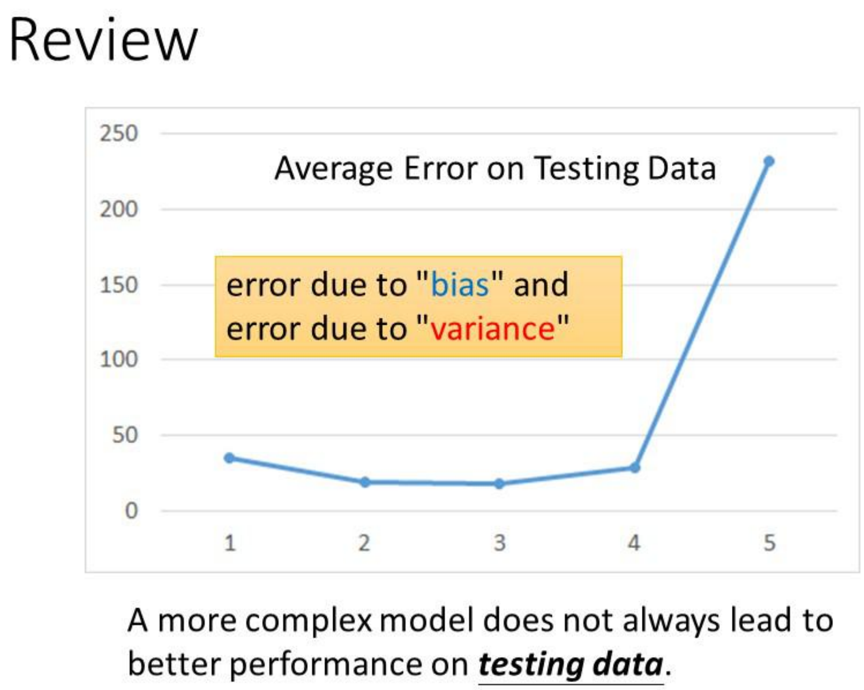
每看完一个视频，要用自己的理解把这个视频的内容陈述出来（下面的内容多数是直接翻译或者把李宏毅老师的原话写入本文档，如果是自己总结和理解的内容会进行说明）。

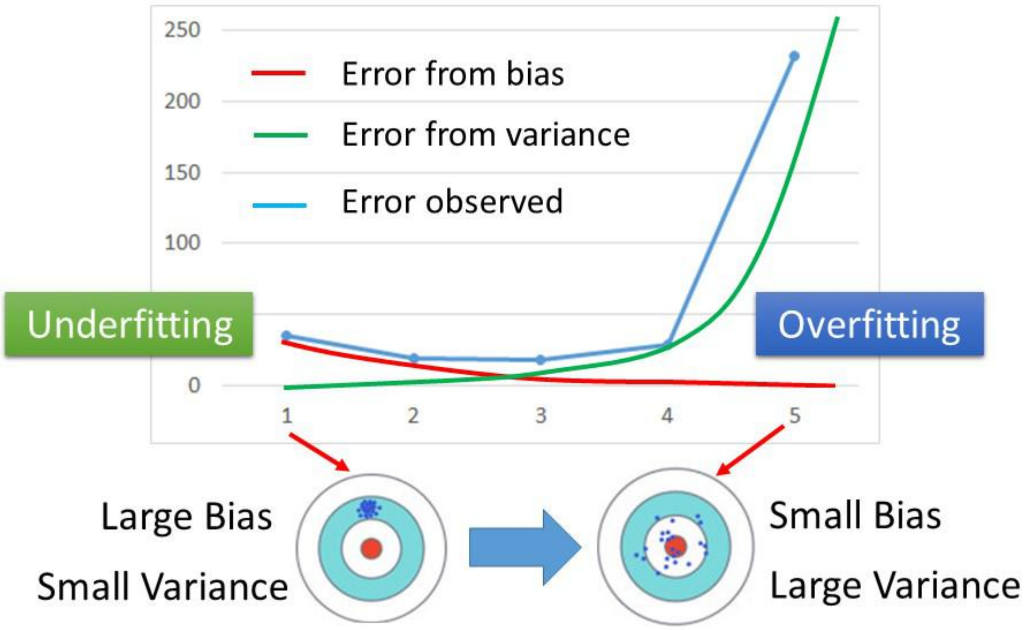
2_Error.mp4

1* 误差来自于 bias 和 variance



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

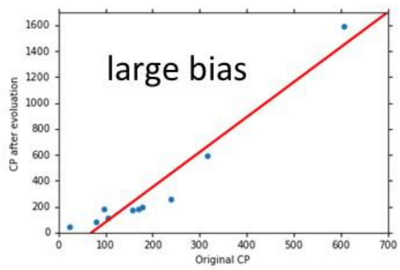
Bias v.s. Variance



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

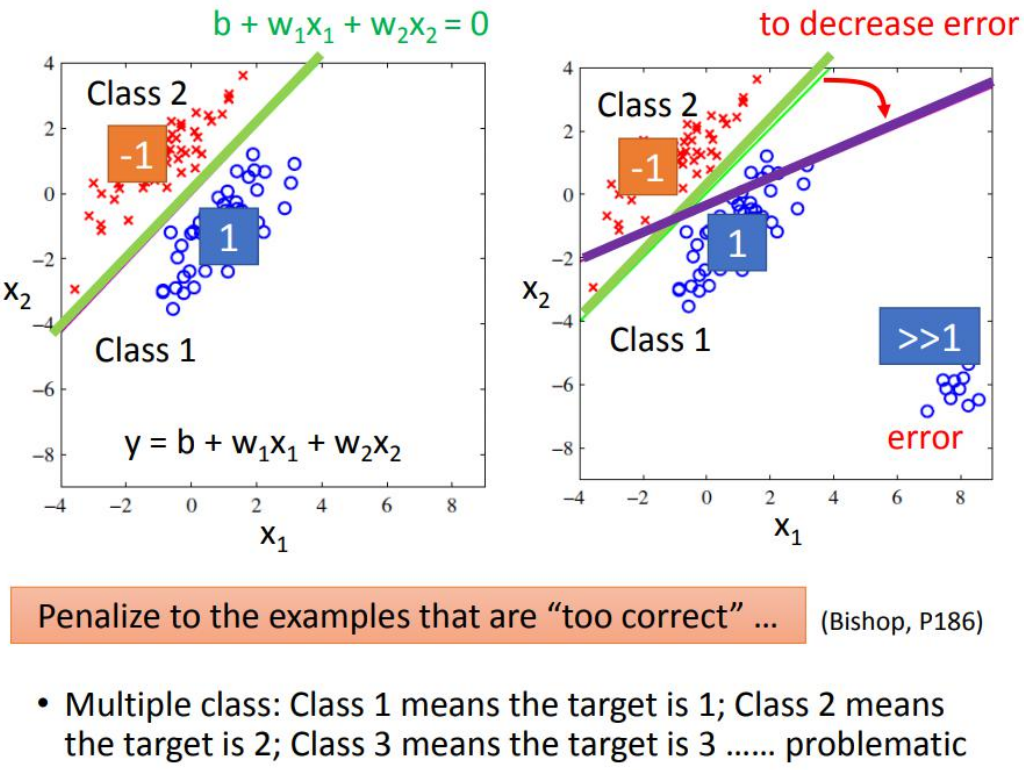


4_Classification.mp4

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



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

	Logistic Regression	*Linear Regression*
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
Step 2:	Training data: $(x^n, \hat{y}^n)$ $\boxed{\hat{y}^n\text{: 1 for class 1, 0 for class 2}}$ $L(f) = \sum_n l(f(x^n), \hat{y}^n)$	Training data: $(x^n, \hat{y}^n)$ $\hat{y}^n$: a real number $L(f) = \frac{1}{2}\sum_n (f(x^n) - \hat{y}^n)^2$

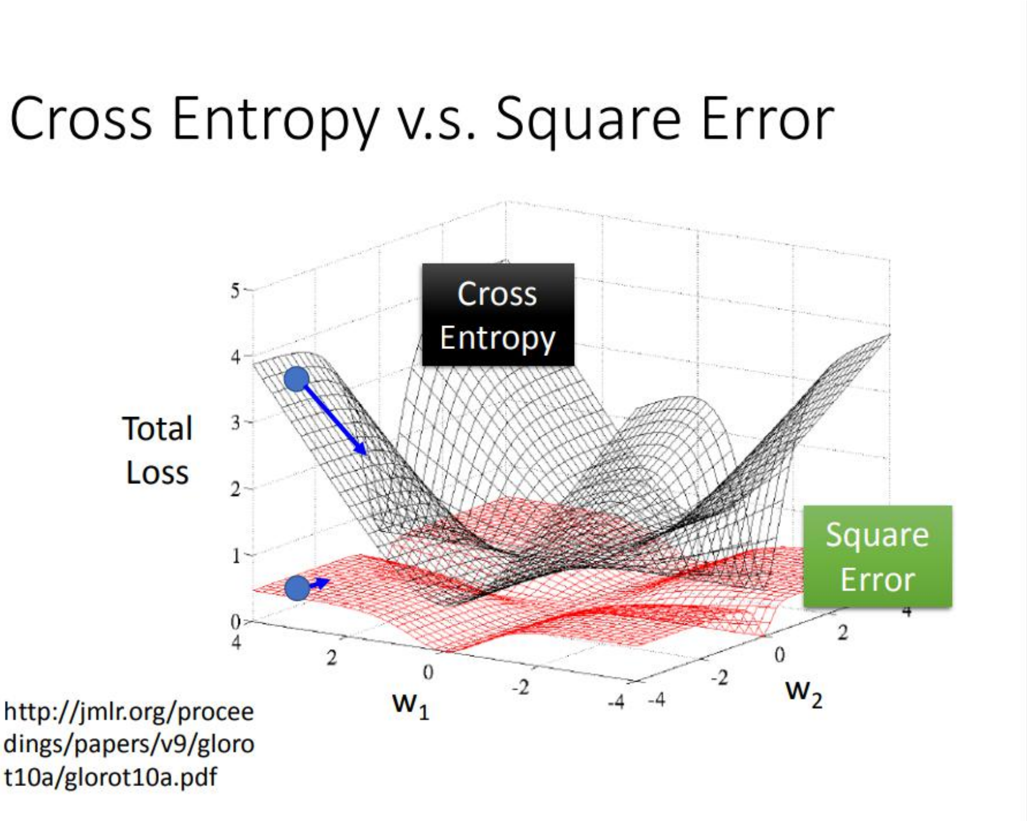
Step 3:

Logistic regression: $$w_i \leftarrow w_i - \eta \sum_n -\left(\underline{\hat{y}^n - f_{w,b}(x^n)}\right) x_i^n$$

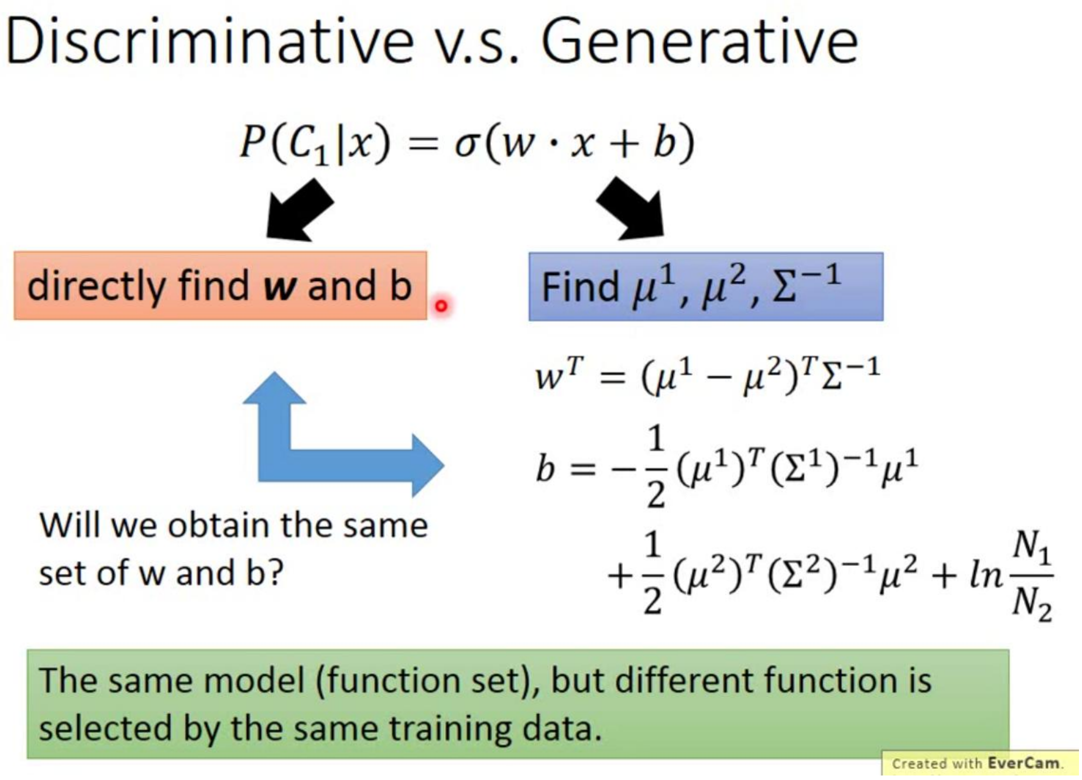
Linear regression: $$w_i \leftarrow w_i - \eta \sum_n -\left(\underline{\hat{y}^n - f_{w,b}(x^n)}\right) x_i^n$$

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，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



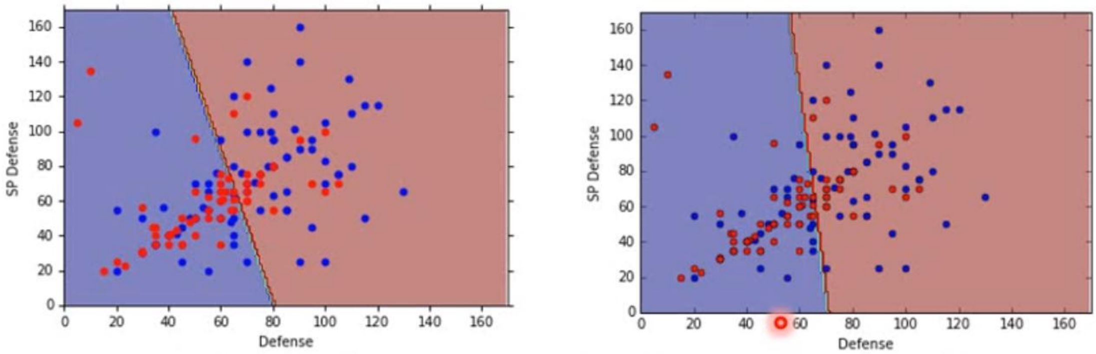
3* 对于同样的训练数据，当分别使用 Discriminative 方法和 Generative 方法训练模型时，得到的模型通常是不同的。



4* 通常认为 Discriminative 模型比 Generative 模型表现得要更好

Generative v.s. Discriminative

Generative ***Discriminative***



All: total, hp, att, sp att, de, sp de, speed

73% accuracy 79% accuracy

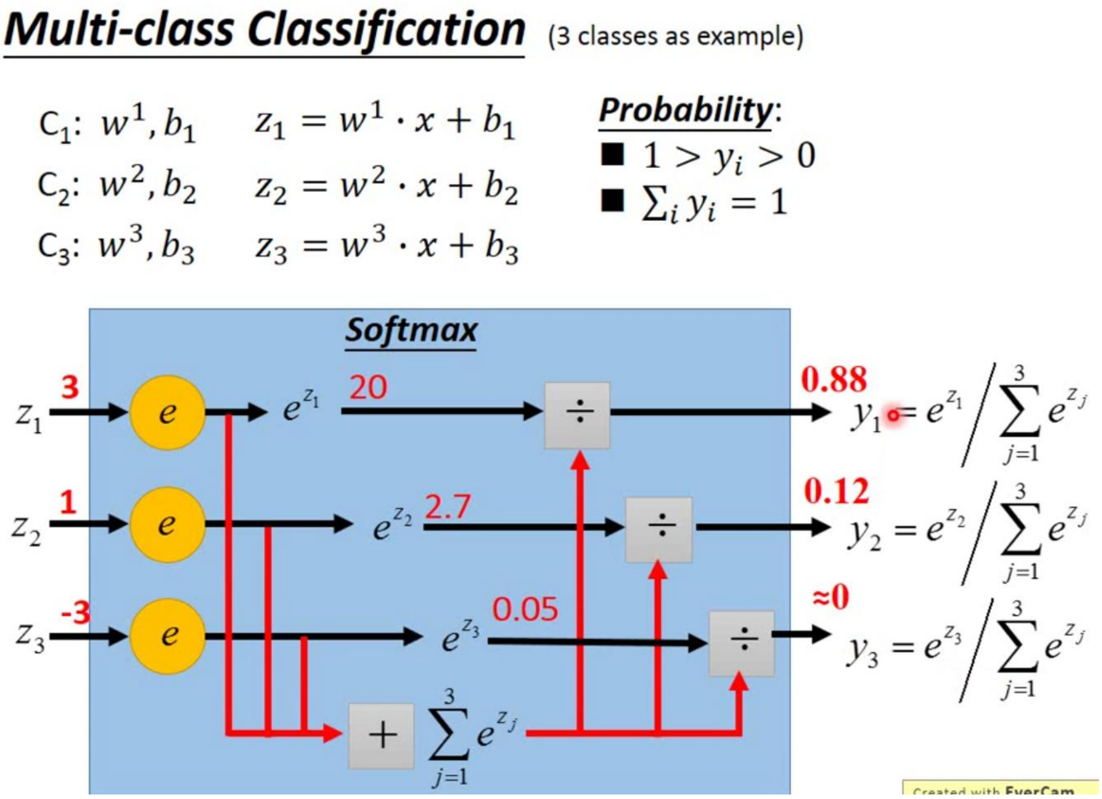
5* Generative 模型的好处

Generative v.s. Discriminative

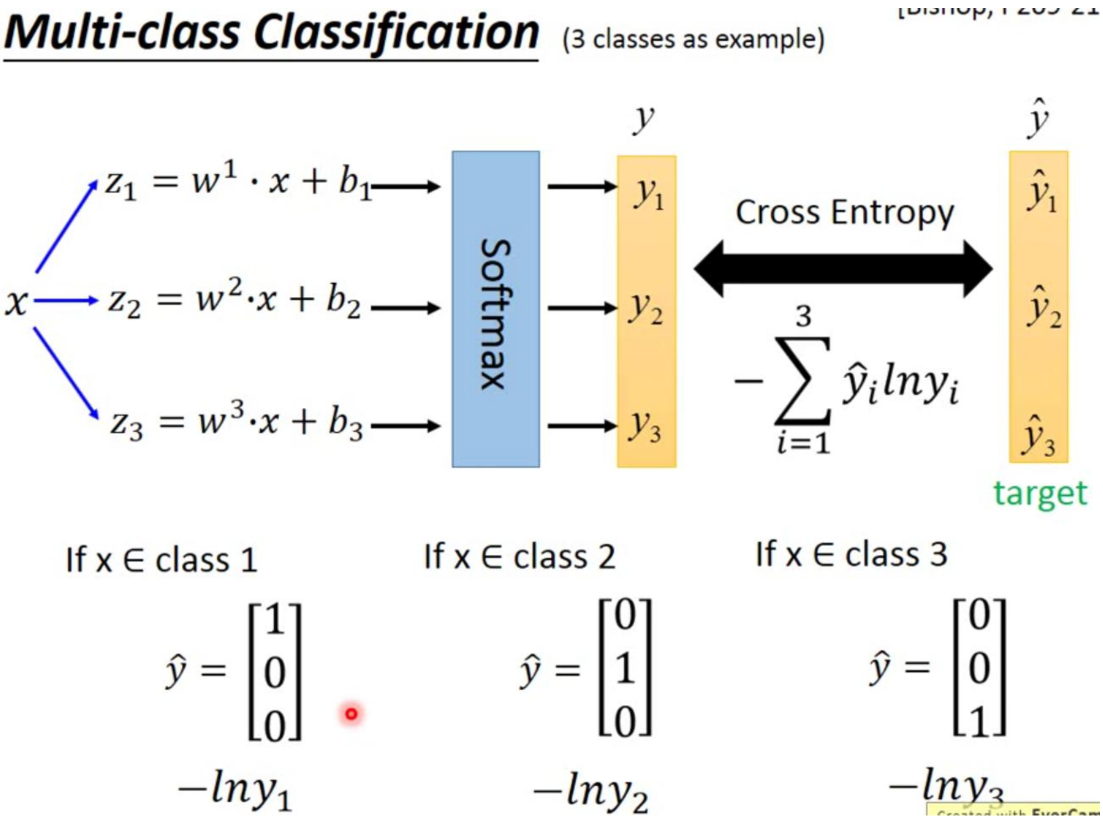
- Usually people believe discriminative model is better
- Benefit of generative model
  - With the assumption of probability distribution
    - less training data is needed
    - more robust to the noise
  - Priors and class-dependent probabilities can be estimated from different sources.

6* softmax 函数：

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

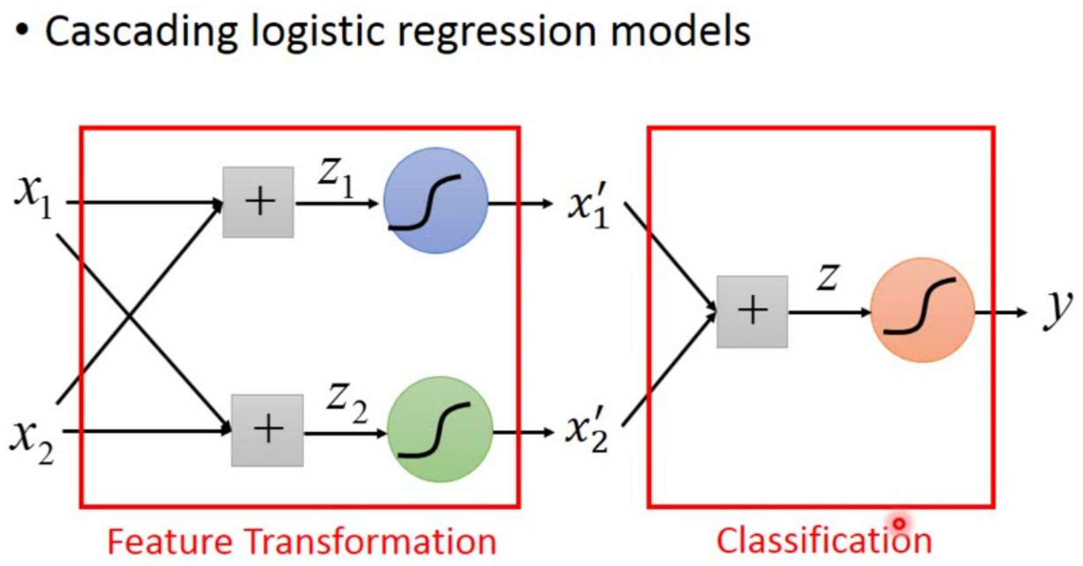


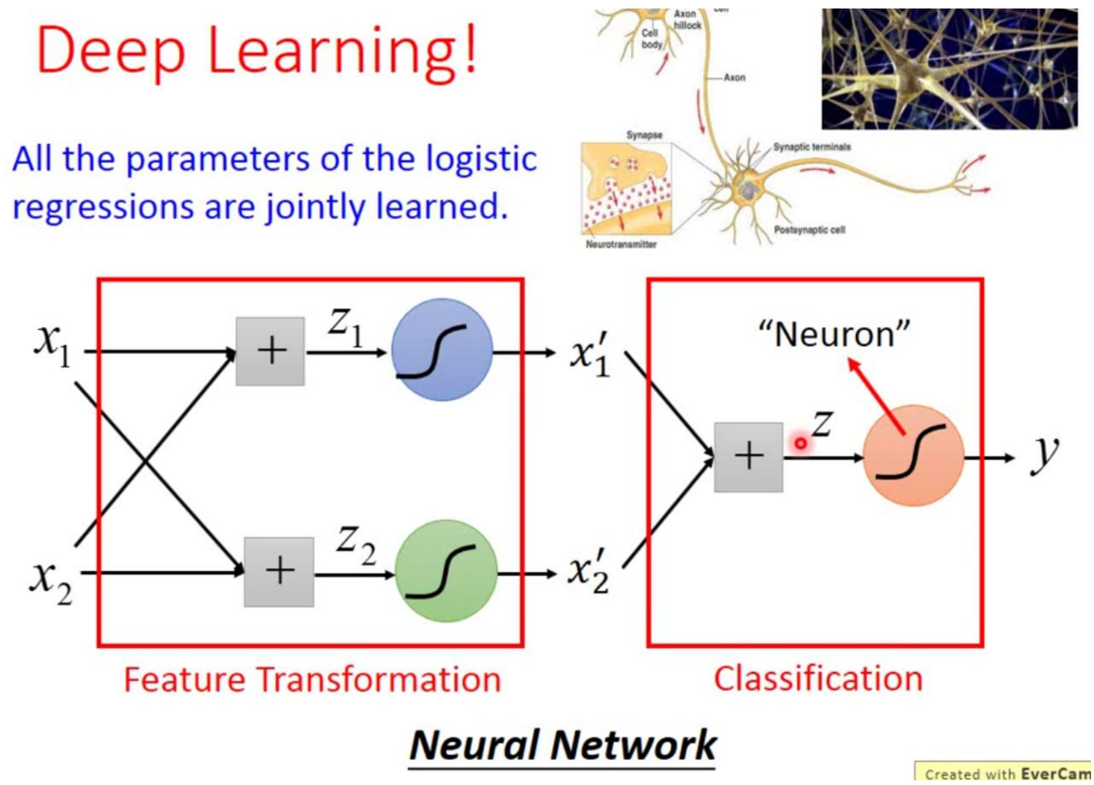
7* Cross Entropy: 交叉熵公式如图中所示



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

1* 链式法则





7_BP.mp4

1* 链式法则

Chain Rule

Case 1 $y = g(x) \quad z = h(y)$

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

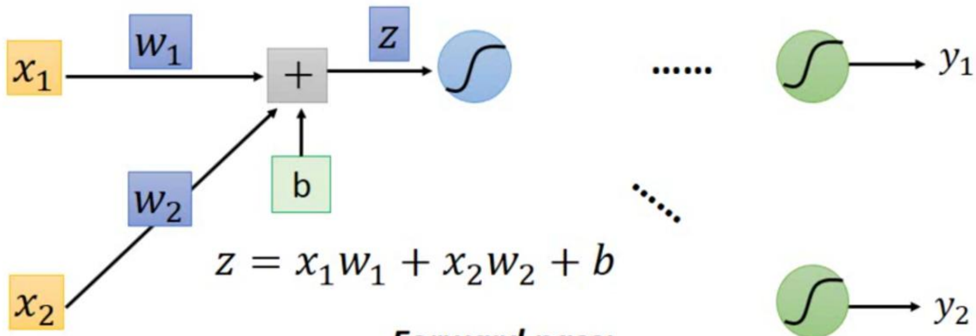
Case 2

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

$$\begin{tikzcd}[column sep=small] & \Delta x \arrow[dr] & \\ \Delta s \arrow[ur] \arrow[dr] & & \Delta z \\ & \Delta y \arrow[ur] & \end{tikzcd} \qquad \frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

2* 计算 ∂l/∂z 的过程实际上是一个反向传播的过程

Backpropagation



$$z = x_1 w_1 + x_2 w_2 + b$$

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

(Chain rule)

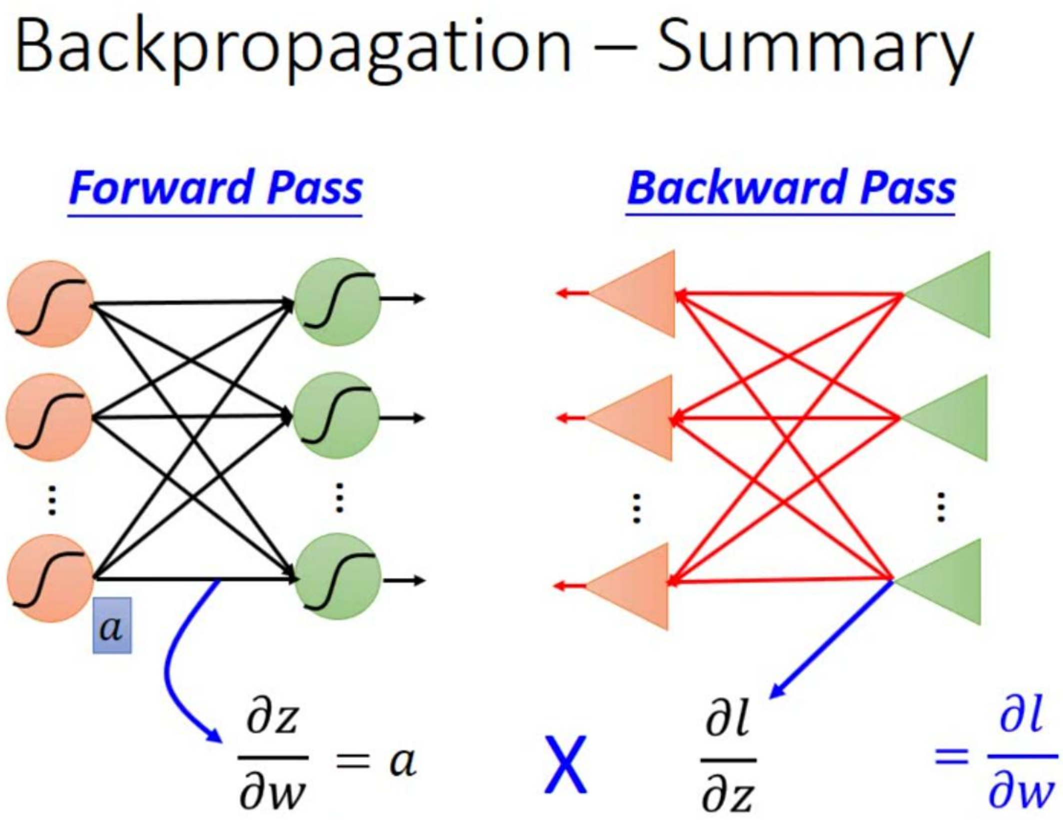
Forward pass:

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

Backward pass:

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

3* BP 感觉比 LR 的推导要简单

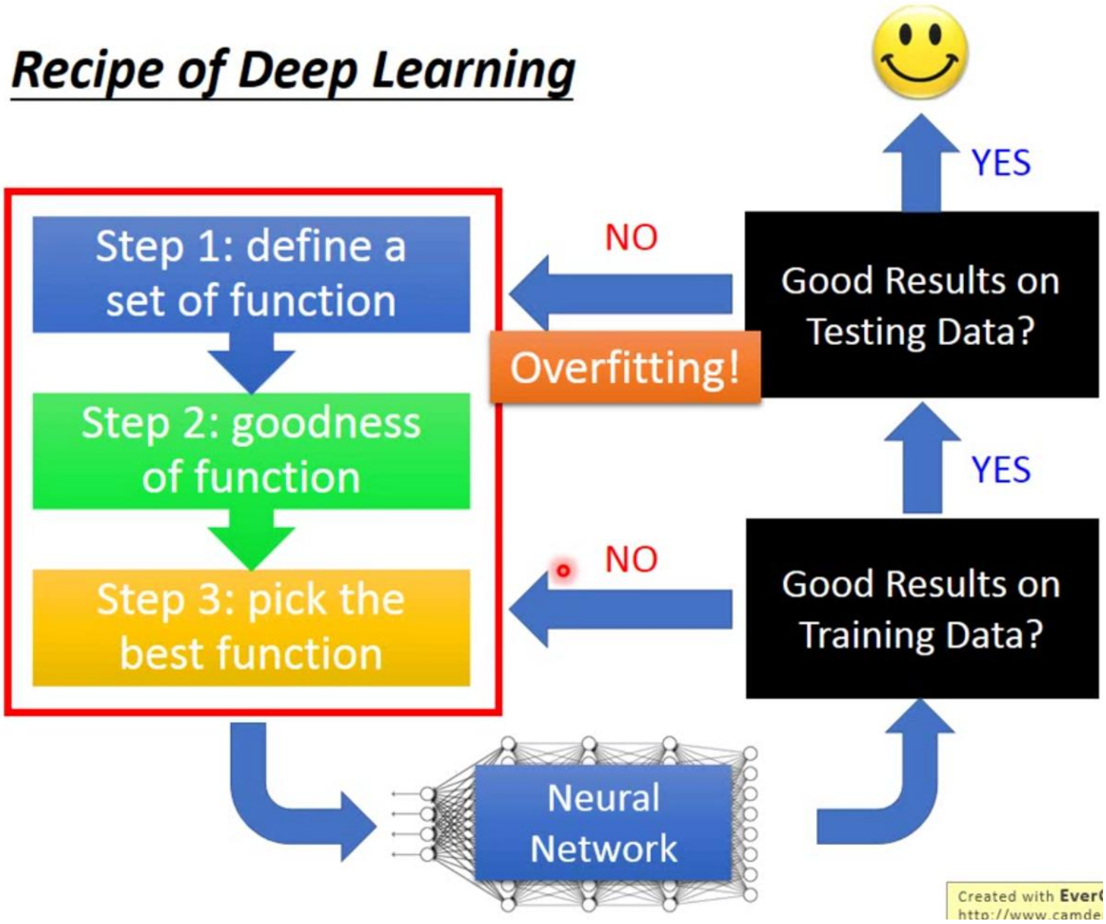


8_Keras.mp4

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

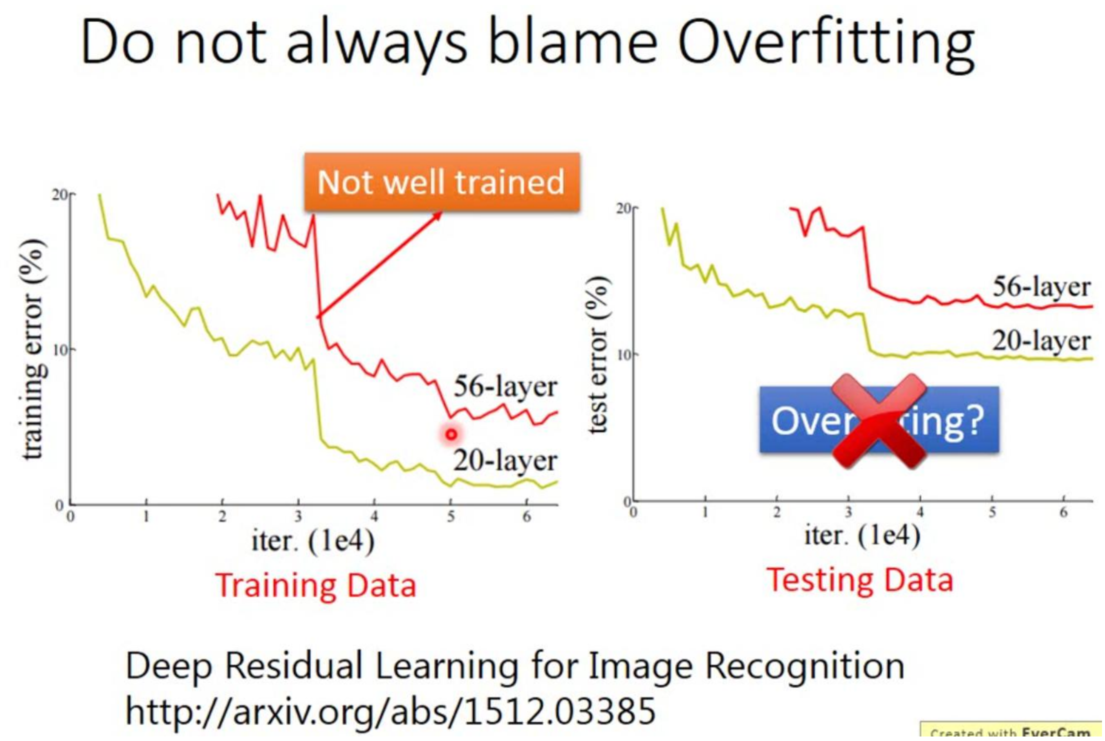
9_DNN_tip.mp4

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

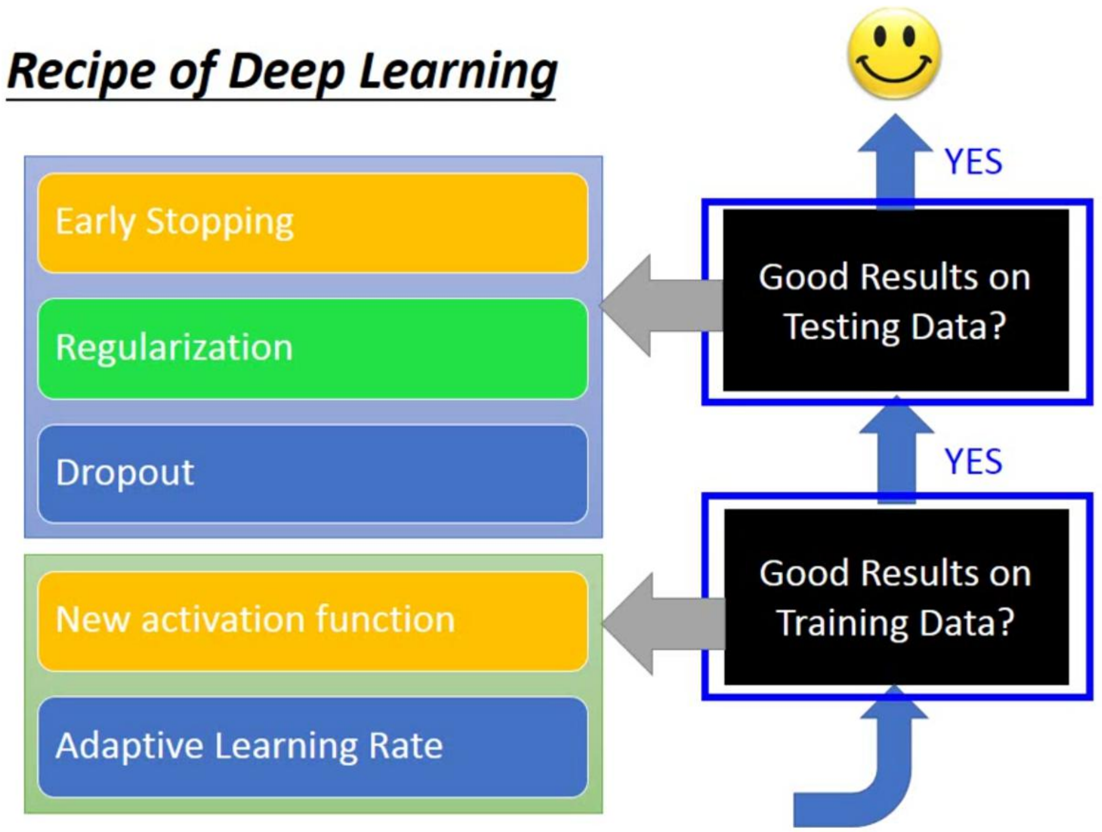


2* 先看右面的图再看左面的图。在右图中，在 testing data 上的结果 56 层的网络比 20 层的网络效果差，不要立刻认为 56 层的网络过拟合了，要先看 56 层的网络和 20 层的网络在训练集上的效果是怎样的？因此去看左图，发现在 training data 上 56 层的网络同样比 20 层的网络效果差，这说明 56 层的网络没有被训练好，而不是过拟合。

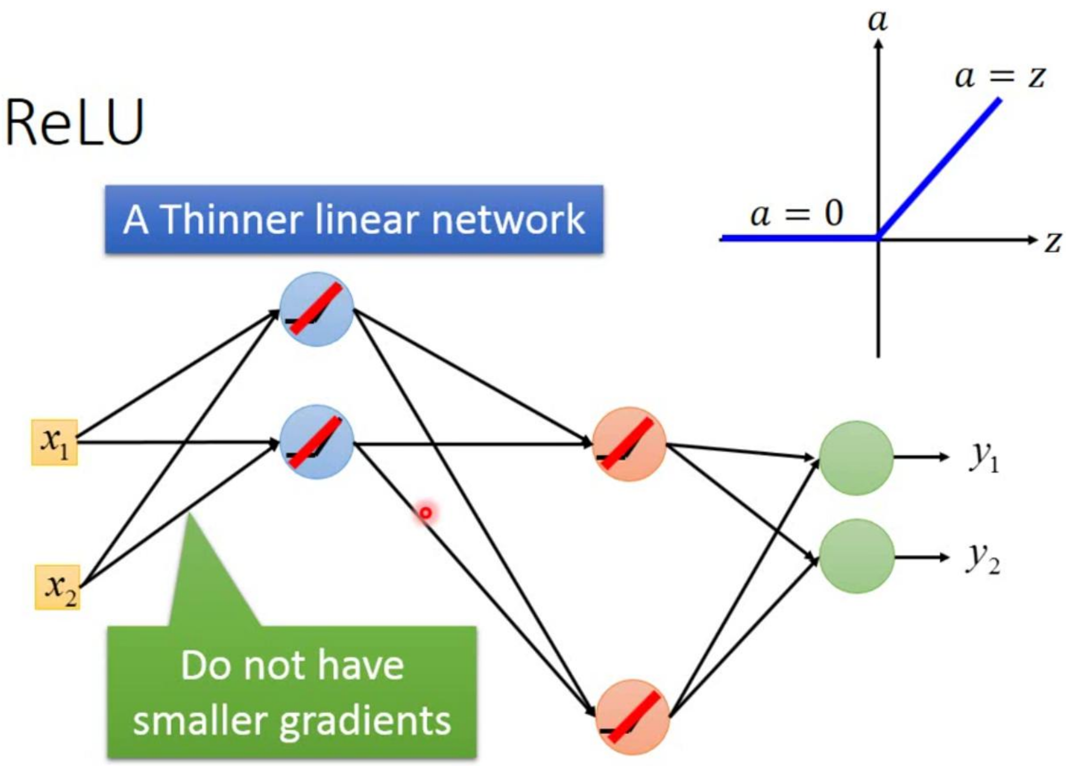
因此，再次强调 过拟合指的是在 training data 上的效果好，但是在 testing data 上的效果不好，才叫过拟合；只是 testing data 上的效果不好，不一定就是过拟合，还要看在 training data 上的效果。



3*



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



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

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

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

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

$$\vdots$$

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

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

Movement: movement of last step minus gradient at present

Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

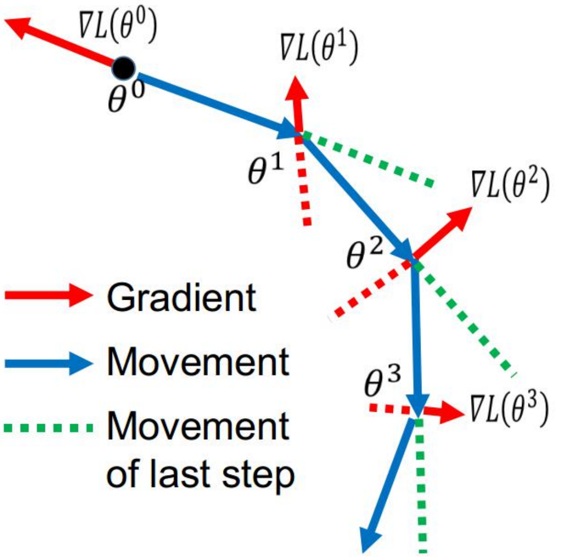
Movement $v^1 = \lambda v^0 - \eta \nabla L(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

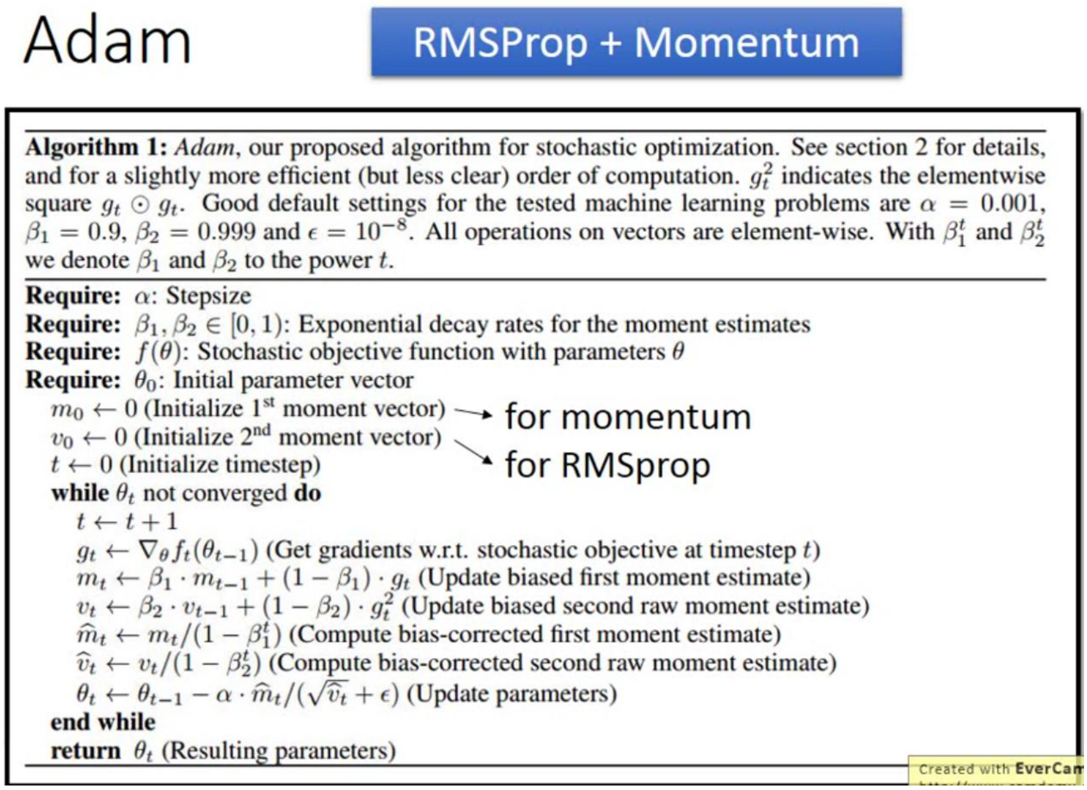
Movement $v^2 = \lambda v^1 - \eta \nabla L(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$



Movement not just based on gradient, but previous movement.

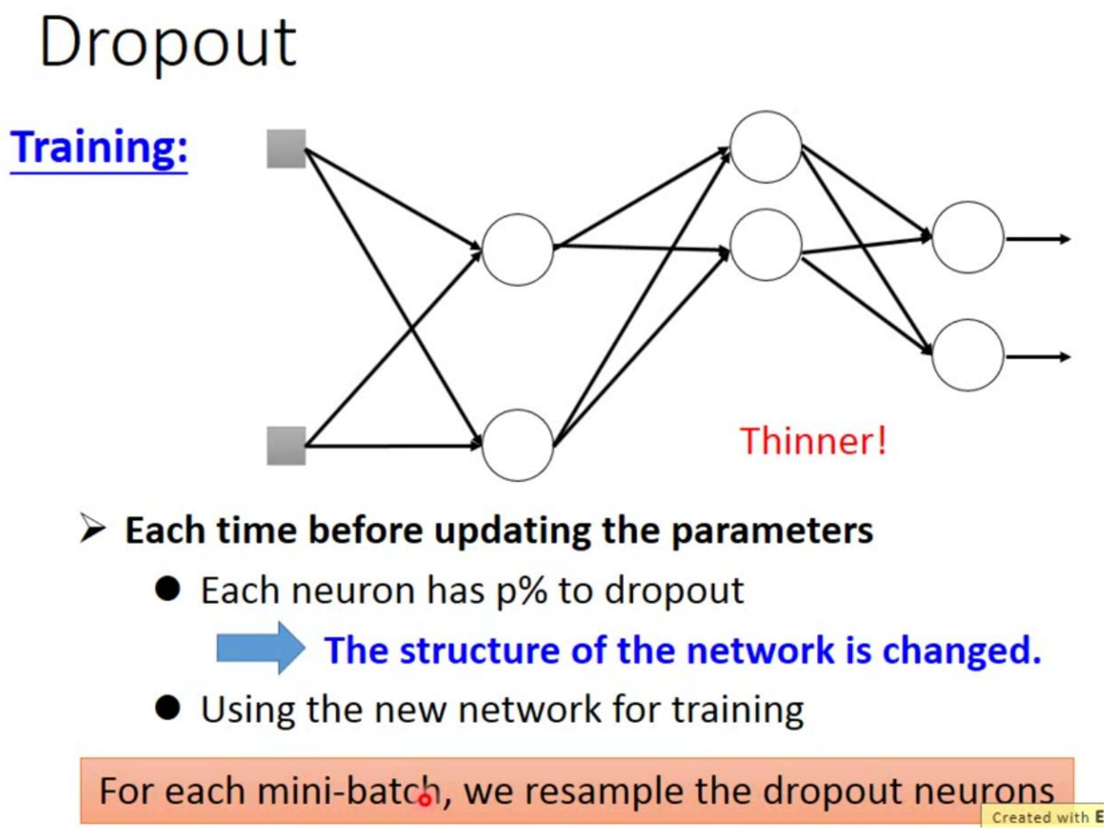
Adam 是综合考虑了 RMSProp 和 Momentum 的 optimizer 方法



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 β_1^t and β_2^t we denote β_1 and β_2 to the power t.

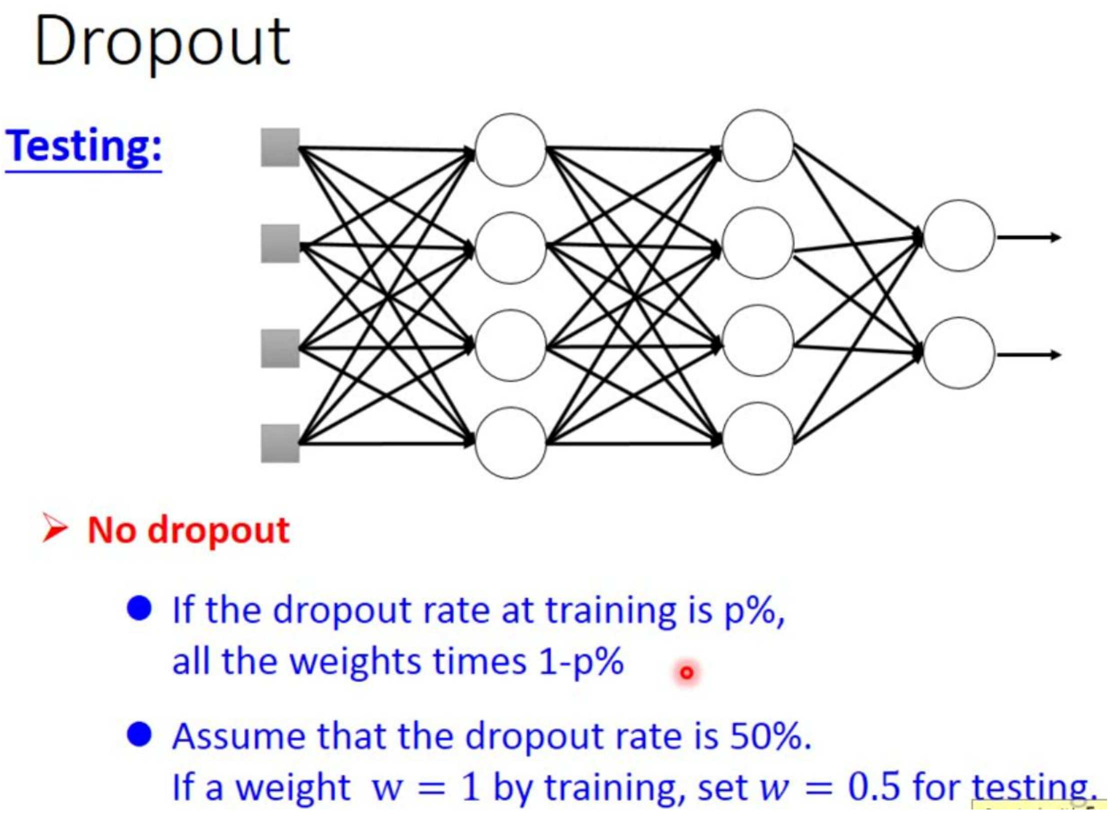
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_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)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

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

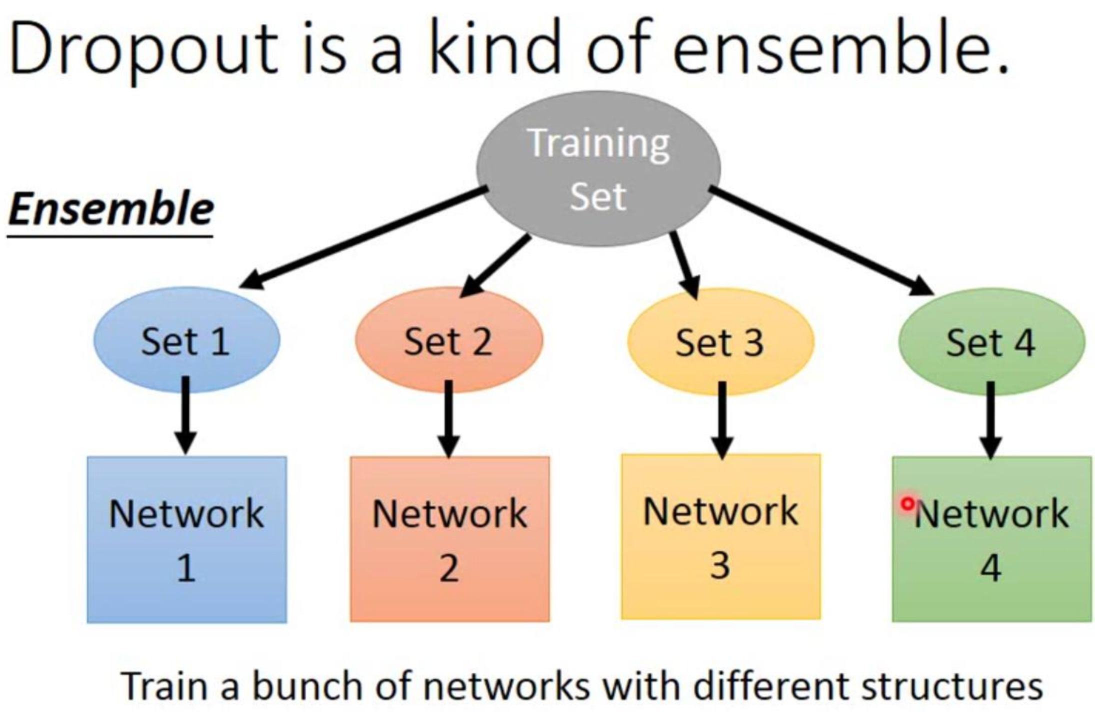


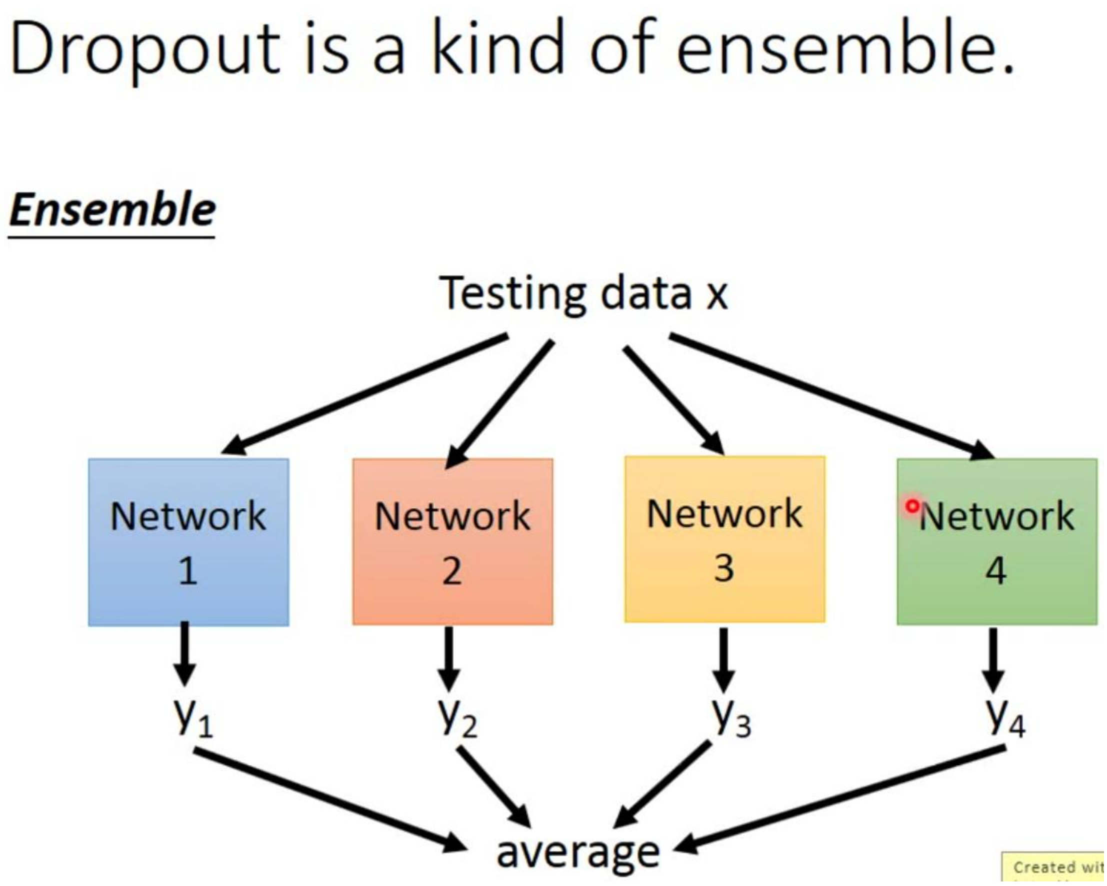
7* Dropout 是在 training 的过程中才会有，在 testing 的过程中不进行 Dropout;

在 testing 时，所有的权重要乘上(1-p%)



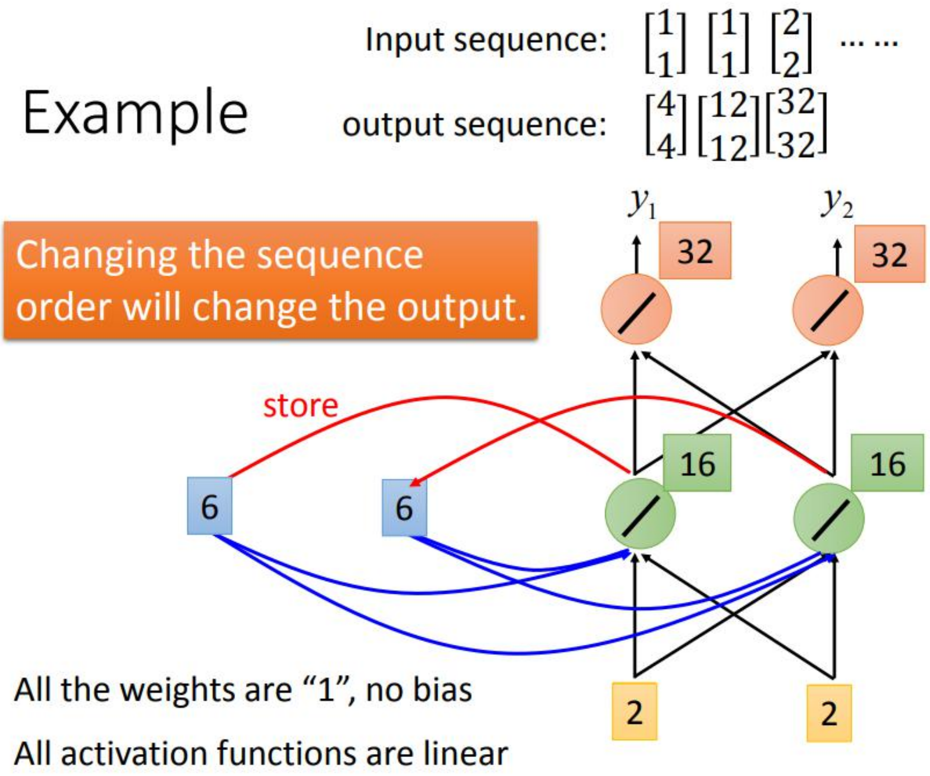
8* Ensemble



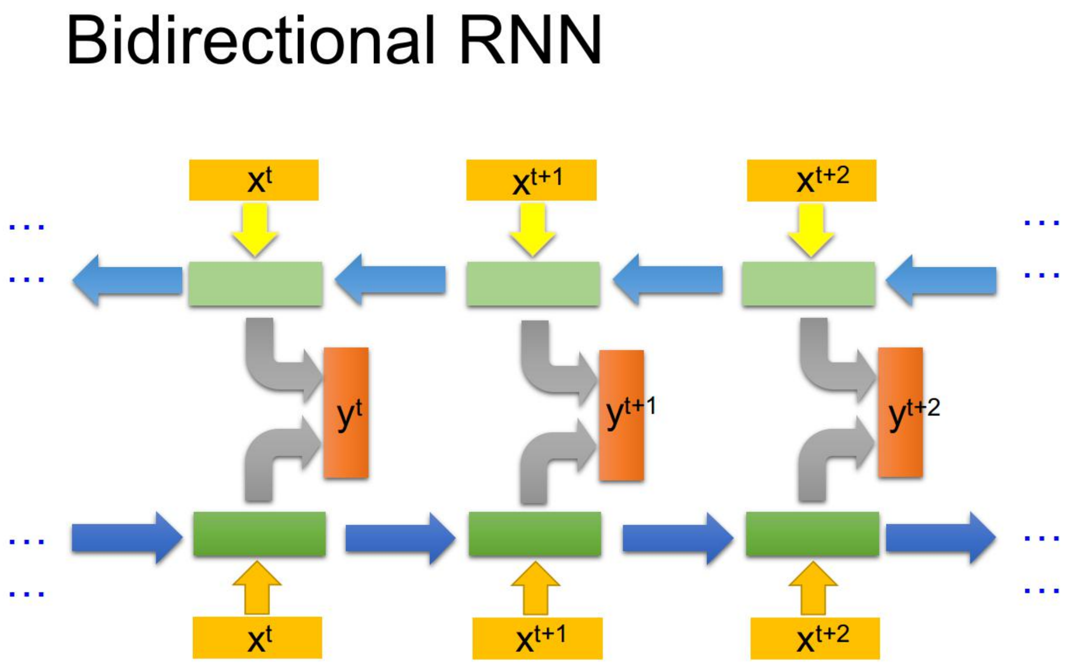


19_RNN1.mp4

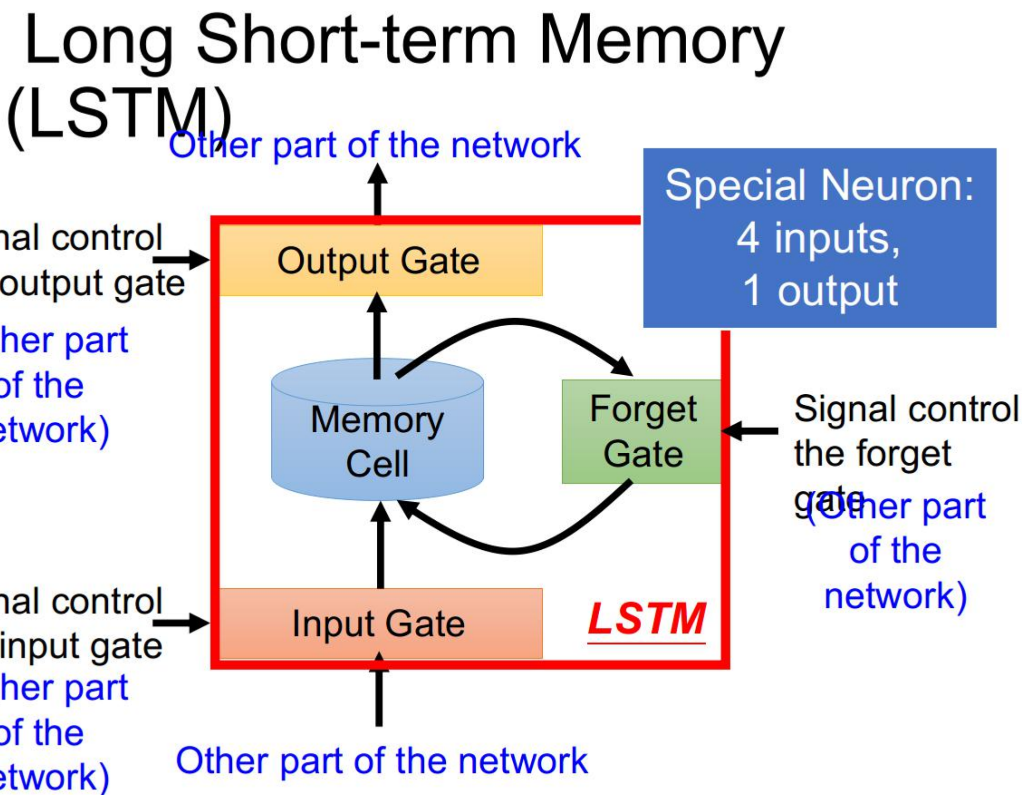
1* 对于 RNN 来说，input 的顺序会影响到输出的结果，也就是说 RNN 会考虑 input 的顺序



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）。



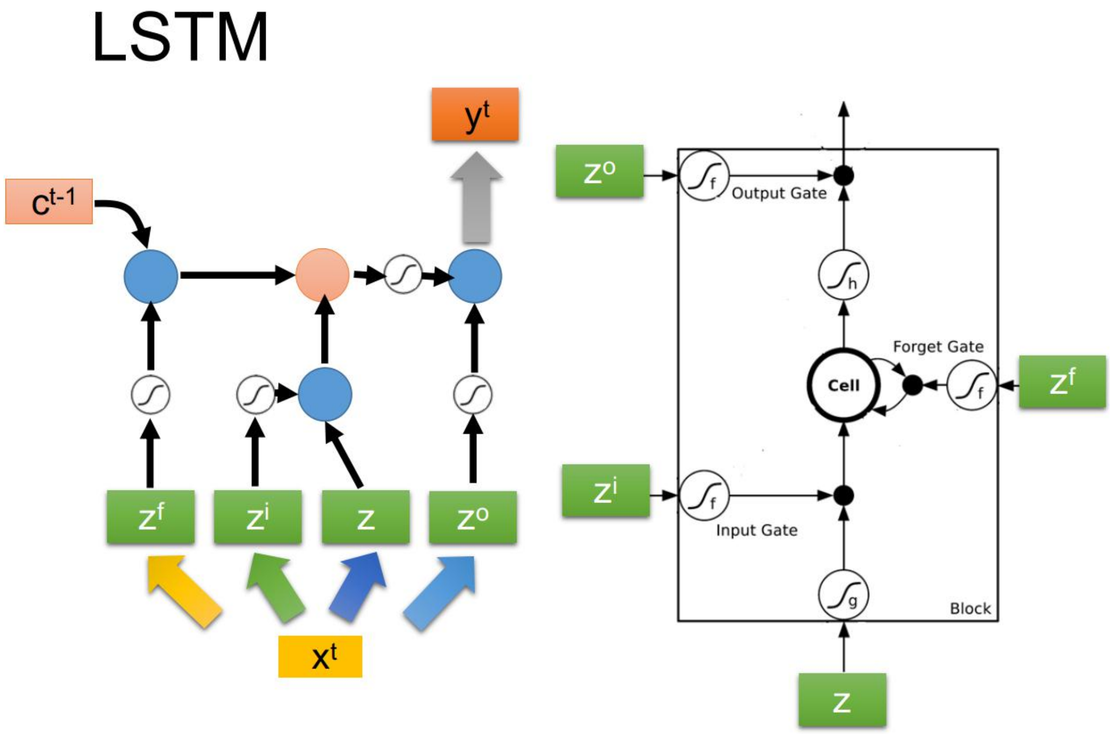
3* LSTM 的神经元是一种特殊的神经元，普通的神经元有一个输入一个输出，但 LSTM 神经元包括四个输入一个输出，四个输入分别是：输入数据、Input Gate 控制信息、Output Gate 控制信息、Forget Gate 控制信息

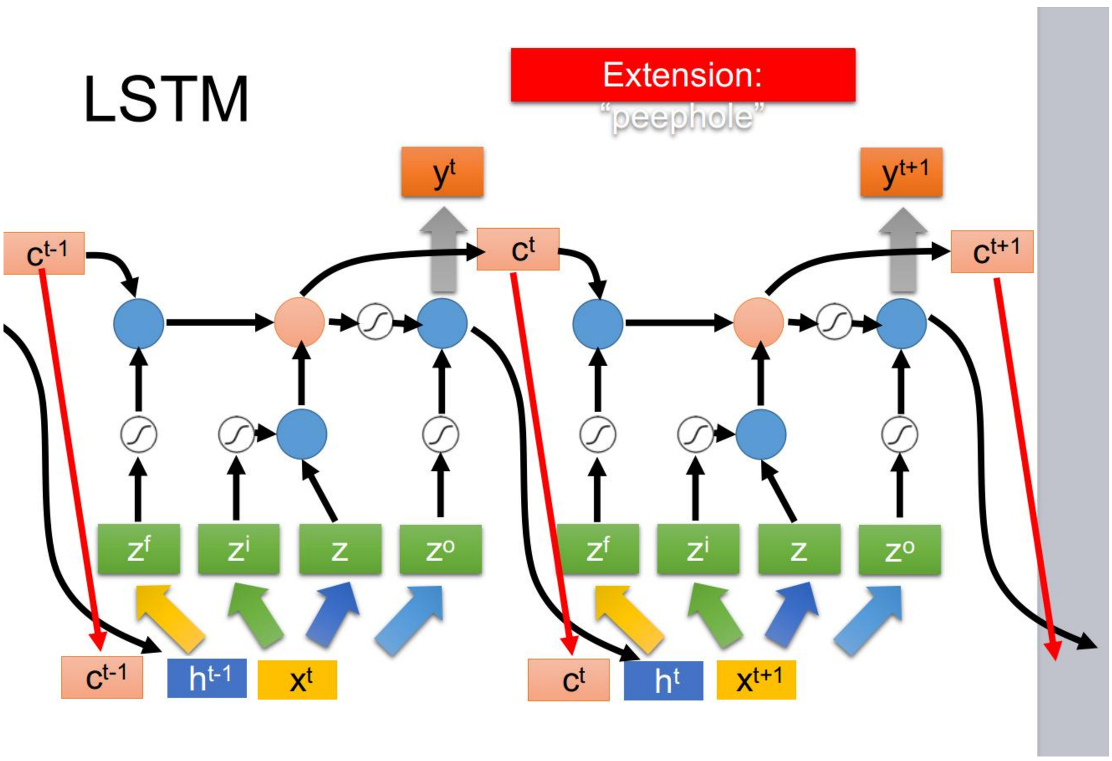


4* LSTM：Long Short-term Memory 连词符号要放在 Short 和 term 之间，LSTM 表示的是比较长（long）的 Short-term 记忆网络

5* LSTM 的三个门的控制信息也是与 input 直接相关的

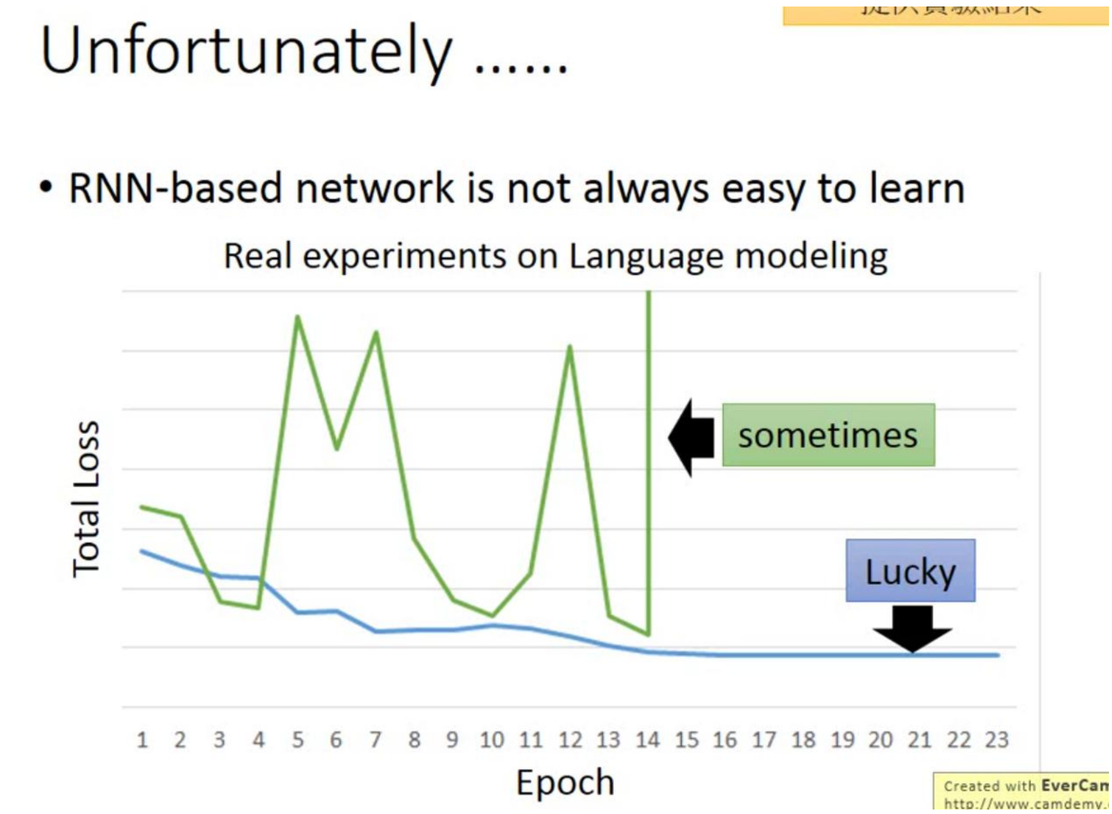
6* LSTM 的记忆信息的传递（与 RNN 本质上是类似的）

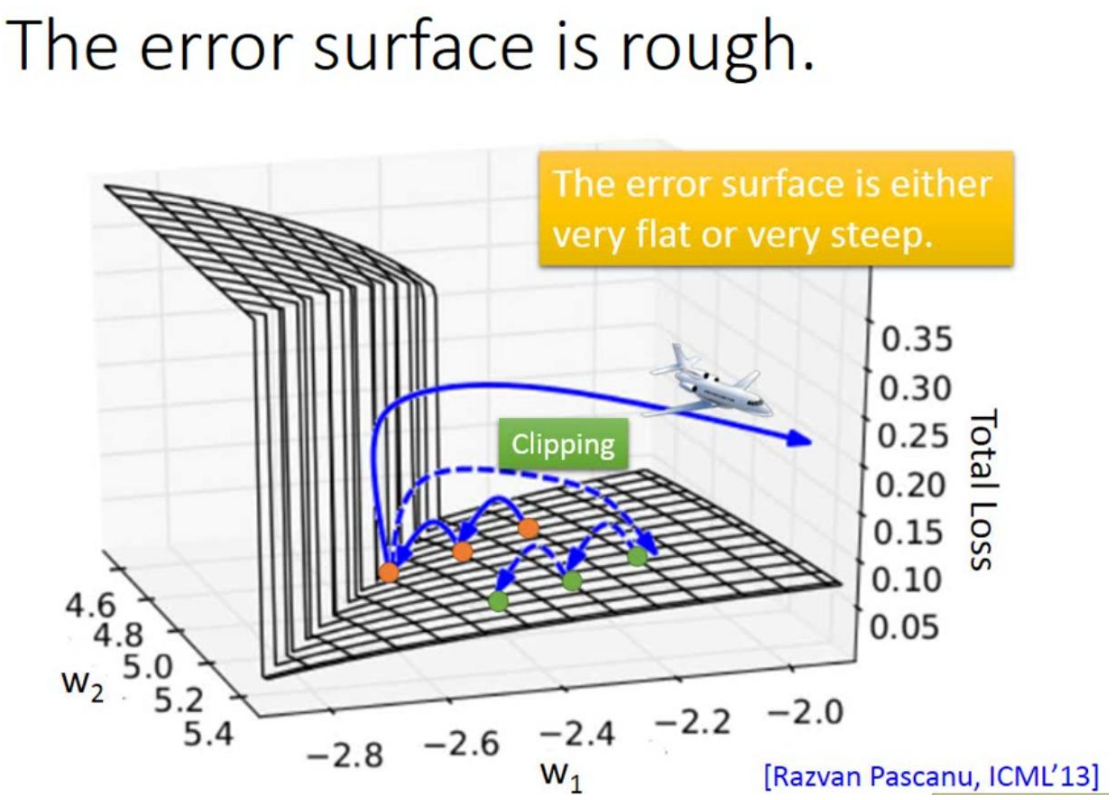




19_RNN2.mp4

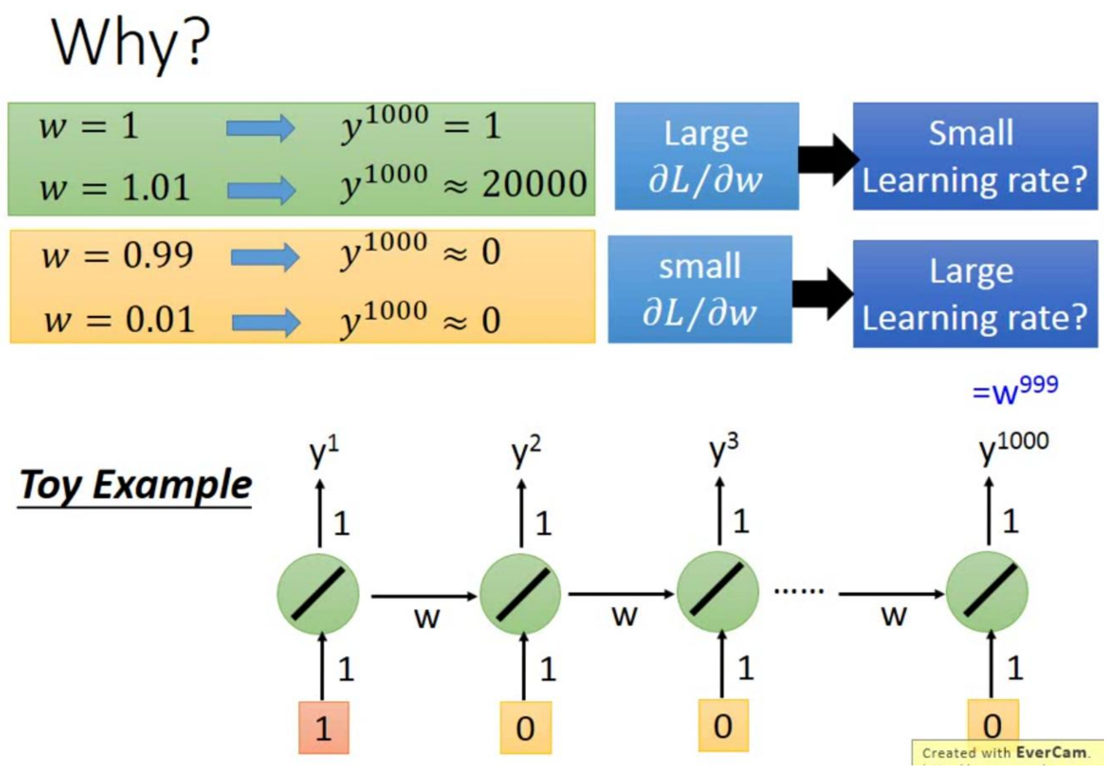
1* RNN 并不总是容易训练的





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



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

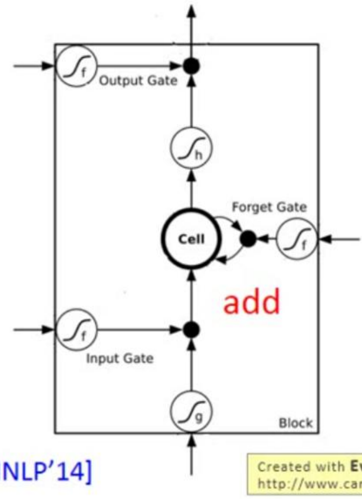
Helpful Techniques

- 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]