# 提纲

1 神经网络构建

1.1 神经网络基础

逻辑回归模型，Softmax，矩阵形式表示

1.2 神经网络的前向传递和反向传播的数学推导

线性，激活函数，损失函数，需要注意的是数据表示，维度问题

2 改善神经网络

2.1 防止过拟合　提前结束，　增加训练样本，　正则项，　Dropout

2.2 梯度消失问题　初始化，梯度剪辑，激活函数，Bitch-Norm，Res-net，LSTM

2.3 优化方法（加速 学习率）　Mini-bitch，动量的梯度下降，RMS-prop，Adam

2.4 梯度检验，调参(网络结构:层,神经元个数;学习率:$\alpha$;优化方法:$\beta_1,\beta_2,\varepsilon$;正则化项:$\lambda$; Dropout; Mini-bitch size; 迭代次数; 初始化方式)

3 结构化机器学习

3.1 快速搭建好第一个系统：框架使用，开源使用，　迁移学习，数据预处理

3.2 误差分析：偏差和方差分析，正交化调试，控制变量法调试

1 偏差问题，调整网络架构，学习率，优化方法，增加数据集

2 方差问题（过拟合），正则化，Dropout，增加训练集

3 真实数据场景差 真实数据与训练测试数据分布不一致

3.3 单一评价标准，如 F1 值结合了精度和召回率

4 卷积神经网络

4.1 卷积，池化，填充，全连接概念以及超参数和参数（参数共享，特征提取思想）

4.2 CNN 构建的流程以及数学表示形式，需要注意的是，数据表示，维度问题

1 卷积，Relu，池化，向量化，全连接，Softmax 层，前向传播

2 卷积，池化，全连接，Softmax 层，链式求导

4.3 经典的 CNN 架构

LeNet-5，AlexNet，VGG（卷积核大小统一化），Resnet（解决梯度消失问题），Inception（可学习卷积核大小），迁移学习构建初始网络架构

5 序列模型

5.1 序列模型架构：many-to-many（实体识别，机器翻译），　many-to-one（情感分类），one-to-many（语音生成），one-to-one

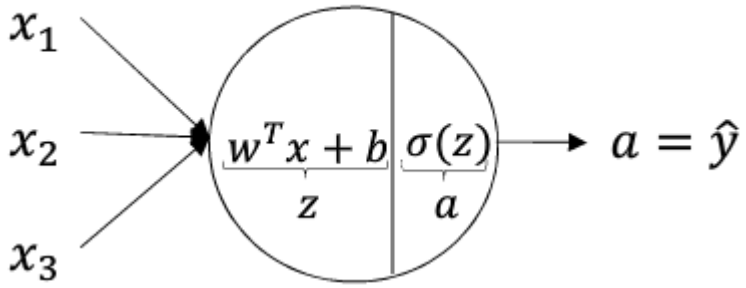
5.2 网络架构：RNN，GRU，LSTM，B-RNN，D-RNN 前向反向数学推导，需要注意的是数据表示，维度问题

5.3 Embeding：上下文关系学习词向量（*），skipgram，负采用，Glove 词向量，去偏见，注意力机制（上下文加权）

# 神经网络构建

## 一、逻辑回归中的线性函数与激活函数

一个典型的逻辑回归可以看作是神经网络中,上一层的每个神经元的线性函数与当前神经元激活函数构成（忽略维度）。

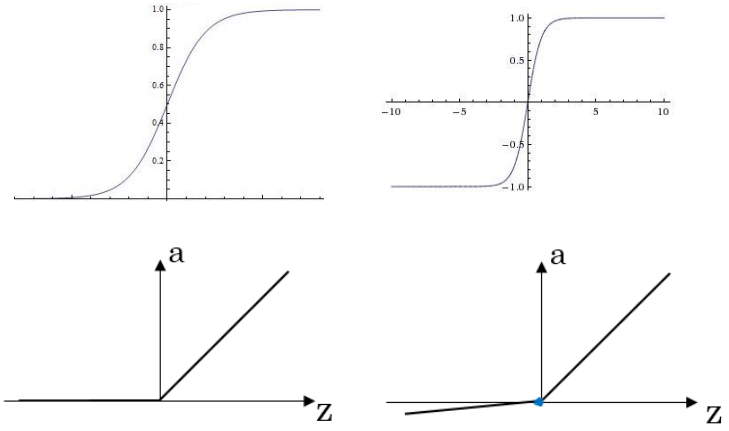



$$z = w^T x + b$$

$$a = \sigma(z)$$

线性函数为简单 $Z = W^T X$，这里需要注意的是传统机器学习中样本 X 的矩阵形式按行为一个样本，而深度学习中按列为一个样本。

激活函数一般采用非线性激活函数，如果采用线性函数是没有意义的。常用的激活函数有 sigmoid，tanh，relu，leaky relu 函数，下面是几个激活函数图，在不同网络层选择合适的激活函数也是调节神经网络的重要参数（超参）。




$$sigmoid(z) = \frac{1}{1 + e^{-z}} \qquad \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$relu(z) = \max(0, z) \quad leaky\ relu(z) = \max(0, z) = \begin{cases} \alpha z & z < 0 \\ z & z > 0 \end{cases}$$

这里需要关心的是线性函数与激活函数本身以及其导数，因为前向传播就是线性函数与激活函数的计算，反向传播就是反向求损失函数，激活函数与线性函数的导数。下面以一个简单的逻辑回归为例，损失函数为对数损失，逻辑回归的对数损失解释可以最大似然推导出来。

$$\begin{cases} Z = W^T X \\ y = \sigma(Z) \end{cases}$$

$$\begin{aligned} loss &= \begin{cases} -\log(yhat) & y = 1 \\ -\log(1 - yhat) & y = 0 \end{cases} \\ &\Leftrightarrow -\left(y \log(yhat) + (1 - y)\log(1 - yhat)\right) \end{aligned}$$

损失函数的导数为

$$-\left(y \frac{1}{yhat} - (1 - y) \frac{1}{1 - yhat}\right)$$

Sigmoid 函数导数：

$$\begin{aligned} sigmoid(z) &= \frac{1}{1 + e^{-z}} \\ dz &= \frac{e^{-z} + 1}{(1 + e^{-z})^2} - \frac{1}{(1 + e^{-z})^2} = \frac{1}{(1 + e^{-z})} - \left(\frac{1}{(1 + e^{-z})}\right)^2 \\ &= s(1 - s) \quad , s = sigmoid(z) \end{aligned}$$

Tanh 函数导数：

$$\begin{aligned} \tanh(z) &= \frac{e^z - e^{-z}}{e^z + e^{-z}} \\ dz &= \frac{\left(e^z + e^{-z}\right)^2 + \left(e^z - e^{-z}\right)^2}{\left(e^z + e^{-z}\right)^2} \\ &= 1 - s^2 \quad , s = \tanh(z) \end{aligned}$$

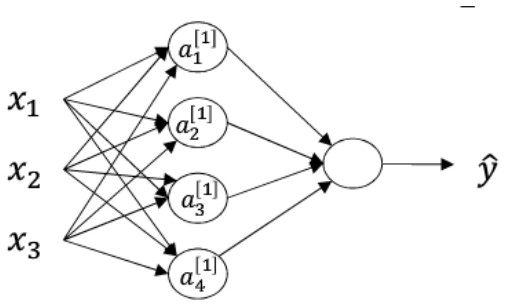
Relu 函数导数：

$$\begin{aligned} relu(z) &= \max(0, z) \\ dz &= \begin{cases} 1 & , z > 0 \\ 0 & , z \le 0 \end{cases} \end{aligned}$$

2 神经网络结构与前向反向　　传导

逻辑回归可以看做是一个没有隐藏层的神经网络，如果忽略输入特征，那么就只有输出层一个神经元。而神经网络是一个多层，每一层有多个神经元的网络结构。如果说逻辑回归是一个线性函数与激活函数的复合函数，那么神经网络就是多个这样的神经元的复合函数。在计算过程中，与逻辑回归不同的地方是前向传导需要计算 L 次，同样反向传播也是**损失函数**对参数求导计算 L 次。同时为了计算方便，后期在神经网络计算中统一采用矩阵形式。

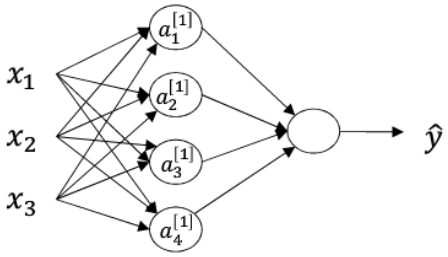
前向传导：




$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

每个神经元与上一层神经元的函数关系，都是一个线性函数与激活函数的复合函数。

矩阵表示形式如下：




Given input x:

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

矩阵表示每一层之间的关系，也是前向传导的计算公式,所以每一层前向传导过程可以分为两步，先对线性函数进行计算，然后对激活函数进行计算，上一层的激活值作为下一层的输入特征。

这里特别注意的是矩阵表示时，X，Y，W，b，Z，A 的矩阵行列,其中 X,Y,W,b 是数据初始化的，Z，A 是计算过程中产生的，最为常见的错误就是矩阵大小不匹配。令 $A^0$=X，那么可以看作第一层神经元个数为特征维度，那么有以上矩阵的大小为：

$$X = (n^0, m)$$
$$Y = (1, m)$$
$$W^l = (n^l, n^{l-1})$$
$$b^l = (n^l, 1)$$
$$Z^l = (n^l, m)$$
$$A^l = (n^l, m)$$

反向传播：

反向传播算法的核心就是一个链式求导，**损失函数**对每一层网络神经元中的参数求偏导，而且神经网络具有表达的形式循环性，使得可以使用循环来进行链式求导，当采用矩阵形式表示时，每一层求导都在一次 for 循环中完成。下面求两层网络的反向求导过程。

损失函数同样采用对数损失，输出层采用 sigmoid，隐藏层采用 relu 激活函数，同样这里采用矩阵形式表示。

损失函数对预测值 AL 求偏导：

$$loss = \begin{cases} -\log(AL) & y = 1 \\ -\log(1 - AL) & y = 0 \end{cases}$$
$$\Leftrightarrow -\left(Y \log(AL) + (1 - Y)\log(1 - AL)\right)$$

$$\frac{\partial \cos t}{\partial AL} = -\left(Y \frac{1}{AL} - (1 - Y)\frac{1}{1 - AL}\right) = \frac{y - AL}{AL(1 - AL)}$$

可见 AL，Y 都是一个行向量。

激活函数求导：

$$\frac{\partial AL}{\partial Z} = s(1 - s), s = \frac{1}{1 + e^{-z}} = AL$$

$$\frac{\partial AL}{\partial Z} = \begin{cases} 1 & Z > 0 \\ 0 & Z \le 0 \end{cases}$$

$$\frac{\partial AL}{\partial Z} = 1 - s^2, s = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

线性函数求导：

$$dZ^l = \begin{bmatrix} | & | & | & | \\ | & | & | & | \\ | & | & | & | \\ | & | & | & | \end{bmatrix}_{(n^l, n)} \qquad A^{l-1^T} = \begin{bmatrix} - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix}_{(n, n^{l-1})}$$

可见 dZ 与 $A^{l-1T}$ 乘积是 n 个样本在一个维度上的反映，所以 Z 对 W 求导前面有 1/n，而 b 则是 dZ 按行求和乘 1/n。

$$\frac{\partial Z}{\partial W^l} = \frac{1}{n} A^{l-1}$$

$$\frac{\partial Z}{\partial b^l} = \frac{1}{n}$$

除输出层后每层网络重复求如下偏导：

$$\frac{\partial \cos t}{\partial Z^l} = \frac{\partial \cos t}{\partial AL} \cdot \frac{\partial AL}{\partial Z^l} = -\left(\frac{Y}{AL} - \frac{1-Y}{1-AL}\right) \cdot AL(1-AL) = AL - Y = dZ$$

$$\frac{\partial \cos t}{\partial W^l} = \frac{\partial \cos t}{\partial AL} \cdot \frac{\partial AL}{\partial Z^l} \cdot \frac{\partial Z^l}{\partial W^l} = dZA^{l-1^T}$$

$$\frac{\partial \cos t}{\partial b^l} = \frac{\partial \cos t}{\partial AL} \cdot \frac{\partial AL}{\partial Z^l} \cdot \frac{\partial Z^l}{\partial b^l} = dZ$$

$$\frac{\partial \cos t}{\partial A^{l-1}} = \frac{\partial \cos t}{\partial AL} \cdot \frac{\partial AL}{\partial Z^l} \cdot \frac{\partial Z^l}{\partial A^{l-1}} = W^T dZ$$

所以每一层神经网络的反向求导都可以分成两步来做，第一步是对激活函数求导，第二步是对线性函数求导，当然输出层 AL 的导数需要单独根据损失函数定义来求导，而且每一层循环求导 $A^{l-1}$ 也只是为了联系上一层网络，所以直到输入层时，对 $A_0$（X）求导可有可无。

***这里有个重点，采用链式求导是最直观的理解，但是很多时候我们并不需要分别计算 A，Z 的导数，dAdZ 可以一步完成（当然与激活函数有关），加快计算效率，这里 dAdZ 简称残差。对于求导公式不必死记，只需要记住根据复合函数，进行链式求导。

反向传播求导的目的是为了实现梯度下降算法，即往负梯度方向求解最小值。所以参数更新表达式如下：

$$W^{k+1} := W^k - \alpha \frac{\partial \cos t}{\partial W}$$

$$b^{k+1} := b^k - \alpha \frac{\partial \cos t}{\partial b}$$

这里需要指出的是 W，b 是指整个网络中所有的 W，b。

3 加载数据

以 deeplearning.ai 教程中提供的数据集为例。

每一张图片数据存储都是 64*64*3 的格式，所以整个训练集是一个 209*64*64*3 的矩阵。所以在加载数据之后，需要对原始的数据进行生成列为样本，行为多个样本的形式。将每一张 64*64*3 图片拉成一个列向量。

4 网络初始化

神经网络中超参有，网路层数，每层神经元个数，激活函数的选择，学习速率，迭代次数，参数有 W，b。

参数的初始化依赖于超参的定义， 定义好网络层数与每一层神经元个数，每一层统一使用的激活函数。可以对 W，b 进行初始化。

5 算法设计流程

1 加载数据，数据预处理

2 参数初始化

3 梯度下降

1）前向传导

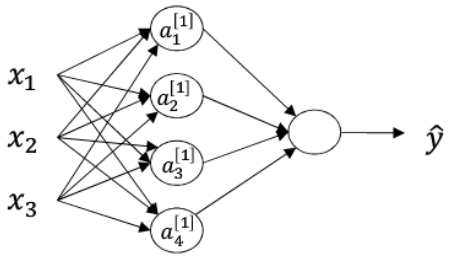
2）计算损失

3）反向传播

4）更新参数

4 预测

数据结构




Given input x:

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

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

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

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

前向传导:（X=A0,(A,W,b,Z)作为数据链表中的一个整体）

Z=WA+b A->next=sgn(Z)

反向传导：（（W,b,Z,A）作为数据链表中的一个整体)

A=sgn(Z),Z=WA->pre+b

# 第二课优化神经网络

## 1．参数初始化

参数的初始化对深度学习作用也很大，最简单的初始化如:初始化为 0，随机初始化，一般初始化为 0 是无法学习的，随机初始化对算法的收敛速度和收敛结果都影响，甚至导致算法的不收敛。

对于参数的初始化，有不少这方面的研究，主要解决的问题是为了**防止**梯度消失和梯度爆炸，虽然不能完全解决，但效果。核心点（假设）

“为了使得网络中信息更好的流动，每一层输出的方差应该尽量相等。”

基于上面的假设，推导出几种不同初始化方法

He

$$\sqrt{\frac{2}{layer\_dim[l-1]}}$$

arxiv

$$\sqrt{\frac{1}{layer\_dim[l-1]}}$$

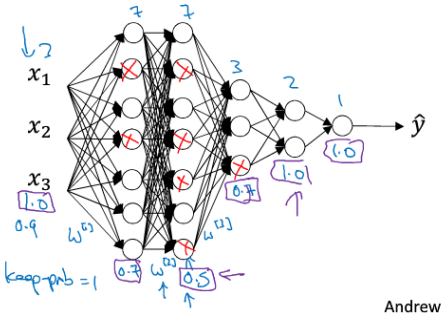
hin

$$\sqrt{\frac{2}{layer\_dim[l-1]+layer\_dim[l]}}$$

## 2．方差/偏差问题

防止过拟合是机器学习中很关键的一个问题，深度学习中更是如此，所以在传统的机器学习方法中防止过拟合的方法在深度学习中同样适用(范数正则项)，由于深度学习特别热，有其独有的防止过拟合的方-Dropout,这里特别讲解 Dropout 技术

个人认为 Dropout 技术关键是每次训练时随机的丢失一些特征/样本防止过拟合。




实现：对每一层的神经单元乘上一个随机矩阵

即随机生成一个 0-1 矩阵 B，每一层的神经单元（矩阵 A）与矩阵 B 进行对应相乘，这样由于随机矩阵 B 每行每列都是随机，所以造成的结果是随机丢失一些特征维度和样本,（矩阵 A 的行看作神经元个数,即特征维度,列数位样本数）。

$$B = \begin{matrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{matrix}$$

另外，增加训练样本，提早结束优化也不失为一种办法。

3．梯度检验

梯度检验是为了检验在设计算法（前向传播和反向传播）时的问题。检验的思想是实际梯度与算法计算的梯度应该非常接近,实际梯度的计算则是采用极限的思想对函数在 e 的邻域内求左右导数来作为函数在该点的近似实际梯度。定义一个如下表达式来判断是否接近。

4． Mini-batch 梯度下降

Mini-batch 梯度下降是介于全梯度下降和随机梯度下降算法中间的一种方法。传统的梯度下降算法每一次求解梯度是遍历所有的样本来确定梯度方向（梯度精准，代价高），随机梯度下降算法每一次求解梯度之看一个样本来确定梯度方向（梯度随意，代价小），而 min-batch 则可以缓解两种方式的极端，每一次根据一个 batch 大小的样本来确定梯度方法,也就是说将所有样本划分为统一大小（batch-size）的 batch，每一次迭代都包括遍历所有的 batch，每遍历一个 batch 就进行一次参数更新。

5．优化算法

5.1 指数加权平均

指数加权平均是一种根据以往数据的平均作为预测后面数据的一个参考，指数加权平均也是一种常用的线性回归模型。用在优化算法中，是通过前 N-1 次的梯度作为第 N 次梯度的一个参考。指数的意思每一次的数据对后面数据的参考价值是指数递减的。

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

其中$V_t$表示第 t 次的加权平均值，$\theta_t$表示第 t 次计算得到的梯度，而我们实际上更新参数是根据$V_t$，也就是说第 t 次更新的梯度是出于前 t 次加权平均值以及当前计算的梯度。如果将上式展开成下式：

$$V_{100} = (1-\beta)\theta_{100} + (1-\beta) * \beta\theta_{99} + (1-\beta) * \beta^2\theta_{98} + \ldots + (1-\beta) * \beta^{100-1}\theta_1$$

可以看出上面递推公式实际上式一个指数加权平均。一般来讲$\beta = 0.9, 0.9^{10} \approx 0.35$，看作第 N-10 次计算的梯度对当前加权梯度的影响为 0.35。

5.2 带修正指数加权

上式中我们可以发现如果从$V_0 = 0$开始递推，则有$V_1 = (1-\beta)\theta_1$，这样得到的并非实际的加权平均值，所以一般需要系数加和为 1 修正，即对等式右边的所有系数进行加和为 1 修正，所以带修正指数加权表达式如下：

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

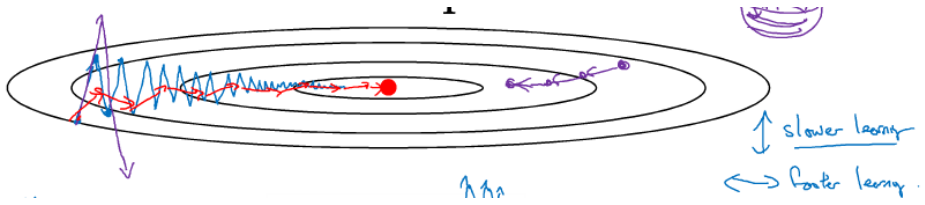
上式的分母其实就是修正系数加的作用。直观理解，如果不加修正，那么在一定的学习速率下，随着迭代次数的增加后面计算的加权平均系数一直在增加，如果不加修正，就会使得越到后面加权梯度越大（或者说前面几次梯度过小），当然直接增加学习速率会解决预热问题，但是对后面的迭代就不利。

值得注意的时

修正的指数加权平均与指数加权平均一定要分开计算，不能将修正后的指数加权值放到加权指数平均值中。

5.3 动量梯度下降

动量梯度下降使用指数加权平均的特点，很好的解决了梯度下降中无法使用更大的学习速率问题。其直观的解释如图

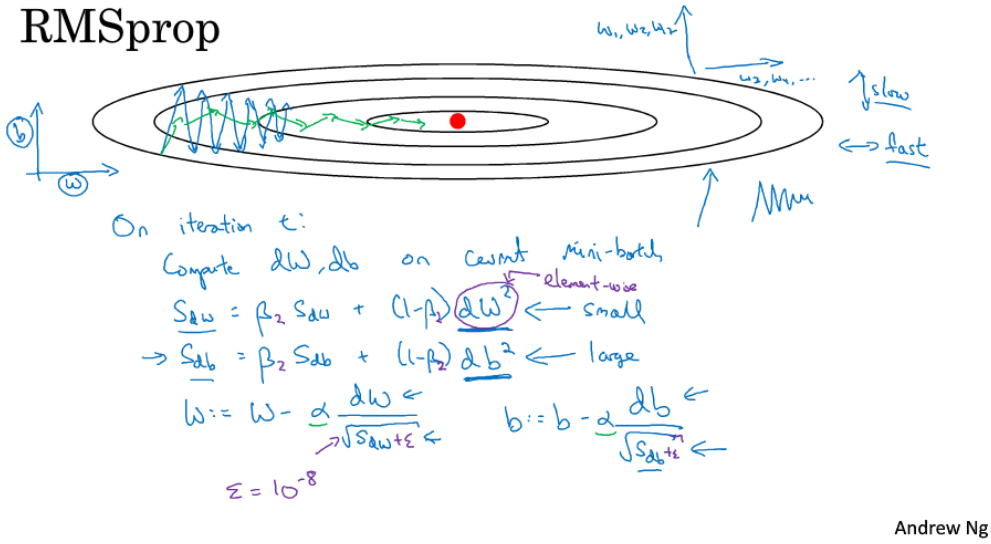



$$V_{dW} = \beta V_{dW} + (1-\beta)dW$$
$$V_{db} = \beta V_{db} + (1-\beta)db$$

$$W = W - \alpha V_{dW}, b = b - \alpha V_{db}$$

RMS

5.4RMS（root mean square）同样是解决梯度下降中无法使用大的学习速率问题。其直观解释如图。




$$S_{dW} = \beta S_{dW} + (1-\beta)dW^2$$
$$S_{db} = \beta S_{db} + (1-\beta)db^2$$
$$W = W - \alpha \frac{dW}{\sqrt{S_{dW}+\varepsilon}}, b = b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$$

同样，RSM 中计算指数加权平均值梯度，但 RSM 不是考虑平均值作为更新梯度，而是通过加权平均梯度得到梯度的一个高维特点，采用“归一化”的方式将椭圆形梯度变成圆形梯度，这样所有维度的梯度不会相差太大，方便采用统一的学习速率，这样其实规避了一开始对原始数据的归一化来加速优化的盲目性。

5.5Adam

Adam 是结合了动量梯度下降和 RMS，动量梯度下降中通过前 N-1 梯度正负平衡来很好的克服梯度上下摆动的问题，RSM 则是通过前 N-1 次梯度的大小来确定梯度在每一个维度的特点，对每一个维度上的梯度进行“归一化”来缓解大梯度与小梯度之前的差距，使得迭代过程中能采用大学习速率。

$$V_{dW} = \beta_1 V_{dW} + (1-\beta_1)dW, V_{db} = \beta_1 V_{db} + (1-\beta_1)db$$
$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2, S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$$
$$W = W - \alpha \frac{V_{dW}}{\sqrt{S_{dW}+\varepsilon}}, b = b - \alpha \frac{V_{db}}{\sqrt{S_{db}+\varepsilon}}$$

6．学习率衰减

随着迭代不断的收敛，学习速率应该不断减小，常用的一种方式是指数衰减

$$\alpha = \frac{1}{1 + rate^{\#iter}}\alpha_0 \text{或者} \alpha = (0.95)^{\#iter}$$

值得注意的是指数衰减不是基于上一次的学习率，而是基于一个固定的衰减率

7. 参数调试

网络结构 网络层数，神经单元个数，激活函数

学习率

优化算法 GD，MGD，Adam

Mini-batch-size

初始化

正则项 范数，Dropout

调参一定要目标明确（单一目评价指标），有针对性（正交化）。

8. 正则化网络激活 Batch-normalizing

前面的对数据归一化以及 RPS 优化算法都是为了加速梯度下降，一个是对原始数据进行归一化，一个是对梯度进行归一化，而数据的归一化是一次性的，即只对输入层归一化，隐藏层无法做到，而梯度归一化是对所有层的所有梯度。而 Batch-Norm 就针对每一层都进行归一化，对每一层的 Z 进行归一化。

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

$$\sigma = \frac{1}{m}\sum_i \left(Z^{(i)} - \mu\right)^2$$

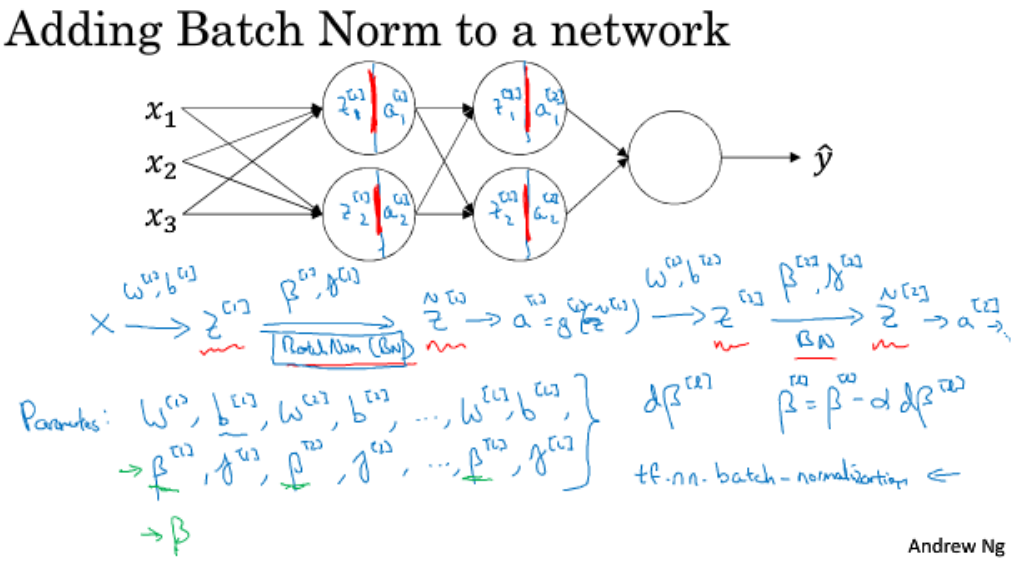
$$Z_{norm}^{(i)} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{Z}^{(i)} = \gamma Z_{norm}^{(i)} + \beta$$

初始化

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

$$\beta = \mu$$




由于在进行归一化 Z 时，需要去均值，所以线性函数的偏置 b 是无效的。对于 Mini-batch 优化，每次更新是在一个 batch 上进行的，所以随机噪声（均值，方差）较大。测试时每一次的归一化采用的均值和方差不是基于测试集，而是基于训练集的，但每一个 batch 的均值和方差随机性较大，所以一般采用指数加权平均值来作为测试时对每一层进行归一化。

9. SoftMax 回归

与 Softmax 回归相对的是 HardMax，即最大置 1 其他置 0。Softmax 与 Logistic 回归很像，广义的 logistic 回归也可以处理 N 类问题。Logistic 回归最后一层是 r-1*m，而 Softmax 最后一层是 r*m；logistic 的损失是标准的交叉熵损失，而 SoftMax 损失是每一维的二分类损失的一半，其中 r 表示类别数，m 表示样本数。

$$SoftMax:\ L(\hat{y},y)=-\sum_{i=1}^{r}y_i\log\hat{y}_i$$

$$Logistic:\ L(\hat{y},y)=-\sum_{i=1}^{r-1}y_i\log\hat{y}_i-\sum_{i=1}^{r-1}(1-y_i)\log(1-\hat{y}_i)$$

# 第三课-ML 策略

1 正交化调试，即找到问题对应的调试策略，控制变量法，保证修改一个变量只会影响其中一个问题，而不会存在交叉。
2 偏差方差问题，一般来讲，将人类表现作为 bayes-error，作为误差上限，通过判断训练结果与人类表现来衡量是偏差问题，用训练集与测试集的误差衡量

1. 偏差/方差

偏差和方差是机器学习中常说两个概念，之所以这么重要是机器学习的理论基石是概率论，我们假设实际数据满足一个目标空间的分布函数（可学习的前提），而我们建立的模型就是尽可能接近目标空间，在逼近的过程中就涉及到模型空间与目标空间的之间的差距就可以用偏差衡量。由于我们的模型是建立在训练集上的，很有可能训练集并不能涵盖整个目标空间，这就导致目标空间方差问题。

一般我们假设模型空间与目标空间的偏差满足一个 Bayes error。但实际上这个误差上限很难求解，我们一般采用人类表现来作为这个误差上限。我们将模型在训练集上的表现与误差上限作为偏差，而训练集上的表现与测试集上的表现作为方差。这样我们就能决策是从偏差角度还是从方差角度改进模型。

1 确定在训练集上表现不错： 偏差没问题
2 确定在测试集上没问题： 方差没问题
3 在真实数据集上没问题： 真实数据与试验数据分布一致，目标函数没问题

2.正交化调试思想

一个好调试的模型应该具备正交化调试的性质，即每个属性之间是正交的，不存在交叉，调试一个参数只影响其中一个属性。
在我们确定从偏差或者方差角度去改进模型时，我们就只调试会影响其中之一的那些参数，一般来讲

偏差： 网络结构，优化算法，学习率
方差：正则化，Dropout，增加训练集，训练集与测试集数据一致性
数据分布不一致： 修改数据集
指标不行： 修改目标函数

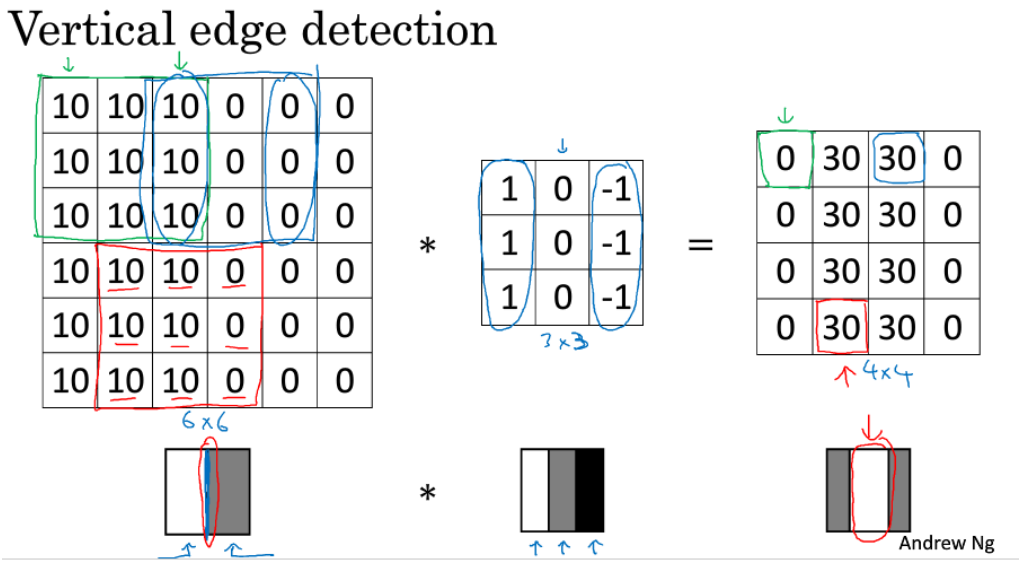
3 迁移学习

针对于训练集和测试来自不同的分布时，如何处理数据不匹配问题引发了迁移学习的概念。迁移学习 把一个训练好的网络的参数作为初始值，对其进行输入输出的一些改变，重新训练。这样做的好处是加速了算法收敛，同时利用了一个训练好的网络的优点，一般来讲迁移学习有两个特点，可迁移，A 与 B 两者目标尽可能接近，值得迁移，即 A 学习到的对 B 有价值，一般体现在 A 数据量大于 B,而 B 无法提供大量数据来学习。

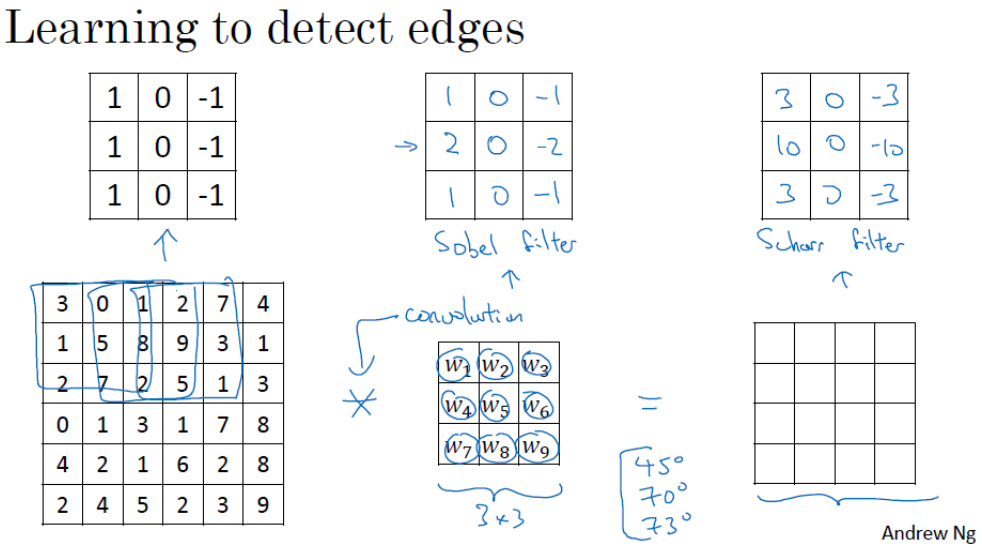
# 第四课 CNN

## 1 卷积

卷积是一个特征提取的过程，在手工提取特征时，常采用一些边缘检测算子依据图片中的像素来检测图片中的边缘。

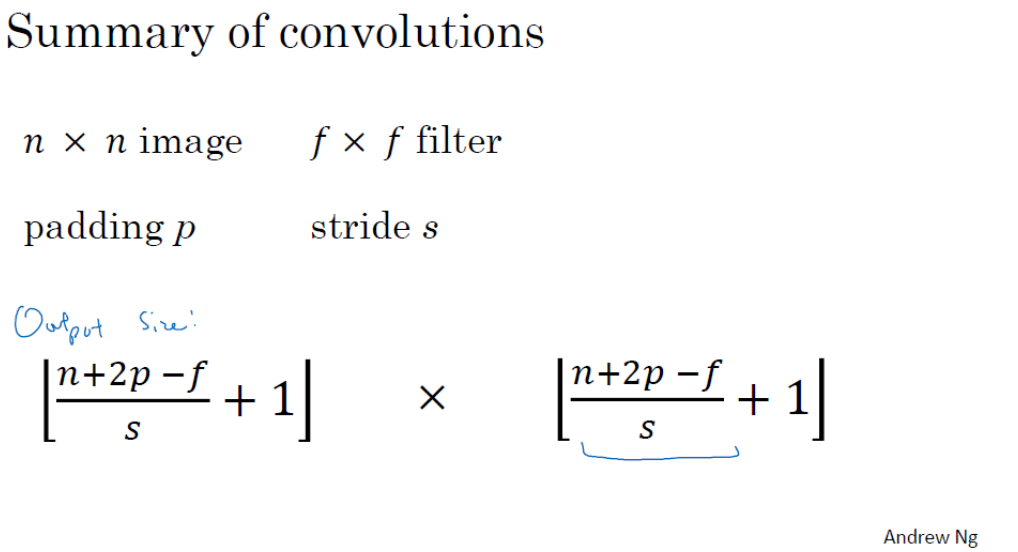


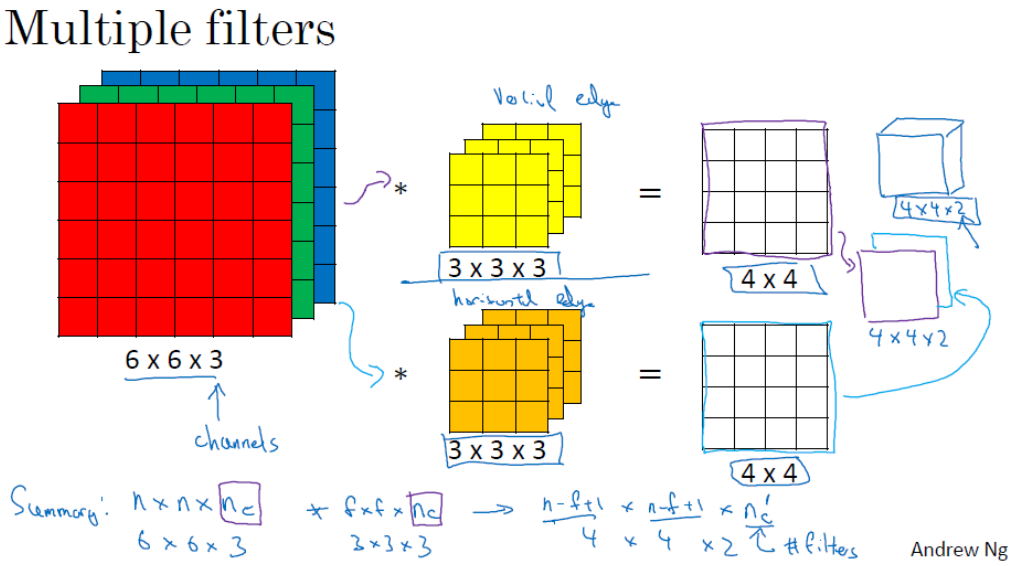

卷积是将卷积核中的权重当做参数来学习，进而学习到不同的特征提取卷积核来提取图像特征。




卷积的过程就是不断提取卷积核大小的区域块与卷积核进行对应相乘，在卷积核的作用下得到新的大小的像素块，卷积的过程中可以发现边缘的像素会比中间的像素少操作几次，而且经过卷积之后图片大小会减小。这里需要注意卷积核不是一个二维矩阵，而是一个包含深度的三维矩阵，原始的图片一般有 RGB 三通道，而在网络中深度由上一层的卷积核的数目而定，

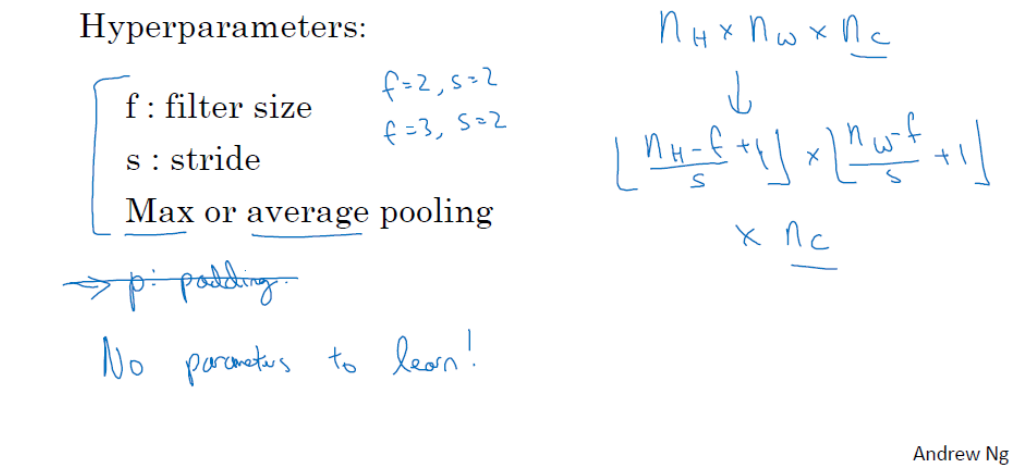
其中超参数有卷积核的大小，步长，填充，卷积核的数目（决定输出图像的深度）。



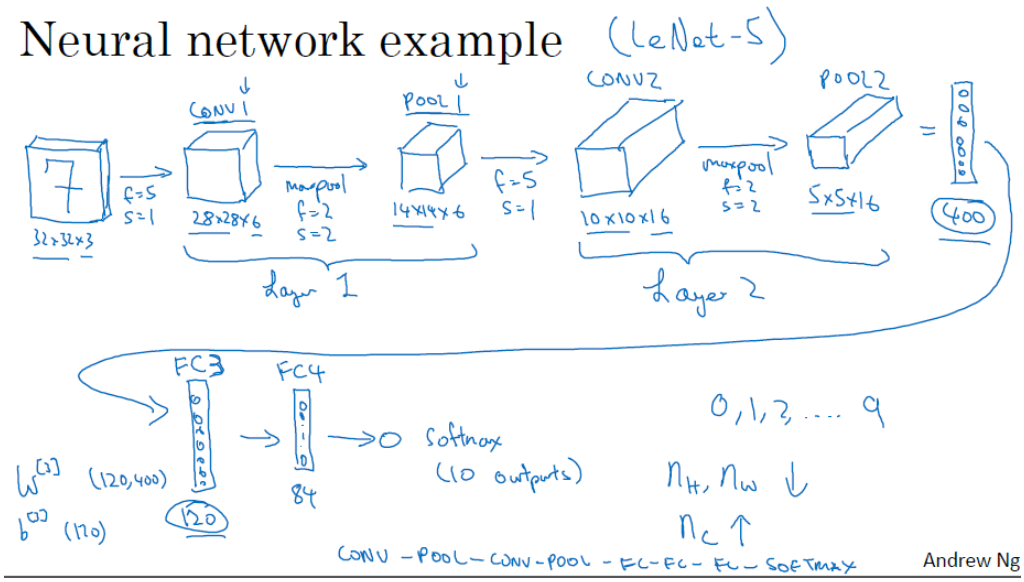

2 池化

池化层是一种固定的操作，只有超参，没有可学习的参数，池化操作主要包括最大池化或者平均池化。池化层中的超参有，核的大小，步长，填充常用 0。池化层只改变图像的高度和宽度，深度不变。




3 全连接层

全连接层就是一个普通的深度神经网络，把一张三维（高，宽，深）的图片拉成一维向量，把每一个像素看作是特征与下一层输出全连接。一个完整的 CNN 包括卷积，池化，全连接，softmax 层。




4 为什么采用卷积

1、提取特征，代替原始的手工特征

2、如果直接把像素点看作特征，直接采用全连接层，那么网络的参数是庞大的，卷积能实现参数共享的作用。

5 CNN 构建完整形式

前向传递

$$X^{[L-1]} : \left(m, H, W, D^{L-1}\right)$$

$$W^{[L]} : \left(f, f, D^{L-1}, D^{L}\right)$$

$$Z^{[L]} : \left(m, \left\lfloor \left(H + 2P - f\right) / S + 1 \right\rfloor, \left\lfloor \left(W + 2P - f\right) / S + 1 \right\rfloor, D^{L}\right)$$

$$A^{[L]} = Z^{[L]} = \left(m, H, W, D^{L}\right)$$

$$P^{[L]} : \left(m, \left\lfloor \left(H + 2P - f\right) / S + 1 \right\rfloor, \left\lfloor \left(W + 2P - f\right) / S + 1 \right\rfloor, D^{L}\right)$$

$$P_f^{[L]} : \left(m, H^{L} \times W^{L} \times D^{L}\right)$$

$$W_y^{[L]} : (n_y^{L}, n_y^{L-1}), n_y^{L-1} = H \times W \times D$$

$$Y^{[L]} : \left(m, n_y^{L}\right)$$

$$Z^{[L]} = F\left(X^{[L]}, W^{[L]}\right)$$

$$A^{[L]} = relu\left(Z^{[L]}\right)$$

$$P^{[L]} = P\left(A^{[L]}\right)$$

$$P_f^{[L]} \leftarrow \left(P^{[L]}\right)$$

$$Z_y^{[L]} = W_y^{[L]} P_f^{[L]} + b_y$$

$$y = soft\max\left(Z_y^{L}\right)$$

$$Loss : J = \frac{1}{m} \sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right), L = -y^{(i)} \log\left(\hat{y}^{(i)}\right)$$

反向传播

$$dZ_y^{[L]} = \left(\hat{y} - y\right)$$

$$dW_y^{[L]} = dZ_y^{[L]} P_f^T$$

$$db_y^{[L]} = \sum dZ_y^{[L]}$$

$$dP_f^{[L]} = W_y^T dZ_y^{[L]}$$

$$\mathrm{d}P^{[L]} \leftarrow dP_f^{[L]}$$

$$dA^{[L]} = 1\left\{\max\left(A^{[L]}\right)\right\} dP^{[L]}, \frac{1}{H^{[L]} \times W^{[L]}} dP^{[L]}$$

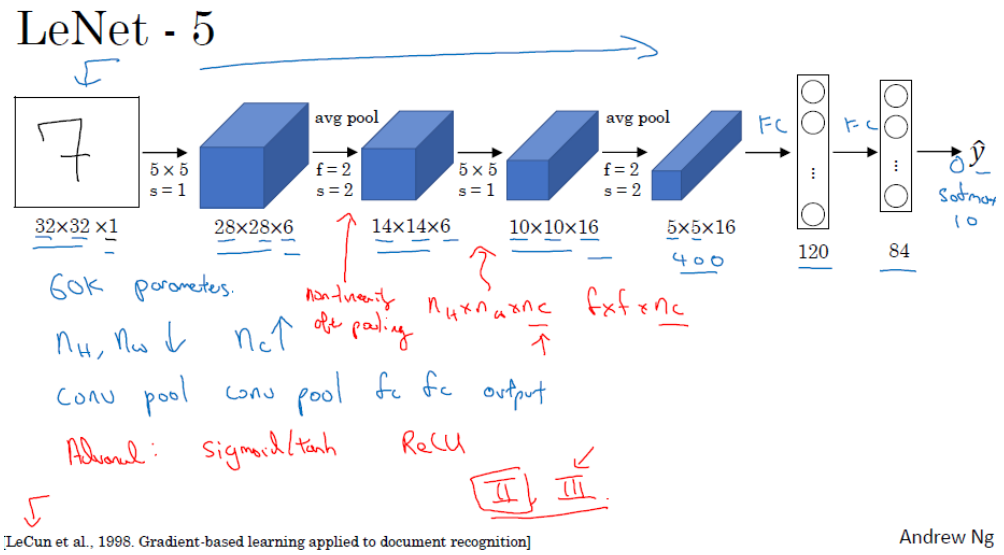
$$dZ^{[L]} = dA^{[L]} * \left(Z > 0\right)$$

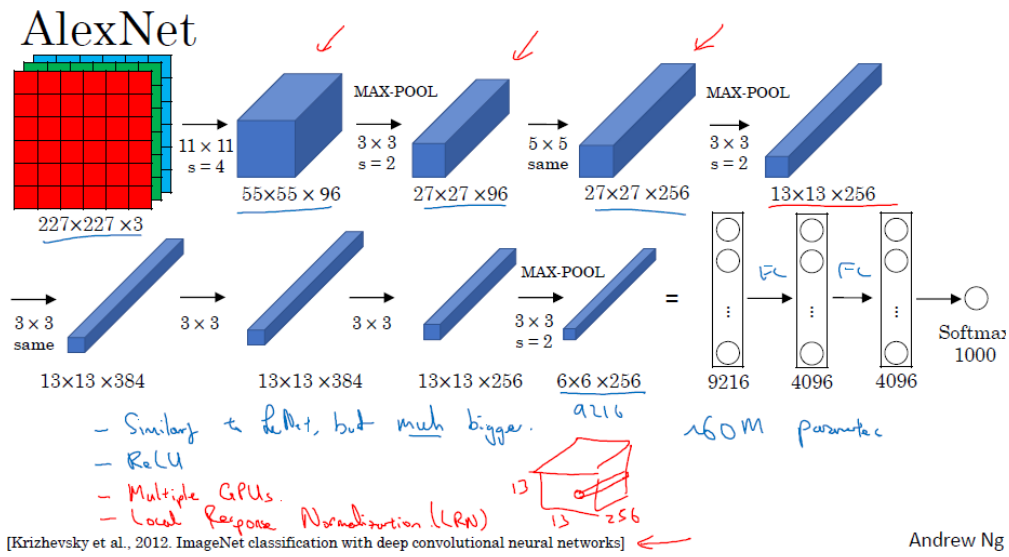
$$dW^{[L]} += \sum_{h=0}^{H^L} \sum_{w=0}^{W^L} a_{conv} * dZ_{hw}^{[L]}$$

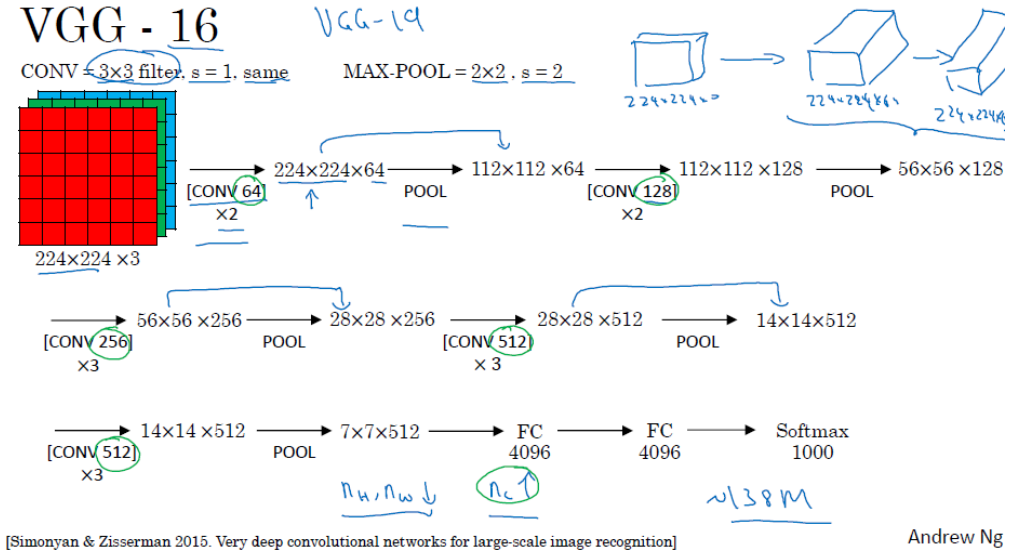
$$db^{[L]} = \sum_{h=0}^{H^L} \sum_{w=0}^{W^L} dZ_{hw}^{[L]}$$

$$dA^{[L]} += \sum_{h=0}^{H^L} \sum_{w=0}^{W^L} W^{[L]} * dZ_{hw}^{[L]}$$

# 6 经典的 CNN 架构

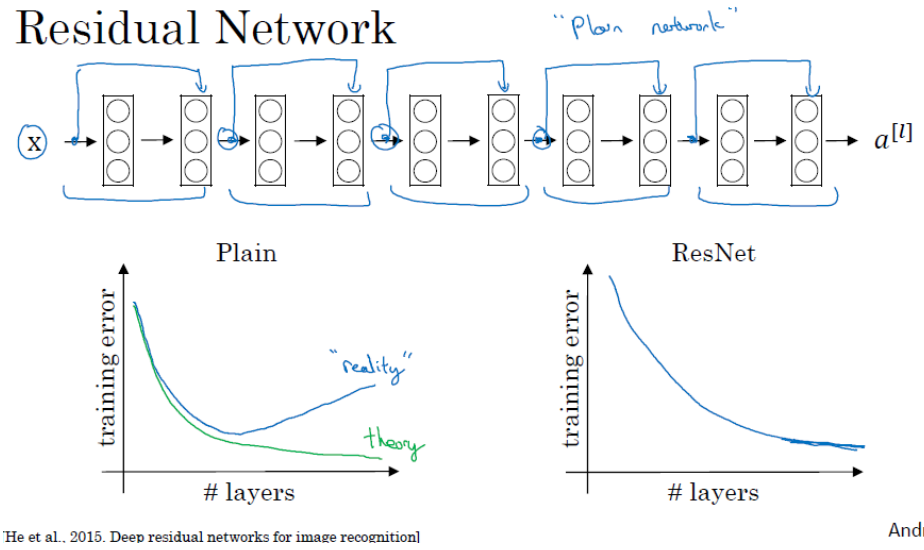

LeNet - 5
32×32 ×1
5 × 5
s = 1
28×28×6
avg pool
f = 2
s = 2
14×14×6
5 × 5
s = 1
10×10×16
avg pool
f = 2
s = 2
5×5×16
FC
120
FC
84
ŷ
Softmax
10
60k parameters.
conv pool conv pool fc fc output
Activation: Sigmoid/tanh ReLU
[LeCun et al., 1998. Gradient-based learning applied to document recognition]
Andrew Ng


AlexNet
227×227 ×3
11 × 11
s = 4
55×55 × 96
MAX-POOL
3 × 3
s = 2
27×27 ×96
5 × 5
same
27×27 ×256
MAX-POOL
3 × 3
s = 2
13×13 ×256
3 × 3
same
13×13 ×384
3 × 3
13×13 ×384
3 × 3
13×13 ×256
MAX-POOL
3 × 3
s = 2
6×6 ×256
=
9216
FC
4096
FC
4096
Softmax
1000
- Similar to LeNet, but much bigger.
- ReLU
- Multiple GPUs.
- Local Response Normalization (LRN)
60M parameters
[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]
Andrew Ng


VGG - 16
VGG-16
CONV = 3×3 filter, s = 1, same
MAX-POOL = 2×2 , s = 2
224×224 ×3
[CONV 64]
×2
224×224×64
POOL
112×112 ×64
[CONV 128]
×2
112×112 ×128
POOL
56×56 ×128
[CONV 256]
×3
56×56 ×256
POOL
28×28 ×256
[CONV 512]
× 3
28×28 ×512
POOL
14×14×512
[CONV 512]
×3
14×14 ×512
POOL
7×7×512
FC
4096
FC
4096
Softmax
1000
$n_H, n_w \downarrow$
$n_c \uparrow$
~138M
[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]
Andrew Ng

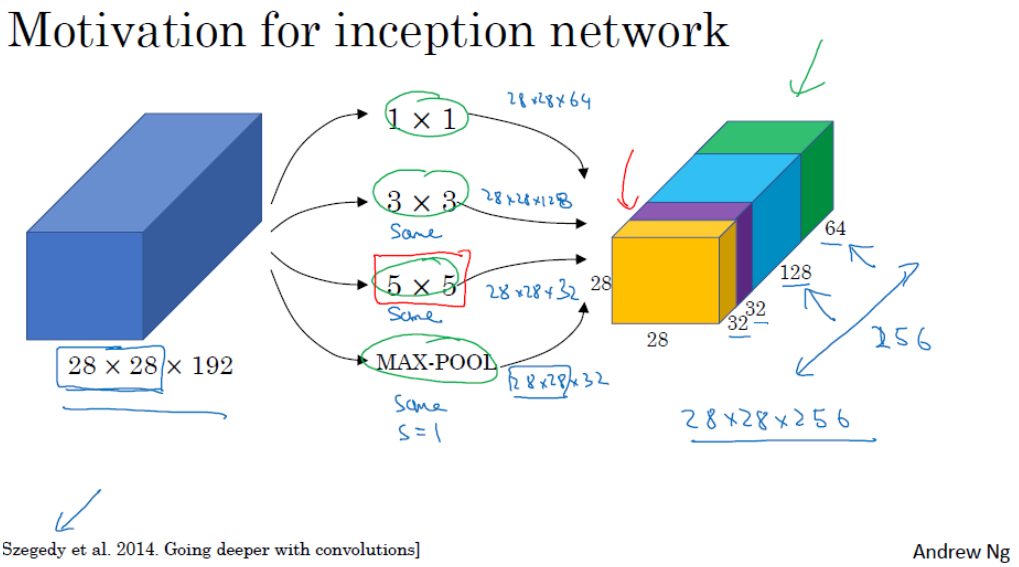
Res-Net

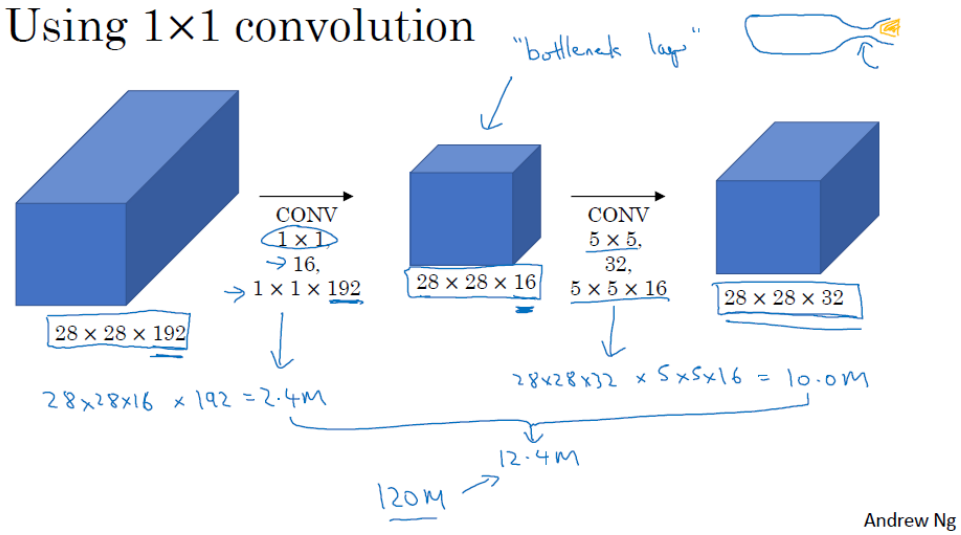
动机是解决深度神经网络中的梯度消失问题，在普通的深度神经网络中加入跳跃连接，使得信息的传递不会快速消失。

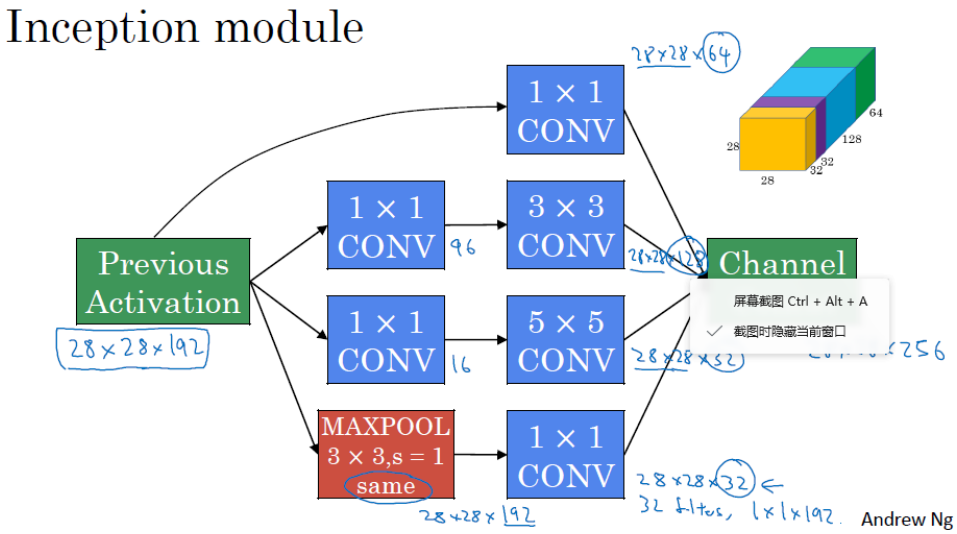



Google-Net（inception）

动机是解决手动设置卷积核大小问题，将大小不同的卷积核放到同一卷积层中自动学习合适的卷积核大小，用到了 1*1 的卷积核来降低图像的深度进而降低复杂度。






7 构建 CNN 网络结构的一般流程

迁移学习思想，在任务的训练数据不足时，可以直接采用已经训练好的网络权重，将其看作一个特征提取的函数封装起来，接上一层自己的 softmax 层，训练学习 softmax 层的参数。当数据量一般时，可以选择性使用已经训练好的网络中的前几层，自定义一些卷积层和池化层。当数据量很大时，可以采用已经训练好的网络权重作为网络架构的初始化。至于网络架构的选择，可以自定义，也可以采用一些经典的网络架构。

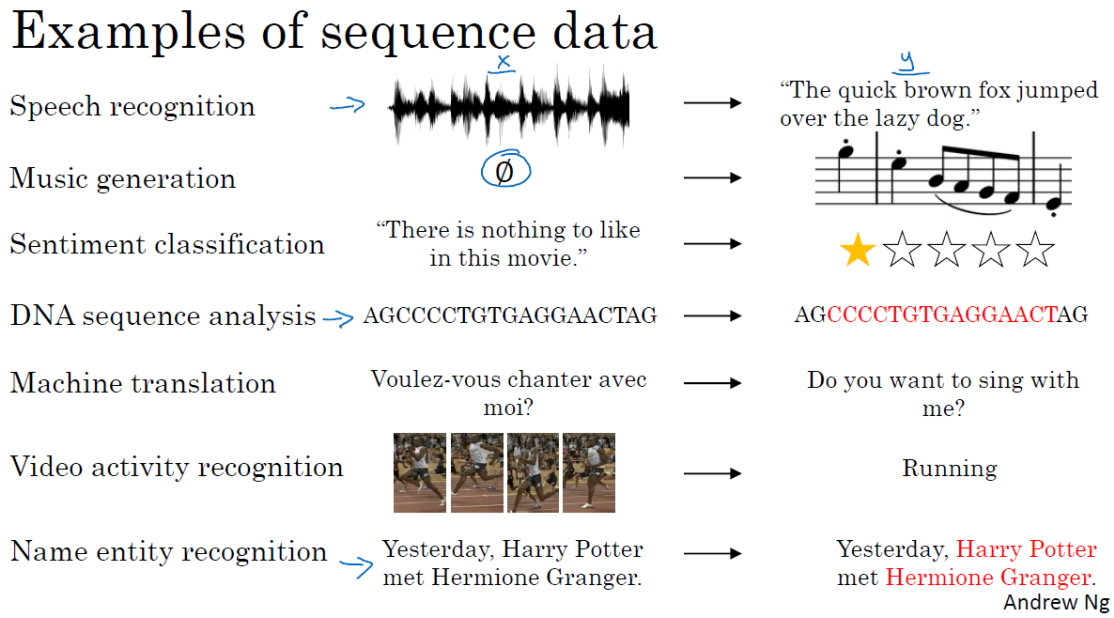
一般认为知识的来源 “手工特征”，“手工的标签”，当数据量大时意味着手工的标签多，这时可以尽可能少的定义手工特征，当数据量小时，手工特征对精度的提升有明显的作用。

数据填充：可以采用“翻转”，“裁剪”，“旋转”，“失真”等操作扩大数据集。

# 第五课循环神经网络（RNN）

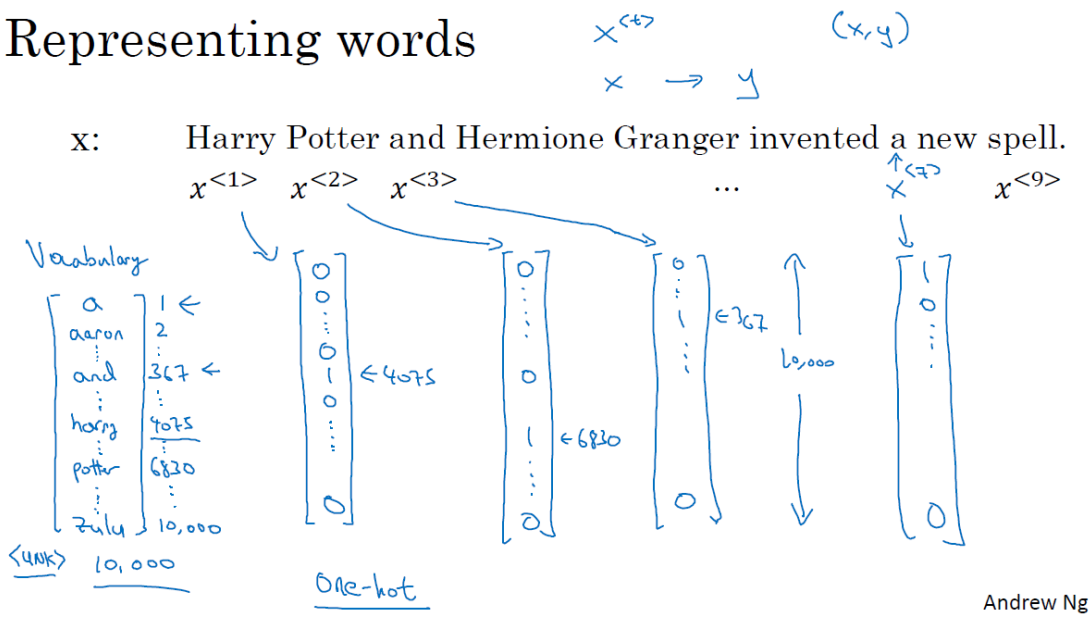
## 1、RNN 介绍

应用于序列模型（时序性）



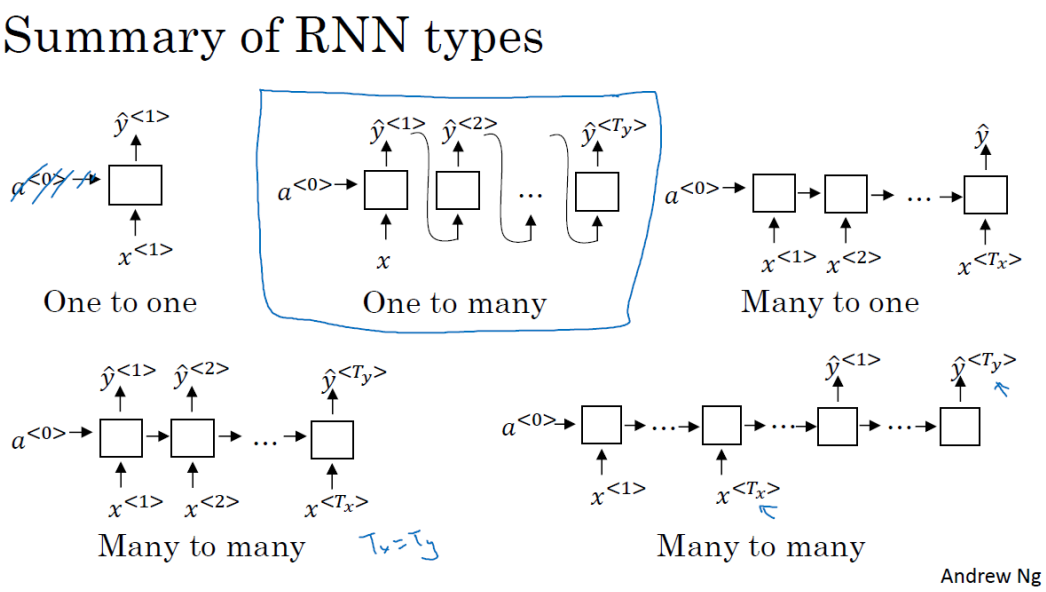

## 2、符号定义

这里采用 RNN 常用的样本来进行符号介绍，假设训练集是一个文本数据集，是 m 条句子。$x^{(i)}$表示第 i 个句子，$x^{(i)\langle t\rangle}$表示第 i 个句子的第 t 个词。而每个词最常采用的表示是建立词典后用 one-hot 表示。这里的 t 可以看作是时序的第 t 个时刻，每一个时刻的输入是在该时刻（位置）上所有样本中在该位置的词，即 $x^{\langle t\rangle} \in R^{N\times m}$，其中 N 是字典长度，m 是训练样本数。所以整个训练集是一个三维数组。



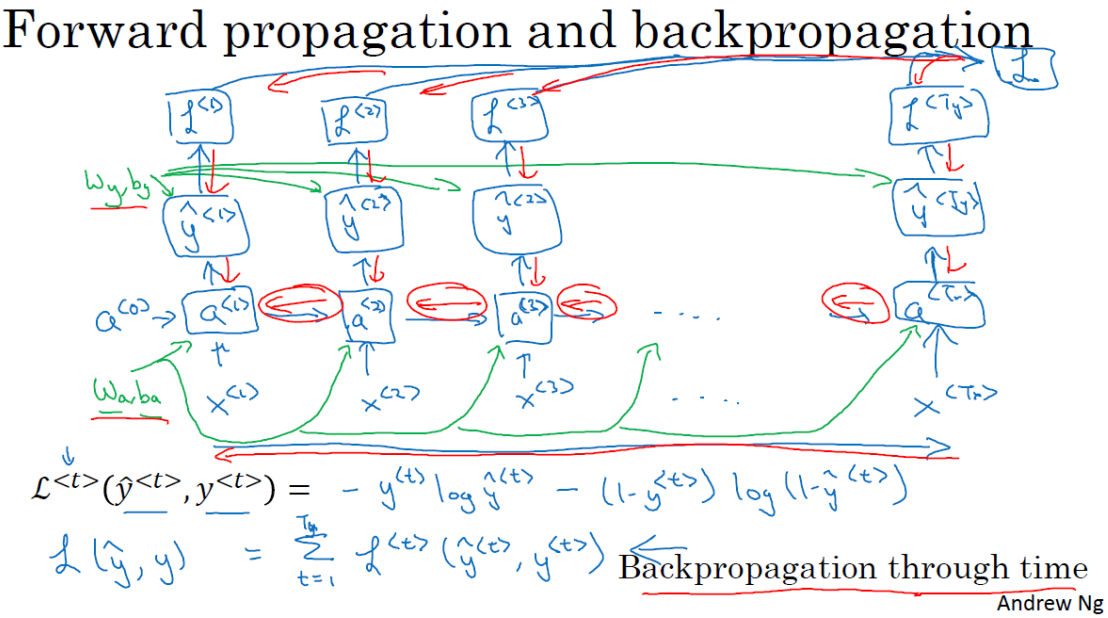
3、RNN 类型

网络的架构取决于目标，从




4、RNN 前向反向传播

其中绿线为前向传播，红线为反向传播，可以看出神经网络的损失是依目标而定的，损失的传播也大同小异。



5 循环神经网络结构（RNN, GRU, LSTM, BRNN, DRNN）

$$
\begin{aligned}
&X:\left(X_n, m, X_T\right)\\
&a:\left(a_n, m\right)\\
&W_a:\left(a_n, a_n+X_n\right)\quad W_{ax}:\left(a_n, X_n\right)\quad W_{aa}:\left(a_n, a_n\right)\\
&Y:\left(Y_n, m, Y_T\right)
\end{aligned}
$$

$soft\ \max$ $RNN$损失函数：$\sum_{i=1}^{m}\sum_{t=1}^{Y_T}-y^{(i)\langle t\rangle}\log\hat{y}^{(i)\langle t\rangle}=\sum_{i=1}^{m}\sum_{t=1}^{Y_T}-y^{(i)\langle t\rangle}\log\hat{y}^{(i)\langle t\rangle}$

RNN 单元

$$a^{\langle t\rangle}=g\left(W_{aa}a^{\langle t-1\rangle}+W_{ax}x^{\langle t\rangle}+b_a\right)$$

$$y^{\langle t\rangle}=g\left(W_{ya}a^{\langle t\rangle}+b_y\right)$$

$$da^{\langle t\rangle}=W_{ya}\left(y^{\langle t\rangle}-\hat{y}^{\langle t\rangle}\right)+0$$

$$dW_{ya}{}^{\langle t\rangle}=\left(y^{\langle t\rangle}-\hat{y}^{\langle t\rangle}\right)a^{\langle t\rangle}$$

$$db_y{}^{\langle t\rangle}=\sum_i\left(y^{\langle t\rangle}-\hat{y}^{\langle t\rangle}\right)$$

$$dW_{aa}{}^{\langle t\rangle}=da^{\langle t\rangle}\left(1-\left(a^{\langle t\rangle}\right)^2\right)a^{\langle t-1\rangle}$$

$$dW_{ax}{}^{\langle t\rangle}=da^{\langle t\rangle}\left(1-\left(a^{\langle t\rangle}\right)^2\right)x^{\langle t-1\rangle}$$

$$db_a{}^{\langle t\rangle}=\sum_i da^{\langle t\rangle}\left(1-\left(a^{\langle t\rangle}\right)^2\right)$$

$$da^{\langle t-1\rangle}=da^{\langle t\rangle}\left(1-\left(a^{\langle t\rangle}\right)^2\right)W_a\left[1:n_a\right]$$

GRU 单元

$$\Gamma_f^{\langle t\rangle} = g\left(W_{fa}a^{\langle t-1\rangle} + W_{fx}x^{\langle t\rangle} + b_f\right)$$

$$\tilde{a}^{\langle t\rangle} = g\left(W_{aa}\left(\Gamma_f^{\langle t\rangle} * a^{\langle t-1\rangle}\right) + W_{ax}x^{\langle t\rangle} + b_a\right) = g\left(W_a\left[\left(\Gamma_f^{\langle t\rangle} * a^{\langle t-1\rangle}\right) : x^{\langle t\rangle}\right]^T + b_a\right)$$

$$\Gamma_u^{\langle t\rangle} = g\left(W_{ua}a^{\langle t-1\rangle} + W_{ux}x^{\langle t\rangle} + b_u\right)$$

$$a^{\langle t\rangle} = \Gamma_u^{\langle t\rangle}\tilde{a}^{\langle t\rangle} + (1 - \Gamma_u^{\langle t\rangle})a^{\langle t-1\rangle}$$

$$y^{\langle t\rangle} = g\left(W_{ya}a^{\langle t\rangle} + b_y\right)$$

$$da^{\langle t\rangle} = \left(y^{\langle t\rangle} - \hat{y}^{\langle t\rangle}\right)W_{ya} + 0$$

$$dW_{ya}{}^{\langle t\rangle} = \left(y^{\langle t\rangle} - \hat{y}^{\langle t\rangle}\right)a^{\langle t\rangle}$$

$$db_y{}^{\langle t\rangle} = \left(y^{\langle t\rangle} - \hat{y}^{\langle t\rangle}\right)$$

$$dot^{\langle t\rangle} = da^{\langle t\rangle} * \left(aat - a^{\langle t-1\rangle}\right)$$

$$daat^{\langle t\rangle} = da^{\langle t\rangle} * ot$$

$$dft^{\langle t\rangle} = \left(daat^{\langle t\rangle} * aat(1 - (aat))\right)W_a[:, : n_a] * a^{\langle t\rangle}$$

$$dW_o{}^{\langle t\rangle} = dot^{\langle t\rangle} * ot^{\langle t\rangle} * \left(1 - ot^{\langle t\rangle}\right)\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right], db_o{}^{\langle t\rangle} = \sum_i dot^{\langle t\rangle} * ot^{\langle t\rangle} * \left(1 - ot^{\langle t\rangle}\right)$$

$$dW_f{}^{\langle t\rangle} = dft^{\langle t\rangle}ft^{\langle t\rangle}\left(1 - ft^{\langle t\rangle}\right)\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right], db_f{}^{\langle t\rangle} = \sum_i dft^{\langle t\rangle}ft^{\langle t\rangle}\left(1 - ft^{\langle t\rangle}\right)$$

$$dW_a{}^{\langle t\rangle} = da^{\langle t\rangle}\left(1 - \left(a^{\langle t\rangle}\right)^2\right)\left[ft * a^{\langle t-1\rangle} : x^{\langle t\rangle}\right], db_a{}^{\langle t\rangle} = \sum_i da^{\langle t\rangle}\left(1 - \left(a^{\langle t\rangle}\right)^2\right)$$

$$da^{\langle t-1\rangle} = W_f[:, : n_a]dft^{\langle t\rangle}ft^{\langle t\rangle}\left(1 - ft^{\langle t\rangle}\right) + W_a[:, : n_a]daat^{\langle t\rangle}\left(1 - \left(aat^{\langle t\rangle}\right)^2\right)$$

$$+ W_o[:, : n_a]dot^{\langle t\rangle}ot^{\langle t\rangle}\left(1 - ot^{\langle t\rangle}\right) + (1 - ot)da^{\langle t\rangle}$$

LSTM 单元

$$\tilde{c}^{\langle t\rangle} = \tanh\left(W_{aa}a^{\langle t-1\rangle} + W_{ax}x^{\langle t\rangle} + b_c\right) = \tanh\left(W_c\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right]^T + b_c\right)$$

$$\Gamma_u^{\langle t\rangle} = \sigma\left(W_{ua}a^{\langle t-1\rangle} + W_{ux}x^{\langle t\rangle} + b_u\right) = \tanh\left(W_u\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right]^T + b_u\right)$$

$$\Gamma_f^{\langle t\rangle} = \sigma\left(W_{fa}a^{\langle t-1\rangle} + W_{fx}x^{\langle t\rangle} + b_f\right) = \tanh\left(W_f\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right]^T + b_f\right)$$

$$c^{\langle t\rangle} = \Gamma_u^{\langle t\rangle} * \tilde{c}^{\langle t\rangle} + \Gamma_f^{\langle t\rangle} * c^{\langle t-1\rangle}$$

$$\Gamma_o^{\langle t\rangle} = \sigma\left(W_{oa}a^{\langle t-1\rangle} + W_{oa}x^{\langle t\rangle} + b_o\right) = \tanh\left(W_o\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right]^T + b_o\right)$$

$$a^{\langle t\rangle} = \Gamma_o^{\langle t\rangle}\tanh\left(c^{\langle t\rangle}\right)$$

$$da^{\langle t\rangle} = \left(y^{\langle t\rangle} - \hat{y}^{\langle t\rangle}\right)W_{ya} + 0$$

$$dW_{ya}{}^{\langle t\rangle} = \left(y^{\langle t\rangle} - \hat{y}^{\langle t\rangle}\right)a^{\langle t\rangle}$$

$$db_y{}^{\langle t\rangle} = \left(y^{\langle t\rangle} - \hat{y}^{\langle t\rangle}\right)$$

$$dot^{\langle t\rangle} = da^{\langle t\rangle} * \tanh(c^{\langle t\rangle})$$

$$dcct^{\langle t\rangle} = dc^{\langle t\rangle} * it + da^{\langle t\rangle} * ot * \left(1 - \tanh\left(c^{\langle t\rangle}\right)^2\right) * it$$

$$dft^{\langle t\rangle} = \left(dc^{\langle t\rangle} * c^{\langle t-1\rangle} + da^{\langle t\rangle} * ot * \left(1 - \tanh\left(c^{\langle t\rangle}\right)^2\right) * c^{\langle t-1\rangle}\right)$$

$$dit^{\langle t\rangle} = \left(dc^{\langle t\rangle} * cct + da^{\langle t\rangle} * ot * \left(1 - \tanh\left(c^{\langle t\rangle}\right)^2\right) * cct\right)$$

$$dW_f{}^{\langle t\rangle} = dft^{\langle t\rangle}ft^{\langle t\rangle}\left(1 - ft^{\langle t\rangle}\right)\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right], db_f{}^{\langle t\rangle} = \sum_i dft^{\langle t\rangle}ft^{\langle t\rangle}\left(1 - ft^{\langle t\rangle}\right)$$

$$dW_i{}^{\langle t\rangle} = dit^{\langle t\rangle}it^{\langle t\rangle}\left(1 - it^{\langle t\rangle}\right)\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right], db_o{}^{\langle t\rangle} = \sum_i dft^{\langle t\rangle}it^{\langle t\rangle}\left(1 - it^{\langle t\rangle}\right)$$

$$dW_o{}^{\langle t\rangle} = dot^{\langle t\rangle}ot^{\langle t\rangle}\left(1 - ot^{\langle t\rangle}\right)\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right], db_o{}^{\langle t\rangle} = \sum_i dot^{\langle t\rangle}ot^{\langle t\rangle}\left(1 - ot^{\langle t\rangle}\right)$$

$$dW_c{}^{\langle t\rangle} = dcc^{\langle t\rangle}\left(1 - \left(cct^{\langle t\rangle}\right)^2\right)\left[a^{\langle t-1\rangle} : x^{\langle t\rangle}\right], db_a{}^{\langle t\rangle} = \sum_i dc^{\langle t\rangle}\left(1 - \left(cct^{\langle t\rangle}\right)^2\right)$$

$$da^{\langle t-1\rangle} = W_f[:,:n_a]dft^{\langle t\rangle}ft^{\langle t\rangle}\left(1-ft^{\langle t\rangle}\right) + W_c[:,:n_a]dcct^{\langle t\rangle}\left(1-\left(cct^{\langle t\rangle}\right)^2\right)$$
$$+ W_i[:,:n_a]dit^{\langle t\rangle}it^{\langle t\rangle}\left(1-it^{\langle t\rangle}\right) + (1-it)da^{\langle t\rangle}$$
$$+ W_o[:,:n_a]dot^{\langle t\rangle}ot^{\langle t\rangle}\left(1-ot^{\langle t\rangle}\right) + (1-ot)da^{\langle t\rangle}$$
$$dc^{\langle t-1\rangle} = dc^{\langle t\rangle} * ft + da^{\langle t\rangle} * ot * \left(1-\tanh\left(c^{\langle t\rangle}\right)^2\right) * ft$$

算法流程

输入输出准备工作（embeding 层）

1 初始化参数，注意参数的维度
RNN 一共两组参数（wa,ba;wy,by），
GRU 一共四组参数（wa,ba;wf,bf;wu,bu;wy,by）
LSTM 一共五组参数（wa,ba;wf,bf;wu,bu;wo,bo;wy,by）

2 前向传递
根据公式实现链式传播，加 softmax 层计算预测值。注意矩阵乘法和一般乘法，sigmoid，tanh，门操作都是一般乘法。保存反向求导时需要的变量到 cache 中，由于参数共享（其实就是一个神经元循环使用），所以参数可以不随每时刻变量一起保存在 cache 中。

3 反向传播
首先对损失函数 softmax 层(因目标而异)求导，后进行链式求导。注意矩阵求导和一般求导，转置等问题。由于参数共享（其实就是一个神经元循环使用），实际梯度由 T 个时刻序列累加确定。

4 更新参数

Word2Vec