[SKT AI Course: Deep Learning Basics]

Gentle Introduction to Deep Learning

Heung-Il Suk

hisuk@korea.ac.kr

http://www.ku-milab.org

Department of Brain and Cognitive Engineering,
Korea University

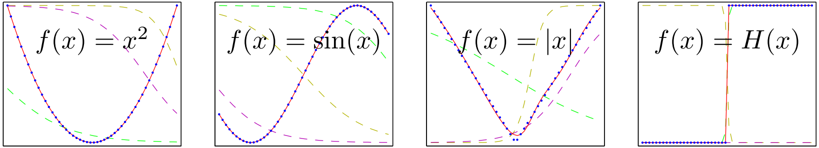
September 19, 2017

Contents

Deep Learning In a Nutshell

Universal Approximate Theorem [Hornik, 1991]

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



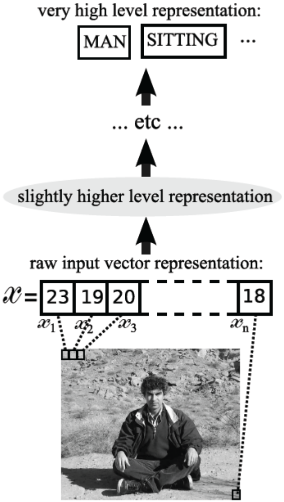
- Blue dots: 50 data points uniformly $(-1, 1)$; 3 hidden units (tanh), linear output units
- Network output (red curve), outputs of thee hidden units

Deep Neural Networks

- Possible to approximate complex functions to the same accuracy using a deeper network with much fewer total units [Bengio, 2009]
- Smaller number of parameters, requiring a smaller dataset to train [Schwarz *et al.*, 1978]
- Hierarchical feature representation (fine-to-abstract)

$$y_k = f\left(\sum_l W_{kl}^{(L)} \underbrace{h\left(\sum_m W_{ml}^{(L-1)} h\left(\cdots h\left(\sum_i W_{ij}^{(1)} x_i\right)\right)\right)}_{\phi_l(\mathbf{x})}\right)$$

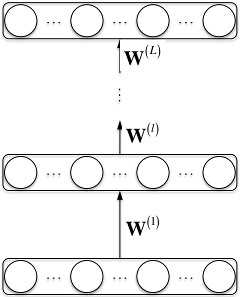
Goal of deep architectures: *to learn feature hierarchies*



(High-level) abstraction

(Intermediate-level) edges, local shapes, object parts, etc.

(Low-level) pixel intensity

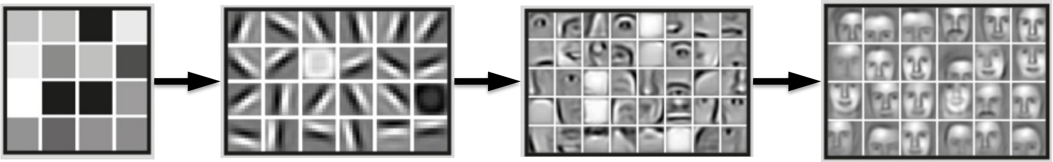


Deep architecture
for hierarchical feature representation

[Bengio, 2009]

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.

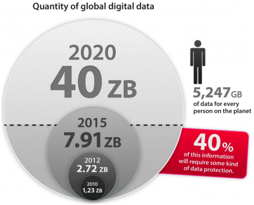
Layer 3: The computer learns to identify more complex shapes and objects.

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



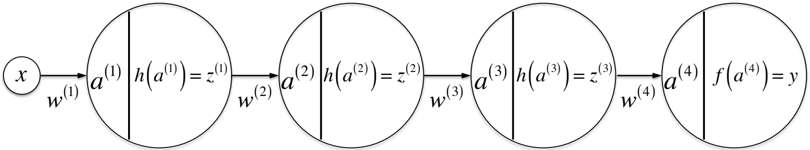
2. Lack of computational power in HW
   - Multi-core CPUs
   - Graphics Processing Unit (GPU)

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.
- Units in the earlier layers learn much more slowly then units in the later laters.



(Recep.) $$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \underbrace{\frac{\partial E_n}{\partial a_j^{(l)}}}_{\equiv \delta_j^{(l)}} \underbrace{\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}}}_{z_i^{(l-1)}} = \left\{ h'\left(a_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)} \right\} z_i^{(l-1)}$$

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

In a general feed-forward neural network: multiple units in each layer

$$\boldsymbol{\delta}^{(l)} = H'\left(\mathbf{a}^{(l)}\right)\left(\mathbf{W}^{(l+1)}\right)^{\top} H'\left(\mathbf{a}^{(l-1)}\right)\left(\mathbf{W}^{(l+2)}\right)^{\top} \cdots H'\left(\mathbf{a}^{(L)}\right)\frac{\partial E_n}{\partial \mathbf{a}^{(L)}}$$

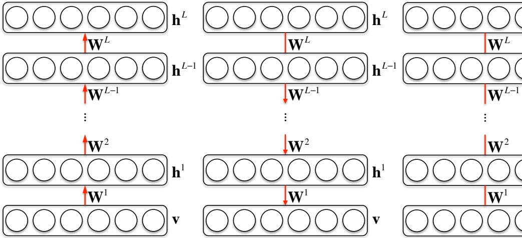
$$\text{where } H'\left(\mathbf{a}^{(l)}\right) = \begin{bmatrix} h'\left(a_1^{(l)}\right) & & & & 0 \\ & \ddots & & & \\ & & h'\left(a_j^{(l)}\right) & & \\ & & & \ddots & \\ 0 & & & & h'\left(a_M^{(l)}\right) \end{bmatrix}$$

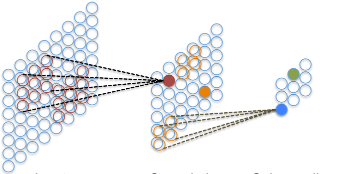
- In general, the weights are initialized by using a Gaussian with mean 0 and std 1.

$$\left|w_j^{(l)}\right| < 1$$

Deep Models

- Stacked Auto-Encoder (SAE)
- Deep Belief Network (DBN)
- Deep Boltzmann Machine (DBM)
- Convolutional NN (CNN)
- Recurrent NN (RNN)





Greedy Layer-wise Pre-training [Hinton and Salakhutdinov, 2006]

Reducing the Dimensionality of Data with Neural Networks

Science AAAS

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.

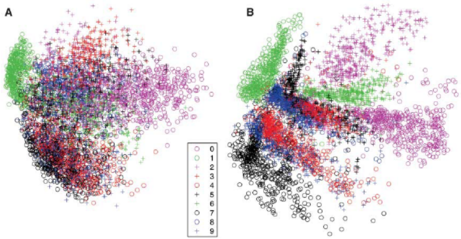
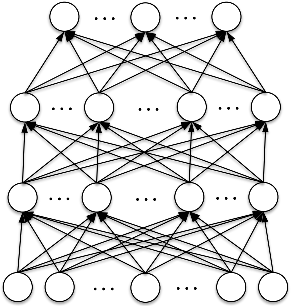


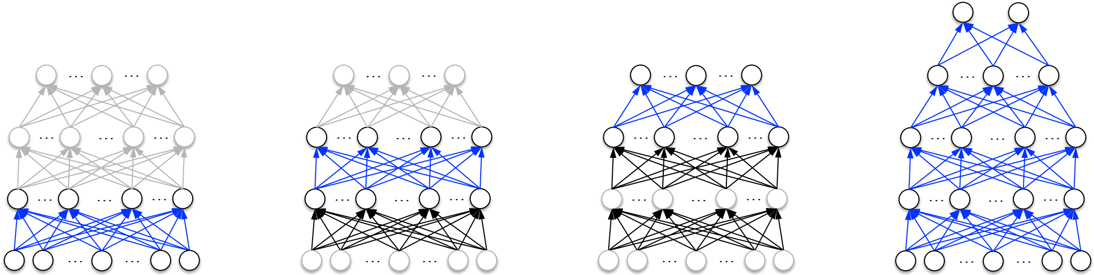
Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

Stacked Auto-Encoder (SAE)



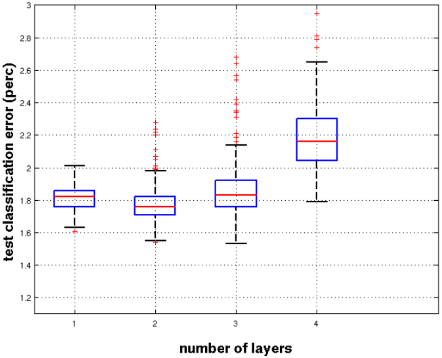
- Train layer 1 in an unsupervised manner
- Train layer 2 while keeping layer 1 fixed in an unsupervised manner
- Train layer 3 while keeping layer 1 & 2 fixed in an unsupervised manner

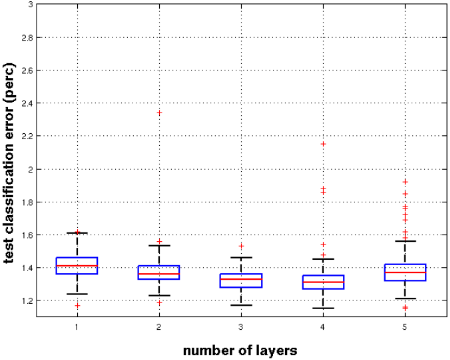
⋆ Fine-tune the whole network in a supervised manner



(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





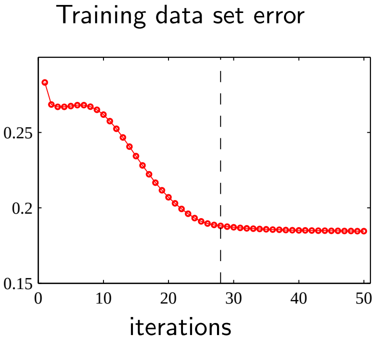
[Erhan *et al.*, 2009]

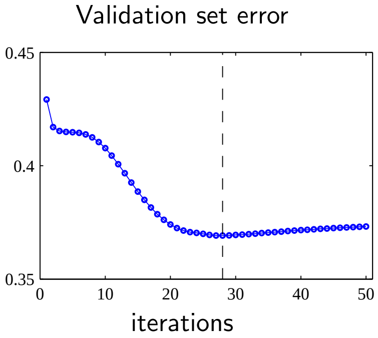
Algorithmic Advances

Overview

- Early stopping
- Regularization (ℓ_1-, $\ell_{1/2}$-, ℓ_2-norm)
- Rectified Linear Unit (ReLU) [Nair and Hinton, 2010]
- Denoising [Vincent *et al.*, 2008]
- Dropout [Hinton *et al.*, 2012]
- Dropconnect [Wan *et al.*, 2013]
- Batch normalization [Ioffe and Szegedy, 2015]

Early Stopping



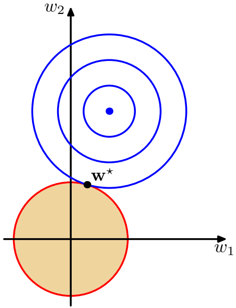


Halting training before a minimum of the training error has been reached represents a way of limiting the effective network complexity.

Weight Decay: ℓ_2-norm Regularization

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

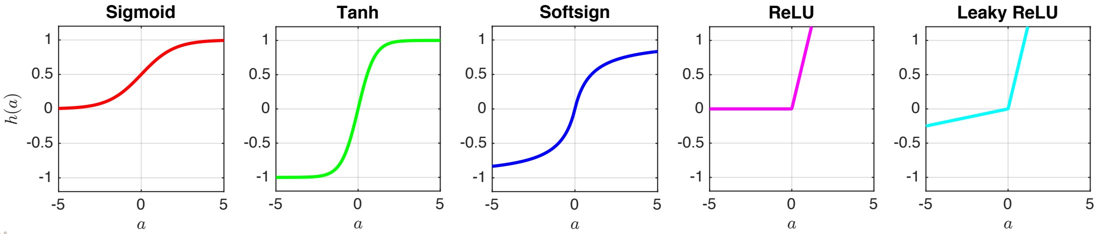
$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}$$



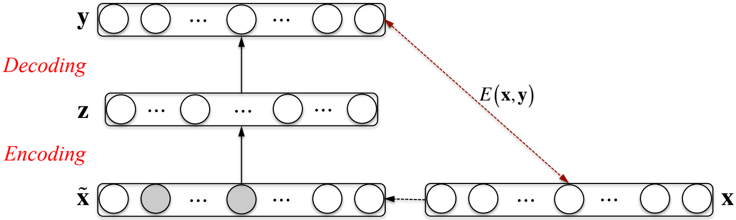
- λ: regularization coefficient
- Can be interpreted as the negative logarithm of a zero-mean Gaussian prior distribution over the weight vector $\mathbf{w}$

Activation Functions

- Logistic sigmoid: $h(a) = 1/\left(1 + \exp[-a]\right)$
- Tanh: $h(a) = \left(\exp[a] - \exp[-a]\right) / \left(\exp[a] + \exp[-a]\right)$
- Softsign [Bergstra *et al.*, 2009]: $h(a) = 1/\left(1 + |a|\right)$
- ReLU [Nair & Hinton *et al.*, 2010]: $h(a) = \max\left(0, a\right)$
- Leaky ReLU [Maas *et al.*, 2013]: $h(a) = \max\left(\kappa a, a\right)$, $0 < \kappa < 1$
- Parametric ReLU [He *et al.*, 2015]: $h(a) = \max\left(\kappa^{*} a, a\right)$



Denoising Auto-Encoder [Vincent et al., 2008]

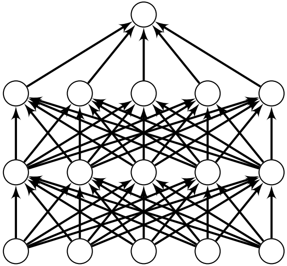


- In order to force the hidden layer to discover more robust features and prevent it from simply learning the identity, train the auto-encoder to reconstruct the input from a corrupted version of it.
- $\tilde{\mathbf{x}}$: corrupted input $\mathbf{x}$ by adding noise
- Train parameters so that the output $\mathbf{y}$ to be equal to $\mathbf{x}$

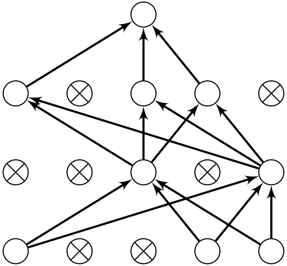
Dropout [Hinton et al., 2012]

- Preventing co-adaptation of neurons: a neuron cannot rely on the presence of particular other neurons [Krizhevsky *et al.*, 2012]

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



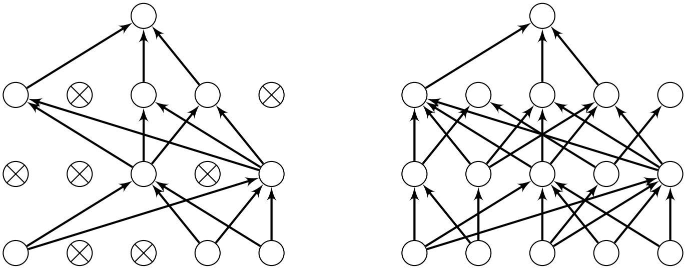
(a) Standard DNN



(b) After applying dropout

Dropconnect [Wan et al., 2013]

- Randomly remove a set (*e.g.*, 50%) of the weights and biases within the network



(a) After applying dropout (b) After applying dropconnect

Batch Normalization [Ioffe and Szegedy, 2015]

- Internal covariate shift: change in the distribution of network activations due to the change in network parameters during training makes the training time slow
- Performing normalization for each mini-batch and backpropagating the gradients through the normalization parameters (*i.e.*, scale and shift)
- For each unit in a layer l, their value is normalized

$$\hat{x}_k^{(l)} = \frac{x_k^{(l)} - \mathbb{E}[x_k^{(l)}]}{\sqrt{\mathrm{Var}[x_k^{(l)}]}}$$

where $k = 1, \ldots, F^{(l)}$ and $F^{(l)}$: number of units in the layer l

- A pair of learnable parameters $\gamma_k^{(l)}$ and $\beta_k^{(l)}$ are then introduced to scale and shift the normalized values to restore the representation power of the network

$$y_k^{(l)} = \gamma_k^{(l)} \hat{x}_k^{(l)} + \beta_k^{(l)}$$

Thank you
for your attention!!!

(Q & A)

hisuk (AT) korea.ac.kr

http://www.ku-milab.org

Machine
Intelligence
Lab