Logistic regression

$w \in \mathbb{R}^{n_x}, \quad b \in \mathbb{R}$

$$\min_{w,b} J(w, b)$$

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2 \qquad \cancel{+ \frac{\lambda}{2m} b^2}$$

omit

L_2 regularization $\quad \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L_1 regularization $\quad \frac{\lambda}{2m} \sum_{i=1}^{n_x} |w| = \frac{\lambda}{2m} \|w\|_1$

w will be sparse

Andrew Ng

Neural network

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \underbrace{\frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|_F^2}$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2 \qquad w: (n^{[l]}, n^{[l-1]}).$$

"Frobenius norm" $\qquad \|\cdot\|_2^2 \qquad \|\cdot\|_F^2$

$$dw^{[l]} = (\text{from backprop}) \qquad \frac{\partial J}{\partial w^{[l]}}$$

$$w^{[l]} := w^{[l]} - \alpha \, dw^{[l]}$$

Andrew Ng

Neural network

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})} + \underbrace{\frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2}$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2 \qquad w: (n^{[l]}, n^{[l-1]}).$$

$\underline{\text{"Frobenius norm"}}$ $\qquad \|\cdot\|_2^2 \qquad \|\cdot\|_F^2$

$$dw^{[l]} = \boxed{(\text{from backprop}) + \frac{\lambda}{m} w^{[l]}} \qquad \underline{\frac{\partial J}{\partial w^{[l]}}} = dw^{[l]}$$

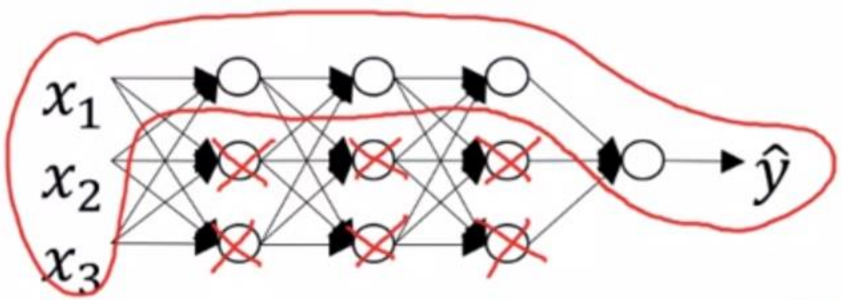
$$\rightarrow w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

"Weight decay"

$$w^{[l]} := w^{[l]} - \alpha\left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]}\right]$$
$$= \underbrace{w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]}} - \alpha(\text{from backprop})$$

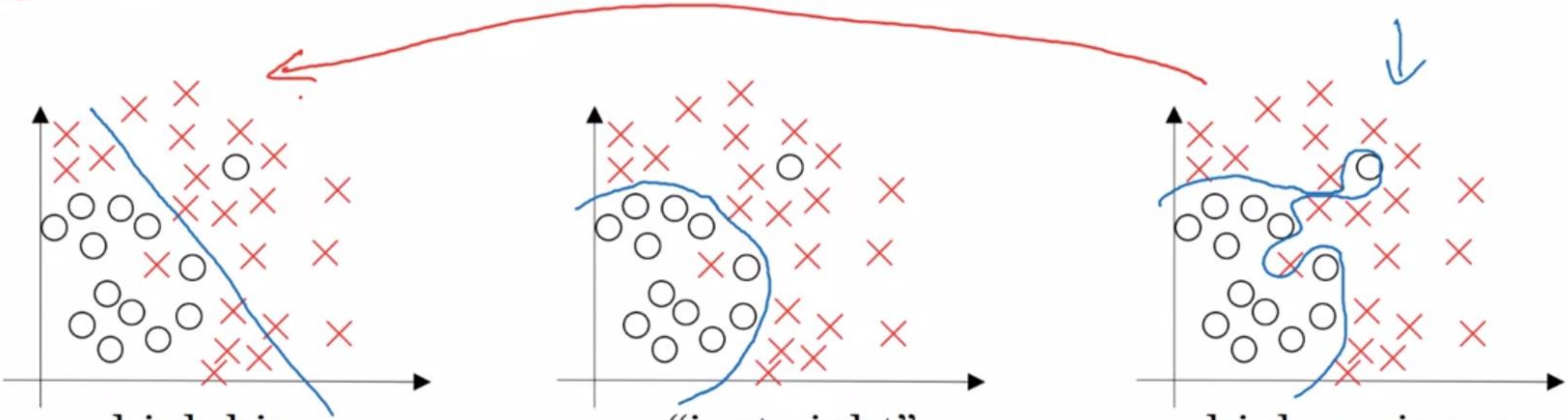
Andrew Ng

How does regularization prevent overfitting?



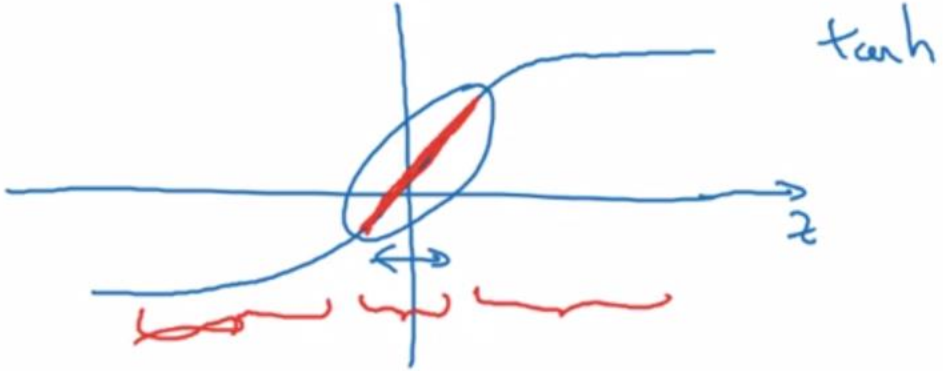
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \underline{\|w^{[l]}\|_F^2}$$

$$w^{[l]} \approx 0$$



high bias "just right" high variance

How does regularization prevent overfitting?



$g(z) = \tanh(z)$

$\lambda \uparrow \quad W^{[l]} \downarrow \quad \underbrace{z^{[l]}} = \underline{W}^{[l]} a^{[l-1]} + b^{[l]}$

Every layer $\approx$ linear.

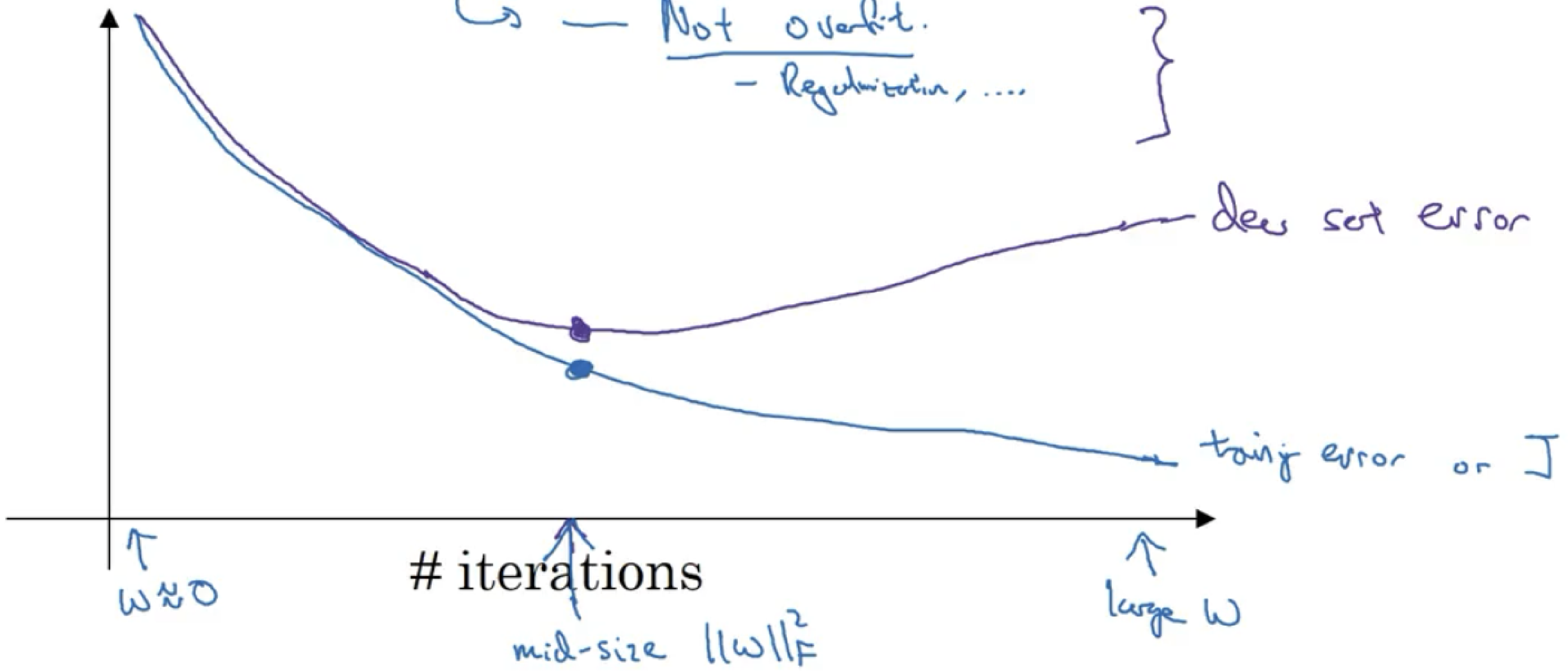
Andrew Ng

Early stopping

Orthogonalization.

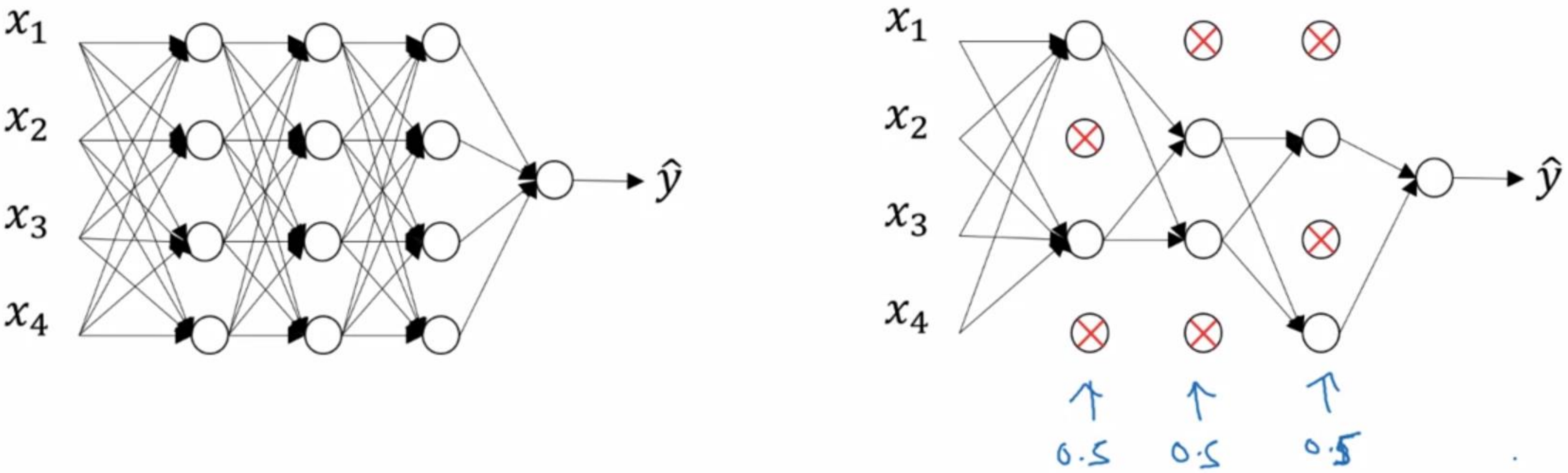
- Optimize cost function J
  - Gradient, ...
- Not overfit.
  - Regularization, ...

$J(w,b)$



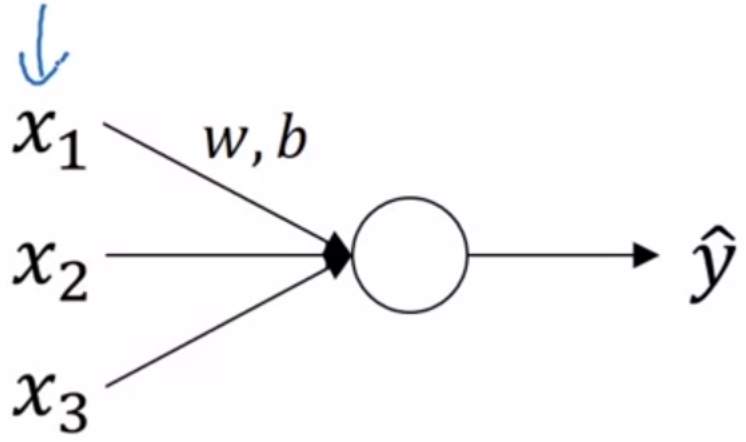
Andrew Ng

Dropout regularization



Andrew Ng

Normalizing inputs to speed up learning



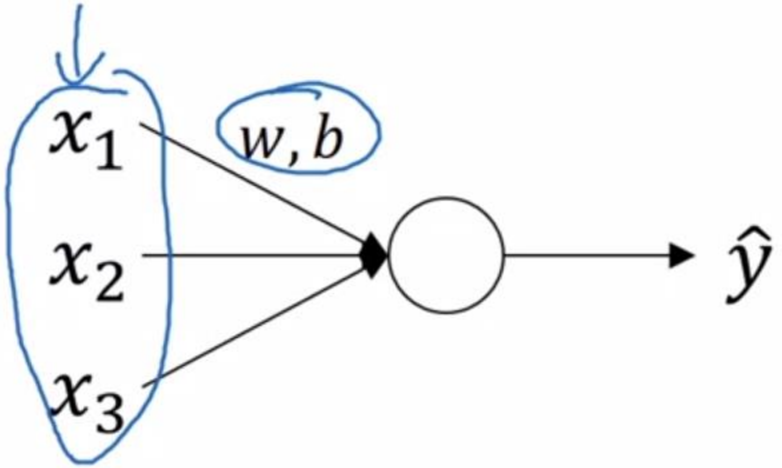
$$\mu = \frac{1}{m} \sum_i x^{(i)}$$

$$X = X - \mu$$ ← element-wise

$$\sigma^2 = \frac{1}{m} \sum_i x^{(i)2}$$

$$X = X / \sigma^2$$

Normalizing inputs to speed up learning

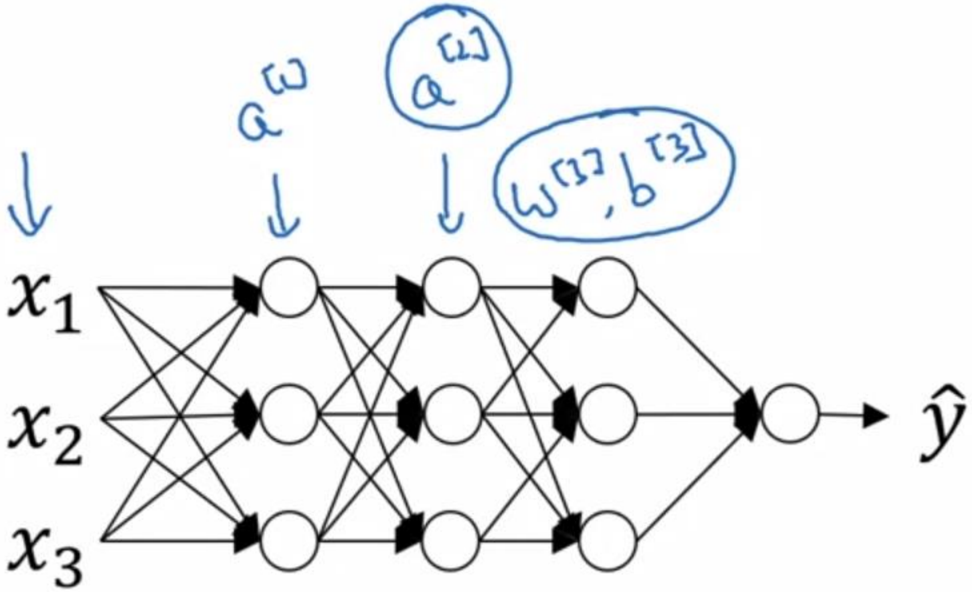


$$\mu = \frac{1}{m} \sum_i x^{(i)}$$

$$X = X - \mu \quad \leftarrow \text{element-wise}$$

$$\sigma^2 = \frac{1}{m} \sum_i x^{(i)2}$$

$$X = X / \sigma^2$$



Can we normalize $\underline{a^{[2]}}$ so as to train $w^{[3]}, b^{[3]}$ faster

Normalize $\underline{z^{[2]}}$

Implementing Batch Norm

Given some intermediate values in NN $\underbrace{z^{(1)}, \ldots, z^{(m)}}$ $z^{[l](i)}$

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z_{norm}^{(i)} = \frac{z^{(i)} - \boxed{\mu}}{\boxed{\sqrt{\sigma^2 + \varepsilon}}}$$

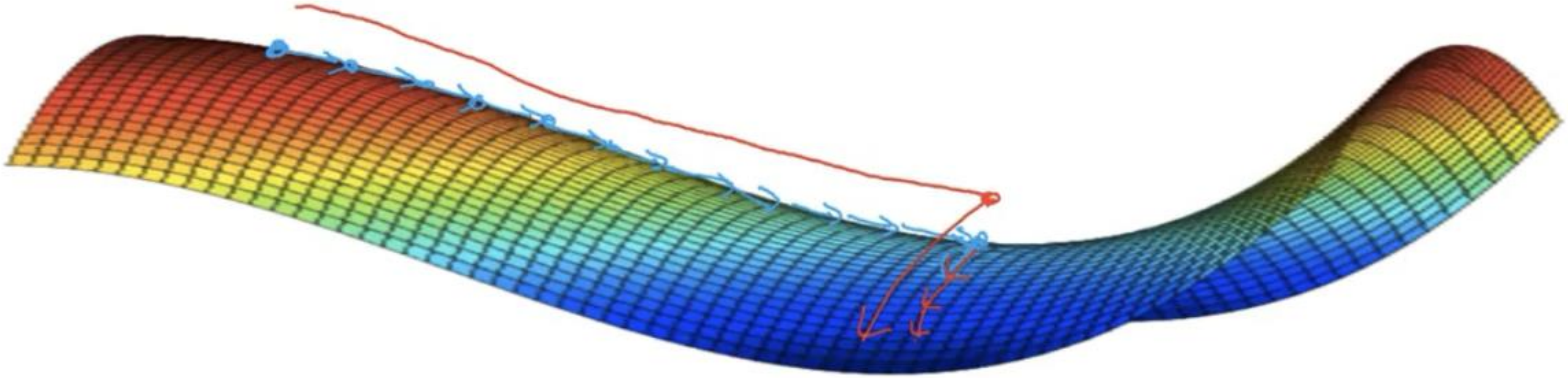
$$\tilde{z}^{(i)} = \underline{\gamma} z_{norm}^{(i)} + \underline{\beta}$$

learnable parameters of model.

$$\gamma = \boxed{\sqrt{\sigma^2 + \varepsilon}}$$

$$\beta = \boxed{\mu}$$

Problem of plateaus



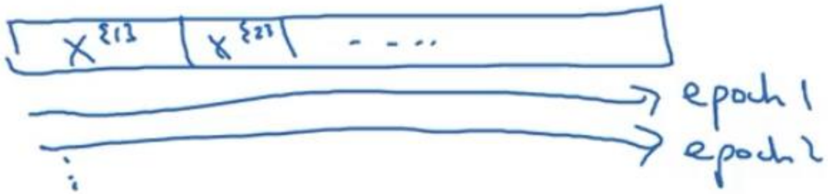
- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow

Andrew Ng

```html
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
    async function run(){
        const MODEL_URL = 'http://127.0.0.1:8887/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const input = tf.tensor2d([10.0], [1, 1]);
        const result = model.predict(input);
        alert(result);
    }
    run();
</script>
<body>
</body>
</html>
```

Learning rate decay

1 epoch = 1 pass through data.

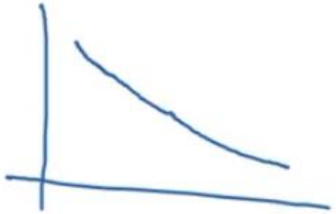


$$\alpha = \frac{1}{1 + \underline{\text{decay-rate}} * \text{epoch-num}} \underline{\alpha_0}$$

$\alpha_0 = 0.2$

decay.rate = 1

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮



Andrew Ng

Learning rate decay

Slowly reduce α