The National University of Computer and Emerging Sciences

Introduction to Neural Networks

Machine Learning for Data Science

Dr. Akhtar Jamil

Department of Computer Science

Goals

- Review of Previous Lecture
- Today's Lecture
  - Neural Network and Backpropagation Desirable Features

Previous Lecture

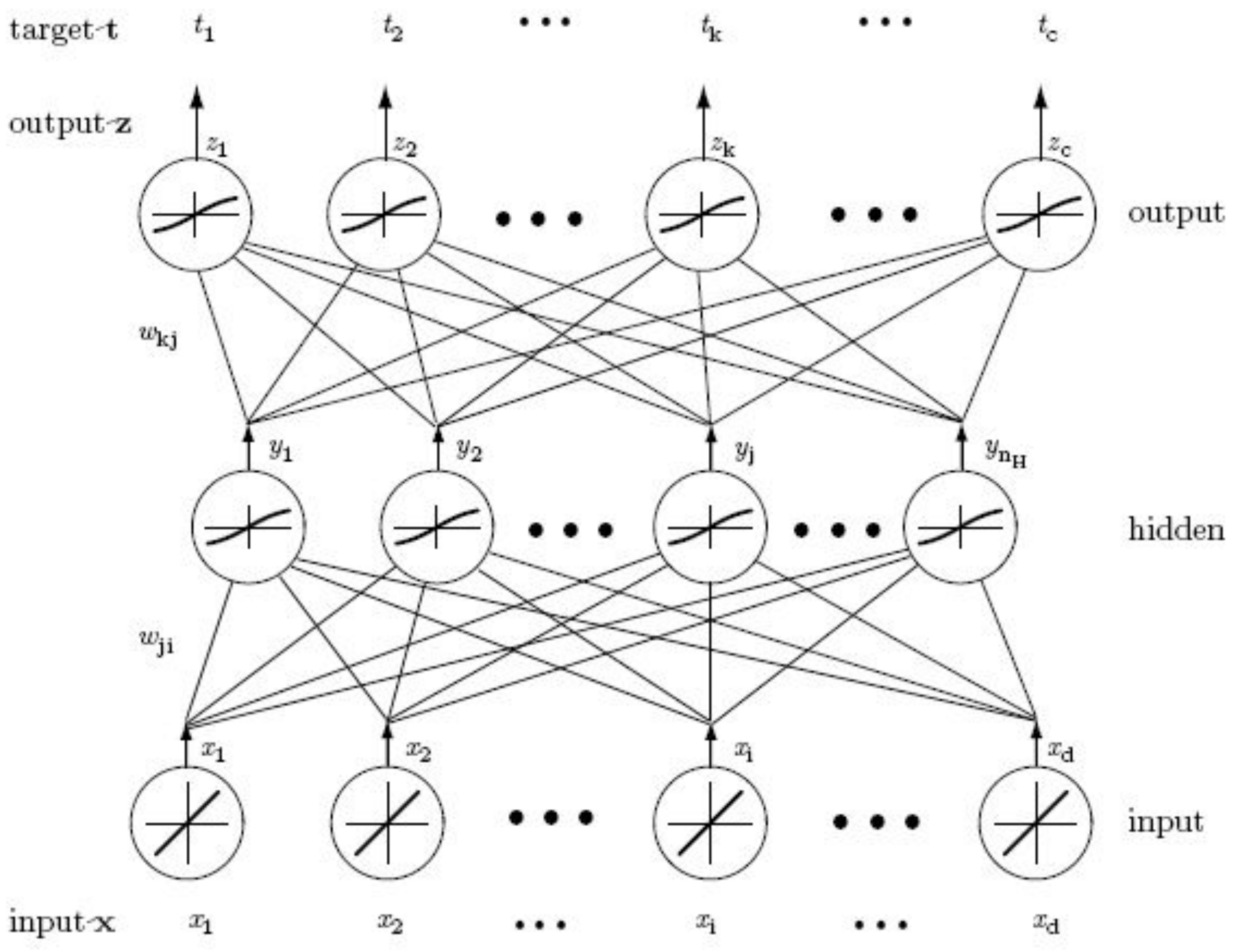
Neural Network

- A popular method for the training of multilayer perceptron is the back-propagation algorithm (Generalized Delta Rule)
- ***Forward phase***:
  - The input data is propagated through the network, layer by layer, until it reaches the output.
- ***Backward phase:***
  - An error is calculated by comparing the output of the network with a desired response.
  - This error is propagated through the network in backward direction.

Neural Network

- **Hidden Neurons**
- The hidden neurons act as *feature detectors*
- The hidden neurons gradually “discover” the main features to understand the training data.
- They perform nonlinear transformations on the input
  - *Feature space.*
- Classes may be more easily separated from each other

Backpropagation algorithm



Backpropagation algorithm

- We can calculate the desired error for the output units as:

$$J(\mathbf{w}) \equiv 1/2 \sum_{k=1}^{c} (t_k - z_k)^2 = 1/2(\mathbf{t} - \mathbf{z})^2$$

- The backpropagation learning rule is based on gradient descent.
- The weights are initialized with random values, and are changed in a direction that will reduce the error:

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

- where η is the *learning rate*

Backpropagation algorithm

- The weight vector will be updated in iterative manner as:

$$w(m+1) = w(m) + \Delta w(m)$$

$$w(m+1) = w(m) - \eta \frac{\partial J}{\partial w}$$

- The weights throughout the network can be updated using.

$$\Delta w_{mn} = -\eta \frac{\partial J}{\partial w_{mn}}$$

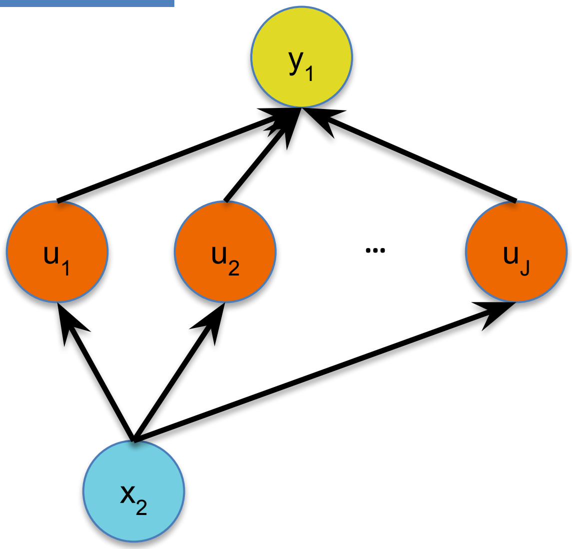
- Now the challenge is to update the weights
  - Some are not explicitly dependent on incoming weights

Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Backpropagation algorithm

- Consider first the hidden-to-output weights, w_{jk}.
- we must use the chain rule for differentiation:

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

- *Sensitivity* of output unit *k* is defined to be

$$\delta_k \equiv -\partial J / \partial net_k$$

Backpropagation algorithm

- Differentiate the cost function with unit's net activation to find how the overall error changes

$$J(\mathbf{w}) \equiv 1/2 \sum_{k=1}^{c} (t_k - z_k)^2$$

$$\delta_k \equiv -\partial J / \partial net_k = -\frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- From the equation

$$\frac{\partial net_k}{\partial w_{kj}} = y_i$$

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0}$$

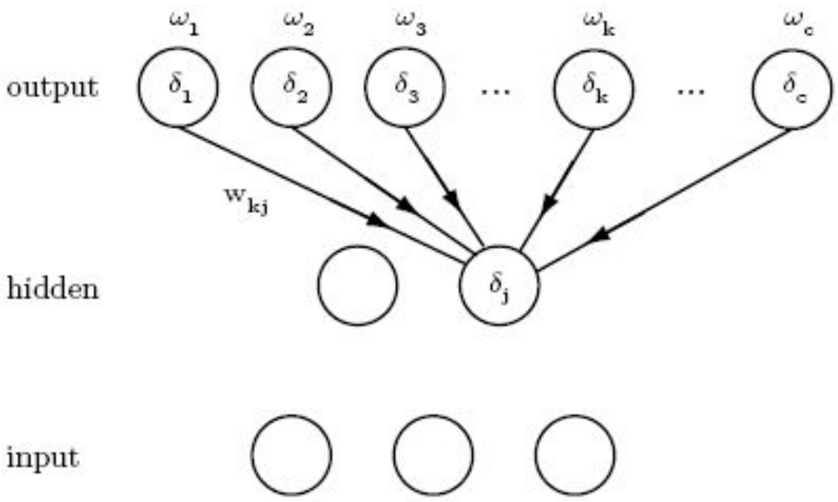
Backpropagation algorithm

- The learning rule for the input-to-hidden units can be calculated as:

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

Backpropagation algorithm

- The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units.
- The output unit sensitivities are thus propagated "back" to the hidden units.



Today’s Lecture

Stochastic backpropagation

Algorithm 1 (Stochastic backpropagation)

begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, m \leftarrow 0$

    do $m \leftarrow m + 1$

        $\mathbf{x}^m \leftarrow$ randomly chosen pattern

        $w_{ij} \leftarrow w_{ij} + \eta \delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta \delta_k y_j$

    until $\nabla J(\mathbf{w}) < \theta$

return $\mathbf{w}$

end

Batch backpropagation

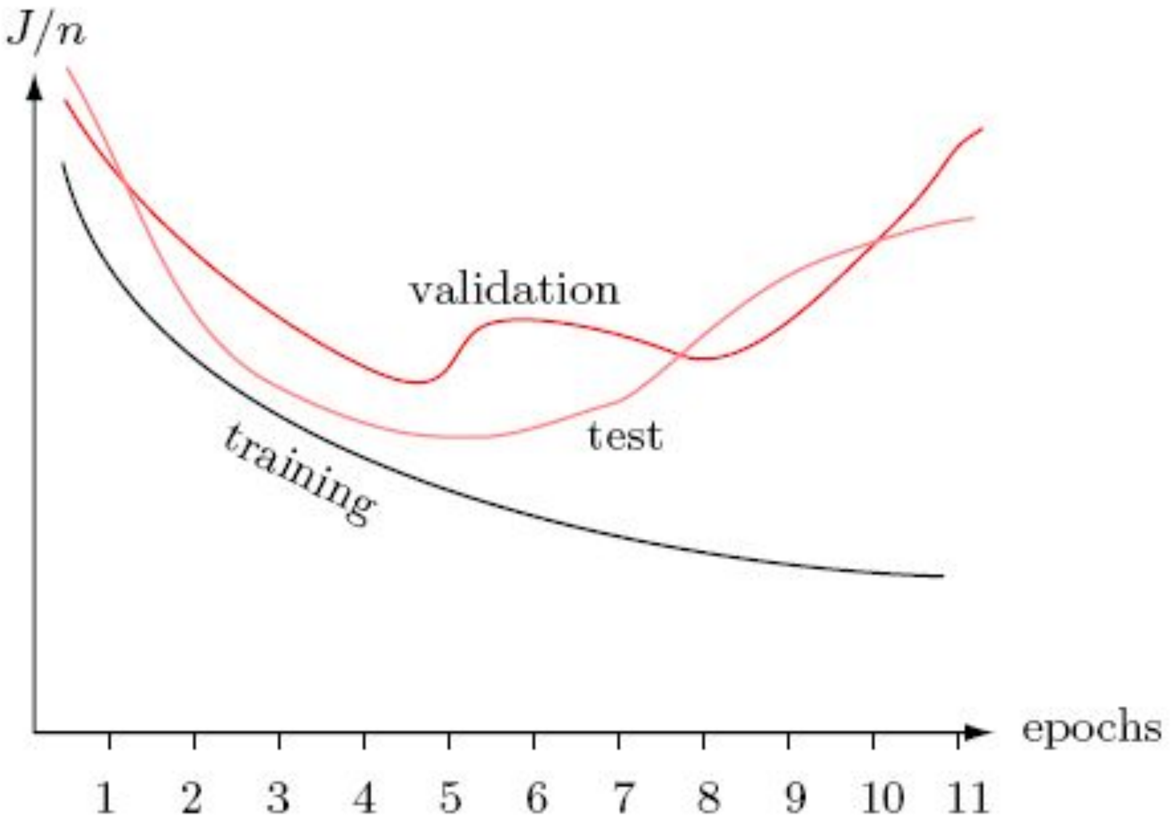
Algorithm 2 (Batch backpropagation)

begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$
    do $r \leftarrow r + 1$ (increment epoch)
        $m \leftarrow 0;\ \Delta w_{ij} \leftarrow 0;\ \Delta w_{jk} \leftarrow 0$
        do $m \leftarrow m + 1$
            $\mathbf{x}^m \leftarrow$ select pattern
            $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i;\quad \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$
        until $m = n$
        $w_{ij} \leftarrow w_{ij} + \Delta w_{ij};\quad w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$
    until $\nabla J(\mathbf{w}) < \theta$
return $\mathbf{w}$
end

Batch backpropagation

- *Stopping criterion*:
  - It is important to stop learning when optimal values of weight are obtained
- Two possibilities:
  - Define Maximum number of epochs
  - Reach a minimum error threshold
- Epochs
  - The number of times the full training set is fed to the model
  - One epoch means model has seen the full training set onces.

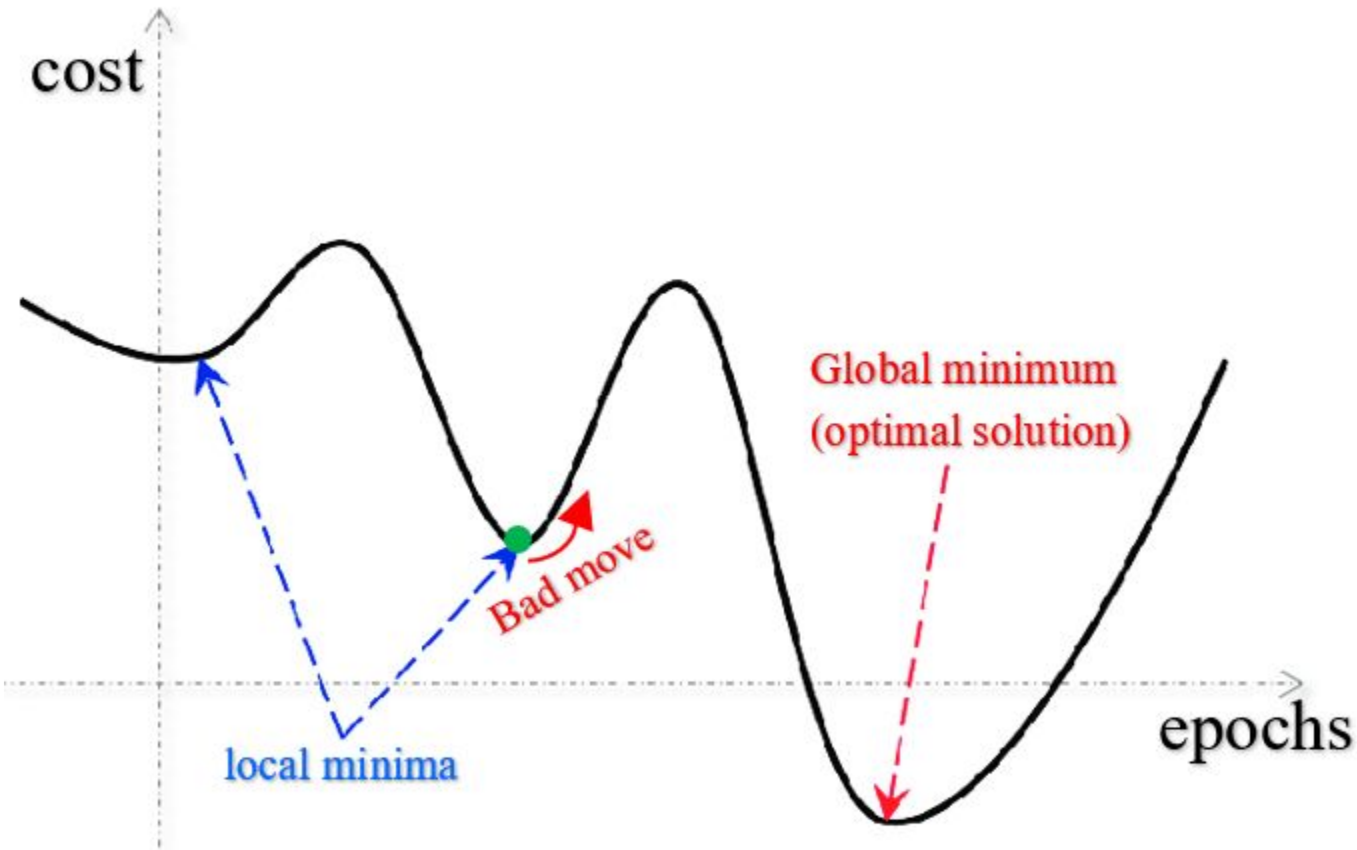
Batch backpropagation



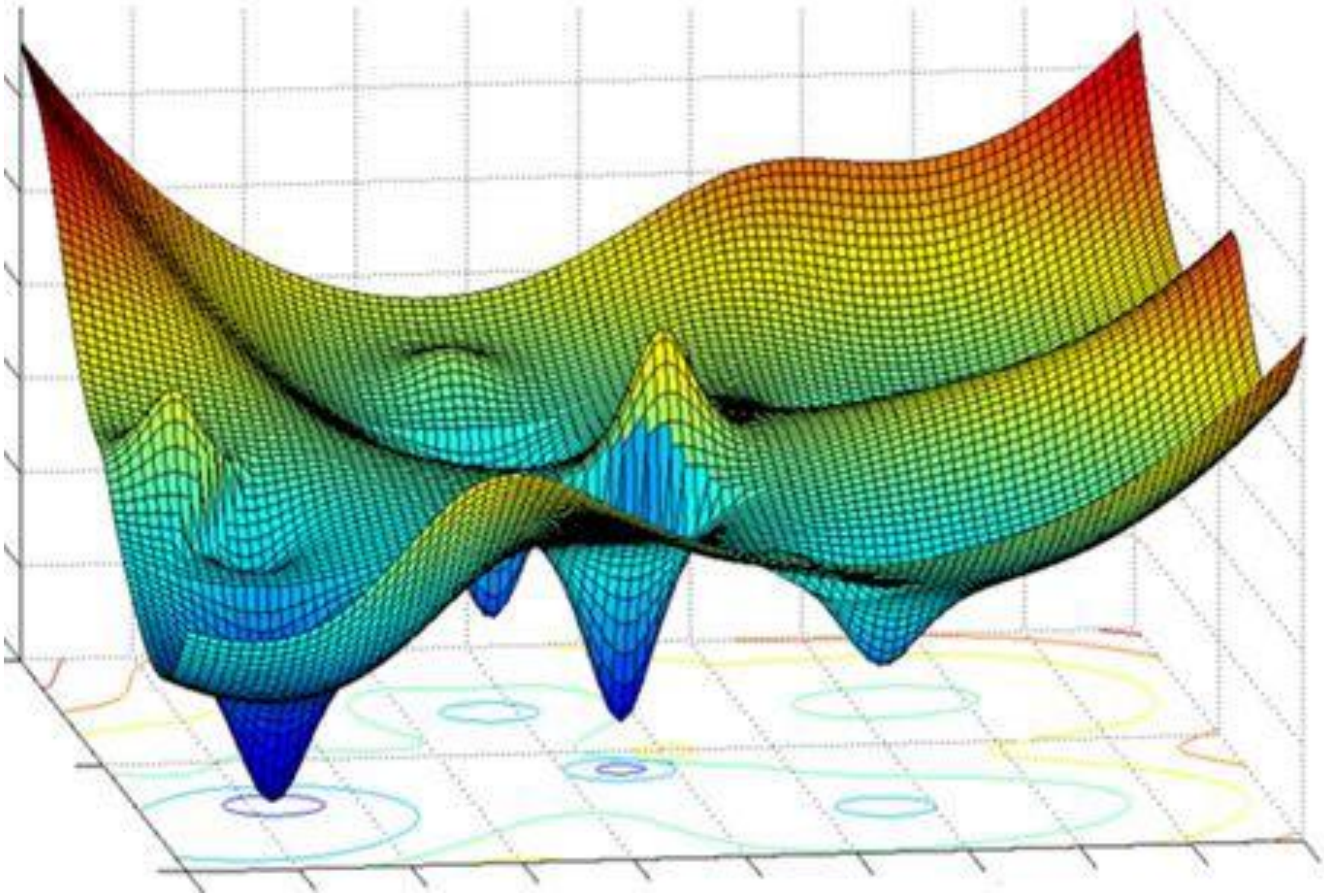
Error surfaces

- Generally, a convex error surface is desirable
- Local minima: if many local minima are present in the error surface, then it is unlikely that the network will find the global minimum.
- Another issue is the presence of plateaus — regions where the error varies only slightly as a function of weights.
  - Training will be slow.

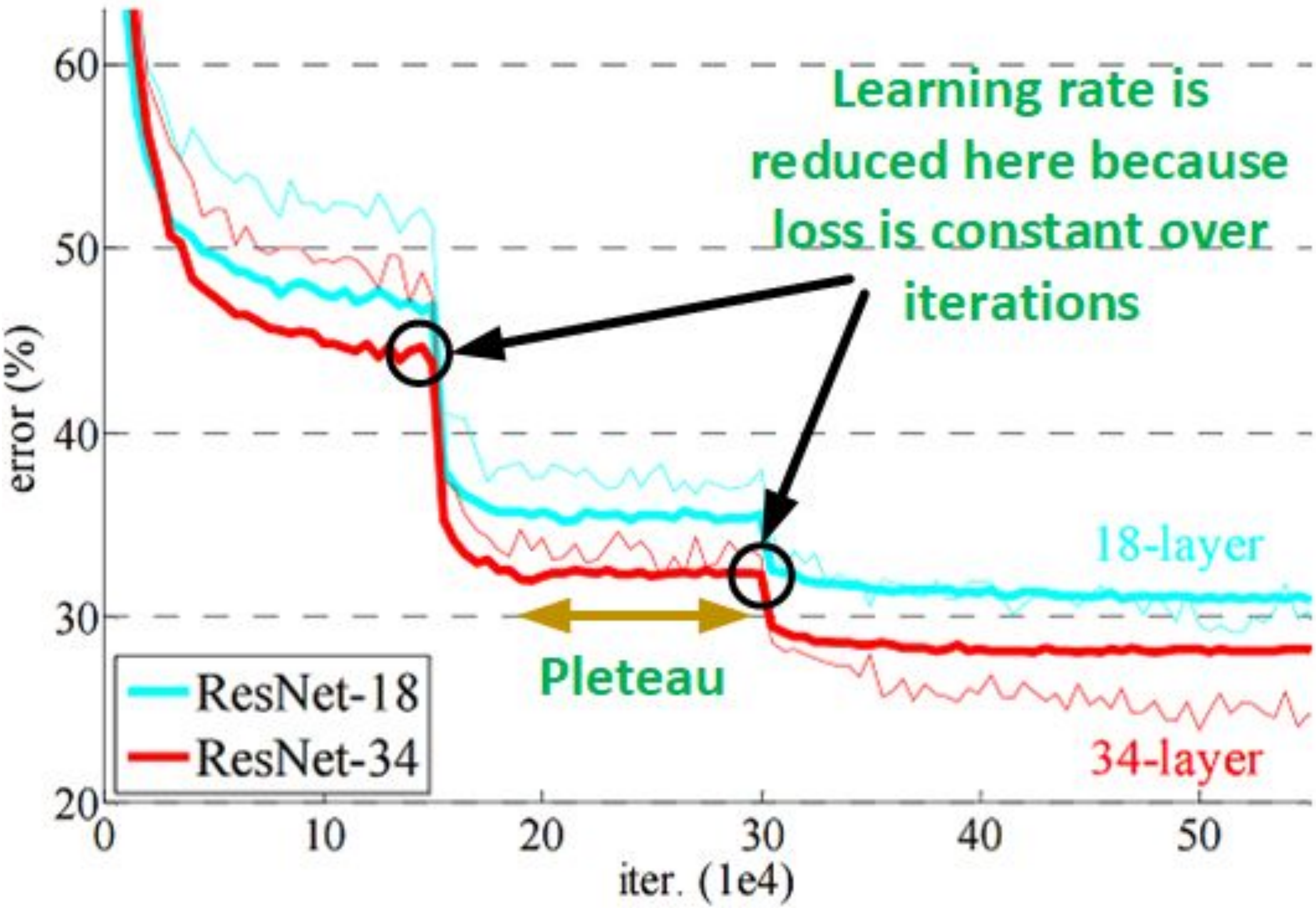
Error surfaces



Error surfaces



Error surfaces



One Hot Encoding

- Machine learning models expect numeric data for training
  - e.g. preprocess text to some numeric representation
  - Image pixels are numeric, can extract more features
  - Sensor data can be represented in numeric form
- In case of classification, the labels should be integers.
- Instead of integers, one-hot encoding is more useful for training some neural networks

One Hot Encoding

Index	Animal
0	Dog
1	Cat
2	Sheep
3	Horse
4	Lion

One-Hot code →

Index	Dog	Cat	Sheep	Lion	Horse
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0

Outputs as probabilities

- One approach is to choose the output neuron's nonlinearity to be exponential rather than sigmoidal:

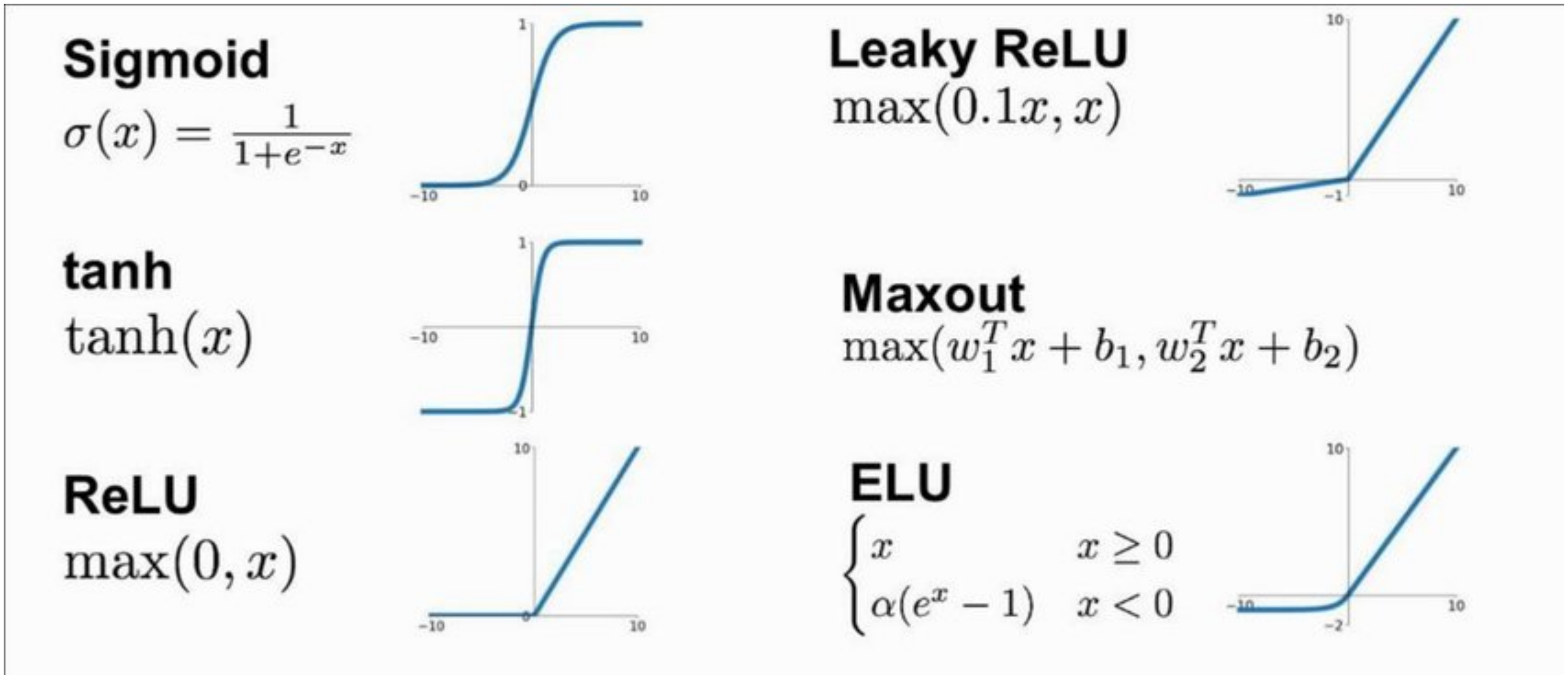
$$z_k = \frac{e^{net_k}}{\sum_{m=1}^{c} e^{net_m}}$$

- Train the model with one hot encoding
- This is the *softmax* method — a smoothed or softmax continuous version of a *winner-take-all* nonlinearity.
- Winner-take- all, in which the maximum output is transformed to 1.0, and all others reduced to 0.0.

Activation Functions

- Backpropagation can work with any activation function
- The activation function should at least meet the criteria of
  - Differentiability, continuity, nonlinearity.
- The *sigmoid function* possesses the desirable properties of the activation function

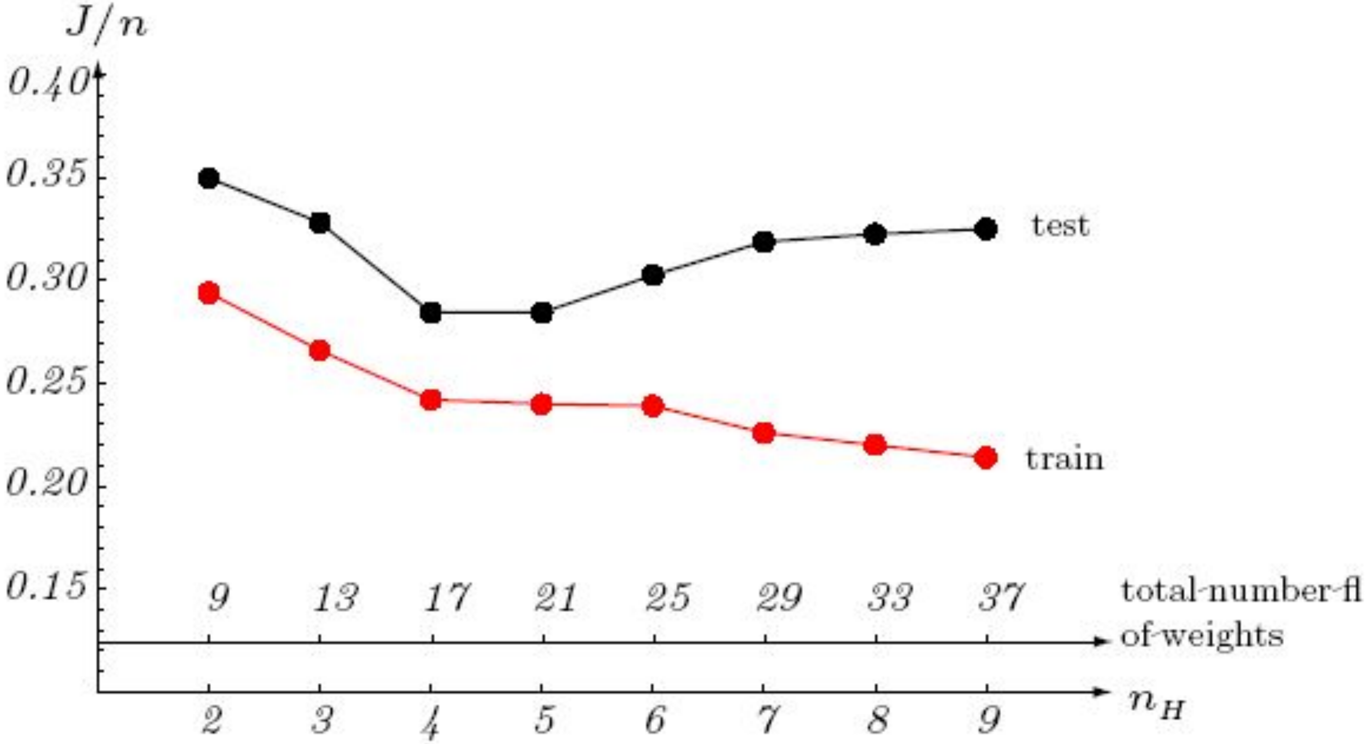
Activation Functions



Number of hidden units

- The number of hidden neurons (n_H) is crucial for extracting powerful features
  - The complexity of the decision boundary.
- For linearly separable data few hidden units are needed
- For complex and nonlinear data, more hidden neurons are needed.
- No recommended number of hidden units
- We should not have more weights than the total number of training samples

Number of hidden units



Initializing weights

- The weight values should be initialized to have fast and uniform learning.
  - All weights reach their optimum values at about the same time.
- If some weights converge significantly earlier than others (non-uniform learning) then the network may not perform well throughout the full range of inputs or for each class
- Example: when class C_i is learned well before C_j .
- A possible solution is to use data standardization

Initializing weights

- Setting weights to zero is not a good idea
- Choose weights randomly from a *single* distribution to help insure uniform learning.
- For standardized input with d features, we can randomly initialize the weight between input – hidden layers as:

$$-1/\sqrt{d} < w_{ji} < +1/\sqrt{d}$$

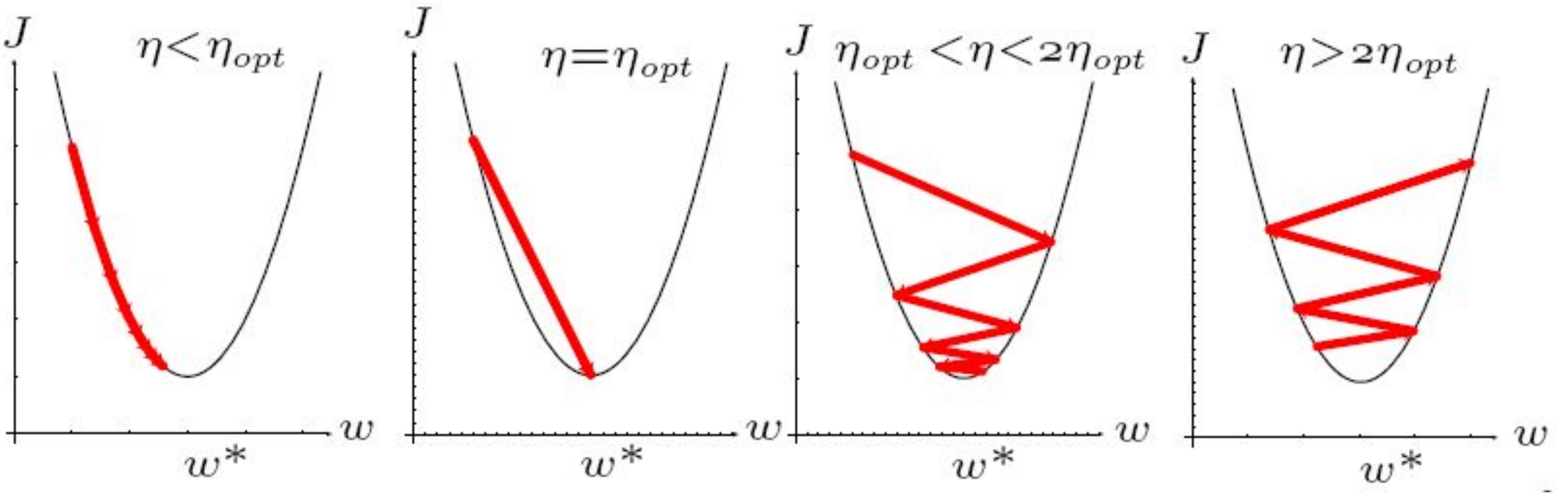
- Similarly for hidden-output weights can be randomly initialized as:

$$-1/\sqrt{n_H} < w_{kj} < +1/\sqrt{n_H}$$

Learning rates

- The optimal learning rate is the one which leads to the minimum local error in one
- Generally, learning step of $\eta = 0.1$ is good for starting
  - Lower it if the cost function diverges
  - Raised it the cost function is too slow to converge.
- How to identify slow convergence?
  - Loss is gradually reducing but take too many epochs

Learning rates



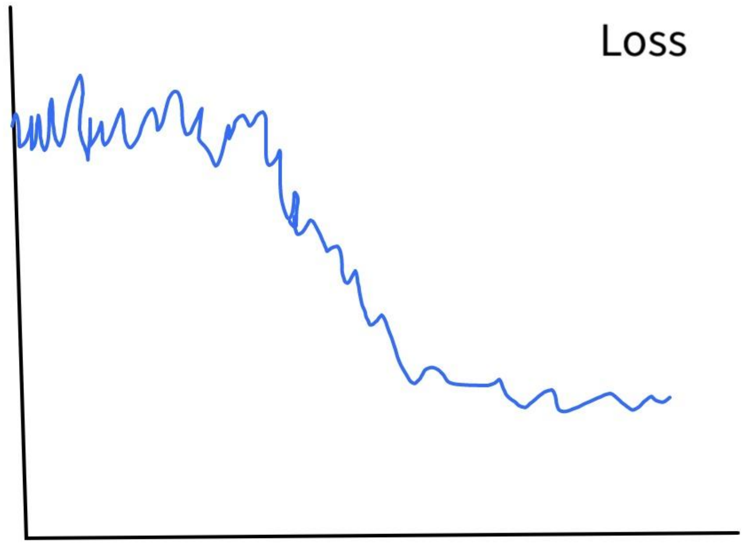
Momentum

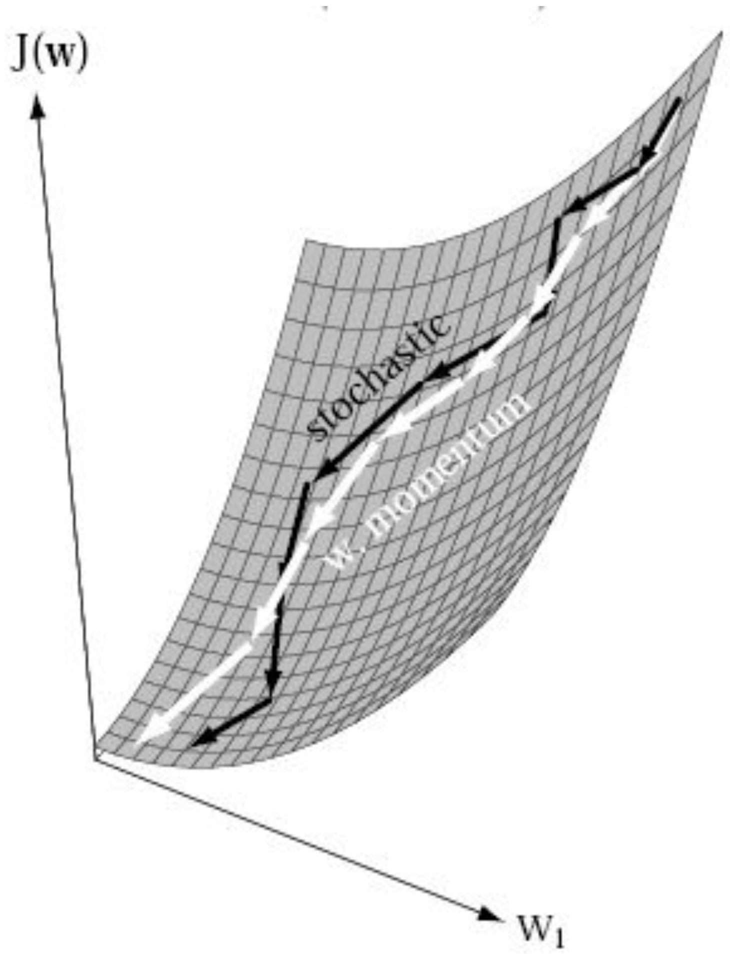
- Error surfaces often have plateaus regions
  - The slope $dJ(\mathbf{w})/d\mathbf{w}$ is very small
  - Too many weights.
- Momentum- allows the network to learn weights more quickly when plateaus in the error surface exist.
- The learning rule in backpropagation can be updated as:

$$\mathbf{w}(m+1) = \underbrace{\mathbf{w}(m) + \Delta\mathbf{w}(m)}_{\substack{\text{gradient} \\ \text{descent}}} + \underbrace{\alpha\Delta\mathbf{w}(m-1)}_{\text{momentum}}$$

- α must be less than 1.0 for stability, typical value $\alpha = 0.9$.
- Affect is like averaging the stochastic variations in weight updates
- Can also help overcome local minima

Momentum





Exploding Gradients

- Exploding gradients are a problem where large error gradients accumulate
- Result in very large weight updates during training.
- Makes model unstable and unable to learn from your training data.
- Cause poor predication results

Weight decay

- Weight decay is a regularization technique
- Simplifying a network and avoiding overfitting is to impose a heuristic that the weights should be small.
- To prevent overfitting.
- To keep the weights small and avoid exploding gradient.
- This will help keep the weights as small as possible, preventing the weights to grow out of control, and thus avoid exploding gradient.

Weight decay

- Small weights favor models
- Popular due to its simplicity.
- After each weight update every w eight is simply "decayed" or shrunk according to:

$$w^{\text{new}} = w^{\text{old}}(1-\epsilon)$$

- where $0 < \epsilon < 1$

$$J_{ef} = J(\mathbf{w}) + \frac{2\epsilon}{\eta}\mathbf{w}^t\mathbf{w}$$

- Achieves a balance between error and overall weight.

Loss functions

- The squared error criterion is the most common training criterion
- It is simple to compute, non-negative,
  - Log loss
- another criterion function is based on the *Minkowski* Minkowski *error*:

$$J(\mathbf{w})_{ce} = \sum_{m=1}^{n} \sum_{k=1}^{c} t_{mk} \ln(t_{mk}/z_{mk})$$

References

- Chapter 4, Neural Networks and Learning Machines, Haykin
- Chapter 5, Pattern Recognition and Machine Learning, Bishop

Thank You ☺