Brain-Inspired Computing

Can the Brain do back-propagation?

Konstantinos Michmizos
Computational Brain Lab

Lecture 8
Friday, December 1st, 2017

Toy Example – Iterative Approach

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions of each.
- The cashier only tells you the total price of the meal
  - After several days, you should be able to figure out the price of each portion.
- The iterative approach: Start with random guesses for the prices and then adjust them to get a better fit to the observed prices of whole meals.

Solving the equations iteratively

- Each meal price gives a linear constraint on the prices of the portions:

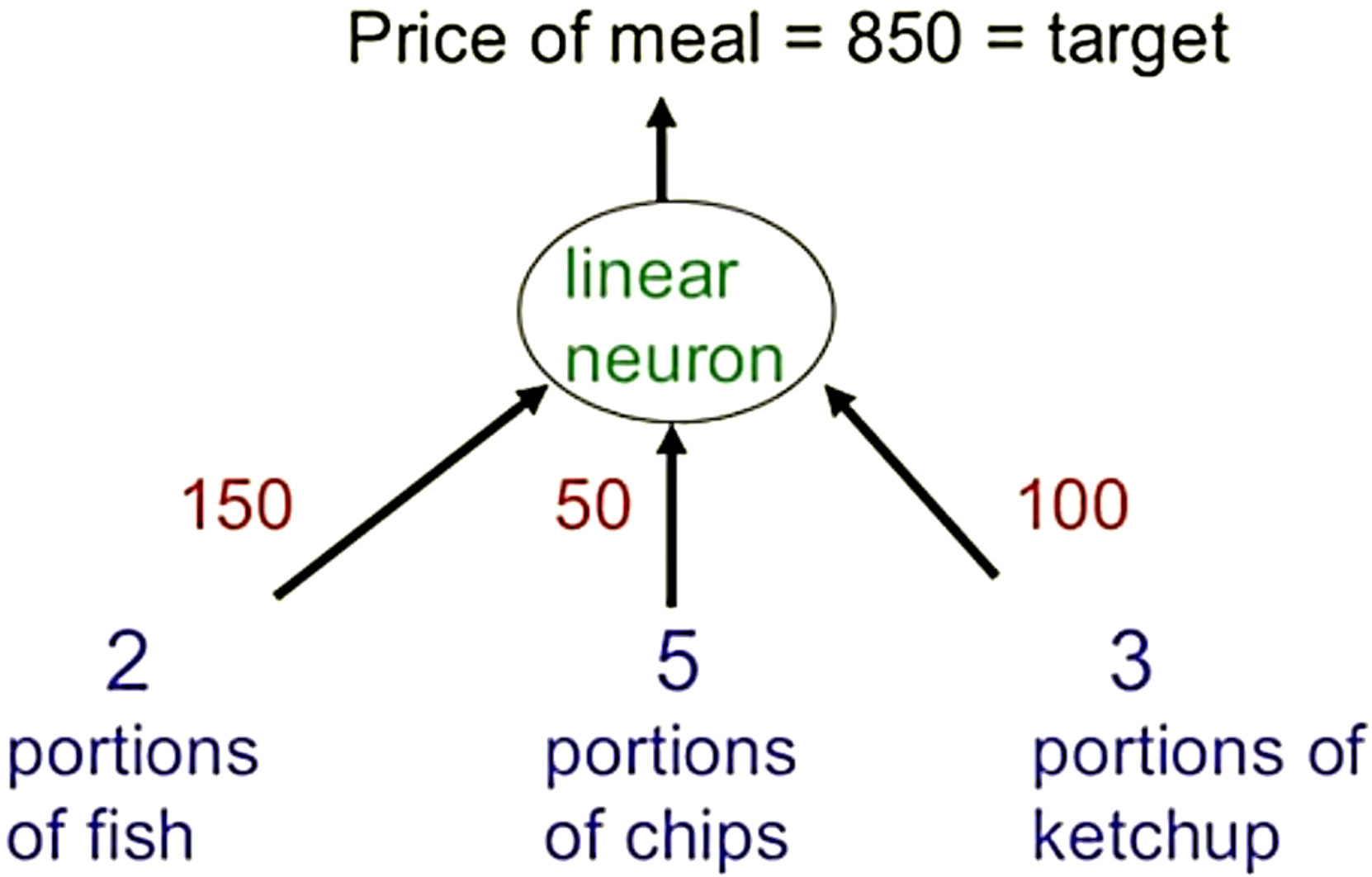
$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{ketchup} w_{ketchup}$$

- The prices of the portions are like the weights in of a linear neuron.

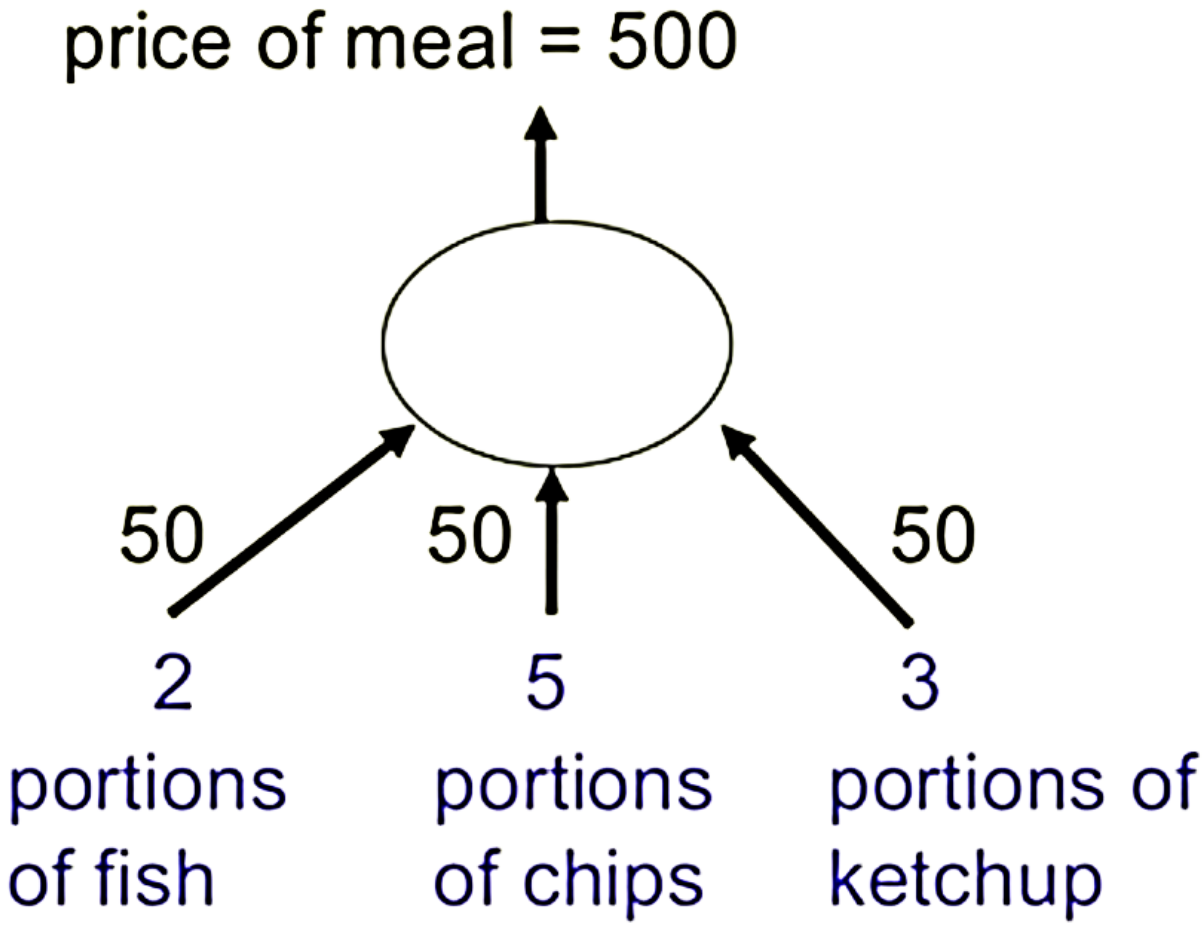
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

- We will start with guesses for the weights and then adjust the guesses slightly to give a better fit to the prices given by the cashier.

The true weights used by the cashier



A model of the cashier with arbitrary initial weights



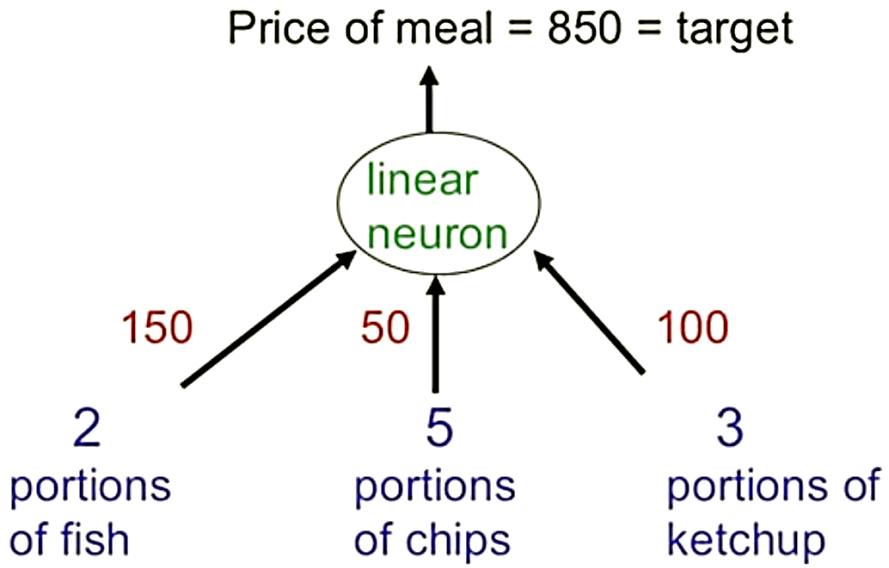
No guarantee that the individual weights will become better – what is guaranteed is that the overall sum will become better

Learning by Perturbing weights

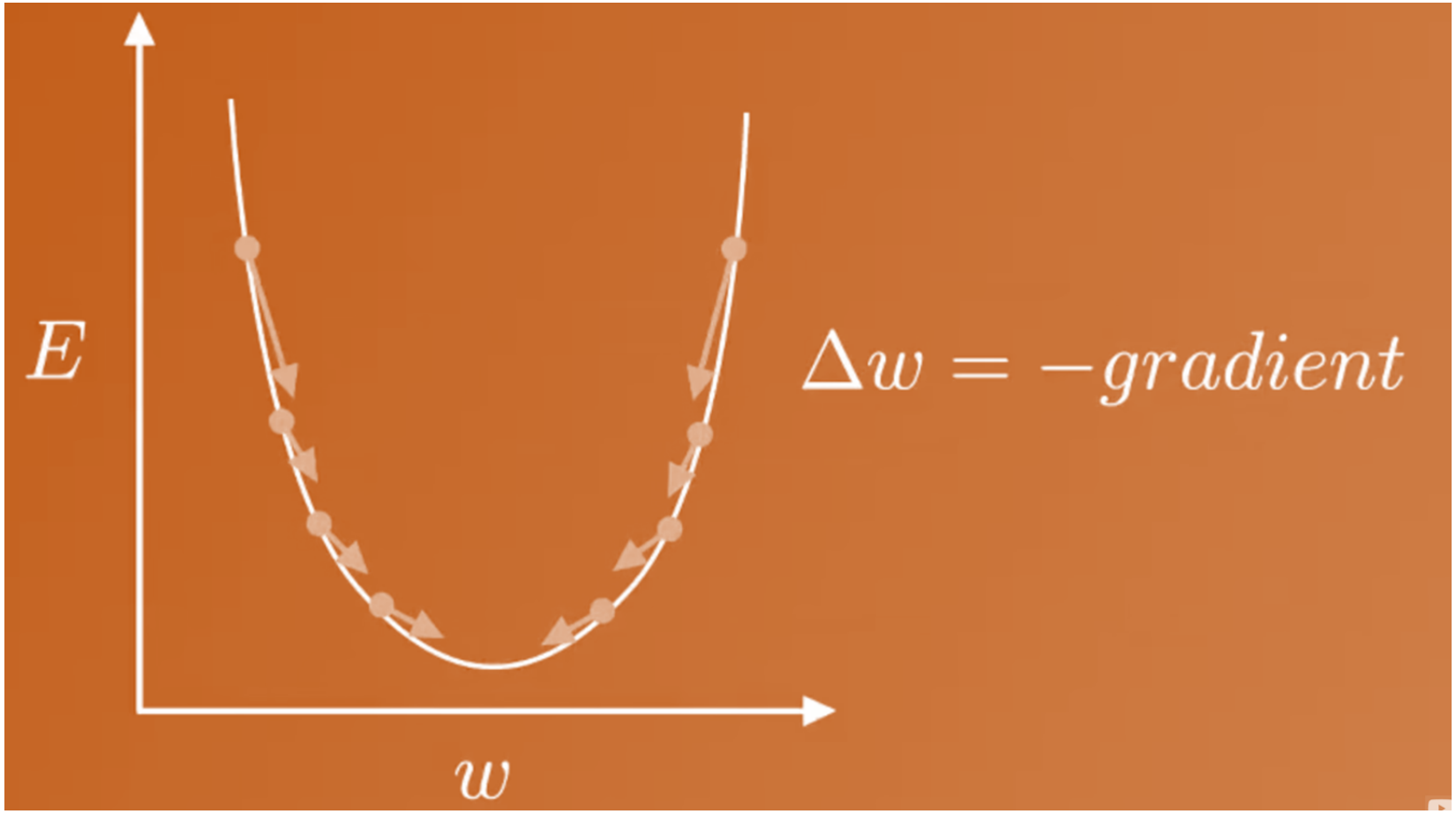
Randomly perturb one weight and see if it improves performance. If so, save the change.

- This is a form of reinforcement learning.
- Very inefficient. We need to do multiple forward passes on a representative set of training cases just to change one weight. Backpropagation is much better.
- Towards the end of learning, large weight perturbations will nearly always make things worse, because the weights need to have the right relative values.

Evolution



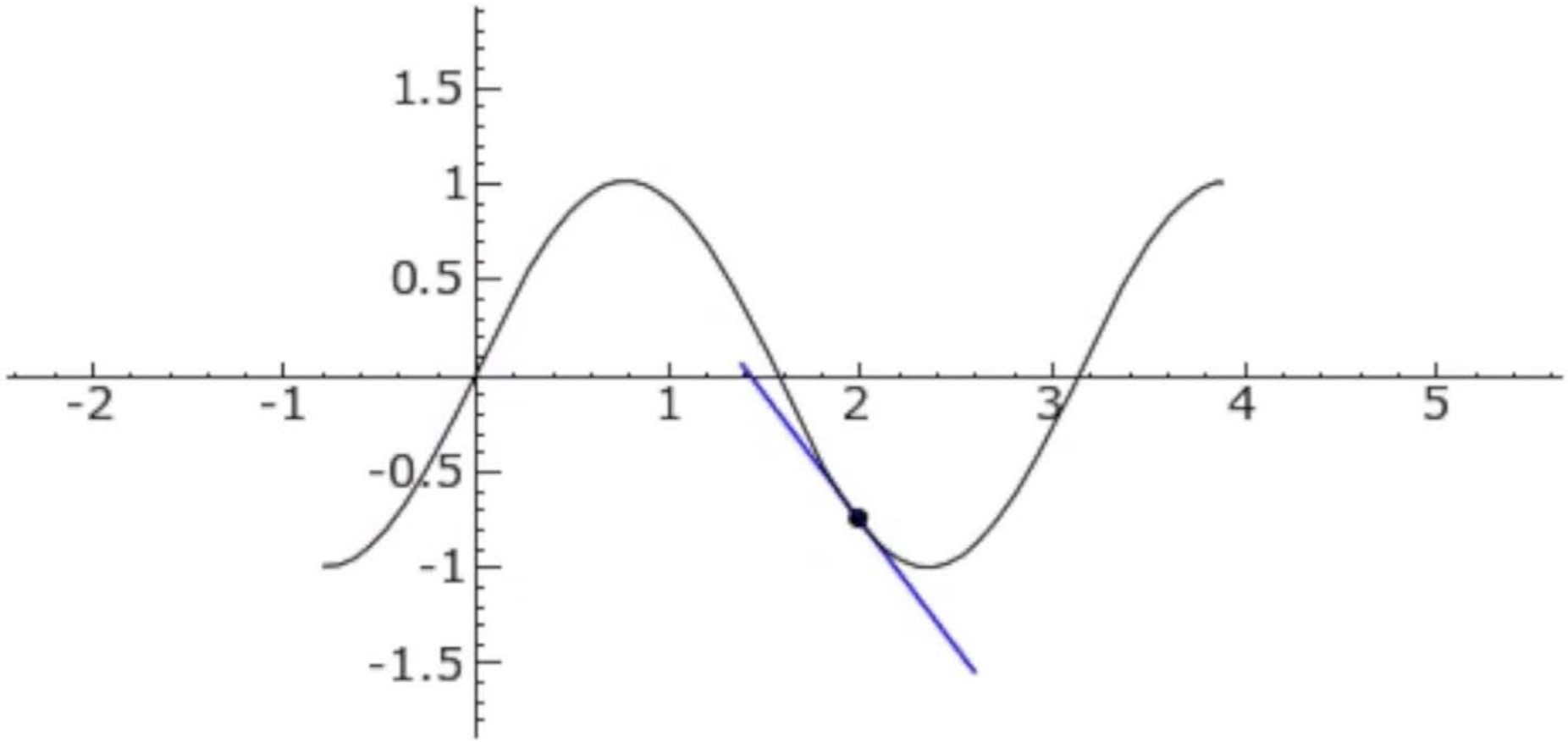
Gradient Descent



Basics

- Derivative
- Partial Derivative
- The Chain Rule

Derivative



Partial Derivative

$$f(x,y) = y^4 + 5xy$$

$$\frac{\partial f}{\partial x} = 5y$$

Partial Derivative with respect to X

$$\frac{\partial f}{\partial y} = 4y^3 + 5x$$

Partial Derivative with respect to Y

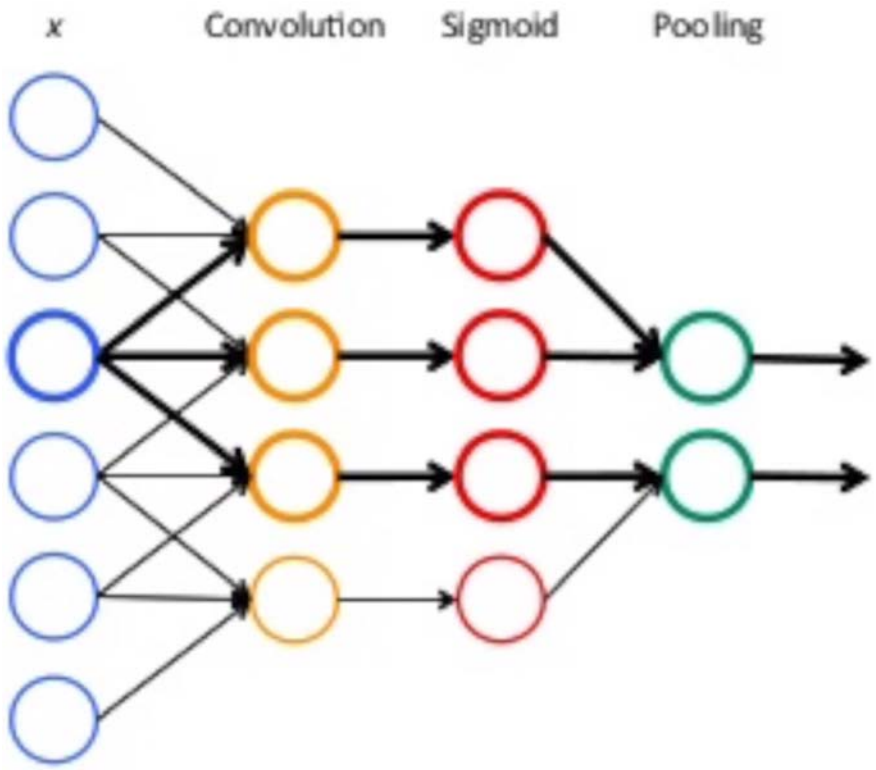
Partial Derivative

$$(h(x))' = (g(f(x)))' = g'(f(x)) \cdot f'(x)$$

derive the outside, keep the inside

derive the inside

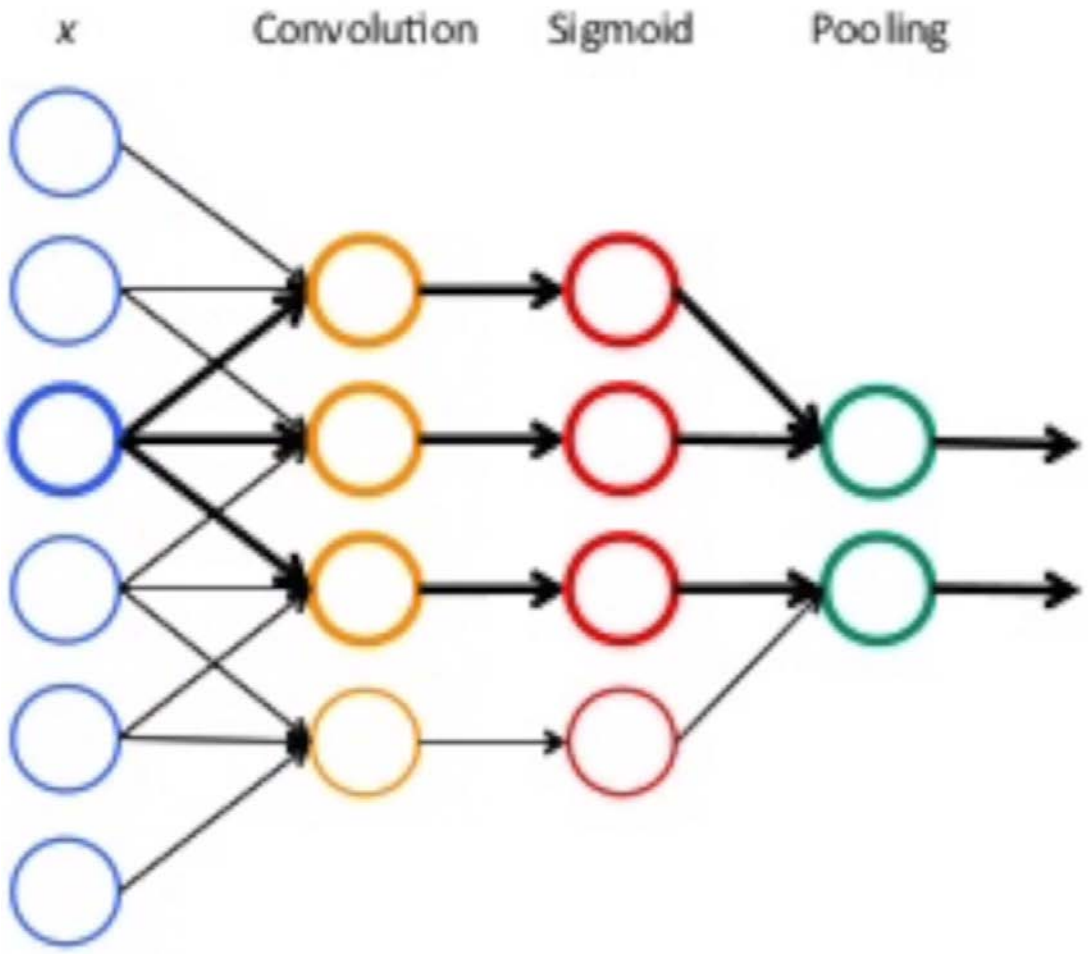
Neural Nets are composite functions



$$\left(f^{(pool)} \circ f^{(sigm)} \circ f_w^{(conv)}\right)(x)$$

Backpropagation

- The purpose of backprop is to compute partial derivatives of an error function with respect to each individual weight



$$\left(f^{(pool)} \circ f^{(sigm)} \circ f_w^{(conv)}\right)(x)$$

Deriving the delta rule

- Define the error as the squared residuals summed over all training cases:
- Now differentiate to get error derivatives for weights
- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases

$$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^n}{\partial w_i} \frac{dE^n}{dy^n}$$

$$= -\sum_n x_i^n (t^n - y^n)$$

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = \sum_n \varepsilon \, x_i^n (t^n - y^n)$$

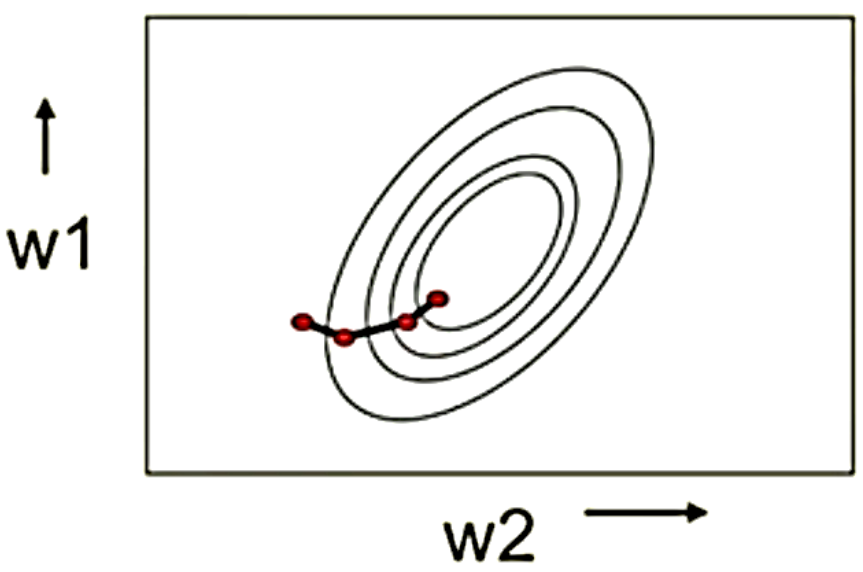
Behavior of the iterative procedure

- Does the learning procedure eventually get the right answer?
  - There may be no perfect answer.
  - By making the learning rate small enough we can get as close as we desire to the best answer.

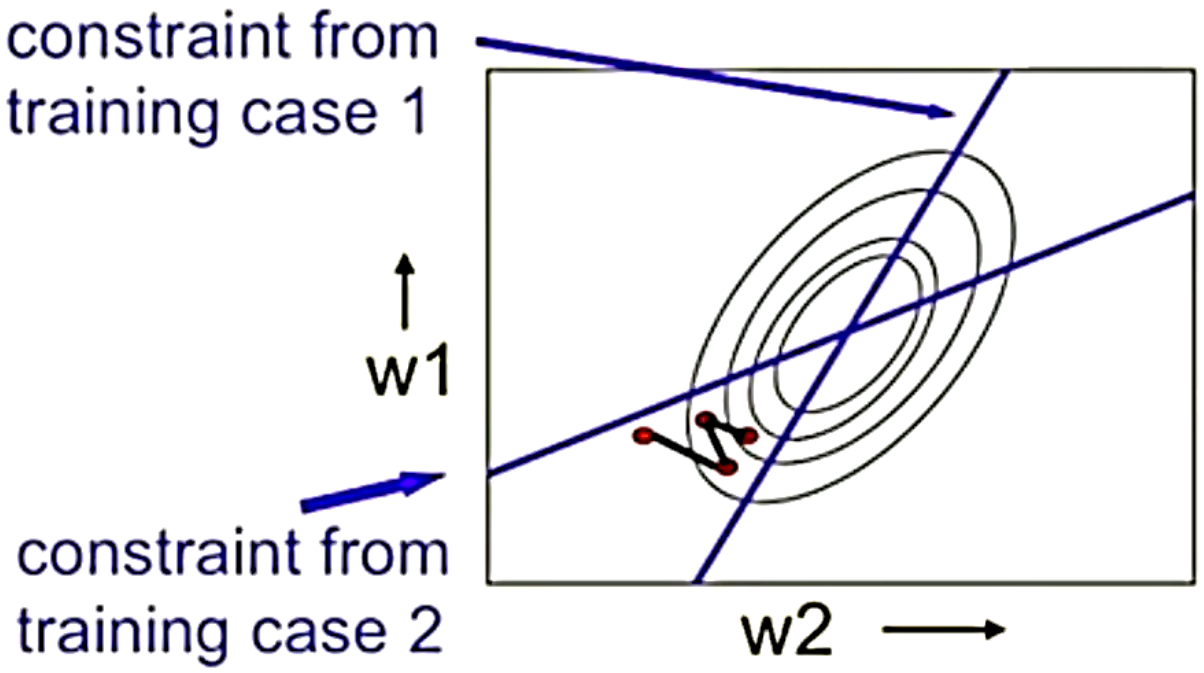
- How quickly do the weights converge to their correct values?
  - It can be very slow if two input dimensions are highly correlated. If you almost always have the same number of portions of ketchup and chips, it is hard to decide how to divide the price between ketchup and chips.

Online vs. batch learning

- The simplest kind of batch learning does steepest descent on the error surface.
  - This travels perpendicular to the contour lines.



- The simplest kind of online learning zig-zags around the direction of steepest descent:



How to train hidden units

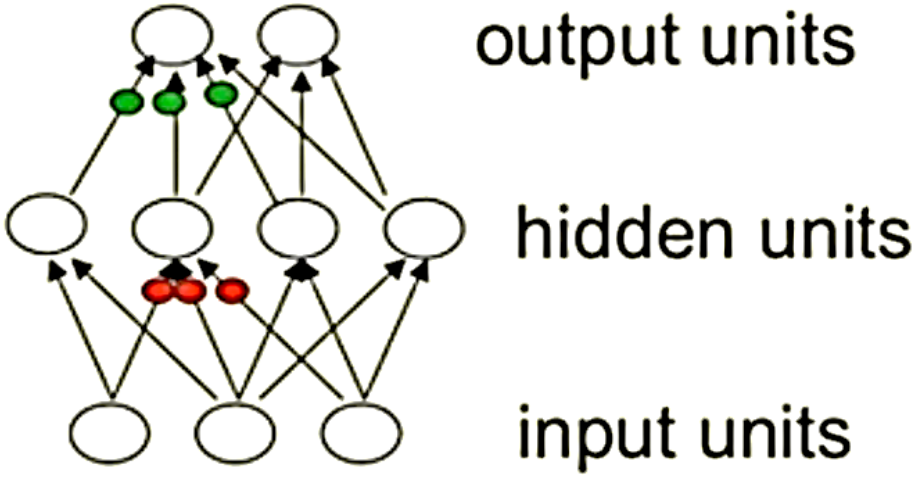
- Networks without hidden units are very limited in the input-output mappings they can model.

- Adding a layer of hand-coded features (as in a perceptron) makes them much more powerful but the hard bit is designing the features.
  - We would like to find good features without requiring insights into the task or repeated trial and error where we guess some features and see how well they work.

- We need to automate the loop of designing features for a particular task and seeing how well they work.

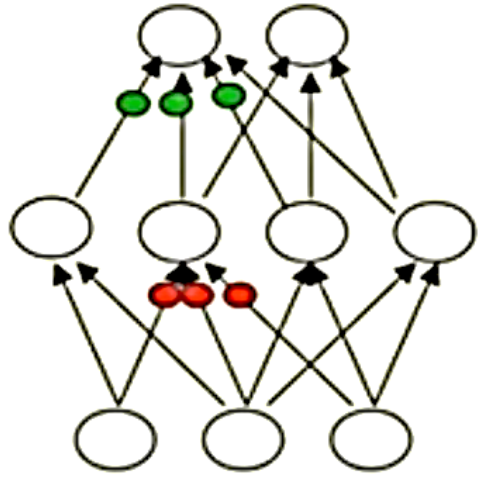
Learning by perturbing weights (evolution)

- Randomly perturb one weight and see if it improves performance. If so, save the change.
  - This is a form of reinforcement learning.
  - Very inefficient. We need to do multiple forward passes on a representative set of training cases just to change one weight. Backpropagation is much better.
  - Towards the end of learning, large weight perturbations will nearly always make things worse, because the weights need to have the right relative values.



Learning by using perturbations

- We could randomly perturb all the weights in parallel and correlate the performance gain with the weight changes.
  - Not any better because we need lots of trials on each training case to "see" the effect of changing one weight through the noise created by all the changes to other weights.
- A better idea: Randomly perturb the activities of the hidden units.
  - Once we know how we want a hidden activity to change on a given training case, we can compute how to change the weights.
  - There are fewer activities than weights, but backpropagation still wins by a factor of the number of neurons.



The idea behind backpropagation

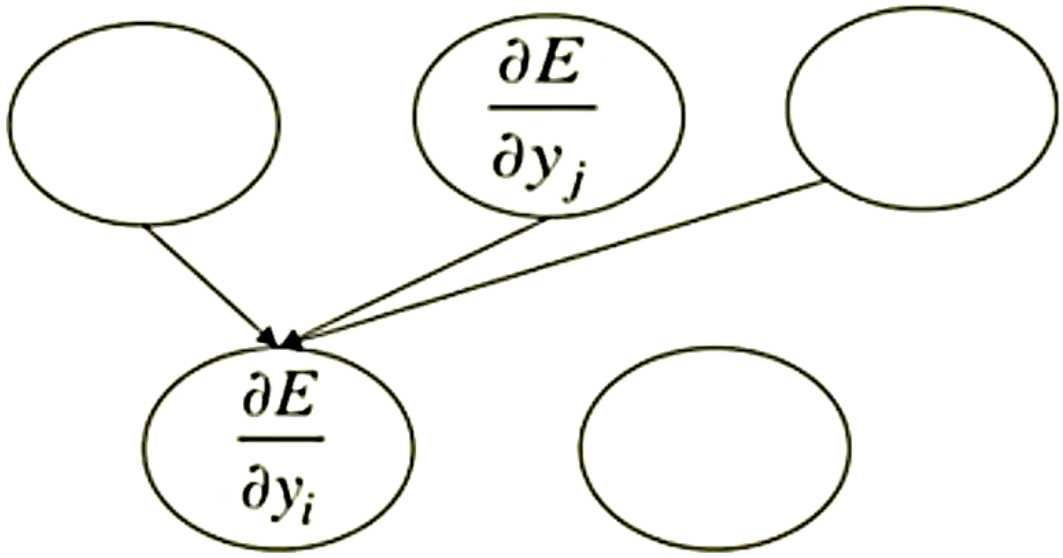
- We don't know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
  - Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
  - Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
- We can compute error derivatives for all the hidden units efficiently at the same time.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

Backpropagation on a single case

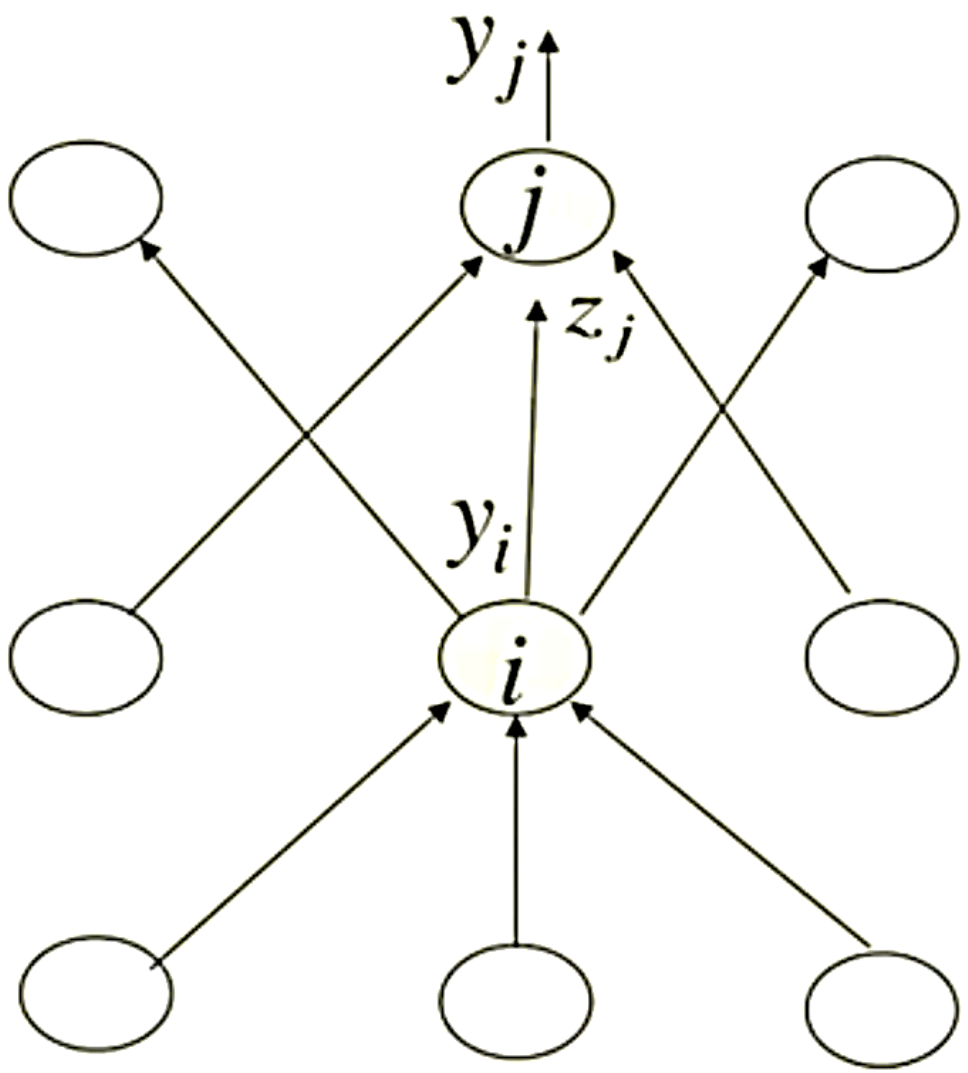
- First convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.
- Then use error derivatives *w.r.t.* activities to get error derivatives *w.r.t.* the incoming weights.

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$



Backpropagating dE/dy

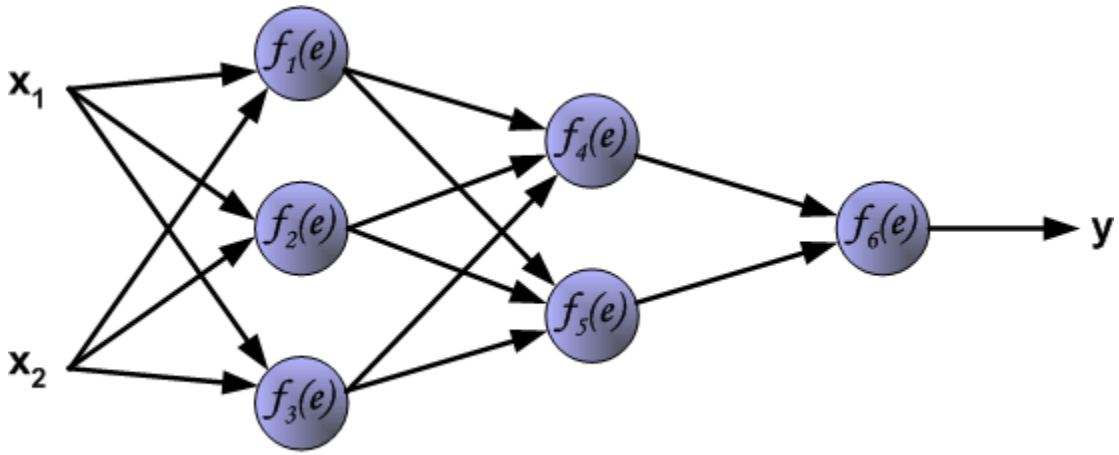


$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

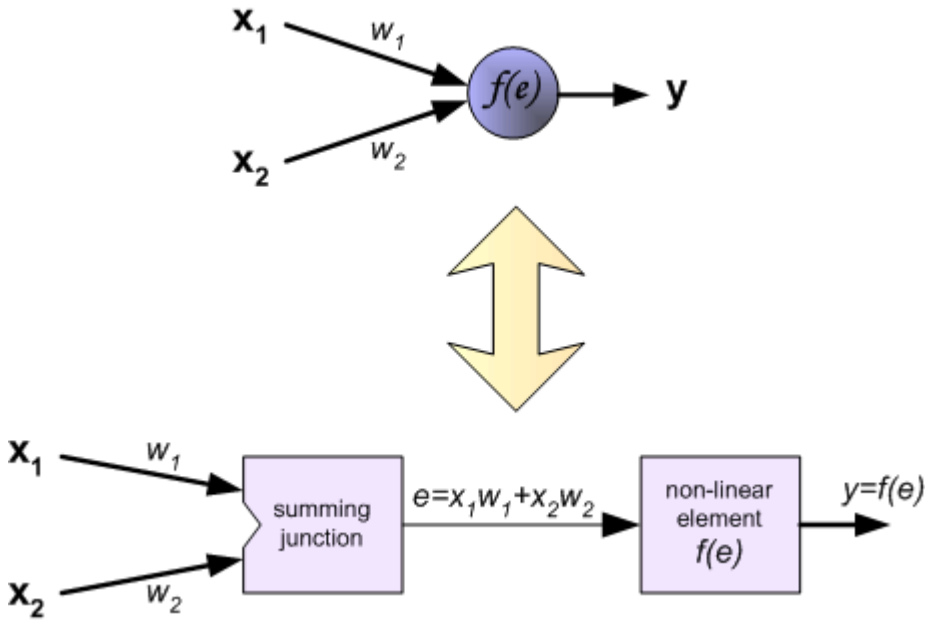
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

Backpropagation



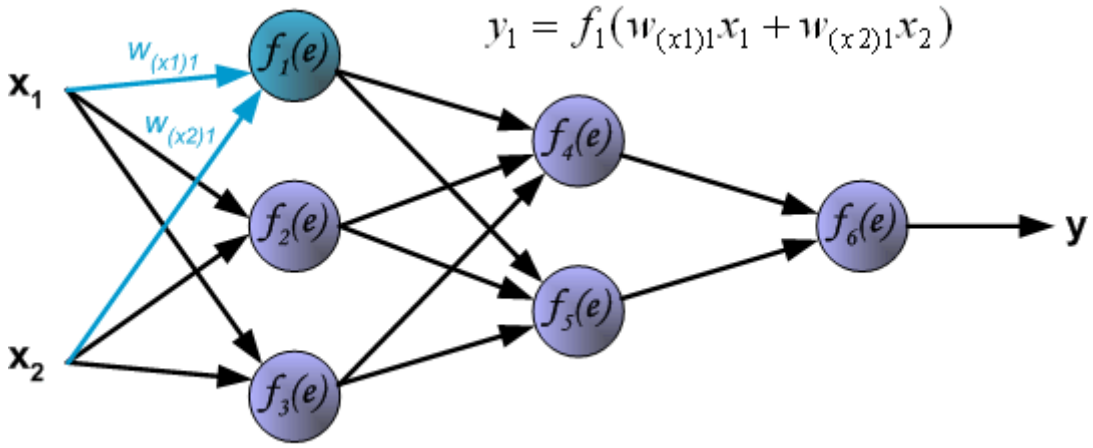
A three layer neural network with two inputs and one output

Backpropagation



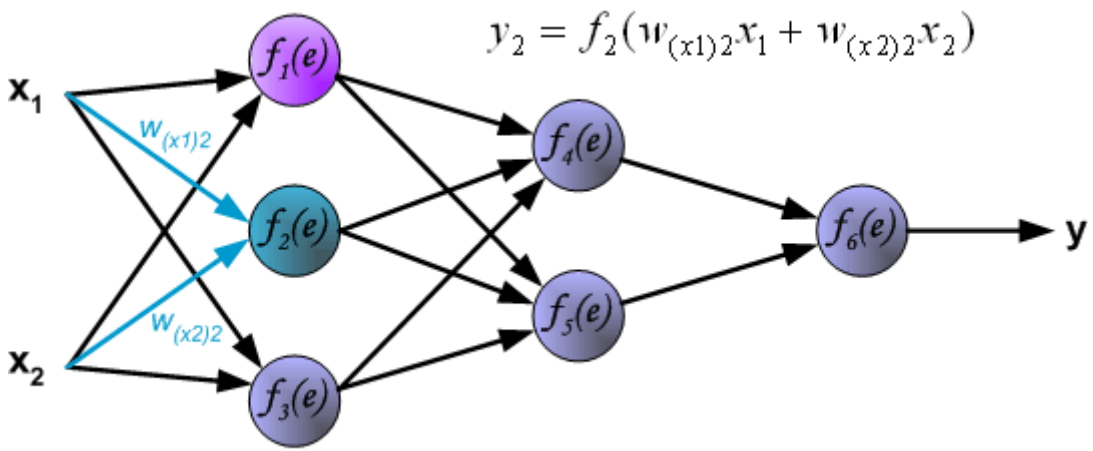
Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realizes a nonlinear function, called neuron activation function. Signal *e* is adder output signal, and *y = f(e)* is output signal of nonlinear element. Signal *y* is also output signal of neuron.

Backpropagation

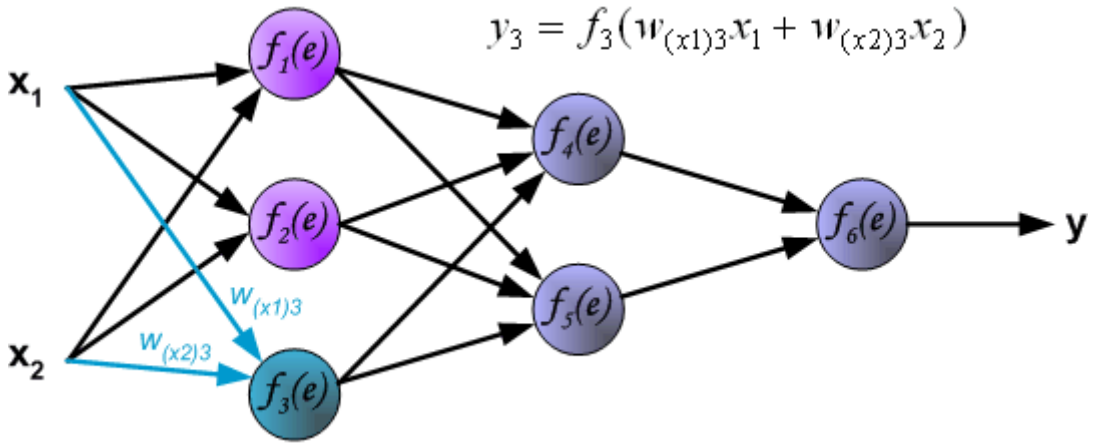


To teach the neural network we need training data set. The training data set consists of input signals (x_1 and x_2) assigned with corresponding target (desired output) *z*. The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set. Modification is calculated using algorithm described below: Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer. Pictures below illustrate how signal is propagating through the network

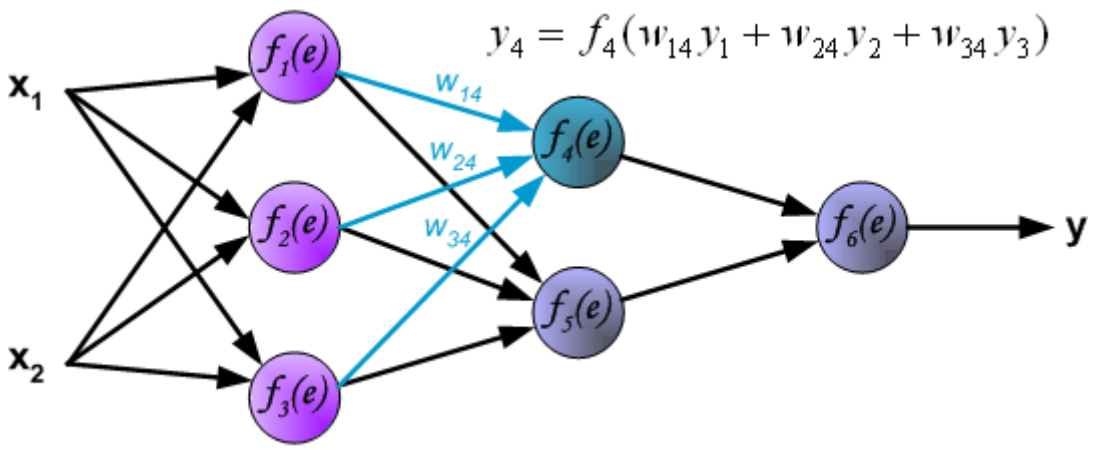
Backpropagation



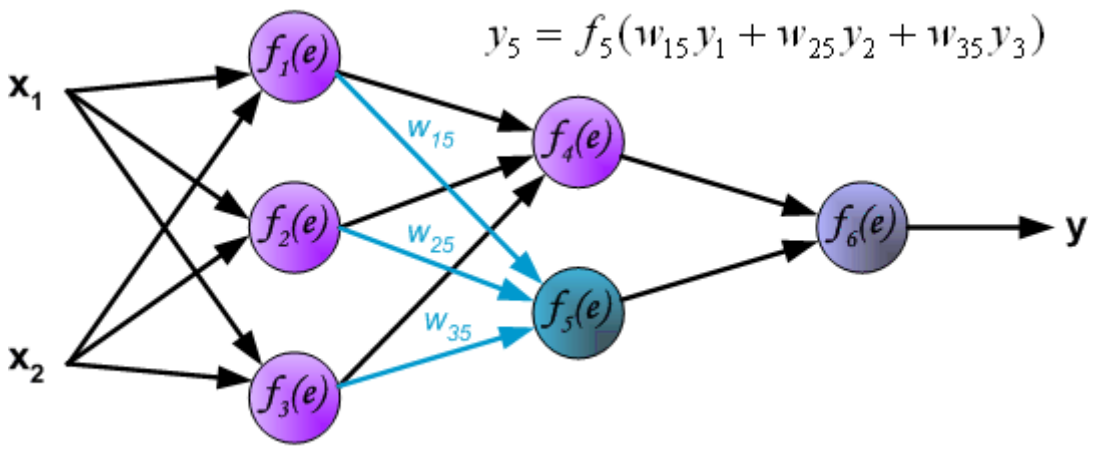
Backpropagation



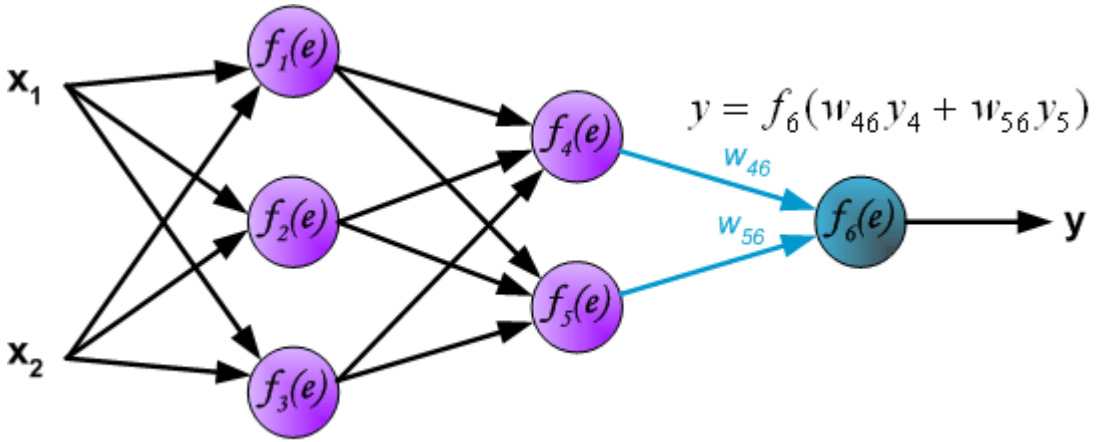
Backpropagation



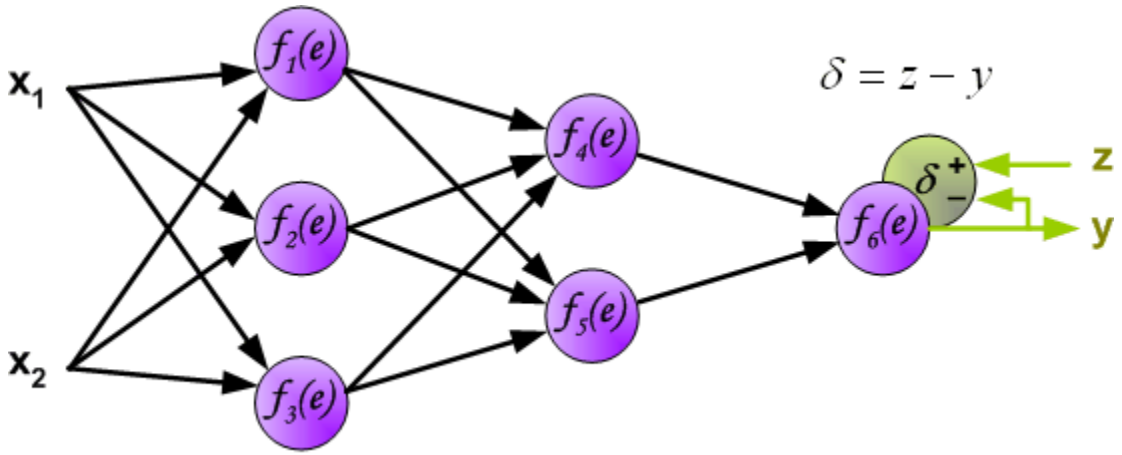
Backpropagation



Backpropagation

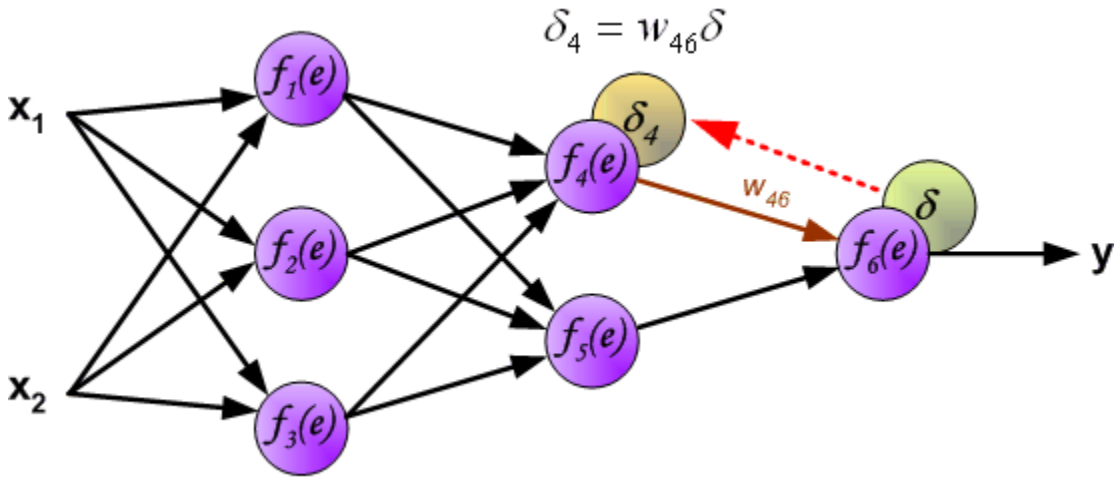


Backpropagation



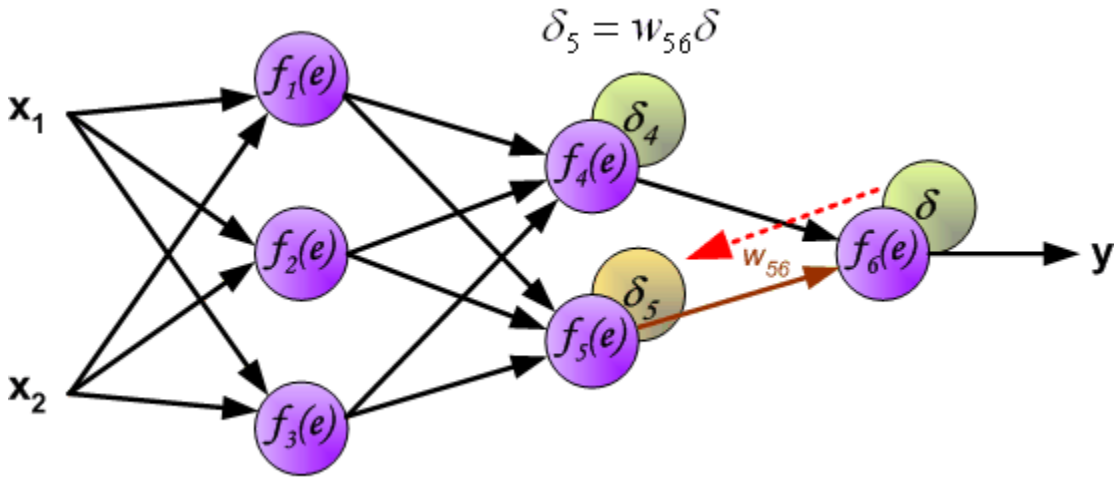
In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal *d* of output layer neuron.

Backpropagation

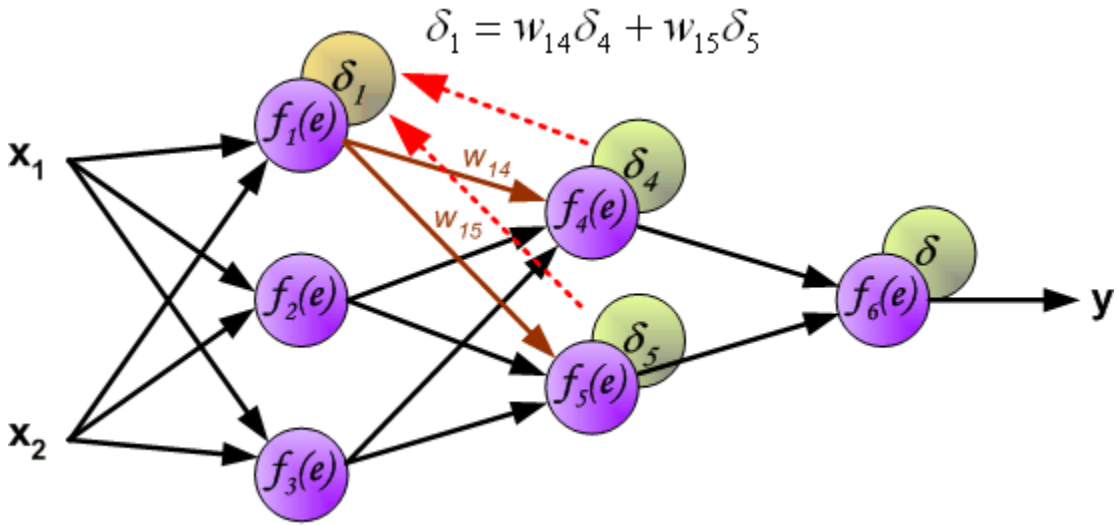


It is impossible to compute error signal for internal neurons directly, because output values of these neurons are unknown. For many years the effective method for training multiplayer networks has been unknown. Only in the mid 80's the backpropagation algorithm has been worked out. The idea is to propagate error signal δ(computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

Backpropagation



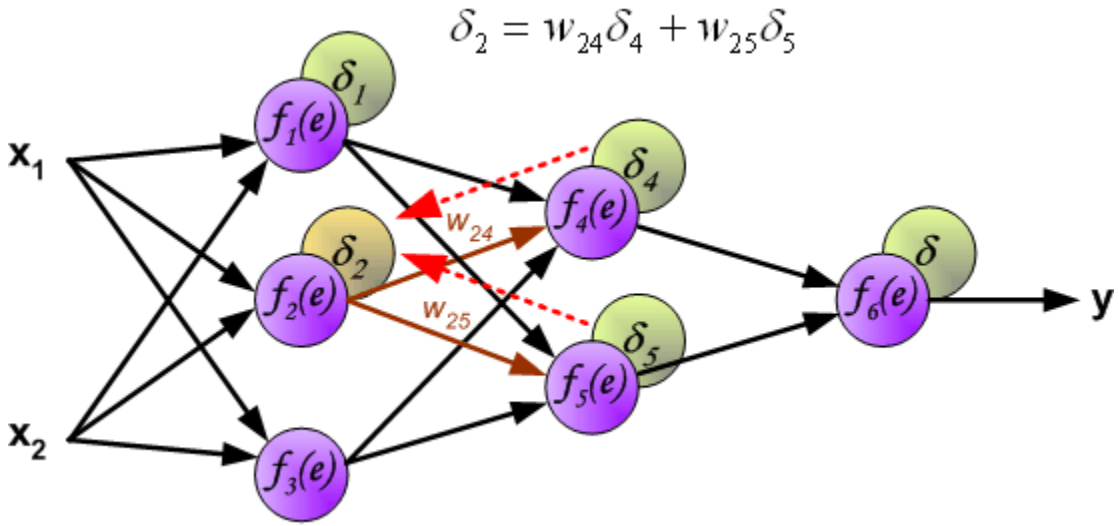
Backpropagation



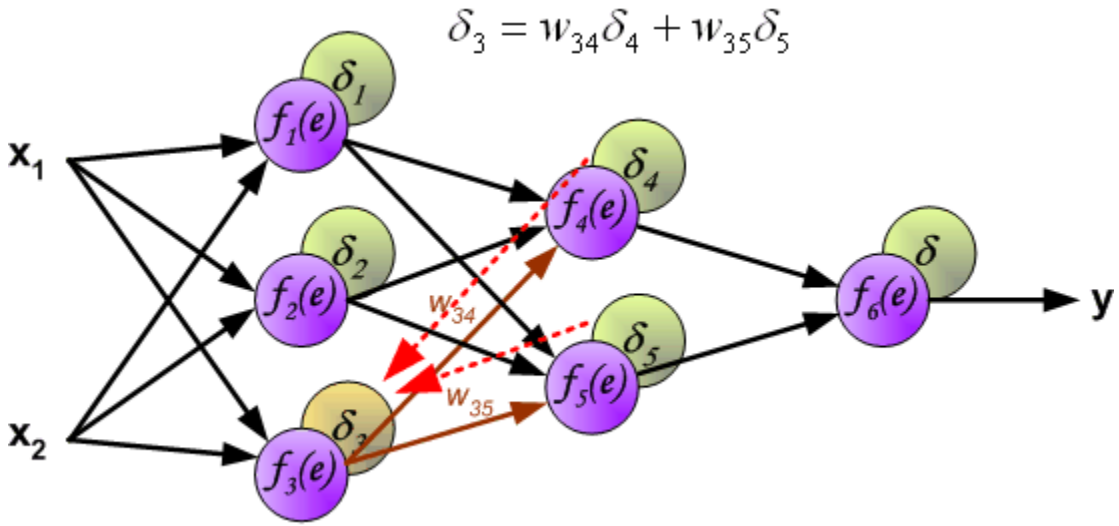
The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers.

If propagated errors came from few neurons they are added.

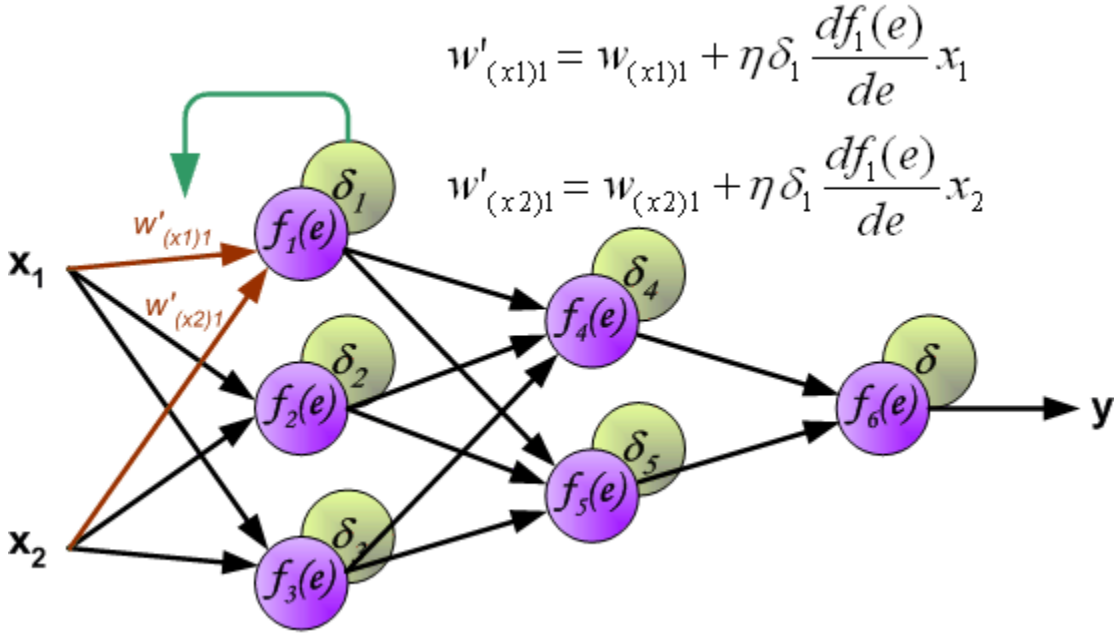
Backpropagation



Backpropagation



Backpropagation

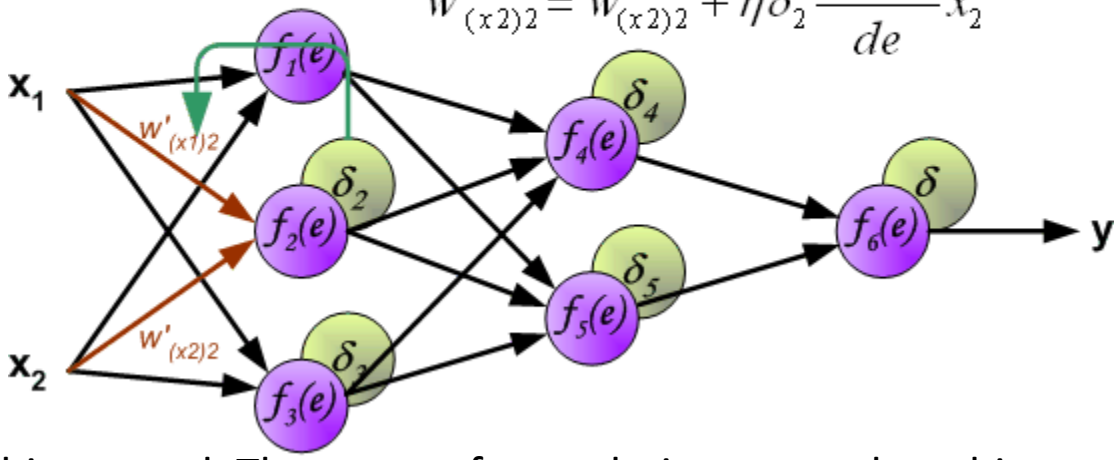


When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas *df(e)/de* represents derivative of neuron activation function (which weights are modified).

Backpropagation

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$

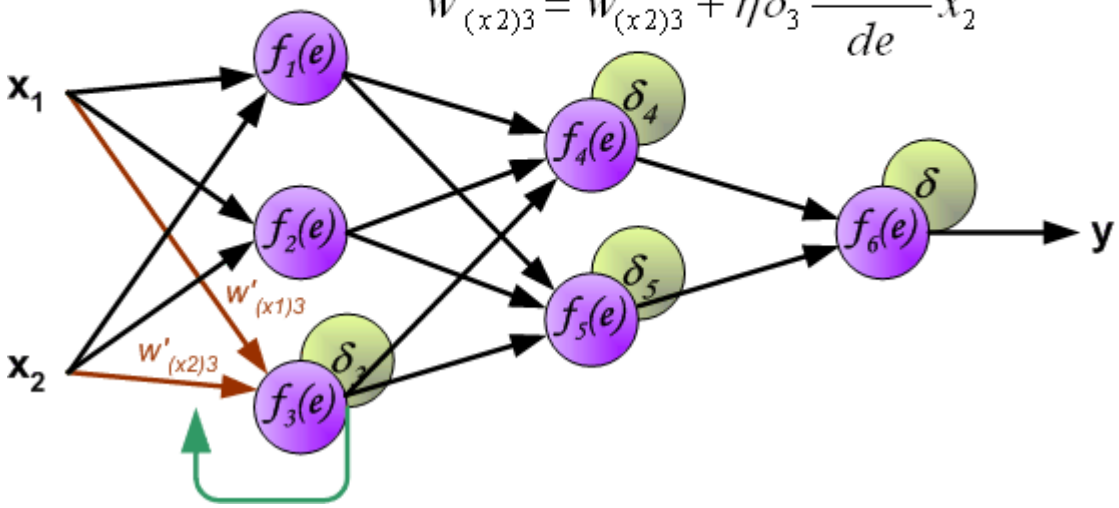


Coefficient η affects network teaching speed. There are a few techniques to select this parameter. The first method is to start teaching process with large value of the parameter. While weights coefficients are being established the parameter is being decreased gradually. The second, more complicated, method starts teaching with small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage. Starting teaching process with low parameter value enables to determine weights coefficients signs.

Backpropagation

$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$

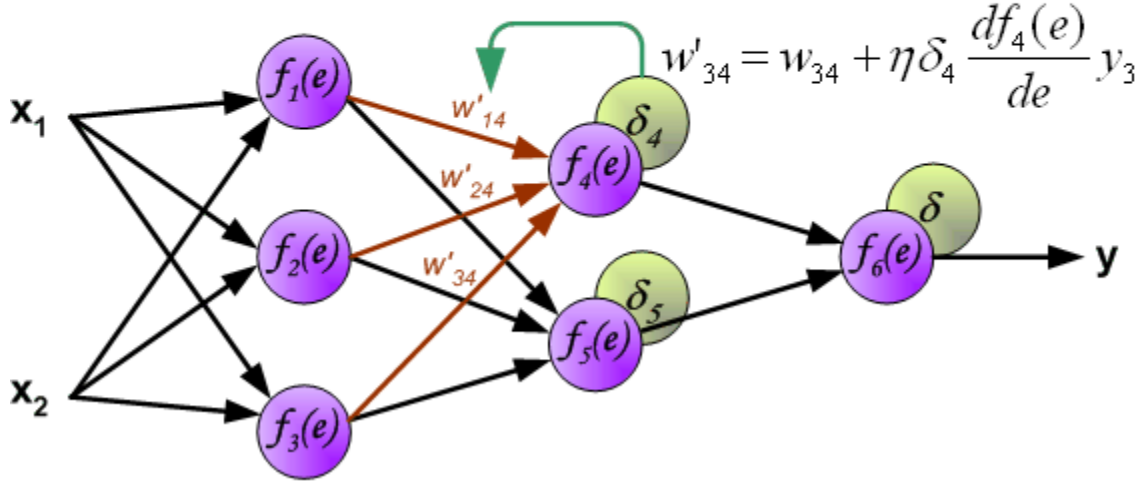


Backpropagation

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$

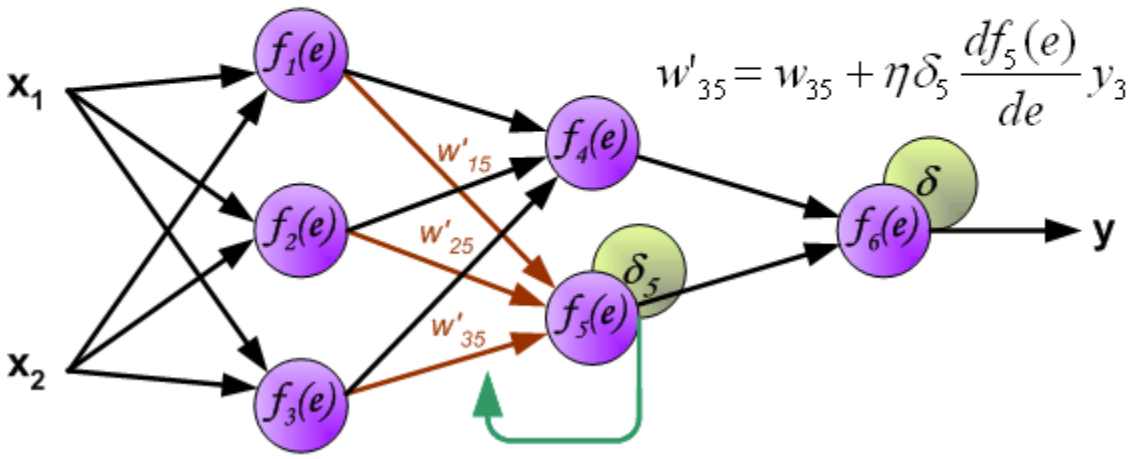


Backpropagation

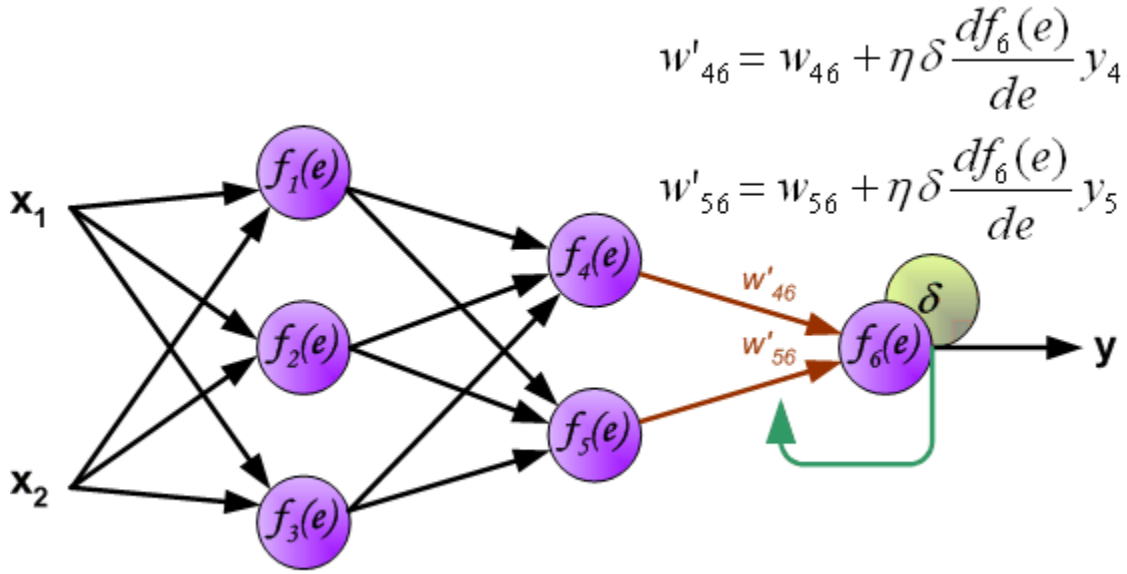
$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

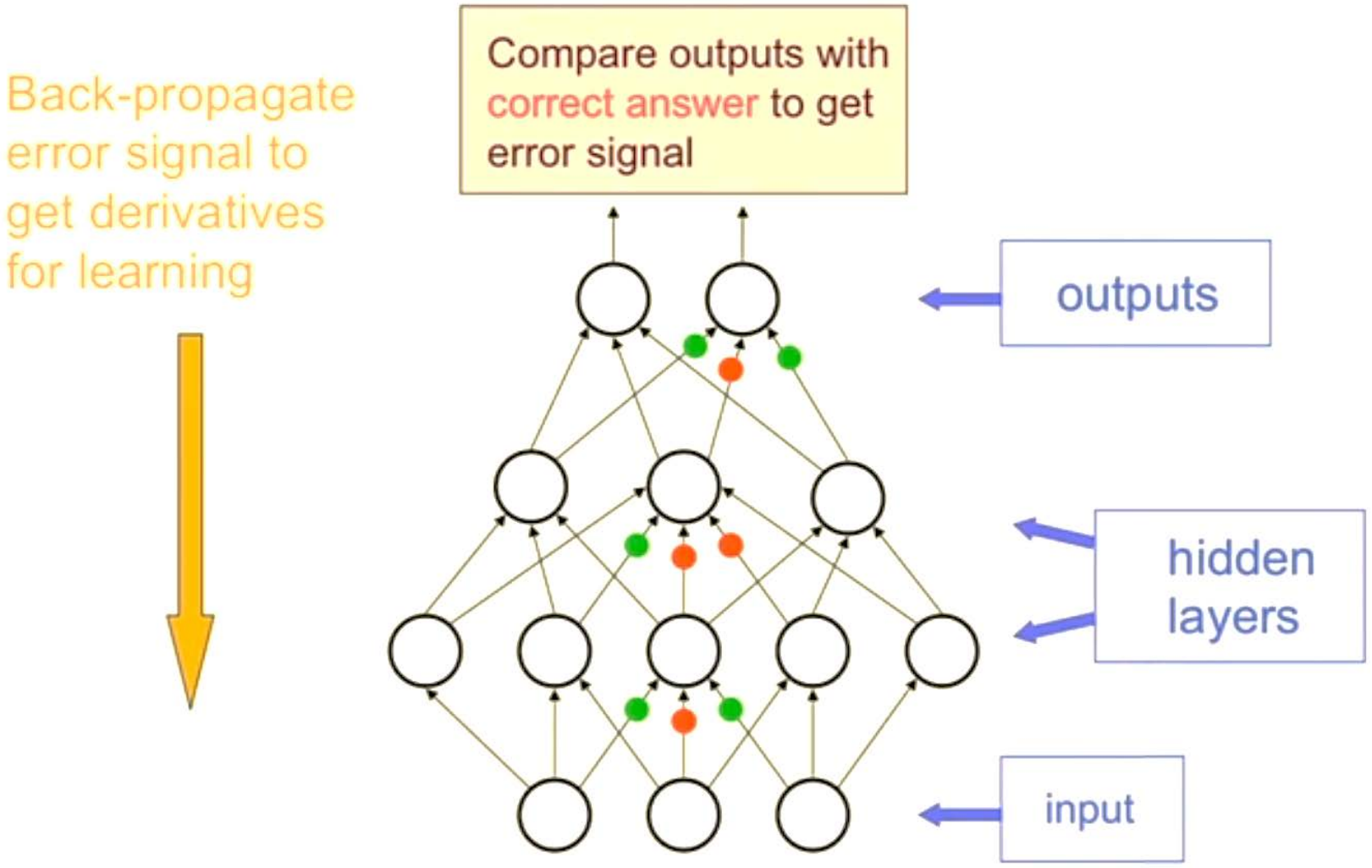
$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$



Backpropagation



Backpropagation



Can the cortex do backpropagation?

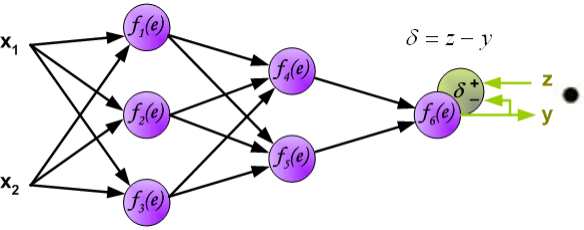
- Neuroscientists often claim that cortex cannot possibly be implementing backpropagation through a hierarchy of cortical areas.

- But backpropagation is what we need to adapt a feature so that it is more useful to higher-level features in the same sensory pathway.
  - And we now know that it works extremely well for really tough practical problems.

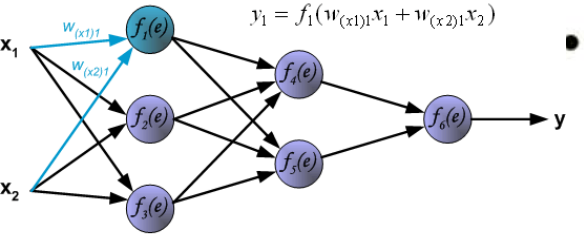
- So why do neuroscientists think its impossible to implement in cortex?

people believed

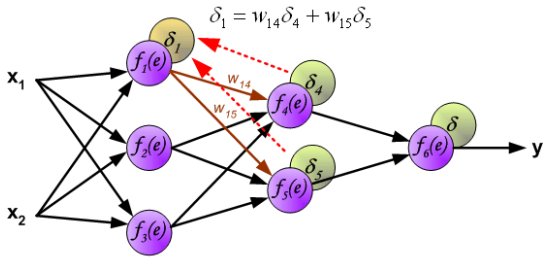
Four reasons why the brain cannot do backprop



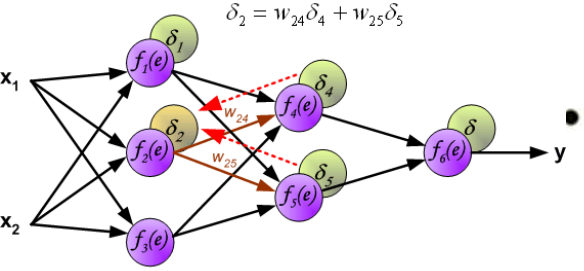
- There is no obvious source for the supervision signal.



- Cortical neurons do not communicate real-valued activities
  - They send all-or-none spikes.

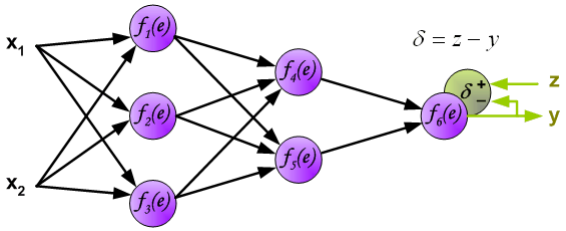


- The neurons need to send two different types of signal
  - Forward pass: output = activity = y
  - Backward pass: output = error deriv w.r.t input = dE/dx



- Neurons do not have symmetric reciprocal connections.
  - The feedback connections do not even go back to the neurons that the feed-forward connections came from.

1. Source for the supervision signal

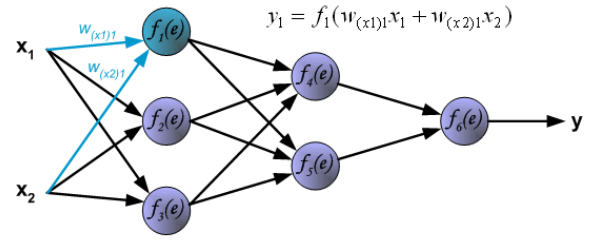


- There are many ways to get supervision signals
  - In fact, we do not need to inject a separate label for each training case
- 1980's: Reconstruct all or part of the input vector (auto-encoders)
- 1990's: Make locally extracted features agree with features predicted from a broader context or from another modality
  - "she scromed him with a frying pan"
- 1990's: Learn a generative model that assigns high log probability to the input vector
  - -e.g., the "wake-sleep" algorithm
- 2010's: variational autoencoder (Welling and Kingma)

2. Can neurons communicate real values?

- Backpropagation uses real-values activities in the FWD pass and real-valued derivatives in the BWD pass

- BUT: backpropagation **still works very well** if we add a lot of unbiased noise in the FWD and BWD passes
  - Take a logistic unit and randomly quantize its output to {0, 1} – the algorithm works!
  - We can use 1 bit in the FWD pass and **2 bits** in the BWD pass, if we make them stochastic so that the expected values are correct



- The brain does not want to send accurate real values
  - (noisy signals that are stochastic and have the right expected value)

3. How are the derivatives sent backwards?

- The output of a neuron represents the presence of a feature in the current input.
  - So its obvious that it cannot also represent the derivative of the cost function with respect to the input to the neuron (which is what needs to be sent back down).
- So we obviously need different neurons to send error derivatives backwards.
  - These neurons probably need to come in pairs because it is hard for one neuron to send both positive and negative derivatives (Dale's law).

Temporal derivatives represent error derivatives

- The rate of change of the neuron's output represents the error derivative w.r.t. the neuron's input

$$-\partial E / \partial x_j = \dot{y}_j$$

$-\partial E / \partial x_j$: error derivative

$=$

$\dot{y}_j$: temporal derivative

This allows the output of a neuron to simultaneously encode both the normal activity and the error derivative w.r.t. its input (for a limited time).

Temporal derivatives – need to be careful

- The output of a neuron represents what is going on in the world
  - The rate at which the output is changing does **NOT** represent how fast the encoded property of the world is changing
  - In other words, if I have a neuron representing the location of my hand, and the activity of the neuron changes, this does not mean that my hand is moving – the rate of change of the firing rate is going to represent an error derivative
  - Major drawback: If the brain does this, then it can no longer use the rate of change of a neuron firing to represent the rate at which the feature changes
    - **In fact, the brain does that**! We have different neurons to represent velocity and other neurons represent acceleration

4. Requirement for symmetric weights

- What happens if the top-down weights are not the same as the bottom-up weights?

- In the extreme case, what happens if we have sparse connectivity in both directions and we do not allow top-down connections between pairs of units that have bottom-up connections?

- In the backpropagation algorithm, when the signal comes backwards, it comes through the transpose of the connectivity matrix – is this happening in the brain?

We do not need symmetric weights!

- Lillicrap, Cownden, Tweed & Akerman (2014) showed that backprop still works almost as well using fixed random top-down connection weights.
  - The bottom-up weights adapt so that the fixed top-down weights are approximately their pseudo-inverse near the data manifold.
- This result means that the stack of autoencoders does not need to have symmetric weights.
  - But it does need to be able to reconstruct well.

Future?

- Google knows what you want to search before you …
- Facebook can automatically tag you in a picture
- Cars drive by themselves
- Automatic Translators are silently becoming more and more useful

vicarious

Fei – Fei Li
Associate Professor
Computer Science
Stanford University

brain CORPORATION

QUALCOMM®

Brain Corporation applies expertise in machine learning and computer vision to create intelligent systems capable of functioning autonomously in complex human environments.

We're building brains for robots, and turning today's manually-operated machines into tomorrow's autonomous solutions.

"Vicarious is bringing us all closer to a future where computers perceive, imagine, and reason just like humans."

Peter Thiel

"The technology that Vicarious is developing has the potential to improve all lives and revolutionize every industry."

Dustin Moskovitz

IBM

Epic Shift to Low-Power Supercomputers

Dharmendra Modha, IBM Fellow

"The architecture can solve a wide class of problems from vision, audition, and multi-sensory fusion, and has the potential to revolutionize the computer industry by integrating brain-like capability into devices where computation is constrained by power and speed."

— Dharmendra Modha, IBM Fellow

Read Dr. Modha's SyNAPSE article, here.