COST FUNCTIONS AND ACTIVATION FUNCTIONS

Singchern D. You

Sigmoid function

- One widely used activation function is sigmoid

$$y = \frac{1}{1 + \exp(-x)}$$

- Usually x is a product of weights and input vector

Sigmoid function

- A widely used cost function is mean squares errors (quadratic cost function)

$$J = \sum_{\ell} (y_\ell - d_\ell)^2$$

where d_ℓ is the desired output value for output node ℓ

Sigmoid function

- Consider the simplest case: one output node

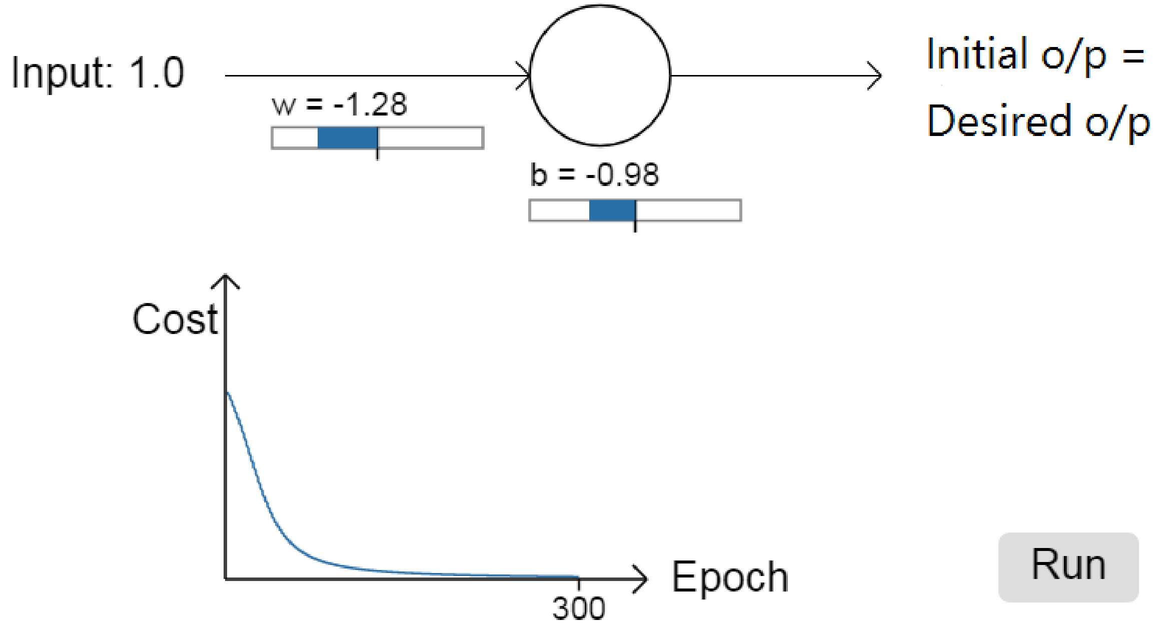
  We know $\frac{d}{dy}J = 2(y-d)\frac{dy}{dx}$

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

Therefore, if $y \to 0$ or $y \to 1$, $\frac{dy}{dx} \to 0$

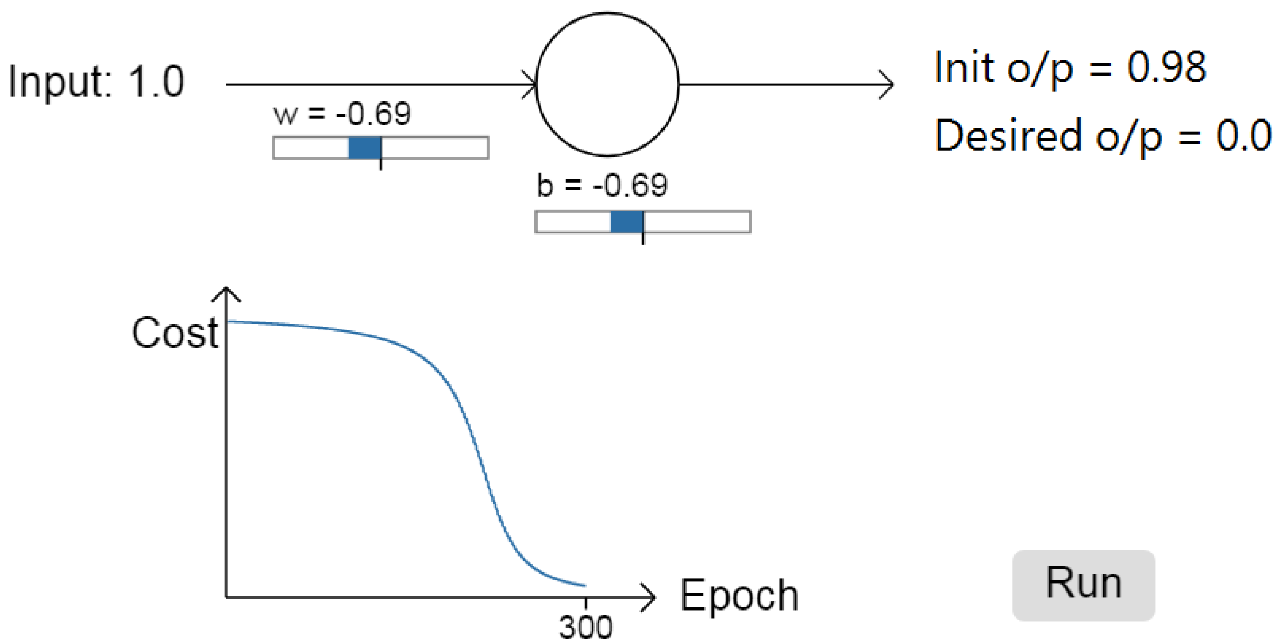
Sigmoid function

- We can see a simple example (one node case)
  - http://neuralnetworksanddeeplearning.com/chap3.html#the_cross-entropy_cost_function



Sigmoid function

- Same example except initial o/p = 0.98



Sigmoid function

- So, we know the gradient becomes very small if output of a node is close to 1 or 0
- Note that $y(1-y)$ has a largest value of 0.25
- Considering that if we have many layers of neurons (like in deep neural networks), $\Delta \boldsymbol{w}$ will be very small

ReLU with quadratic cost

- It can be easily proved that linear function (no sigmoid) does not suffer from the problem we mentioned previously
- That is one of the reasons to use ReLU activation function
- The detailed is left as exercise

Cross-entropy cost function

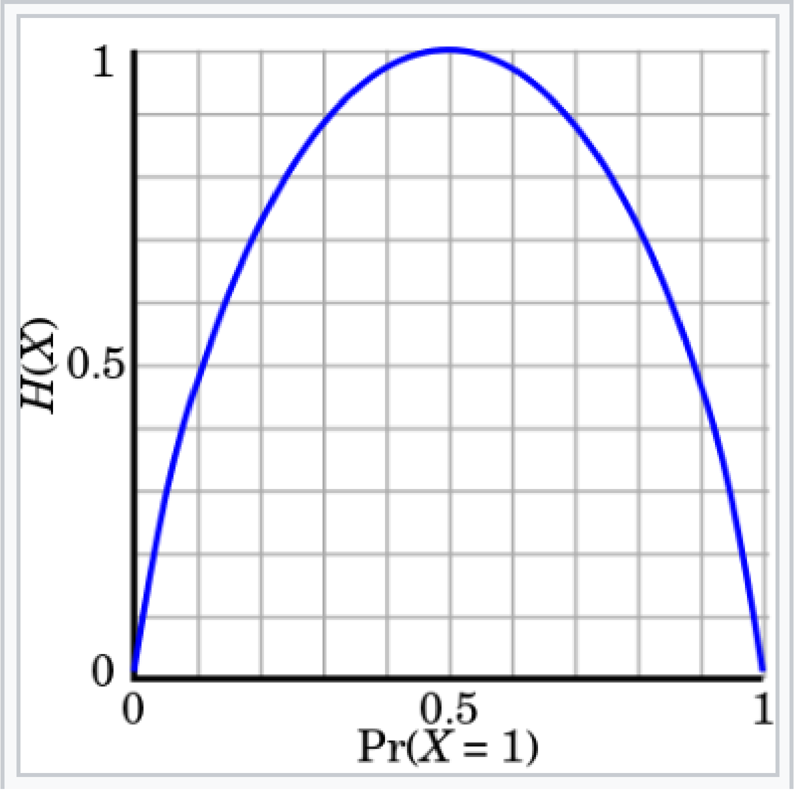
- Another widely used cost function is called cross-entropy cost function
- It is from information theory
- If x is a Bernoulli random variable with P(x=1) = p, then entropy of x is computed as

$$H(\mathrm{x}) = -(p\log_2 p + (1-p)\log_2(1-p))$$

- H(x) is the average number of bits needed to identify an event

Cross-entropy cost function

- Plot of H(x) function is like a (reversed) parabola (from wiki)



Cross-entropy cost function

- If we consider the actual output y and desired output d as probability values, we may use cross-entropy as the cost function
$$J = -(d \log_2 y + (1-d)\log_2(1-y))$$
- We can easily verify that if *d* is large and *y* is small (or vice versa), *J* is large. If both are small or large, *J* is small (we define $0 \log_2 0 = 0$)
- Note: we assume $0 \le d \le 1$ (recall $0 \le y \le 1$)

Cross-entropy cost function

- With this understanding, we confirm that cross-entropy can actually be used as a cost function
- In the case that we have multiple training samples and multiple output nodes, we just need to sum up over all samples and all nodes

Cross-entropy cost function

- Consider the case $y = f(\mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{w}^T\mathbf{x})}$

  where $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$
- With some algebraic work, we have

$$\frac{\partial}{\partial w_j} J = x_j(y - d)$$

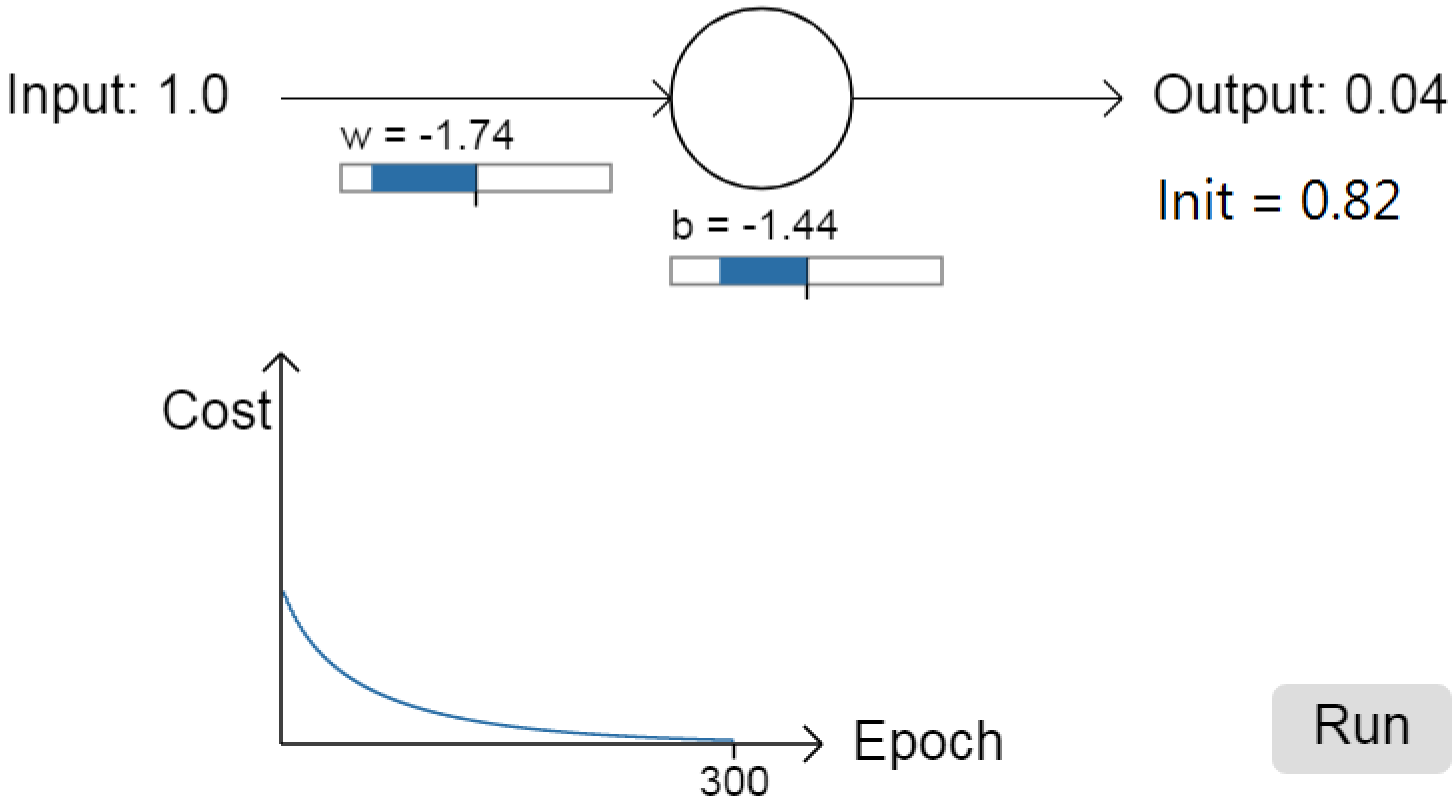
- Again, multiple examples and output modes are omitted in the equation to simplify the discussion

Cross-entropy cost function

- It is observed that the cost function is proportional to the error, i.e., $(y - d)$
- Thus, sometimes it converges faster than the mean squares error (cost function) does

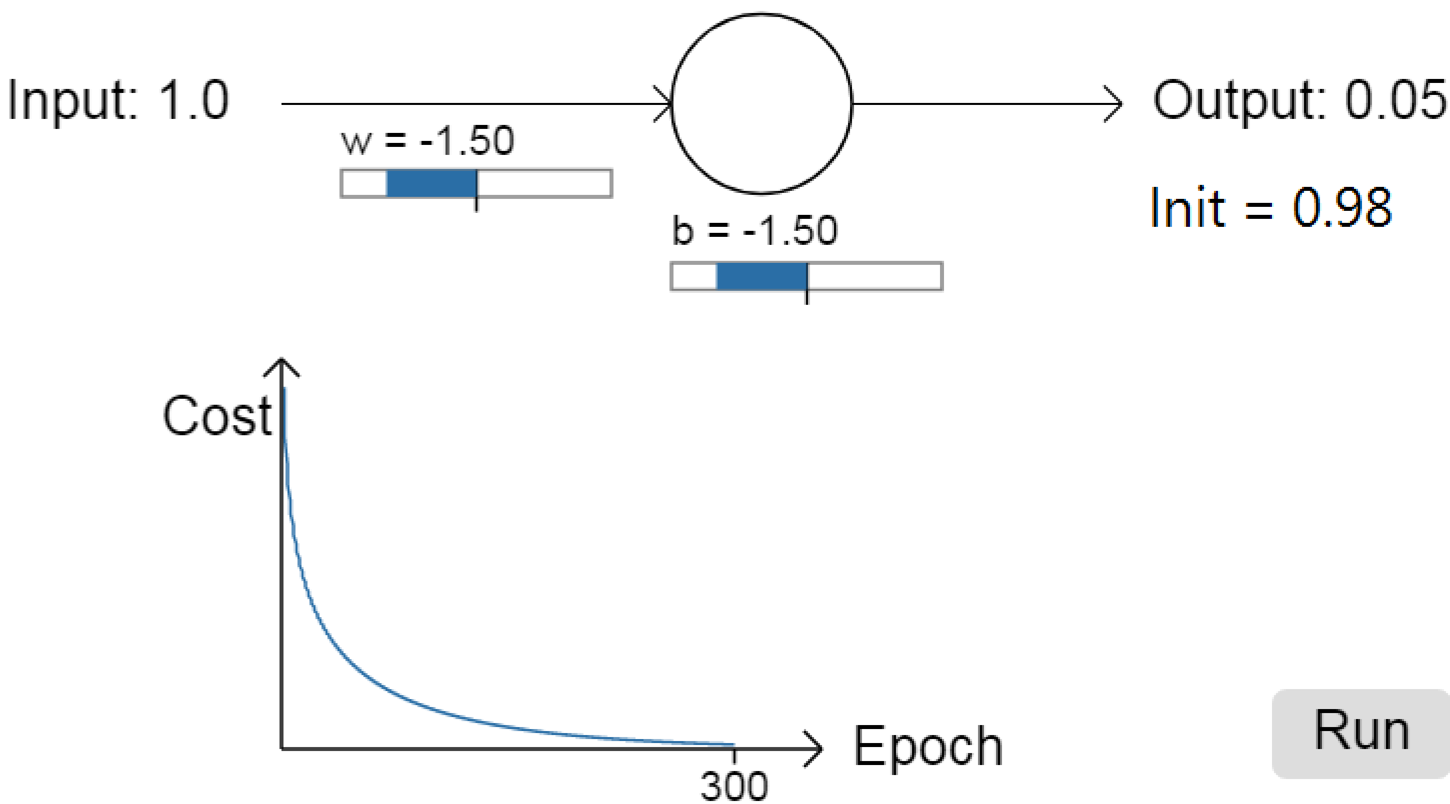
Cross-entropy cost function

- Re-do previous example (init o/p = 0.82)



Cross-entropy cost function

- Init o/p = 0.98



Cross-entropy cost function

- One word of caution when reading the figure: It is the plot of cost function over epoch
- Remember in our present case the cost function is cross-entropy (not mean-square error)
- That means, plots from these two cost functions can not be directly compared (like this one has smaller ultimate cost function, so this one must be better trained …)

Softmax activation function

- Another widely used activation function for nodes at output layer is softmax
- Suppose that we have multiple output nodes with values of $y_1, \ldots, y_m$
- We will use $y_{\ell,(k)}$ to represent output value at output node ℓ at epoch k

Softmax activation function

- Let $z = \boldsymbol{w}^T\mathbf{x}$, we define softmax output for node ℓ as

$$y_\ell = \frac{\exp(z_\ell)}{\sum_{j=1}^{m}\exp(z_j)}$$

- Note: In actual implementation, sometimes exp() function would produce very large values and overflow may occur
- Use a scaling factor to limit these values

Softmax activation function

- The softmax output layer is said to represent the probability of each output class (assuming one class per output neuron)
- In our simulations, it might not be so obvious though
- Why it is called softmax

Softmax activation function

- A nice property of softmax activation function is that its derivative is in simple form
- https://deepnotes.io/softmax-crossentropy
- If $i = \ell, \dfrac{\partial}{\partial z_i} y_\ell = y_\ell (1 - y_\ell)$
- If $i \neq \ell, \dfrac{\partial}{\partial z_i} y_\ell = -y_\ell y_i$

Softmax activation function

- The cross-entropy function for softmax layer is usually chosen as

$$J = -\sum_{\ell} d_\ell \log_2 y_\ell$$

Softmax activation function

- In our previous case, we assume that we have only one output value, and the probability model is Bernoulli (two possible values)
- In the present case, we have multiple output values representing distribution (like prob “one”, “two”, “three”, etc, shown in tossing a dice)
- Thus, we need to sum over all cases with their respective probability values
- In short, the cross-entropy equation is actually the same (though looks different …)

Softmax activation function

- With some algebraic work, we can show that

$$\frac{\partial}{\partial w_j} J = x_j(y_j - d_j)$$

- It is observed that this equation is same as we have previously