Fundamentals

- More than one layer is called deep neural networks.

Activation Function

- The author demonstrates how a neural network with weight -2 and a bias of 3 can function as a simple NAND gate, which can then be used to perform basic bit-wise arithmetic operations
- Instead of perceptron, which was presented as motivation, we can use activation function.
  (1) ReLU(rectified linear unit): solve vanishing gradient issue
  (2) Sigmoid: smoothed out version of heaviside function
  (3) Logitic: generalized version of sigmoid
  (4) many variants exist - for example Gaussian function.
- In designing neural network, heuristics help a lot. For example we can reasonably guess why we have ten outpus, instead of four (using binaries).
- Now we are going into the classification problem of MNIST data. For neural network in general, the goal in the end is to minimize the loss function.
- With caveat though - for example we may penalize overly complex model or there might be inbalanced data problem.
-

Vanishing gradient problem

- Small gradient does not necessarily mean that we are close to "local minima." This is more about computational infeasibility.
- For example, tanh activation function has gradients less than 1, so when we have many layers, by chain rule, we'll have exponentially decreasing gradients, which makes our learning almost impossible.

Intuition of Deep Neural Network

- As we build up layers, we ask questions that are more manageable.
- This is analogous to how programming language works.

Backpropagation

- One of the key efficiencies of **backpropagation** comes from the fact that it reuses the computed gradients from later layers in the network to calculate the gradients for earlier layers.
- Choices of activation function actually facilitates learning, so it's important to learn how it works.
- Four fundamental equations
  (1) $\delta^L = \nabla_a C \odot \sigma'(z^L)$: routine chain rule.
  (2) $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$: remember that a forms z.
  (3) $\dfrac{\partial C}{\partial b^l_j} = \delta^l_j$: remember that b forms z.
  (4) $\dfrac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$: remember that w forms z.
- A weight will learn slowly if either the input neuron is low-activation (4), or if the output neuron has saturated (2) (in the case of sigmoid, we have flat surface near 0 and 1).

- The alternative method is finite difference method. But this would require us to compute cost function many more times (we might need to compute for each weight) while back-propagation effectibely compute forward and backward one time each (and backward costs almost the same as forward).

Improving the Neural Network

- Quadratic loss we have $\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x$: note that when σ is sigmoid, we have a stall when σ is close to 0 or 1.
- Benefits of cross entropy: we have $\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z)-y)$
- We do not have a slow-down of backpropagation, since essentially the $a(1-a)$ term is cancelled.
- If output neuron is linear then quadratic cost is an appropriate one.