DIP: Machine Learning _ Basic Concepts

Presenter: Dr. Ha Viet Uyen Synh.

Motivation

PyTorch and TensorFlow are two of the most commonly used deep learning frameworks. Both of these tools are notable for their ability to compute gradients automatically and do operations on GPU, which can be by orders of magnitude faster than running on CPU. Even though both libraries serve the same purpose, they are quite different in the ways they approach calculations.

PyTorch provides two main features:

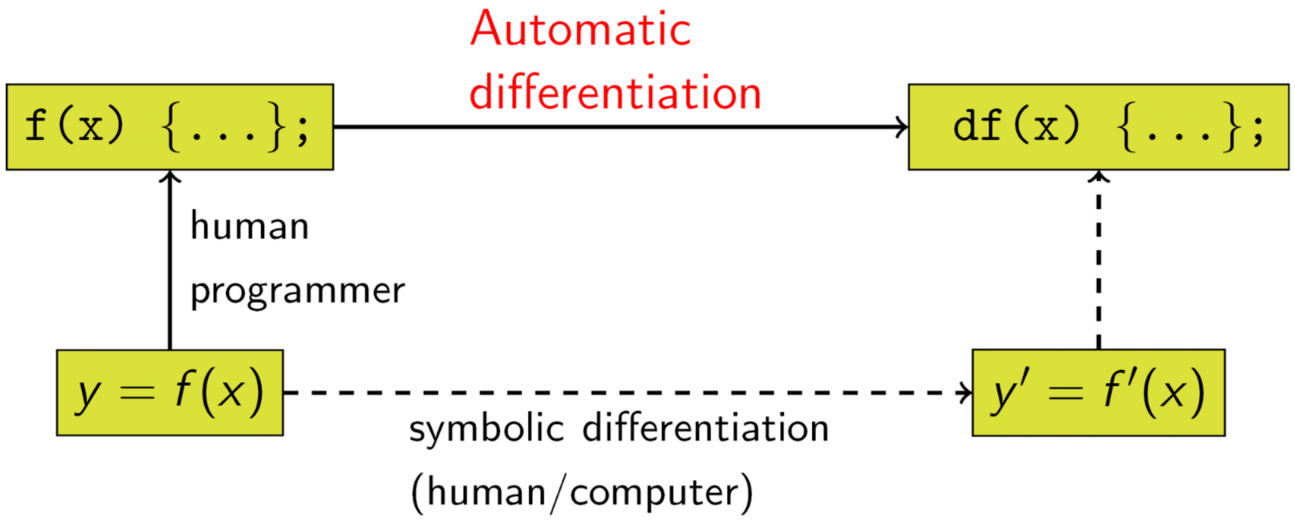
- An n-dimensional Tensor, similar to NumPy but can run on GPUs
- Automatic differentiation for building and training neural networks

In PyTorch, there's only one world: the graph is defined by python code. Also all the tensors have a numeric value. You can compute outputs on the fly without pre-declaring anything. Also the code looks (almost) exactly as in pure NumPy.

Part A

AUTOMATIC DIFFERENTIATION

Autograd


Automatic differentiation
f(x) {...};
df(x) {...};
human programmer
$y = f(x)$
$y' = f'(x)$
symbolic differentiation (human/computer)

Fundamental to AD is the decomposition of differentials provided by the chain rule

$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

$$\begin{aligned} w_0 &= x \\ w_1 &= h(w_0) \\ w_2 &= g(w_1) \\ w_3 &= f(w_2) = y \end{aligned}$$

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx} = \frac{df(w_2)}{dw_2}\frac{dg(w_1)}{dw_1}\frac{dh(w_0)}{dx}$$

Autograd

Usually, two distinct modes of AD are presented, forward accumulation (or forward mode) and reverse accumulation (or reverse mode).

Forward accumulation specifies that one traverses the chain rule from inside to outside, while reverse accumulation has the traversal from outside to inside

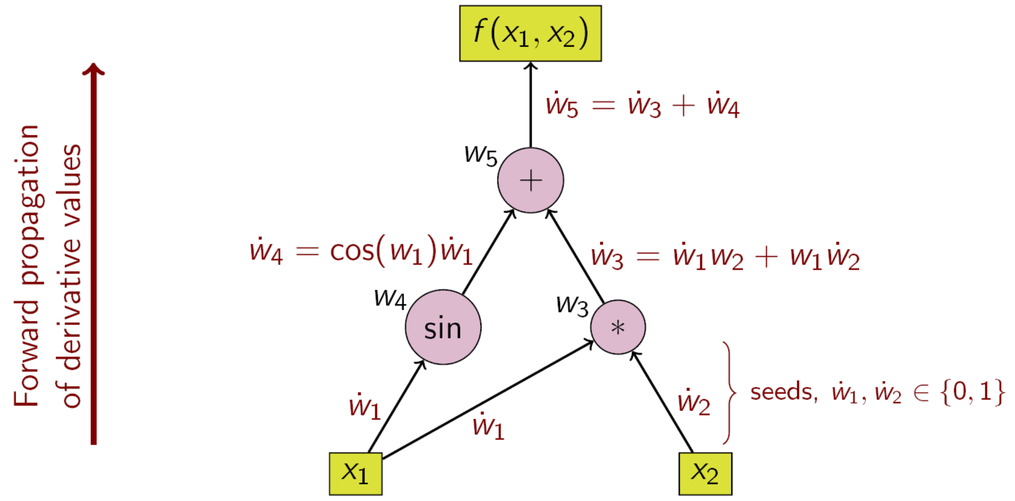
forward accumulation computes the recursive relation:

$$\frac{dw_i}{dx} = \frac{dw_i}{dw_{i-1}} \frac{dw_{i-1}}{dx} \qquad w_3 = y$$

reverse accumulation computes the recursive relation:

$$\frac{dy}{dw_i} = \frac{dy}{dw_{i+1}} \frac{dw_{i+1}}{dw_i} \qquad w_0 = x$$

Forward accumulation


$f(x_1, x_2)$
$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$
w_5
$+$
$\dot{w}_4 = \cos(w_1)\dot{w}_1$
$\dot{w}_3 = \dot{w}_1 w_2 + w_1 \dot{w}_2$
w_4
sin
w_3
$*$
seeds, $\dot{w}_1, \dot{w}_2 \in \{0, 1\}$
$\dot{w}_1$
$\dot{w}_1$
$\dot{w}_2$
x_1
x_2
Forward propagation of derivative values

$$\begin{aligned}
\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial w_{n-1}} \frac{\partial w_{n-1}}{\partial x} \\
&= \frac{\partial y}{\partial w_{n-1}} \left(\frac{\partial w_{n-1}}{\partial w_{n-2}} \frac{\partial w_{n-2}}{\partial x} \right) \\
&= \frac{\partial y}{\partial w_{n-1}} \left(\frac{\partial w_{n-1}}{\partial w_{n-2}} \left(\frac{\partial w_{n-2}}{\partial w_{n-3}} \frac{\partial w_{n-3}}{\partial x} \right) \right) \\
&= \cdots
\end{aligned}$$

$$\dot{w} = \frac{\partial w}{\partial x}$$

Example

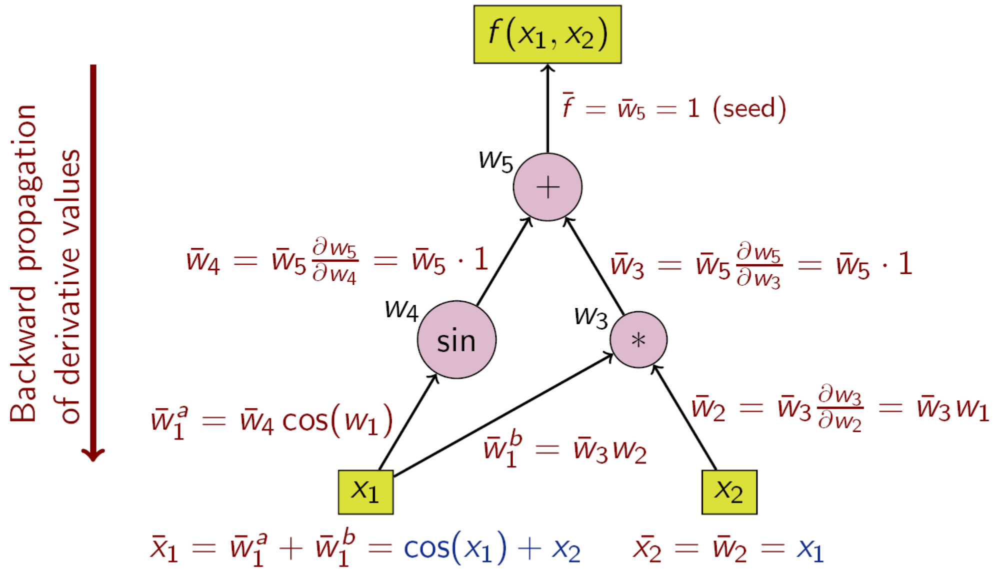
$$\begin{aligned} z &= f(x_1, x_2) \\ &= x_1 x_2 + \sin x_1 \\ &= w_1 w_2 + \sin w_1 \\ &= w_3 + w_4 \\ &= w_5 \end{aligned}$$

$$\dot{w}_1 = \frac{\partial x_1}{\partial x_1} = 1$$

$$\dot{w}_2 = \frac{\partial x_2}{\partial x_1} = 0$$

Operations to compute value	Operations to compute derivative
$w_1 = x_1$	$\dot{w}_1 = 1$ (seed)
$w_2 = x_2$	$\dot{w}_2 = 0$ (seed)
$w_3 = w_1 \cdot w_2$	$\dot{w}_3 = w_2 \cdot \dot{w}_1 + w_1 \cdot \dot{w}_2$
$w_4 = \sin w_1$	$\dot{w}_4 = \cos w_1 \cdot \dot{w}_1$
$w_5 = w_3 + w_4$	$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$

Reverse accumulation


Backward propagation of derivative values
$f(x_1, x_2)$
$\bar{f} = \bar{w}_5 = 1$ (seed)
w_5
$+$
$\bar{w}_4 = \bar{w}_5 \frac{\partial w_5}{\partial w_4} = \bar{w}_5 \cdot 1$
$\bar{w}_3 = \bar{w}_5 \frac{\partial w_5}{\partial w_3} = \bar{w}_5 \cdot 1$
w_4
sin
w_3
$*$
$\bar{w}_1^a = \bar{w}_4 \cos(w_1)$
$\bar{w}_2 = \bar{w}_3 \frac{\partial w_3}{\partial w_2} = \bar{w}_3 w_1$
$\bar{w}_1^b = \bar{w}_3 w_2$
x_1
x_2
$\bar{x}_1 = \bar{w}_1^a + \bar{w}_1^b = \cos(x_1) + x_2$
$\bar{x}_2 = \bar{w}_2 = x_1$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \cdots$$

$$\bar{w} = \frac{\partial y}{\partial w}$$

Operations to compute derivative

$$\bar{w}_5 = 1 \text{ (seed)}$$
$$\bar{w}_4 = \bar{w}_5$$
$$\bar{w}_3 = \bar{w}_5$$
$$\bar{w}_2 = \bar{w}_3 \cdot w_1$$
$$\bar{w}_1 = \bar{w}_3 \cdot w_2 + \bar{w}_4 \cdot \cos w_1$$

Part C

ARTIFICIAL NEURAL NETWORK

Introduction To Neural Networks

Some NNs are models of biological neural networks and some are not, but historically, much of the inspiration for the field of NNs came from the desire to produce artificial systems capable of sophisticated, perhaps intelligent, computations similar to those that the human brain routinely performs, and thereby possibly to enhance our understanding of the human brain.

Most NNs have some sort of training rule. In other words, NNs learn from examples (as children learn to recognize dogs from examples of dogs) and exhibit some capability for generalization beyond the training data.

Neural computing must not be considered as a competitor to conventional computing. Rather, it should be seen as complementary as the most successful neural solutions have been those which operate in conjunction with existing, traditional techniques.

Applications off NNs

classification

- in marketing: consumer spending pattern classification
- In defence: radar and sonar image classification
- In agriculture & fishing: fruit and catch grading
- In medicine: ultrasound and electrocardiogram image classification, EEGs, medical diagnosis

recognition and identification

- In general computing and telecommunications: speech, vision and handwriting recognition
- In finance: signature verification and bank note verification

assessment

- In engineering: product inspection monitoring and control
- In defence: target tracking
- In security: motion detection, surveillance image analysis and fingerprint matching

forecasting and prediction

- In finance: foreign exchange rate and stock market forecasting
- In agriculture: crop yield forecasting
- In marketing: sales forecasting
- In meteorology: weather prediction

What can you do with an NN and what not?

In principle, NNs can compute any computable function, i.e., they can do everything a normal digital computer can do. Almost any mapping between vector spaces can be approximated to arbitrary precision by feedforward NNs

In practice, NNs are especially useful for classification and function approximation problems usually when rules such as those that might be used in an expert system cannot easily be applied.

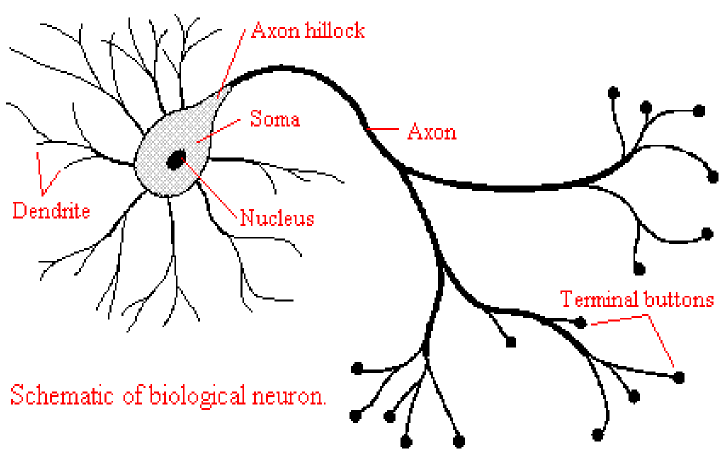
NNs are, at least today, difficult to apply successfully to problems that concern manipulation of symbols and memory. And there are no methods for training NNs that can magically create information that is not contained in the training data.

The Biological Neuron

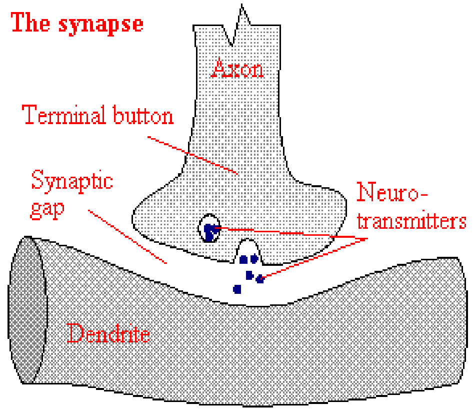
The brain is a collection of about 10 billion interconnected neurons. Each neuron is a cell that uses biochemical reactions to receive, process and transmit information.

Each terminal button is connected to other neurons across a small gap called a synapse.

A neuron's dendritic tree is connected to a thousand neighbouring neurons. When one of those neurons fire, a positive or negative charge is received by one of the dendrites. The strengths of all the received charges are added together through the processes of spatial and temporal summation.

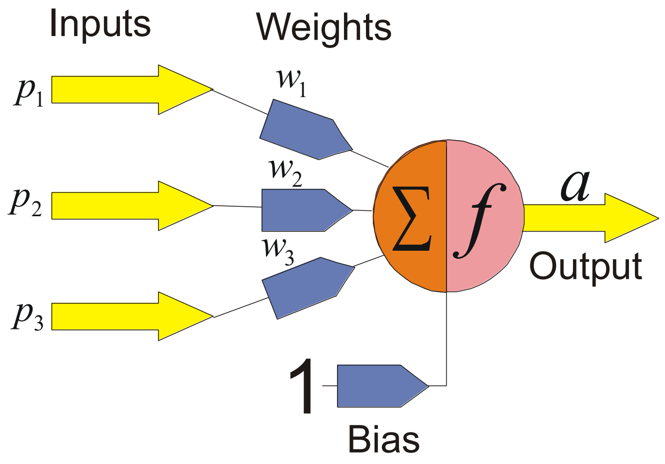


Schematic of biological neuron.



The Key Elements of Neural Networks

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

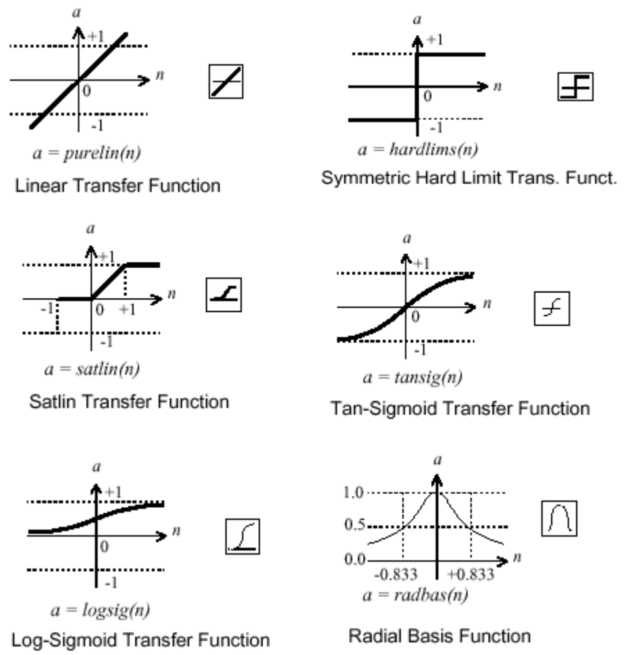


$$a = f(p_1w_1 + p_2w_2 + p_3w_3 + b) = f\left(\sum p_iw_i + b\right)$$

Each neuron within the network is usually a simple processing unit which takes one or more inputs and produces an output. At each neuron, every input has an associated weight which modifies the strength of each input. The neuron simply adds together all the inputs and calculates an output to be passed on.

Activation functions

The activation function is generally non-linear. Linear functions are limited because the output is simply proportional to the input.



Training methods

Supervised learning

In supervised training, both the inputs and the outputs are provided. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights which control the network. This process occurs over and over as the weights are continually tweaked. The set of data which enables the training is called the training set. During the training of a network the same set of data is processed many times as the connection weights are ever refined.

Example architectures : Multilayer perceptrons

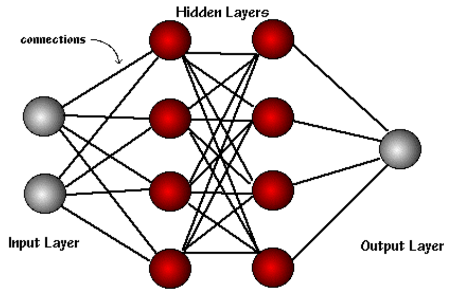
Unsupervised learning

In unsupervised training, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. This is often referred to as self-organization or adaption.

Example architectures : Kohonen, ART

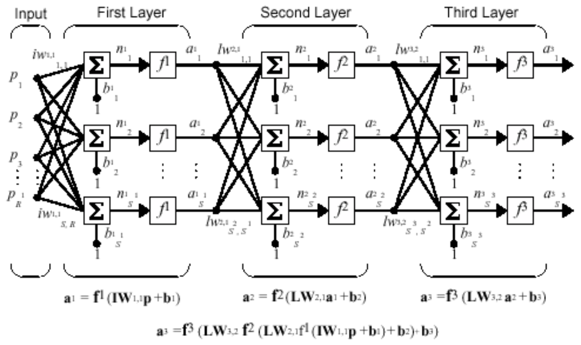
Feedforword NNs

The basic structure off a feedforward Neural Network



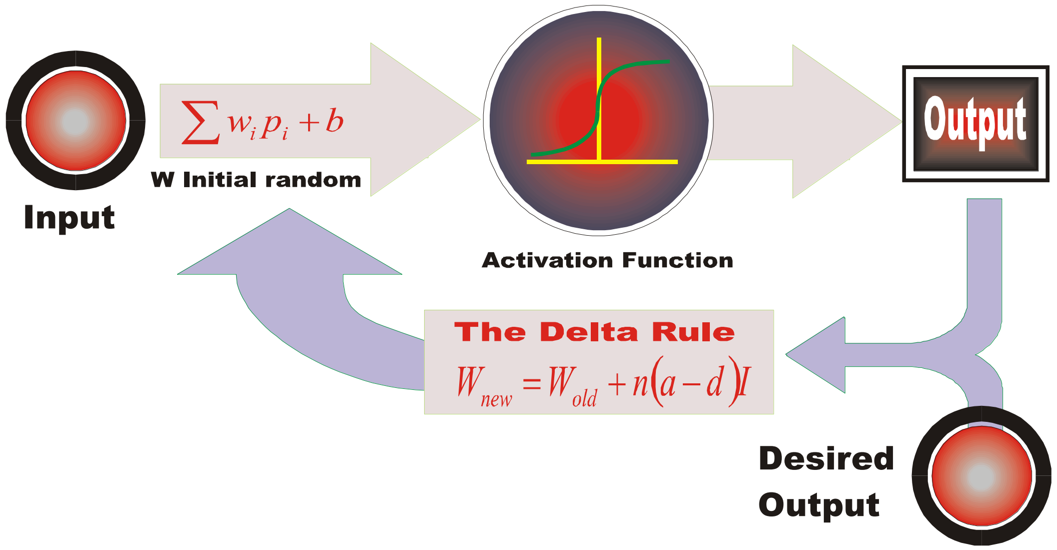
The learning rule modifies the weights according to the input patterns that it is presented with. In a sense, ANNs learn by example as do their biological counterparts.

When the desired output are known we have supervised learning or learning with a teacher.



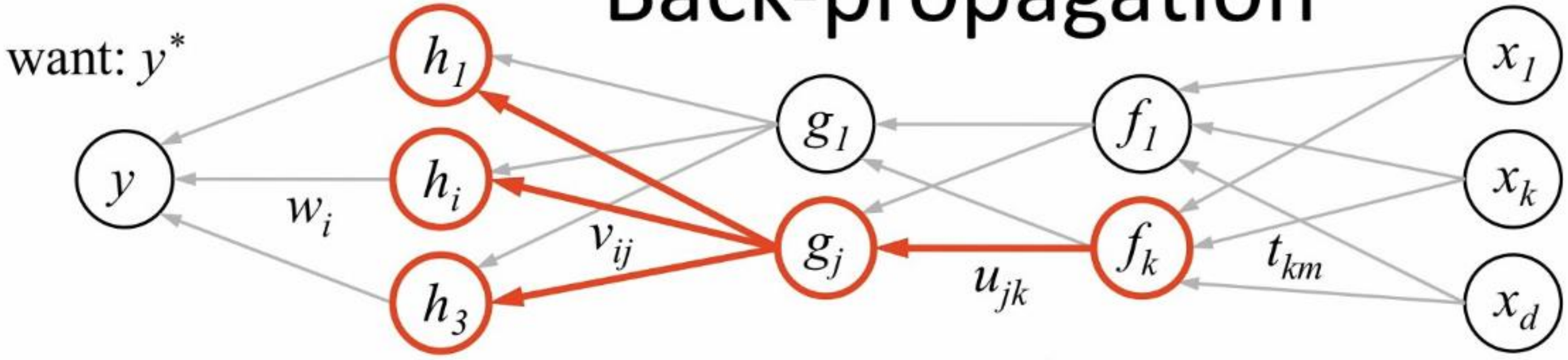
The Learning Rule

The delta rule is often utilized by the most common class of ANNs called backpropagational neural networks.



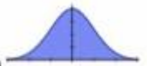
When a neural network is initially presented with a pattern it makes a random guess as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights.

Back-propagation



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y-y^*)$
4. **back-propagate error:** for each unit g_j in each layer L…1

(a) compute error on g_j



$$\underbrace{\frac{\partial E}{\partial g_j}}_{\text{should } g_j \text{ be higher or lower?}} = \sum_i \underbrace{\sigma'(h_i) v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

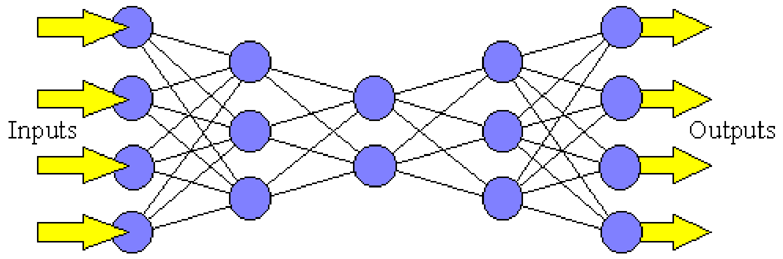
$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

Auto-associative NNs

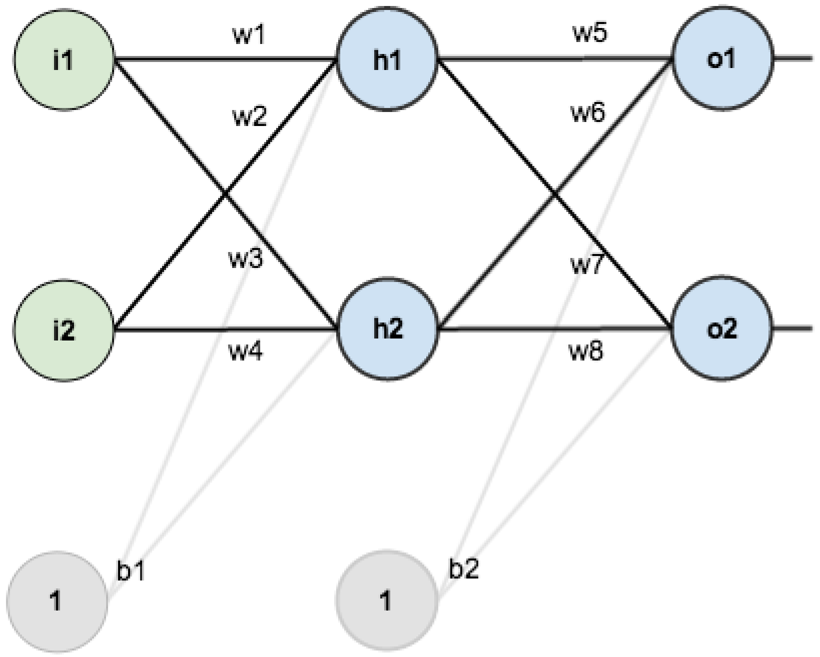
The auto-associative neural network is a special kind of MLP - in fact, it normally consists of two MLP networks connected "back to back“. The other distinguishing feature of auto-associative networks is that they are trained with a target data set that is identical to the input data set.



In training, the network weights are adjusted until the outputs match the inputs, and the values assigned to the weights reflect the relationships between the various input data elements. This property is useful in, for example, data validation: when invalid data is presented to the trained neural network, the learned relationships no longer hold and it is unable to reproduce the correct output. Ideally, the match between the actual and correct outputs would reflect the closeness of the invalid data to valid values. Auto-associative neural networks are also used in data compression applications.

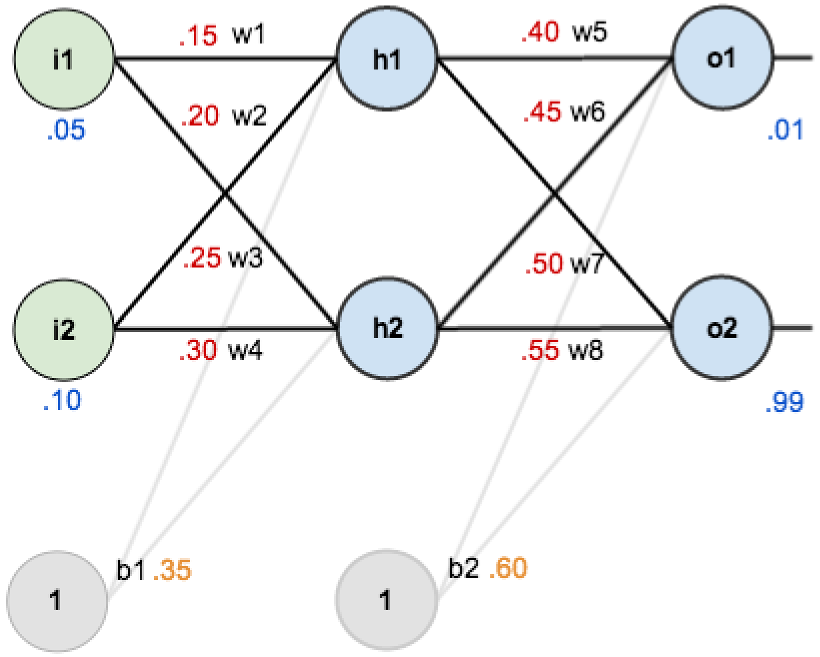
Example

We're going to use a neural network with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.



Example

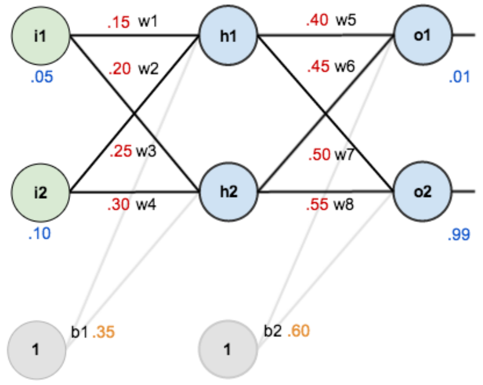
In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



Example

The Forward Pass

We figure out the total net input to each hidden layer neuron, squash the total net input using an activation function (here we use the logistic function), then repeat the process with the output layer neurons.



$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

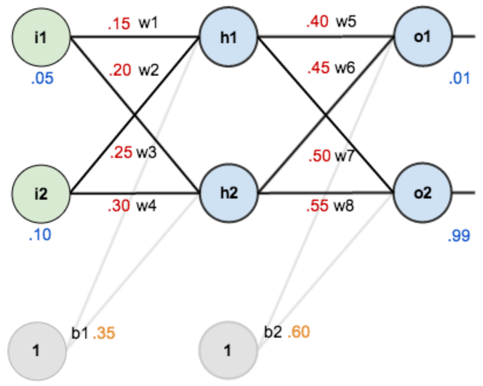
$$out_{h2} = 0.596884378$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$



Example

Calculating the Total Error

We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

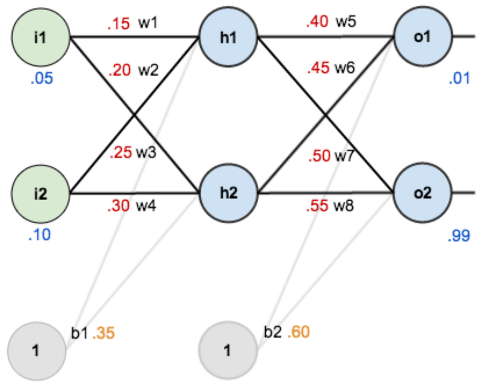
The target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$



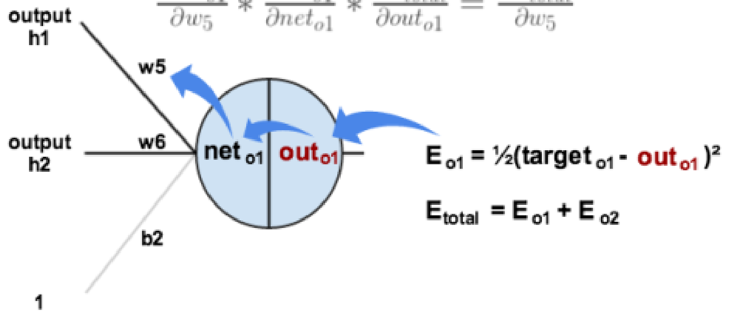
Example

The Backwards Pass

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$E_{o1} = ½(target_{o1} - out_{o1})^2$

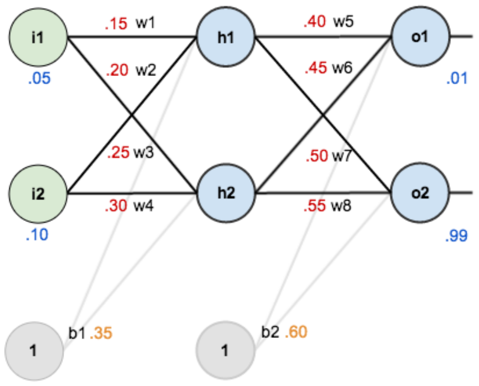
$E_{total} = E_{o1} + E_{o2}$

First, how much does the total error change with respect to the output?

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$



Example

Next, how much does the output of *o*$_1$ change with respect to its total net input?

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Finally, how much does the total net input of o$_1$ change with respect to w$_5$?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Example

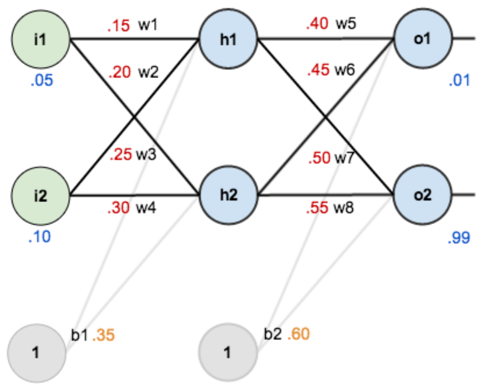
We can repeat this process to get the new weights w6, w7, and w8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

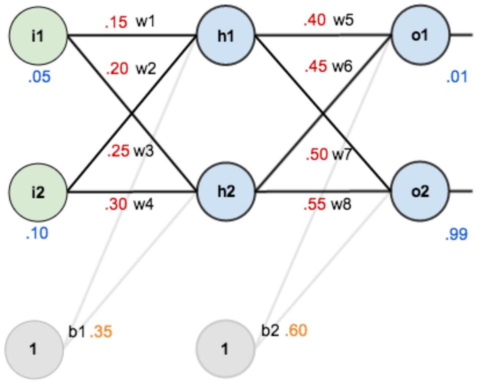
We perform the actual updates in the neural network after we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).



Example

Hidden Layer

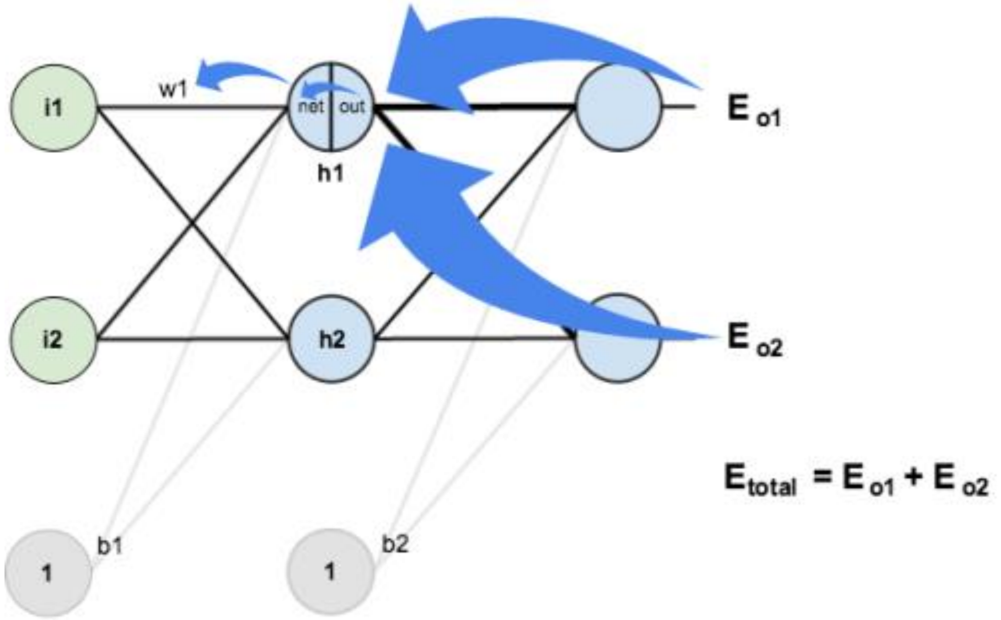
Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3, and w_4.



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

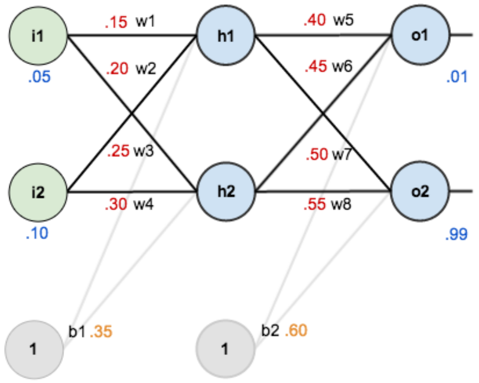


Example

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

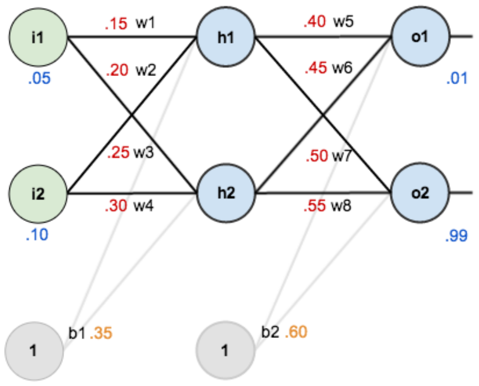
$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$



Example

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

We can now update w_1:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Example 2

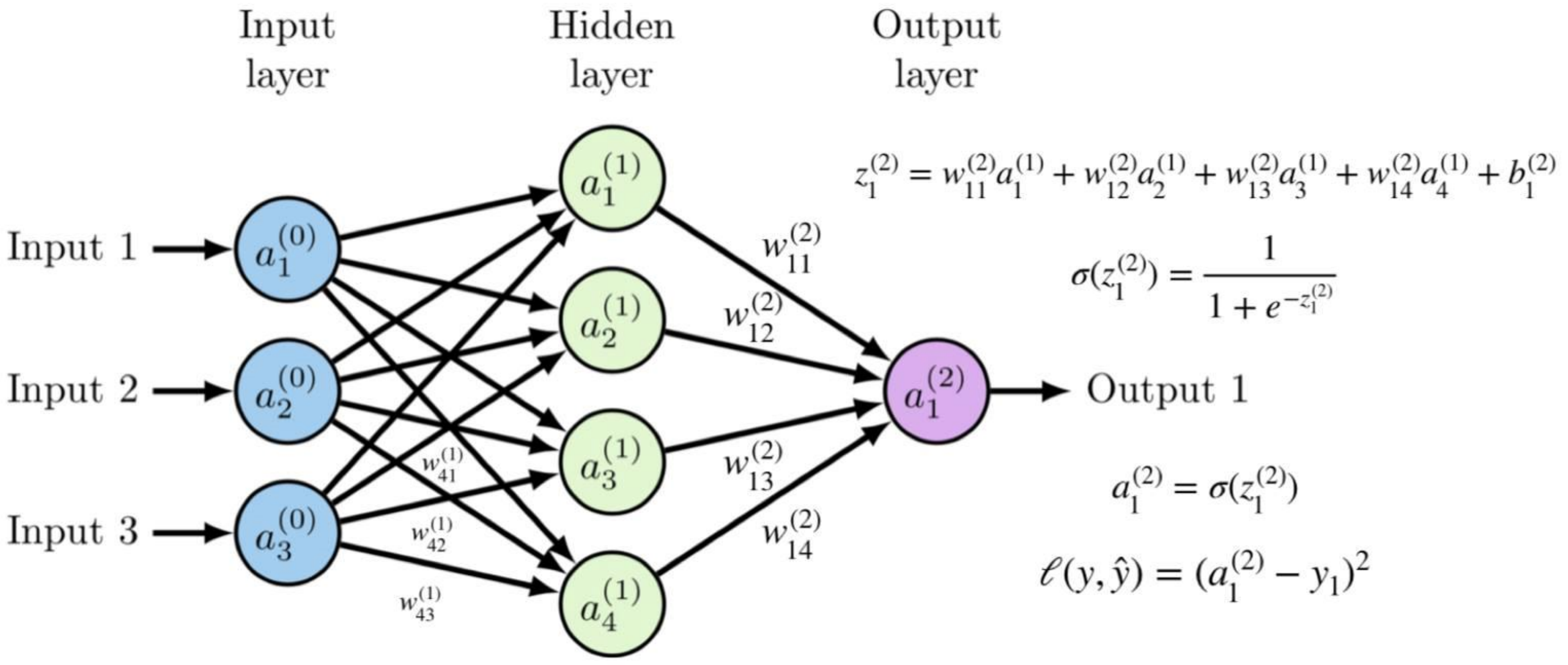
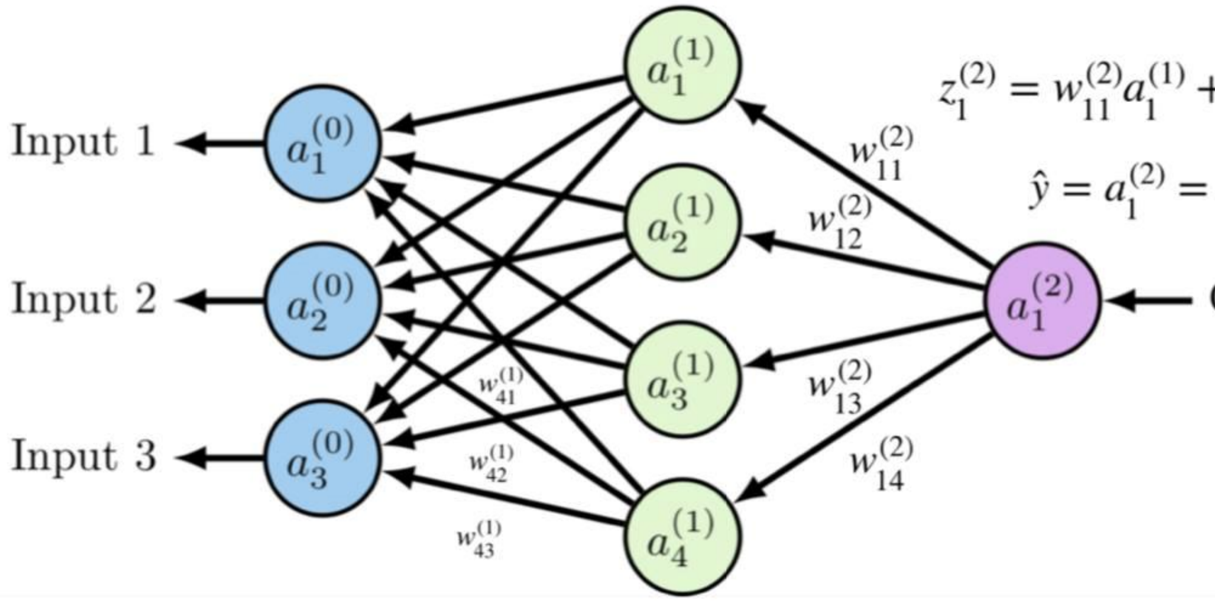


Figure 1: **Multilayer perceptron (MLP)**.

Example 2



$$z_1^{(2)} = w_{11}^{(2)}a_1^{(1)} + w_{12}^{(2)}a_2^{(1)} + w_{13}^{(2)}a_3^{(1)} + w_{14}^{(2)}a_4^{(1)} + b_1^{(2)}$$

$$\hat{y} = a_1^{(2)} = \sigma(z_1^{(2)}) = \sigma[W_2\sigma[W_1\mathbf{a}_0 + b_1] + b_2]$$

$$\ell(y, \hat{y}) = (a_1^{(2)} - y_1)^2$$

$$\sigma(z_1^{(2)}) = \frac{1}{1 + e^{-z_1^{(2)}}}$$

$$\frac{\delta\ell(y,\hat{y})}{\delta a_3^{(1)}} = \frac{\delta\ell(y,\hat{y})}{\delta a_1^{(2)}}\frac{\delta a_1^{(2)}}{\delta z_1^{(2)}}\frac{\delta z_1^{(2)}}{\delta a_3^{(1)}} = 2(a_1^{(2)} - y_1)z_1^{(2)}(1 - z_1^{(2)})w_{13}^{(2)}$$

Figure 2: **All-purpose parameter fitting: Backpropagation**.

Any Questions?

✉ hvusynh@hcmiu.edu.vn