机器学习：神经网络(Neural Network)与深度学习(Deep Learning)

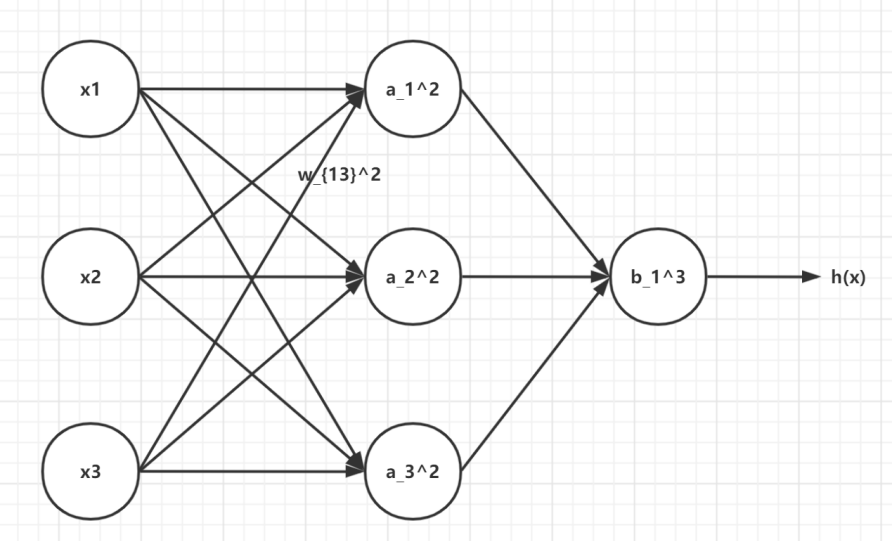
深度学习的任务涉及高维数据的回归和分类，图像与视频序列的分类和预测，文本书写与翻译，情感分析，文字检测与分类，图像风格迁移，图像语义分割，图像降维和去噪，样本对抗生成，图卷积半监督学习，目标检测与跟踪等 cv, nlp, 语音方面的广泛应用。也涉及不同的深度学习框架，PyTorch, TensorFlow, Keras, Caffe, MXNet...

本章只是一些最基础的概念科普，详细的可以阅读个人的博客专栏，带经典论文和网络模型解析，不同算子介绍与推导，算法原理阐述，深度学习的各种应用和 PyTorch 代码实现。

[我的Pytorch教程专栏目录链接]

1. 结构

$X = [x_1, x_2, x_3 \cdots], \hat{y} = h(x)$ 。



符号定义：

$L-$ 神经网络总层数
$w_{jk}^{l}-$ 从第$(l-1)$层的第k个神经元到第l层的第j个神经元的权重
$a_{j}^{l}-$ 第l层第j个神经元的激活函输出
$b_{j}^{l}-$ 第l层的第j个神经元的偏差
$a^{L}-$ 最后一层的输出结果即为神经网络的输出
$z_{j}^{l}-$ 第l层第j个神经元的加权输入

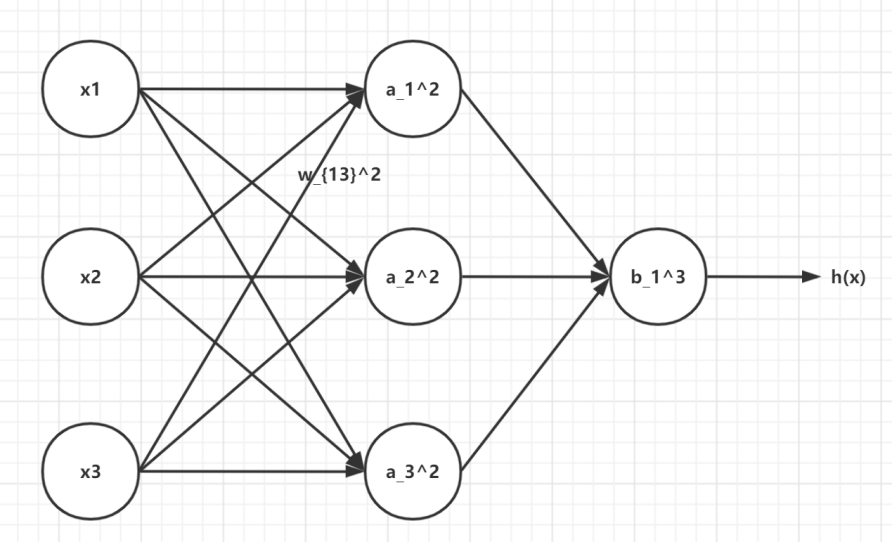
$\sigma(x)=\frac{1}{1+\exp^{-x}}$ 为 $sigmoid$ 激活函数(Lesson 5 监督学习之分类(Perceptron, Fisher, Logistic, Softmax, Bayes)的逻辑回归中，用到了这个函数)。

$$\text{激活函数}：h(x)=\frac{1}{1+\exp^{-x}}, z=\sum_{i=1}^{m}w_ix_i+b$$

$$\begin{matrix}x_1\\x_2\\x_3\end{matrix}\rightarrow\Big(z=w^Tx+b|a=\sigma(z)\Big)\rightarrow a=\hat{y}$$

权重表示，偏差是输入中各元素在决定输出结果中的重要性，偏差是输入的补充项，表示预测输出和标记之间的偏移量。

2. 向量化



$$
\begin{aligned}
a_1^2 &= \sigma(w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3 + b_1^2) \\
a_2^2 &= \sigma(w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3 + b_2^2) \\
a_3^2 &= \sigma(w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3 + b_3^2)
\end{aligned}
$$

即：

第一个节点：

$$
\begin{aligned}
z_1^2 &= w_1^2 x + b_1^2 \\
a_1^2 &= \sigma(z_1^2)
\end{aligned}
$$

第二个节点：

$$
\begin{aligned}
z_2^2 &= w_2^2 x + b_2^2 \\
a_2^2 &= \sigma(z_2^2)
\end{aligned}
$$

第三个节点：

$$
\begin{aligned}
z_3^2 &= w_3^2 x + b_3^2 \\
a_3^2 &= \sigma(z_3^2)
\end{aligned}
$$

∴，第一层可以向量化得到：

$$
\begin{aligned}
x &= (x_1, x_2, x_3)^T \\
w^2 &= \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{bmatrix}, b = (b_1^2, b_2^2, b_3^2) \\
a^2 &= (a_1^2, a_2^2, a_3^2)^T
\end{aligned}
$$

最后可以写成： $a^2 = \sigma(w^2 x + b^2)$ 。

同理，第二层可以向量化得到：

$$
\begin{aligned}
h(x) &= \sigma(w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + w_{13}^3 a_3^2 + b_1^3) \\
w^3 &= (w_{11}^3, w_{12}^3, w_{13}^3) \\
a^2 &= (a_1^2, a_2^2, a_3^2)^T \\
b^3 &= (b_1^3)
\end{aligned}
$$

最后可以写成： $h(x) = \sigma(w^3 a^2 + b^3)$ 。

总结：实际上就是个矩阵堆叠的过程。该结构的前向传播过程分为如下四步：

$$
\begin{aligned}
z^2 &= w^2 x + b_2 \\
a_2 &= \sigma(z^2) \\
z^3 &= w^3 a^2 + b^3 \\
h(x) &= a^3 = \sigma(z^3)
\end{aligned}
$$

3. 前向传播

定义：从输入层通过隐含层一步步计算出输出层的结果。

$Input: x \rightarrow Hidden\ output: a^2 \rightarrow Prediction: h(x)$

4. 损失函数

逻辑回归的损失函数为：

$$
J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log\big(h(x_i)\big) - (1 - y_i) \log(1 - h(x_i)
$$

神经网络的损失函数为：

$$
J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{N} y_{jk} \log\big(h(x_i)_k\big) + (1 - y_{jk}) \log(1 - h(x_i)_k)
$$

其中， k 表示最后一层 N 个神经元中的第 k 个神经元。

5. 反向传播

神经网络通过前向传播输出预测结果，通过反向传播更新网络参数，为了更好理解反向传播，我们引入一个新的中间变量：

$$
\mathrm{d}z_j^l - \text{第}l\text{层第}j\text{个神经元的误差}(error)
$$

$$
\mathrm{d}z_j^l = \frac{\partial J}{\partial z_j^l},\ J\text{为损失函数}
$$

通过引入中间变量，可以总结反向传播算法如下：

1. 计算输出层的误差

$$\mathrm{d}z^L = \frac{\partial J}{\partial z^L} = \nabla_a J \odot \sigma'(z^L)$$

其中 $\odot$ 表示 $Hadamard$ 乘积，用于矩阵或者向量之间点对点的乘法运算。

2. 由第 $l+1$ 层的误差 δ^{l+1} 计算第 l 层的误差 δ^l:

$$\mathrm{d}z^l = \left((w^{l+1})^T \mathrm{d}z^{l+1}\right) \odot \sigma'(z^l)$$

3. 由第 l 层的误差 δ^l 计算偏差梯度：

$$\frac{\partial J}{\partial b_j^l} = \mathrm{d}z_j^l$$

4. 由第 l 层的误差 δ^l 计算权重梯度：

$$\frac{\partial J}{\partial w_{jk}^l} = a_k^{l-1} \mathrm{d}z_j^l$$

对四个重要方程的总结：

1. 计算输出误差：$\mathrm{d}z^L = \frac{\partial J}{\partial z^L} = \nabla_a J \odot \sigma'(z^L)$ 。
2. 向后传播计算每一层的误差：$\mathrm{d}z^l = \left((w^{l+1})^T \mathrm{d}z^{l+1}\right) \odot \sigma'(z^l)$
3. 由误差计算当前层的偏差梯度：$\frac{\partial J}{\partial b_j^l} = \mathrm{d}z_j^l$
4. 由误差计算当前层的权重梯度：$\frac{\partial J}{\partial w_{jk}^l} = a_k^{l-1} \mathrm{d}z_j^l$

每次循环都要重复这样的过程，即不断迭代更新。

6. 梯度下降

输入数据，前向传播，网络输出，计算输出层误差，反向传播计算每一层的误差，计算各层的权重和偏差梯度，梯度下降更新网络参数，如此循环。

$$W_{t+1}^l = W_t^l - \eta \frac{\partial J}{\partial w_{jk}^l}$$

更多梯度下降优化算法可参考Lesson 3 优化方法基础。

7. CNN 卷积神经网络

用于图像处理和计算机视觉。

图像与视频序列的分类和预测，图像语义分割，目标检测与跟踪，样本对抗生成，图卷积半监督学习等。

$Convolution\ layer, Pooling\ layer, Fully\ Connected\ layer$ 。

8. RNN 循环神经网络

用于自然语言处理，有时间序列关系。

文本书写与翻译，情感分析，文字检测与分类。

9. 无监督学习网络的应用

对抗神经网络，图像风格迁移，图像上色，图像自编码器(降维和去噪)。

10. 梯度消失

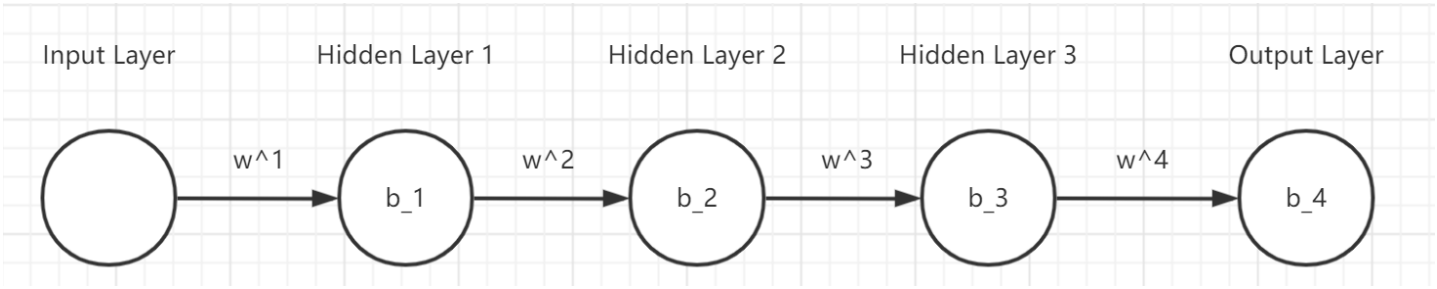
神经网络通过非线性隐藏层堆叠可以拟合复杂的函数，但是随着层数的增加，神经网络会出现学习缓慢甚至无法学习的情况，这就是梯度消失问题。

通过 $MNIST$ 数据集训练神经网络对比，每层激活函数用 $sigmoid \in (0,1)$ ，伴随着隐藏层的增加，梯度出现上升现象。

随着网络模型不断训练，我们发现靠前的层比靠后的层梯度要小，并且随着网络隐藏层的增加，这种问题愈发明显。深度神经网络中，隐藏层无法学习的情况被称为梯度消失问题。

深度神经网络的学习提督是不稳定的，这种不稳定性是深度神经网络基于梯度下降学习网络参数的一个最基本的问题。

为了探究为什么会梯度消失，以一个三层隐藏层的神经网络为例：



第一个隐藏层偏差变化率：

$$\frac{\partial J}{\partial b^1} = \sigma'(z^1) \times w^2 \times \sigma'(z^2) \times w^3 \times \sigma'(z^3) \times w^4 \times \sigma'(z^4) \times \frac{\partial J}{\partial a^4}$$

对于 $sigmoid$ 函数的导数，在 $z = 0$ 的时候其最大梯度为 0.25 。

由链式法则求出来的梯度公式中，越靠前的梯度收到后面层的梯度影响越大，加上 $sigmoid$ 函数的梯度时总小于 1 ，所以靠前的神经网络层容易出现梯度变得极小的情况，没有学习效果，这就是梯度消失。

$\sigma'(z^2) < \frac{1}{4}, \sigma'(z^3) < \frac{1}{4}$ ，带入第一个隐藏层变化率，会发现结果很小，导致梯度不更新。

11. 梯度爆炸问题

$$\frac{\partial J}{\partial b^1} = \sigma'(z^1) \times w^2 \times \sigma'(z^2) \times w^3 \times \sigma'(z^3) \times w^4 \times \sigma'(z^4) \times \frac{\partial J}{\partial a^4}$$

令 $w^2 = w^3 = w^4 = 100$ 。通过链式法则权重不断相乘是的前面曾的变化率变得非常大，这就是梯度爆炸。但是这种现象相对少见。

12. $ReLU$

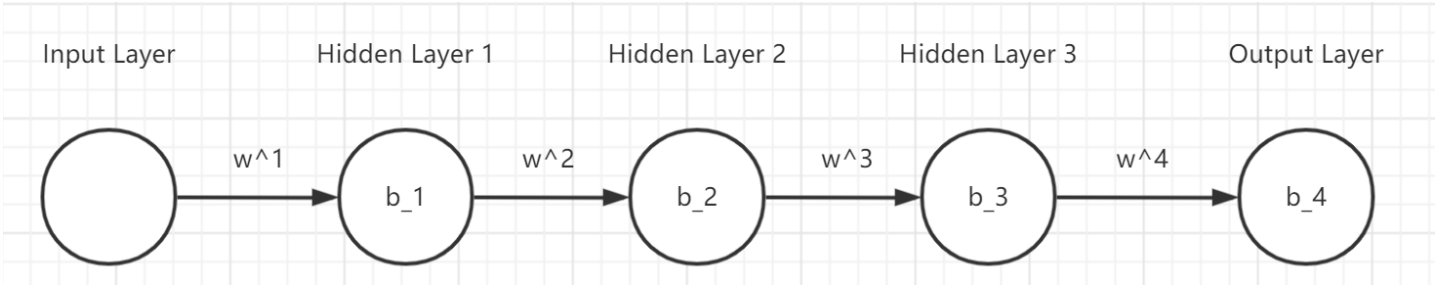
引入校正线性单元，即$Rectified\ Linear\ Units$ 。

$$A(x) = \max(0, x)$$

其梯度为：

$$\frac{\mathrm{d}A(x)}{\mathrm{d}x} = \begin{cases} 1, & if\ x > 0 \\ 0, & if\ x \leqslant 0 \end{cases}$$

在 x 大于 0 的时候， $ReLU$ 的导数始终为 1 。在将其作为神经网络每层的激活函数时，及时经过层层网络的传播也可以保证前面层的变化率可以保持相对稳定，这也是为什么 $ReLU$ 可以使得深层神经网络更加容易训练的原因。



$$\frac{\partial J}{\partial b^1} = A'(z^1) \times w^2 \times A'(z^2) \times w^3 \times A'(z^3) \times w^4 \times A'(z^4) \times \frac{\partial J}{\partial a^4}$$

$ReLU$ 的其他好处：

1. $ReLU$ 通过分段使得函数本身仍然是非线性的，不会减弱神经网络的表现能力。
2. 小于 0 的梯度为 0 使得部分神经元不会被激活，使得神经网络更加稀疏与高效。
3. $ReLU$ 运算简单，比 $sigmoid$ 函数更节约计算开销。

它也存在缺点，小于 0 的分段梯度同样为 0 造成，这意味着输出结果为负的神经元将停止对误差的响应，可能会导致许多神经元的"死亡"。

解决方案：$LeakyReLU$, $PReLU$, $RReLU$, $SoftPlus$ 。

13. 参数初始化

除了深度神经网络存在的梯度不稳定的情况之外，还有许多其他的问题。比如说参数初始化使得网络处于不同的七点，往往带来截然不同的训练结果。有些初始化会导致网络陷入局部最优，选择合适的初始化方法可以减少这种情况。

14. 神经网络的稀疏性($sparse$)

这部分是我当时跟 MIT 的 $Prof.Mark$ 做暑研的时候，他告诉我的一些相关经验。

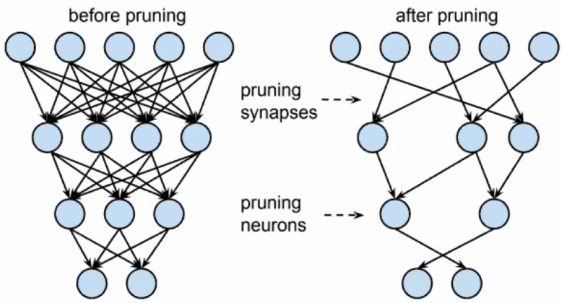
$Pruning$ 剪枝后，每层变成了稀疏矩阵，加速效率(我感觉就是 $Dropout$ 操作)。

坏处是会损失一部分的 $performance$ 。

还有一种稀疏方式是 $Binary$ ，即让网络权重只有 $\{+1,-1\}$ 。相当于将 $float32$ 改成 int ，即 $1\ bit$ 。

Questions

- Sparsity of the network?
  - Pruning
  - Binary Neural Networks



Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1

Matthieu Courbariaux[*1] MATTHIEU.COURBARIAUX@GMAIL.COM
Itay Hubara[*2] ITAYHUBARA@GMAIL.COM
Daniel Soudry[3] DANIEL.SOUDRY@GMAIL.COM
Ran El-Yaniv[2] RANI@CS.TECHNION.AC.IL
Yoshua Bengio[1,4] YOSHUA.UMONTREAL@GMAIL.COM

[1]Université de Montréal
[2]Technion - Israel Institute of Technology
[3]Columbia University
[4]CIFAR Senior Fellow
*Indicates equal contribution. Ordering determined by coin flip.

15. 指数爆炸解决?

Questions

Type	Smallest Positive Value	Largest value	Precision
float	1.17549 x 10^-38	3.40282 x 10^38	6 digits
double	2.22507 x 10^-308	1.79769 x 10^308	15 digits

- logsumexp

$$\log\left(\frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}\right) = \log(e^{x_j}) - \log\left(\sum_{i=1}^{n} e^{x_i}\right)$$
$$= x_j - \log\left(\sum_{i=1}^{n} e^{x_i}\right) \quad (1)$$

$$\begin{aligned} LogSumExp(x_1 \dots x_n) &= \log\left(\sum_{i=1}^{n} e^{x_i}\right) \\ &= \log\left(\sum_{i=1}^{n} e^{x_i - c} e^{c}\right) \\ &= \log\left(e^{c} \sum_{i=1}^{n} e^{x_i - c}\right) \\ &= \log\left(\sum_{i=1}^{n} e^{x_i - c}\right) + \log(e^{c}) \\ &= \log\left(\sum_{i=1}^{n} e^{x_i - c}\right) + c \quad (2) \end{aligned}$$

It turns out $max(x_1 \dots x_n)$ works really well.