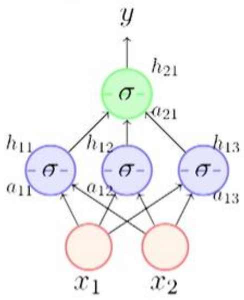
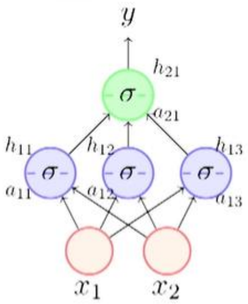


$$a_{11} = w_{11}x_1 + w_{12}x_2$$
$$a_{12} = w_{21}x_1 + w_{22}x_2$$
$$\therefore a_{11} = a_{12} = 0$$
$$\therefore h_{11} = h_{12}$$

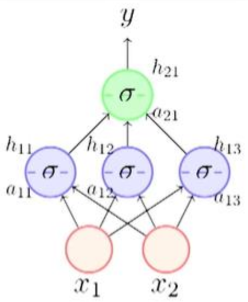
- What happens if we initialize all weights to 0?
- All neurons in layer 1 will get the same activation



$$a_{11} = w_{11}x_1 + w_{12}x_2$$
$$a_{12} = w_{21}x_1 + w_{22}x_2$$
$$\therefore a_{11} = a_{12} = 0$$
$$\therefore h_{11} = h_{12}$$

- Now what will happen during back propagation?

$$\nabla w_{11} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} . x_1$$
$$\nabla w_{21} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{12}} \cdot \frac{\partial h_{12}}{\partial a_{12}} . x_1$$
$$but \quad h_{11} = h_{12}$$
$$and \quad a_{12} = a_{12}$$
$$\therefore \nabla w_{11} = \nabla w_{21}$$



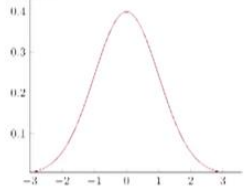
$$a_{11} = w_{11}x_1 + w_{12}x_2$$
$$a_{12} = w_{21}x_1 + w_{22}x_2$$
$$\therefore a_{11} = a_{12} = 0$$
$$\therefore h_{11} = h_{12}$$

- Hence both the weights will get the same update and remain equal
- Infact this symmetry will never break during training
- The same is true for w_{12} and w_{22}
- And for all weights in layer 2 (infact, work out the math and convince yourself that all the weights in this layer will remain equal)
- This is known as the **symmetry breaking problem**
- This will happen if all the weights in a network are initialized to the **same value**

```python
D = np.random.randn(1000,500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1]
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

We will now consider a feedforward network with:

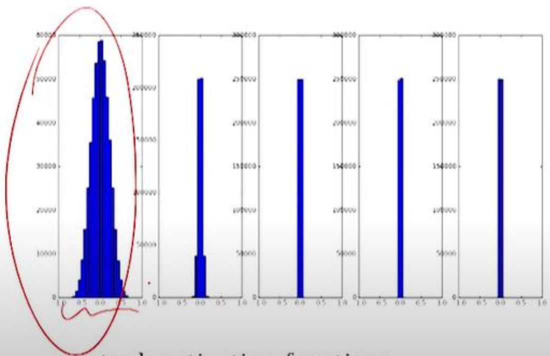
- input: 1000 points, each $\in R^{500}$
- input data is drawn from unit Gaussian



- the network has 5 layers
- each layer has 500 neurons
- we will run forward p this network with differ tializations

```python
W = np.random.randn(fan_in, fan_out) * 0.01
```

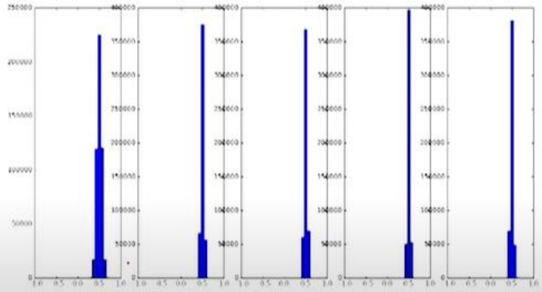
- Let's try to initialize the weights to small random numbers
- We will see what happens to the activation across different layers



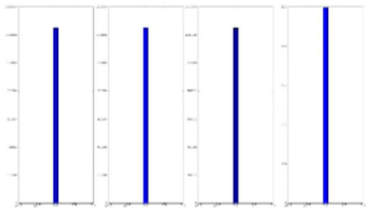
tanh activation functions

```
W = np.random.randn(fan_in, fan_out) * 0.01
```

- Let's try to initialize the weights to small random numbers
- We will see what happens to the activation across different layers



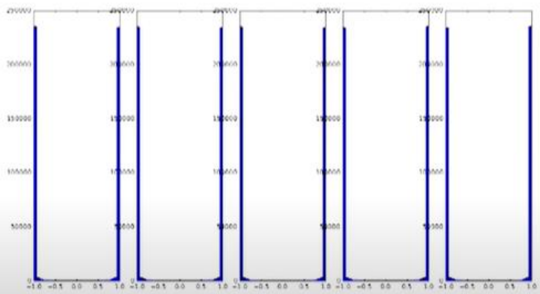
sigmoid activation functions



- What will happen during back propagation?
- Recall that ∇w_1 is proportional to the activation passing through it
- If all the activations in a layer are very close to 0, what will happen to the gradient of the weights connecting this layer to the next layer?

```
W = np.random.randn(fan_in, fan_out)
```

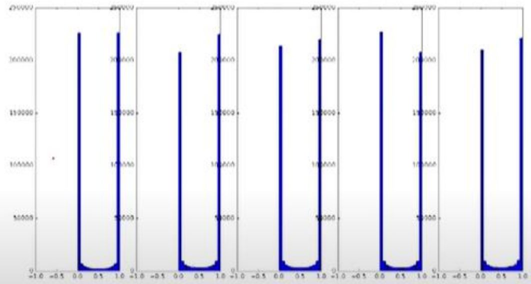
- Let us try to initialize the weights to large random numbers



tanh activation with large weights

```python
W = np.random.randn(fan_in, fan_out)
```

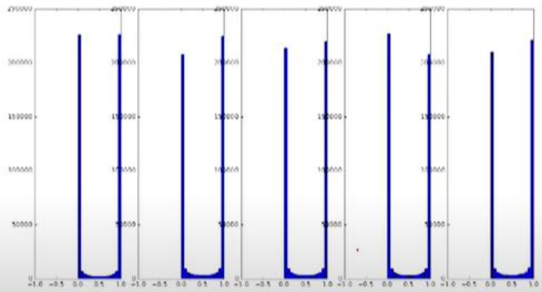
- Let us try to initialize the weights to large random numbers



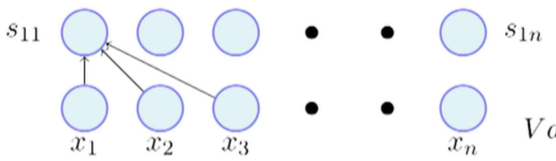
sigmoid activations with large weights

```python
W = np.random.randn(fan_in, fan_out)
```

- Let us try to initialize the weights to large random numbers
- Most activations have saturated
- What happens to the gradients at saturation?
- They will all be close to 0 (vanishing gradient problem)



sigmoid activations with large weights



- Let us try to arrive at a more principled way of initializing weights

$$s_{11} = \sum_{i=1}^{n} w_{1i} x_i$$

$$Var(s_{11}) = Var(\sum_{i=1}^{n} w_{1i} x_i) = \sum_{i=1}^{n} Var(w_{1i} x_i)$$

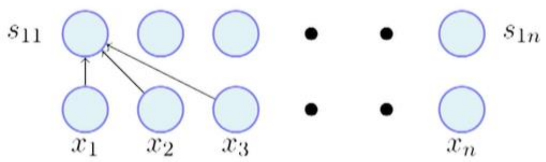
$$= \sum_{i=1}^{n} [(E[w_{1i}])^2 Var(x_i)$$

$$+ (E[x_i])^2 Var(w_{1i}) + Var(x_i)Var(w_{1i})]$$

$$= \sum_{i=1}^{n} Var(x_i)Var(w_{1i})$$

$$= (nVar(w))(Var(x))$$

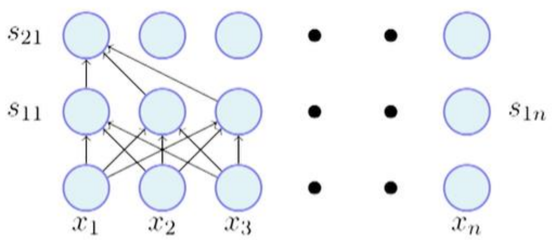
- [Assuming 0 Mean inputs and weights]
- [Assuming $Var(x_i) = Var(x) \forall i$]
- [Assuming $Var(w_{1i}) = Var(w) \forall i$]



- In general,

$$Var(S_{1i}) = (nVar(w))(Var(x))$$

- What would happen if $nVar(w) \gg 1$?
- The variance of S_{1i} will be large
- What would happen if $nVar(w) \to 0$?
- The variance of S_{1i} will be small



$$\boxed{Var(S_{i1}) = nVar(w_1)Var(x)}$$

- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

$$Var(s_{21}) = \sum_{i=1}^{n} Var(s_{1i})Var(w_{2i})$$

$$= nVar(s_{1i})Var(w_2)$$

$$Var(s_{21}) \propto [nVar(w_2)][nVar(w_1)]Var(x)$$

$$\propto [nVar(w)]^2 Var(x)$$

Assuming weights across all layers have the same variance

$$\boxed{Var(az) = a^2(Var(z))}$$

- In general,

$$Var(s_{ki}) = [nVar(w)]^k Var(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

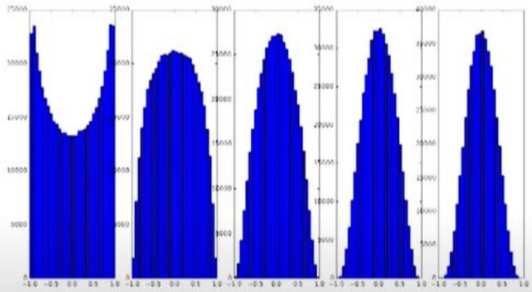
$$nVar(w) = 1$$

- If we draw the weights from a unit Gaussian and scale them by $\frac{1}{\sqrt{n}}$ then, we have :

$$nVar(w) = nVar\left(\frac{z}{\sqrt{n}}\right)$$
$$= n * \frac{1}{n} Var(z) = 1$$

```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in)
```

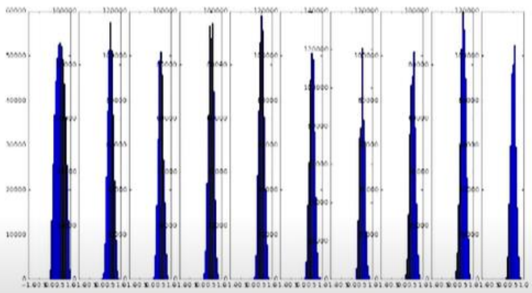
- Let's see what happens if we use this initialization



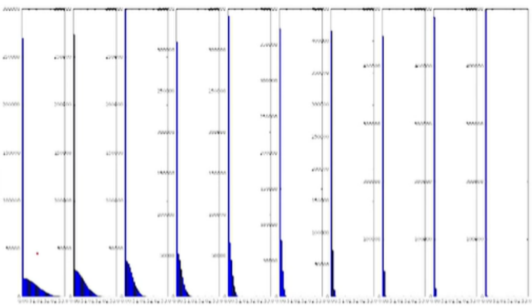
tanh activation

```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in)
```

- Let's see what happens if we use this initialization

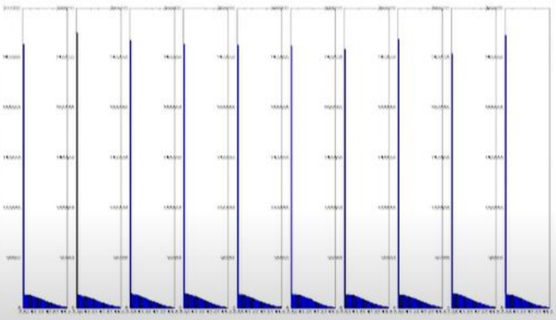


sigmoid activations



- However this does not work for ReLU neurons
- Why ?
- Intuition: *He et.al.* argue that a factor of 2 is needed when dealing with ReLU Neurons

```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in/2)
```



- Indeed when we account for this factor of 2 we see better performance