Mathematics for Deep Learning

Forward Step

At one single layer, we have mathematically two equation: the transfoer equation and the activation function (1)

transfer function $\sum_i w_i x_i + b \,,\ z = W^T x$

activation function $y = sigma(z)$

BackStep

backstep error $E = \frac{1}{2} \sum_i (y_i - t_i)^2$

weight updates $wij \leftarrow wij - \nabla wij$

gradient computed using partial derivates – hyperparameters nano (n), the learning rate to account for how large a step should be in direction opposite to the gradient

$$\nabla w = -n \frac{\partial E}{\partial wij}$$

- z_i *is the input to the nodej for layer i*
- $y_i = sigmaj(z_i)$ *the output of the activation of node j in ;ayer i*
- *wij is the matrix of weights connecting neuron i in layer l − 1 to neuron j in layer l*
- *bij is the bias for unit j in layer l*
- *to is the target value for node o in the output layer*

Compute Partial derivates for the error at the output layer ∂E when the weights are changed by ∂wij. There are two different cases:
- *Case 1: Weight update equations for a neuron from hidden (input) layer to output layer*
- *Case 2: Weight update equation from a neuron from hidden (input) layer to hidden layer* (2)

Case 1: From hidden layer to the output layer

$$\frac{\partial E}{\partial w_{jo}} = \frac{\partial \frac{1}{2} \sum_o (y_o - t_o)^2}{\partial w_{jo}} = (y_o - t_o) \frac{\partial (y_o - t_o)}{\partial w_{jo}}$$

Weight updates for each hidden connections $w_{jo} \leftarrow w_{jo} - n \frac{\partial E}{\partial w_{jo}}$

Gradient with respect to the output layer biases $\frac{\partial z_o}{\partial b_o} = \frac{\partial \sum_j w_{jo} sigma'_j(z_j) + b_o}{b_o} = 1$

$$\frac{\partial z_o}{\partial b_o} = v_o$$

Case 2 : From hidden layer to the hidden layer

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial \frac{1}{2}\sum_o (y_o - t_o)^2}{\partial w_{ij}} = \sum_o (y_o - t_o)\frac{\partial (y_o - t_o)}{\partial w_{ij}}$$

Weight updates for each hidden connections $w_{ij} \leftarrow w_{ij} - n\frac{\partial E}{\partial w_{ij}}$

Gradient with respect to the output layer biases

$$\frac{\partial z_o}{\partial b_o} = \sum_o (y_o - t_o)\, sigma'_o(z_o)\, w_{jo} sigma'_o(z_j) y_i$$

$$= y_i\, sigma'_o(z_j) \sum_o v_o w_{jo}$$

– *compute the feedforward signals from the input to the output*
– *compute the output error E based on the predictions of* y_o *and the value of true* t_o
– *backpropagate the error signals, multiply them with the weights in previous layers*
– *compute the gradients* $\frac{\partial E}{\partial theta}$ *for all the parameters of theta based on the backpropagated error signals and the feedforward signals from the inputs*
– *update the parameters using the computed gradients* $theta \leftarrow theta - \frac{\partial E}{\partial theta}$

Cross entropy and its derivative

Gradient descnet is used when cross – entropy is adopted as the loss function.
TF logistic function defined

$$E = L(c, p) = -\sum_i [c_i\, ln(p_i) + (1 - c_i) ln(1 - p_i)]$$

where c is the one – hot – encoded classes (or labels) and p is softmax applied probabilities

$$\frac{\partial E}{\partial score_i} = \frac{\partial E}{\partial p_i}\frac{\partial p_i}{\partial score_i}$$

computing each part separately, and combining the results, we have $\frac{\partial E}{\partial score_i} = p_i - c_i$

Batch Gradient descent

Adjust the weights such as loss function is minimized; represent loss function in the sum of all loss functions commonly used

$$Q(w) = \frac{1}{n}\sum_{i=1-n} Q_i(w)$$

Perform derivation steps very similar with update rules, where nano is the learning rate and the ∇ *is the gradient*

$$w = w - nano\nabla Q(w) = w - nano \sum_{i=1-n} \nabla Q_i(w)$$