

DL

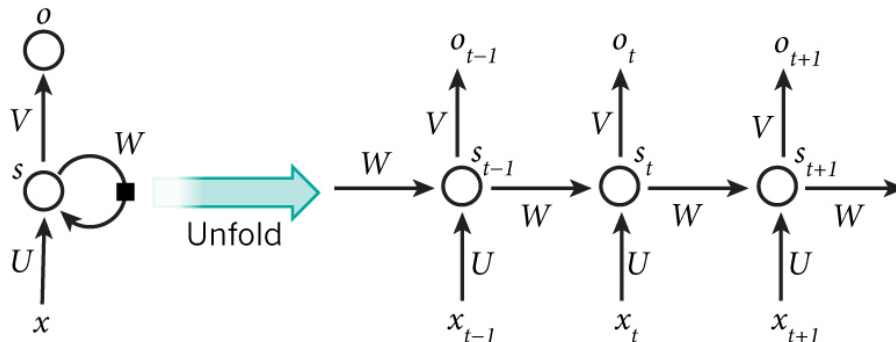
RNN

- 原理

- 基于图展开和参数共享的思想。

- 展开计算图：输出的每一项是前一项的函数， $h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$

- 公式推导



$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$\hat{y}_t = \text{softmax}(Vs_t)$$

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- loss (softmax 交叉熵对数损失，(将 RNN 的输出解释为条件概率分布))

$$E_t(y_t, \hat{y}_t) = -y_t \log(\hat{y}_t)$$

$$E_t(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log(\hat{y}_t)$$

- 优化&trick (通过时间反向传播 (BPTT))

- BPTT & 梯度消失：两个方向，沿时间反向传播(与 W 有关) & 传递到上一层网络(与 U 有关)

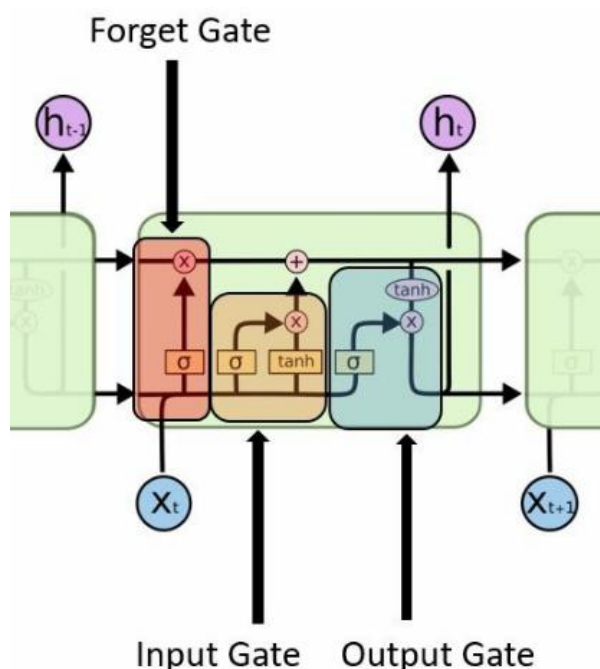
- 解决梯度消失/爆炸问题

- 解决梯度消失：使用 ReLU 而不是 tanh 或者 sigmoid；换用 LSTM 或 GRU
- 解决梯度爆炸：使用梯度裁减 (gradient clipping) (比较少出现)

- 适用性&优缺点

- 比较

★ LSTM



- Gates：在 LSTM 中，网络首先构建了 3 个 gates 来控制信息的流量（注：gates 并不提供额外信息，只起到限制信息的量的作用，是过滤器作用，所以所用的激活函数是 sigmoid 而不是 tanh。）
- Activations：tanh 可以保证输出在(-1,1)之间，这里的两个 tanh 也可以换成别的
- 正则化：可以对输入输出加 L2 正则或者使用 dropout，但是对循环连接不能用

• 历史信息累积：

- 式子： $c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$
- 其中 $new = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ 是本次要累积的信息来源。
- 改写： $c_t = f_t \odot c_{t-1} + i_t \odot new$

所以历史信息的累积是并不是靠隐藏状态 h 自身，而是依靠memory cell这个自连接来累积。在累积时，靠遗忘门来限制上一时刻的memory cell的信息，并靠输入门来限制新信息。并且真的达到了leaky units的思想，memory cell的自连接是线性的累积。

- **当前隐藏状态的计算**：如此大费周章的最终任然是同普通RNN一样要计算当前隐藏状态。

- 式子： $h_t = o_t \odot \tanh(c_t)$
- 当前隐藏状态 h_t 是从 c_t 计算得来的，因为 c_t 是以线性的方式自我更新的，所以先将其加入带有非线性功能的 $\tanh(c_t)$ 。随后再靠输出门 o_t 的过滤来得到当前隐藏状态 h_t 。

- 长短期记忆解决梯度消失问题了吗？

○ 【<http://qr.ae/TU86H9>】

- 有两个因素会影响梯度的大小：梯度经过的权重和激活函数的导数。如果这些因素中的任何一个小于 1，则梯度可能会消失（tanh 导数 ≤ 1 ；sigmoid 导数 ≤ 0.25 ）；如果大于 1，则可能会爆炸（使用梯度截断（gradient clipping））。

$$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$$

if $\|\hat{\mathbf{g}}\| \geq threshold$ then

$$\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$$

end if

- 在 LSTM 的循环中，使用加法修正而非连乘，所以若 forget gate 接近 1 就不容易梯度消失。
- LSTM 通过**加法修正**而不同于 RNN 连乘，在很大程度上解决了梯度消失，**但遗忘门本身也是连乘的**，所以实际上**没有本质解决** RNN 的梯度消失。（RNN 把新信息和历史信息对当下的影响用统一一个权重 W 表示，而 LSTM 通过遗忘门把历史信息嵌进来，通过另一个不同的权重去表示历史信息对当下的影响。）
- LSTM 和 RNN 进行对比和优劣分析

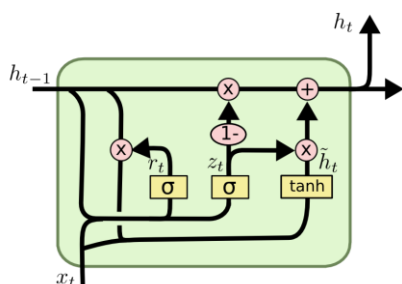
比较公式：最大的区别是多了三个神经网络(gates)来控制数据的流通。

 - **普通RNN：** $h_t = \tanh(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b)$
 - **LSTM：**

$$h_t = o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c))$$
 - **比较：**二者的信息来源都是 $\tanh(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b)$ ，不同的是 **LSTM靠3个gates将信息的积累建立在线性自连接的memory cell之上，并靠其作为中间物来计算当前 h_t 。**
- 为什么要用 batch trianning? dynamic_rnn?
 - 尽管说 RNN/LSTM 能够处理可变大小的输入并计算可变大小的输出，但在实践中，由于需要高效的批训练，输入总是要预处理到相同的大小。dynamic_rnn 可以允许不同 batch 的 sequence length 不同，但 static_rnn 不能，每次迭代都是保持 max_seq，所以可以对于 sequence 先做聚类，预设几个固定的长度 bucket，然后每个 sequence 都放到它所属的 bucket 里面去，然后 pad 到固定的长度。
 - 关于 dynamic_rnn 的实现，假设当前 batch 总时间步数为 T ，那么一定会在时间轴上循环 T 次；对于其中某个时间步 t ，假如 t 超过了该序列的长度 seq_len，那么复制 zero output 和之前的 state 到下一个时间步；否则进行一步 RNNCell 的计算得到下一时间步的状态和输出。

GRU

GRU 是 LSTM 的变体，将忘记门和输入们合成了一个单一的更新门。



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

★ Attention

- Seq2seq

- seq2seq 是一个 Encoder-Decoder 结构的序列到序列模型，用于机器翻译、摘要生成、阅读理解等。这类问题的核心是要对条件概率 $P(\text{target}|\text{context})$ 进行建模。
- 输入一个 context 序列，输出一个 target 序列。encoder 读入 context 序列，得到一个 context vector（语意摘要，RNN 的最后一个 hidden state），然后 decoder 以这个 context vector 为起始 state，依次生成 target 的每一个单词。
- 这种做法的缺点是，无论之前的 context 有多长，包含多少信息量，最终都要被压缩成一个几百维的 vector。这意味着 context 越大，最终的 state vector 会丢失越多的信息。因为 context 在输入时已知，一个模型完全可以在 decode 的过程中利用 context 的全部信息，而不仅仅是最后一个 state。Attention 机制的核心思想就是这个。

- Attention 机制

- 首先是所有的 hidden state 都会被作为 Decoder 的输入，但 Decoder 当前要生成的状态只需要关注这些输入的特定部分，所以要乘一个权重向量 α ，本质上是加权平均。

★ BP

Bp 算法介绍，BP 的推导写一遍

求 $\frac{\partial}{\partial W_{ij}^{(l)}} J(W)$?

问题拆解: $\frac{\partial}{\partial W_{ij}^{(l)}} J(W) = \frac{\partial J(W)}{\partial Z_i^{(l+1)}} * \frac{\partial Z_i^{(l+1)}}{\partial W_{ij}^{(l)}}$

神经元求和: $Z_i^{(l+1)} = \sum_j^n W_{ij}^{(l)} * a_j^{(l)}$

输出对权值的偏导数: <https://blog.csdn.net/u014403897>

Summary: the equations of backpropagation

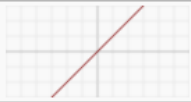






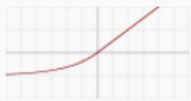
$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad (\text{BP1})$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad (\text{BP2}) \quad \frac{\partial Z_i^{(l+1)}}{\partial W_{ij}^{(l)}} = \frac{\partial \sum_j^n W_{ij}^{(l)} * a_j^{(l)}}{\partial W_{ij}^{(l)}} = a_j^{(l)}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (\text{BP3}) \quad \text{设神经元的错误变化率为:} \quad \text{最终:}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (\text{BP4}) \quad \delta_i^{l+1} = \frac{\partial J(W)}{\partial Z_i^{(l+1)}} \quad \boxed{\frac{\partial}{\partial W_{ij}^{(l)}} J(W) = \delta_i^{l+1} * a_j^{(l)}}$$

激活函数

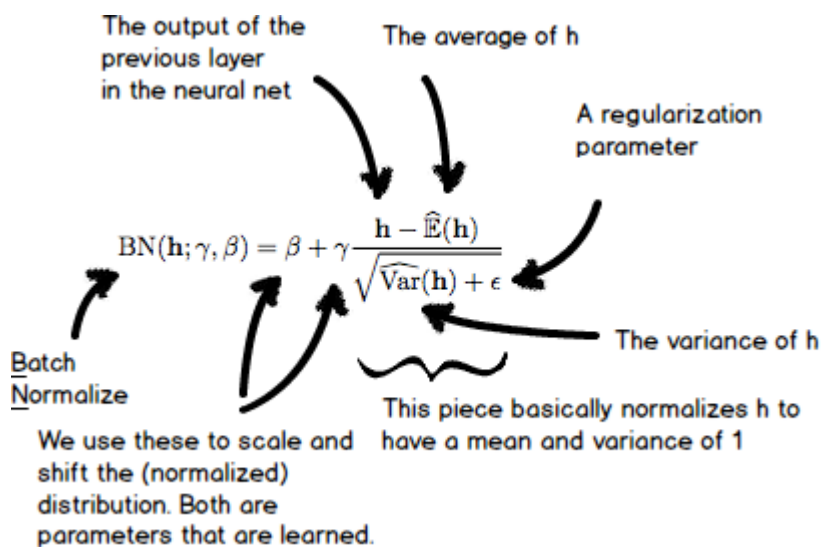
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

- 激活函数的意义?DL 常用的激活函数有哪些?不同激活函数的差异?
 - 引入非线性,不再是输入的线性组合,可以逼近任意函数
- 激活函数,饱和性质,饱和区间在哪?有什么影响?
- Relu 和 sigmoid 图像是什么样的?有什么区别,Relu 相比 sigmoid 函数的优点有哪些?有什么缺点?
 - 优点:
 - **节省计算量**: sigmoid/tanh 需要计算指数,计算复杂度高,ReLU 计算简单。
 - **防止梯度消失**: 对于深层网络,sigmoid 函数由于两端饱和,很容易就会出现梯度消失。ReLU 更容易学习优化,因为其分段线性性质,导致其前传,后传,求导都是分段线性)。
 - **稀疏性 (缓解了过拟合)**: Relu 会使一部分神经元的输出为 0,这样就造成了网络的稀疏性,并且减少了参数的相互依存关系,缓解了过拟合问题的发生。
 - 缺点:
 - ReLU 的输出不是 zero-centered
 - dead ReLU (神经元坏死现象),因为负轴上的梯度为 0,导致相应的参数永远不能被更新。导致这个现象的两个原因:倒霉的参数初始化; learning rate 太高了;
 - ReLU 不会对数据做幅度压缩 (在激活层前用 batch normalization,防止梯度爆炸)

- CNN 为什么用 ReLU，为什么不用 sigmoid？RNNs 为什么用 tanh 不用 ReLU？ReLU 的问题以及解决方案。
 - sigmoid 的输出在[0,1]之间，然后与其它信号做乘积，起到控制信息量的作用；而 relu 输出 [0, +inf]，还能放大信号，可能会发生梯度爆炸，需要做梯度裁剪，这恐怕是我们不想要的。
- 梯度消失，梯度爆炸：定义，原因，以及解决方法

实践

- 常用的深度学习的 trick？（Data Augmentation，Batch Normalization，Dropout，Gradient Clipping，Weight Decay，Shuffle，Learning rate 调整(Adam 等)，Attention。）
 - Batch Normalization？为什么有用？



- BN 应作用在非线性映射前。
- 先作平移缩放，得到均值为 0、方差为 1 的标准分布；然后通过再平移和再缩放，保证模型的表达能力不因规范化而下降。
- 输入层能够因 Normalization 提速，中间层也是一样。主要在于它减少了隐层的协方差转变（Internal Covariate Shift）。ICS 的影响：会影响数据的独立同分布，降低学习速度（导致迁移学习/领域适应问题）。权重伸缩不变性可以有效地提高反向传播的效率。数据伸缩不变性可以有效地减少梯度弥散，简化对学习率的选择。
- 初始化参数如何选择？有哪些权值初始化方法（跳过）？
 - 在最基本的 DNN 中，参数的选择很重要。以 sigmoid 函数（logistic neurons）为例，当 x 的绝对值变大时，函数值越来越平滑，趋于饱和，这个时候函数的导数趋于 0。
 - 一种比较简单、有效的方法是：权重参数初始化从区间均匀随机取值。我们可以通过将其权重向量按其输入的平方根(即输入的数量)进行缩放，从而将每个神经元的输出的方差标准化到 1。

$$\left(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right), \text{ 其中 } d \text{ 是一个神经元的输入数量。}$$

- 四层神经网络，初始化权重为 0，会导致什么情况？（公式说明）

The model is predicting 0 for every example.

In general, initializing all the weights to zero results in the network failing to break symmetry. This means that every neuron in each layer will learn the same thing, and you might as well be training a neural network with $n^{[l]} = 1$ for every layer, and the network is no more powerful than a linear classifier such as logistic regression.

What you should remember:

- The weights $W^{[l]}$ should be initialized randomly to break symmetry.
- It is however okay to initialize the biases $b^{[l]}$ to zeros. Symmetry is still broken so long as $W^{[l]}$ is initialized randomly.
 - 更一般地，如果权重初始化为同一个值，网络就是对称的。设想你在爬山，但身处直线形的山谷中，两边是对称的山峰。由于对称性，你所在之处的梯度只能沿着山谷的方向，不会指向山峰；你走了一步之后，情况依然不变。结果就是你只能收敛到山谷中的一个极大值，而走不到山峰上去。
- 为什么 feature scaling 会使 gradient descent 的收敛更好？【<https://www.zhihu.com/question/37129350>】
 - 不归一化会走弯路（前进方向是切向方向，而错误横截面不是圆形）
- batch size 大小会怎么影响收敛速度
 - Batch 的选择，首先决定的是下降的方向。如果数据集比较小，完全可以采用全数据集，能[更准确地朝向极值所在的方向](#)；另一个极端，就是每次只训练一个样本，即 Batch_Size = 1。这就是在线学习（Online Learning），每次修正方向以各自样本的梯度方向修正，横冲直撞各自为政，难以达到收敛。
- 池化方法作用
 - 1. invariance(不变性)，这种不变性包括 translation(平移)，rotation(旋转)，scale(尺度)
 - 2. 保留主要的特征同时减少参数(降维，效果类似 PCA)和计算量，防止过拟合，提高模型泛化能力
- 如何网络调优（见 ML 优化部分）
- 什么样的资料集不适合用深度学习？（数据集太小 or 没有局部相关性）
- 如何做矩阵的卷积（加权叠加。本质是滤波器，对特定的模式有高的激活）