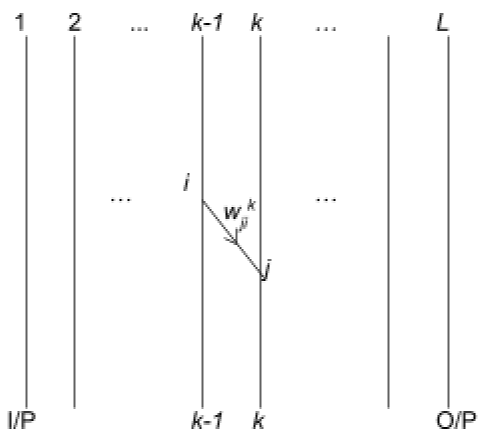
CSE 4502/5717 Big Data Analytics
Lecture #19, 4/9/2018

with Dr. Sanguthevar Rajasekaran
Notes from Katherine Riedling

<u>Recap: A generic neural network (NN)</u>



The activation value at node *j* of level *k* is denoted as a_j^k . Let the number of nodes in level *k* be n_k . The weighted input for node j of level k is denoted as z_j^k, for all *j* and *k*.

$$a_j^k = \sigma(z_j^k) = \sigma(\Sigma_{i=1}^{n_{k-1}} w_{ji}^k a_i^{k-1}).$$

We can use a matrix-vector multiplication to get a_j^k , for every *j*. We demonstrated this last time.

If $\boldsymbol{a^k} = (a_1{}^k, a_2{}^k, \cdots, a_{n_k}{}^k)^T$,

and $W^k =$

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n\_(k-1)} \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ w_{n\_k\,1} & w_{n\_k\,2} & \cdots & w_{n\_k\,n\_(k-1)} \end{bmatrix}$$

then $\boldsymbol{a^k} = \boldsymbol{W^k a^{k-1}} + \boldsymbol{b^k}$

Total time taken in the forward propagation =

$$O(\sum_{k=2}^{L} n_{k-1} n_k).$$

or O(|V|+|E|), if our neural network G(V,E) is **sparse.**

<u>COMPUTING THE GRADIENT</u>

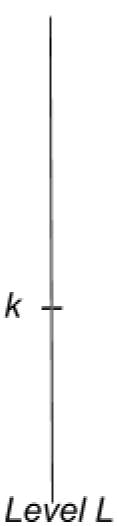
We make two primary assumptions as follows:

1. The cost function can be summed over examples, i.e.,

$$C = \sum_x C_x$$

2. C is a function of $\boldsymbol{a}^L$.

<u>Plan:</u> First compute $\frac{\partial C}{\partial z_i^l}$ for every $2 \le l \le L$ and for every i, $1 \le i \le n_l$.



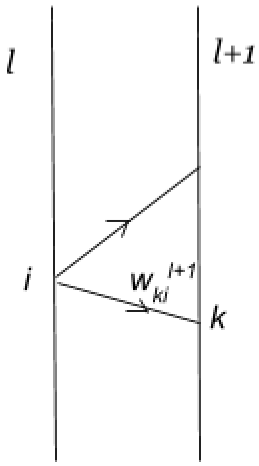
Activation value at node k is not dependent on the weighted input of any other node in the same level. It only depends on the inputs coming into this node.

Followed by this, compute $\frac{\partial C}{\partial b_i^l}$ and $\frac{\partial C}{\partial w_{ik}}$ for every i, k, l. We start from level L and proceed towards level 2.

1. $$\frac{\partial C}{\partial z_i^L} = \Sigma_{k=1}^{n_L} \left(\frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_i^L} \right) = \frac{\partial C}{\partial a_i^L} \text{ X } \frac{\partial a_i^L}{\partial z_i^L} = \frac{\partial C}{\partial a_i^L} \frac{\partial \sigma(z_i^L)}{\partial z_i^L} = \frac{\partial C}{\partial a_i^L} \sigma'(z_i^L)$$

------ (1) ------

Let $\frac{\partial C}{\partial z_i^l} = \delta_i^l$. Assume that we have computed δ_j^{l+1} for $1 \le j \le n_{(l+1)}$. We'll now show how to compute δ_i^l for every i, $1 \le i \le n_l$.



Let *k* be any nodel in level *l+1*; If i is any node in level *l*, *i* affects all the nodes in level *l+1*. So we keep the summation in equation (A) as is and cannot delete any term.

$$\frac{\partial C}{\partial z_i^l} = \Sigma_{k=1}^{n_{l+1}} \left(\frac{\partial C}{\partial z_k^{l+1}} \; \frac{\partial z_k^{l+1}}{\partial z_i^l} \right)$$

----(A) ----

$z_k^{l+1} = \Sigma_{j=1}^{n_l} w_{kj}^{l+1} a_j^l + b_k^{l+1} = \Sigma_{j=1}^{n_l} w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$. Thus,

$$\frac{\partial z_k^{l+1}}{\partial z_i^l} = w_{ki}^{l+1} \sigma'(z_i^l) \text{ ------ (B) -------}$$

Substitute (B) in (A):

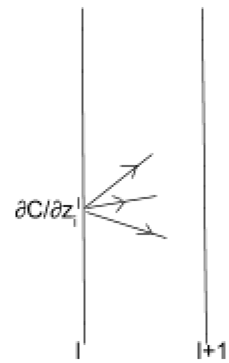
$$\frac{\partial C}{\partial z_i^l} = \Sigma \delta_k^{l+1} * w_{ki}^{l+1} \sigma'(z_i^l) \text{ ------ (2) ------}$$

$\sigma'(z_i^l)$ is easy to compute if we know what activation function is used.
Equation (2) can be thought of as a matrix multiplication.

We see that: $\delta^l = (W^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$ where $\odot$ is the Hadamard multiplication and $\delta^l = (\delta_1^l, \delta_2^l, \cdots, \delta_{n_l}^l)^T$.
If A_{nxn} and B_{nxn} are matrices, then $A \odot B =$

$$\begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \dots & a_{1n}b_{1n} \\ . & & & \\ . & & & \\ . & & & \\ . & & & \\ a_{n1}b_{n1} & a_{n2}b_{n2} & \dots & a_{nn}b_{nn} \end{bmatrix}$$

Time needed to compute $\delta^l = O(n_l n_{l+1})$ (given δ^{l+1}). → We can compute $z_i^l\ \forall l$ and all i in $O(\sum_{l=2}^{L-1} n_l n_{l+1})$ time.



If D_i^l is the out-degree of node i at level l, we can compute: $\frac{\partial C}{\partial z_i^l}$ in $O(D_i^l)$ TIME. → total time for computing δ^l (for every l) = O(|V|+|E|) where G(V,E) is the NN.

3. $\frac{\partial C}{\partial b_i^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial b_i^l} = \delta_i^l \frac{\partial z_i^l}{\partial b_i^l}$

$z_i^l = \sum_{j=1}^{n_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$

$\frac{\partial z_i^l}{\partial b_i^l} = 1 \rightarrow \frac{\partial C}{\partial b_i^l} = \delta_i^l$ ----- (3) -----

$\frac{\partial C}{\partial w_{ik}} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ik}} = \delta_i^l a_k^{l-1}$ ----- (4) -----

Put together, the entire back propagation takes O(|E|+|V|) time. We can also employ matrix-vector multiplication in back propagation.
The entire process of training a feed forward network can be summarized as follows:

<u>INPUT:</u> examples $E_1, E_2, \ldots, E_m$.
<u>OUTPUT:</u> a NN model

1. Determine the values for the hyperparameters:
    a. Number of layers
    b. Number of neurons/nodes in each layer
    c. Edges
    d. Number of epochs

e. Learning rate
f. Minibatch size
g. (Note that these are determined empirically.)

2. TRAIN the network:
<u>For</u> every EPOCH <u>do</u>
    Shuffle the input and partition the input into minibatches;
    <u>For</u> every minibatch <u>do</u>
        Do a forward propagation;
        Do a backward propagation;
        Compute $\frac{\partial C}{\partial w}$ for every parameter w;
    If the gradient is close to 0, STOP;
    Compute the average value of $\frac{\partial C}{\partial w}$ over the examples in the minibatch;
    Update the parameter values by using the gradient;
    w = w - $\alpha\frac{\partial C}{\partial w}$ for every parameter *w;*

Total run time = $O(qm(|V|+|E|))$ where q = # of epochs

(M.Nielson 2017) has employed the above algorithm for the recognition of handwritten digits.

Nielson constructed a neural network with one input layer, one hidden layer, and one output layer. There were thirty neurons in the hidden layer. The MNIST dataset has been employed for training as well as testing. The MNIST data has 60,000 examples and 10,000 test points:

Input layer size: 784 nodes, one node per pixel. Each example has an image of size (28*28);
Output layer size: 10 nodes (one for each possible digit)
Hidden layer size: 30 neurons
Number of epochs: 30
$\alpha = 3$
Minibatch size: 10

He got an accuracy of 95.34%.
When the number of neurons in the hidden layer was increased to 100, the accuracy became 96.59%.

When

$\alpha = 0.001$, the accuracy decreased dramatically to 11.39%.

For $\alpha = 100$, the accuracy was even lower at 10.09%.

The conclusion drawn from this experiment was that there is no analysis one could use to figure out optimal values for the hyperparameters. Fine-tuning the values of these parameters is an empirical task. Trial and error dominates this exercise.

Next time we will have a brief introduction to different types of NNs and visit techniques for improving the test accuracy.