Deep learning

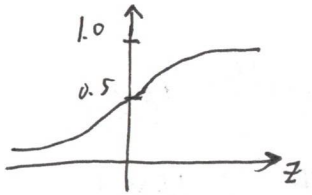
~~Week~~ Course 1 : Neural network and deep learning

week 2 : Basics of NN programming.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}, \quad Y = [y^{(1)}, y^{(2)}, \cdots y^{(m)}]$$

$1 \times m$

$n_x \times m$



$$\hat{y} = \sigma(w^T x + b)$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

~~cost func~~. $\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

Loss func : $L(\hat{y}, y) = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$

cost func:

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}).$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(-y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)})\right)$$

Gradient Descent :

$$w := w - a \frac{\partial J(w,b)}{\partial w}, \quad b := b - a \frac{\partial J(w,b)}{\partial b}$$

computation graph (参看 2.7)

2.9 logestic regression Gradient ~~Descent~~

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

$$\hat{y} = a = \sigma(z) = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

$$L(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

考虑单个样本，其 cost func:

$$L(a, y) = -[y \log(a) + (1-y)\log(1-a)]$$

a 是 ~~regress~~ 逻辑回归输出，y 是样本的标签值

$$w := w - a \frac{\partial J(w,b)}{\partial w}, \quad b := b - a \frac{\partial J(w,b)}{\partial b}$$

$$\frac{dL(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{dL}{dz} = \frac{dL}{da} \cdot \frac{da}{dz}, \quad \text{已知 } \frac{da}{dz} = a \cdot (1-a) \quad (a = \sigma(z))$$

$$\frac{dL}{dz} = \frac{dL}{da} \cdot \frac{da}{dz} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) \cdot a(1-a) = a - y$$

$\frac{\partial L}{\partial w_1} = x_1 \cdot dz$

$\frac{\partial L}{\partial w_2} = x_2 \cdot dz$ (dw₂)

$\frac{\partial L}{\partial b} = dz$ (db)

使用 $dw_1 = x_1 \cdot dz$, $dw_2 = x_2 \cdot dz$, $db = dz$

更新 $w_1 = w_1 - \alpha dw_1$

$w_2 = w_2 - \alpha dw_2$

$b = b - \alpha db$

2.11 vectorization.

2.18 Explanation of logistic regression cost fun.

$\hat{y} = \sigma(w^T x + b)$

$\sigma(z) = \sigma(w^T x + b) = \frac{1}{1+e^{-z}}$

设 $\hat{y} = P(y=1|x)$, 给定特征 x 条件下 $y=1$ 的概率

if $y=1$: $P(y|x) = \hat{y}$

if $y=0$: $P(y|x) = 1-\hat{y}$

令其上面两个概率公式: $P(y|x) = \hat{y}^y (1-\hat{y})^{(1-y)}$

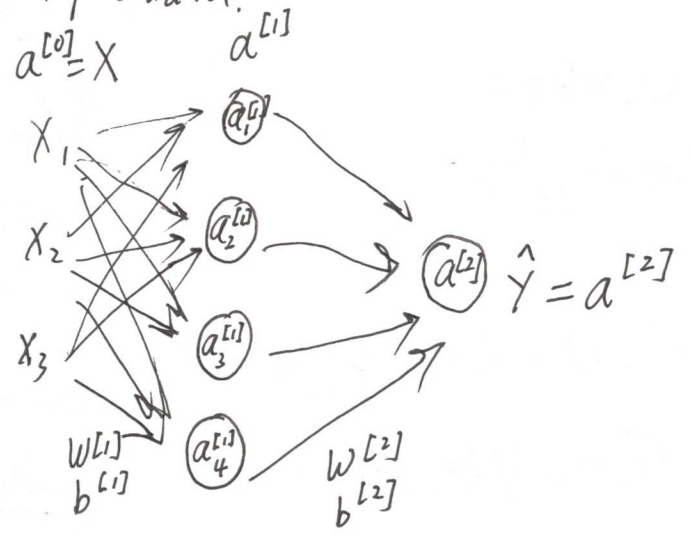
log 是单调递增函数, $\log\left(\hat{y}^{(y)}(1-\hat{y})^{(1-y)}\right) \Rightarrow$

$$y \log \hat{y} + (1-y)\log(1-\hat{y})$$

week 3. shallow ~~neural~~ neural networks

3.1 NN overview.

3.2 NN representation.



$W^{[1]}$: 4×3

$b^{[1]}$: 4×1

$W^{[2]}$: 1×4

$b^{[2]}$: 1×1

3.3 Computing a NN's output

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}, \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \overbrace{\begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ \cdots & w_3^{[1]T} & \cdots \\ \cdots & w_4^{[1]T} & \cdots \end{bmatrix}}^{W^{[1]}} * \overbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}^{\text{input}} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$z^{[1]} = W^{[1]} x + b^{[1]}$, $a^{[1]} = \sigma(z^{[1]})$, $z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$, $a^{[2]} = \sigma(z^{[2]})$

3.4 Vectorizing across multiple examples

若有m个样本，

用第一个样本 $x^{[1]}$ 计算预测值 $\hat{y}^{[1]}$

... = ... $x^{[2]}$ ——— $\hat{y}^{[2]}$

到 m $x^{[m]}$ ~~~ $\hat{y}^{[m]}$

$a^{[2](1)}$, $a^{[2](2)}$ ...

$a^{[2](i)}$, (i)是第i个训练样本，[2]是第二层

向量化：

$$X=\begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$$Z^{[1]}=\begin{bmatrix} \vdots & \vdots & & \vdots \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$$A^{[1]}=\begin{bmatrix} \vdots & \vdots & & \vdots \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$$\left.\begin{aligned} z^{[1](i)} &= W^{[1]}x^{(i)}+b^{[1]} \\ a^{[1](i)} &= \sigma(z^{[1](i)}) \\ z^{[2](i)} &= W^{[2]}a^{[1](i)}+b^{[2]} \\ a^{[2](i)} &= \sigma(z^{[2](i)}) \end{aligned}\right\} \Rightarrow \left\{\begin{aligned} A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]}+b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \end{aligned}\right.$$

3.6 Activation functions

之前之使用过sigmoid函数一种 g activation func.

可使用不同的函数 $g(z^{[1]})$，如tan函数，双曲正切函数

$a=\tan(z)$ 的值域 $+1\sim-1$

$a=\tanh(z)=\frac{e^z-e^{-z}}{e^z+e^{-z}}$，值域：$+1\sim-1$

$g(z^{[1]})=\tanh(z^{[1]})$ 优于 sigmoid，因其数据均值更接近0而非0.5.

二分类问题还是用 sigmoid 函数好.

sigmoid, tanh 的共同缺点：z特大 or 特小时，梯度 ≈ 0，↓Gradient Descent 加速度.

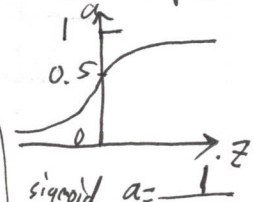
Relu函数：$a=\max(0,z)$ ：z为正，导数为1；z为负，导数为0，z=0，导数为1 or 0.

选择 activation func 的经验：

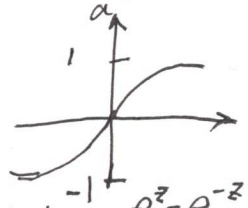
① 若输出是二分类问题：引输出用 sigmoid，其他单元用 Relu.

② 都告不确定时，用 Relu，有时也用 tanh，但 Relu 优点是z为负时，导数为0.

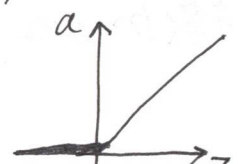
③ 另一版本的 Relu，Leaky Relu.，z为负时导数有很小



sigmoid, $a=\frac{1}{1+e^{-z}}$



tanh, $a=\frac{e^z-e^{-z}}{e^z+e^{-z}}$



Relu, $a=\max(0,z)$

leaky Relu,
$a=\max(0.01z,z)$

综之：

① Relu, Leaky Relu 的导数都>0，只需 if-else 语句，sigmoid 需浮点运算，前者更快。

② sigmoid, tanh 在正负饱和区的梯度≈0，造成梯度弥散，Relu, Leaky Relu 已改大于0部分导数为常数，不会弥散，Relu 在负半区梯度为0，神经元此时不训练，产生稀疏性，Leaky Relu 不会。

注在 Relu 的负一半为0，但 hidden layer 足够大使得 $z>0$，仍足够快。

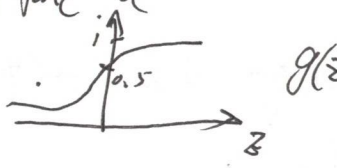
3.7 Why need a nonlinear activation function.

总而言之：若全部都是在 hidden layer 用线性 activation func，在输出层用 sigmoid func，那么这个模型的复杂度和没有任何 hidden layer 的标准 logistic regression 一样。

线性隐层的组合还是线性函数，所以引入非线性，否则神经网络就失去了意义。

3.8 Derivatives of activation func.

1) sigmoid activation func:



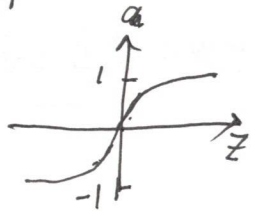
$$g(z)=\frac{1}{1+e^{-z}}$$

$$\frac{dg(z)}{dz}=\frac{-(-e^{-z})}{(1+e^{-z})^2}=\frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right)=g(z)(1-g(z))$$

$z=\pm 10$ 时，$\frac{dg(z)}{dz}\approx 0$

$z=0$ 时，$\frac{dg(z)}{dz}=\frac{1}{4}$

2) Tanh activation func.



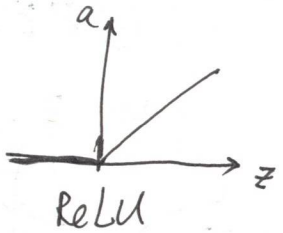
$$g(z)=\tanh(z)=\frac{e^z-e^{-z}}{e^z+e^{-z}}$$

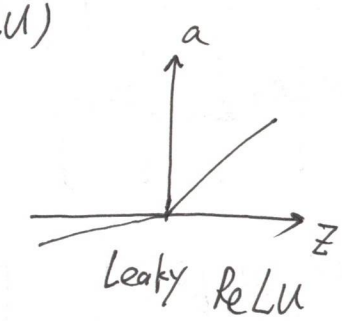
$$\frac{dg(z)}{dz}=1-(\tanh(z))^2$$

$z=\pm 10$，$\frac{dg(z)}{dz}\approx 0$

$z=0$，$\frac{dg(z)}{dz}=1-0=1$

3) Rectified Linear Unit (ReLU)





$$g(z)=\max(0,z)$$

$$g(z)'=\begin{cases}0 & \text{if } z<0\\ 1 & \text{if } z>0\\ \text{undefined} & \text{if } z=0\end{cases}$$

$z=0$ 时，z 导数为0 or 1.

W) Leaky linear unit (Leaky ReLU)

$$g(z) = \max(0.01z, z)$$

$$g(z)' = \begin{cases} 0.01 & , \text{if } z<0 \\ 1 & , \text{if } z>0 \\ \text{undefined} & , \text{if } z=0 \end{cases}$$

$z=0$ 时，导数为 0.01 or 1

3.9 Gradient descent for NN

n_x：输入特征个数

$n^{[1]}$：隐藏层单元个数

$n^{[2]}$：输出层单元个数

$w^{[1]}$: $n^{[1]} \times n^{[0]}$

$b^{[1]}$: $n^{[1]} \times 1$

$w^{[2]}$: $n^{[2]} \times n^{[1]}$

$b^{[2]}$: $n^{[2]} \times 1$

forward propagation:

① $z^{[1]} = W^{[1]}x + b^{[1]}$

② $a^{[1]} = \sigma(z^{[1]})$

③ $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

④ $a^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

back propagation:

① $dz^{[2]} = A^{[2]} - Y$, $Y = [y^{[1]}, y^{[2]}, \dots y^{[m]}]$

② $dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$

③ $db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims=True)$ 别错.

④ $dz^{[1]} = \underbrace{W^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} * g^{[1]'} * \underbrace{(z^{[1]})}_{(n^{[1]}, m)}$ — ?

activation func. of hidden layer

⑤ $dW^{[1]} = \frac{1}{m} dz^{[1]} x^T$

⑥ $db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims=True)$ $(n^{[1]}, 1)$

3.10 Back propagation Intuition.

$\left.\begin{matrix} x \\ w \\ b \end{matrix}\right\} \Rightarrow z = w^T x + b \Rightarrow a = \sigma(z) \Rightarrow L(a,y)$

$$da = \frac{dL(a,y)}{da} = (-y\log a - (1-y)\log(1-a))' = -\frac{y}{a} + \frac{1-y}{1-a}$$

$dz^{[2]} = a^{[2]} - y$, $dw^{[2]} = dz^{[2]} a^{[1]T}$, $db^{[2]} = dz^{[2]}$

$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$.

$dw^{[1]} = dz^{[1]} x^T$, $db^{[1]} = dz^{[1]}$

回归 2.9.

$\leftarrow dz = a - y$

$\leftarrow dw_2 = x_2 \cdot dz$

$\leftarrow db = dz$

$\leftarrow dw_1 = x_1 \cdot dz$

强化推导.

3.11 Random Initialization

若w全初始化为0，则每层各单元相同.

随机初始化：

$W^{[1]} = np.random.randn(2,2) * 0.01$

$b^{[1]} = np.zeros(2,1)$

$W^{[2]} = np.random.randn(2,2) * 0.01$

$b^{[2]} = 0$

为什么 *0.01，因为若w很大，z很大，tanh or sigmoid就在斜部，梯度下降慢.

Week 4 Deep Neural Networks

4.1 Deep L-layer neural network

符号约定：L：层数

输入层不算在L中，为0层

$n^{[l]}$ 为第l层层数

4.2 Forward and backward propagation.

forward propagation

$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$, $a^{[l]} = g^{[l]}(z^{[l]})$

向量化：$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$, $A^{[l]} = g^{[l]}(Z^{[l]})$

back propagation：

① $dz^{[l]} = da^{[l]} .* g^{[l]\prime}(z^{[l]})$?

② $dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$

③ $db^{[l]} = dz^{[l]}$

④ $da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$

⑤ $dz^{[l]} = w^{[l+1]T} dz^{[l+1]} \cdot g^{[l]\prime}(z^{[l]})$

（⑤由④代入①中得到）

循环

向量化：① $dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$

② $dw^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$

③ $db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$

④ $dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$

4.3 Forward propagation in a Deep network.

4.4 Getting your matrix dimensions right

维数：第 l 层：

$w^{[l]}$: $(n^{[l]}, n^{[l-1]})$

$b^{[l]}$: $(n^{[l]}, 1)$

$z^{[l]}, a^{[l]}$: $(n^{[l]}, 1)$

$dw^{[l]}, w^{[l]}$ 维数相同

$db^{[l]}, b^{[l]}$ 一样同.

w, b 维数不变，但 z, a 以及 X 的维数会在向量化后改变.

向量化，m 个样本，

$Z^{[l]} = (z^{[l][1]}, z^{[l][2]}, \dots z^{[l][m]})$

$(n^{[l]}, m)$

$A^{[l]}$: $(n^{[l]}, m)$, $A^{[0]} = X = (n^{[l]}, m)$

$Z^{[l]} = W^{[l]} \cdot X + b^{[l]}$

4.5 Why Deep representation?

前面几层学简单的，之后的层学更复杂的特征。

若不个层数，就要指数级↑ neuron 数量

4.6 Buildg blocks of deep neural networks.

4.7 parameters vs Hyperparameters.

Hyperparameters: learning rate α, Iteration, L (# of hidden layers)

$n^{[l]}$ (# neurons in one hidden layer), choice of activation function.

momentum, mini batch size, regularization parameters

如何寻找最优值：idea - code - experiment - idea 循环往复

第二门课：Improving Deep NN：

Hyperparameter tuning, Regularization and optimization.

Week 1：practical aspects of Deep learning

1.1 Train / dev / test sets

train set, dev set (cv set), test set，60%, 20%, 20%

百万级数据，后两个set可小一些

1.2 Bias / Variance.

如何减小 high bias, high variance.

1.3 Basic Recipe for Machine Learning

训练model时，先减重bias，把bias↓到能拟合，再想法（更多模型，更多数据，正则化）减重 ↓ variance.

1.4 Regularization.

减小overfitting (high variance)的办法：① regularization ② 更多数据.

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

L_2 regulation：$\|w\|_2^2=\sum_{j=1}^{n} w_j^2=w^Tw$ ← 把有元素的平方和

L_1 reg —：$\|w\|_1$　　$\|\cdot\|_2^2$ 称为 Frobenius norm

GD：

$$w^{[l]}=w^{[l]}-\alpha[\text{from back propa.}+\frac{\lambda}{m}w^{[l]}]$$

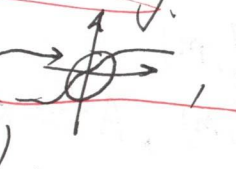
$$=\left(1-\frac{\alpha\lambda}{m}\right)w^{[l]}-\alpha\cdot\text{from back prop.}$$

如regularized logistic regression：（参考 ML课 7.3）

$$\theta_j:=\theta_j-\alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}+\frac{\lambda}{m}\theta_j\right],\ \text{for } j\neq 0.$$

1.5 why regularization reduces overfitting

$\lambda\uparrow$, $w\downarrow$, $z\downarrow$, z 都在小区域时



，tanh 呈现近似线性，使网络简单。（or，使每层的影响↓）

1.6 Drop out regularization

x_1	0	0	
x_2	0	0	0
x_3	0	0	
x_4	0	0	

在用 NN overfitting，用 drop out 法，在每一层，设置消除节点个体立的概率，最终消除一些节点，得到节点更少的 NN

获得概率分布向量 $d3$ 后，

$$a_3=d_3*a_3$$

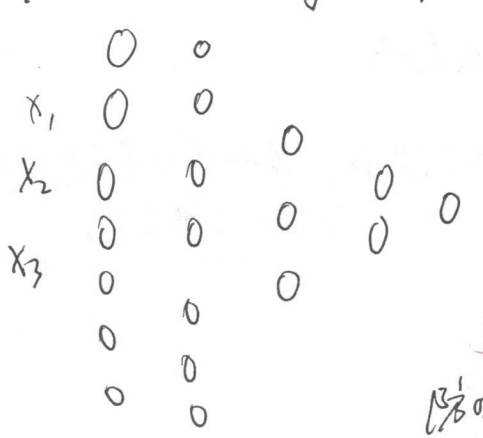
使 a_3 中相应元素归0

最后：$a_3 = \frac{a_3}{keep\_prob}$

使 $z^{[4]}$ 的期望不受影响。

测试阶段不用 drop out 方法。

1.7 understanding Dropout



$w^{[1]}: 7\times 3$

$w^{[2]}: 7\times 7$

$w^{[3]}: 3\times 7$

$w^{[2]}$ 权重最大，(7×7)，为防 overfitting，keep-prob 选一些，0.5.

如某层 overfitting 不严重，keep-prob 高一些，0.7.

若不担心 overfitting，keep-prob = 1.

1.8 Other regularization methods.

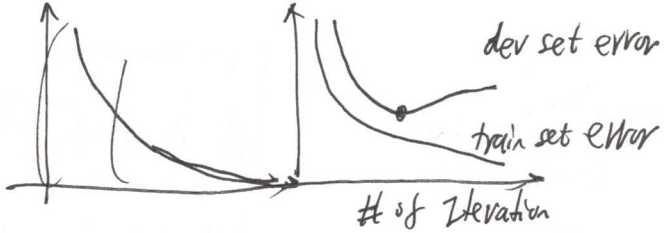
(1). Data augmentation.

图片翻转，剪切 | 最好加更多数据，但很昂贵

扭曲.

(2) early stopping

训练刚开始 w 较小，但随着训练↑，w↑



early stopping 就是在 w 较大时停下.

"变化"：一次二个时间只做一个任务。如防 overfitting，只关注 w, b 使 J(w,b) 最小；防 high variance，再用另一套工具去实现.

early stopping 的缺点：不能独立处理上面两个问题.

优点：只运行一次 Gradient Descent，找出 w 较小值.

L2 regularization 缺点：训练时间长，尝试很多种入参数.

1.9 Normalizing Inputs

把"归一化"输入可以加速训练 model.

假设有两个特征，normalization 需 2 步

① subtract out or to zero out the mean. $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$

$X := X - \mu$

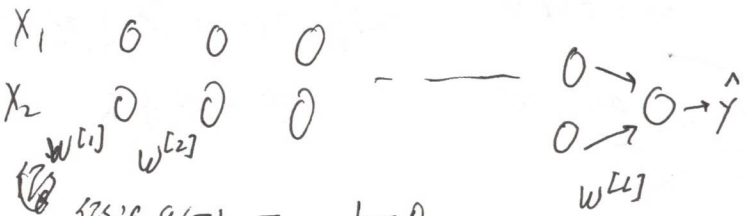
② Normalized variance: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)**2}$

$X = \frac{X}{\sigma^2}$

在 train set 上得到的 μ, σ^2 也全用到 dev set 上.

1.10 Vanishing / Exploding gradients.

导数或坡度有时会变得非常大，或非常小.



假设 $g(z)=z$, $b=0$.

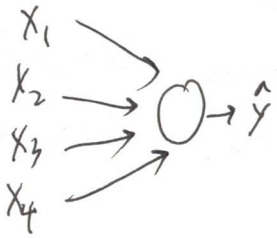
$\hat{y}=w^{[L]}w^{[L-1]}\cdots w^{[3]}w^{[2]}w^{[1]}x$ explode

若 $w^{[i]}=\begin{bmatrix}1.5&0\\0&1.5\end{bmatrix}$, 则 $\hat{y}=w^{[L]}\begin{bmatrix}1.5&0\\0&1.5\end{bmatrix}^{L-1}x$

$w^{[i]}=\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix}$, $\hat{y}=w^{[L]}\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix}^{L-1}x$ Vanish

1.11 Weight Initialization for deep networks

一个 neuron 的例子



$z=w_1x_1+w_2x_2+\cdots+w_nx_n$, $b=0$

为防 z ~~更大也要~~ 过大or过小, 设 $w_i=\frac{1}{n}$

n 为 输入特征数量. , $Var(w)=\frac{1}{n}$

$w^{[L]}=$ np.random.randn(shape) * np.sqrt($\frac{1}{n^{[L-1]}}$)

若 Relu, $g^{[L]}(z)=Relu(z)$, ↑替换 np.sqrt($\frac{2}{n^{[L-1]}}$)

variance $=\frac{2}{n}$

~~求 tanh~~ $\sqrt{\frac{1}{n^{[L-1]}}}$ ~~替换 np.sqrt 项~~,

求用 tanh 激活, 用 $\sqrt{\frac{1}{n^{[L-1]}}}$ 替换 np.sqrt() 项, 称为 Xavier 初始化

Yoshua Bengio : $\sqrt{\frac{2}{n^{[L-1]}+n^{[L]}}}$

1.12 Numerical approximation of gradients

$\frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon}$ 双边误差比单边准, 确认 $g(\theta)$ 是否是 $f(\theta)$ 导数的正确实现.

1.13 Gradient Checking

$w^{[1]},b^{[1]},\cdots w^{[L]},b^{[L]}$ → 向量化, θ

$dw^{[1]},db^{[1]},\cdots dw^{[L]},db^{[L]}$ → 向量化, $d\theta$

$d\theta_{approx}[i]=\frac{J(\theta_1,\theta_2,\cdots\theta_i+\varepsilon,\cdots)-J(\theta_1,\theta_2,\cdots\theta_i-\varepsilon,\cdots)}{2\varepsilon}$

计算 $\frac{\|d\theta_{approx}-d\theta\|_2}{\|d\theta_{approx}\|_2+\|d\theta\|_2}$ ← 误差平方和 ← 平方根

$\varepsilon=10^{-7}$, $\approx 10^{-7}$ - good

$\approx 10^{-5}$ 担忧

$\approx 10^{-3}$ - worry

1.14 Gradient Checking Implementation Notes.

① 不要在训练中用 Gradient checking，只用来检查

② 若 Gradient checking 失败，要检查bug所在

③ 注意正则化

④ Gradient checking 与 dropout 不能同时用

⑤ ---

Week 2 Optimization algorithms

2.1 Mini-batch Gradient ~~checking~~ descent.

$X = [x^{(1)}\ x^{(2)}\ x^{(3)} \cdots x^{(m)}]$, (n_x, m)

$Y = [y^{(1)}\ y^{(2)}\ y^{(3)} \cdots y^{(m)}]$, $(1, m)$

把训练集分割为小一点的子集，每个为 mini-batch

假设 500 万的样本，每个 mini-batch 为 1000，共有 5000 个 mini-batch.

$X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \cdots x^{(1000)}}_{X^{\{1\}}} \mid \underbrace{x^{(1001)} \cdots x^{(2000)}}_{X^{\{2\}}} \mid \cdots \mid \underbrace{\cdots x^{(m)}}_{X^{\{5000\}}}]$ (n_x, m)

$Y = [\underbrace{y^{(1)}\ y^{(2)} \cdots y^{(1000)}}_{Y^{\{1\}}} \mid \underbrace{y^{(1001)} \cdots y^{(2000)}}_{Y^{\{2\}}} \mid \cdots \mid \underbrace{\cdots y^{(m)}}_{Y^{\{5000\}}}]$ $(1, m)$

mini-batch 记号为 t，$X^{\{t\}}$ $(n_x, 1000)$，$Y^{\{t\}}$ $(1, 1000)$

repeat

{ For $t = 1, 2, \cdots 5000$

{ forward prop. on $X^{\{t\}}$

$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$\vdots$

$A^{[L]} = g^{[L]}(Z^{[L]})$

compute cost. $J = \frac{1}{1000} \sum_{i=1}^{l} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum \|w^{[l]}\|_F^2$

Backprop to compute gradients $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$

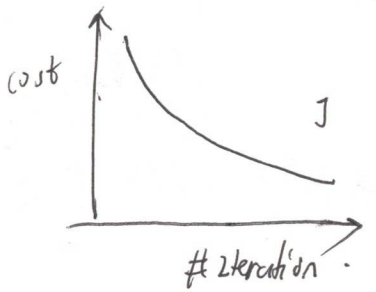
$b^{[l]} := b^{[l]} - \alpha db^{[l]}$

}

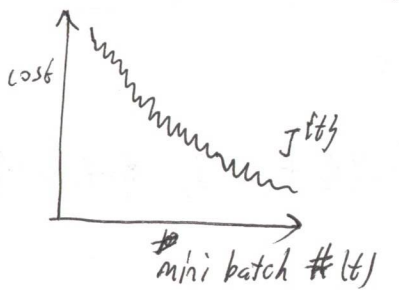
}

"1 epoch" ← single pass through training set.

2.2 Understanding Mini-batch gradient ~~descent~~ descent



Batch GD

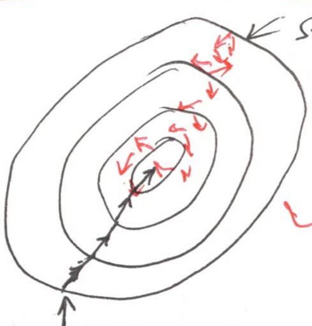


Mini-batch GD.

要注意 mini batch 的大小，~~(即 m 的大小)~~

若等于训练样本的大小(m)，即为 Batch GD.

若 $=1$，即为 stochastic GD.



mini Batch 的大小在 $1 \sim m$ 之间.

经验：

若 mini-Batch GD 大范围波动，↓ learning rate

若样本小于2000，直接用 Batch GD

样本大，mini batch 在 64～512之间

(必须是2的次方)

2.3 Exponentially weighted averages

温度的例子.

V_θ

$V_1 = 0.9V_0 + 0.1\theta_1$ 前一天的0.9，今天的0.1

⋮

$V_t = 0.9V_{t-1} + 0.1\theta_t$

（$0.9 \to \beta$，$0.1 \to 1-\beta$）

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$ 可记 V_t 为 $\frac{1}{1-\beta}$ 的平均温度

$\beta=0.9$，$\frac{1}{1-\beta}=10$，也就是十天的平均值

$\beta=0.98$，$\frac{1}{1-\beta}=50$，— 50天 —

2.4 Understanding exponentially weighted averages → 占内存少

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$\theta_{100} = 0.9V_{99} + 0.1\theta_{100}$

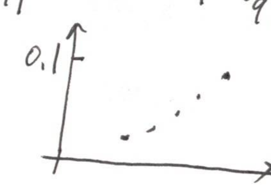
$\theta_{99} = 0.9V_{98} + 0.1\theta_{99}$

$\theta_{98} = 0.9V_{97} + 0.1\theta_{98}$

⋮ 不停代入

$$V_{100} = 0.1\theta_{100} + 0.1\times0.9\theta_{99} + 0.1\times0.9^2\theta_{98} + 0.1\times0.9^3\theta_{97} + 0.1\times(0.9)^4\theta_{96} + \cdots$$

exponential decay func:



相当于当日温度与 exponential decay func 相乘，最后求和，得到 V_{100}

偏差修正：$0.1\times0.1\times(0.9)^1\times0.1\times(0.9)^2\times0.1\times(0.9)^3\times\cdots$ → 趋近!

代码中，令 $V_\theta = 0$

$V_\theta := \beta V + (1-\beta)\theta_1$

$V_\theta := \beta V + (1-\beta)\theta_2$

⋮

2.5 Bias correction in exponentially weighted averages.

设 $V_0 = 0$ 时, $V_1 = 0.98 V_0 + 0.02\theta_1 = 0.02\theta_1$

$V_2 = 0.98 V_1 + 0.02\theta_2$

$= 0.0196\theta_1 + 0.02\theta_2$

前面几天估计不准, 偏小,

故不用 V_t, 改用 $\frac{V_t}{1-\beta^t}$ ✓

$t=2$ 时, $\frac{V_2}{1-0.98^2} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$

主要为了修正, 使

t 很大时, $\beta^t \approx 0$.

2.6 Gradient Descent with momentum (快于梯度的 GD)

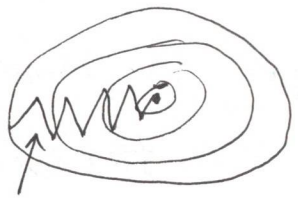
基本思想: 计算梯度的指数加权平均数, 并利用该梯度更新权重.

$V_{dw} = \beta V_{dw} + (1-\beta) dw$

$V_{db} = \beta V_{db} + (1-\beta) db$

$w := w - \alpha V_{dw}$

$b := b - \alpha V_{db}$



GD.

想减小纵向波动

加快向右移动

回想之前天气变化例子, GD with momentum 就是在减小纵向波动.

2.7 Root Mean Square prop. (RMSprop)

$S_{dw} = \beta S_{dw} + (1-\beta) dw^2$

$S_{db} = \beta S_{db} + (1-\beta) db^2$

$w := w - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon}$

$b := b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$

db 较大, dw 较小

S_{db} 大 S_{dw} 小

b 小 w 大

为避免除 0, 在 w, b 的更新中加入 $\varepsilon = 10^{-8}$

结论: 减小摆动, 可使用更大的学习率 α, 加快算法.

2.8 Adam Optimization Algorithm (Adaptive Moment Estimation).

$V_{dw} = 0$, $S_{dw} = 0$, $V_{db} = 0$, $S_{db} = 0$

One Iteration t:

compute dw, db using current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ ← momentum, β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db$ ← RMSprop

修正

$V_{dw}^{corrected} = \frac{V_{dw}}{(1-\beta_1^t)}$, $V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_1^t)}$

$S_{dw}^{corrected} = \frac{S_{dw}}{(1-\beta_2^t)}$, $S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$

$w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}$, $b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$

默认 $\beta_1 = 0.9$

$\beta_2 = 0.999$

Hyperparameters choice:

- α: needs to be tune
- β_1: 0.9
- β_2: 0.999
- ε: 10^{-8}

2.9 Learning rate decay

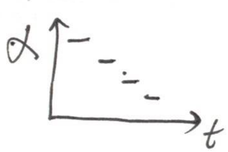
加快学习算法的一方法：随时间 ↓ learning rate
称为 learning rate ~~decy~~ decay

衰减 $a = \frac{1}{1 + \text{decay rate} * \text{epoch\_num}} a_0$

decay rate ↑ 需tune

epoch_num ↑ mini-batch中的代数

other ~~l~~ learning rate decay methods

- $\alpha = 0.95^{\text{epoch\_num}} \cdot \alpha_0$ exponential decay
- $\alpha = \frac{k}{\sqrt{\text{epoch\_num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \alpha_0$
-  discrete stair cease

2.10 The problem of local optima.

高维的 saddle point.

Week 3. Hyperparameter tuning, Batch Normalization, and programming framework

3.1 Tuning process

Hyperparameters:

- α
- β
- $\beta_1, \beta_2, \varepsilon$
  (0.9), (0.999) (10^{-8})
- # layers
- # hidden units
- learning rate decay
- mini-batch size.

选值：
- ① 随机取一些点
- ② 先粗后细的取样点

3.2 Using an appropriate scale to pick hyperparameters.

对0.001~0.1这种取对数再平均选择

3.3 Hyperparameters tuning in practice: Pandas vs. Caviar

两种方式：
- 一种是一个model慢慢调
- 一种是对多个model不同参数同时执行

3.4 Normalizing activations in a network

既可以 normalize 输入 $X_1, X_2, \cdots$

也可以 normalize 隐藏层的 $Z^{[1]}, Z^{[2]}, \cdots$

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

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

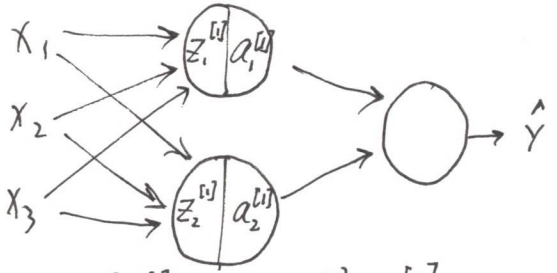
$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

这时 hidden units 总是 $\sim N(0, 1)$，我们想要其分布不那么相似，

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$，γ, β 要训练

if $\gamma = \sqrt{\sigma^2 + \varepsilon}$，$\beta = \mu$，then $\tilde{z}^{(i)} = z^{(i)}$

3.5 Fitting Batch norm into a NN.



$$X \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]})$$

$$\xrightarrow{w^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[\text{Batch Norm}]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]} \cdots$$

in tensorflow, Batch norm: tf.nn.batch_normalization.

Batch Norm 常和训练集的 mini-batch 一起使用

对第一个 mini-batch $X^{\{1\}}$，计算 $z^{[1]}$，

X 得 $\xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow g^{[1]}(\tilde{z}^{[1]}) = a^{[1]} \xrightarrow{w^{[2]}, b^{[2]}} z^{[2]} \rightarrow \cdots$

类似地对 $X^{\{2\}}$ 去做

这其中每次 B.N. 计算 $z^{[l]}$ 时，(前一步 $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$)，$b^{[l]}$ 都被抹去了，可设其为 0，$z^{[l]} = w^{[l]} a^{[l-1]}$，$\tilde{z}^{[l]} = \gamma^{[l]} z^{[l]} + \beta^{[l]}$

$\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$，$\gamma^{[l]} = \gamma^{[l]} - \alpha d\gamma^{[l]}$

3.6 Why does Batch Norm work?

① 对隐藏层归一化

② 使权重比你的网络更滞后或更深层。

"covariate shift"。

Batch 归一化能保证 $z_1^{[2]}, z_2^{[2]}$ 的均值和方差不变，使后层更坚实。

有 BN 有轻微的 regularization 的效果，因每次对一个 mini-batch 算 μ, σ^2 有噪声。

用较大的 mini-batch 可减噪声。

3.7 Batch Norm at test time.

BN 对 mini-batch 逐一处理，但测试时要对每个样本处理，how?

$$\mu = \frac{1}{m}\sum_i z^{(i)}, \quad \sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \quad \tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

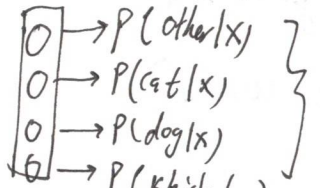
训练时，μ, σ^2 是对整个 mini-batch 来说的，测试时需要的 μ, σ^2 是根据训练集估计的。

通常用指数加权平均来估计 μ, σ^2。

3.8 Softmax Regression

识别大于两个类，

比如猫(1)，狗(2)，鸡(3)，其他(0)四类别：



$\hat{y}$, (4,1)

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

Activation function:

$t = e^{(z^{[l]})}$

$a^{[l]} = \frac{e^{(z^{[l]})}}{\sum_{j=1} t_j}$

$z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$

$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$, $\sum_{j=1}^{4} t_j = 176.3$

$a^{[l]} = \frac{t}{176.3}$

3.9 Training a softmax classifier

hard max $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 只在概率最大的类别置1，其他为0。

soft max：四个概率

softmax 也被称为 logistic regression 推广到 C 类 ($C \geq 2$)

Loss function 都比较倾向于把预测对的类别，使该类别概率变大↑

对一个样本，loss func：$L(\hat{y}, y) = -\sum_{j=1} y_j \log \hat{y}_j$

整个训练集：$J(w^{[1]}, b^{[1]}, \cdots) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

$Y = [y^{(1)}\ y^{(2)}\ y^{(3)} \cdots y^{(m)}] = \begin{bmatrix} 0 & 0 & 1 & \cdots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}$ (4,m)

$\hat{Y} = [\hat{y}^{(1)}, \hat{y}^{(2)} \cdots y^{(m)}] = \begin{bmatrix} 0.3 & & \\ 0.2 & & \\ 0.1 & \cdots & \\ 0.4 & & \end{bmatrix}$ (4,m)

反向的其他一步：$dz^{[l]} = \frac{\partial J}{\partial z^{[l]}} = \hat{y} - y$，其他的都差不多

3.10 Deep learning frameworks

3.11 Tensorflow 见 Notes

Course 3 Structuring Machine Learning projects.

Week 1 ML

1.1 Why ML strategy?

指导哪些改进策略更值得尝试

1.2 Orthogonalization. (正交化)

正交化指：一个维度只调一个变量.

1.3 Single # Evaluation metric

有一个单一指标评估 model 很重要，

precision

Recall

F_1 score

平均值，

⋮

1.4 Satisficing and optimizing metrics.

一个优化指标

其他作为满足指标

1.5 Train / Dev / test distributions.

development set 也叫 hold out cross validation set.

流程：用 train set 训练不同 model，用 dev set 选出最好的 model，用 test set 评估 model.

dev set，和 test set 必须来自同一分布. ← 把他们 shuffle.

1.6 Size of dev and test sets.

1.7 When to change dev/test sets and metrics.

当你的评估指标无法正确衡量算法之间的优劣排序时，应该考虑评估指标.

你用的错误率指标可以写成：$Error = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} I\{y_{pred}^{(i)} \neq y^{(i)}\}$

加权重后：

$$Error = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} w^{(i)} \cdot I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

$x^{(i)}$ 是猫时 $w^{(i)}=1$，是 pornographic 时 $w^{(i)}=10$ or 100，还要↑.

1.8 Why human-level performance?

Bayes optimal error：理论上可能达到的最优错误率

(有时写作 Bayesian)

1.9 Avoidable bias：

~~最优错~~ Bayes optimal error 的估计与训练错误率之间的差值.

1.10 Understanding human-level performance.

得到 Bayes optimal error 的差距，告诉你 avoidable bias ~~多~~ 空间还多大

训练错误率和开发错误率的差值，告诉你 variance 有多大

1.11 surpassing human-level performance.

1.12 · Improving your model performance. 见notes.

名词：

avoidable bias：

Week2 ML Strategy 2.

2.1 · Carrying out error analysis

Error analysis：算法回过到人类的表现，检查算法中的错误。

看那些错的比例，分别那些错误大，就优先改进那些。

2.2 Cleaning up Incorrectly labeled data

Deep learning are quite robust to random errors in traing set.

但对系统性错误就会有问题，

对 dev, test set 上的 incorrectly labeled data，若影响评估就修正，否则就不用，

同时修正 dev, test set 保证他们分布相同

2.3 Build your first system quickly, then iterate

先快速的建立一个系统，可以知道 bias, variance 在哪，有优化的方向。

2.4 · Training and · testy on different distributions.

训练集很大，但与 dev, test set 不太一样

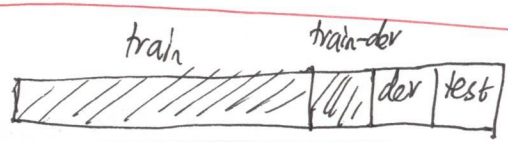
200k 网上图片，10k 用户上传图片

200k + 5000 作 train set，2500，2500 分作 dev, test set

2.5 Bias and Variance with mismatched data distributions

若 Train set data 与 dev/test data 来自不同分布，很难确认误差代表高 variance 还是高 bias.

把 train set 随机打乱，再抽出一个 train-dev set



Train set error : 1% → variance

Train-dev error : 9%

Dev error : 10%

Human error ---- 0% → (avoidable) bias ---- 0% → bias

Traing error : 10% , 10%

Train-dev error : 11% , 11% → mismatch

Dev error : 12% 20%

2.6 addressing data mismatch

mismatch时要做误差分析，理解 difference between training set and dev/test set.

一种解决方法是 Artificial data analysis，但可能只用了一个子集去造model，会 overfitting.

2.7 Transfer learning

用从一个model中习得的知识用于另一个任务中.

~~10000张图~~

什么时候用 transfer learning：当迁移来源问题中有很多数据，迁移目标没那么多数据.

例子：对很多图识别猫图片训练（pre-training），把最后一层拿走，再随机权重，随机初始权重，对放射数据训练，甚至引入几层新层）（fine tuning）.

2.8 Multi-task learning

一张图中同时识别 行人，车辆，停车标志，交通灯，

	$y^{(i)}$, (4×1)
pedestrians	0
cars	1
stop sign	1
traffic light	0

$$Y = \begin{bmatrix} | & | & | & & | \\ y^{(1)} & y^{(2)} & y^{(3)} & \cdots & y^{(m)} \\ | & | & | & & | \end{bmatrix} \quad (4, m)$$

对一个输出 $\hat{y}$，是4×1，对整个训练集的平均损失：

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{4} L(\hat{y}_j^{(i)}, y_j^{(i)})$$

logistic loss $L(\hat{y}_j^{(i)}, y_j^{(i)}) = -y_j^{(i)}\log\hat{y}_j^{(i)} - (1-y_j^{(i)})\log(1-\hat{y}_j^{(i)})$

2.9 What is end-to-end deep learning

用单个神经网络训练.（数据量非常大的时候才好用）

2.10 Whether to use end-to-end learning

优点：
① let the data speak
② less hand-designing of components needed.

缺点：
① May need large amount of data
② Excludes potentially useful hand-designed components.

Course 4, Convolutional Neural Networks

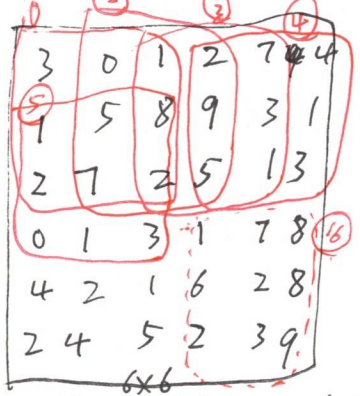
week 1, Fandations of CNN

1.1 Computer vision

1.2 Edge detection example.

一个 6×6 灰度图片，记作 6×6×1。

构造一个 3×3 的 filter，记作符号 "*"



3×3 的框从左上角，右移，下移，直到最后

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$$

3×3　　4×4

用 3×3 的 filter 对 6×6 图片做卷 convolution。

输出是一个 4×4 矩阵。

推①: $\begin{pmatrix} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 2 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} = 3\times1+1\times1+2\times1+0\times0+5\times0+7\times0$
$-1\times1-8\times1-2\times1=-5$

sum

先 element_wise products，再求和。

依次计算。

最左边是 6×6 图片，中间是 filter，右边是另一张 4×4 图片

为什么上面的矩阵可以做 vertical edge detection？

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

6×6　　3×3　　4×4

左亮右暗，中间亮的为垂直边缘

1.3 More Edge detection

上面的矩阵左右翻转一下，

$$\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

6×6

亮→暗过渡（负边）

暗→亮过渡（负边）

Vertical Edge detection

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Horizontal Edge detection

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

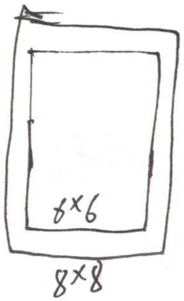
把这九个数字当成参数，通过 back prop，让 NN 自己学出参数是多少是好

1.4 Padding → 填充

对 $n\times n$ 图像，用 $f\times f$ 的 filter，输出为 $(n-f+1)\times(n-f+1)$

两个缺点：
① 图像缩小
② 角落边缘的像素只被记录一次，边缘信息缺失。

为解决缺点，在卷积之前在图像外填充图像。（↑ 用0填充）



$* \; 3\times3 = 6\times6$

若 p 是填充的数量，

$$(n+2p-f+1)\times(n+2p-f+1)$$

2种填充方法：一种是 valid 卷积，不填充

一种是 same 卷积：输出与输入大小一样（填充后）

$$n+2p-f+1 = n \Rightarrow p=\frac{f-1}{2}$$

f 通常是 odd.

(stride 步幅)

1.5 strided convolutions

在上一节中 stride=1，即框向右，向下都只移一格

还可令 stride=2, 3, ...

$f\times f$ 的 filter 卷积 $n\times n$ 的图像，padding 为 p，stride 为 s，输出为 $\left(\frac{n+2p-f}{s}+1\right)\times\left(\frac{n+2p-f}{s}+1\right)$

若 $\frac{n+2p-f}{s}+1$ 不是整数，则向下取整 floor.

保证方框在图像(or 填充后的)内时计算.

真正的卷积是要先把 filter 翻转，$\begin{bmatrix}3&4&5\\1&0&2\\-1&9&7\end{bmatrix}$ 变为 $\begin{bmatrix}7&2&5\\9&0&4\\-1&1&3\end{bmatrix}$

相当于 3×3 的镜像。（顺时针90度，再水平翻转）

用翻转后的 3×3 filter 与 $n\times n$ 图像 element-wise product.

但在NN中，用不翻转的作 filter，（准确应叫 cross-correlation convolution）

1.3 Convolutions over volumes.

$6\times6\times3$ 的彩色图，$3\times3\times3$ 的 filter，输出 $4\times4\times1$（通道匹配）

检测红色边缘：$\begin{matrix}1&0&-1\\1&0&-1\\1&0&-1\end{matrix}$，绿色 $\begin{matrix}1&0&-1\\1&0&-1\\1&0&-1\end{matrix}$ 蓝色 $\begin{matrix}1&0&-1\\1&0&-1\\1&0&-1\end{matrix}$

也可用2个或多个 $3\times3\times3$ 的 filter.

维度：$n\times n\times n_c$（通道数）图像，$f\times f\times n_c$ filter，

输出：$(n-f+1)\times(n-f+1)\times n_c'$，$n_c'$ 为用的 filter 个数

1.7 One layer of a convolutional network.

6×6×3 的输入 ($a^{[0]}$) * 3×3×3 的 filter → ReLu(4×4输出 + b_1) → 4×4

* 3×3×3 的第一个 filter → ReLu(4×4输出 + b_1) → 4×4

$W^{[1]}$ $W^{[1]}a^{[1]}$ $a^{[1]}$

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

input : $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

output : $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

$$n_W^{[l]} = \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

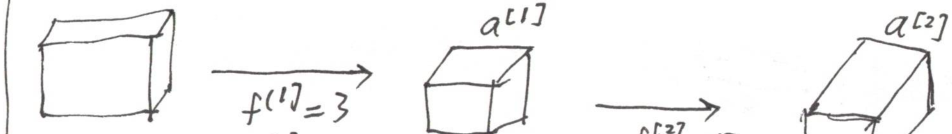
Each filter : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activation : $a^{[l]} \to n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ $A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

weights : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias : $n_c^{[l]}$ — $(1,1,1,n_c^{[l]})$

1.8 A simple convolution network example.



39×39×3, $n_H^{[0]} = n_W^{[0]} = 39$, $n_c^{[0]} = 3$

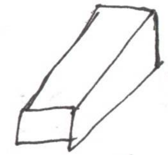
$f^{[1]} = 3$, $s^{[1]} = 1$, $p^{[1]} = 0$, 10 filters

37×37×10 ($a^{[1]}$), $n_H^{[1]} = n_W^{[1]} = 37$, $n_c^{[1]} = 10$

$$37 = \frac{n + 2p - f}{s} + 1 = \frac{39 + 0 - 3}{1} + 1$$

$f^{[2]} = 5$, $s^{[2]} = 2$, $p^{[2]} = 0$, 20 filters

17×17×20 ($a^{[2]}$), $n_H^{[2]} = n_W^{[2]} = 17$, $n_c^{[2]} = 20$



$f^{[3]} = 5$, $s^{[3]} = 2$, 40 filters

7×7×40 → 1960个数字,展开成长向量,填充到 softmax 回归函数中

1960 → logistic softmax → $\hat{y}$

Types of layer in a convolutional network:

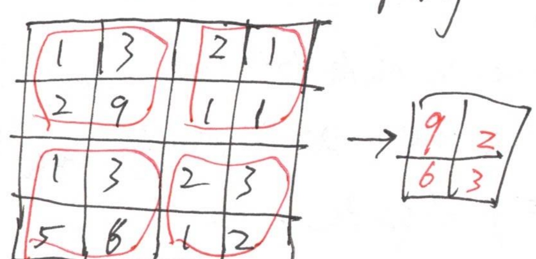
① convolution (CONV)

② pooling (POOL)

③ Fully connected (FC)

1.9 pooling layer

例子 : pooling layer : max pooling



4×4 → 2×2

执行 max pooling 的是4×4输入,把它分成不同区域,输出该区域最大的值.

$$\|f(A)-f(P)\|^2 \le \|f(A)-f(N)\|^2$$

or $\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 \le 0$

但有种情况 $0-0 \le 0$ 满足，

为阻止这情况，

$$\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha \le 0$$

↑ margin

loss func:

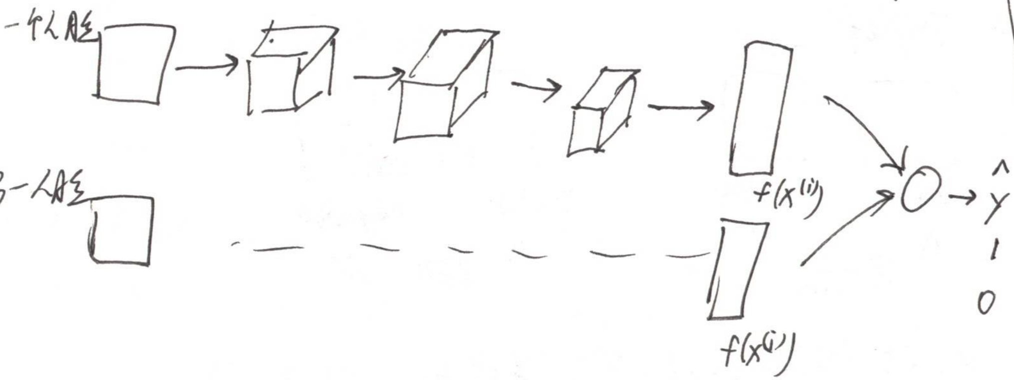
$$L(A,P,N) = \max\left(\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha,\ 0\right)$$

10000 pictures of 1000 persons.

$$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$$

选择难训练的三元组，where $d(A,P) \approx d(A,N)$

4.5 Face verification and binary classification



siamese 网络，→最终的128维输入到逻辑回归单元，用以识别，相同的人为1，否则为0

$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

k代表这个向量中的第k个元素.

其他形式：χ^2 公式：$\dfrac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$

预先算好每个员工的编码，当他刷脸时，用卷积网络计算编码与预存的编码比较，进行识别.

4.6 What is Neural ~~net~~ style transfer?

4.7 What are deep ConvNet learning

浅层先检测边缘，  ，然后 texture，更深更复杂的物体

4.8 Cost func.

给内容 C，风格 S，生成的图片 G.

评判生成图片的好坏：定义 cost func.

$J_{content}(C,G)$，$J_{style}(S,G)$

$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

2.3 Residual Network (ResNets)

skip connection：一层反馈给另外一层（跳层了）

Residual block:

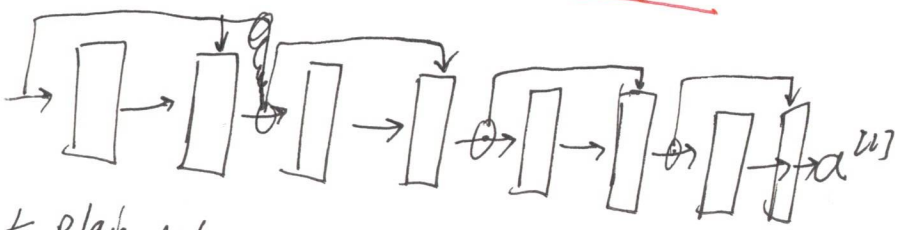
$z^{[l+1]} = w^{[l+1]} a^{[l]} + b^{[l+1]}$, $a^{[l+1]} = g(z^{[l+1]})$, $z^{[l+2]} = w^{[l+2]} a^{[l+1]} + b^{[l+2]}$

$a^{[l+2]} = g(z^{[l+2]})$

$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

in ResNet, $a^{[l]}$ 直接向后，拷贝到网络的深层

用 Residual block 可训练更深的 NN.

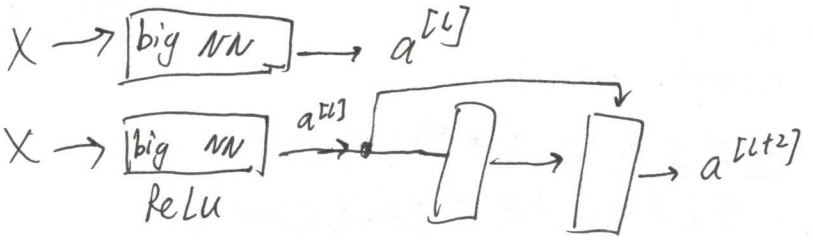


在 plain network 上加上 skip connection 就成 ResNet.

2.4 Why ResNets work?

再去好好理解 Why ResNet work?

回看 code.



$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

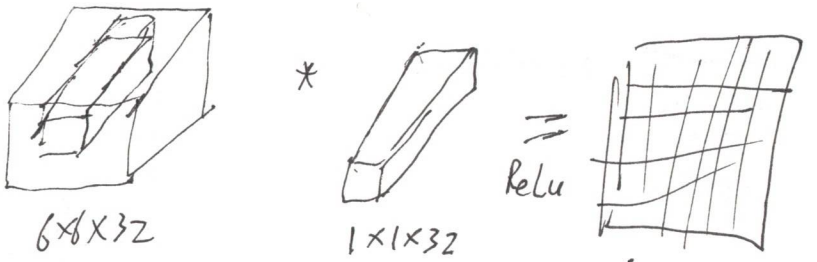
$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$

if $w^{[l+2]} = 0$, $b^{[l+2]} = 0$, $a^{[l]} = g(a^{[l]}) = a^{[l]}$

Relu

⋮

2.5 Network in Network and 1×1 convolutions.



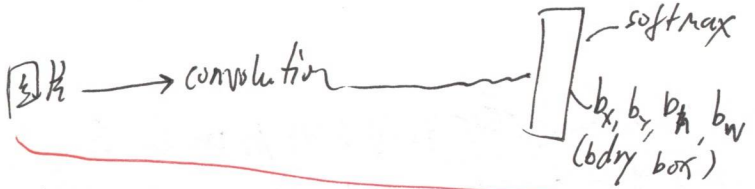
1×1×32 的 32 卷积理解：一个神经元的输入是 32 个数字，即相同在多宽度上某一切片上的 32 个数字，32 个数字有不同的通道，乘以 32 个权重，再用 Relu 非线性 func，输出结果。

这方法称为 1×1 convolution or network in network.

它给 NN 添加一个 nonlinear func，从而（或许）能够缩减输入层中的通道数量。

Week 3 object detection:

3.1 object localization.



Defining target label y

1. pedestrian
2. car
3. motorcycle
4. background

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

p_c → 是否有1~3类, 有则1, 否则为0

loss func:

$$L(\hat{y}, y) = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \cdots + (\hat{y}_8 - y_8)^2$$

3.2 Landmark detection.

3.3 object detection

用滑动窗口的方法. (sliding windows detection)

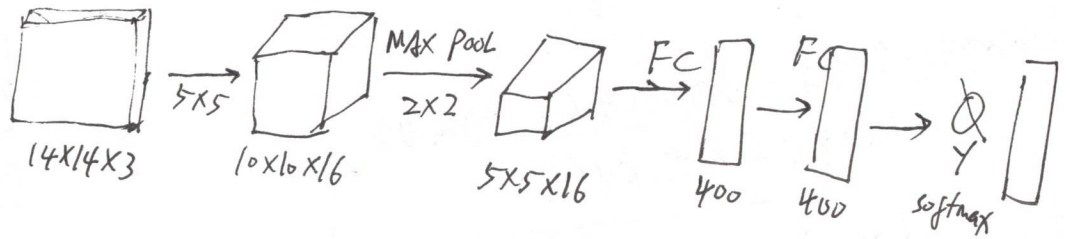
一个小窗口, 从左上角, →, ↓ 滑动

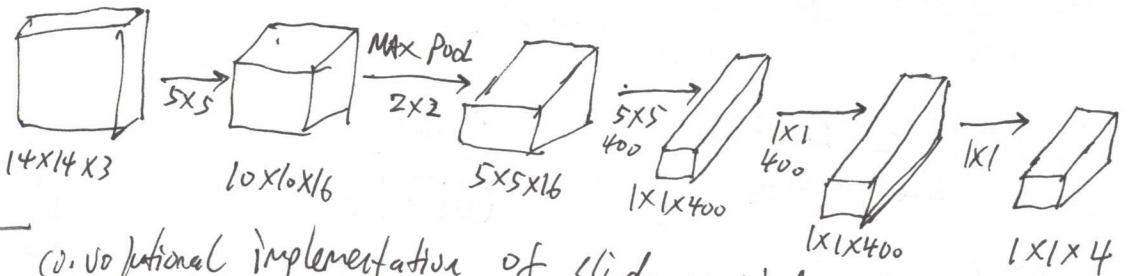
大一点的窗口, ----------.

再大一点. ----------.

3.4 Convolutional implementation of sliding windows

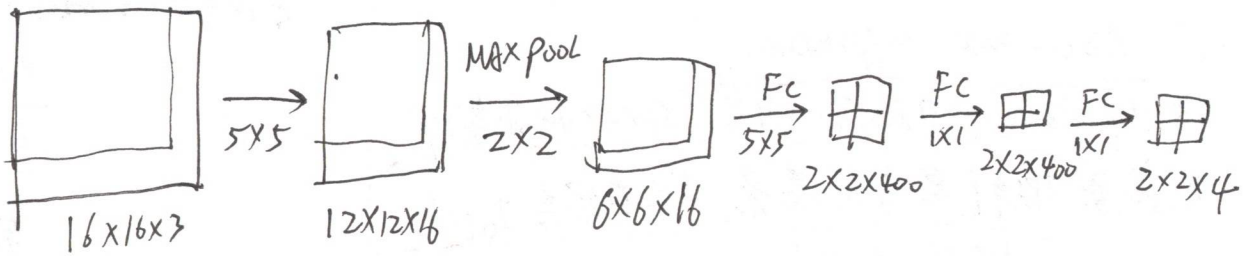
Turning FC layer into convolutional layers





Convolutional implementation of sliding windows

输入图片是16×16×3, 输入NN是14×14×3图片.



3.8 Attention Model

技术原理：

$$a^{<t'>} = (\vec{a}^{<t'>}, \overleftarrow{a}^{<t'>})$$

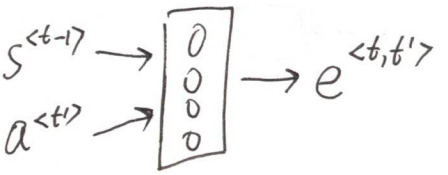
$$\sum_{t'} \alpha^{<1,t'>} = 1$$

$$c^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$$

$\alpha^{<t,t'>}$ = amount of attention ~~should~~ $y^{<t>}$ should pay to $a^{<t'>}$

$$c^{<2>} = \sum_{t'} \alpha^{<2,t'>} a^{<t'>}$$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$



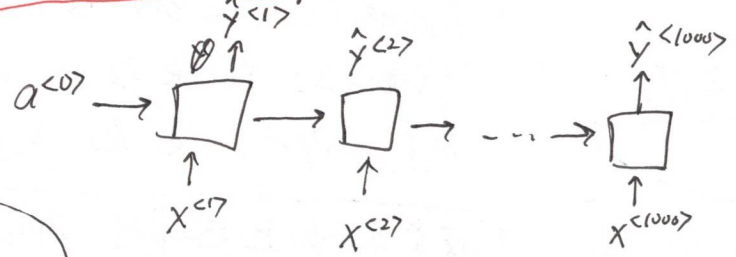
q - Speech recognition

audio 经过处理为 spectrogram.

CTC (connection is temporal Classification)

CTC cost for speech recognition.

"the quick brown fox"



ttt_.h_eee ---(␣)---qqq_ _ → the q

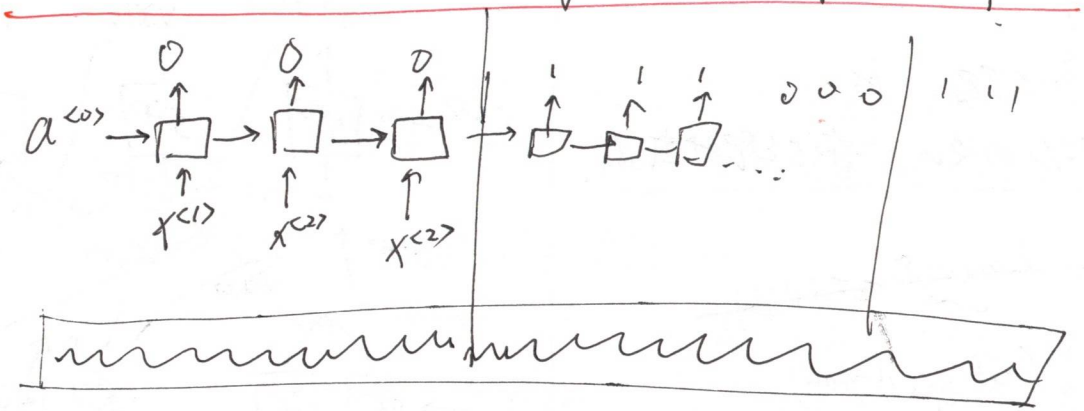
space.

例：10 seconds audio, 100Hz (means 每秒100个样本)，共有1000个输入

CTC cost 的规则：把空白符之间 和 重复的字符 折叠起来。

3.10 Trigger word Detection.

在说话检测到 trigger word 之前 为0，检测到后为1



3.9 Putting it together: YOLO algorithm.

3.10. Region proposals

R-CNN: 带区域的CNN.
不是一点点的sliding window，而是选择某些窗口，
这些窗口基于图像分割算法.

Fast R-CNN

Faster R-CNN

Week 4 Special applications: Face recognition & Neural style transfer.

4.1 What is face recognition.

Face verification vs. face recognition.

verification:
- input image, name, ID
- output whether the input image is that of the claimed person

Recognition:
- Has a database of K persons,
- Get an output image
- Output ID if the image is any of the K persons,

4.2 One-shot learning

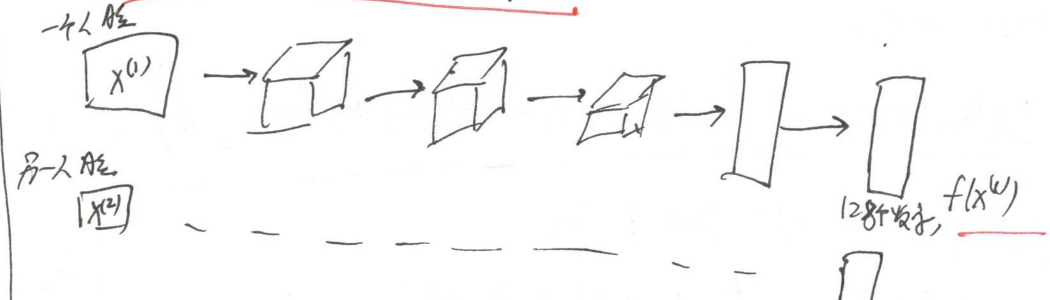
Learning from one example to recognize the person again.

Learning a "similarity" func.

d(img1, img2) = degree of difference between images,

If $d(img1, img2) \le \tau$ "same"
$> \tau$ "different"

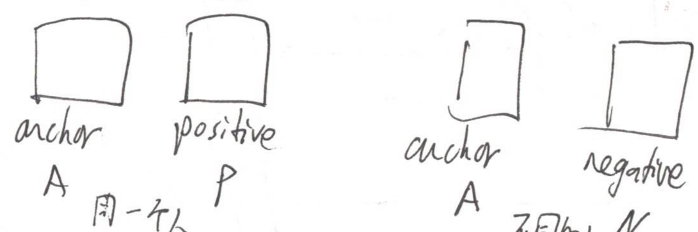
4.3 Siamese network



$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$$

训练Siamese网络的参数，若$x^{(1)}$，$x^{(2)}$是同一人，则两个编码距离就小.

4.4 Triplet Loss



triplet means A, P, N 三张图片.

Find the generated image G

1. Initiate G randomly, G: 100×100×3

2. Use gradient descent to minimize $J(G)$

$$G := G - \frac{\partial}{\partial G} J(G)$$

4.9 Content cost func:

$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

$J_{content}(C,G)$:

use hidden layer l to compute $J_{content}(C,G)$

选L不浅不深，用一个预训练的卷积模型 (VGG or other).

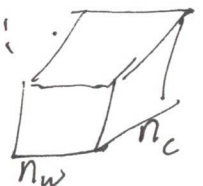
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content.

$$J_{content}(C,G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

4.10 Style cost func.

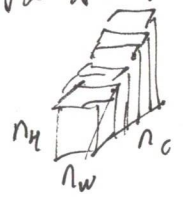
use layer l's activation to measure "style".

定义 activation:



怎么测量不同通道之间 activation 的 correlation?

例如有5个 channels



let $a^{[l]}_{i,j,k}$ = activation at (i,j,k) (H, W, C)

$a^{[l](s)}$ is $n_C^{[l]} \times n_C^{[l]}$, n_C 为通道数

$$G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{i,j,k} \, a^{[l](S)}_{i,j,k'}$$

k, k' 为不同通道 k', k'

$k = 1 \dots n_C$

$$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{i,j,k} \, a^{[l](G)}_{i,j,k'}$$

$$J^{[l]}_{style}(S,G) = \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$

(Forbenius norm)

$$J_{style}(S,G) = \sum_l \lambda^{[l]} J^{[l]}_{style}(S,G)$$

$$J(G) = \alpha J_{content}(S,G) + \beta J_{style}(S,G)$$

4.11 1D and 3D generalizations of models

Course 5, Sequence Models

Week 1 Recurrent NN.

1.1 Why Sequence Models?

1.2 Notation

例如识别名字的例子：

"Harry Potter and Hermione Granger invented a new spell"

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ... $x^{<t>}$ 代表序列中第 t 个位置（输入）

$y^{<t>}$ ---------- （输出）

T_x：输入序列长

T_y：输出 ——

$x^{(i)<t>}$：训练样本 i 中第 t 个元素，$T_x^{(i)}$：第 i 个样本的长度

$y^{(i)<t>}$：- - - - - - - -> $T_y^{(i)}$：- - - - 输出长度

构造一个 1万单元大小，上百万的常用字字典，

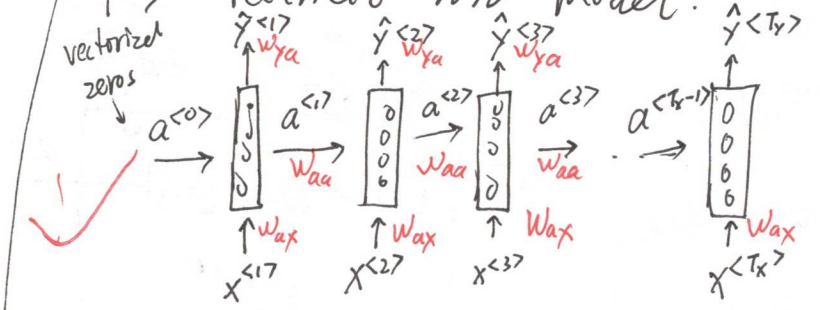
用 one-hot 表示每个单词.

$x^{<1>}$ 是 "Harry"，在字典中的4075位，故 $x^{<1>} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$ ← 4075行

上面每个单词都有各个 one-hot 向量

若字典里没有的词，则由 Unknown word 的标签，<UNK> 表示

1.3 Recurrent NN model.



$x^{<1>}$ 输入一个 hidden layer，输出 $\hat{y}^{<1>}$

$x^{<2>}$ 输入，预测 $\hat{y}^{<2>}$ 时会用上 1 中的信息 $a^{<1>}$.

一个缺点：只使用了序列前面的信息.

$a^{<0>} = \vec{0}$，$\begin{cases} a^{<1>} = g_1(W_{aa}\, a^{<0>} + W_{ax}\, x^{<1>} + b_a) \Leftarrow \tanh, \text{ReLU} \\ \hat{y}^{<1>} = g_2(W_{ay}\, a^{<1>} + b_y) \Leftarrow \text{sigmoid}, \dots \end{cases}$

$$\begin{cases} a^{<t>} = g(W_{aa}\, a^{<t-1>} + W_{ax}\, x^{<t>} + b_a) \\ \hat{y}^{<t>} = g(W_{ya}\, a^{<t>} + b_y) \end{cases}$$

(W_{aa}: (100, 100)，W_{ax}: (100, 10000))

simplified RNN notation:

$$a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

$$W_a = \underset{100}{\left[W_{aa} \mid W_{ax} \right]}, \quad W_{aa}: 100 \times 100, \; W_{ax}: 100 \times 10000, \; W_a: (100, 10100)$$

$$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \begin{matrix} \updownarrow 100 \\ \updownarrow 10000 \end{matrix}$$

$[W_{aa} : W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa} a^{<t-1>} + W_{ax} x^{<t>}$

1.5 Different types of RNN

并不是所有情况都满足 $T_x = T_y$ ← many to many

一个电影评价的例子：

输入一个句子，输出一个分数，many to one.

还有 one-to-one, one-to-many, many-to-many ($T_x \neq T_y$).

1.6 Language model and sequence generation

两句读音相近的话，language model 比较的就是算出某一种话的概率，

Language modeling with an RNN:

例句：Cats average 15 hours of sleep a day. <EOS>

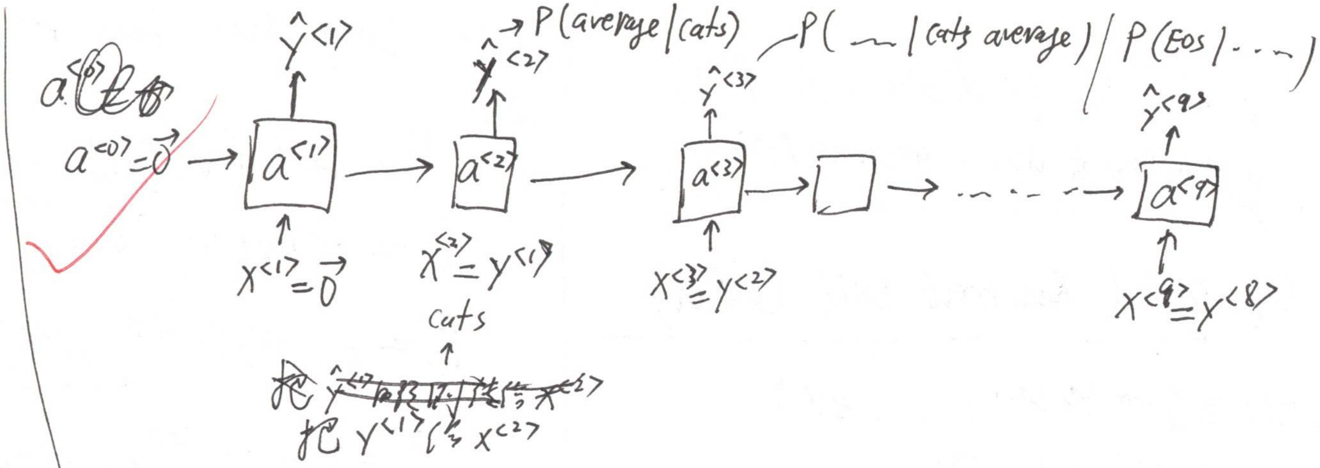
The Egyptian ~~Mau~~ is a bread of cat. <EOS>
<UNK>

加个 <EOS> 记到句子末尾

没有的词用 <UNK> 代替.

~~构造~~ 构造完一个 one-hot，开始构建 RNN.

来构建这些序列的概率模型

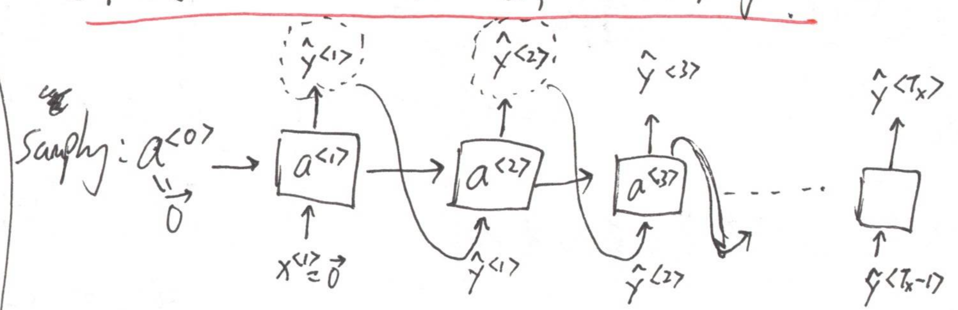


Loss func: $L(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$

总 loss func: $L = \sum_t L^{<t>}(\hat{y}^{<t>}, y^{<t>})$

1.7 Sampling novel sequences.

了解训练的 model 学到了什么，做 sampling.



第一步 对语言模型生成的第一个词采样，输入 $x^{<1>} = 0$, $a^{<0>} = 0$

得到 softmax 计算的概率，根据概率进行随机采样

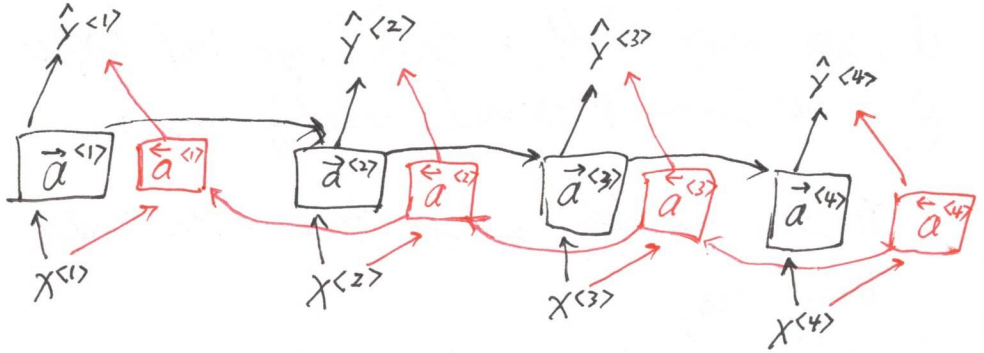
把上一步采样得到的 $\hat{y}^{<1>}$ 输入.

⋮

直到得到一个随机采样的句子.

1.11 Bidirectional RNN

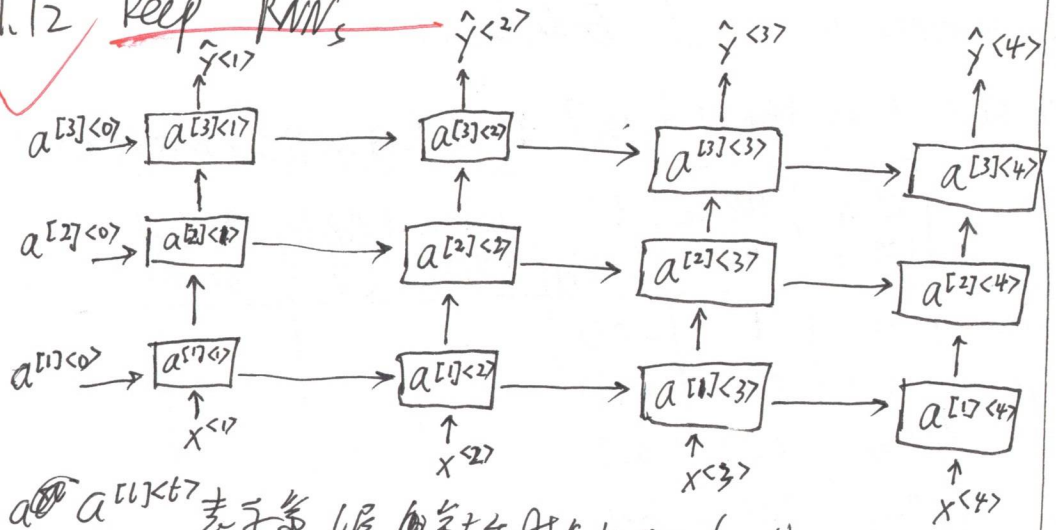
不仅获取之前的信息，还有未来的信息.



预测值 $\hat{y}^{<t>} = g(W_g[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$

基本单元可以是 RNN, GRU, LSTM 等.

1.12 Deep RNNs



$a^{[l]<t>}$ 表示第 l 层，第 t 个时间的 activation unit.

例如: $a^{[2]<3>} = g(W_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$

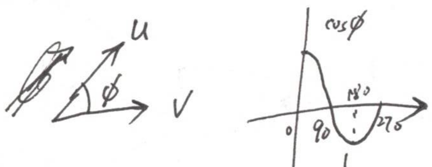
当算法被问及 man 对 woman 相当于 king 对什么时，算法做的就是 $e_{man} - e_{woman}$，找出.

$$e_{man} - e_{woman} \approx e_{king} - e_?$$

找 Find word w: $\arg\max \, sim(e_w, e_{king} - e_{man} + e_{woman})$

余弦相似度：$sim(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2} = \cos\phi$

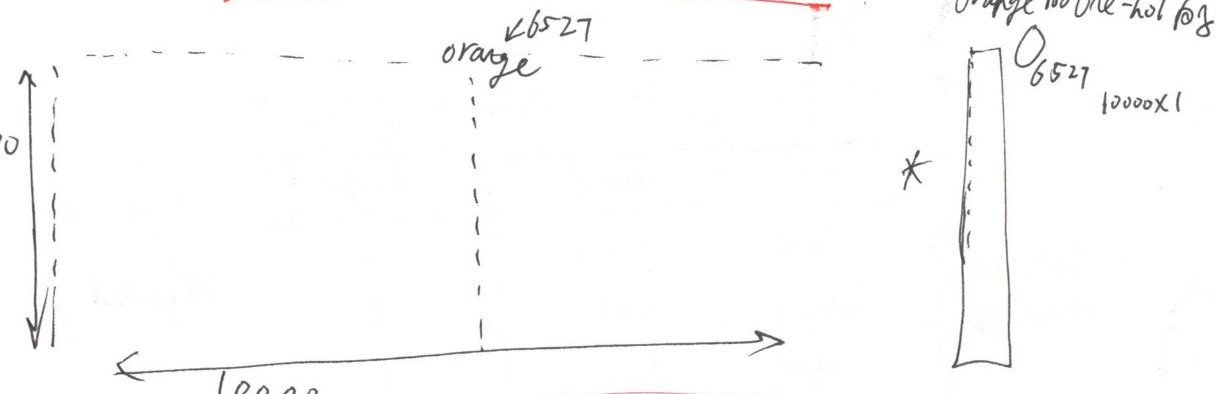
cosine similarity



当 φ 为 0 时，$\cos\phi = 1$，u, v 很相近.

2.4 Embedding matrix

学习一个 embedding matrix E，10000 个单词（加上 <UNK>），每词 300 个特征，so E: 300×10000



$$E * O_{6527} = e_{6527}$$

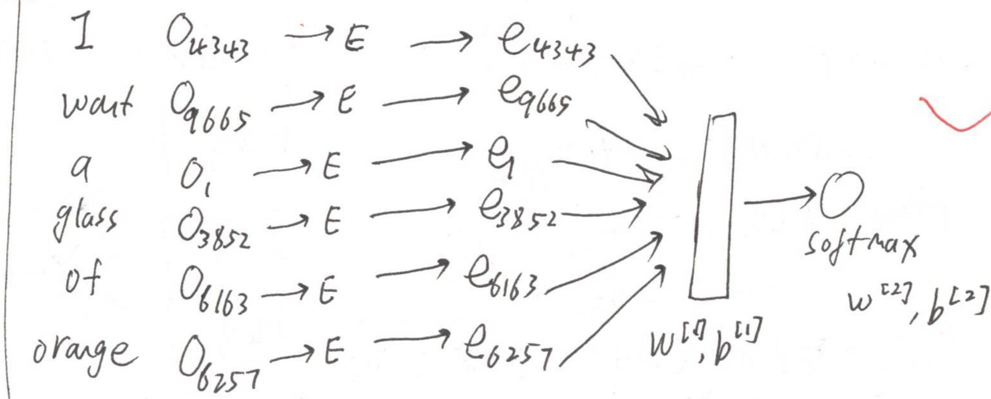
300×10000　10000×1　300×1

任意一单词 w，e_w 代表其嵌入向量，O_i 是只有第 i 个位置是 1 的 one-hot 向量.

2.5 Learning word Embedding

I want a glass of orange.

4343 9665 1 3852 6163 6257



更常见的是固定历史窗口：

例：总是根据前面 4 个单词预测下一个单词

建立语言模型：用目标词的前几个单词作为上下文

学习词嵌入：其他类型上下文

- 4 words on left & right
- Last 1 word
- nearby 1 word

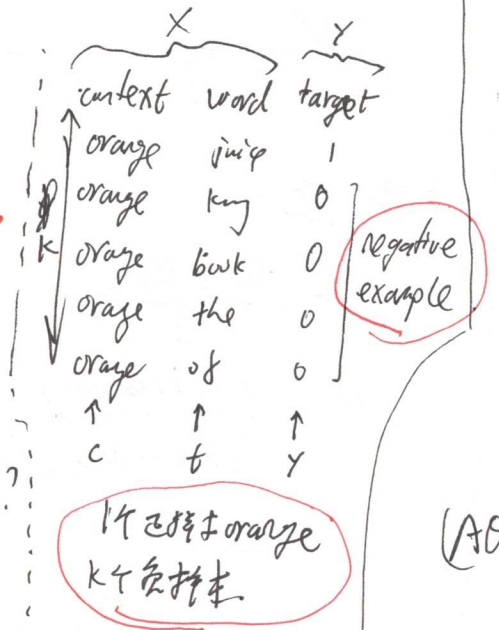
small data set, $k = 5 \sim 20$

large data set: $k = 2 \sim 5$

Model:

Softmax: $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$

$P(y=1|c,t) = \sigma(\theta_t^T e_c)$



Orange
6257

$O_{6257} \rightarrow E \rightarrow e_{6257} \rightarrow$ o juice?
o king

softmax转成二分类问题.

如何选负样本?

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=1}^{10000} f(w_j)^{\frac{3}{4}}}$$

$f(w_i)$ 是观测到的在语料库中等于英文词的词频.

2.8 GloVe Word Vectors

(global vectors for word representation)

X_{ij}: 单词 i 在单词 j 的 context 中出现的次数

对于 GloVe, 定义 context & target 为任意两个位置相近的词.

GloVe 就是将他们之间差的最小化:

$$\text{minimize} \sum_{i=1}^{10000} \sum_{j=1}^{10000} f(X_{ij}) (\theta_i^T e_j + b_i + b_j' - \log X_{ij})^2$$

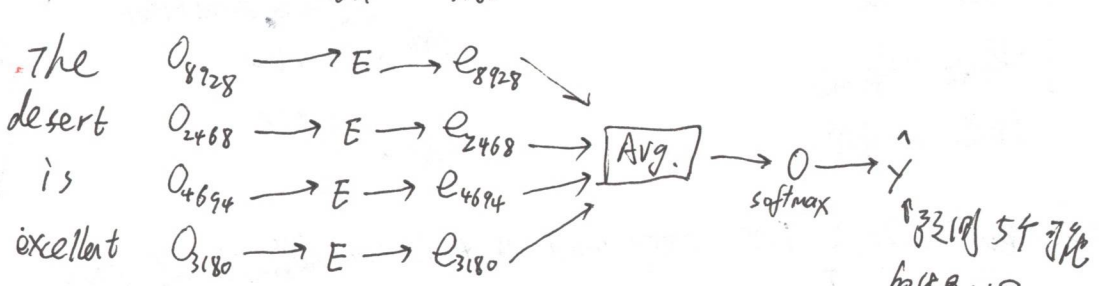
当 $X_{ij} = 0$, $\log X_{ij}$ 负无穷, 故有一加权项 $f(X_{ij})$, $f(0) = 0$

$f(X_{ij})$ 还要对 of, the 这些高频无意义词的权重, 对 durian 这种不常见词的权重考虑.

$$(A\theta_i)^T (A^{-T} e_j) = \theta_i^T A^T A^{-T} e_j = \theta_i^T e_j$$

2.9 Sentiment Classification

"The desert is excellent"
8928 2468 4694 3180



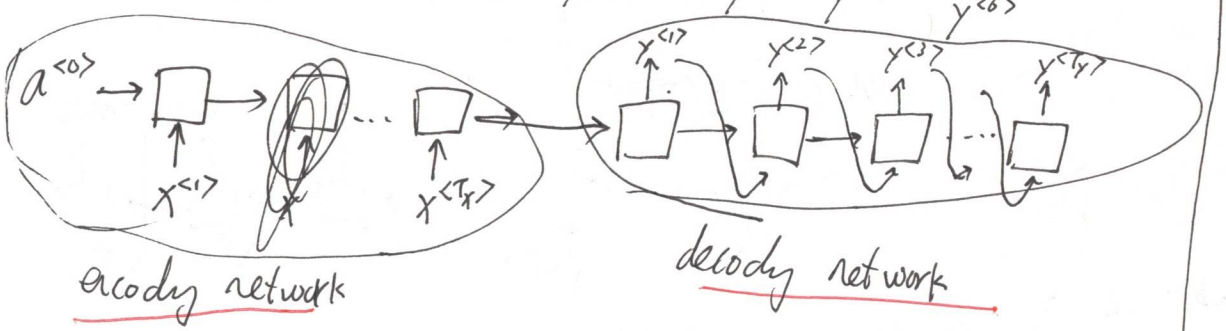
缺点: 未考虑词序.

Week 3 · Sequence Models & Attention mechanism

1.1 Various sequence to sequence architectures

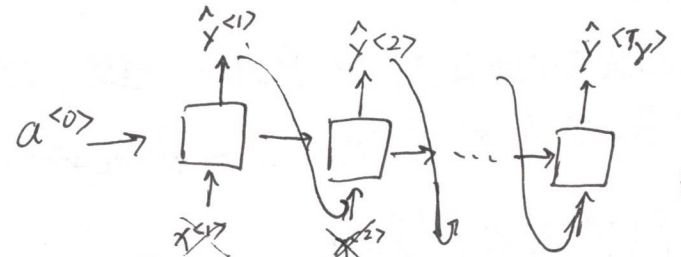
输入法语：Jane visite l'Afrique en septembre

输出英语：Jan is visity Africa in seppember.



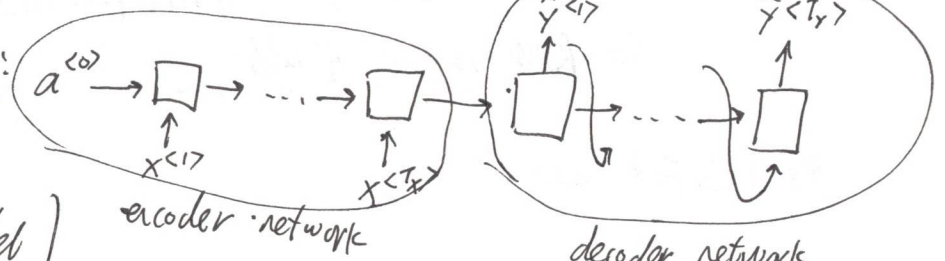
3.2 Picking the most likely sentence

language model:



machine translation:

(conditional language model)



$$\arg\max_{y^{<1>}\cdots y^{<T_y>}} P(y^{<1>},\cdots,y^{<T_y>}|x)$$

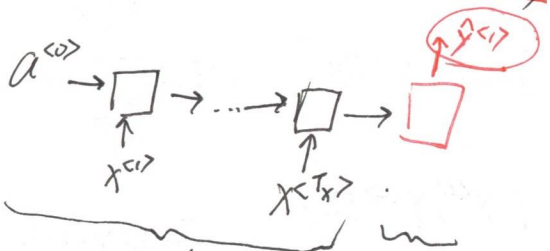
用beam search找出令这个值，使$P(y|x)$最大

Greedy search是一个词一个词搜索的，这里更要考虑整个句子的概率

3.3 Beam search

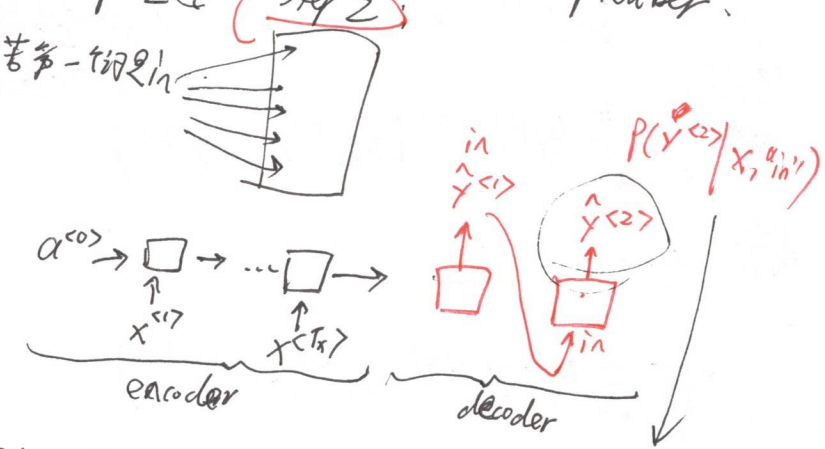
Beam search algorithm

Beam width = 3, $P(y^{<1>}|x)$



beam width = 3 means 第一个单词有3个可能的结果的概率

选择出 in, Jane, september.



$$P(y^{<1>}, y^{<2>}|x) = P(y^{<1>}|x)\, P(y^{<2>}|x, y^{<1>})$$

3.8 Bleu Score

Bilingual Evaluation understudy

对于多个很不错的翻译，BLEU给出评分；

一个极端的例子：

机翻译输出：the the the the the the the.

衡量翻译质量：检查每一输出结果是否出现在参考树中.

7个the都出现了，故 $precision = \frac{7}{7}$

显然不好，改进：

参考1，the出现2次，参考2，the出现1次

故上限为2

$precision = \frac{2}{7}$

母7为单词the总出现次数

分子是单词the出现的次数，但在达上限时截断

~~BLEU score on bigrams (二元词组)~~:

$$P_n = \frac{\sum_{n\text{-gram} \in \hat{y}} Count_{clip}(n\text{-gram})}{\sum_{n\text{-gram} \in \hat{y}} Count(n\text{-gram})}$$

BLEU得分：$\exp\left(\frac{1}{4}\sum_{n=1}^{4} P_n\right)$

Evaluating machine translation:

French: Le chat est sur le tapis

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

MT output: the the the the the the the

precision: $\frac{7}{7}$

modified precision: $\frac{2}{7}$

通常前面加上惩罚项 Brevity penalty, BP, $BP \cdot \exp\left(\frac{1}{4}\sum_{n=1}^{4} P_n\right)$

$$BP = \begin{cases} 1, & \text{if } MT\_output\_length > reference\_output\_length \\ \exp(1 - MT\_output\_length / reference\_output\_length), & \text{otherwise} \end{cases}$$

3.7 Attention Model Intuition

对于长句子，encoder-decoder做的不好，Belu score 低.

Attention model intuition:

$\alpha^{<1,1>}$ tells how much attention should pay to that unit.