PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

PadhAI Week 7: Activation Functions and Initialization Methods

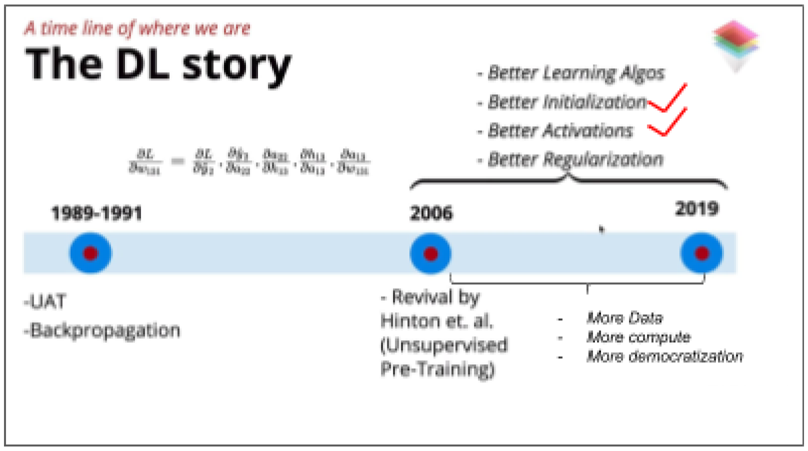
by Manick Vennimalai

7.1: Activation Functions

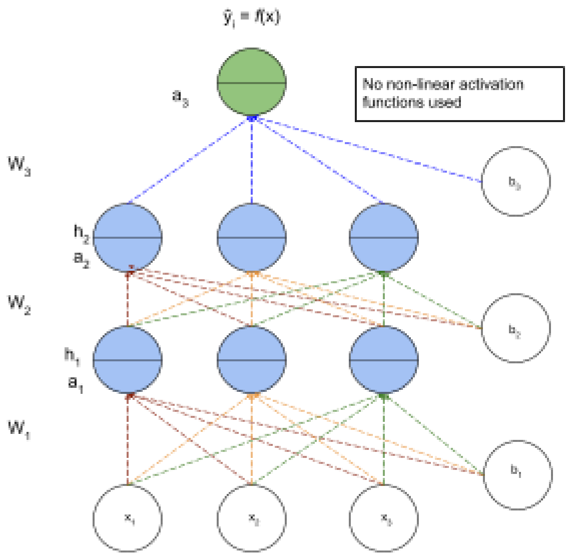
7.1.1: Setting the context

The progress of DL over the past decade

1. In this section, we will be looking at how better activation functions and better weight initialization has sped up the growth of DL over the last decade



2. Why are activation functions important?
   a. Consider a network where there are no non-linear activation functions like sigmoid etc.



   b. Here $\hat{y}_i = W_3(W_2(W_1(x_i))) = Wx_i$
   c. It can only represent linear relations between x and y
   d. Universal Approximation Theorem does not hold good.
   e. The **representation power** of a deep NN is due to its **non-linear activation functions**
3. Some popular non-linear activation functions are
   a. Logistic - can be called a sigmoid function
   b. tanh - can be called a sigmoid function
   c. ReLU
   d. Leaky ReLU

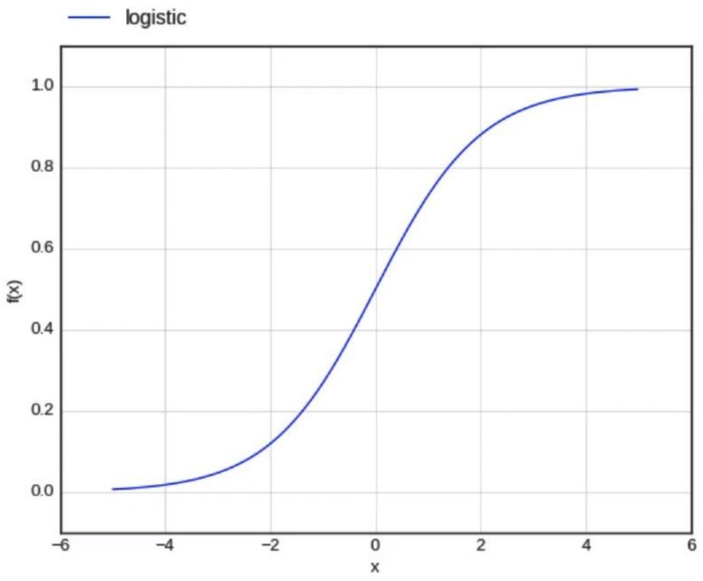
PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

7.1.2: Saturation in logistic neuron

Let's look at the logistic function

1. The following figure illustrates the logistic function



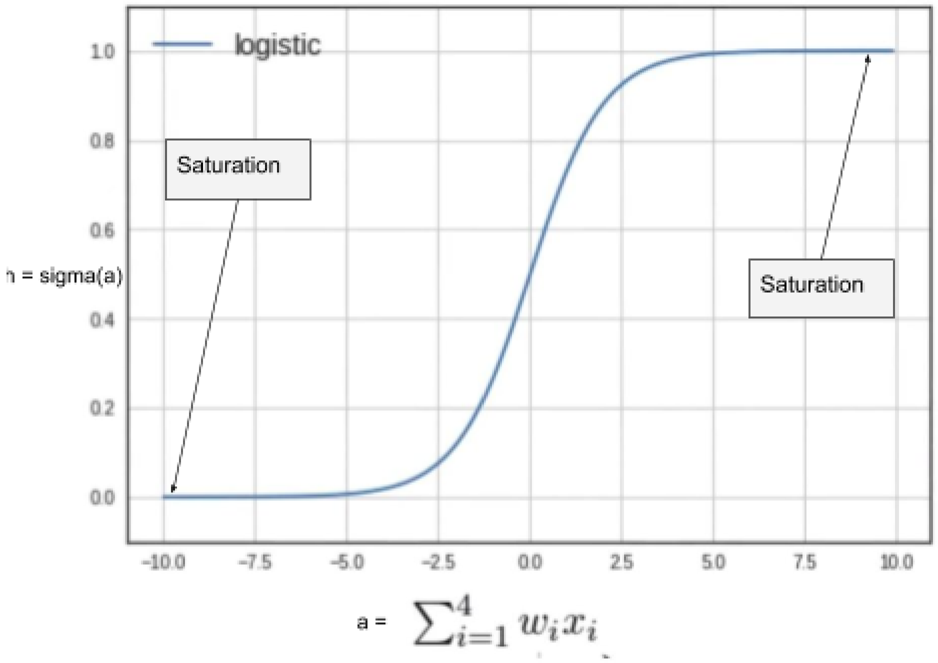
   a. $f(x) = \frac{1}{1+e^{-x}}$
   b. $f'(x) = \frac{\partial f(x)}{\partial x} = f(x) * (1 - f(x))$
2. A logistic neuron is said to be saturated when it reaches its peak values when it is given high extremes of positive or negative values
   a. When $f(x) = 0$
   b. And hence $f'(x) = 0$
   c. In the case where we are calculating the gradient w.r.t a weight associated with a saturated neuron, the saturated neuron derivative is 0, thus resulting in the entire gradient becoming 0
   d. This is because the term assocaited with the saturated neuron in the chain rule for gradient calculation becomes 0, thus making the entire gradient 0
   e. Due to this, the weights are not updated.
   f. This is called the **Vanishing Gradient Problem**, because the gradient vanishes or becomes 0 due to the presence of a saturated neuron.
3.

PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

7.1.3: Zero centered functions

1. Why do logistic neurons saturate?
   a. Consider a pre activation function: $a = \Sigma^{n}_{i=1} w_n x_n$
   b. And the activation function $h = \sigma(a)$

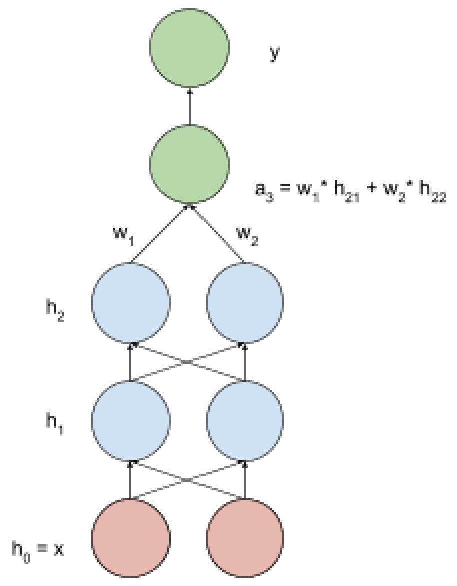


   c. In cases where the weights are initialised to very high or very low values, the weighted summation term (a) will become very large or very small (very negative).
   d. This could result in the neuron attaining saturation
   e. Remember to initialise the weights to small values
2. Another shortcoming with the logistic function is that it is not zero centered
   a. Zero centered: The function is spread out equidistant around the 0 point, i.e. it takes an equal number of positive and negative values.
   b. The logistic function ranges from 0 to 1
   c. The tanh function is a zero centered sigmoid function

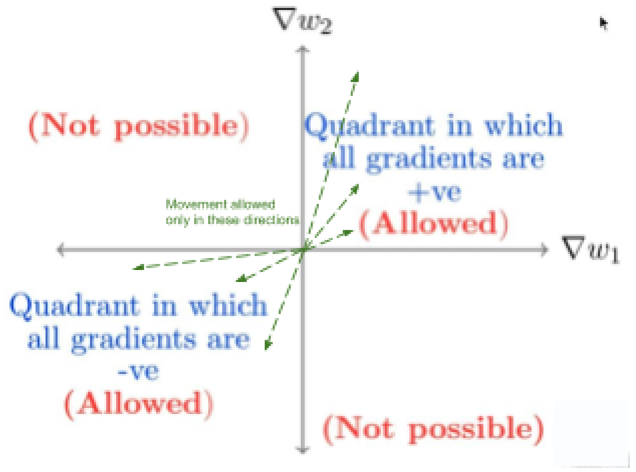
PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

3. Consider the simple neural network with logistic sigmoid neurons



   a. Consider the following gradients
   b. $\nabla w_1 = \left(\frac{\partial L(w)}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3}\right) * \frac{\partial a_3}{\partial w_1} = \left(\frac{\partial L(w)}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3}\right) * h_{21}$
   c. $\nabla w_2 = \left(\frac{\partial L(w)}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3}\right) * \frac{\partial a_3}{\partial w_2} = \left(\frac{\partial L(w)}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3}\right) * h_{22}$
   d. The bracketed terms are common
   e. Both h_{21} and h_{22} are outputs of the logistic function, so they are always positive (i.e. ranging from 0 to 1)
   f. Due to this, at all times, both ∇w_1 and ∇w_2 will always be of the same sign, either positive or negative. They cannot be different from each other since the bracketed part is common between them and the logistic function output is always positive
   g. The gradients w.r.t all the weights connected to the same neuron are either all +ve or all -ve
   h. Thus, this limits the directions in which the weights can be updated


∇w_2
(Not possible)
Quadrant in which all gradients are +ve (Allowed)
Movement allowed only in these directions
∇w_1
Quadrant in which all gradients are -ve (Allowed)
(Not possible)

4. Thus, we cannot arrive at the local minima as fast as possible by moving in all directions.
5. Also, logistic function is computationally expensive because of e^x

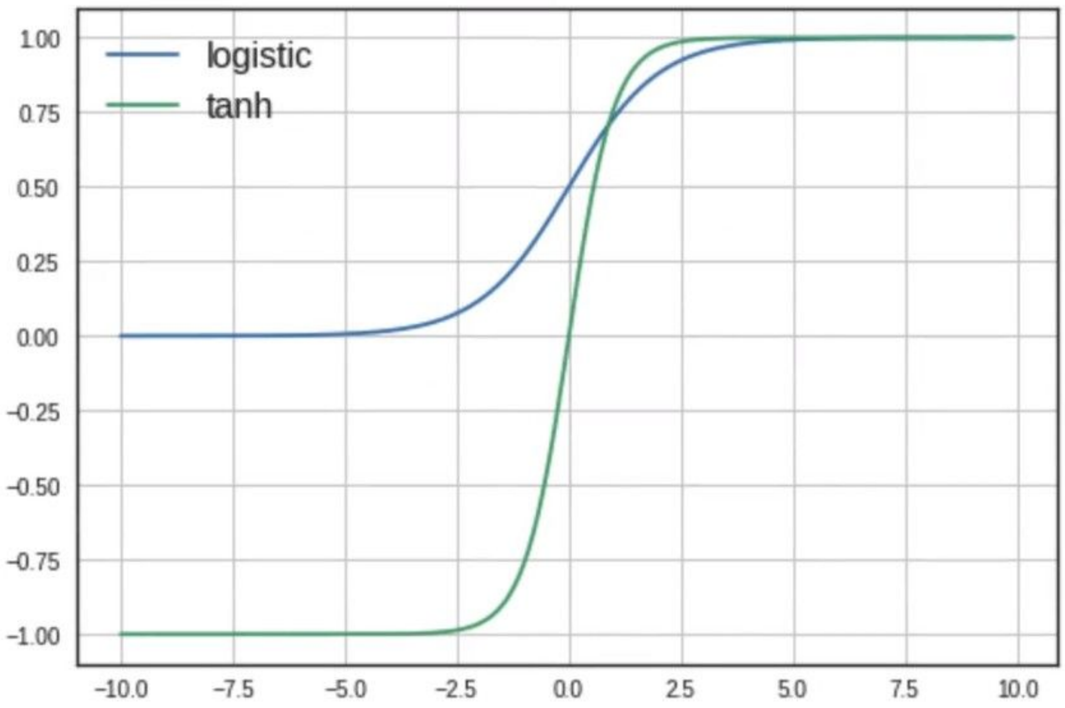
PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

7.1.4: Introducing Tanh and ReLU activation functions

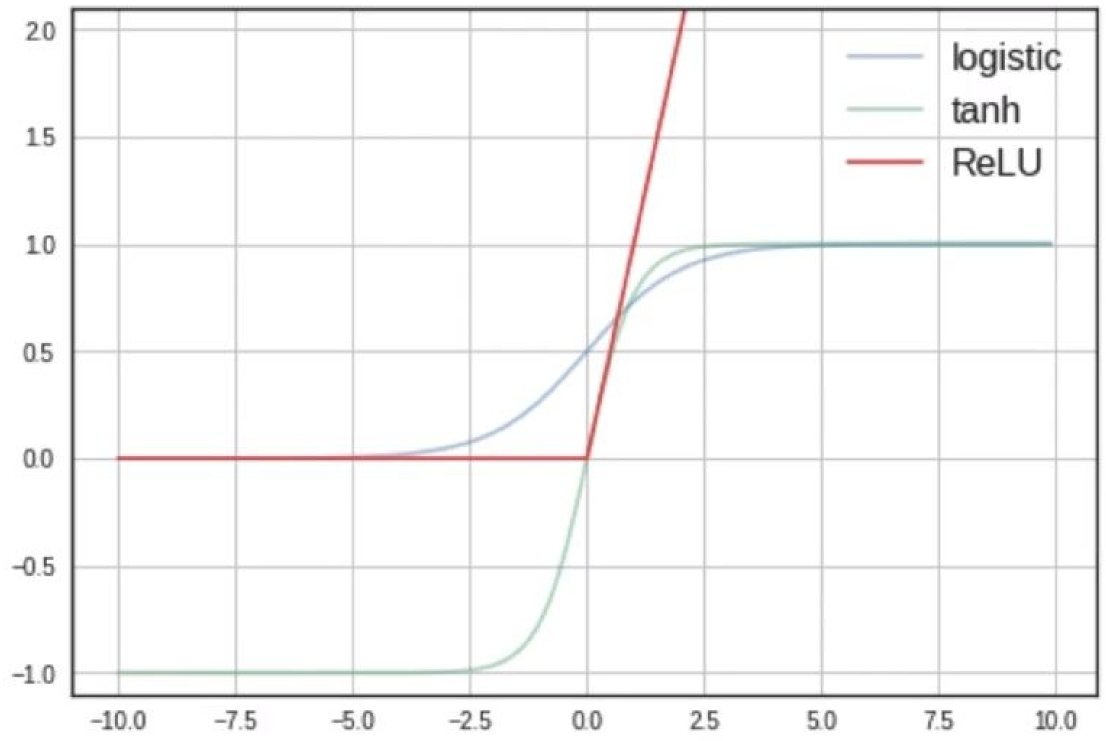
What are the other alternatives to the Logistic function?

1. **tanh**
   a. The following figure illustrates the tanh function



   b. $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
   c. $f'(x) = \frac{\partial f(x)}{\partial x} = (1 - (f(x))^2)$
   d. The tanh function ranges from -1 to +1, whereas the logistic function ranges from 0 to 1
   e. It is a zere centered function.
   f. The function saturates at f(x) = -1 or 1, thus causing the gradients to vanish.
   g. tanh is computationally expensive because of e^x
   h. However, it is still preferred over the logistic function

2. **ReLU**
   a. The following figure illustrates the ReLU function



   b. $f(x) = max(0, x)$
   c. $f'(x) = \frac{\partial f(x)}{\partial x} = 0 \; if \; x < 0 \mid 1 \; if \; x > 0$
   d. ReLU outputs the input value itself if it is positive, else it outputs zero, i.e. f(1) = 1, f(-1) = 0
   e. It does not saturate in the positive region
   f. It is not zero centered
   g. Easy to compute (no expensive e^x)

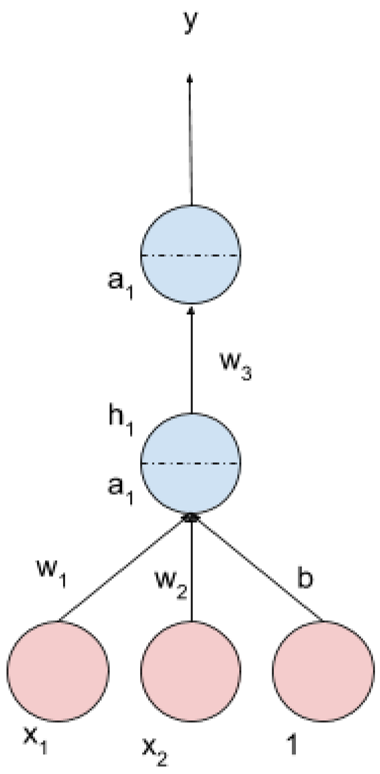
PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

7.1.5: Tanh and ReLU Activation Functions

Is there any caveat in using ReLU?

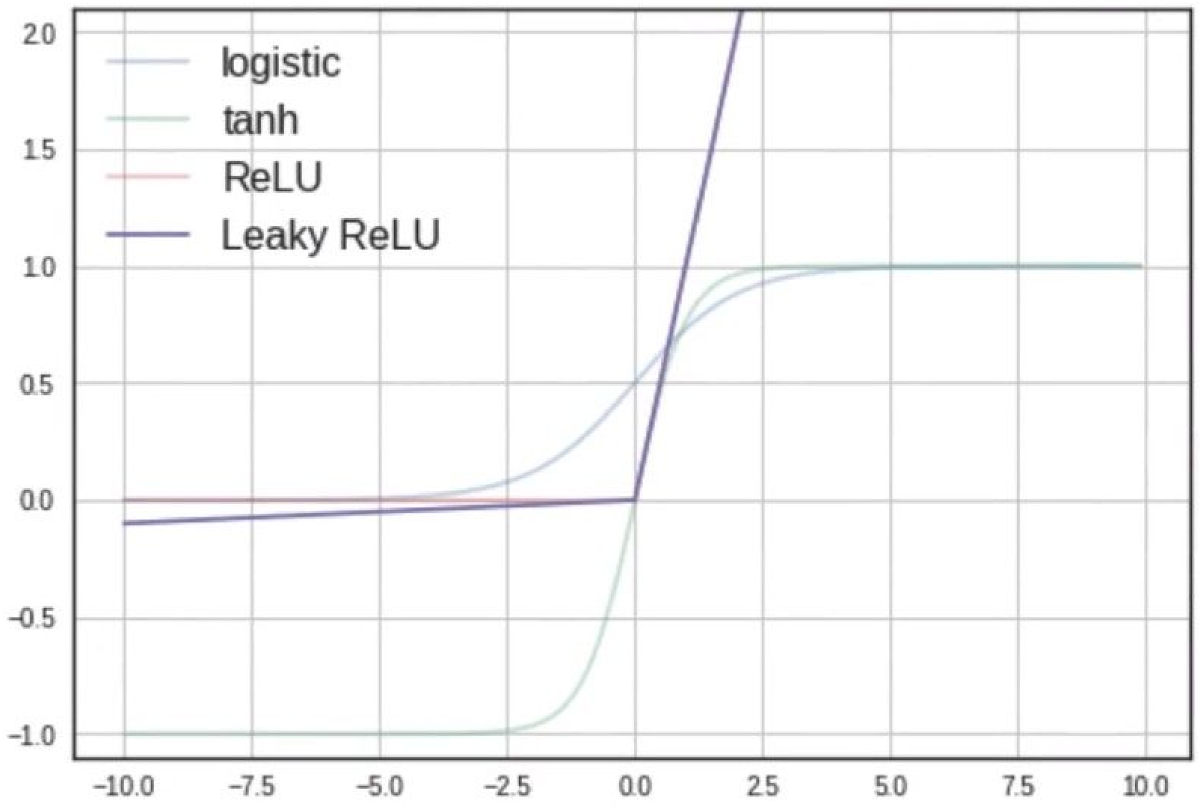
1. Consider the following deep neural network that uses the ReLU activation function



   a. $h_1 = ReLU(a_1) = max(0, a_1) = max(0, w_1x_1 + w_2x_2 + b)$
   b. What happens if b takes on a large negative value due to a large negative update ∇b at some point?
   c. $w_1x_1 + w_2x_2 + b < 0$ [if b << 0]
   d. Therefore $h_1 = 0$ [dead neuron]
   e. Which means $\frac{\partial h_1}{\partial a_1} = 0$
   f. This zero derivative is involved in the chain rule for computing the gradient w.r.t ∇w_1
   g. ∇w_1 *becomes* 0 leading to the weight not being updated, as in the case of a **saturated neuron**.
   h. This also applies to ∇w_2 and ∇b, their parameters are not updated.
   i. Here, x_1 and x_2 have been normalised, so they range between 0-1 and are therefore unable to counterbalance any large negative value b
   j. This means that once a neuron has died, it remains dead forever, as no new input would be large enough to counter the negative b value
   k. Thus, there is a very real problem of saturation of a ReLU neuron in the negative region.
   l. In practice, if there is a large number of ReLU neurons, a large fraction (up to 50%) may die during operation if the learning rate is set too high
   m. It is advised to initialise the bias to a positive value

n. Using other variants of ReLU is recommended

2. A good alternative is the **Leaky ReLU**
   a. The following figure illustrates the leaky ReLU function



   b. $f(x) = max(0.01x,\ x)$
   c. $f'(x) = \frac{\partial f(x)}{\partial x} = 0.01\ if\ x < 0\ |\ 1\ if\ x > 0$
   d. ReLU outputs the input value itself if it is positive, else it outputs a fraction of the input value, i.e. f(2) = 2, f(-2) = 0.02
   e. It does not saturate in the positive or negative region
   f. Will not die (0.01x ensures that at least a small gradient will flow through), this means that there isn't any 0 valued derivative, thereby ensuring that the gradients are all non-zero. Thus, the weights are always updated.
   g. It is easy to compute (no expensive e^x)
   h. Close to zero centered outputs

PadhAI Week 7: Activation Functions and Initialization Methods

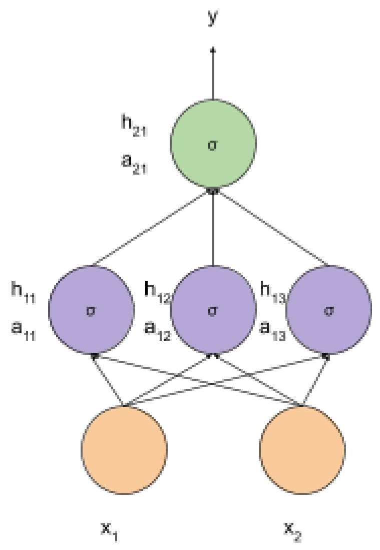
by Manick Vennimalai

7.2: Initialization Methods

7.2.1: Symmetry Breaking Problem

Why not simply initialise all weights to 0?

1. Consider the following neural network that uses the logistic activation function



   a. $a_{11} = w_{11}x_1 + w_{12}x_2$
   b. $a_{12} = w_{21}x_1 + w_{22}x_2$
   c. Let us initialise all the weights to 0
   d. $a_{11} = a_{12} = 0$
   e. $h_{11} = h_{12}$
   f. $\nabla w_{11} = \frac{\partial L(w)}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} . x_1$
   g. $\nabla w_{21} = \frac{\partial L(w)}{\partial y} \cdot \frac{\partial y}{\partial h_{12}} \cdot \frac{\partial h_{12}}{\partial a_{12}} . x_2$
   h. But $h_{11} = h_{12}$ and $a_{11} = a_{12}$
   i. Therefore $\nabla w_{11} = \nabla w_{21}$
   j. We can see that if we initialise the weights to equal values, then the equality carries through to the gradients associated with these weights, thereby keeping the weights equal throughout the training.
   k. This is known as the Symmetry Breaking Problem, where if you start with equal initialised weights, they remain equal through the training.
   l. Hence weights connected to the same neuron should never be initialised to the same value?
2. Some conclusions we can make are as follows
   a. Never initialise all weights to 0
   b. Never initialise all weights to the same value

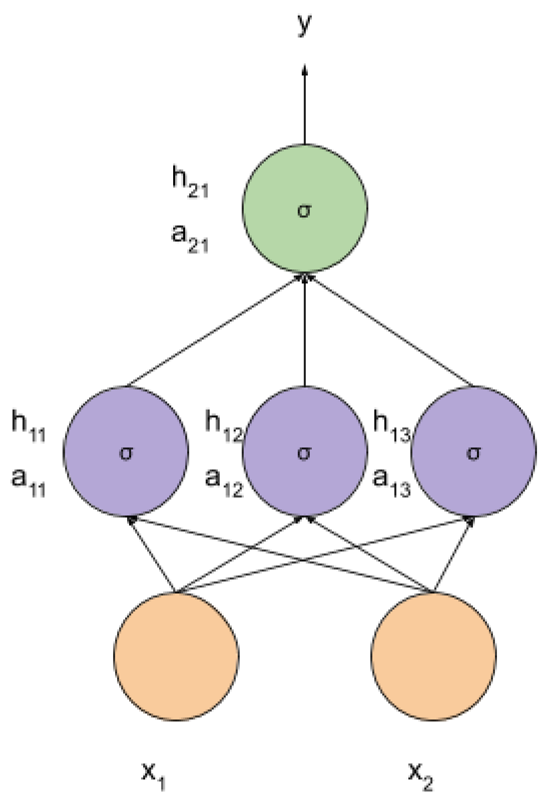
PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

7.2.2: Xavier and He Initialization

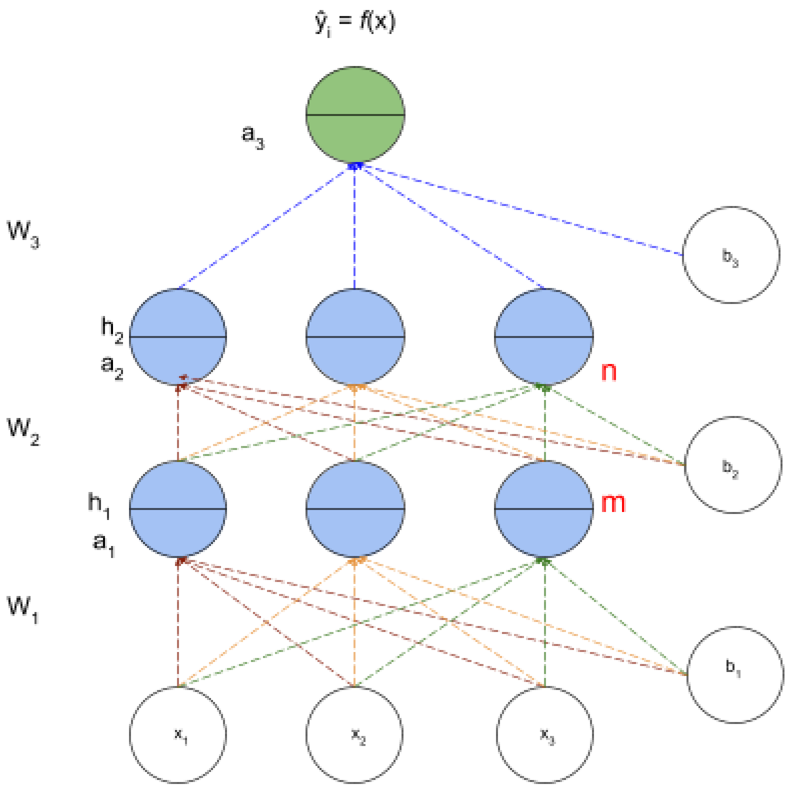
Why shouldn't you initialise all the weights to large values?

1. Consider the following neural network that uses the logistic activation function



   a. $a_{11} = w_{11}x_1 + w_{12}x_2$
   b. $a_{12} = w_{21}x_1 + w_{22}x_2$
   c. Here, input values are normalised (0-1) and the weights are initialised to large values
   d. This would result in the function attaining saturation.
   e. Thus, a few noteworthy points are:
      i. Always normalise the inputs (should lie between 0 to 1). If not, they too could contribute to saturation if their values are very large or small
      ii. Never initialise weights to large values
   f. To reinforce the points we made earlier about initialisation
      i. Never initialise all weights to 0
      ii. Never initialise all weights to the same value
      iii. Never initialise all weights to large values

2. Let's look at a sample neural network and consider some alternate initialisation methods



3. **Xavier initialisation**:
   a. Consider $a_{21} = w_{21}h_{11} + w_{22}h_{12} + ... + w_{2m}h_{1m}$
   b. Here, as the number of input neurons increase (m>>0), the value of a_{21} could potentially be very high. Thus to avoid saturation due to a extremely large value, it is recommended that the weights scale inversely with the number of input neurons, i.e. $w \propto \frac{1}{m}$
   c. The python implementation is as follows

```
W_2 = np.random.randn(num_in, num_out) / sqrt(num_in)
      # This term is an m x n matrix     divided by sqrt(m)
```

   d. Here, np.rand.randn gives a value between 0-1 from the normal distribution. This m x n matrix is then divided by the sq.root of m. This prevents a_{21} from blowing up to a very large value. Used in the case of **tanh and logistic activations**
4. **He Initialisation**:
   a. It is used for ReLU and Leaky ReLU
   b. Here is the python implementation

```
W_2 = np.random.randn(num_in, num_out) / sqrt(num_in/2)
      # This term is an m x n matrix     divided by sqrt(m/2)
```

   c. Here, we divide by $\sqrt{m/2}$ because of the rough intuition that in ReLU, around half the neurons die during training.

PadhAI Week 7: Activation Functions and Initialization Methods

by Manick Vennimalai

7.2.3: Summary and what next

What have we learned in this chapter?

1. **Activation Functions**
   a. Logistic (❌ Not used much in practice)
   b. tanh (RNNs)
   c. ReLU (CNNs)
   d. Leaky ReLU (CNNs)
2. **Initialisation Methods**
   a. Zero, Equal, Large (❌ Not used)
   b. Xavier initialisation (tanh, logistic)
   c. He initialisation (ReLU)
3. Activation functions and initialisation methods go hand-in-hand, and together, they try to avoid the problem of vanishing gradients and saturation