

本文的主要思路：针对training set和testing set上的performance分别提出针对性的解决方法
1、在training set上准确率不高： new activation function: ReLU、Maxout adaptive learning rate: Adagrad、RMSProp、Momentum、Adam 2、在testing set上准确率不高：Early Stopping、Regularization or Dropout

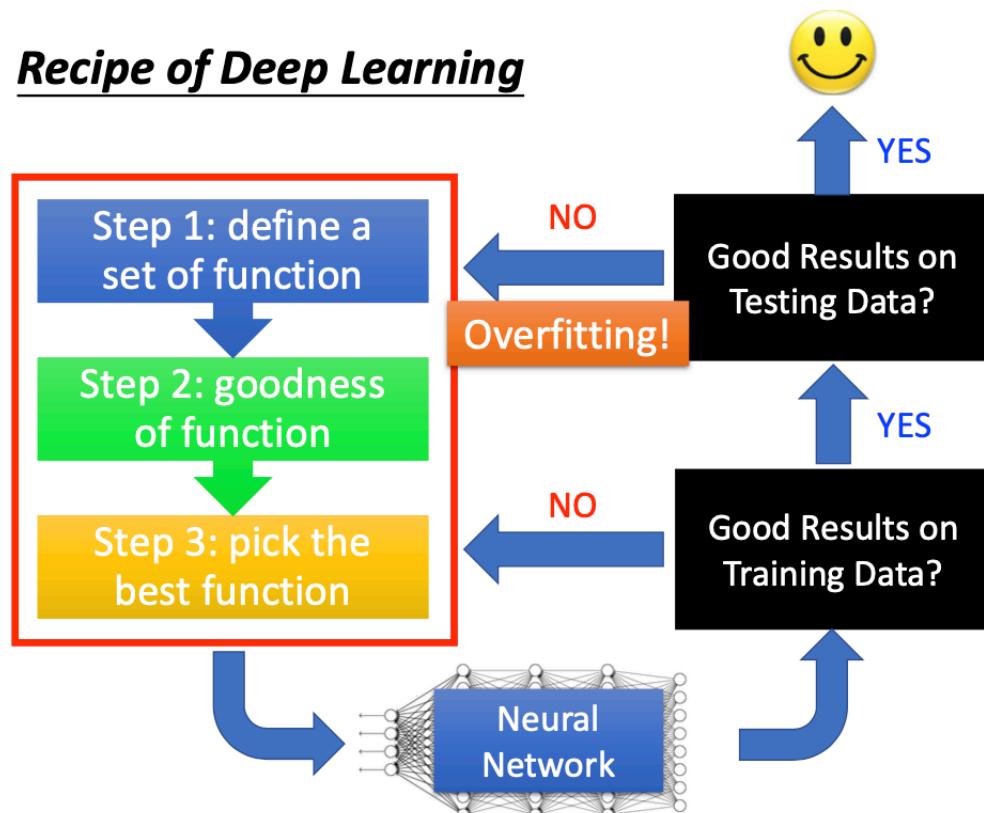
Recipe

three steps of deep learning

做深度学习也遵循这三个步骤：

- define a set of function,
- goodness of function, 找到loss function
- pick the best function, 找到使loss最小化的参数

overfitting是指模型在训练集上表现良好，但在测试集上表现却很差的现象。

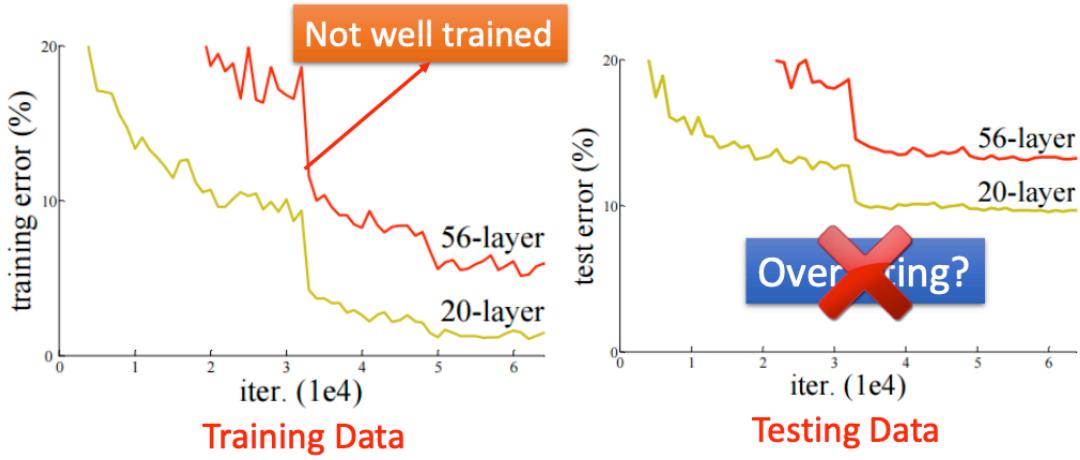


Do not always blame Overfitting

在下图中，我们展示了一个20层和56层的network，在训练集和测试集上的error。黄色表示20层network，红色表示56层network。

由于模型在训练集上的表现，20层的network表现得比较好，有同学就认为这是overfitting，但其实这并不是overfitting问题，因为这个模型在训练集上的表现，也是20层的network表现好

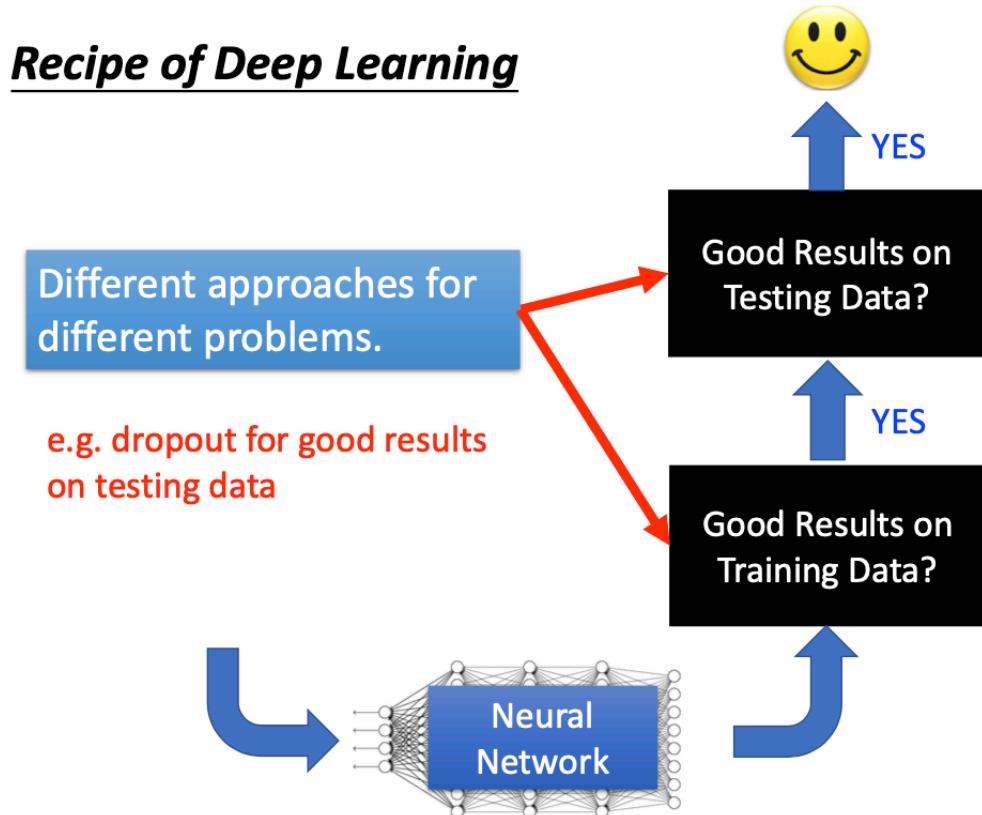
之所以出现这个20层和56层network表现（都是20层network表现好），是由于模型的训练没有训练好



Different approaches for different problems.

在网络的训练过程中，我们要针对网络的不同问题提供不同的解决方法，主要有两个问题

- 在training data上表现不好
- 在testing data上表现不好



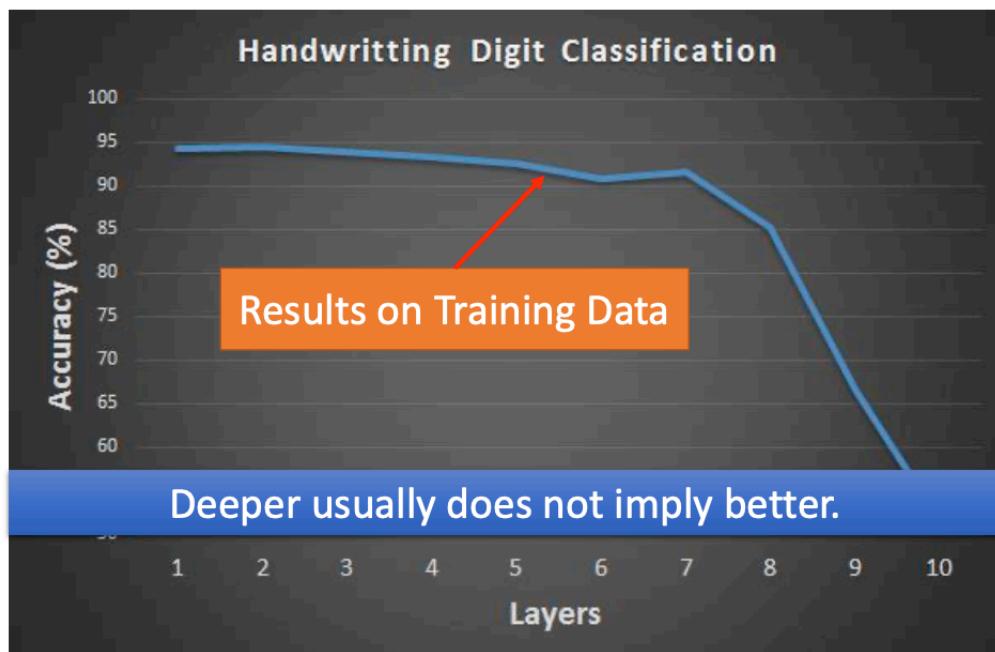
Good Results on Training Data?

要在training data上获得好的结果，可以使用new activation function和adaptive learning rate

New Activation Function

deeper is better ?

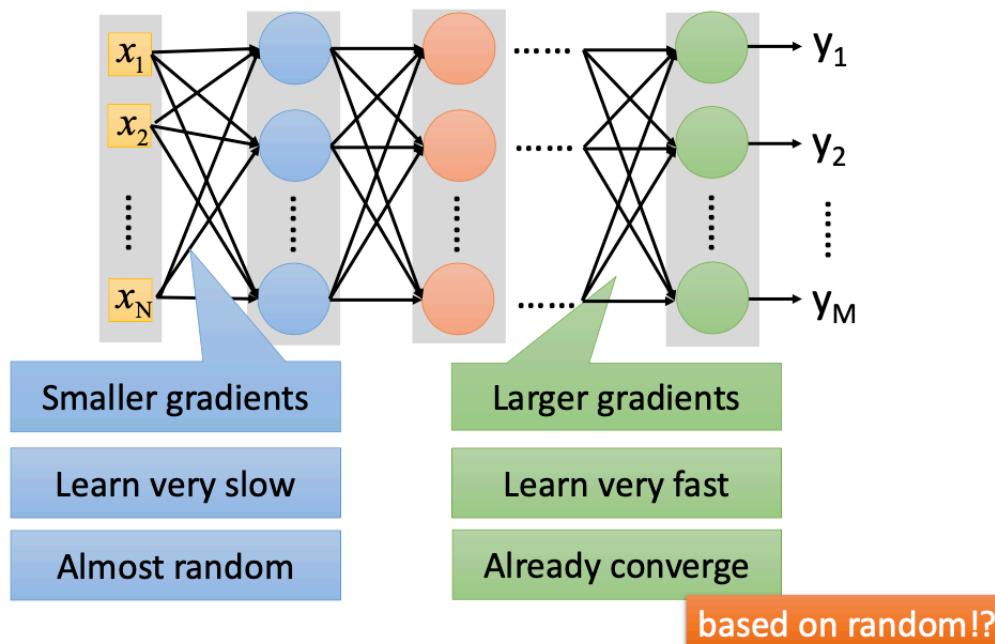
在1980年代，network中主要使用sigmoid function作为激活函数，从下图中我们可以看出，使用sigmoid function并不能保证网络结构越深，训练结果越好



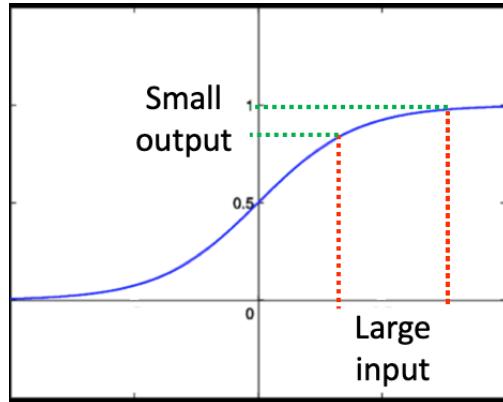
Vanishing Gradient Problem

出现上面这个问题的原因并不是overfitting, 而是vanishing gradient (梯度消失)

当网络层数很深的时候, 在靠近input layer的位置, 常常会有很小的gradient, 学习速度也很慢; 而在靠近output layer的地方, 常常会有更大的gradient, 学习速度也会很快, 很快就到了converge (收敛) 了



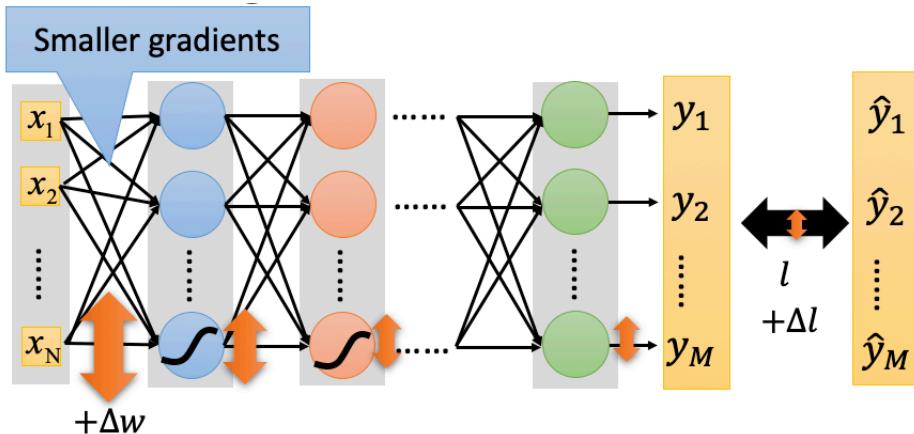
下面将叙述出现这个问题的原因。对于下图的中sigmoid function, 输出范围为[0,1], 对于很大的输入, 输出往往会被压缩成一个较小的值



对于loss function对其中一个参数w的微分 $\frac{\partial l}{\partial w}$, 这里我们将其表达式写为 Δw , 可以表示当前参数w对结果loss的影响, 当把这个参数w进行变化时, 对loss的影响会有多大

如下图所示, 如果我们输入一个很大的 Δw , 在经过sigmoid function运算之后, 其值就缩小一次; 当经过后面多层的压缩之后, Δw 的值就变得越来越小;.....; 因此gradient在input layer附近的值会很大, 但在output layer附近的值经过多次的压缩就变得很小了。

当缩小的gradient达到我们设置的那个临界值, 即gradient接近于0, 就发生了梯度消失问题



$$\frac{\partial l}{\partial w} = ? \frac{\Delta l}{\Delta w}$$

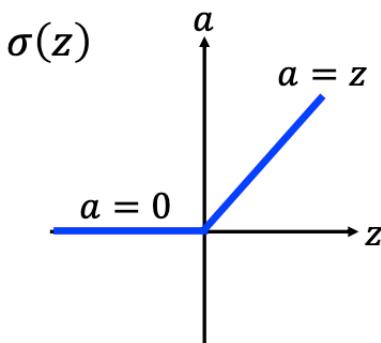
要解决这个问题, 可以修改一下network中用到的激活函数

ReLU

我们选取ReLU函数的原因有以下几个:

- 可以快速计算, 不管是函数值还是对应的梯度
- 结合了生物上的一些观察
- 无穷多个不同bias的sigmoid function叠加的结果可以变成ReLU
- 可以解决梯度消失问题;

- Rectified Linear Unit (ReLU)



[Xavier Glorot, AISTATS'11]
 [Andrew L. Maas, ICML'13]
[\[Kaiming He, arXiv'15\]](#)

Reason:

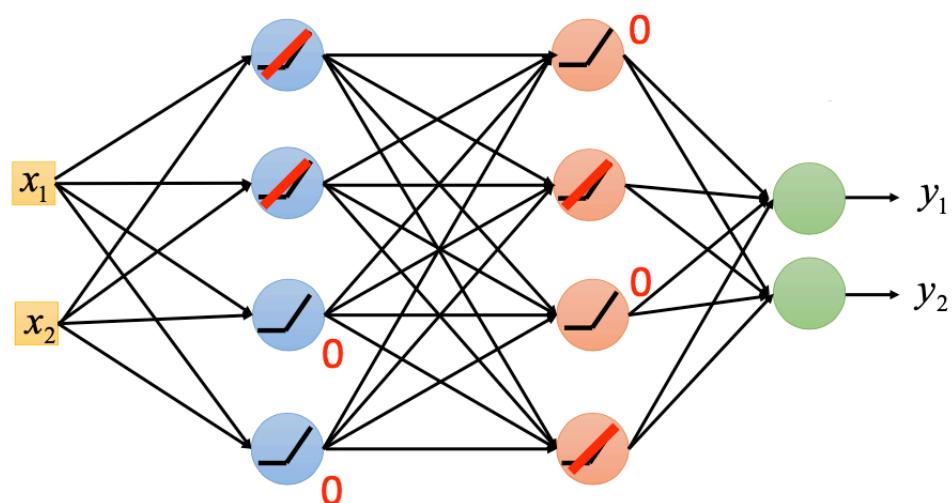
1. Fast to compute

2. Biological reason

3. Infinite sigmoid
with different biases

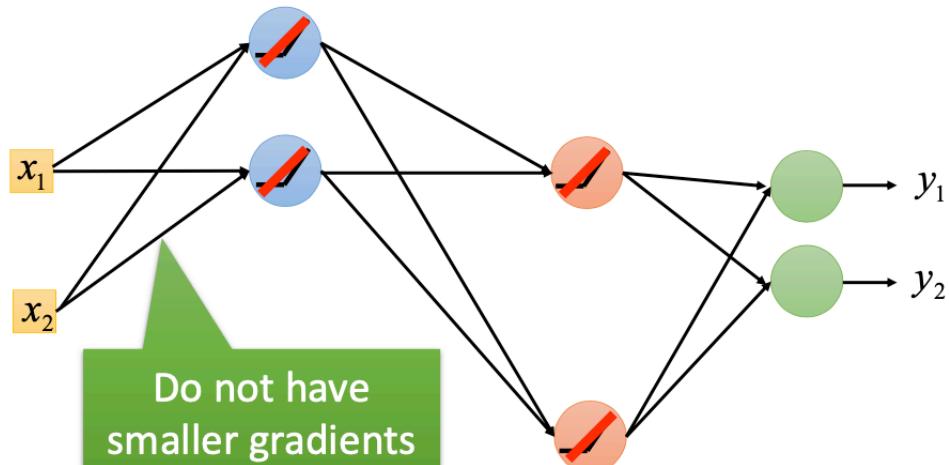
4. Vanishing gradient
problem

使用ReLU函数之后，代入具体的网络结构



对于input为0的值，network将不再计算其相对应的weight，而对于input不为0的值，就相当于一个线性函数 $y = x$ 。这样做可以简化网络结构，网络结构中也就不存在gradient很小的neural

A Thinner linear network



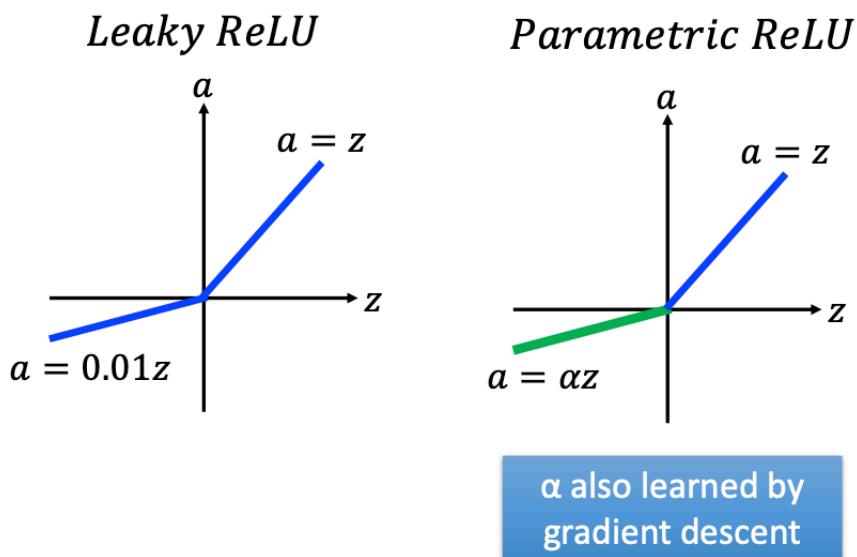
Q: 但这时出现了一个新问题, ReLU函数在 $z=0$ 这一点是不可导的, 那么我们在根据loss function如何来计算gradient呢?

A: 这里我们将输入 $z<0$ 的数的gradient看作0, 相当于从network中抹去了这部分神经元; 对于 $z>=0$ 的数, gradient=1

ReLU - variant

对于ReLU, 当 $x<=0$ 时, 函数的输出值就为0了, 网络中的参数也没办法更新。因此, 就有学者提出了Leaky ReLU, 当 $x<=0$ 时, 函数的输出值不是0, 而是乘以一个系数0.01, 这时的函数就称作**Leaky ReLU**

还有另外一种ReLU函数的变体, **Parametric ReLU**, 前面乘上的系数也可以进行训练



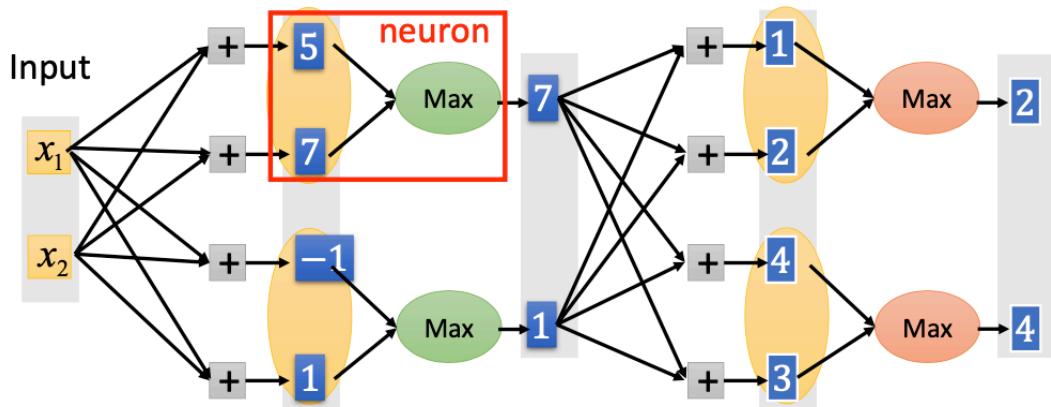
Maxout

ReLU is a special cases of Maxout

Maxout的主要思想是: 让network自己去学习对应的activation function, 可以学习出ReLU, 也可以是其他的activation function

Maxout激活函数是对前几个神经元取最大值, 再输出相应的最大值

- Learnable activation function [Ian J. Goodfellow, ICML'13]

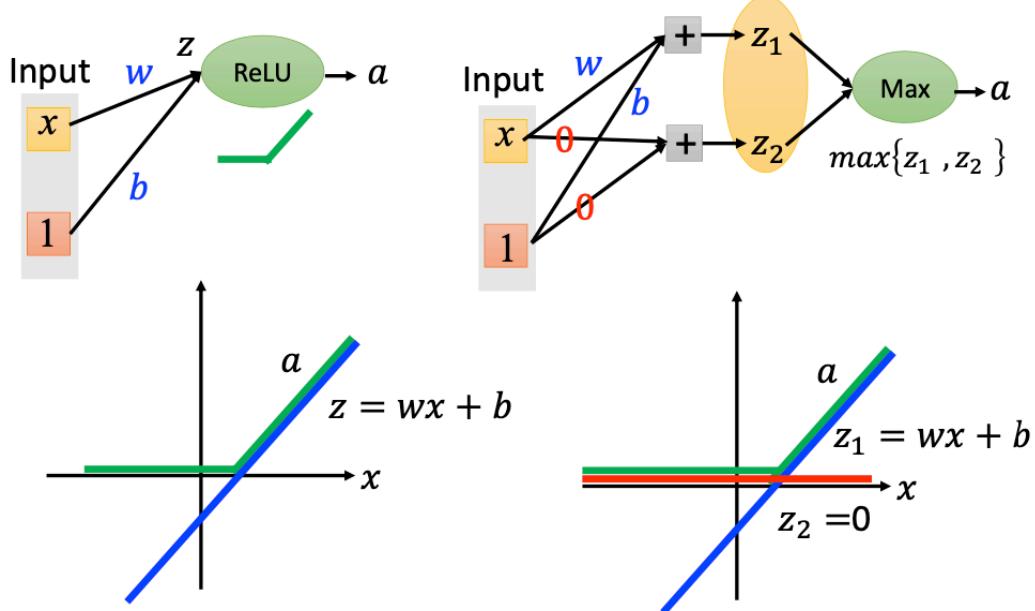


You can have more than 2 elements in a group.

Maxout->ReLU

在下图中，对于左图中的ReLU function

- input为蓝色直线，表示 $z = wx + b$,
- output: 当 $z < 0$ 时，ReLU也输出为0；当 $z > 0$ 时，ReLU的图像和 z 是一致的



对于右图中的Maxout function, z_1 对应的权重是 w, b , 而 z_2 对应的权重则是0,

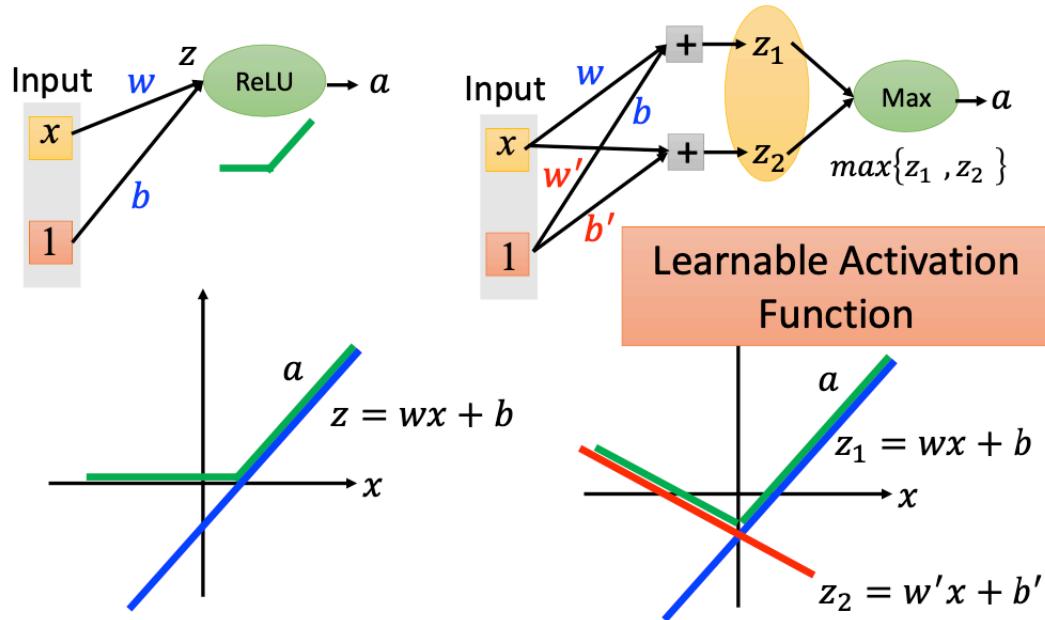
- input为 z_1, z_2 ,
 - 蓝色直线表示 $z_1 = wx + b$,
 - 红色表示 $z_2 = 0$
- 这时neural的output为 $\max\{z_1, z_2\}$, 输出则是和ReLU一致的 (图中绿色直线)

Maxout->more than ReLU

maxout不仅可以学习ReLU，也可以学习其他的activation function

对于右图中的新的输入， z_2 对应的权重则变成了 w' , b' ，那么相应的input和output为

- input, $z_1 = wx + b, z_2 = w'x + b'$, 分别对应图中蓝色和绿色直线；
- output为 $\max\{z_1, z_2\}$, 表现为图中绿色直线

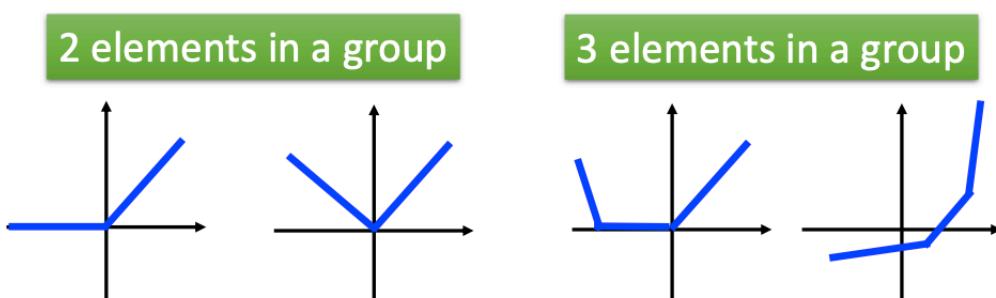


这时我们得到的activation function就是图中的绿色直线，是通过网络training出来的，训练的参数为 w, w', b, b' ，训练结束即可得出我们的activation function

Summary

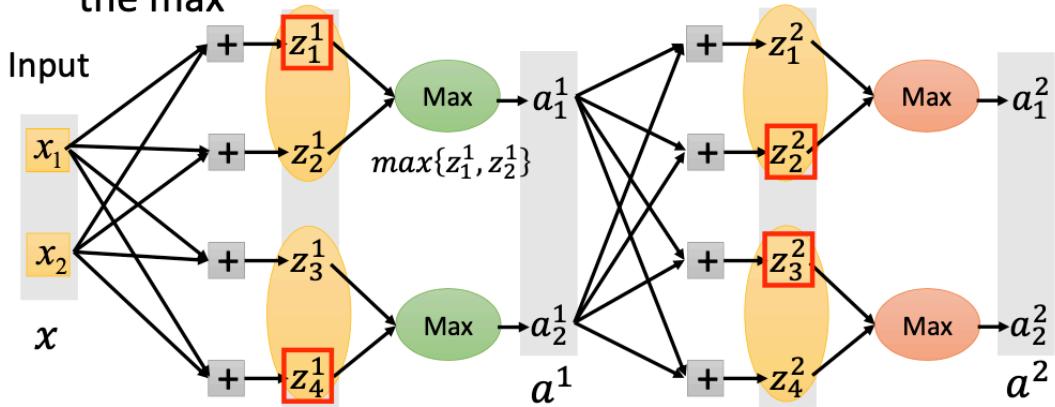
这里我们先对maxout做一个总结，maxout是一个可学习的activation function，可以学习出任何分段线性凸函数（piecewise linear convex function），具体的分段数取决于在group中的元素个数

- Learnable activation function [Ian J. Goodfellow, ICML'13]
 - Activation function in maxout network can be any piecewise linear convex function
 - How many pieces depending on how many elements in a group

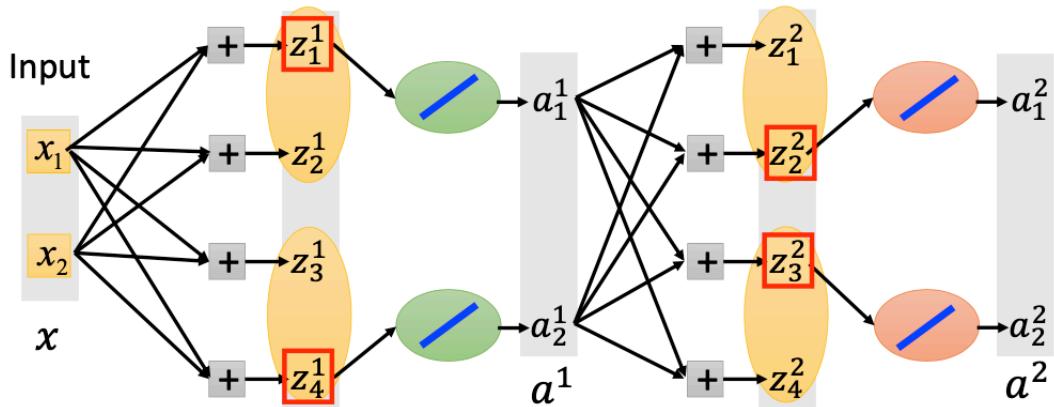


对于maxout函数的训练，如果是下图中的网络结构，我们假设已经知道 $z_1^1, z_4^1, z_2^2, z_3^2$ 为对应的最大值，

- Given a training data x , we know which z would be the max



那么network可以再次被化简, neural也可以变少



- Train this thin and linear network

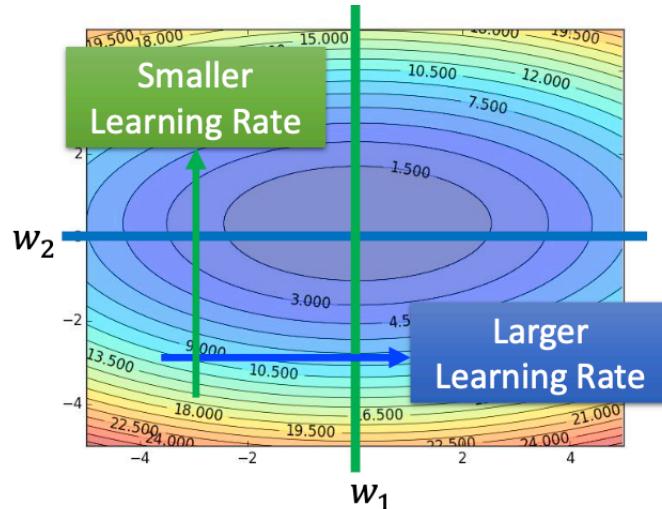
Different thin and linear network for different examples

Adaptive Learning Rate

Review

Adagrad

Review



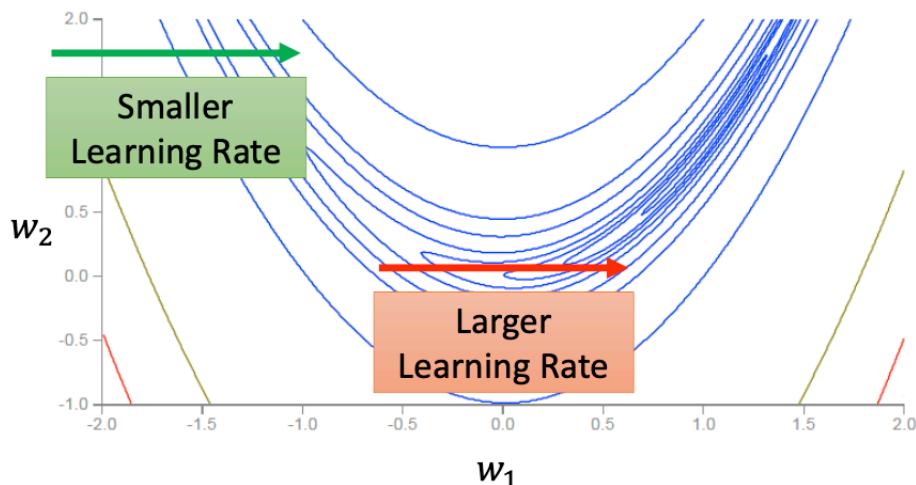
Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}} g^t$$

Use first derivative to estimate second derivative

RMSprop

Error Surface can be very complex when training NN.

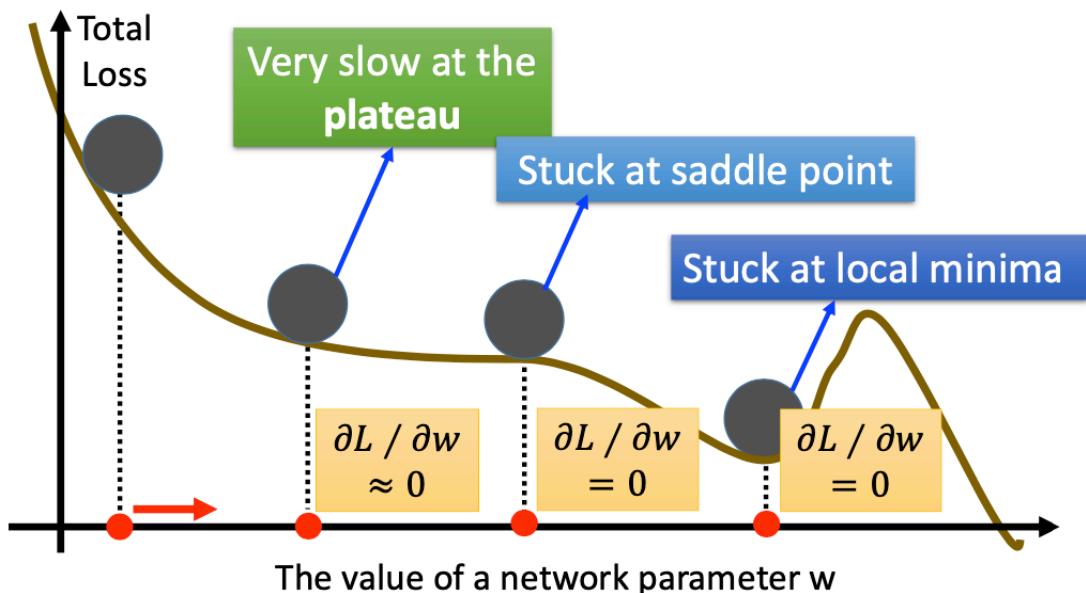


$$\begin{aligned}
 w^1 &\leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 & \sigma^0 &= g^0 \\
 w^2 &\leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 & \sigma^1 &= \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2} \\
 w^3 &\leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 & \sigma^2 &= \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2} \\
 &\vdots && \\
 w^{t+1} &\leftarrow w^t - \frac{\eta}{\sigma^t} g^t & \sigma^t &= \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}
 \end{aligned}$$

Root Mean Square of the gradients
with previous gradients being decayed

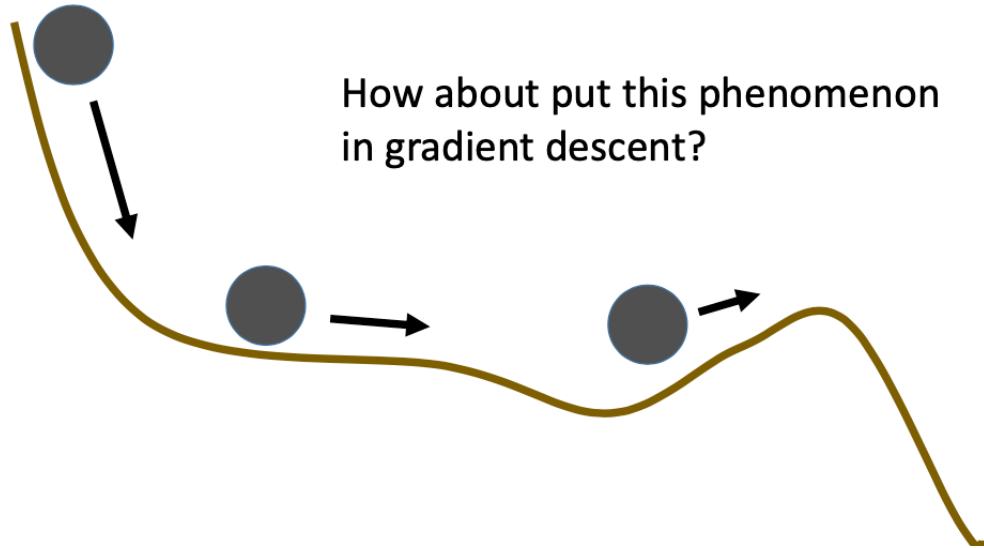
gradient为0的点可以是local minimum, 也可以是saddle point, 还可以是在很平缓的plateau中的某些点

Hard to find optimal network parameters



而在物理世界，物体本身是带有momentum的，再加上gradient的作用，就很可能可以跳出saddle point，继续训练

• Momentum



Momentum

图中蓝色箭头表示Movement (前进方向), 红色箭头表示gradient的方向, 绿色虚线表示上一次movement对本次的影响 (惯性)

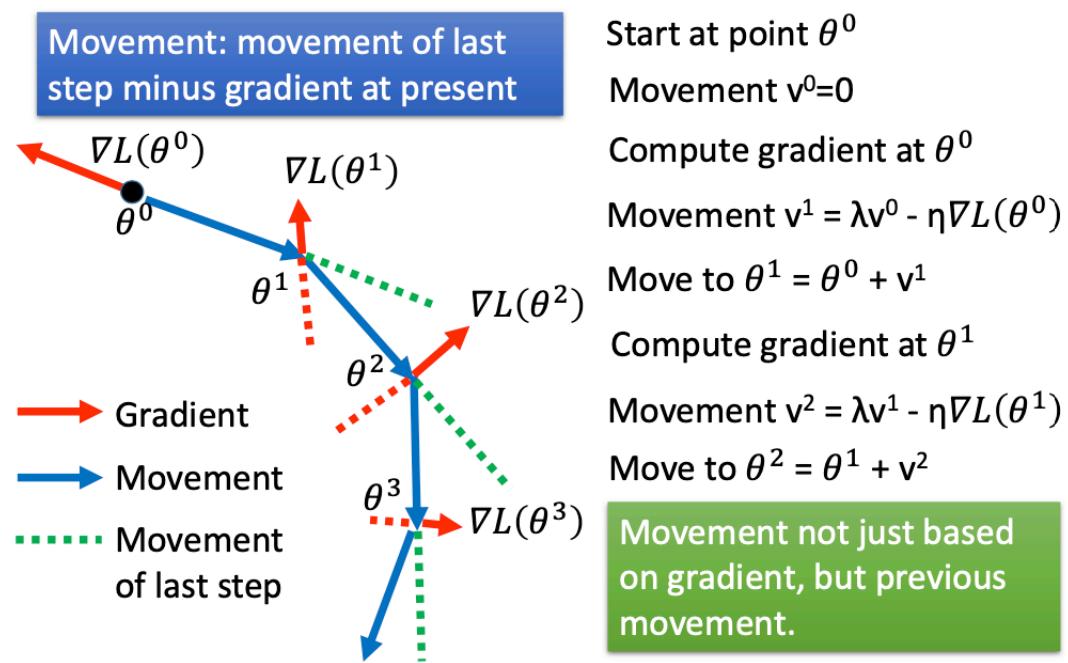
再 θ^0 处, movement为 $v^0 = 0$; 对于在 θ^1 处的前进方向, 先计算出在 θ^0 处的梯度 $\nabla L(\theta^0)$, 我们要移动的方向是由上一个时间点的gradient为 $\nabla L(\theta^0)$ 和前进方向 v_0 决定的, 即

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

其中 λ 也是一个可以手动调整的参数

对于下一个时间点的移动方向 v^2 , 是和当前时间节点的移动方向和梯度 $v^1, \nabla L(\theta^1)$ 决定的, 即

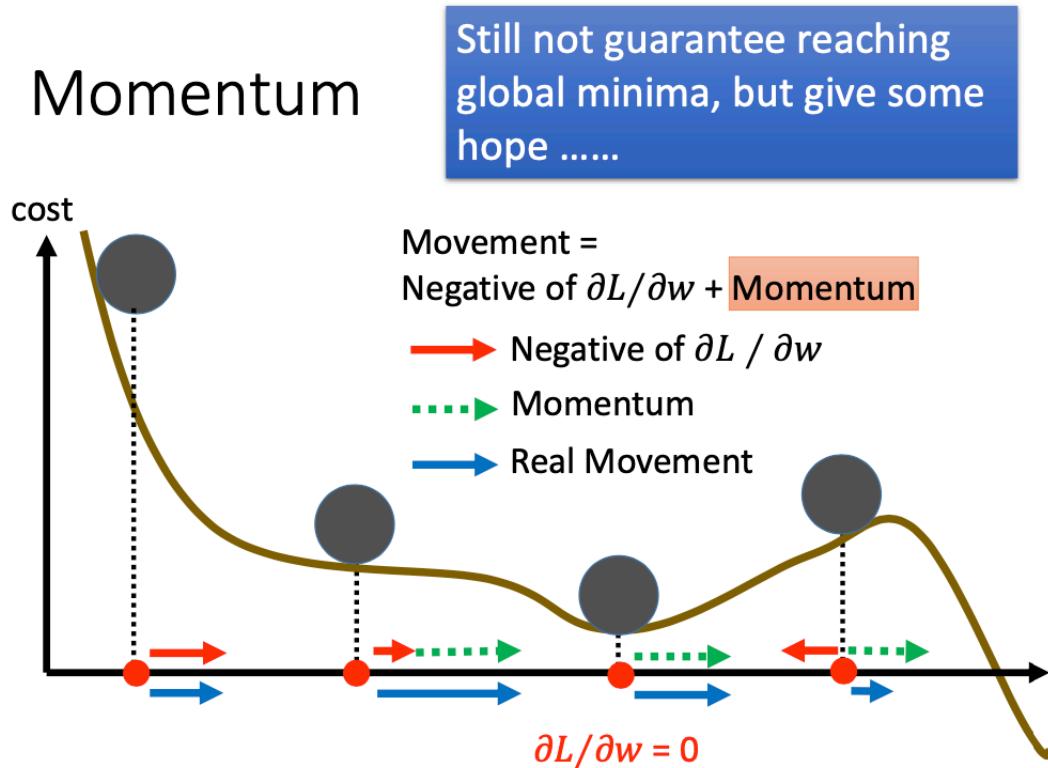
$$\begin{aligned} v^2 &= \lambda v^1 - \eta \nabla L(\theta^1) \\ &= -\lambda \eta \nabla L(\theta^0) - \eta \nabla L(\theta^1) \end{aligned}$$



再回到之前的例子，红色箭头表示gradient的反方向，绿色表示momentum的方向，蓝色箭头表示受到gradient和momentum影响后的真实运动方向

初始点的momentum值为0；在下一个plateau上的点，虽然gradient的值很小很小，但由于受到上一个很大的momentum的影响，真实的movement还是向前的，步长也没有因为gradient的变小而变得很小；

如果我们现在走到了local minimum，此时gradient=0，此时由于momentum的影响，如果momentum的值足够大，还会继续向前运动



Adam

RMSProp + Momentum

Adam其实就是结合了RMSProp和Momentum思想的方法

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:  $\alpha$ : 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 1st moment vector) → for momentum
 $v_0 \leftarrow 0$  (Initialize 2nd moment vector) → 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)

```

先将动量momentum和初始移动方向初始化为0, 即 $m_0 = 0, v_0 = 0$, v_0 表示RMSProp中分母上的参数 σ

计算在t时的梯度 g_t ,

$$g_t = \Delta_{\theta} f_t(\theta_{t-1})$$

根据上一个时间点的要走的方向 m_t 和gradient, 因此t时的移动方向为 m_t -----Momentum

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

根据上一个时间点的移动方向 v_{t-1} 和gradient, 则此时的真实移动方向 v_t 为-----RMSprop

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t$$

该算法还进行了bias corrected,

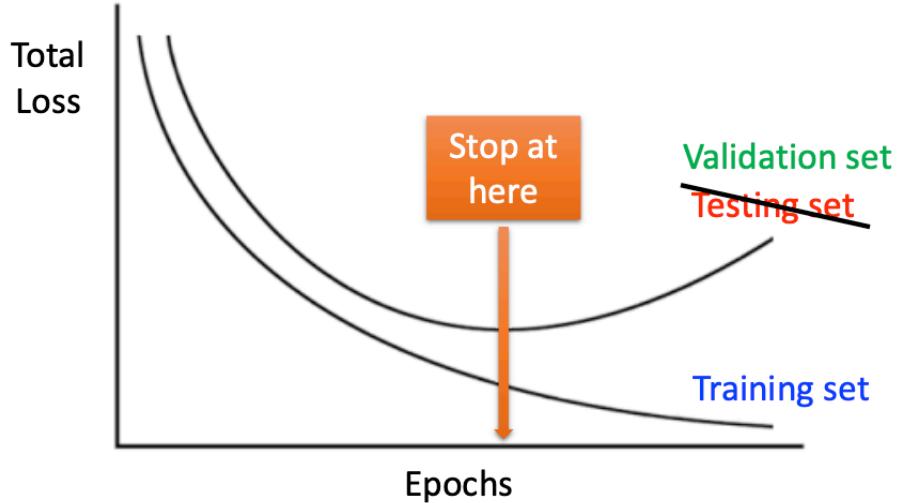
$$\hat{m}_t = \frac{m_t}{(1 - \beta_1^t)}, \quad \hat{v}_t = \frac{v_t}{(1 - \beta_2^t)}$$

将进行了bias corrected的参数再输入公式, 更新参数

$$\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t} + \epsilon$$

Good Results on Testing Data?

Early Stopping



Regularization

L2 Regularization

正则化就引入了一个新的loss function，加上了一个新的正则项，这个正则项将所有需要training的参数都包括进来了，通常不包括bias

- New loss function to be minimized
 - Find a set of weight not only minimizing original cost but also close to zero

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2 \rightarrow \text{Regularization term}$$

$$\theta = \{w_1, w_2, \dots\}$$

Original loss

(e.g. minimize square error, cross entropy ...)

L2 regularization:

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

(usually not consider biases)

这个新的loss function再对w求偏微分，对参数进行更新

$$w^{t+1} = (1 - \eta\lambda)w^t - \eta \frac{\partial L}{\partial w}$$

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2^2 \quad \text{Gradient: } \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda w$$

Update: $w^{t+1} \rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta \left(\frac{\partial L}{\partial w} + \lambda w^t \right)$

 $= \underbrace{(1 - \eta \lambda)w^t}_{\downarrow \text{Closer to zero}} - \eta \underbrace{\frac{\partial L}{\partial w}}_{\text{Weight Decay}}$

与原来的参数更新公式相比，可以发现 w^t 前面多了一项 $(1 - \eta \lambda)$ ，通常这个 η, λ 都是很小的值，这里我们假设 $(1 - \eta \lambda)$ 是很接近于1的值，约等于0.99；regularization所做的事就是，在每次更新参数时，全都在前面乘上了一个小于1的数，在经过若干次的训练之后， $(1 - \eta \lambda)w^t$ 的值就很接近0了。

虽然 $(1 - \eta \lambda)w^t$ 的值每次都会变得越来越小，但参数更新的公式中，后面还有另外一项 $\eta \frac{\partial L}{\partial w}$ ，会使得梯度的值不会变成0，达到平衡。

L1 Regularization

既然L2可以作为正则项，L1也可以作为正则项，正则项为参数的绝对值相加。对这些参数求导，当 w_i 大于0时，gradient=1，当 w_i 小于0时，gradient=-1，即为 $\text{sgn}(w)$ 函数。

L1的参数更新公式为

$$w^{t+1} = w^t - \eta \frac{\partial L}{\partial w} - \eta \lambda \text{sgn}(w^t)$$

L1 regularization:

Regularization $\|\theta\|_1 = |w_1| + |w_2| + \dots$

- New loss function to be minimized

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_1 \quad \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda \text{sgn}(w)$$

Update:

$$\begin{aligned} w^{t+1} &\rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta \left(\frac{\partial L}{\partial w} + \lambda \text{sgn}(w^t) \right) \\ &= w^t - \eta \frac{\partial L}{\partial w} - \eta \lambda \text{sgn}(w^t) \quad \text{Always delete} \\ &= (1 - \eta \lambda)w^t - \eta \frac{\partial L}{\partial w} \quad \dots \text{L2} \end{aligned}$$

与原来的参数更新公式相比较，可以发现后面多了一项 $-\eta\lambda \operatorname{sgn}(w^t)$ ，表示参数在原来的基础上都要减去一个小于1的数

L1 vs L2

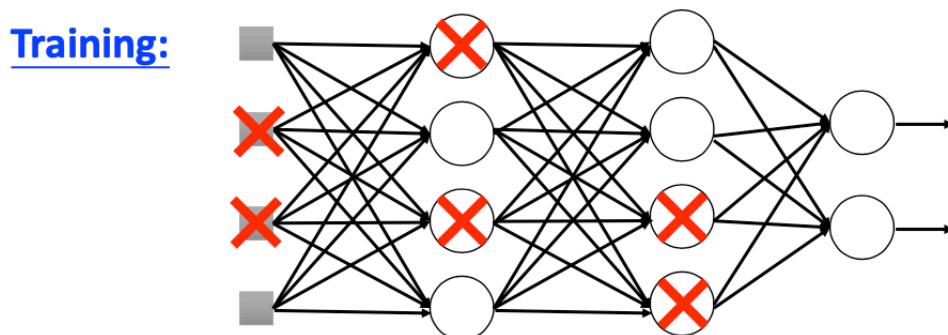
参数更新公式分别如下，

$$L1 : w^{t+1} = w^t - \eta \frac{\partial L}{\partial w} - \eta\lambda \operatorname{sgn}(w^t)$$

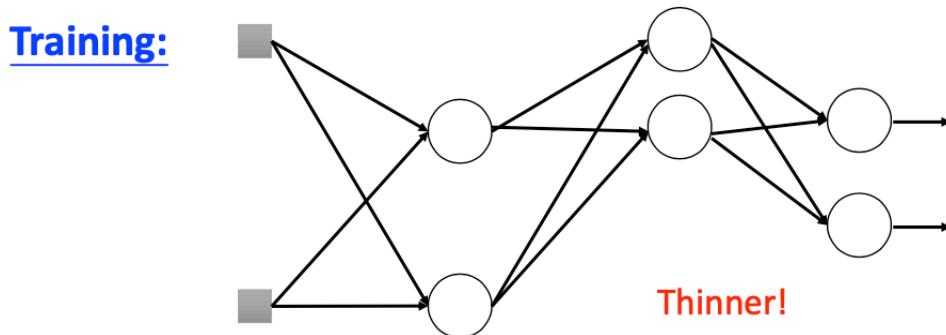
$$L2 : w^{t+1} = (1 - \eta\lambda)w^t - \eta \frac{\partial L}{\partial w}$$

- L1参数更新公式每次都多减去了一个小于1的固定值 (**constant**)
- L2参数更新公式中每次都将前一次的参数乘上一个小于1的值

Dropout



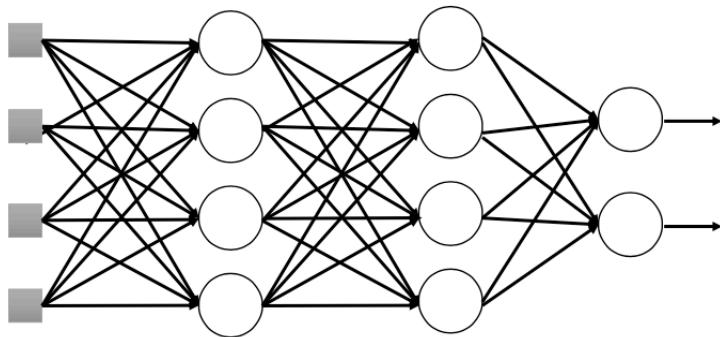
- Each time before updating the parameters
- Each neuron has p% to dropout



- 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

Testing:



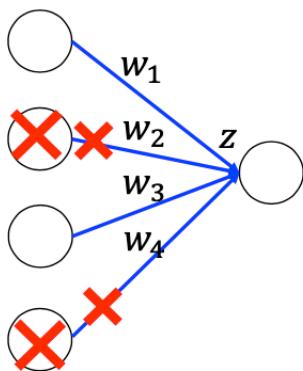
➤ No dropout

- 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.

- Why the weights should multiply $(1-p)\%$ (dropout rate) when testing?

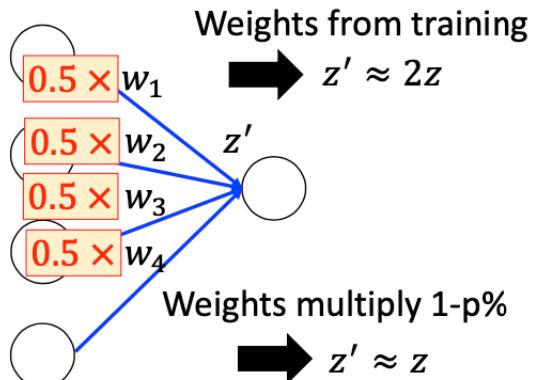
Training of Dropout

Assume dropout rate is 50%

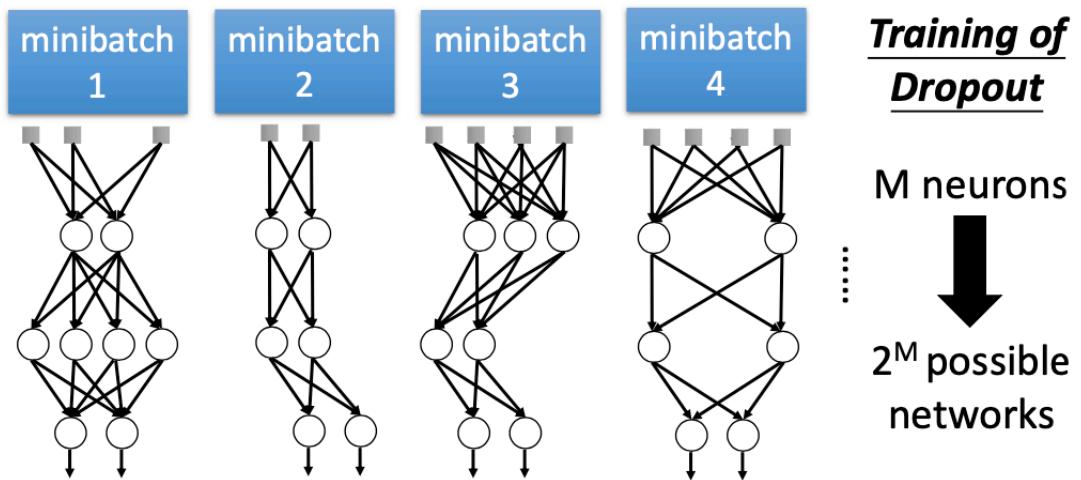


Testing of Dropout

No dropout

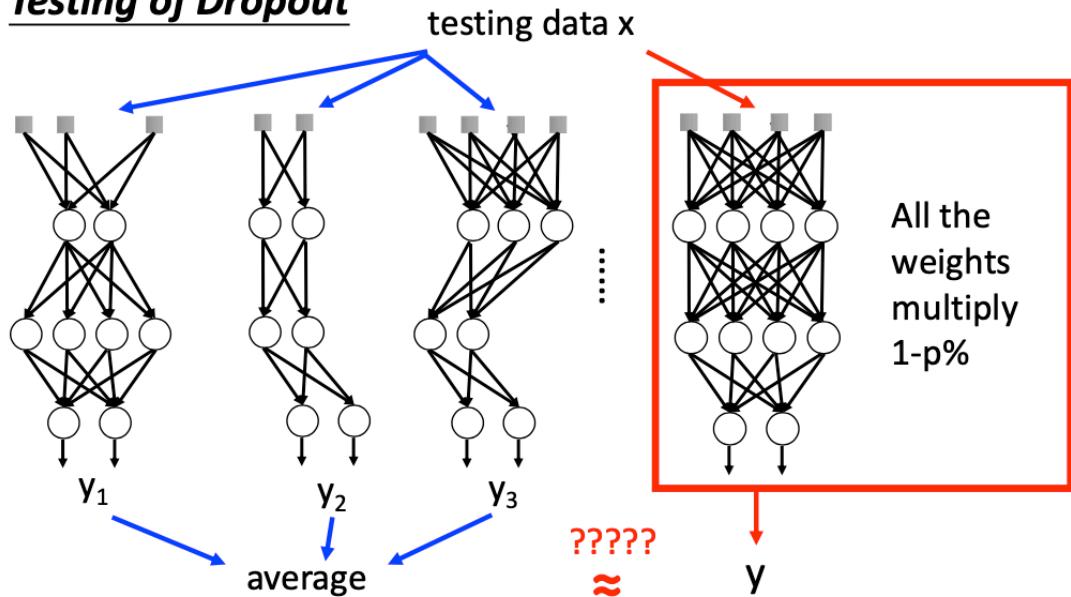


Dropout is a kind of ensemble.



- Using one mini-batch to train one network
- Some parameters in the network are shared

Testing of Dropout



Testing of Dropout

