Notes on General Learning

ERMO WEI

Contents

Activation Function Activation Function are important for neural network to have non-linearity. The commonly used activation function are summarize below. Most of them are typically combined with an affine transformation $\mathbf{a} = \mathbf{b} + \mathbf{W}\mathbf{x}$ and applied element-wise.

$$\mathbf{h} = f(\mathbf{a}) \rightarrow h_i = f(a_i) = f(b_i + \mathbf{W}_{\mathbf{i},:} * \mathbf{x})$$

- Sigmoid

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

- Hyperbolic tangent

$$f(a) = \tanh(a)$$

- Softmax
- Rectifier or rectified linear unit (ReLU)

$$f(a) = \max(0, a)$$

- Maxout
- Softplus
  A smooth version of the rectifier, basically the same shape. But this function has differentiability and non-zero derivative everywhere.

$$f(a) = \log(1 + e^a)$$

Back Propagation Let's start with linear neurons (also called linear filters). Here we first make two assumptions:

- The neuron has a real- valued output which is a weighted sum of its inputs
- The aim of learning is to minimize the square error summed over all training cases.

The model of the linear neuron is like this :

$$y = \sum w_i x_i = \mathbf{w}^T \mathbf{x}$$

Since we are dealing with a simple linear neuron, we first derive the special case of back propagation algorithm, called *delta rule*, which is a gradient descent learning rule for updating the weights of single layer neural network.

1. We first define the error of our training cases:

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

2. Now we take the derivative of the error with respect to the weights

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{1}{2} \sum_n \frac{dE^n}{dy^n} \frac{\partial y^n}{\partial w_i} \\ &= -\sum_n x_i^n (t^n - y^n) \end{aligned}$$

3. The **batch** delta rule changes the weights in proportion to their error derivatives summed over all training cases

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

One thing to be notice here, we can get as close as we desire to the best answer by making the learning rate small enough

Ok, since we have the simplest neuron, let's making things little interesting by adding the logistic function. And now, our model become this:

$$z = b + \sum_i w_i x_i$$

$$y = \frac{1}{1 + e^{-z}}$$

their corresponding derivatives are

$$\frac{\partial z}{\partial w_i} = x_i \, , \frac{\partial z}{\partial x_i} = w_i$$

$$\frac{dy}{dz} = y(1 - y)$$

Thus, take the derivative of y with respect to w_i is

$$\frac{\partial y}{\partial w_i} = \frac{dy}{dz} \frac{\partial z}{\partial w_i} = x_i y(1 - y)$$

and

$$\frac{\partial E}{\partial w_i} = \sum \frac{\partial E}{\partial y^n} \frac{\partial y^n}{\partial w_i} = -\sum_n x_i^n y^n (1 - y^n)(t^n - y^n)$$

In this equation, the first and last terms come from the delta rule, and the term in the middle is the slope of logistic

Still working on it ...

Long-Short Term Memory Still working on it ...

Universal Approximator Accroding to (Hornik et al., 1989), Multilayer feedforward networks are universal approximators. Single hidden layer feedforward networks can approximate any measurable function arbitrarily well regardless of the activation function Ψ, the dimension of the input space r, and the input space environment μ

1. universal approximators: standard multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy
2. there are no theoretical constraints for the success of feedforward networks
3. lack of success is due to inadequate learning, insufficient number of hidden units or the lack of a deterministic relationship between input and target
4. rate of convergence as the number of hidden units grows
5. rate of increase of the number of hidden units as the input dimension increases for a fixed accuracy

References

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.