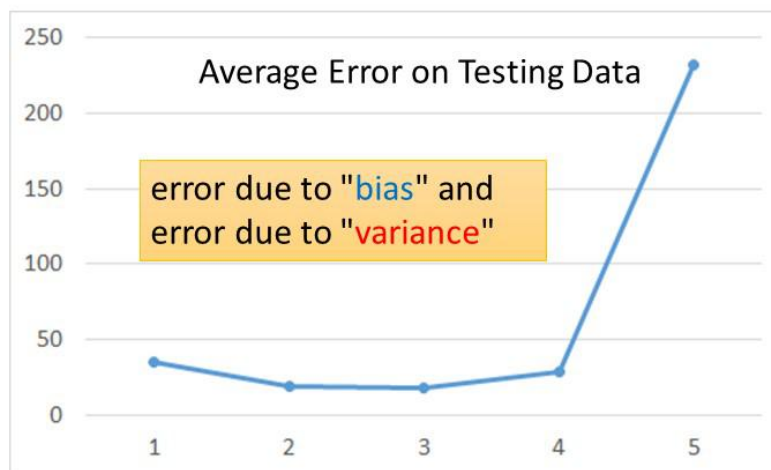


每看完一个视频，要用自己的理解把这个视频的内容陈述出来（下面的内容多数是直接翻译或者把李宏毅老师的原话写入本文档，如果是自己总结和理解的内容会进行说明）。

## 2\_Error.mp4

1\* 误差来自于 bias 和 variance

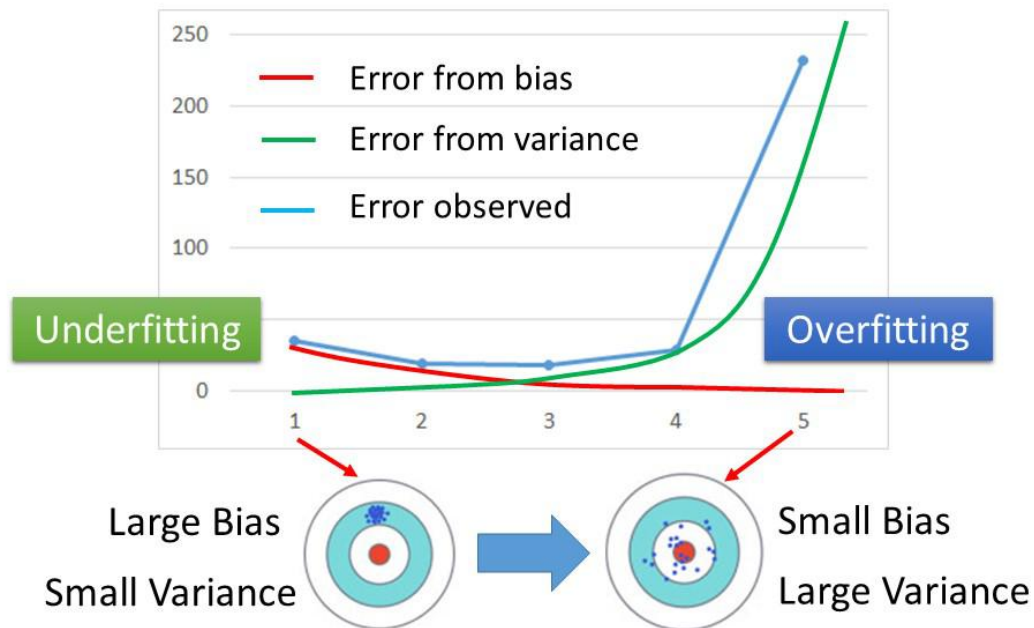
### Review



A more complex model does not always lead to better performance on testing data.

2\* 简单的模型通常会大 bias，小 variance；复杂的模型通常会大 variance，小 bias

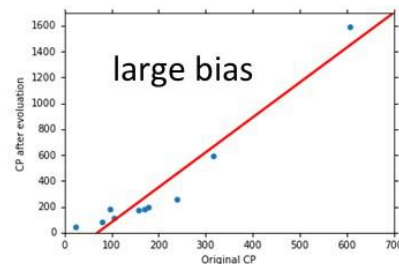
# Bias v.s. Variance



若模型连训练数据都没有拟合好，说明有大的 bias，属于明显的 underfitting，此时应该重新设计模型：增加更多的特征；或使用更加复杂的模型。

## What to do with large bias?

- Diagnosis:
  - If your model cannot even fit the training examples, then you have large bias **Underfitting**
  - If you can fit the training data, but large error on testing data, then you probably have large variance **Overfitting**
- For bias, redesign your model:
  - Add more features as input
  - A more complex model

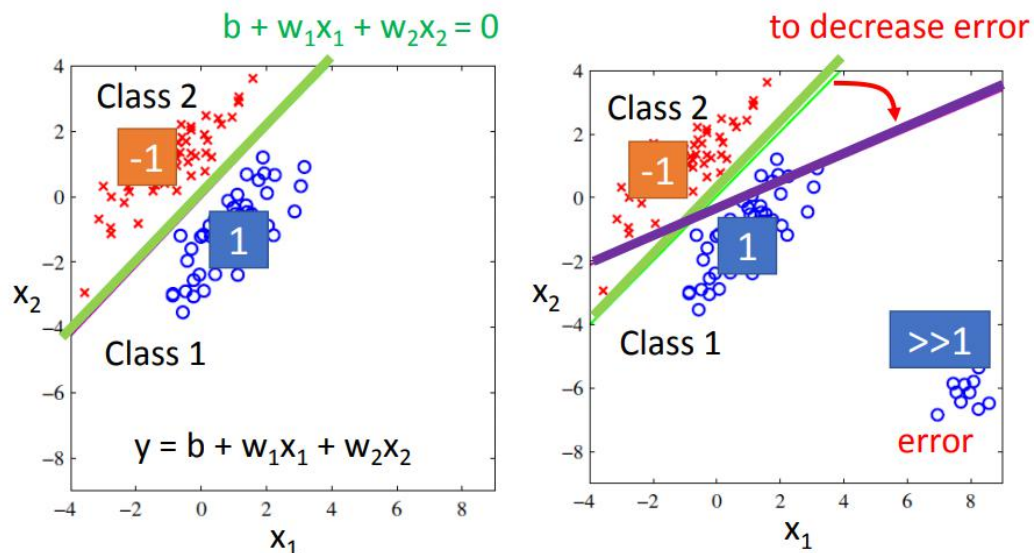


## 4\_Classification.mp4

1\* 使用 Regression 来做 Classification 是不合适的（不总是合适）

如果  $X$  的分布是下图中左边的情况，那么使用 Regression 来做 Classification 是没问题的；但如果  $X$  的分布是下图中右边的情况，就会使得学出来的回归直线向右下方偏，导致 Classification 的效果很差。

## 5\_LR.mp4



Penalize the examples that are "too correct" ... (Bishop, P186)

- Multiple class: Class 1 means the target is 1; Class 2 means the target is 2; Class 3 means the target is 3 ..... problematic

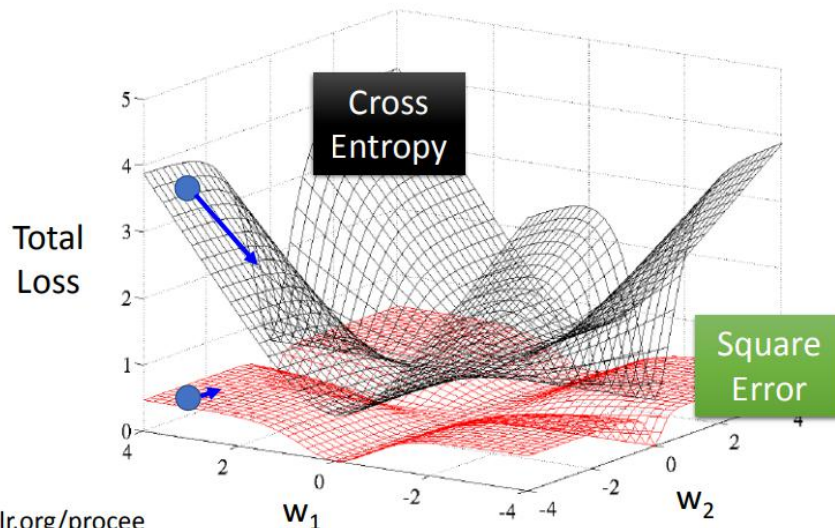
1\* LR 的梯度更新公式与线性回归的梯度更新公式完全一致

<u><b>Logistic Regression</b></u>		<u><b>Linear Regression</b></u>	
Step 1:	$f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$ Output: between 0 and 1	$f_{w,b}(x) = \sum_i w_i x_i + b$ Output: any value	
Training data: $(x^n, \hat{y}^n)$		Training data: $(x^n, \hat{y}^n)$	
Step 2:	$\hat{y}^n$ : 1 for class 1, 0 for class 2 $L(f) = \sum_n l(f(x^n), \hat{y}^n)$	$\hat{y}^n$ : a real number $L(f) = \frac{1}{2} \sum_n (f(x^n) - \hat{y}^n)^2$	
Step 3:	Logistic regression: Linear regression:	$w_i \leftarrow w_i - \eta \sum_n -(\hat{y}^n - f_{w,b}(x^n)) x_i^n$ $w_i \leftarrow w_i - \eta \sum_n -(\hat{y}^n - f_{w,b}(x^n)) x_i^n$	

2\* 为什么不用 Square Error，使用 Square Error 在当前值与目标值较远时，计算出的梯度值还是很小时，导致训练很慢（甚至无法训练完成）

而使用 Cross Entropy（即对数损失函数）在当前值与目标值较远时，计算出的梯度值较大，能够使得模型能够快速收敛，提升训练速度

## Cross Entropy v.s. Square Error



<http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf>

3\* 对于同样的训练数据,当分别使用 Discriminative 方法和 Generative 方法训练模型时,得到的模型通常是不同的。

## Discriminative v.s. Generative

$$P(C_1|x) = \sigma(w \cdot x + b)$$

directly find  $w$  and  $b$

Find  $\mu^1, \mu^2, \Sigma^{-1}$

$$w^T = (\mu^1 - \mu^2)^T \Sigma^{-1}$$

$$b = -\frac{1}{2}(\mu^1)^T (\Sigma^1)^{-1} \mu^1 + \frac{1}{2}(\mu^2)^T (\Sigma^2)^{-1} \mu^2 + \ln \frac{N_1}{N_2}$$

Will we obtain the same set of  $w$  and  $b$ ?

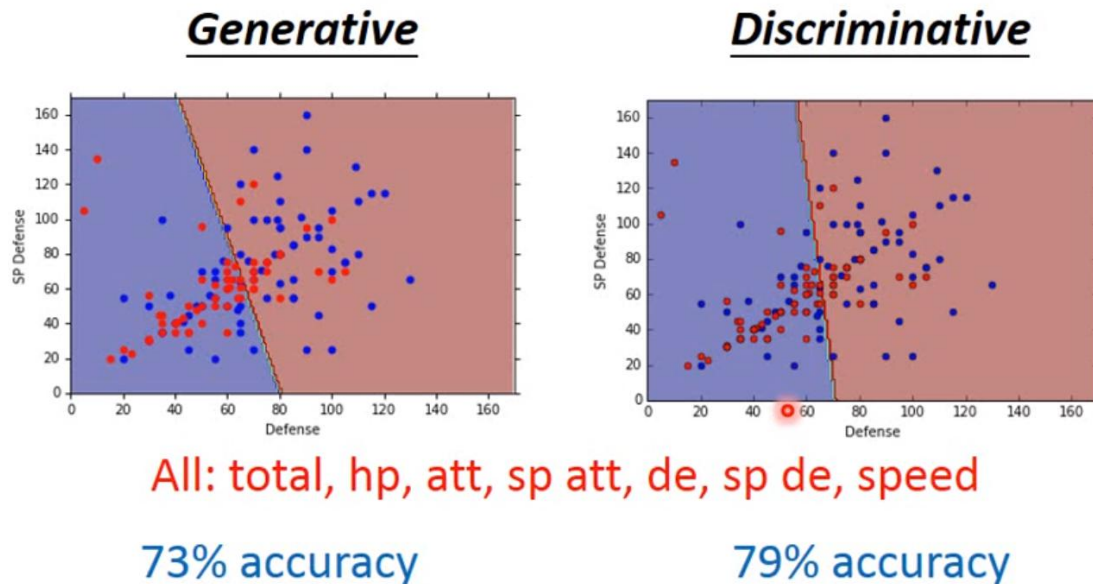
The same model (function set), but different function is selected by the same training data.

Created with EverCam.

4\* 通常认为 Discriminative 模型比 Generative 模型表现得更好



# Generative v.s. Discriminative



5\* Generative 模型的好处

## Generative v.s. Discriminative

- Usually people believe discriminative model is better
- Benefit of generative model
  - With the assumption of probability distribution
    - less training data is needed
    - more robust to the noise
  - Priors and class-dependent probabilities can be estimated from different sources.

6\* softmax 函数:

当只有两个 Class 时 (如 C1, C2), softmax 就 reduce 到了 sigmoid

## Multi-class Classification (3 classes as example)

$$C_1: w^1, b_1 \quad z_1 = w^1 \cdot x + b_1$$

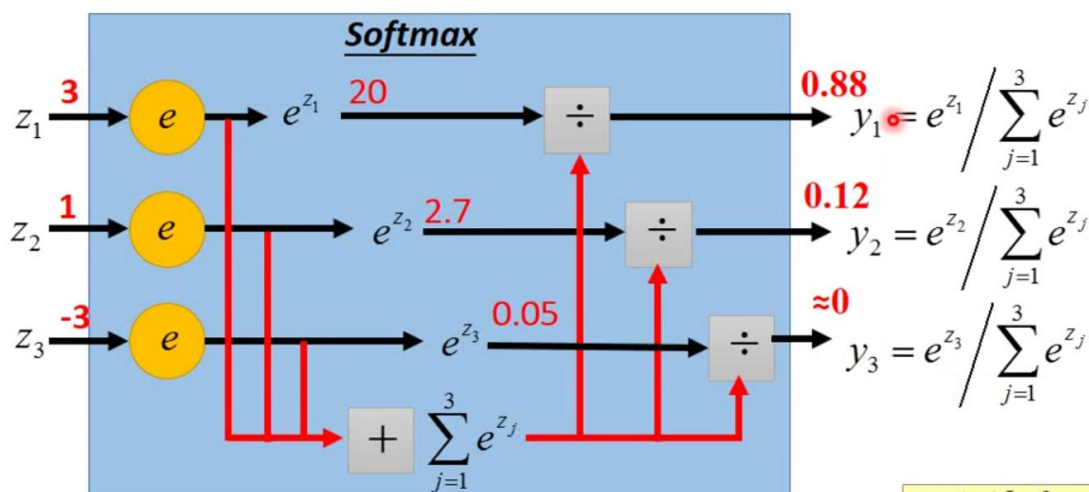
$$C_2: w^2, b_2 \quad z_2 = w^2 \cdot x + b_2$$

$$C_3: w^3, b_3 \quad z_3 = w^3 \cdot x + b_3$$

**Probability:**

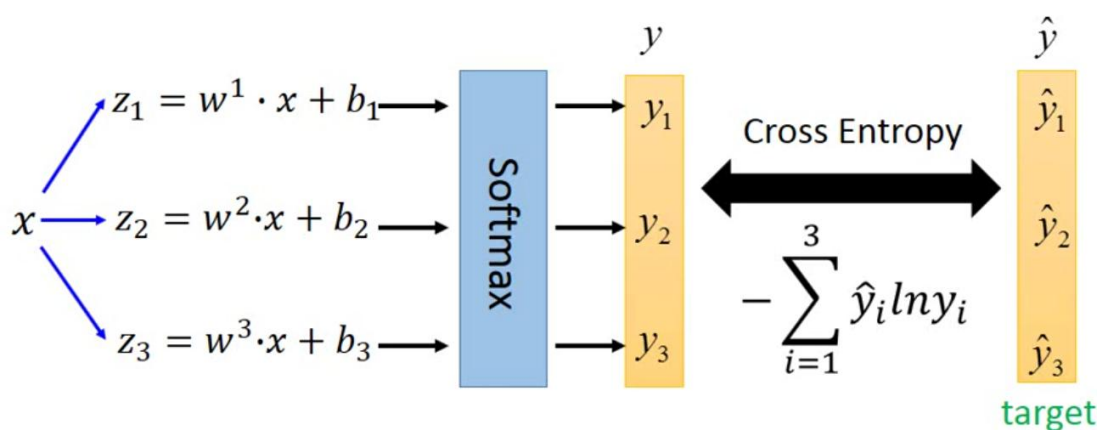
$$\blacksquare 1 > y_i > 0$$

$$\blacksquare \sum_i y_i = 1$$



7\* Cross Entropy: 交叉熵公式如图中所示

## Multi-class Classification (3 classes as example)



If  $x \in \text{class 1}$

$$\hat{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$-\ln y_1$$

If  $x \in \text{class 2}$

$$\hat{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$-\ln y_2$$

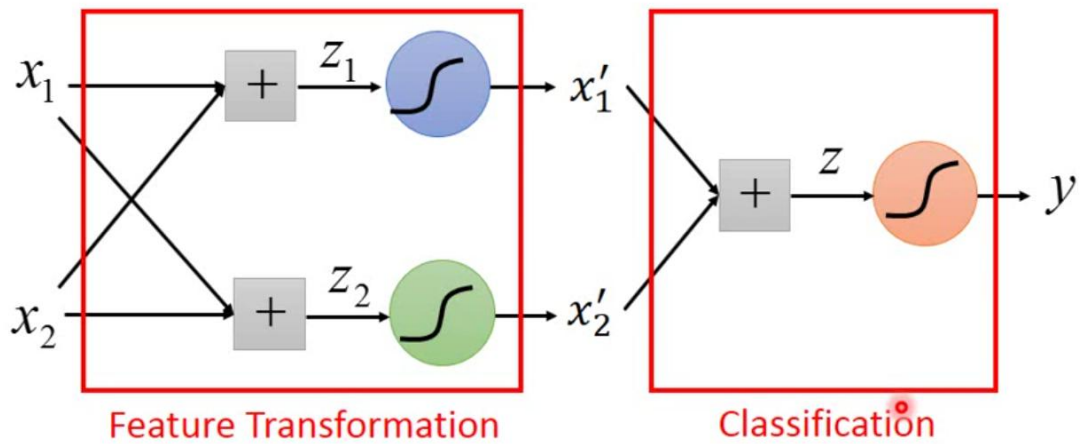
If  $x \in \text{class 3}$

$$\hat{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$-\ln y_3$$

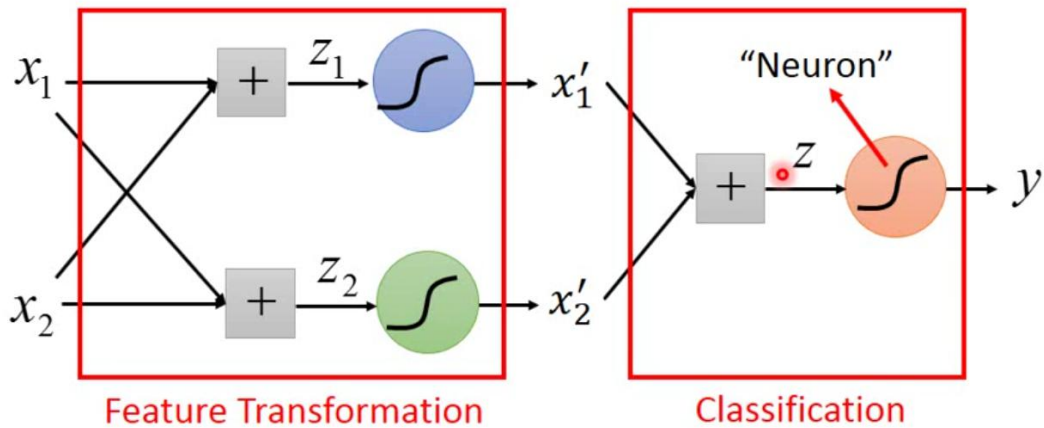
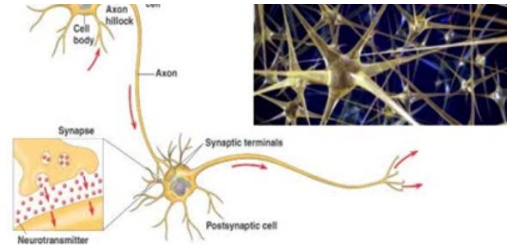
8\* 从 LR 自然过渡到 NN，cascading LR 就可以理解为是 NN

- Cascading logistic regression models



## Deep Learning!

All the parameters of the logistic regressions are jointly learned.



## Neural Network

Created with EverCam

7\_BP.mp4

1\* 链式法则



# Chain Rule

**Case 1**  $y = g(x)$   $z = h(y)$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z \quad \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

**Case 2**

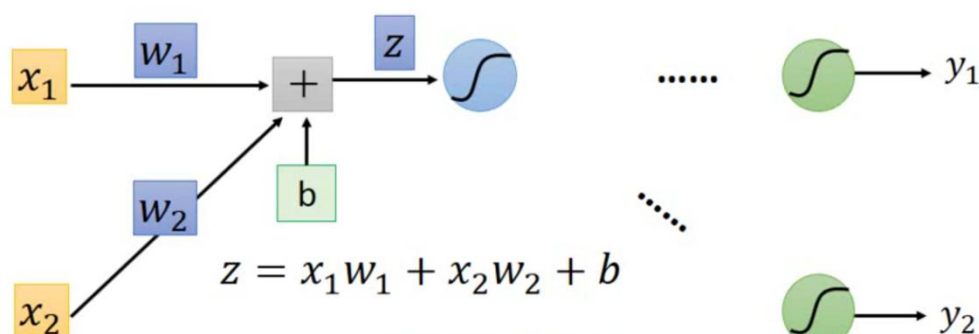
$x = g(s)$   $y = h(s)$   $z = k(x, y)$

$$\begin{array}{c} \Delta s \swarrow \searrow \\ \Delta x \quad \Delta y \\ \swarrow \searrow \\ \Delta z \end{array} \quad \frac{dz}{ds} = \frac{\partial z}{\partial x} \frac{dx}{ds} + \frac{\partial z}{\partial y} \frac{dy}{ds}$$

Created with EverCam.  
<http://www.camdemy.com>

2\* 计算  $\partial l / \partial z$  的过程实际上是一个反向传播的过程

## Backpropagation



**Forward pass:**

Compute  $\partial z / \partial w$  for all parameters

**Backward pass:**

Compute  $\partial l / \partial z$  for all activation function inputs  $z$

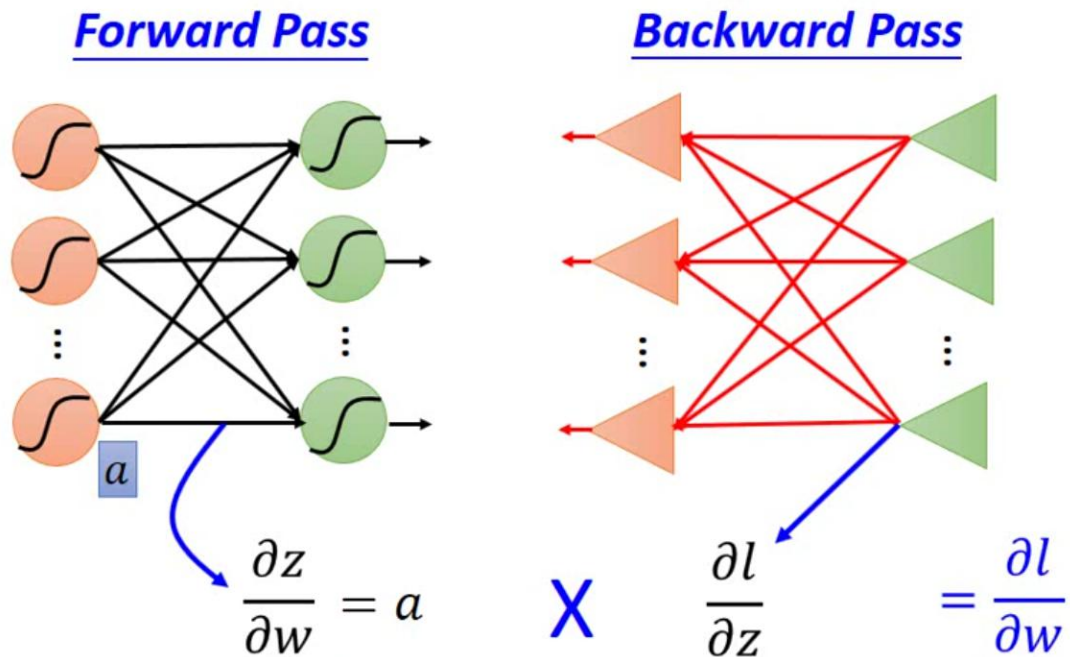
$$\frac{\partial l}{\partial w} = ? \quad \frac{\partial z}{\partial w} \frac{\partial l}{\partial z}$$

(Chain rule)

Created with EverCam.  
<http://www.camdemy.com>

3\* BP 感觉比 LR 的推导要简单

## Backpropagation – Summary



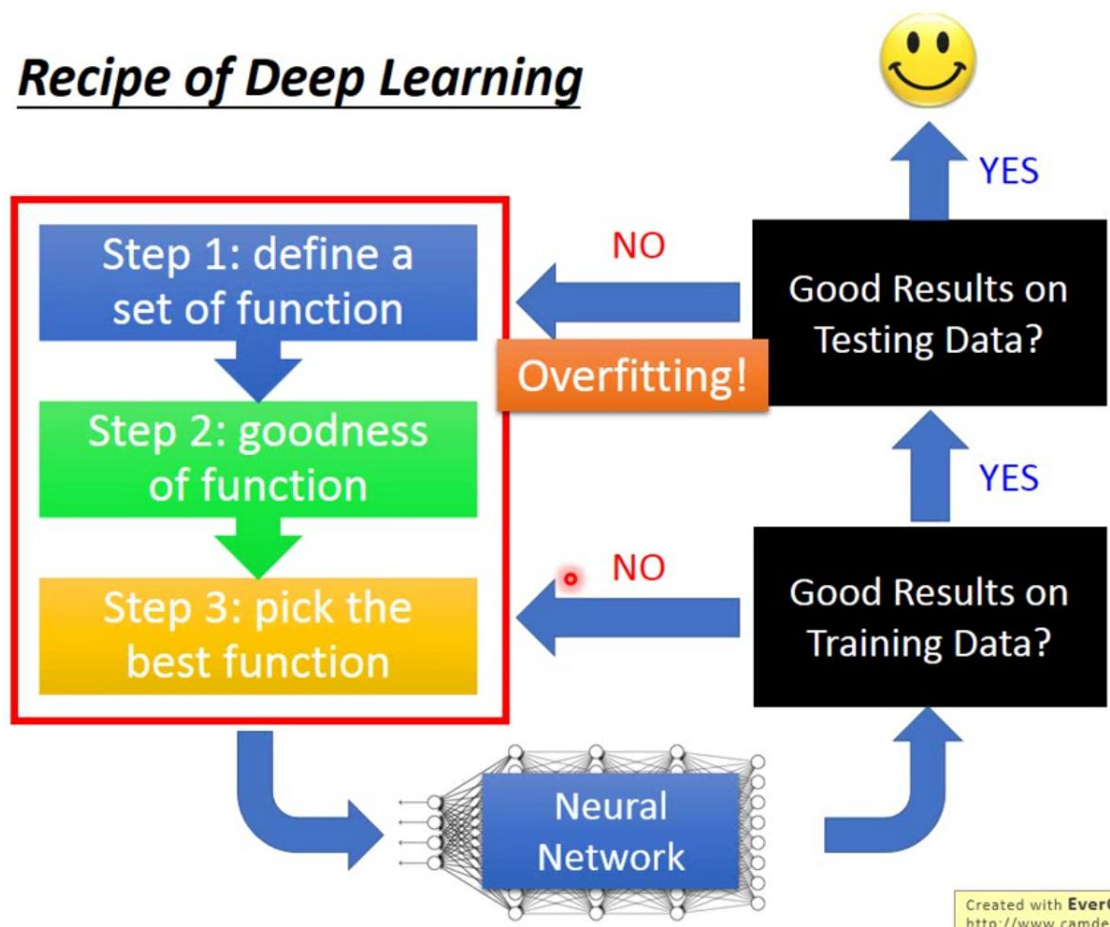
### 8\_Keras.mp4

1\* 若要使用 GPU 加速，需要设置 min\_batch 参数才行，否则不会起到加速的效果

### 9\_DNN\_tip.mp4

1\* 首先检查在 training data 上的效果，然后再检查在 testing data 上的效果

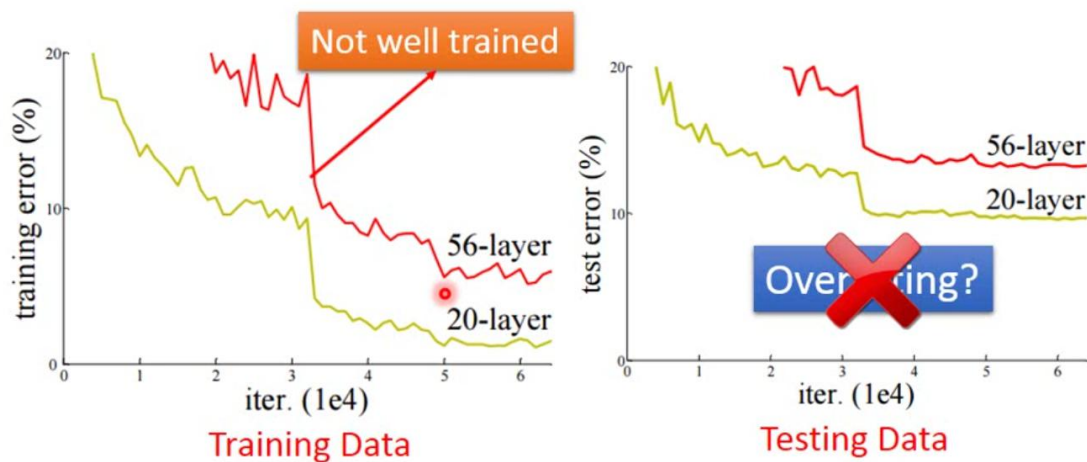
## Recipe of Deep Learning



2\* 先看右面的图再看左面的图。在右图中，在 testing data 上的结果 56 层的网络比 20 层的网络效果差，不要立刻认为 56 层的网络过拟合了，要先看 56 层的网络和 20 层的网络在训练集上的效果是怎样的？因此去看左图，发现在 training data 上 56 层的网络同样比 20 层的网络效果差，这说明 56 层的网络没有被训练好，而不是过拟合。

因此，再次强调 过拟合指的是在 training data 上的效果好，但是在 testing data 上的效果不好，才叫过拟合；只是 testing data 上的效果不好，不一定就是过拟合，还要看在 training data 上的效果。

# Do not always blame Overfitting

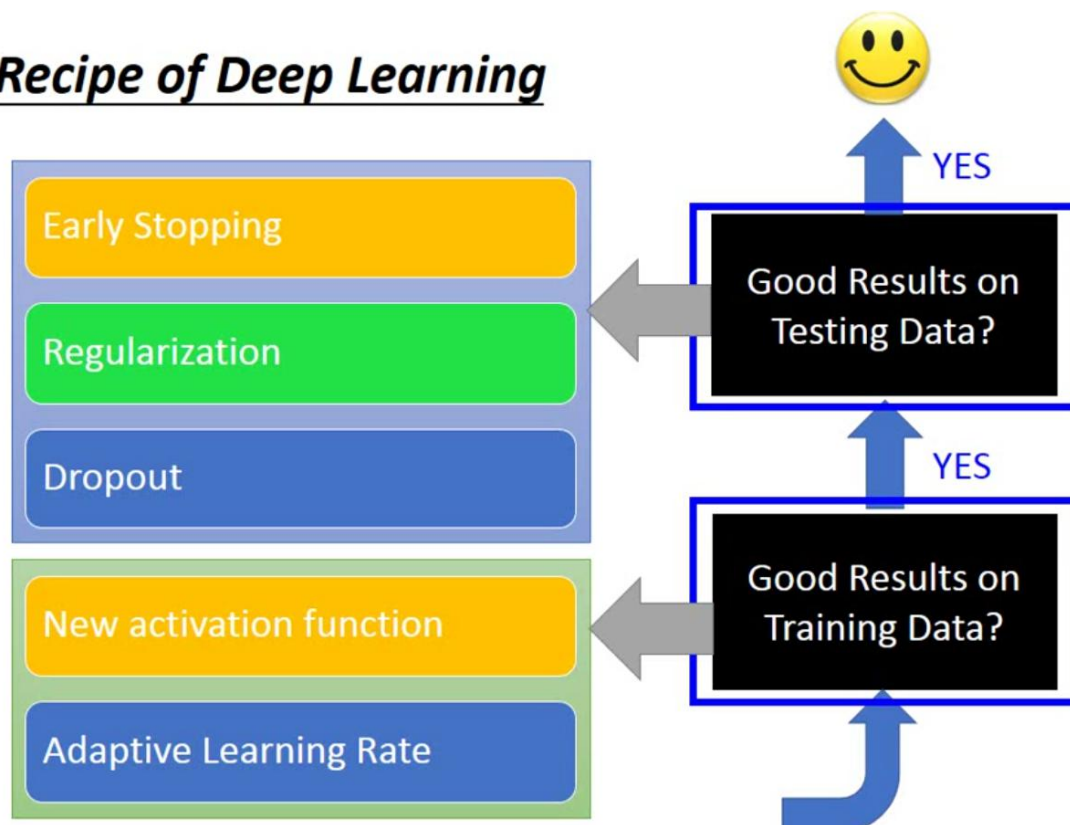


Deep Residual Learning for Image Recognition  
<http://arxiv.org/abs/1512.03385>

Created with EverCam

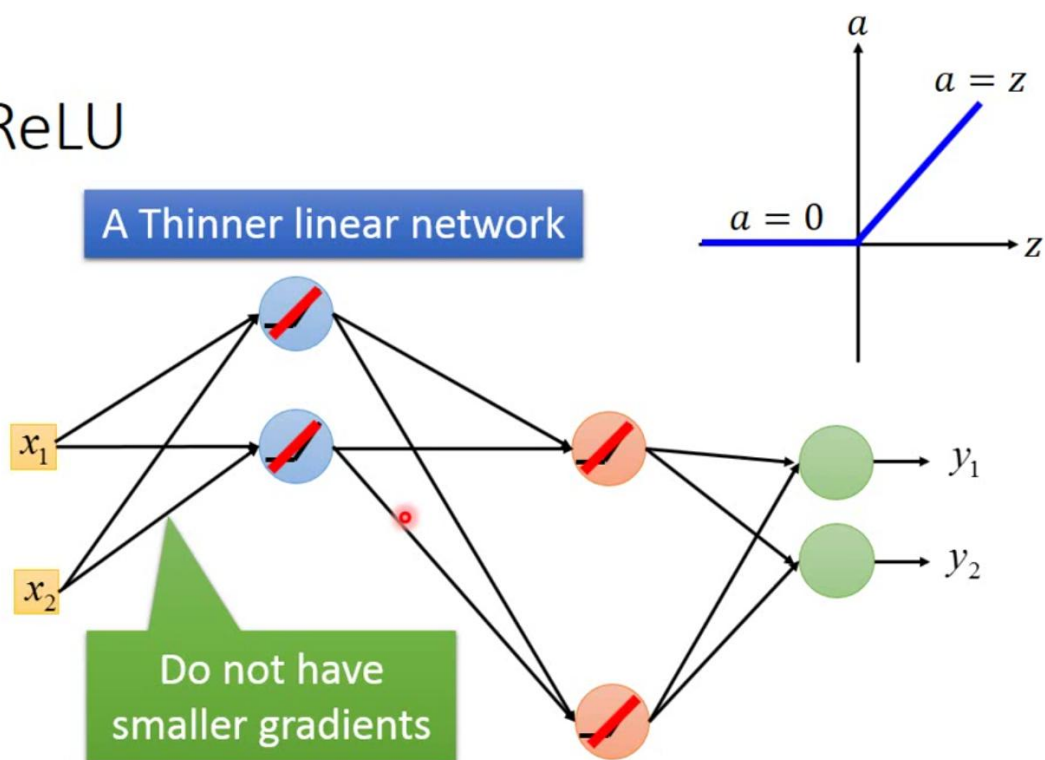
3\*

## Recipe of Deep Learning



4\* 使用 ReLU 激活函数只是在局部范围内是线性的,整体来看整个 function 仍然是非线性的

# ReLU



自己的理解：不同的  $w$  和  $b$  会导致不同的输出，因此可能导致往下一层传递的输入个数也是不同的（小于零的被过滤掉，但不同的  $w$  和  $b$  会导致小于零的个数是不同的，也就相当于网络的结构是动态调整的）

5\* RMSProp 是一种动态调整学习率的 optimizer 方法，RMSProp 考虑了之前的梯度值。



# RMSProp

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \quad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \quad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1 - \alpha)(g^1)^2}$$

$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \quad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1 - \alpha)(g^2)^2}$$

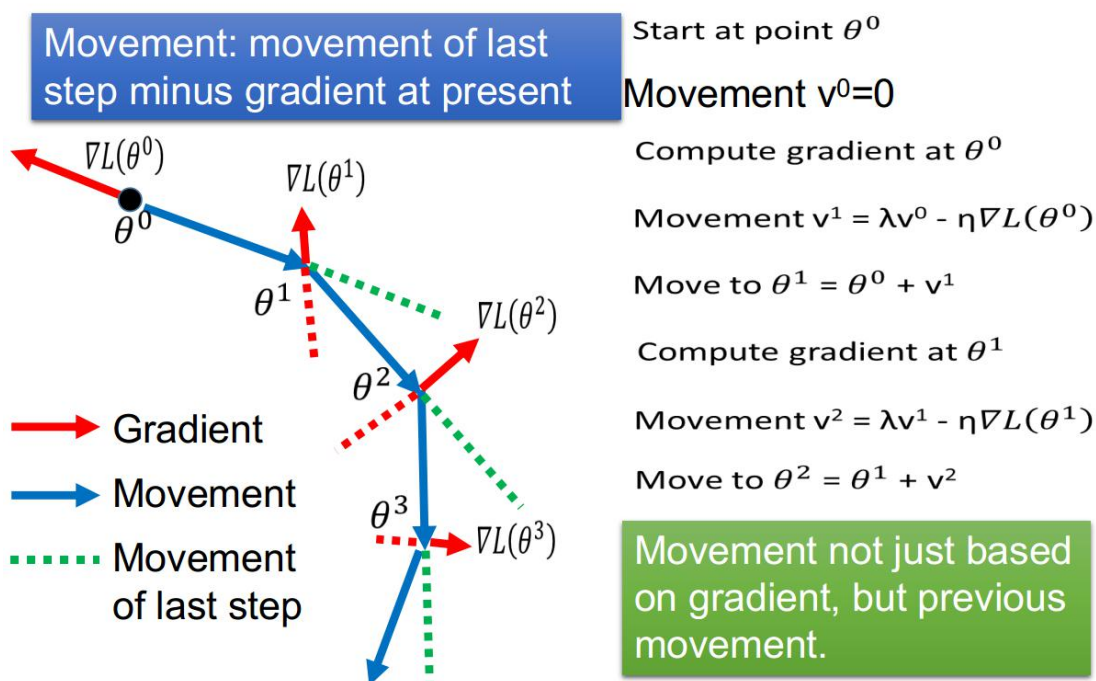
⋮

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \quad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1 - \alpha)(g^t)^2}$$

Root Mean Square of the gradients  
with previous gradients being  
decayed

Momentum 是一种动态调整学习率的 optimizer 方法，Momentum 考虑了上一次的梯度（向量，包括方向和大小）。

# Momentum



Adam 是综合考虑了 RMSProp 和 Momentum 的 optimizer 方法

# Adam

## RMSProp + Momentum

**Algorithm 1:** Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation.  $g_t^2$  indicates the elementwise square  $g_t \odot g_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

**Require:**  $\alpha$ : Stepsize

**Require:**  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates

**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$

**Require:**  $\theta_0$ : Initial parameter vector

$m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)  $\rightarrow$  for momentum

$v_0 \leftarrow 0$  (Initialize 2<sup>nd</sup> moment vector)  $\rightarrow$  for RMSprop

$t \leftarrow 0$  (Initialize timestep)

**while**  $\theta_t$  not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)

**end while**

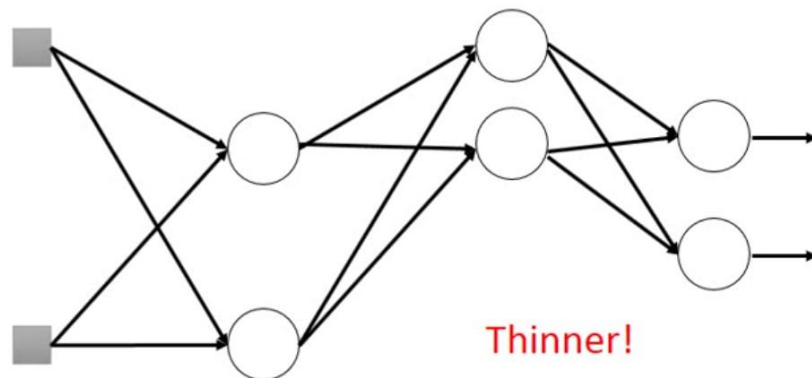
**return**  $\theta_t$  (Resulting parameters)

Created with EverCam

6\* Dropout 是在每个 minibatch 内选择要 Dropout 的神经元

## Dropout

### Training:



➤ Each time before updating the parameters

- Each neuron has  $p\%$  to dropout



**The structure of the network is changed.**

- Using the new network for training

For each mini-batch, we resample the dropout neurons

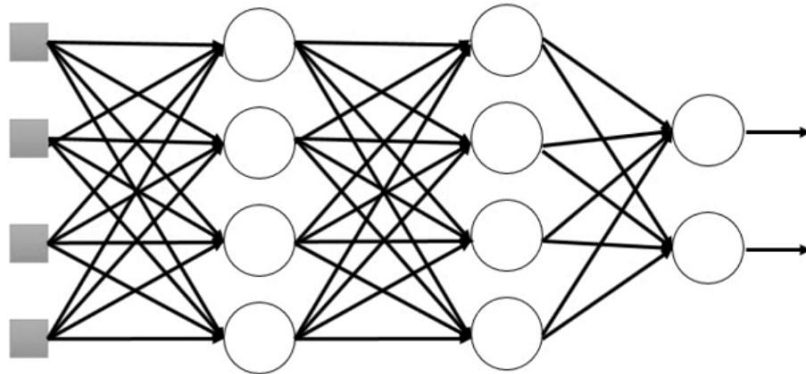
Created with E

7\* Dropout 是在 training 的过程中才会有, 在 testing 的过程中不进行 Dropout;

在 testing 时，所有的权重要乘上 $(1-p\%)$

## Dropout

Testing:



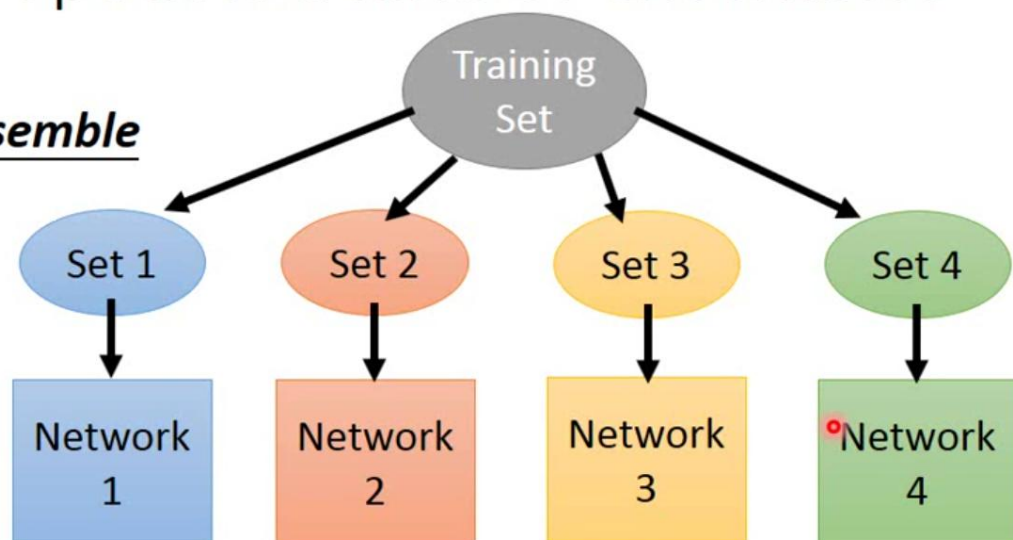
➤ No dropout

- If the dropout rate at training is  $p\%$ , all the weights times  $1-p\%$
- Assume that the dropout rate is 50%.  
If a weight  $w = 1$  by training, set  $w = 0.5$  for testing.

8\* Ensemble

Dropout is a kind of ensemble.

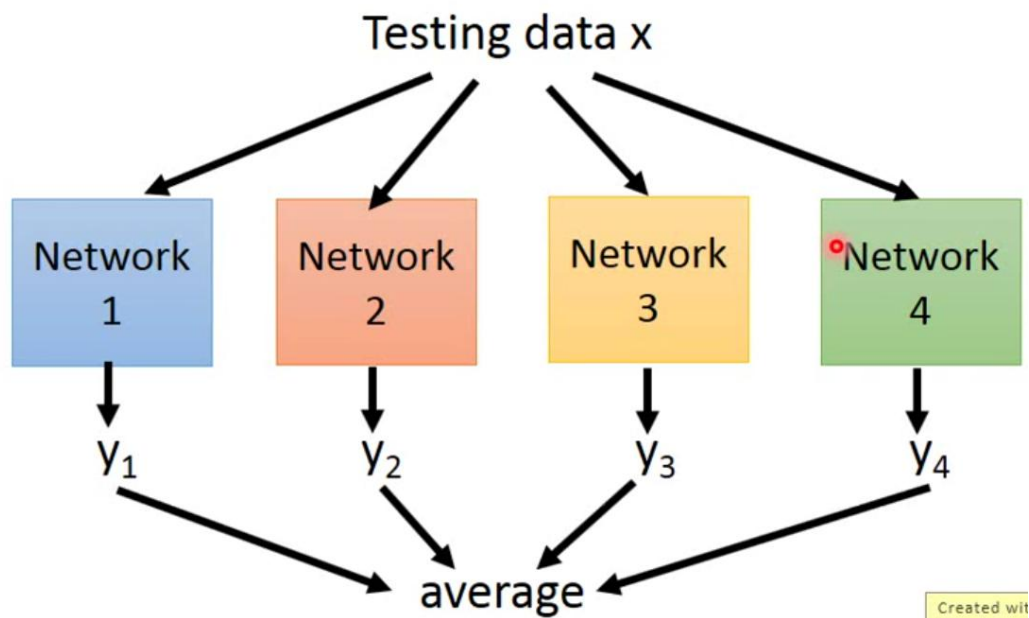
Ensemble



Train a bunch of networks with different structures

# Dropout is a kind of ensemble.

## Ensemble



## 19\_RNN1.mp4

1\* 对于 RNN 来说，input 的顺序会影响到输出的结果，也就是说 RNN 会考虑 input 的顺序

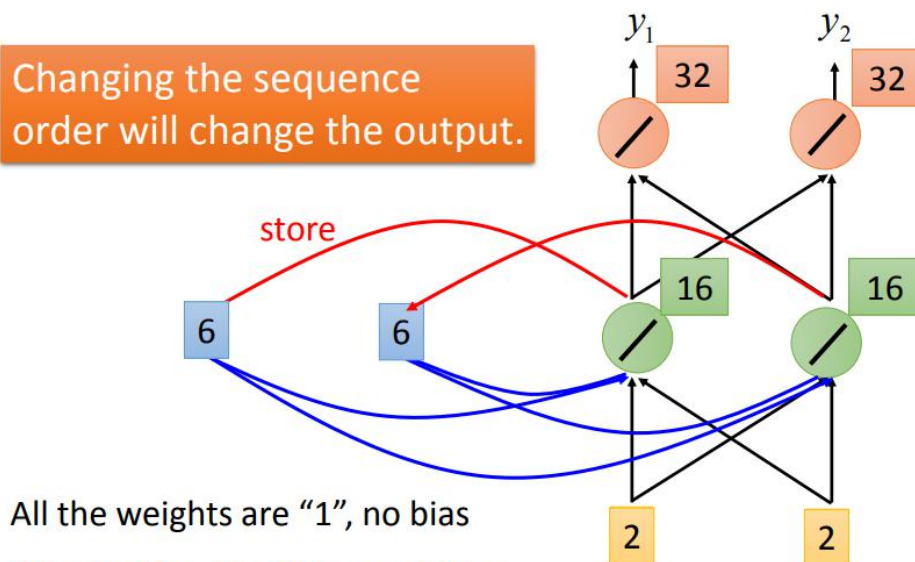


## Example

Input sequence:  $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} \dots$

output sequence:  $\begin{bmatrix} 4 \\ 4 \end{bmatrix} \begin{bmatrix} 12 \\ 12 \end{bmatrix} \begin{bmatrix} 32 \\ 32 \end{bmatrix}$

Changing the sequence order will change the output.

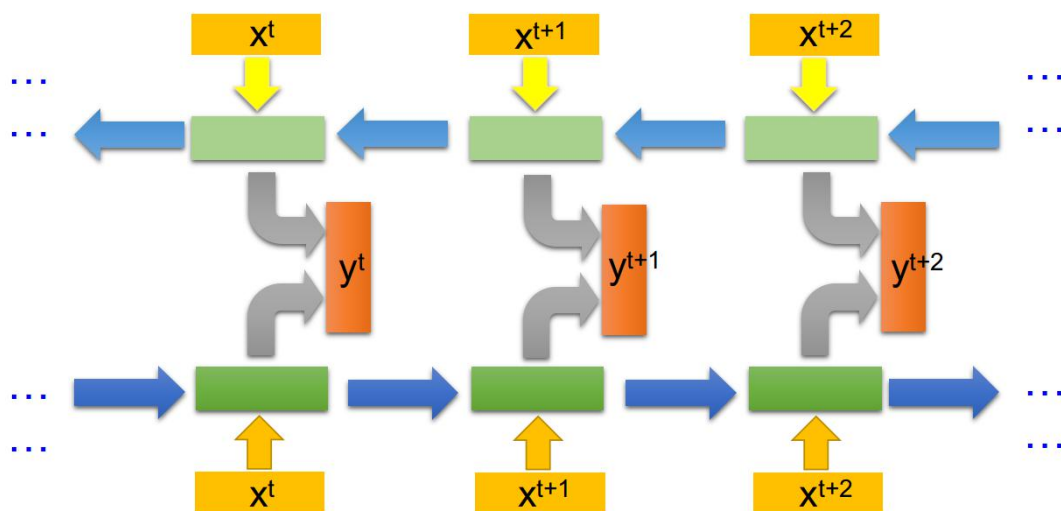


All the weights are "1", no bias

All activation functions are linear

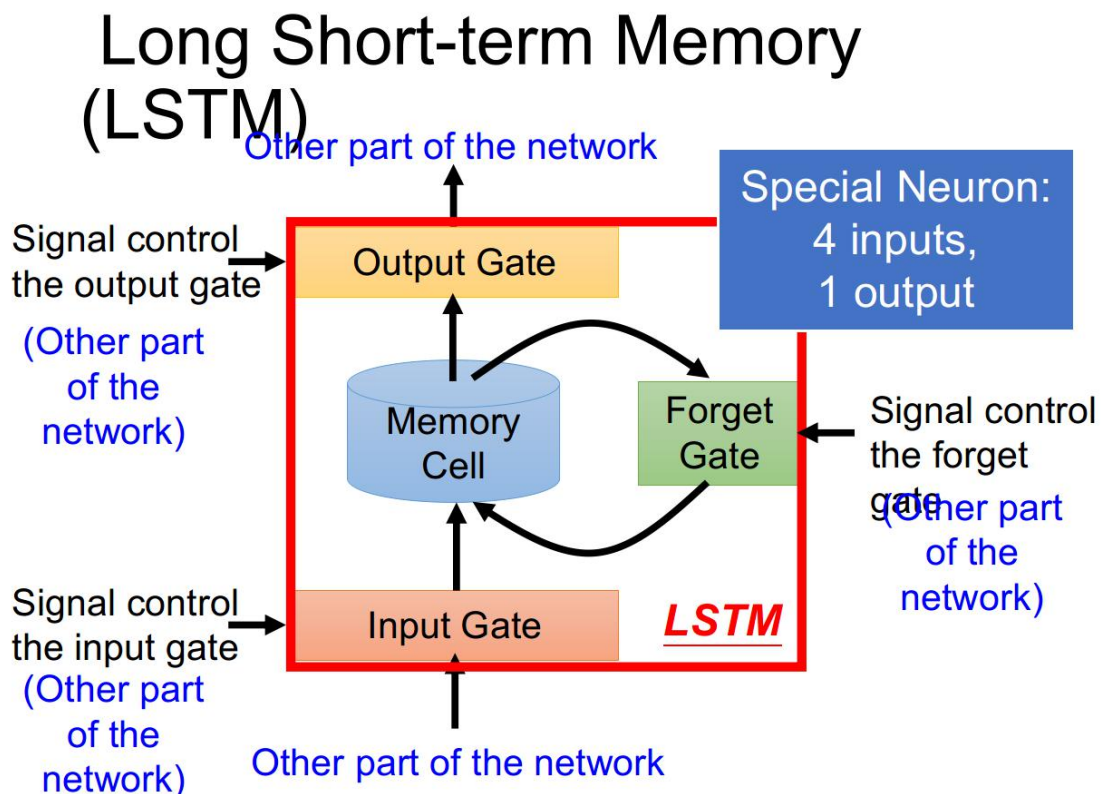
2\* 双向 RNN 有时比单向 RNN 好的原因是：双向 RNN 有更大的视野（看到的上下文信息更多），双向 RNN 不仅能够看到当前输入前面的输入信息（往右方向的 RNN），还能够看到当前输入后面的输入信息（往左方向的 RNN）。

## Bidirectional RNN





3\* LSTM 的神经元是一种特殊的神经元，普通的神经元有一个输入一个输出，但 LSTM 神经元包括四个输入一个输出，四个输入分别是：输入数据、Input Gate 控制信息、Output Gate 控制信息、Forget Gate 控制信息



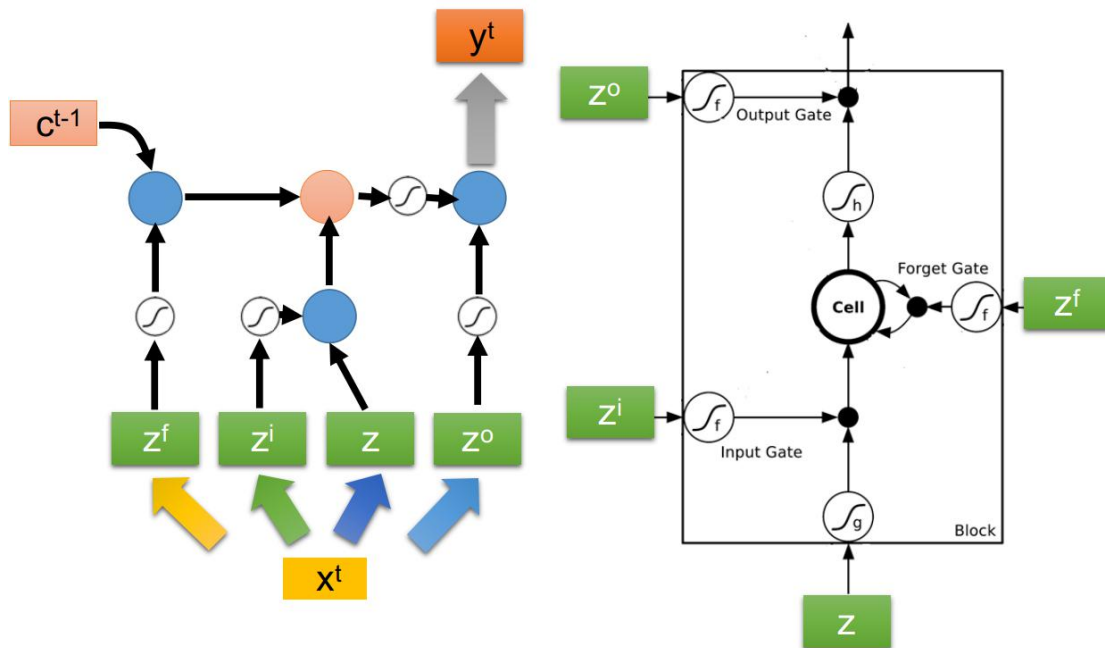
4\* LSTM: Long Short-term Memory 连词符号要放在 Short 和 term 之间, LSTM

表示的是比较长 (long) 的 Short-term 记忆网络

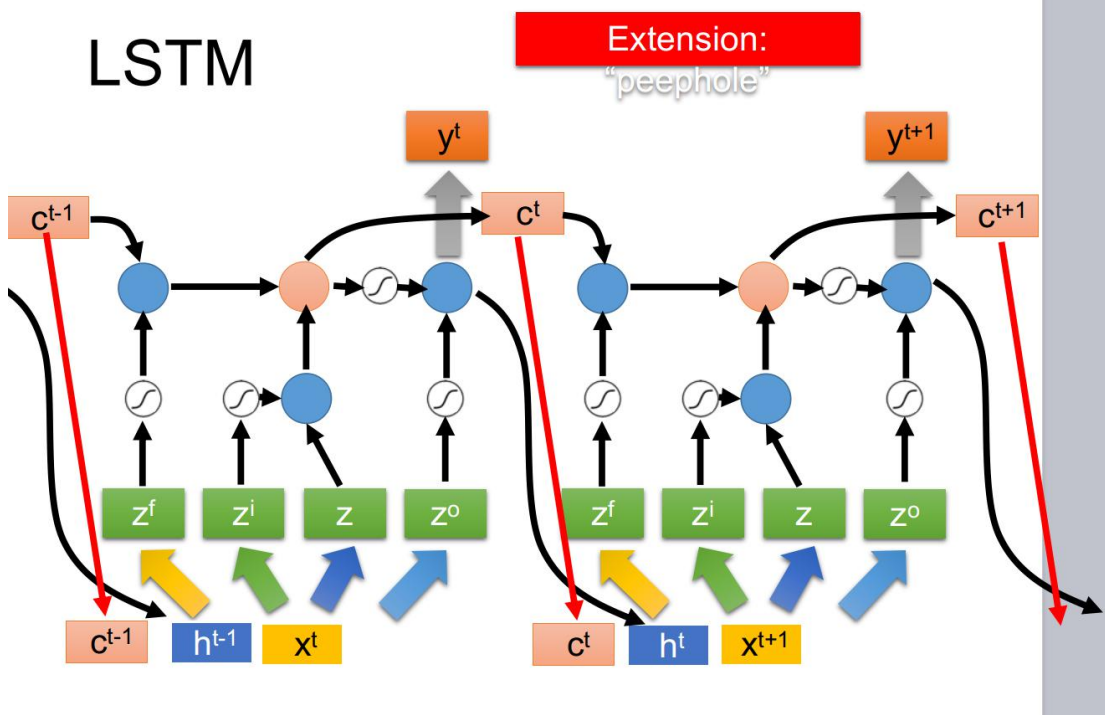
5\* LSTM 的三个门的控制信息也是与 input 直接相关的

6\* LSTM 的记忆信息的传递 (与 RNN 本质上是类似的)

# LSTM



# LSTM

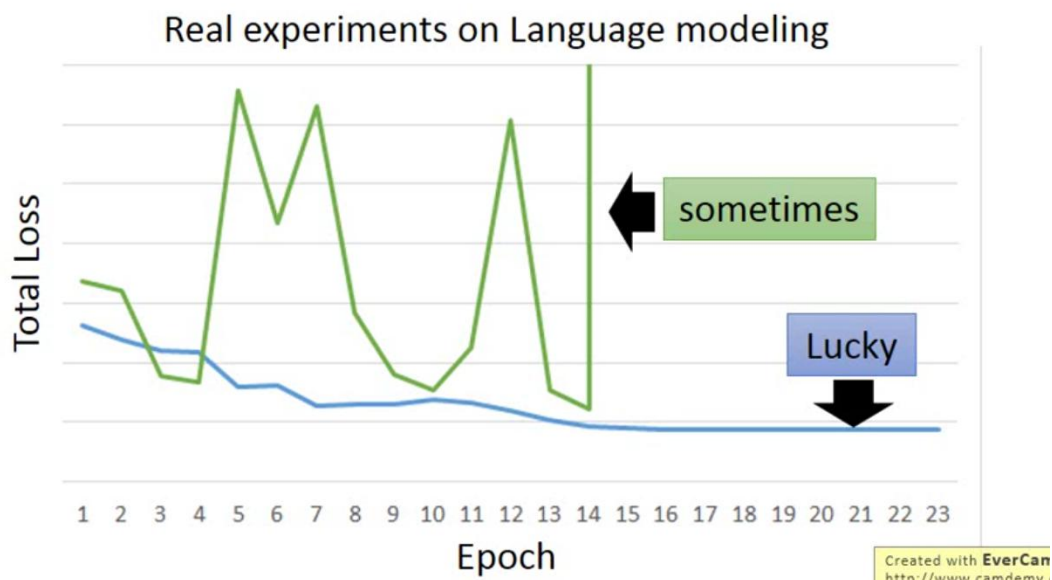


19\_RNN2.mp4

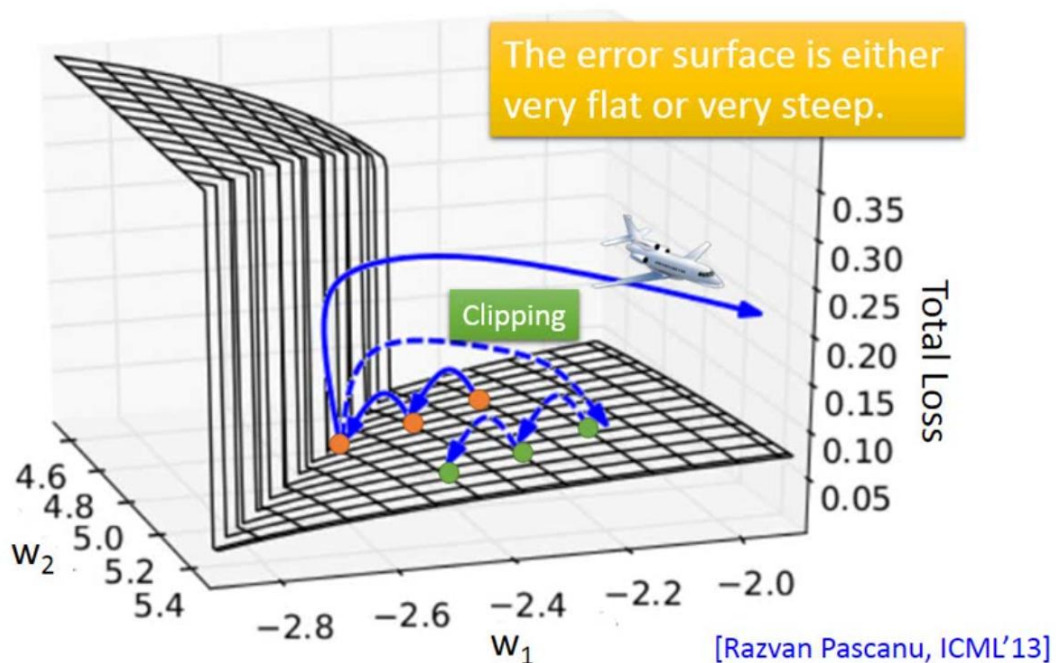
1\* RNN 并不总是容易训练的

Unfortunately .....

- RNN-based network is not always easy to learn



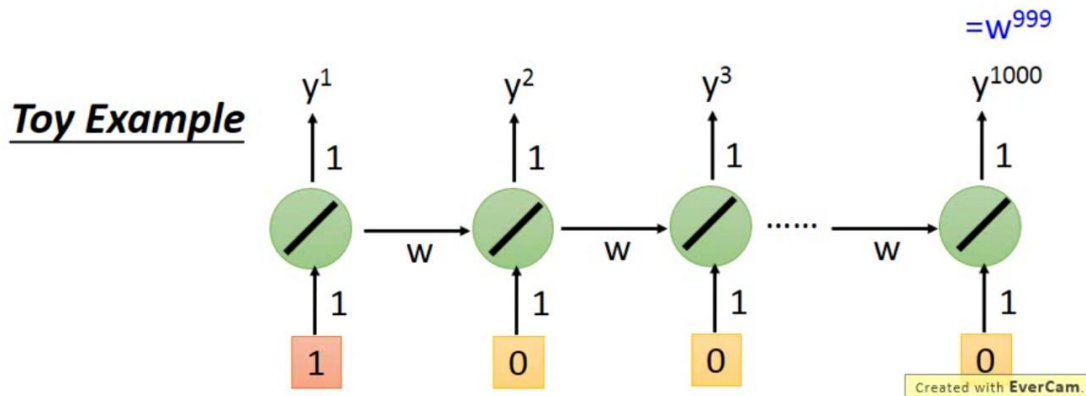
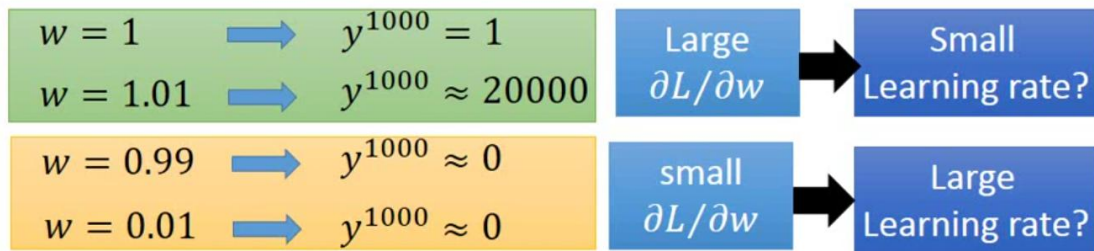
The error surface is rough.



为什么会有上面的情况呢 (very flat or very steep), 为什么 RNN 很难训练呢?

因为同一个 weight 会在不同的时刻被反复的使用

# Why?



2\* RNN 很少使用 ReLU 作为激活函数，使用 ReLU 作为激活函数的训练效果通常很差 视频 13'20"

3\* 为什么采用 LSTM 而不是 RNN ? 因为 LSTM 能够解决梯度消失的问题，但梯度爆炸的问题仍然存在 ( i.e. 特别陡峭的地方仍然存在，但没有特别平坦的地方 )

为什么 LSTM 能够解决梯度消失的问题 ?

如下图所示，在 LSTM 中 input 和 Memory 的值是相加的关系，只要 forget gate 开启，memory 中就永远存在数据，不会出现梯度消失

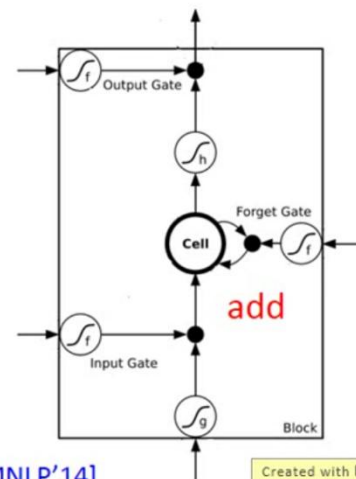
# Helpful Techniques

- Long Short-term Memory (LSTM)

- Can deal with gradient vanishing (not gradient explode)
- Memory and input are added
- The influence never disappears unless forget gate is closed

➡ No Gradient vanishing  
(If forget gate is opened.)

Gated Recurrent Unit (GRU):  
simpler than LSTM



[Cho, EMNLP'14]

Created with EverCam.  
<http://www.camdemy.com>