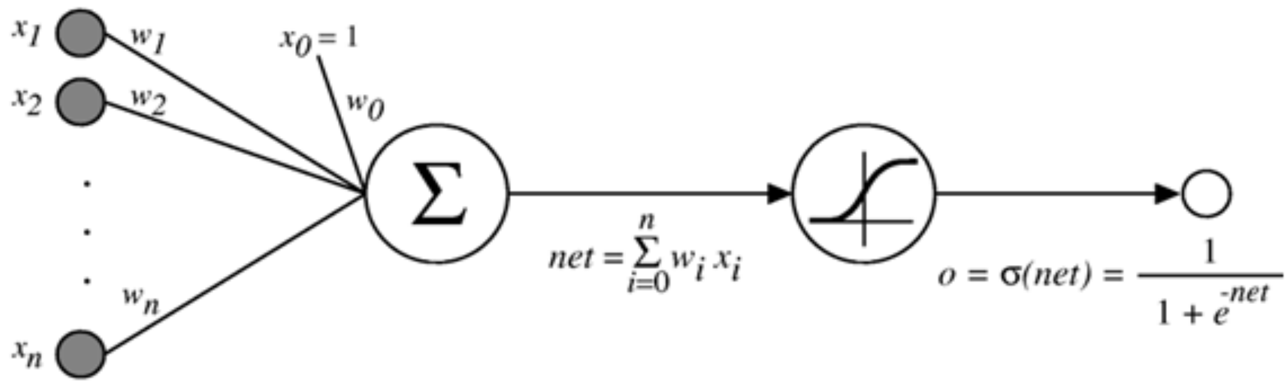
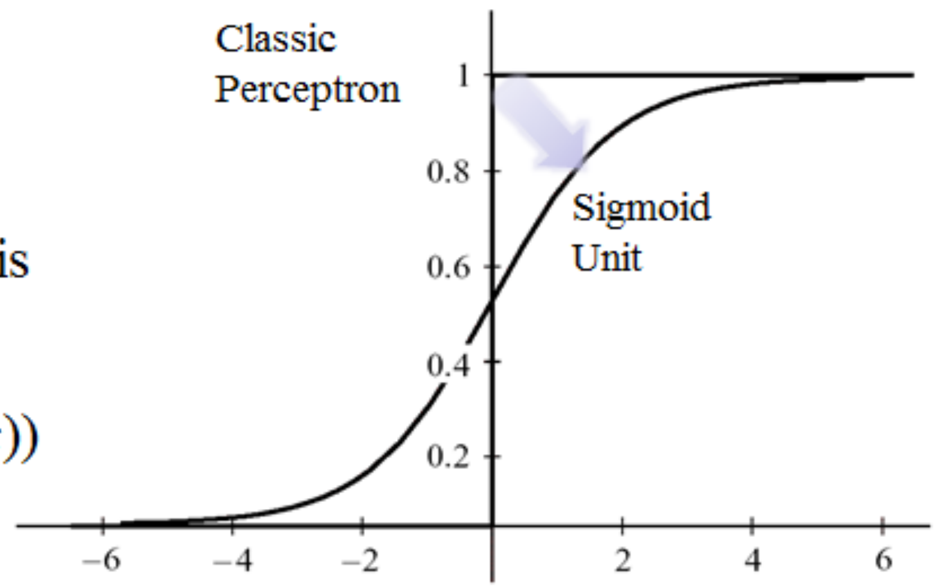
Multilayer Perceptron

Eun-Sol Kim (김은솔)

Artificial Intelligence

Sigmoid Unit





Sigmoid function is **Differentiable**

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

Learning Algorithm of Sigmoid Unit

- Loss Function

Target (Label) — Model Output

$$\varepsilon = (d - f)^2$$

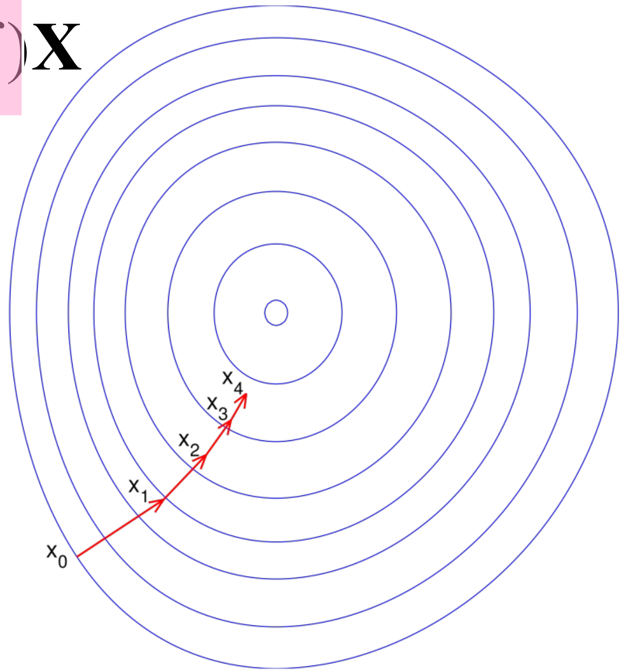
- Gradient Descent Update

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d - f)\frac{\partial f}{\partial s}\mathbf{X} = -2(d - f)\boxed{f(1 - f)}\mathbf{X}$$

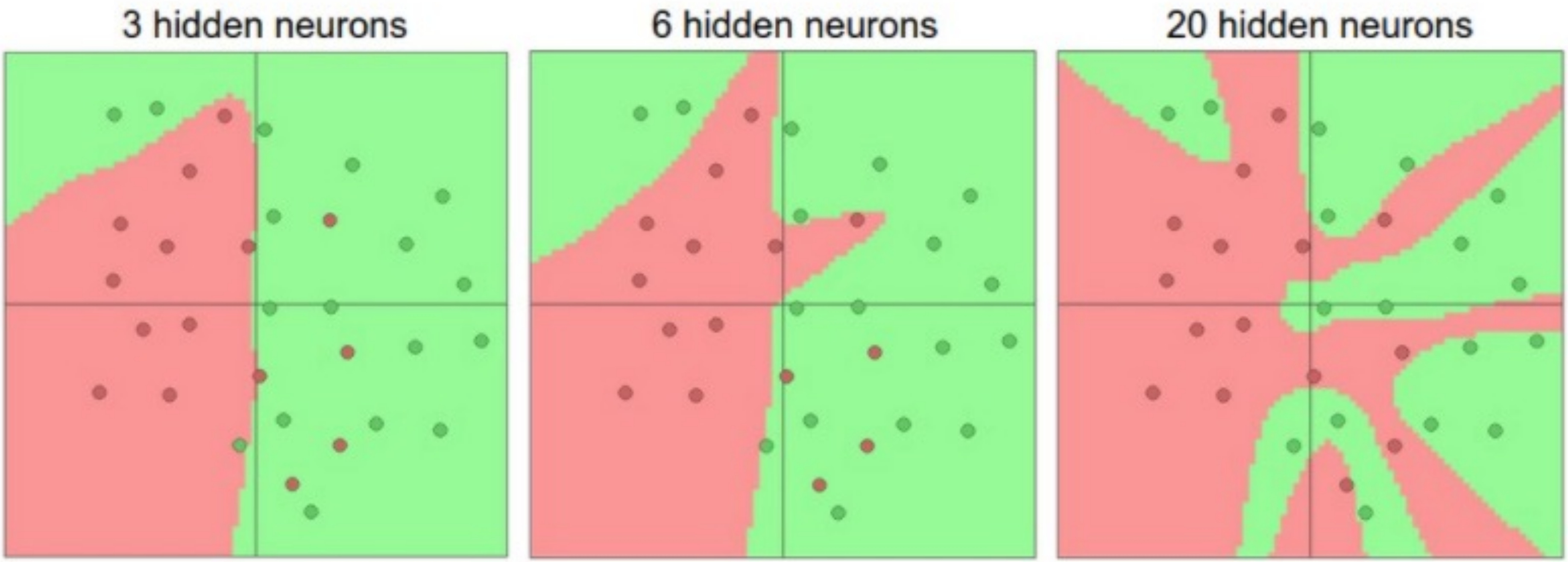
$$f(s) = 1/(1 + e^{-s})$$

$$f'(s) = f(s)(1 - f(s))$$

$$\mathbf{W} \leftarrow \mathbf{W} + c(d - f)f(1 - f)\mathbf{X}$$



Setting number of layers and their sizes



https://cs231n.github.io/neural-networks-1/

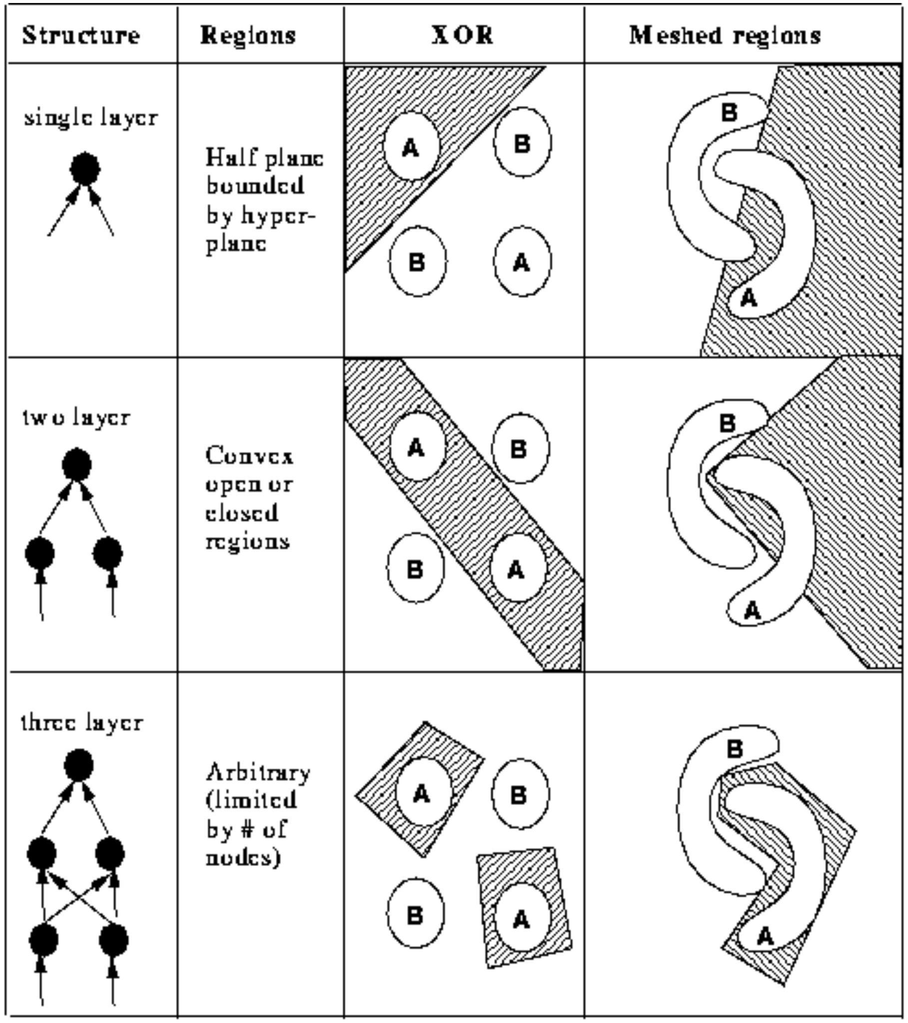
Need for Multiple Units and Multiple Layers

- Multiple boundaries are needed (e.g. XOR problem)

→ Multiple Units

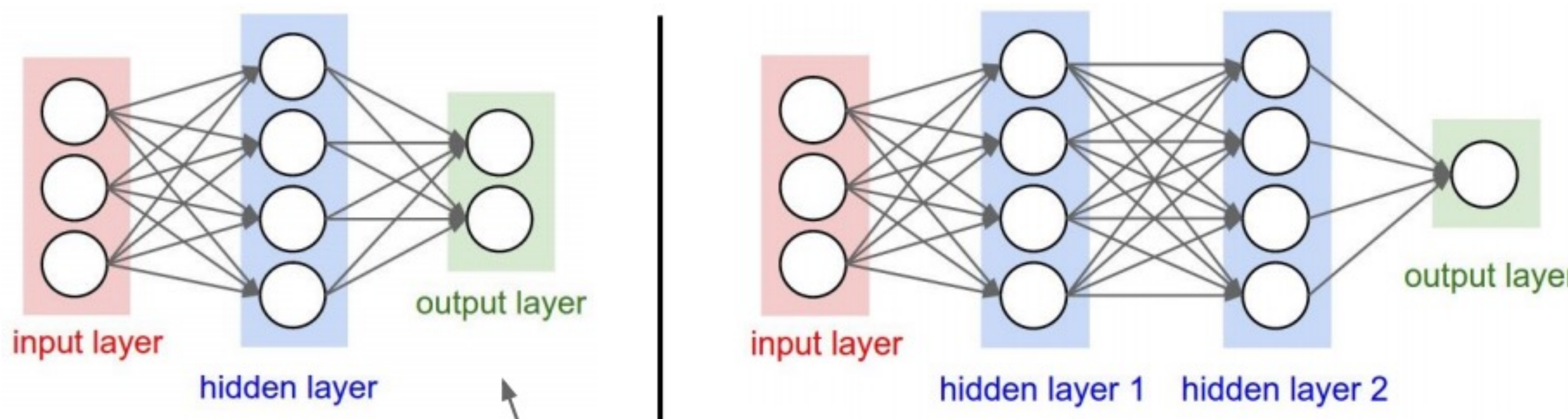
- More complex regions are needed (e.g. Polygons)

→ Multiple Layers



http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Structure of Multilayer Perceptron



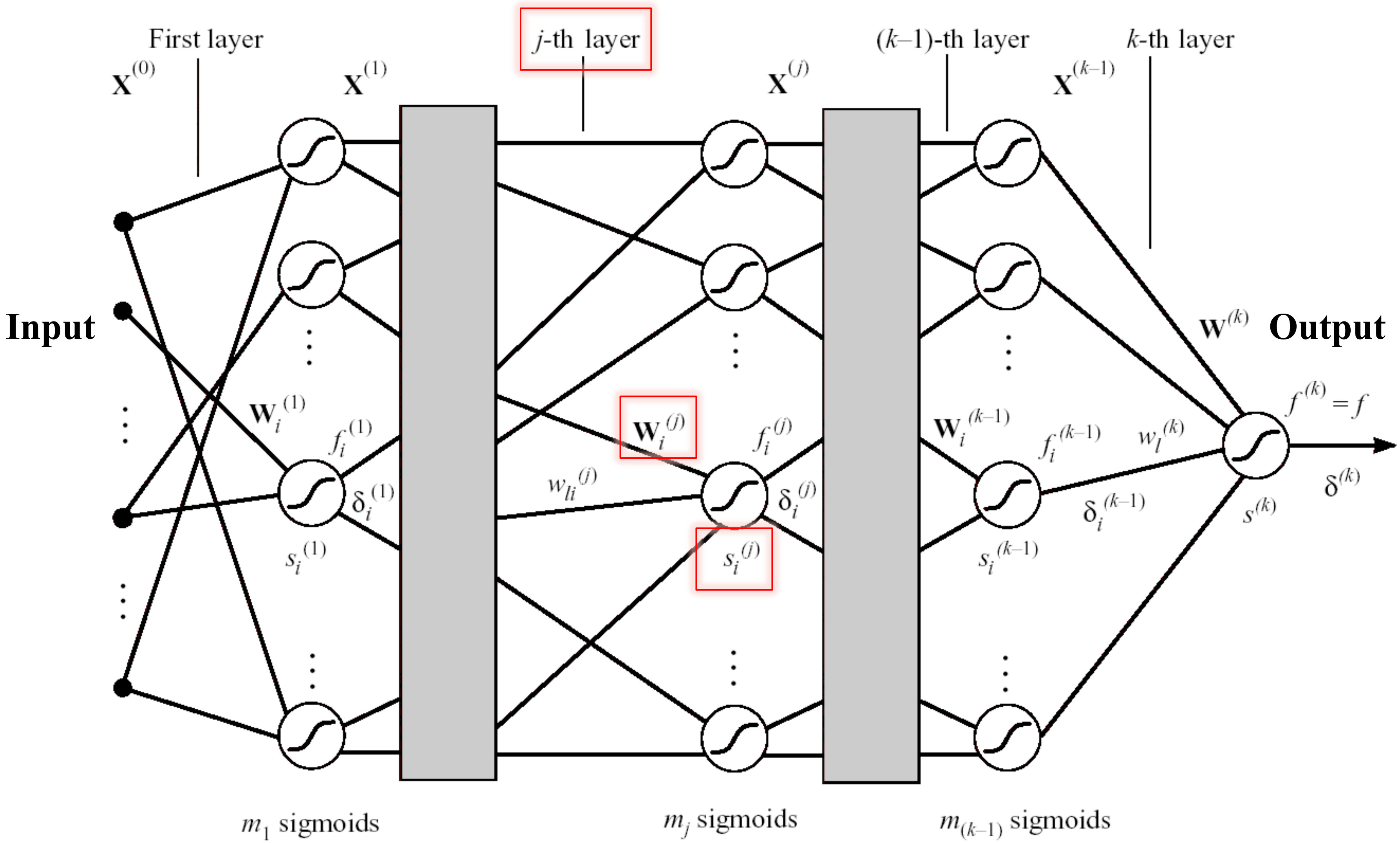
“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Structure of Multilayer Perceptron (MLP; Artificial Neural Network)

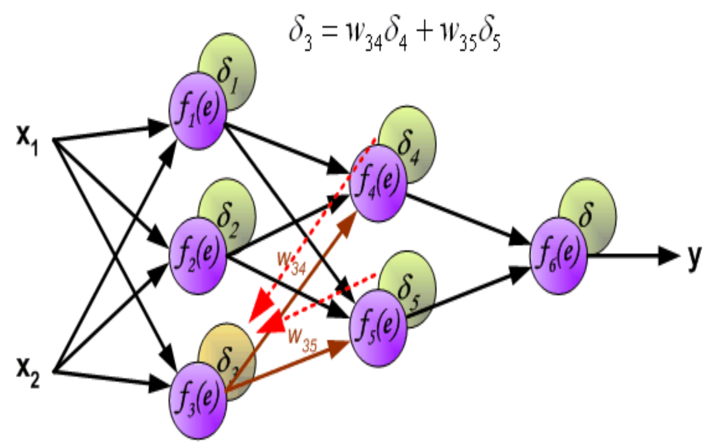


Learning Parameters of MLP

- Loss Function
  - We have the same Loss Function
  - But the # of parameters are now much more (Weight for **each layer** and **each unit**)
  - To use Gradient Descent, we need to calculate the **gradient for all the parameters**

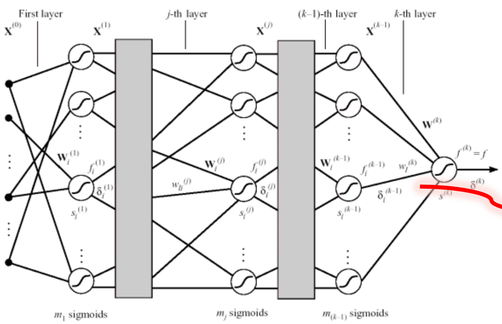
$$\varepsilon = (\overbrace{d}^{\text{Target}} - \overbrace{f}^{\text{Unit Output}})^2$$

- Recursive Computation of Gradients
  - Computation of loss-gradient of **the top-layer** weights is **the same** as before
  - Using the **chain rule**, we can compute the loss-gradient of lower-layer weights **recursively (Back Propagation)**



Back Propagation Learning Algorithm (1/3)

- Gradients of top-layer weights and update rule



Gradient Descent update rule

$$\varepsilon = (d - f)^2$$

$$\frac{\partial \varepsilon}{\partial \mathbf{W}} = -2(d - f)\frac{\partial f}{\partial s}\mathbf{X} = -2(d - f)f(1 - f)\mathbf{X}$$

$$\mathbf{W} \leftarrow \mathbf{W} + c(d - f)f(1 - f)\mathbf{X}$$

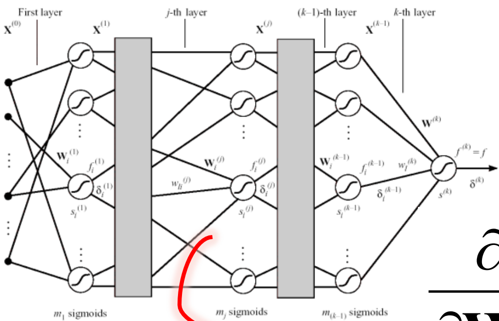
- Store intermediate value delta for later use of chain rule

$$\delta^{(k)} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} = (d - f)\frac{\partial f}{\partial s_i^{(j)}}$$

$$= (d - f)f(1 - f)$$

Back Propagation Learning Algorithm (2/3)

- Gradients of <u>lower-layer</u> weights



Weighted sum

$$s_i^{(j)} = \mathbf{X}^{(j-1)} \cdot \mathbf{W}_i^{(j)}$$

$$\frac{\partial \varepsilon}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \frac{\partial s_i^{(j)}}{\partial \mathbf{W}_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \mathbf{X}^{(j-1)}$$

$$= -2(d-f)\frac{\partial f}{\partial s_i^{(j)}} \mathbf{X}^{(j-1)} = -2\delta_i^{(j)} \mathbf{X}^{(j-1)}$$

Local gradient

$$\frac{\partial \varepsilon}{\partial s_i^{(j)}} = \frac{\partial (d-f)^2}{\partial s_i^{(j)}} = -2(d-f)\frac{\partial f}{\partial s_i^{(j)}}$$

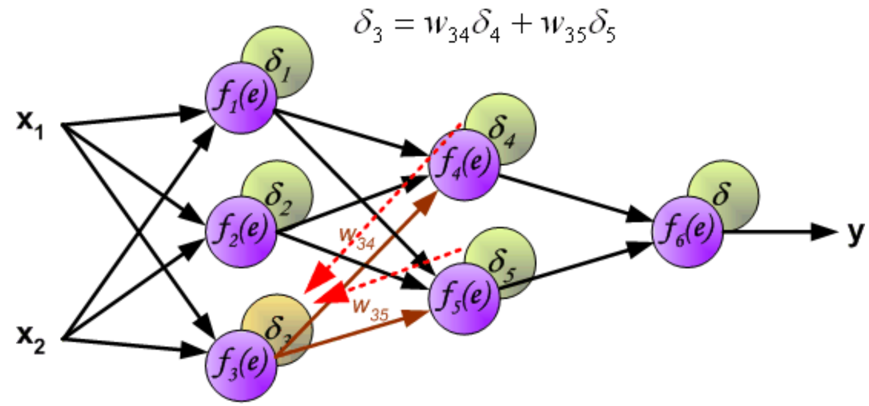
Gradient Descent Update rule for lower-layer weights

$$\mathbf{W}_i^{(j)} \leftarrow \mathbf{W}_i^{(j)} + c_i^{(j)} \delta_i^{(j)} \mathbf{X}^{(j-1)}$$

Back Propagation Learning Algorithm (3/3)

- Applying chain rule, recursive relation between delta's

$$\delta_i^{(j)} = f_i^{(j)}(1 - f_i^{(j)}) \sum_{l=1}^{m_{j+1}} \delta_i^{(j+1)} w_{il}^{(j+1)}$$



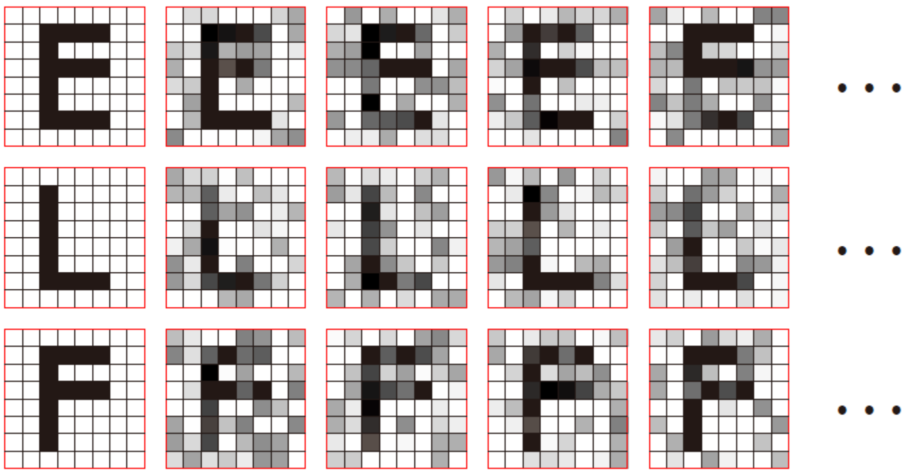
Algorithm: Back Propagation

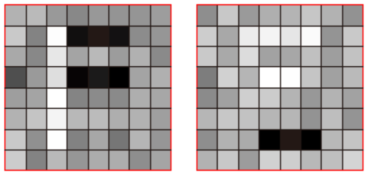
1. **Randomly Initialize weight parameters**
2. **Calculate the activations of all units (with input data)**
3. **Calculate top-layer delta**
4. **Back-propagate delta from top to the bottom**
5. **Calculate actual gradient of all units using delta's**
6. **Update weights using Gradient Descent rule**
7. **Repeat 2~6 until converge**

An example of the MLP

- Example

(a) sample training patterns



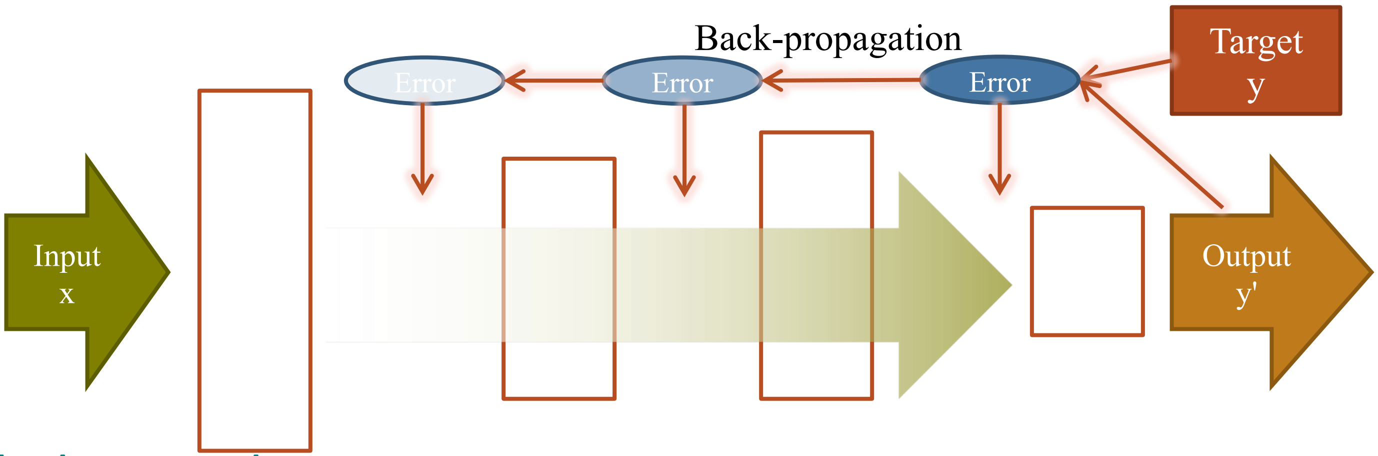


(b) learned input-to-hidden weights

- 64-2-3 network for classifying 3 characters
  - 64-dim inputs
  - 2 hidden units
  - 3 output units
- Learned i-to-h weights
  - Describe feature groupings useful for classification

Limitations and Breakthrough

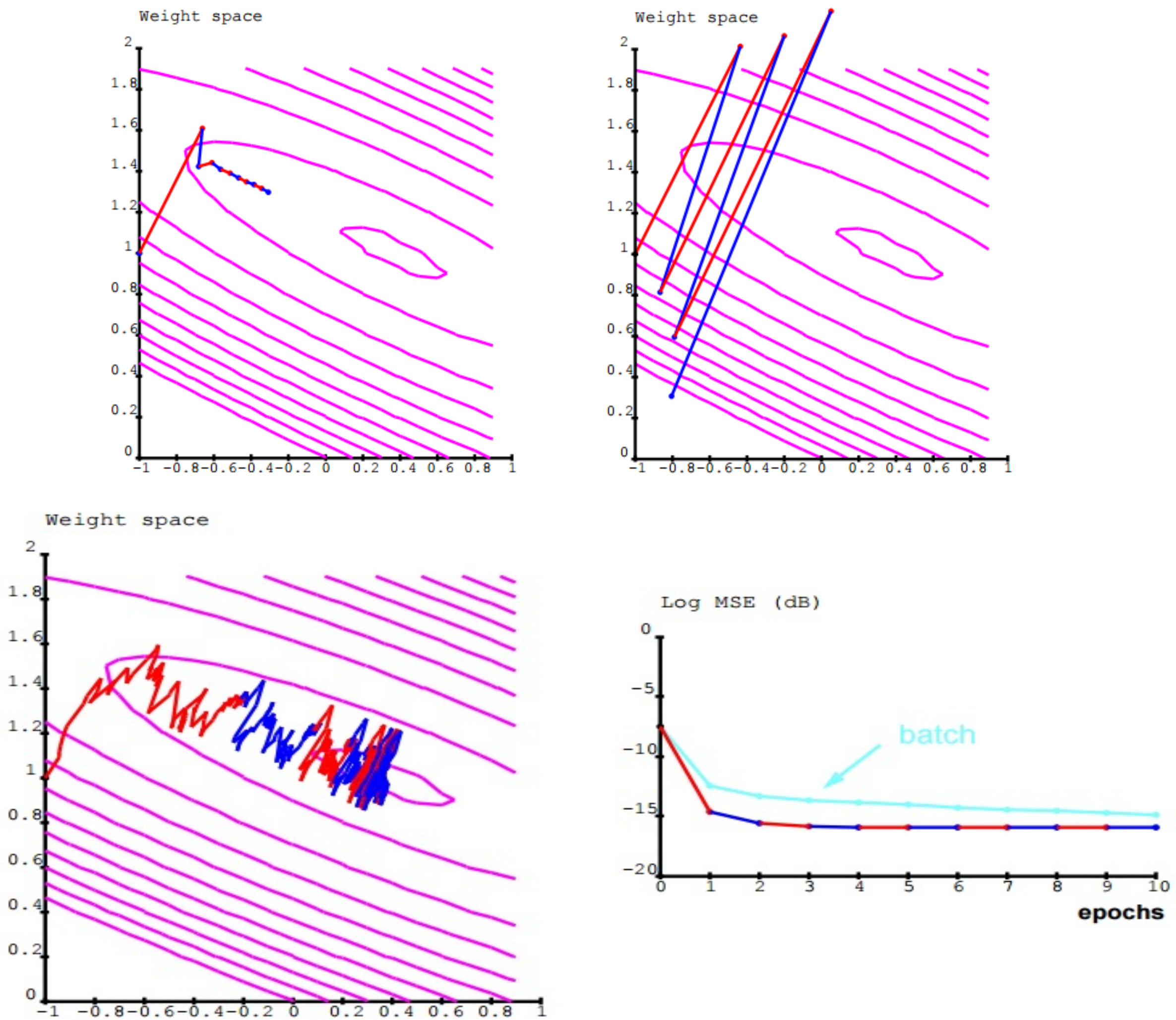
- Limitations
  - Back Propagation barely changes lower-layer parameters (Vanishing Gradient)
  - Therefore, Deep Networks cannot be fully (effectively) trained with Back Propagation



- Breakthrough
  - Deep Belief Networks (Unsupervised Pre-training)
  - Convolutional Neural Networks (Reducing Redundant Parameters)
  - Rectified Linear Unit (Constant Gradient Propagation)

Solutions

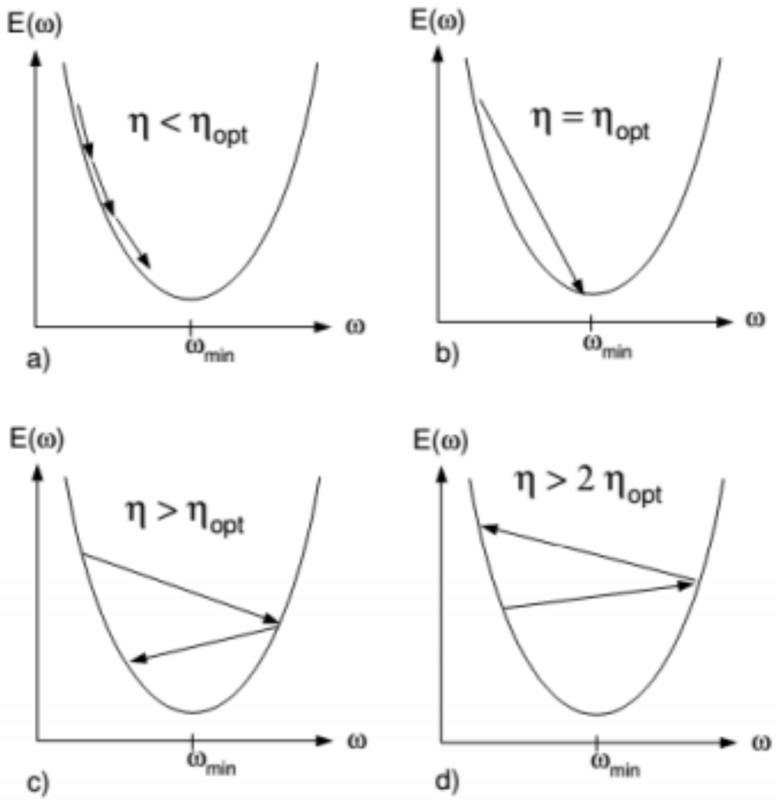
- Stochastic Gradient Descent

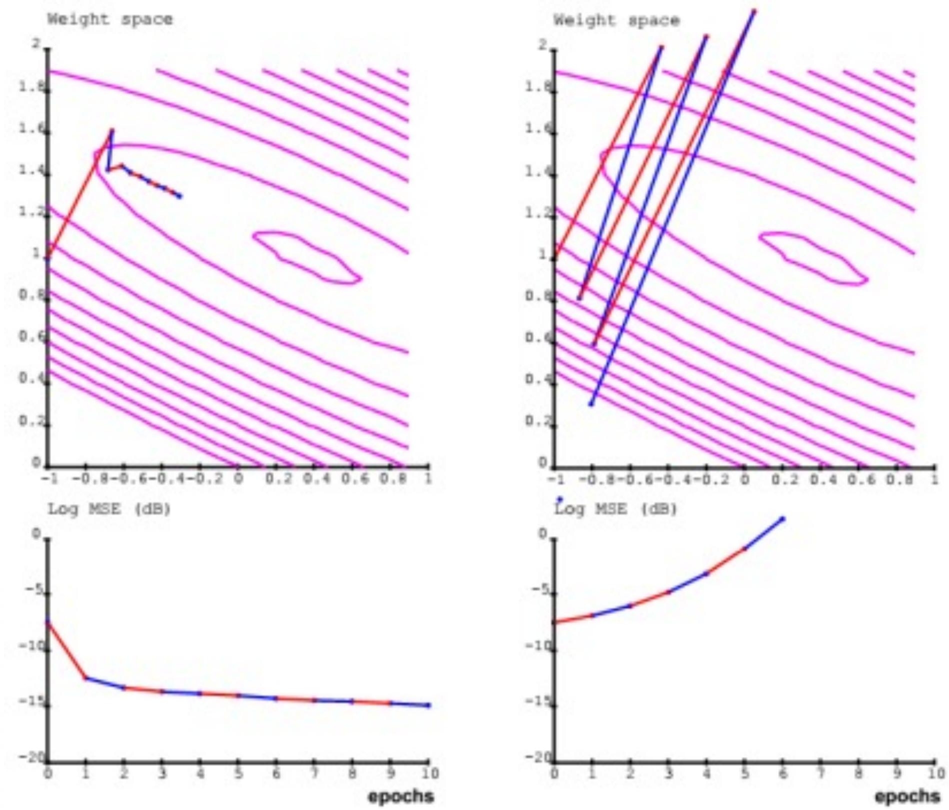


http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

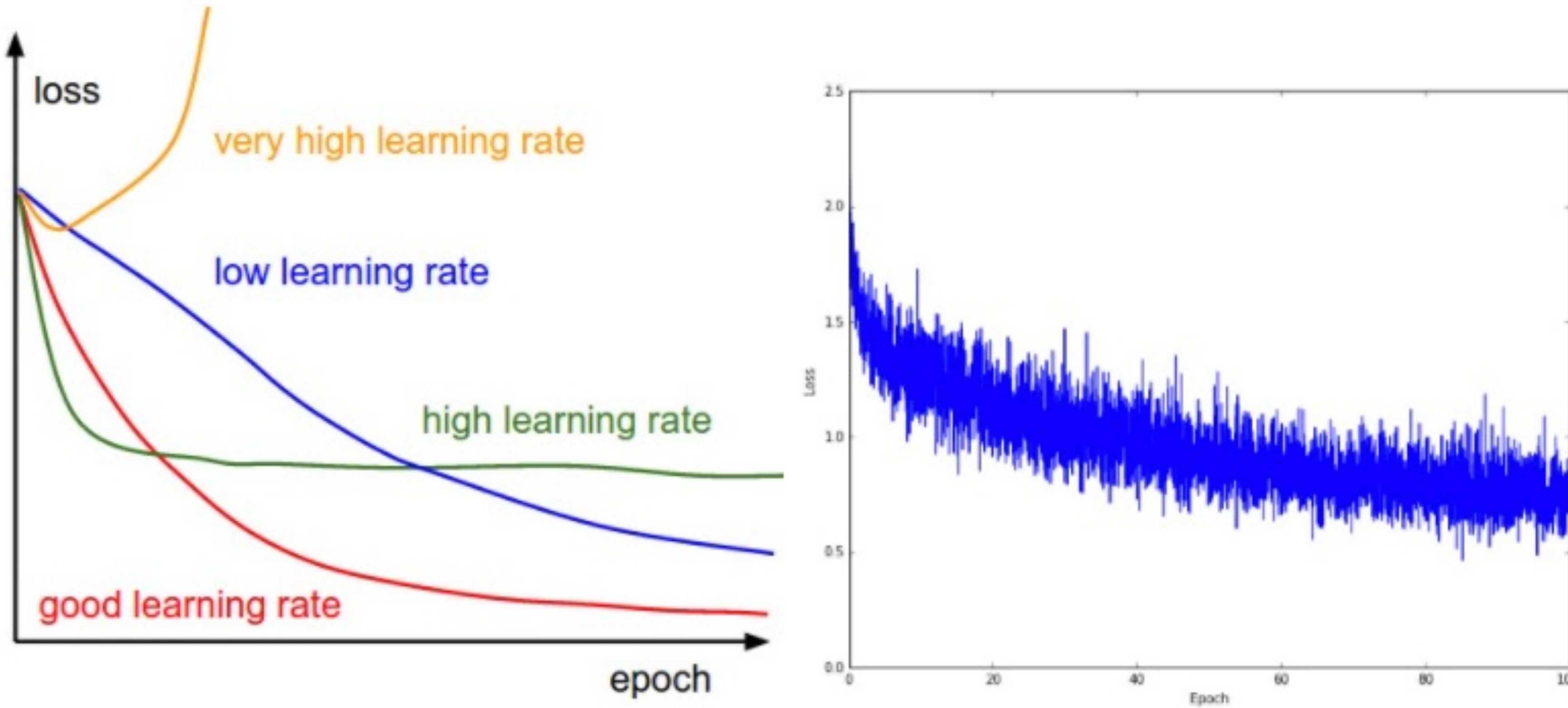
Solutions

- Learning Rate Adaptation





http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
http://courses.cs.tau.ac.il/Caffe_workshop/Bootcamp/pdf_lectures/Lecture%204%20CNN%20-%20optimization.pdf



Left: A cartoon depicting the effects of different learning rates. With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape. Right: An example of a typical loss function over time, while training a small network on CIFAR-10 dataset. This loss function looks reasonable (it might indicate a slightly too small learning rate based on its speed of decay, but it's hard to say), and also indicates that the batch size might be a little too low (since the cost is a little too noisy).

Solutions

- State-of-the-art optimization techniques on NN