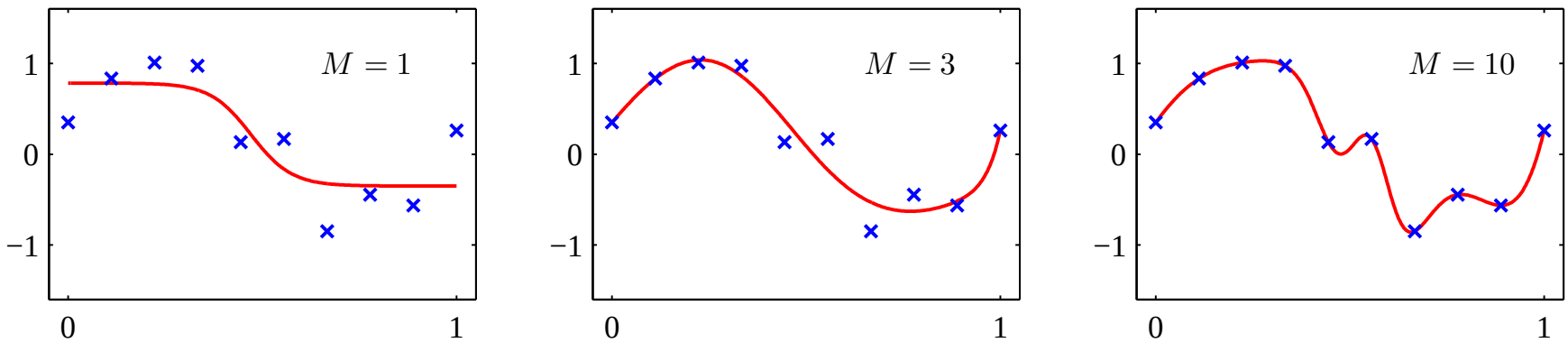
(Erhan *et al.*, AISTATS 2009)

Tricks for Better Learning

- Regularization (ℓ_1-, $\ell_{1/2}$-, ℓ_2-norm)
- Rectified Linear Unit (ReLU) function
- Dropout

Regularization

- # input units = data dimension, # output units = # of target values
- # hidden units (M): free parameter



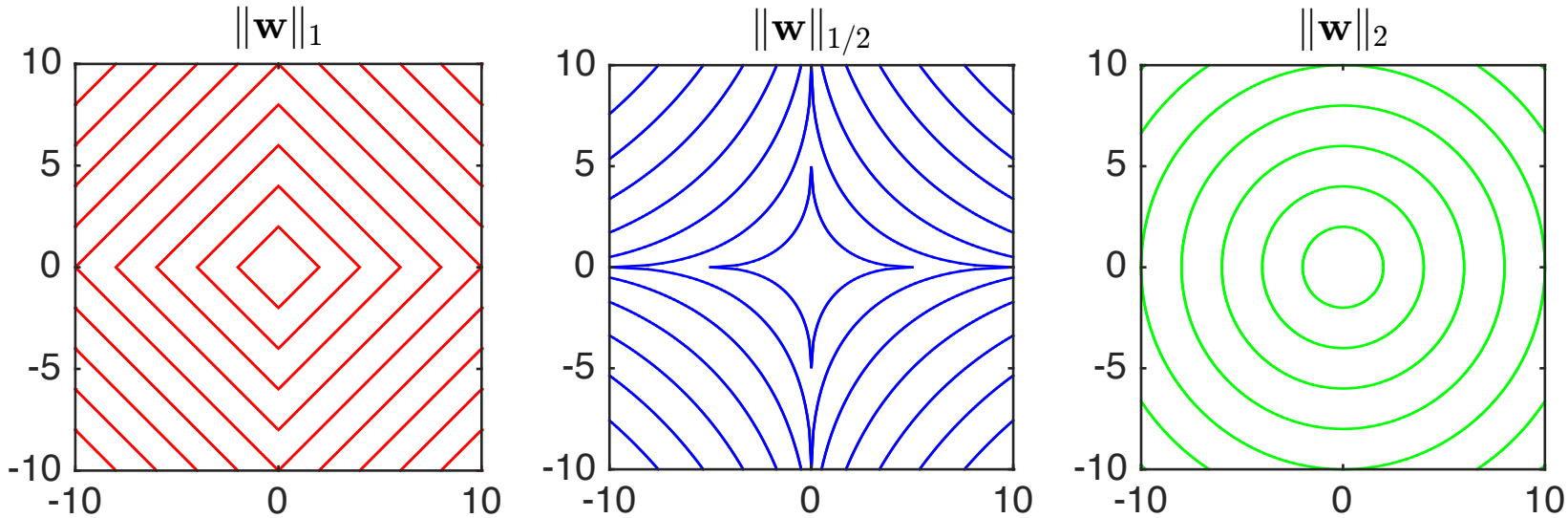
Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set.

Choose a relatively large M and control complexity by the addition of a regularization term to the error function

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda R(\mathbf{w})$$

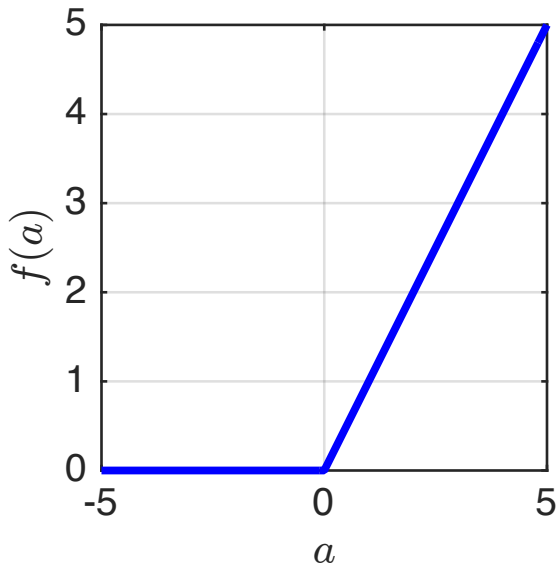
$$R(\mathbf{w}) \in \left\{ \|\mathbf{w}\|_1, \|\mathbf{w}\|_{1/2}, \|\mathbf{w}\|_2 \right\}$$

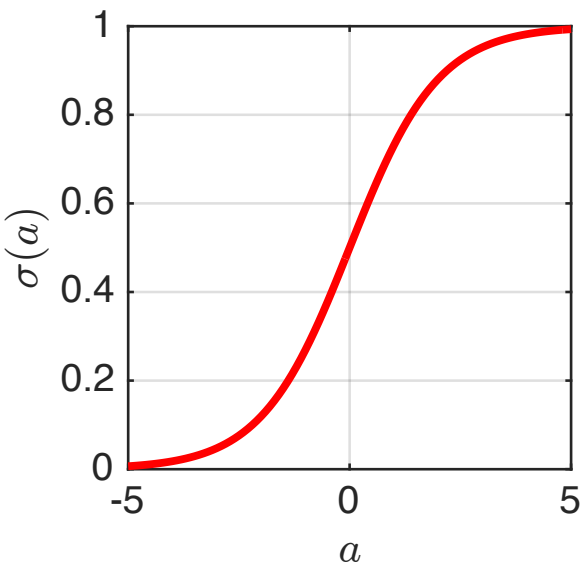
- λ: regularization coefficient

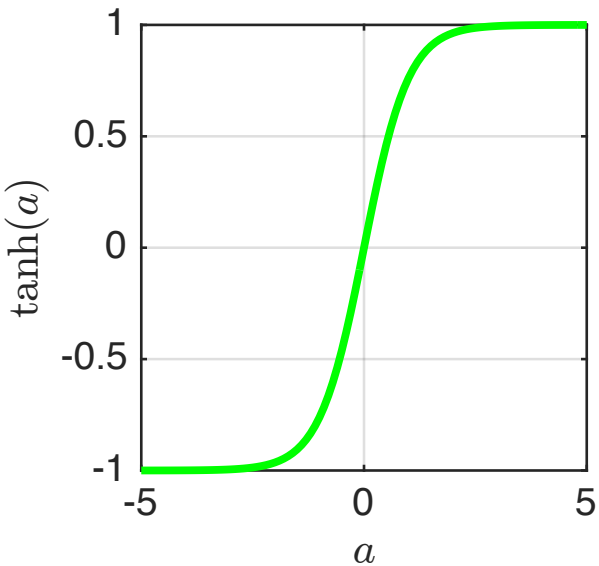


ReLU [Glorot *et al.*, 2011]

- Rectified linear unit: $f(a) = \max(0, a)$
- Sigmoid(logistic): $\sigma(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent: $\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$





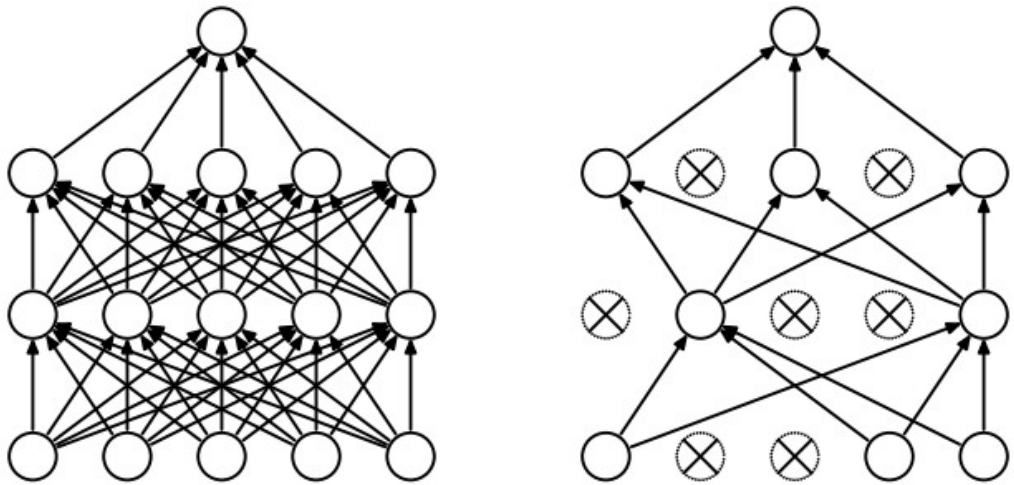


- Non-differentiable at $z = 0$
  - Highly unlikely that it will be at exactly $z = 0$ at any time
  - In practice, set either 0 or 1

- Unbounded on the positive side
  - *Regularization* to limit the magnitude of weights
  - ReLU+ℓ_2-regularization without pre-training: slightly superior or similar performance to pre-training [Glorot *et al.*, 2011]

Dropout [Hinton *et al.*, 2012]

- Randomly deactivate, *e.g.*, 50%, of the neurons in a network on each training iteration



(a) Standard Neural Net (b) After applying dropout.

- Preventing co-adaptation of neurons: a neuron cannot rely on the presence of particular other neurons [Krizhevsky *et al.*, 2012]
- In testing, all neurons are on but the weights are halved to maintain the same output range [Hinton *et al.*, 2012]

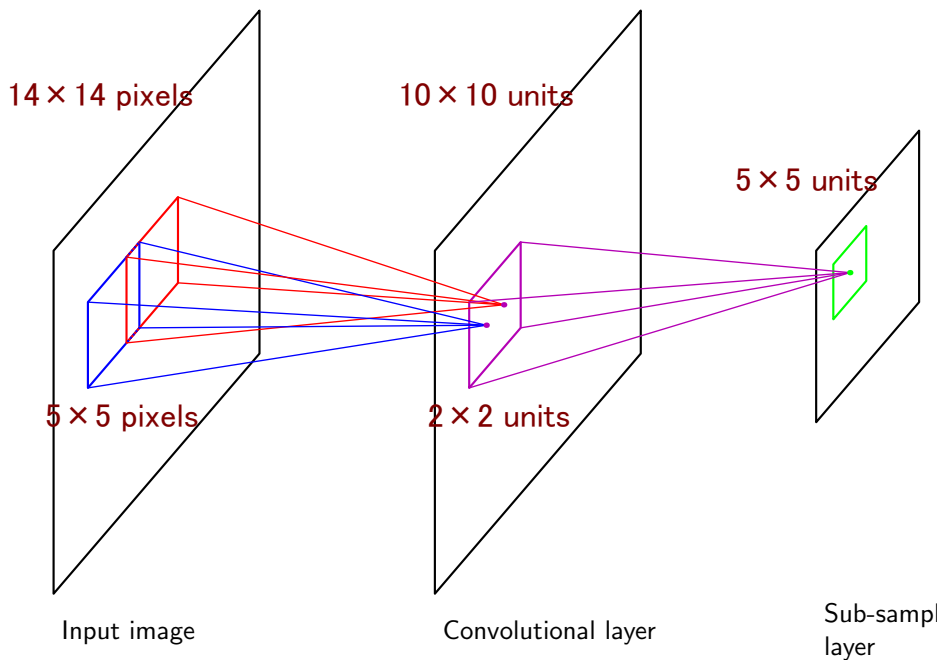
Convolutional Neural Networks

Convolutional Neural Networks

- Nearby pixels are more strongly correlated than more distant pixels
  - Modern approaches to computer vision exploit this property by extracting local features
- Information can be merged at later stages to get higher order features and ultimately to yield information about the image as whole

Three mechanisms

1. Local receptive fields
2. Weight sharing
3. Subsampling



[Convolutional Layer]

- Units are organized into planes ('*feature maps*')
- Units in a feature map take inputs only from a subregion of the input
- All units in a feature map are constrained to share the same weight values

- Input values from a patch (receptive field) are linearly combined using the weights and the bias, and the result is transformed by a sigmoidal nonlinearity

$$C_j = h\left(X * W_j + b_j\right)$$

where $*$: a convolution operator

  - All of the units in a feature map detect the same pattern but at different locations in the input

[Subsampling Layer]

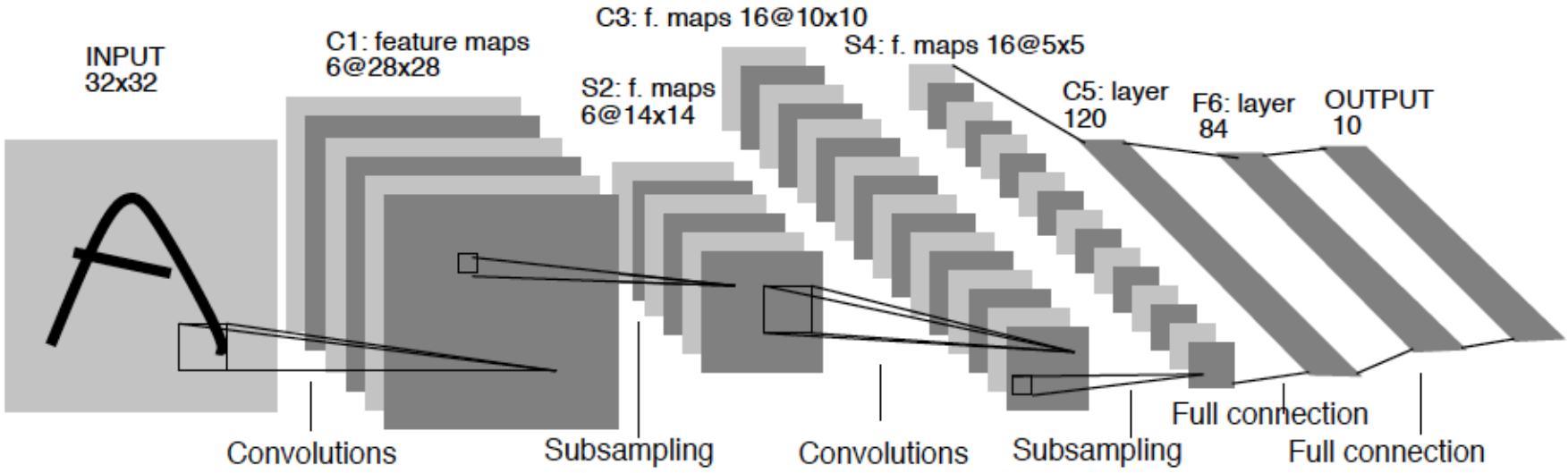
$$S_j = \text{down}\left(C_j\right)$$

- $\text{down}\left(\cdot\right)$: average or max function
- The receptive fields are chosen to be continuous and non-overlapping

- To reduce computation time and to gradually build up further *spatial* and *configural* invariance
  - Relatively insensitive to small shifts of the image in the corresponding regions of the input space
  - A small sub-sampling factor is desirable in order to maintain *specificity* at the same time

Practical Architecture

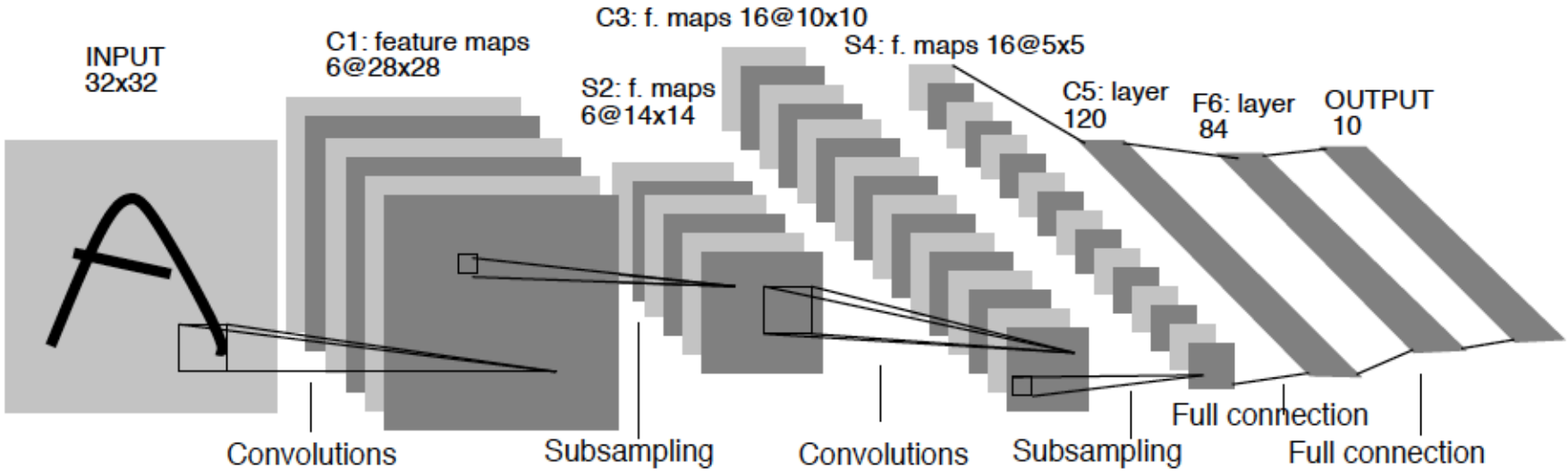
LeNet [LeCun, 1998]



- Several pairs of convolutional and subsampling layers

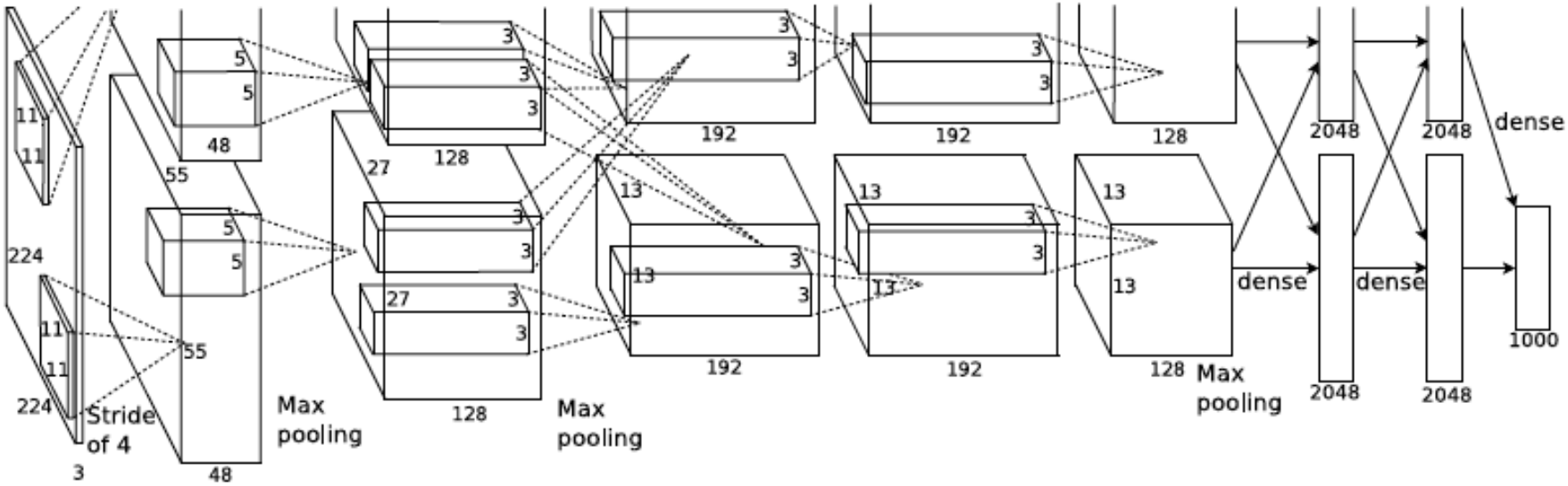
$$C_j^{(l)} = h\left(\sum_i S_i^{(l-1)} * W_{ji}^{(l)} + b_j^{(l)}\right)$$

$$S_j^{(l)} = \text{down}\left(C_j^{(l-1)}\right)$$

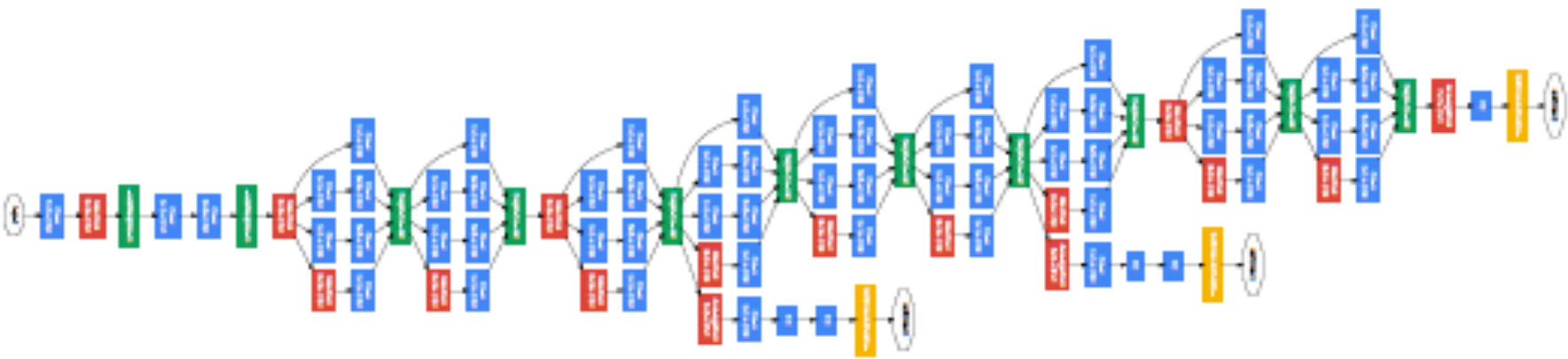


- Gradual reduction in spatial resolution is compensated by an increasing number of features
- The final layer of the network would typically be fully connected

Deep Convolutional Neural Net



AlexNet [Krizhevsky *et al.*, 2012]



GoogleNet [Szegedy *et al.*, 2015]

Tools for DL

- Caffe (https://github.com/BVLC/caffe)
- Theano (http://deeplearning.net/software/theano)
- Torch (http://torch.ch/)
- Matlab (https://github.com/rasmusbergpalm/DeepLearnToolbox)

	Linux	Windows	Easy to install	Easy to use	Speed	Easy to prototype	Large user base
Caffe	✔	✔	Many dependencies	✔	✔	Tuned for image applications	✔
Theano	✔	✔	✔	Steep learning curve	✔	✔	✔
Torch	✔	?			✔		
Ramus	✔	✔	✔	✔		✔	

References


C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006


Y. LeCun, Y. Bengio and G. Hinton, "Deep Learning," Nature, 2015.


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 1998.


J. Bourvrie, "Notes on Convolutional Neural Networks," 2006.


G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, 2006


G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, 2006


G. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," In: Montavon, G., Orr, G. B., Müller, K.-R. (Eds.), Neural Networks: Tricks of the Trade (2nd ed.), LNCS, 2012


Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," NIPS, 2007


Y. Bengio, Learning deep architectures for AI. Foundations and Trends in Machine Learning, 2009


A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," NIPS, 2012


G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv, 2012


Y. LeCun, L. Bottou, G. Orr, and K.-R. Müller, "Efficient BackProp," Neural Networks: Tricks of the Trade, LNCS, 2002


A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," NIPS, 2012


C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," CVPR, 2015


M. Nielson, Neural Networks and Deep Learning, `http://neuralnetworksanddeeplearning.com/index.html`

MICCAI 15 MUNICH
18th International Conference 2015 · October 5th to 9th · Munich, Germany
on Medical Image Computing and Computer Assisted Interventions

Thank you for your attention!

Q&A

hisuk AT korea.ac.kr

KOREA UNIVERSITY MiL
Machine Intelligence Lab.