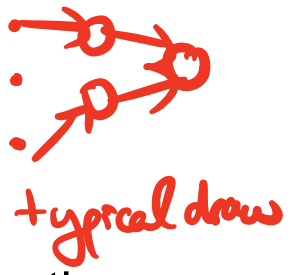
Artificial Neural Networks

MultiLayer Perceptrons & Backpropagation

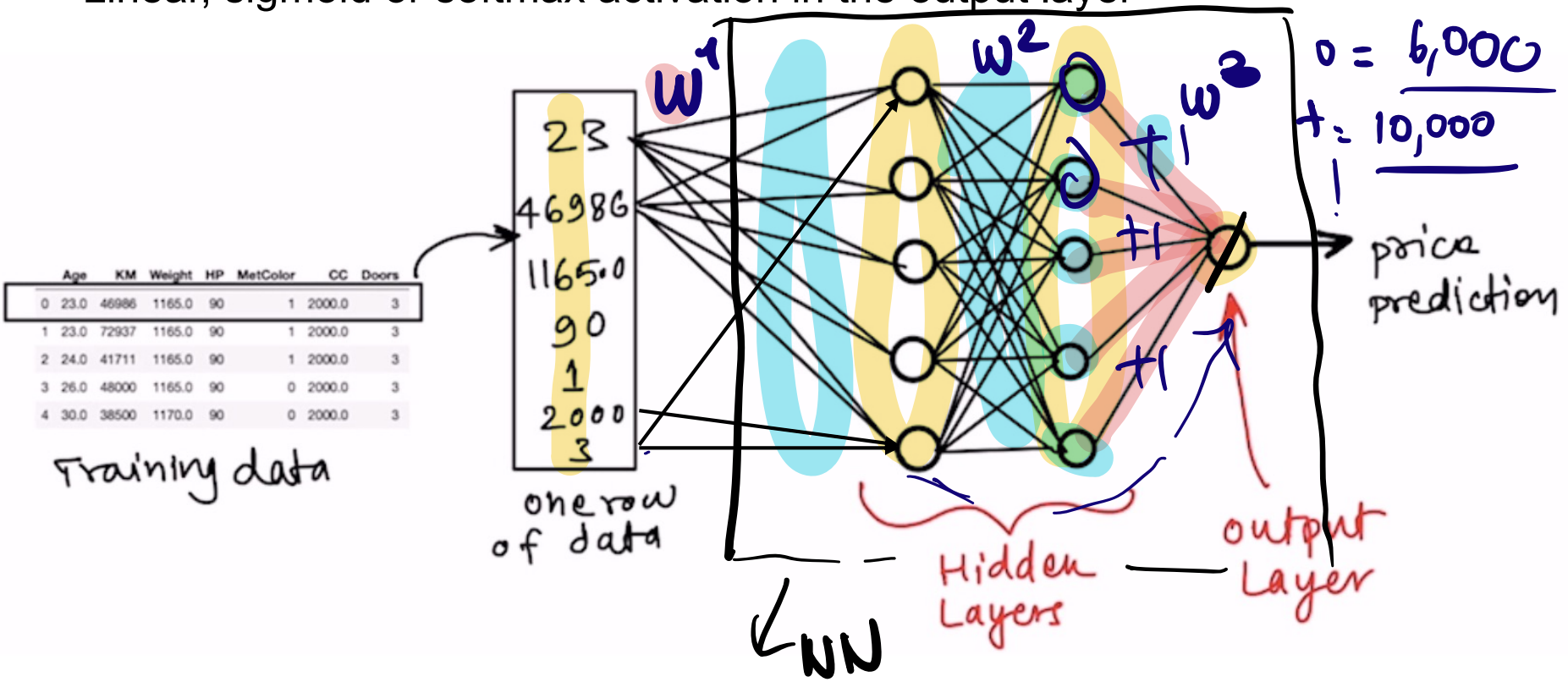
Berrin Yanikoglu

2/2019

Multilayer Perceptron (Feed-forward Neural Networks)



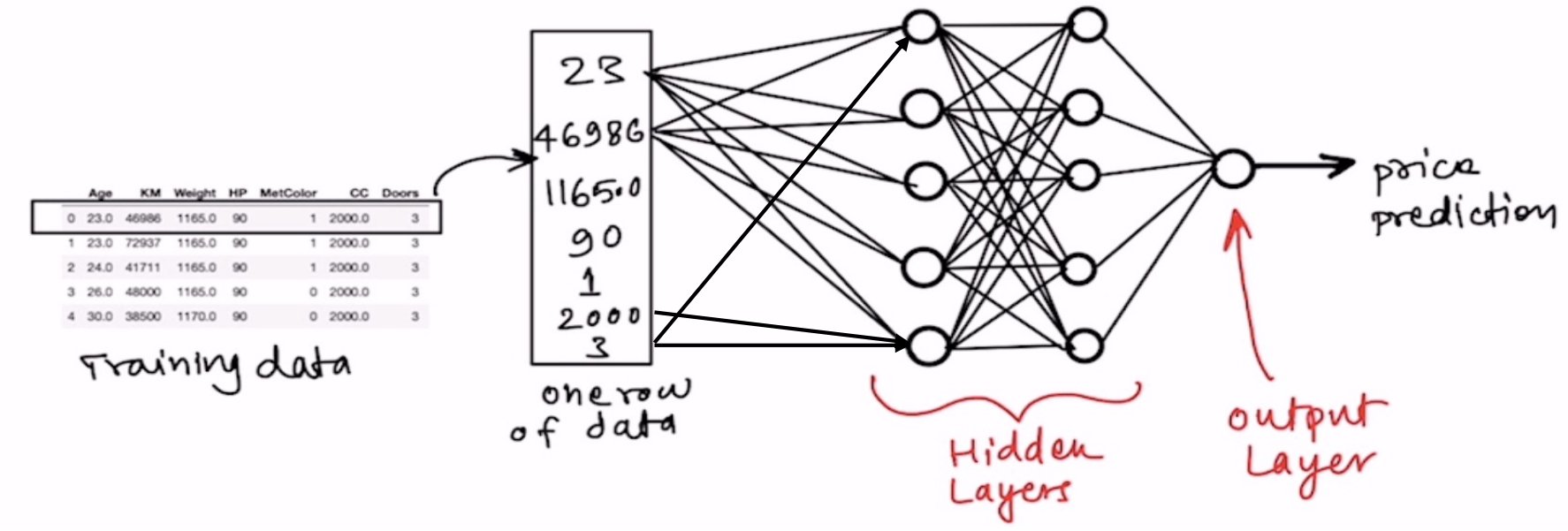
- There may be one or more hidden layer(s) which are called hidden since they are not observed from the outside.
- Activations are passed only from one layer to the next.
- Each layer may have different number of nodes that share the same activation function (sigmoid, tanh, RELU…).
- Linear, sigmoid or softmax activation in the output layer



Neural Networks

Given an input (an image, text, vector, or sequence…), each layer of neurons compute simple functions of their input based on the current parameters (weights) of the network, passing their output as input to next layer.

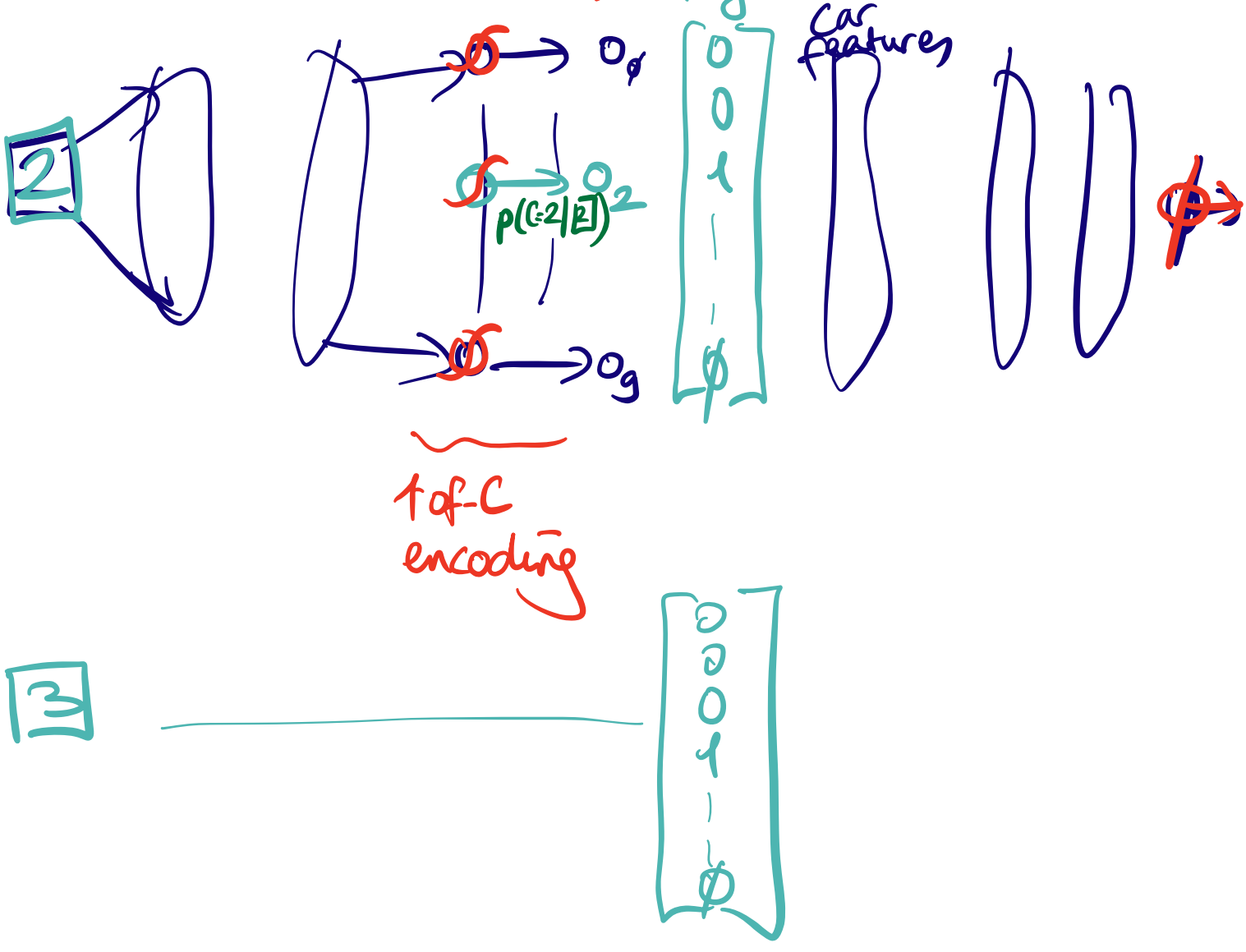
- Forward-pass: Compute neuron activations from the first to last layer
- Measure error/loss between target and what is predicted
- Backpropagation: Modify the network weights so as to reduce the error over the training set



MNIST

10-class (C class classif.)

Regression (car price pred



Shallow vs Deep Networks

Shallow neural networks: Typically 1-2 layers of weights

Deep neural networks: Many layers (often 100s of layers) of weights.

Underlying work since 1980s, but new progress thanks to:

- **Data**
- **Computing power (GPUs)**
- **Theoretical novelties**
  - ReLU
  - Dropout
  - Maxpool
  - BatchNorm
  - …

Image: http://cs231n.stanford.edu/

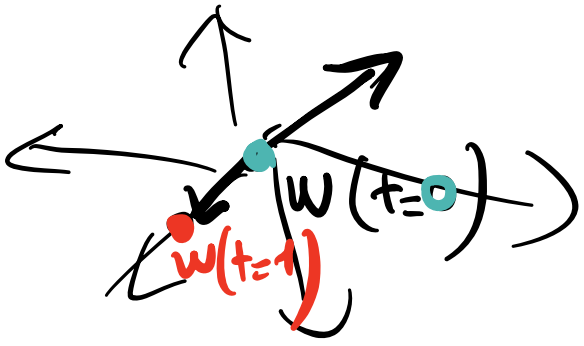
Performance Learning

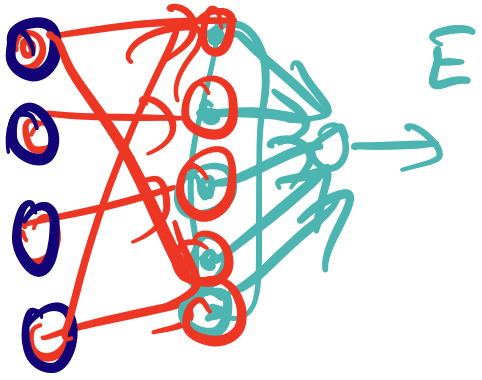
Performance Learning

A learning paradigm where the network adjusts its parameters (weights and biases) so as to optimize its "performance"

- Need to define a performance index (cost function)
  - e.g. mean square error

- Search the parameter space to minimize the performance index with respect to the parameters





Performance Optimization

Iterative minimization techniques is a form of performance learning:

- Define E(.) as the performance index
- Starting with an initial guess w(0), find w(n+1) at each iteration such that

$$E(w(n+1)) < E(w(n))$$

- In particular, **we will see that Gradient Descent** is an iterative (error) minimization technique

Basic Optimization Algorithm

Start with initial guess $\mathbf{w}_0$ and update the guess in each stage moving along the search direction:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{p}_k$$

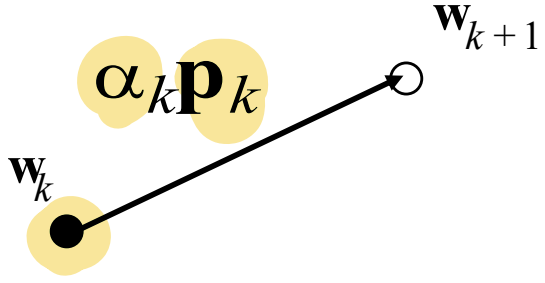
or

$$\Delta \mathbf{w}_k = (\mathbf{w}_{k+1} - \mathbf{w}_k) = \alpha_k \mathbf{p}_k$$

A new state in the search involves deciding on a search direction and the size of the step to take in that direction:

$\mathbf{p}_k$ - Search Direction

α_k - Learning Rate



>> Modifications of these two lead to different optimization algorithms

Gradient Descent (aka Steepest Descent)

Successive adjustments to w are in the direction of the steepest descent (direction opposite to the gradient vector)

$$w(n+1) = w(n) - \eta \nabla E(w(n))$$

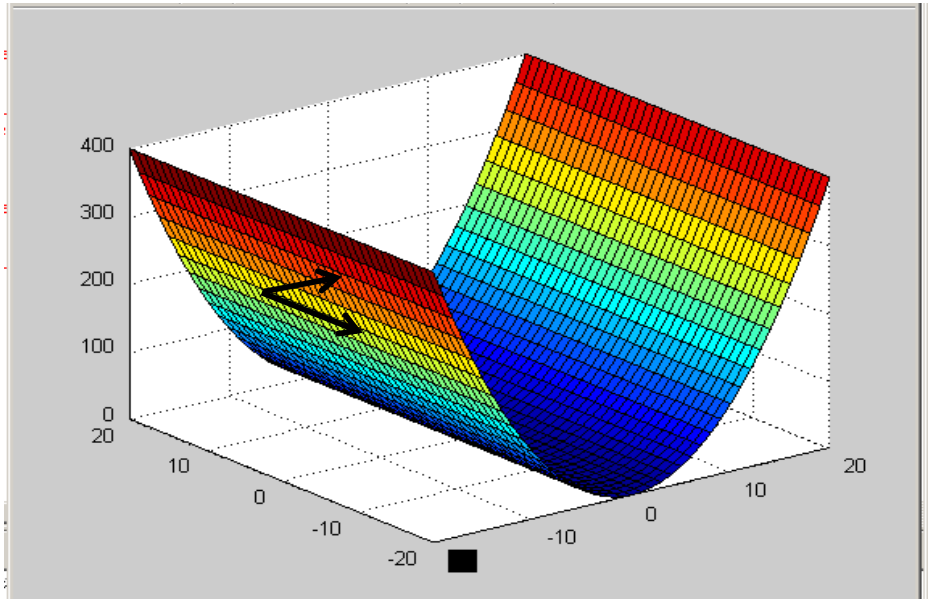
gradient

Gradient Vector

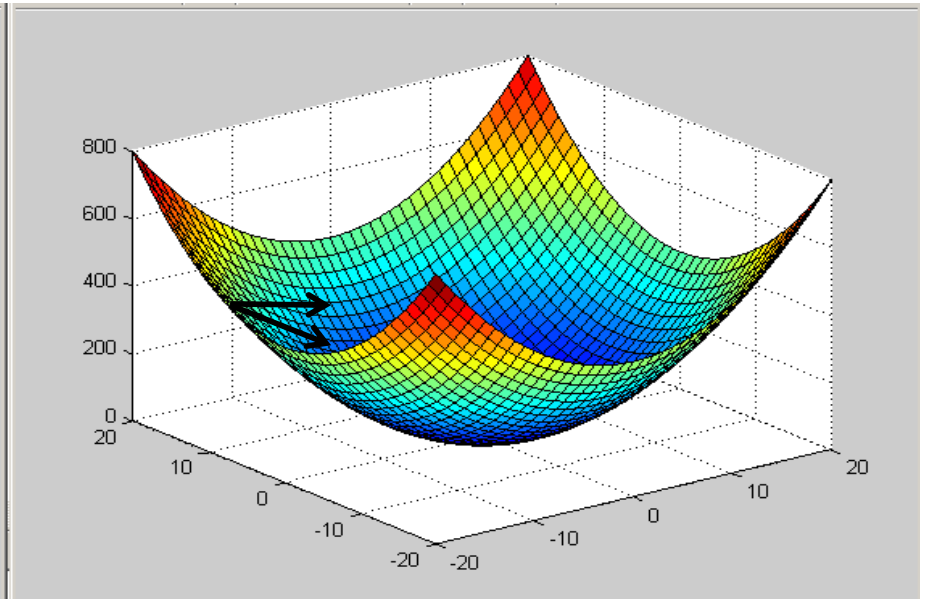
Each dimension of the gradient vector is the vector of partial derivatives of the function E with respect to each of the variables w_i.

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1} E(\mathbf{w}) \\ \frac{\partial}{\partial w_2} E(\mathbf{w}) \\ \vdots \\ \frac{\partial}{\partial w_n} E(\mathbf{w}) \end{bmatrix}$$

Two simple error surfaces (for 2 weights)



a)

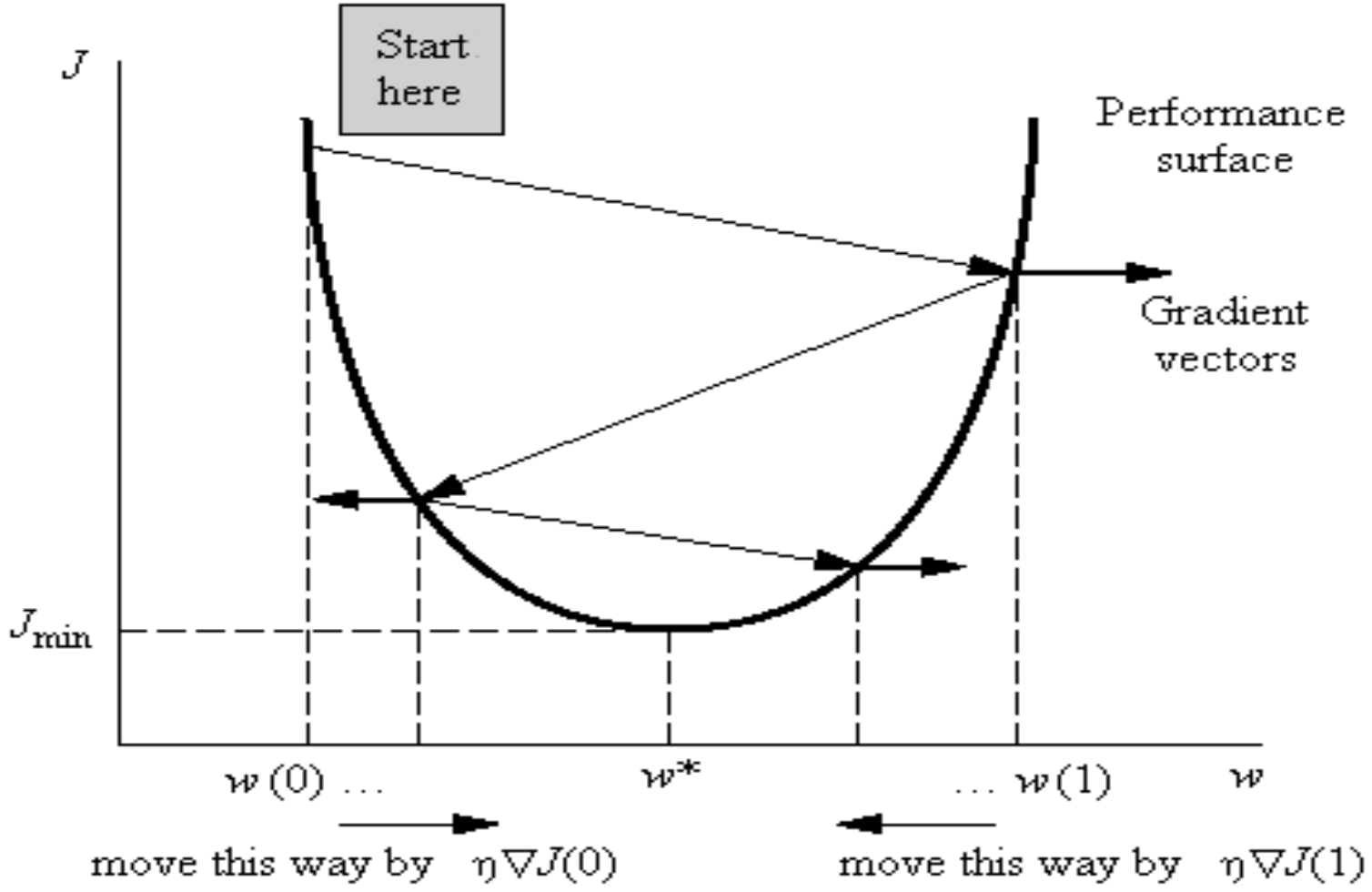


b)

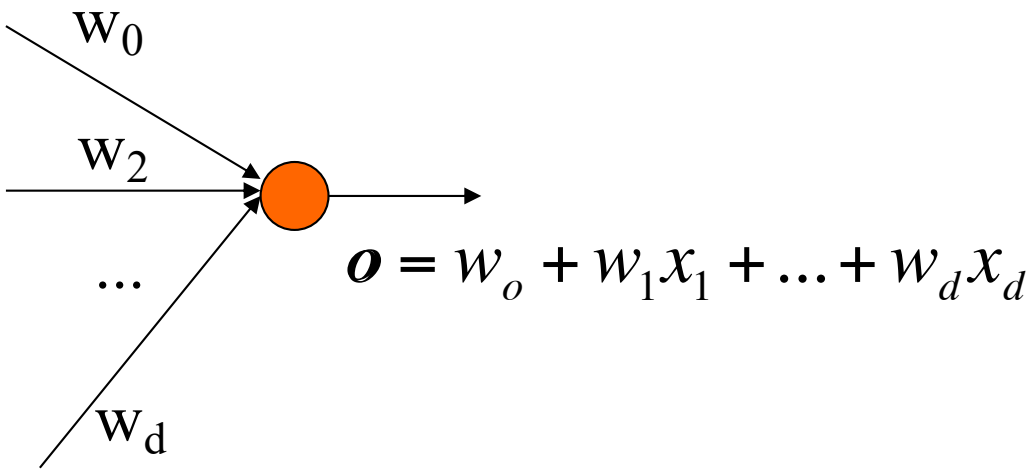
In a) as you move parallel to one of the axis, there is no change in error

In b) moving diagonally, rather than parallel to the axes, brings the biggest change.

Steepest Descent



ADALINE



Let's find/learn the weights w_j to minimize the squared error over the training set with N samples:

$$E(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (t_i - o_i)^2$$

where N is the number of training samples;
t_i is the target for the ith sample and
o_i is the output for the ith sample.

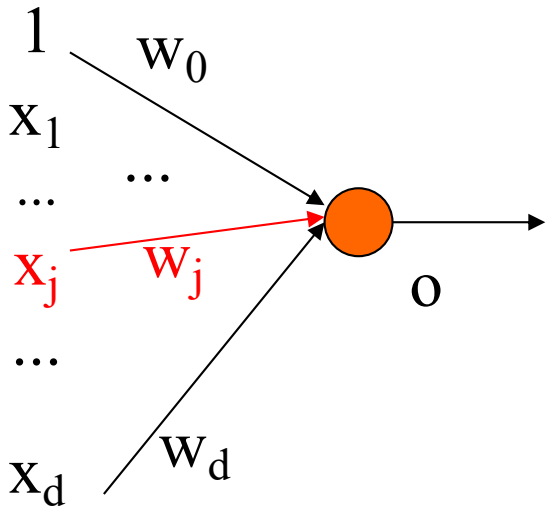
Linear Neuron

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(t_i - o_i)^2$$

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial (t - o_i)^2}{\partial w_j}$$

$$= \frac{1}{2N}\sum_{i=1}^{N} 2(t_i - o_i)(-1)\frac{\partial o_i}{\partial w_j}$$

$$= -\frac{1}{N}\sum_{i=1}^{N}(t_i - o_i)\frac{\partial (\mathbf{w}^T \mathbf{x}_i)}{\partial w_j} = -\frac{1}{N}\sum_{i=1}^{N}(t_i - o_i)x_{ij}$$



Stochastic/Approximate Gradient Descent

Approximate Gradient Descent
(Stochastic Backpropagation)

Normally, in gradient descent, we would need to compute how the error over all input samples (true gradient) changes with respect to a small change in a given weight.

But the common form of the gradient descent algorithm takes one input pattern, **compute the error of the network on that pattern only**, and updates the weights using only that information.

- Notice that the new weight may not be good/better for all patterns, but we expect that if we take a small step, we will average and approximate the true gradient.

Stochastic Approximation to Steepest Descent

Instead of updating every weight until all examples have been observed, **we update on every example**:

$$\nabla w_i \cong \eta\,(t-o)\,x_i$$

Remarks:

- Speeds up learning significantly when data sets are large
  - Use a smaller learning step!
- When there are multiple local minima, stochastic gradient descent may avoid the problem of getting stuck on a local minimum.

stopped here

Gradient Descent Backpropagation Algorithm

Derivation for General Activation Functions Networks