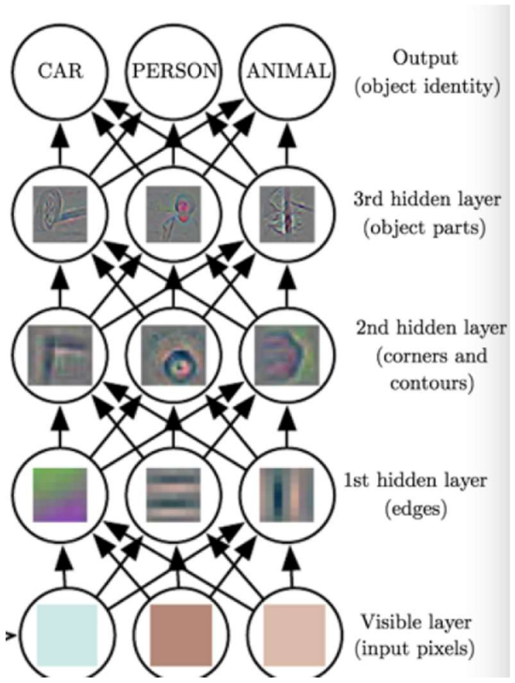
01-04-20-notes

Feedforward neural networks (or multilayer perceptron)

In perceptron NN we have only 2 layers i.e., the input and output layer. In feedforward NN we have hidden layers between the input and output layer which increase the accuracy in prediction.


CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

Each arrow carries weight(**w**) and every circle in hidden layer adds a bias(**b**) to the input variable. Output of every circle is given by a sigmoid function.

Sum(**z**) = $\sum w_i * x_i$ where xi is the input

Output **a** of a circle in hidden layer= g(z)

Where g is the sigmoid function (or any normalizing function)

We want to minimize the square error function given by

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

The resulting function is non convex so convergence is not guaranteed. Also the number of training examples in such a algorithm is very large so normal gradient descent is computationally heavy so we use stochastic gradient descent. Stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee, and is sensitive to the values of the initial parameters. For feedforward neural networks, it is important to initialize all weights to small random values. The biases may be initialized to zero or to small positive values.

Ref

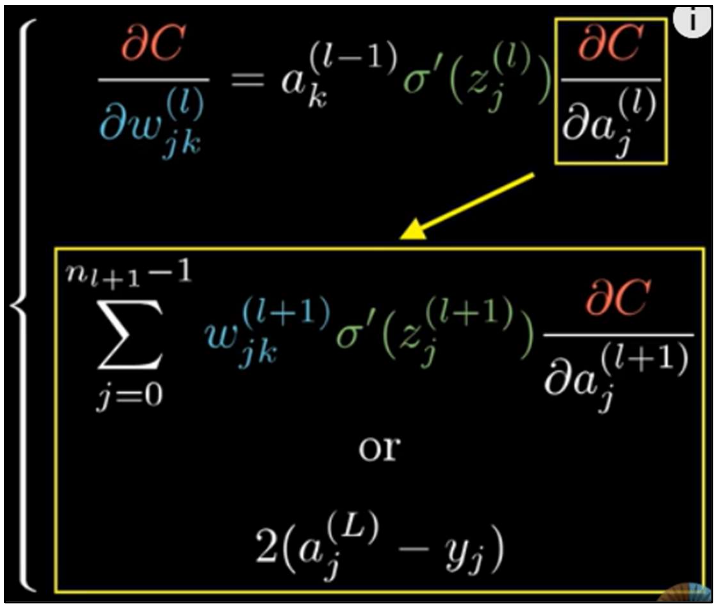
https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7

https://www.youtube.com/watch?v=IHZwWFHWa-w&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=2

Back propagation

We use back propagation algorithm to calculate the partial derivative of cost w.r.t weights by using chain rule.

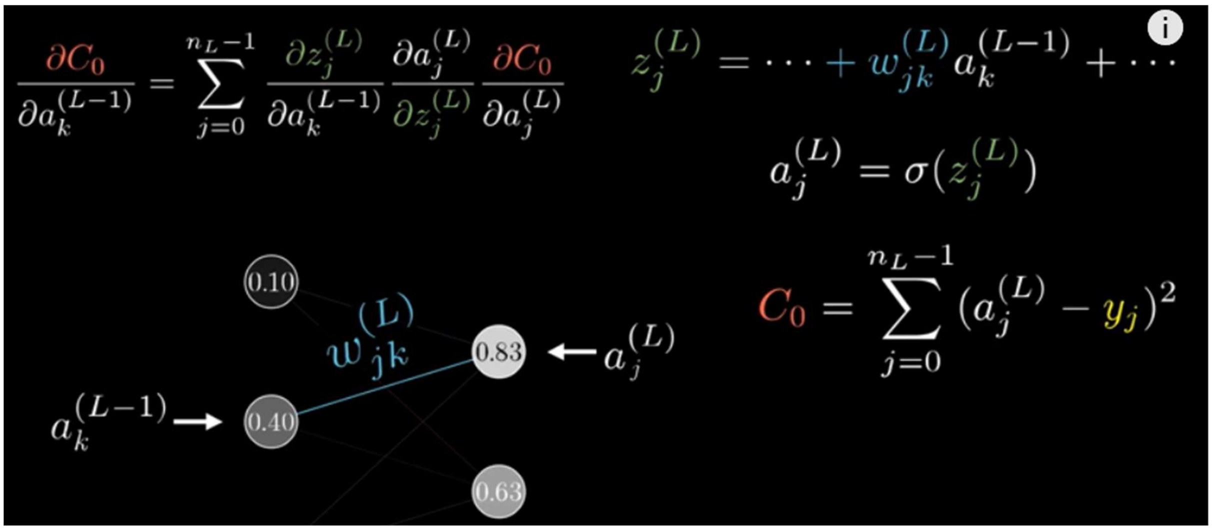
The gradient is given by



$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'\left(z_j^{(l)}\right) \boxed{\frac{\partial C}{\partial a_j^{(l)}}}$$

$$\boxed{\begin{gathered}\sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'\left(z_j^{(l+1)}\right) \frac{\partial C}{\partial a_j^{(l+1)}} \\ \text{or} \\ 2\left(a_j^{(L)} - y_j\right)\end{gathered}}$$

And the terms in the equations are given by



$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

$$a_j^{(L)} = \sigma\left(z_j^{(L)}\right)$$

$$C_0 = \sum_{j=0}^{n_L-1} \left(a_j^{(L)} - y_j\right)^2$$

First we calculate the gradient for the output layer and then we back propagate using these recursive algorithm.

https://www.youtube.com/watch?v=tIeHLnjs5U8&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=4

https://www.youtube.com/watch?v=x_Eamf8MHwU