AI Neural Networks: Backpropagation

Margrit Betke

September 2024

Backpropagation is the name of the neural net training algorithm.

Measuring Performance P

Performance P = negative loss function
(see previous slides on loss functions)

Input sample = $\vec{x}_s$

Desired output = Label of sample = $\vec{d}_s$

Computed output = $\vec{o}_s$

Z = # components of output vector

M = # training samples

P = Sum over all samples and output components of the squared error per labeled training sample $s = \{\vec{x}_s, \vec{d}_s\}$

$$P = -\sum_{s=1}^{M}\sum_{z=1}^{Z}(d_{s,z} - o_{s,z})^2$$

In Acquaintance/Sibling network:

P = negative Root Mean Squared (RMS) error

Main Ingredients for Training Algorithm

P: Performance of neural net (depends on $\vec{o}$)

$\vec{o}$: output vector of a node (depends on $\vec{w}$)

$\vec{w}$: weight vector for node

Most general chain rule:

$$P : \mathbb{R}^W \to \mathbb{R}^J$$

Change in performance P when adjusting ith weight w_i during training:

$$\frac{\partial P}{\partial w_i} = \frac{\partial P(\vec{o}(\vec{w}))}{\partial w_i} = \sum_{j=1}^{J} \frac{\partial P(\vec{o})}{\partial o_j} \cdot \frac{\partial o_j(\vec{w})}{\partial w_i}$$

Example: $o : \mathbb{R}^3 \to \mathbb{R}^2$ and $P : \mathbb{R}^2 \to \mathbb{R}^1$

$P(\vec{o}) = P(o_1, o_2) = 3o_1 - 7o_2$

$o_1(\vec{w}) = w_1 - 2w_2 + 5w_3$

$o_2(\vec{w}) = 2w_1 - w_2 - 6$

$\frac{\partial P}{\partial w_1} =$

$\frac{\partial P}{\partial w_2} =$

$\frac{\partial P}{\partial w_3} =$

Interpretation:

Gradient $= \nabla P(\vec{w}) = (\Delta w_1, \Delta w_2, \Delta w_3)^T = (-11, 1, 15)^T$

Symbol Delta Δ is for Difference or Change

Solution space for P? Can only draw 2D:

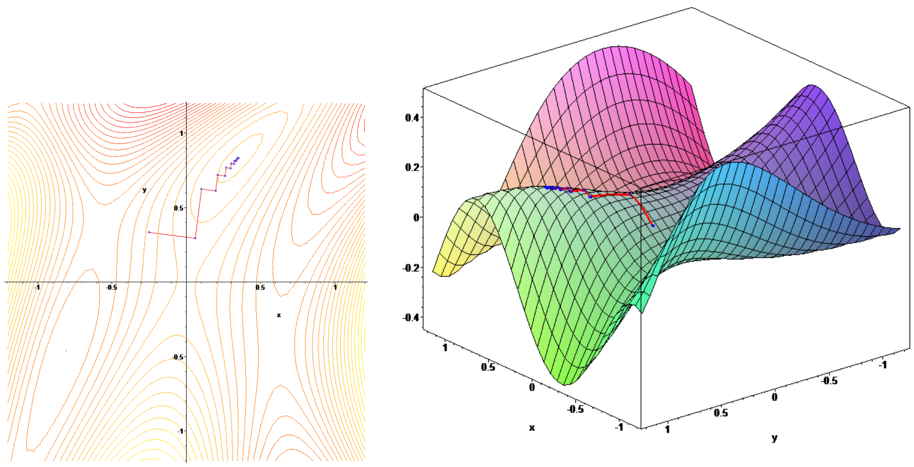


Figure 1: Solution space of P if P is more complicated than in our example [Source of images: Wikipedia.]

Climbing uphill, following gradient direction $(1, 15)^T$

Update Rule of Backprop Algorithm:

$$w_{1,new} = w_{1,old} + \Delta w_1 = w_{1,old} - 11$$

$$w_{2,new} = w_{2,old} + \Delta w_2 = w_{2,old} + 1$$

$$w_{3,new} = w_{3,old} + \Delta w_3 = w_{3,old} + 15$$

Adjust weights in each iteration of back prop **proportional** to the gradient length. Here the **learning rate** $r = 1$. If $r = 2$, the update rules above would add $r\Delta\vec{w} = (-22, 2, 30)$ to the old weight estimates.

Backpropagation Neural Net Training Algorithm

Input: NN structure (# nodes, # layers),
Labeled training data = input/output pairs $\{\vec{x}, \vec{o}_{desired}\}$

1) Choose weights randomly (or some other way)

2) Compute performance P on training data

3) WHILE performance P not satisfactory:

{ FOR EACH input vector $\vec{x}$:

{

Compute $\vec{o}_{last}$ (= evaluate NN)

Compute β_j's (explained later)

Compute weight changes: $\Delta w_{i \to j} = \ r\ o_i\ o_j(1 - o_j)\ \beta_j$

}

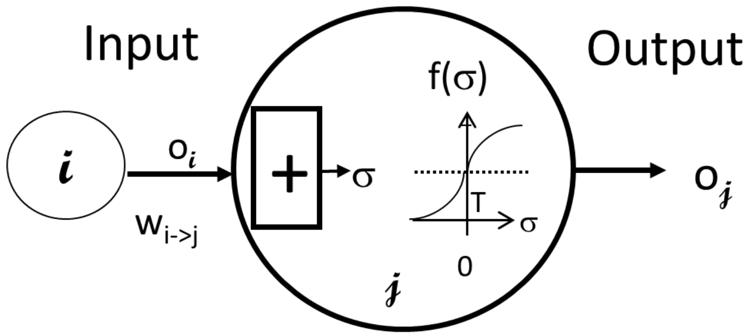
Add all Δw's computed for *all* input vectors

}

Output: Weights (= trained neural net)
Performance P on trained net

Some remarks on training:

Above, we showed how to compute $\Delta w_{i\to j}$ for an example with specific linear functions P and $\vec{o}$. Now we need to derive a general solution, which will lead us to the update rule $\Delta w_{i\to j} = r\, o_i\, o_j(1-o_j)\,\beta_j$



What is the change in the output o_j when weight $w_{i\to j}$ is adjusted?
This means: What is the following?

$$\frac{\partial o_j(\sigma(\vec{w}))}{\partial w_{i\to j}} = \frac{d o_j(\sigma)}{d\sigma} \cdot \frac{\partial \sigma(\vec{w})}{\partial w_{i\to j}}. \tag{1}$$

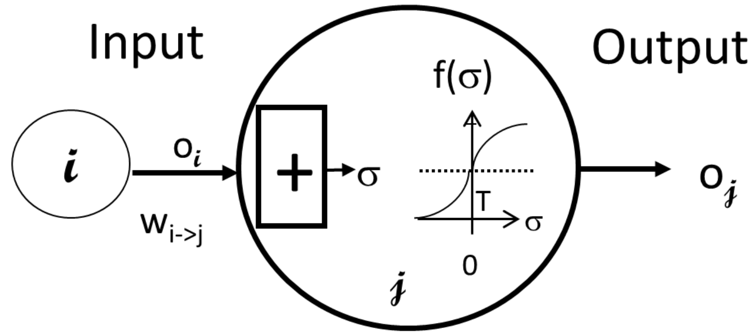
A few explanations:

- Here is the general chain rule again that we used for our performance function P:
$$\frac{\partial P(\vec{o}(\vec{w}))}{\partial w_i} = \sum_{z=1}^{Z} \frac{\partial P(\vec{o})}{\partial o_z} \cdot \frac{\partial o_z(\vec{w})}{\partial w_i}$$
- No $\sum$ in the chain rule applied to a single node because o_j is a scalar.
- "Regular" derivative notation d and not ∂ because σ is a scalar and so the output o_j of node j is a function dependent on a single variable σ.
- Note that $\sigma(\vec{w}) =$

Let us solve Equation (1) above in two steps:
1) What is

$$\frac{\partial \sigma(\vec{w})}{\partial w_{i\to j}} \quad =$$

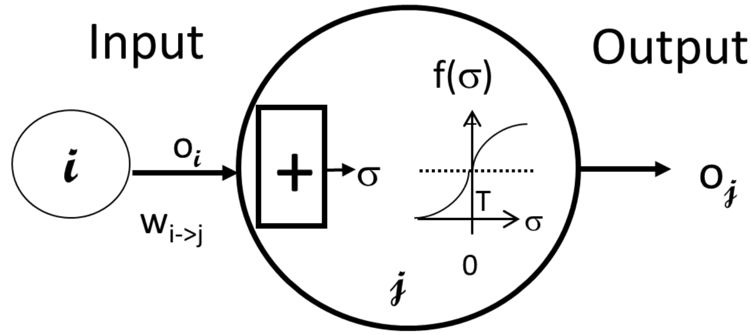
2) What is $\frac{d o_j(\sigma)}{d\sigma}$? This depends on the type of activation function we use.



We here use a sigmoid function for $f(\sigma)$:

$$o_j(\sigma) = f(\sigma) = \frac{1}{1+e^{-\sigma}}$$

$$\frac{d\,f}{d\,\sigma} =$$



Now we have the answer to: What is the change in the output o_j when weight $w_{i \to j}$ is changed?

$$\frac{\partial o_j(\sigma(\vec{w}))}{\partial w_{i \to j}} = \frac{d o_j(\sigma)}{d\sigma} \cdot \frac{\partial \sigma(\vec{w})}{\partial w_{i \to j}} =$$

Backpropagation Neural Net Training Algorithm

Input: NN structure (# nodes, # layers),
Labeled training data = input/output pairs $\{\vec{x}, \vec{o}_{desired}\}$
1) Choose weights randomly (or some other way)
2) Compute performance P on training data
3) WHILE performance P not satisfactory:
{ FOR EACH input vector $\vec{x}$:
{
Compute $\vec{o}_{last}$ (= evaluate NN)
Compute β_j's (explained later)
Compute weight changes: $\Delta w_{i \to j} = r\, o_i\, o_j(1 - o_j)\, \beta_j$
}
Add all Δw's computed for all input vectors

}
Output: Weights (= trained neural net)
Performance P on trained net

Missing piece? Backpropagation of β's.

$$\frac{\partial\, P}{\partial w_{i\to j}} = \frac{\partial P(\vec{o}(\vec{w}))}{\partial w_{i\to j}} = \sum_{j=1}^{J} \frac{\partial P(\vec{o})}{\partial o_j} \cdot \frac{\partial o_j(\vec{w})}{\partial w_i} = \sum_{j=1}^{J} o_j\,(1 - o_j)\, o_i \quad \beta_j$$

How to compute the β's:

Last network layer:

Performance $P(\vec{o}_{last}) =$

Change in performance $\frac{\partial P(\vec{o}_{last})}{\partial o_z} =$

Earlier network layer:

$$\frac{\partial P(\vec{o}_{last})}{\partial o_j} = \sum_{k=1}^{K} \frac{\partial P(\vec{o}_{last})}{\partial o_k} \cdot \frac{\partial o_k}{\partial o_j}$$

Simplify notation:

$\frac{\partial o_k}{\partial o_j} =$

$$\frac{\partial o_k}{\partial o_j} =$$

Equations in Backpropagation Algorithm now fully derived.

Patrick Winston's book: pp. 453-457, 458-468