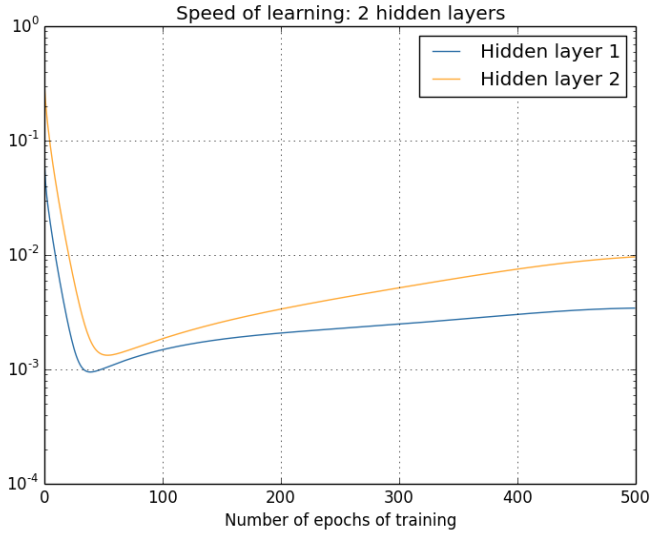
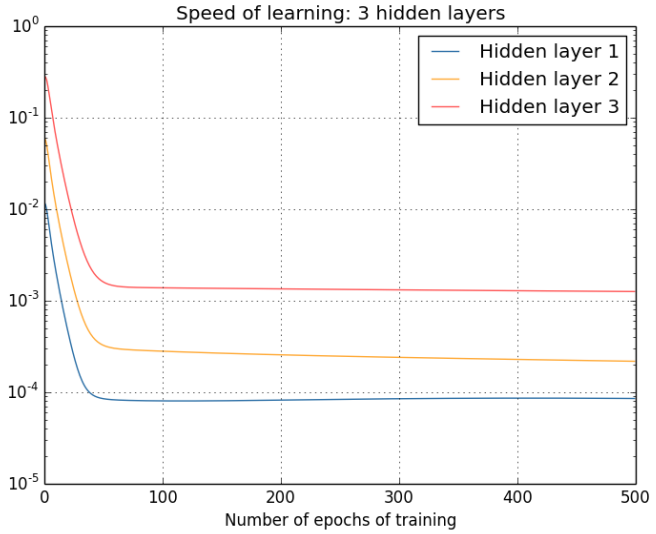
# 论文阅读

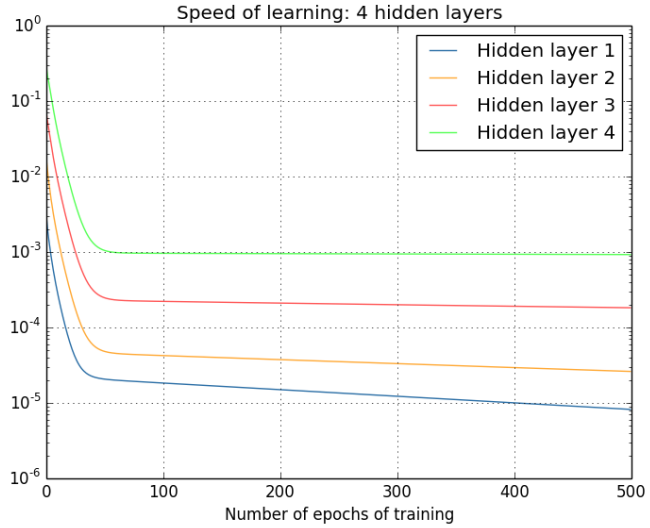
姓名：魏亚东
人工智能与机器人研究所
2017年7月

# CONTENS

# 1.1神经网络梯度消失问题



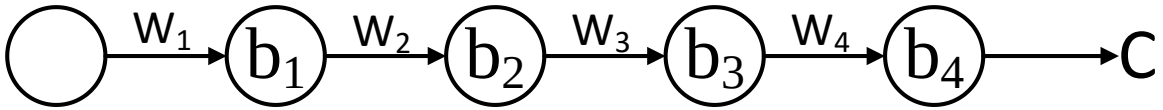




- 前面的隐层的学习速度低于后面的隐层的学习速度，这种现象普遍存在于神经网络中，叫做**梯度消失问题。**
- 另外一种情况是内层的梯度被外层大很多，叫做**激增的梯度问题**。
- 更加一般地说，在深度神经网络中的梯度是不稳定的，在前面的层中或会消失，或会激增。这种**不稳定性**才是深度神经网络中基于梯度学习的根本问题。

# 1.2梯度消失原因

- 先看一个极简单的深度神经网络：每一层都只有一个单一的神经元。如下图：




- Loss Function:

$$C = \frac{1}{2}(\hat{Y} - Y)^2 (1)$$

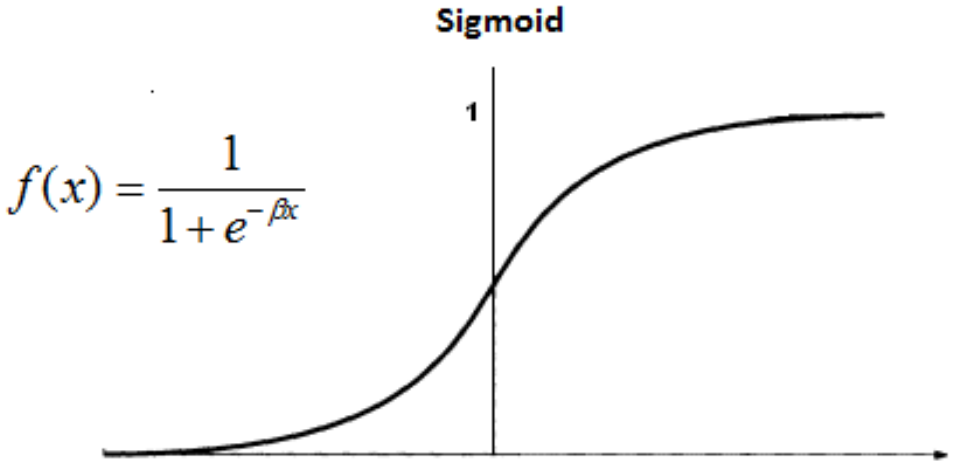
- 输出求解：

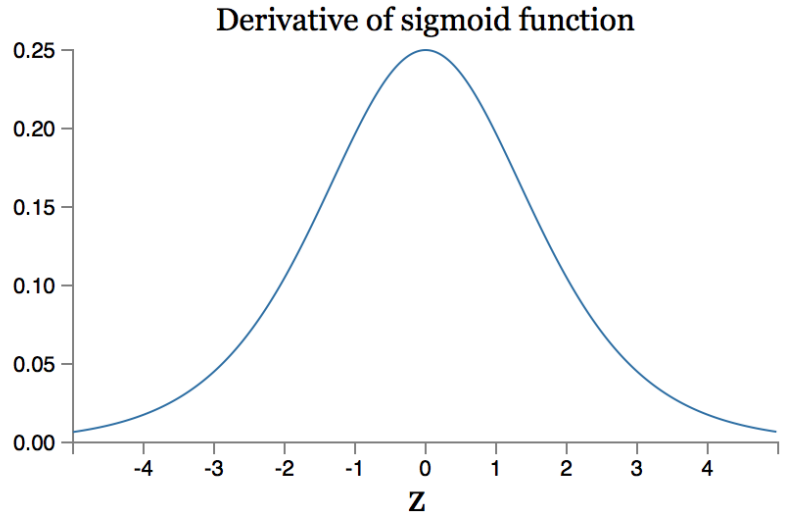
$$Y = \sigma(W_4\sigma(W_3\sigma(W_2\sigma(X_0W_1 + b_1) + b_2) + b_3) + b_4)(2)$$

- 代价函数C对偏置$b_1$的偏导数的结果计算如下：

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4} (3)$$

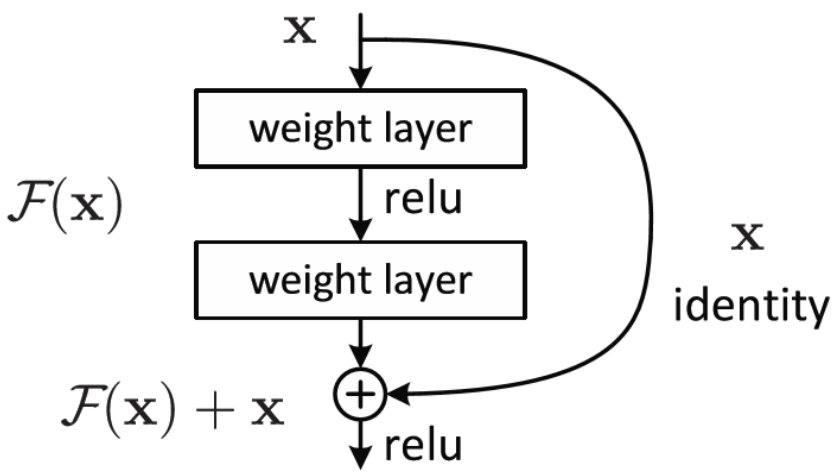
# 1.2梯度消失原因




- 该导数在σ′(0) = 1/4时达到最高。现在，如果我们使用标准方法来初始化网络中的权重，那么会使用一个均值为0 标准差为1 的高斯分布。因此所有的权重通常会满足$|w_j|<1$。从而有$w_j\sigma'(z_j) < 1/4$。这就是梯度消失的本质原因。
- 只要是sigmoid函数的神经网络都会造成梯度更新的时候极其不稳定，产生梯度消失或者激增问题。

# 1.3梯度消失问题的解决方案(残差网络)




$$\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l),$$
$$\mathbf{x}_{l+1} = f(\mathbf{y}_l),$$

- 上面公式中：h 表示 shortcut 使用什么形式的变换
- f 为激活函数，一般是ReLU

- 当h(x)=x，yl=h(xl)的时候，可以列出梯度求解公式：

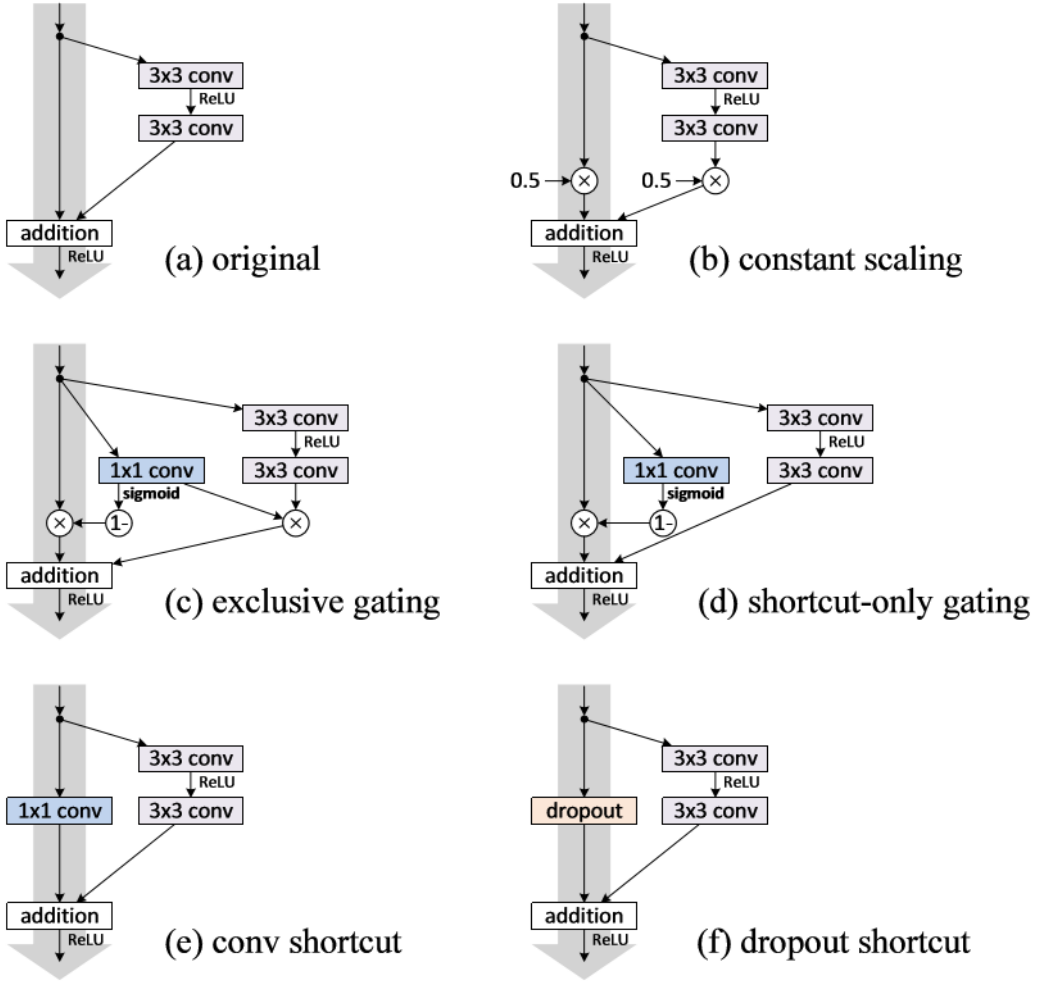
$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right)$$

- 可以看到，在ResNet的梯度求解过程中，并没有出现正常神经梯度计算过程中出现的累乘的问题，所以说梯度消失的问题得到解决。
- 当然在H不断变化的时候梯度求解公式也会发生变化，但是后面证明得到当H(X)=X的时候网络准确率是最高的

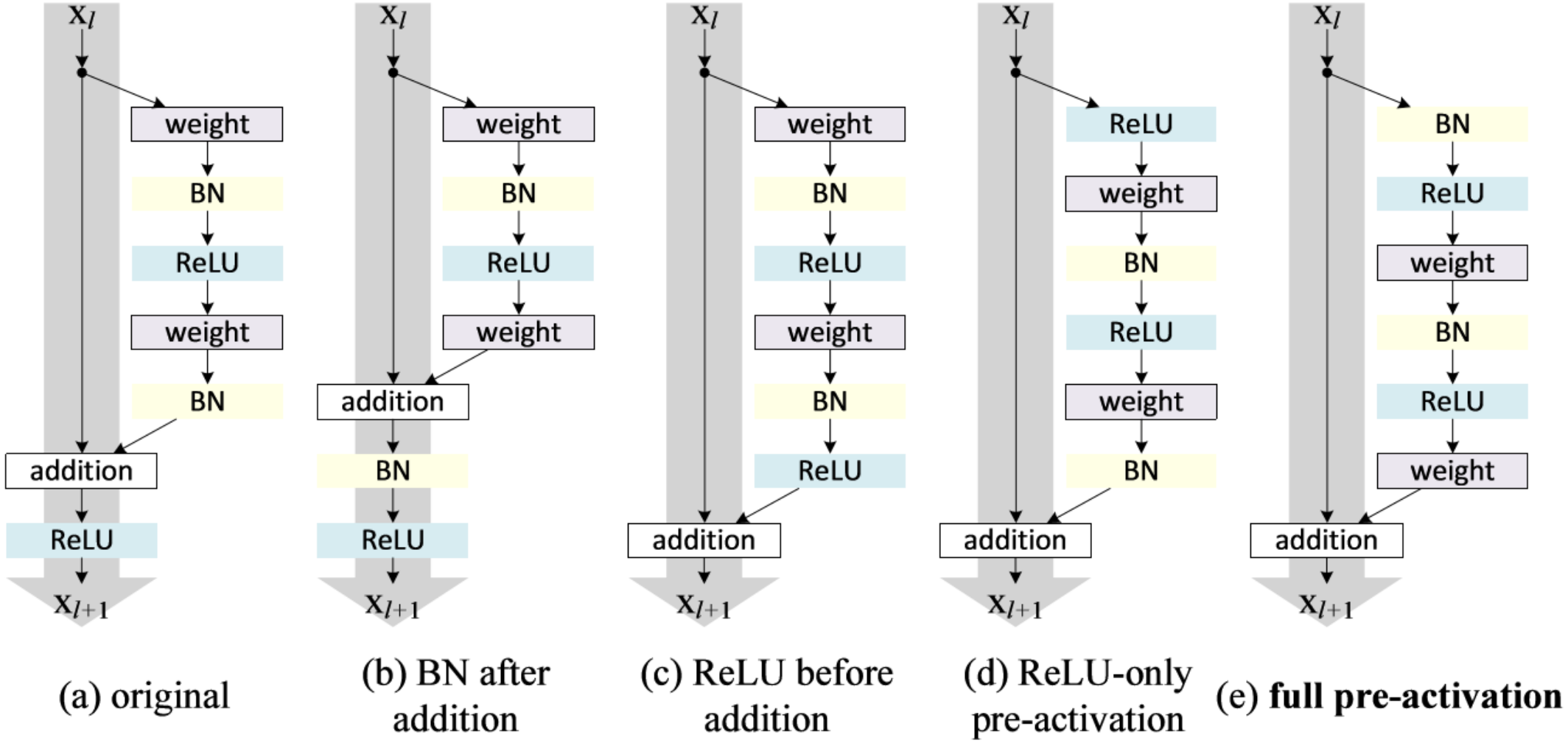
# 1.4ResNet



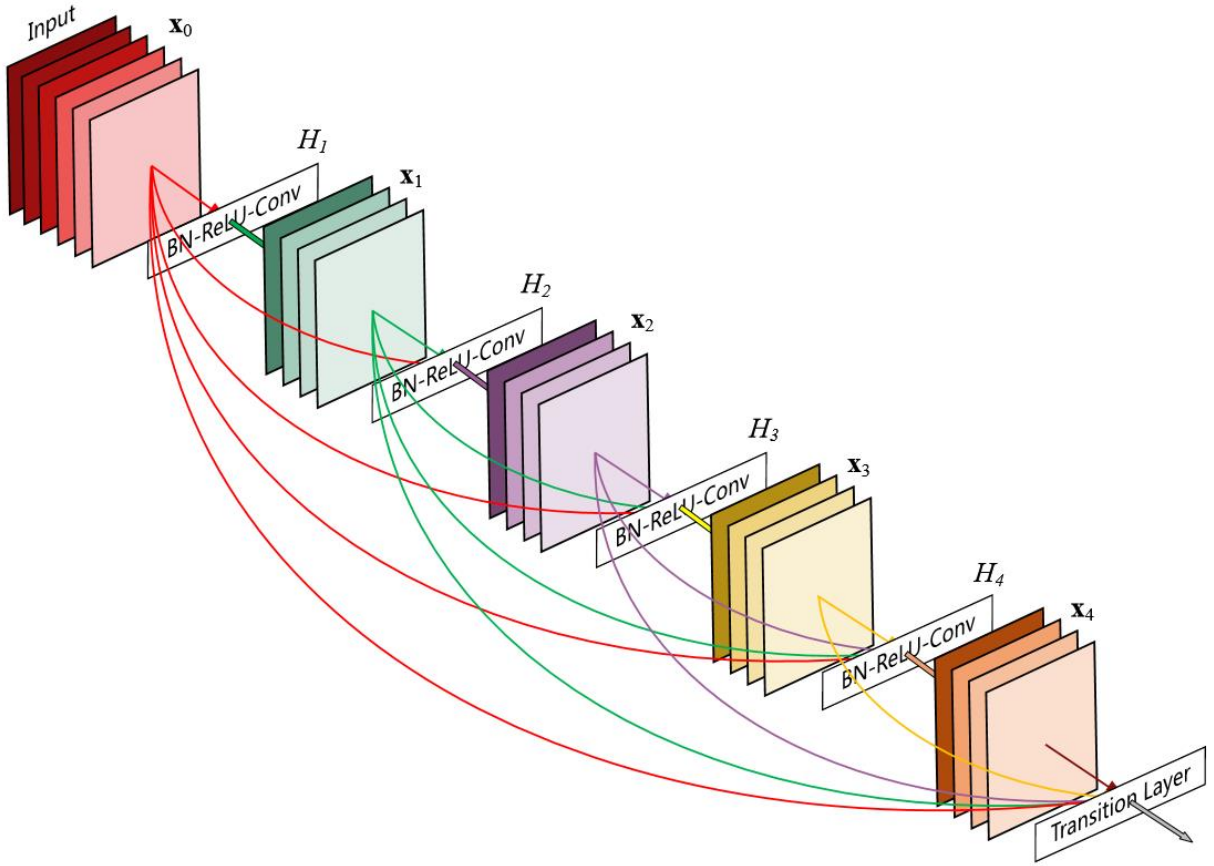

| case | Fig. | on shortcut | on $\mathcal{F}$ | error (%) | remark |
|---|---|---|---|---|---|
| original [1] | Fig. 2(a) | 1 | 1 | **6.61** | |
| constant scaling | Fig. 2(b) | 0 | 1 | fail | This is a plain net |
| | | 0.5 | 1 | fail | |
| | | 0.5 | 0.5 | 12.35 | frozen gating |
| exclusive gating | Fig. 2(c) | $1-g(\mathbf{x})$ | $g(\mathbf{x})$ | fail | init $b_g$=0 to −5 |
| | | $1-g(\mathbf{x})$ | $g(\mathbf{x})$ | 8.70 | init $b_g$=-6 |
| | | $1-g(\mathbf{x})$ | $g(\mathbf{x})$ | 9.81 | init $b_g$=-7 |
| shortcut-only gating | Fig. 2(d) | $1-g(\mathbf{x})$ | 1 | 12.86 | init $b_g$=0 |
| | | $1-g(\mathbf{x})$ | 1 | 6.91 | init $b_g$=-6 |
| 1×1 conv shortcut | Fig. 2(e) | 1×1 conv | 1 | 12.22 | |
| dropout shortcut | Fig. 2(f) | dropout 0.5 | 1 | fail | |

- **在ResNet中，对于h函数提出了很多种模型，最终发现h(x)=x这个模型拥有最高的准确度，是已知方案中最好的也是最简单的一组方案**

# 1.4ResNet

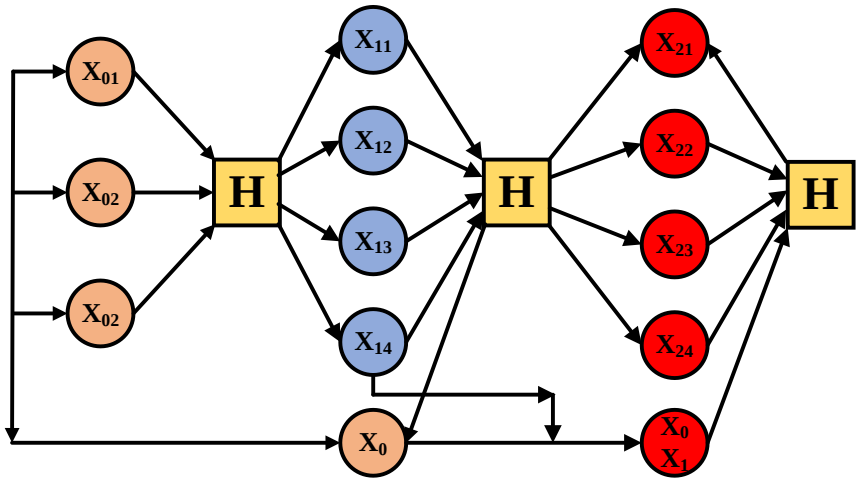

$x_l$
weight
BN
ReLU
weight
BN
addition
ReLU
$x_{l+1}$
(a) original
(b) BN after addition
(c) ReLU before addition
(d) ReLU-only pre-activation
(e) full pre-activation

# 2.1Densely Connect Network

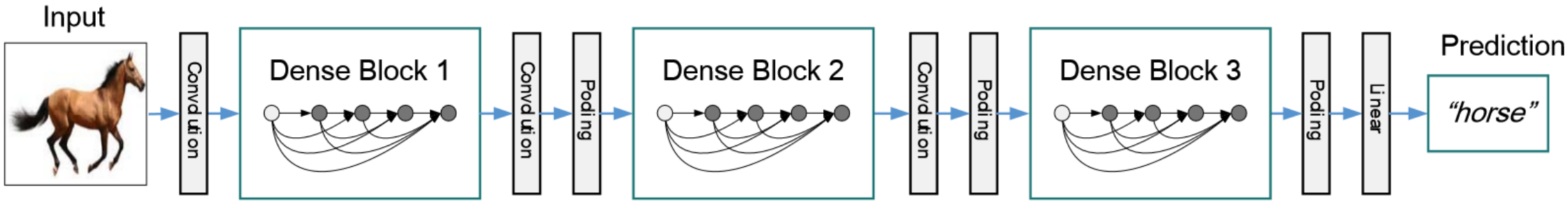



- 稠密连接：每层以之前层的输出为输入，对于有L层的传统网络，一共有L个连接，对于DenseNet，则有L(L+1)/2。
- 这篇论文主要参考了Highway Networks，Residual Networks (ResNets)以及GoogLeNet，通过加深网络结构，提升分类结果。加深网络结构首先需要解决的是梯度消失问题，解决方案是：尽量缩短前层和后层之间的连接。H4层可以直接用到原始输入信息X0，同时还用到了之前层对X0处理后的信息，这样能够最大化信息的流动。反向传播过程中，X0的梯度信息包含了损失函数直接对X0的导数，有利于梯度传播。

# 2.1Densely Connect Network

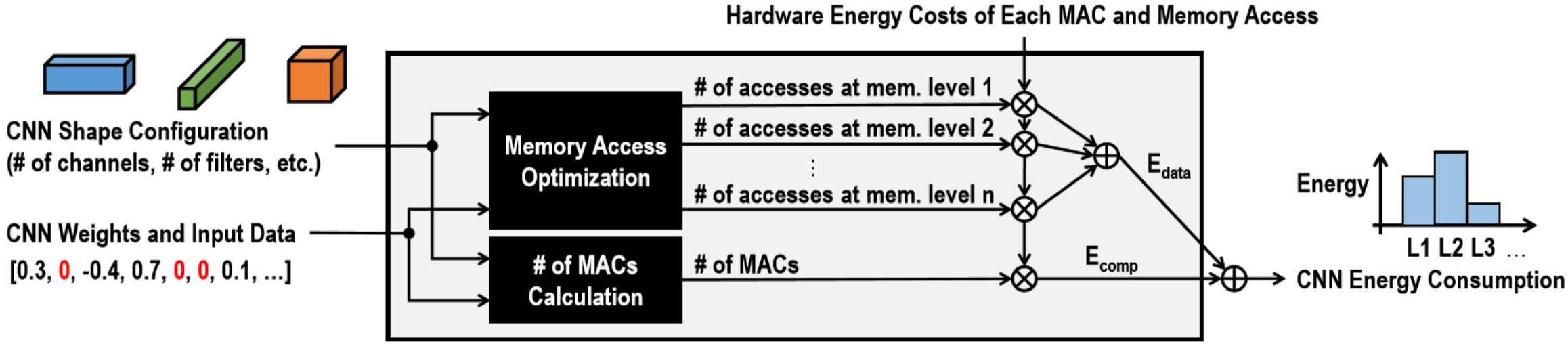


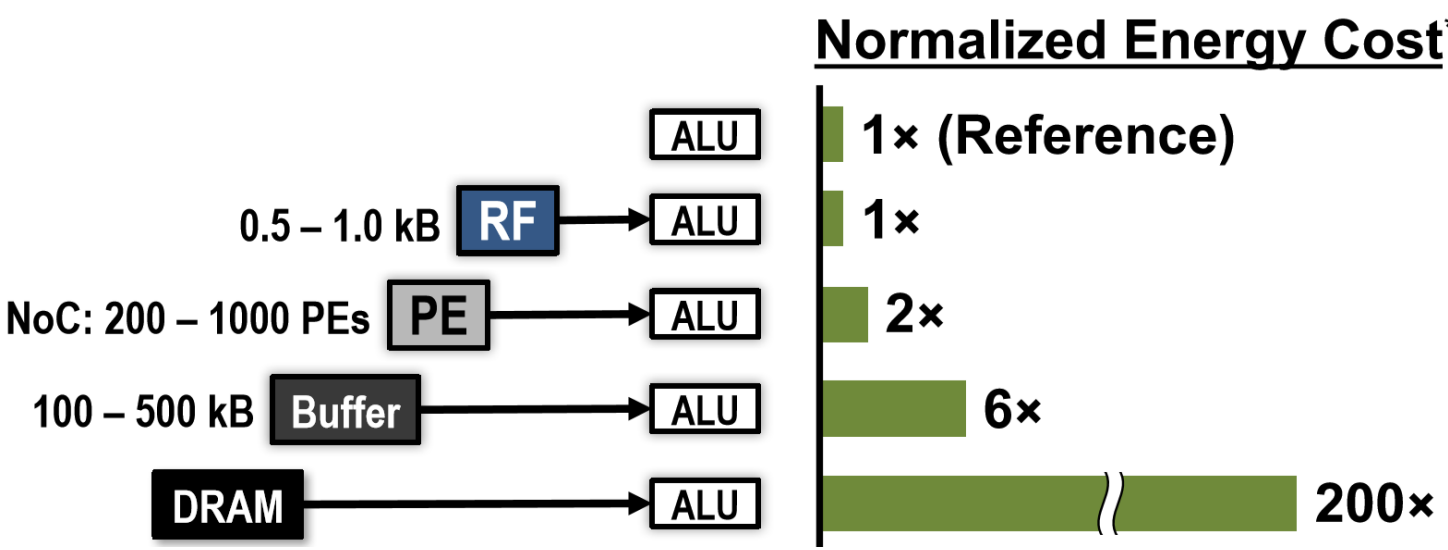

- 左图表现了我对Densely Connect Network的理解，假设有K0=3个输入，且每经过一个H产生K=4个输入，需要注意的问题是，后面的层的H的输入是前面所有层的输入，并不是相加的关系，而是直接的使用于计算之中、

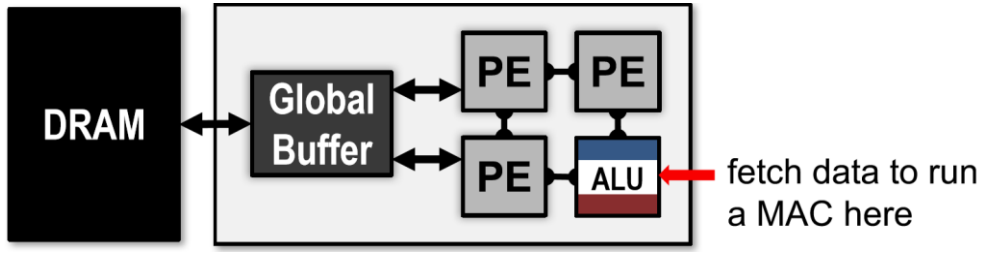



- 前面已经提到了，稠密连接的连接数是L(L+1)/2,也就是说如果我们把完整的神经网络用一个dense block 进行连接会导致连接的数量过多。
- 所以文中提出了将局部的层连成block,然后不同的block之间按照顺序进行连接

# 3.1一种基于能量高效的神经网络剪枝方案

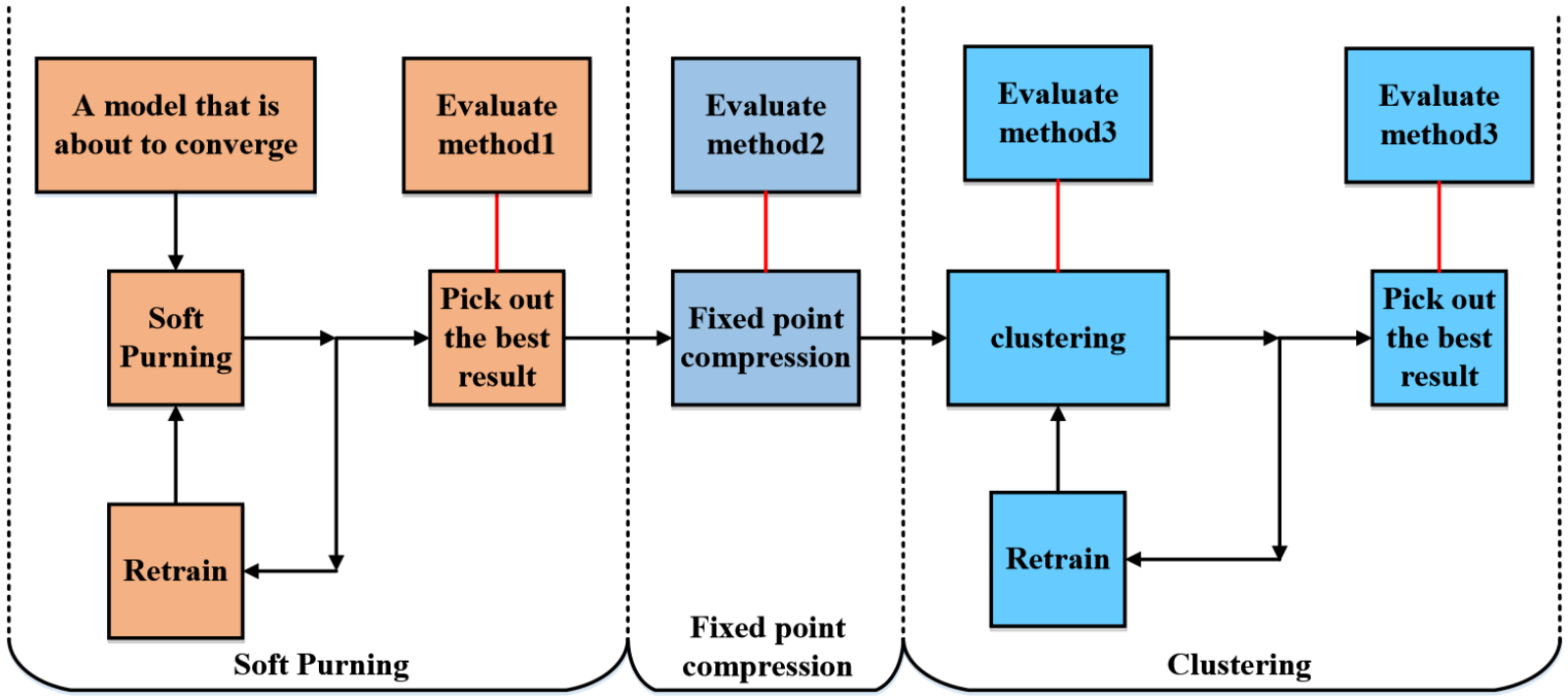


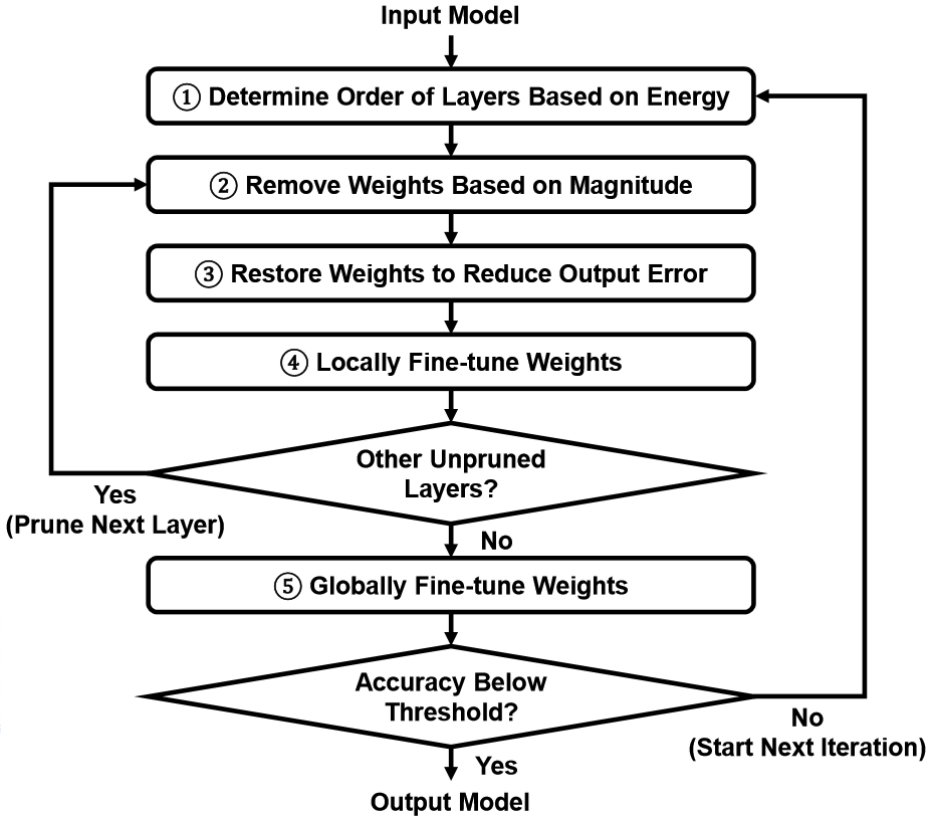




- 右面的是论文中提出的基于能量的剪枝的方法，基本的思路是不同的层剪枝次序不同（按照能量消耗由大到小进行剪枝），其中能量消耗分为两个部分，数据存取和数据计算。

# 3.1一种基于能量高效的神经网络剪枝方案

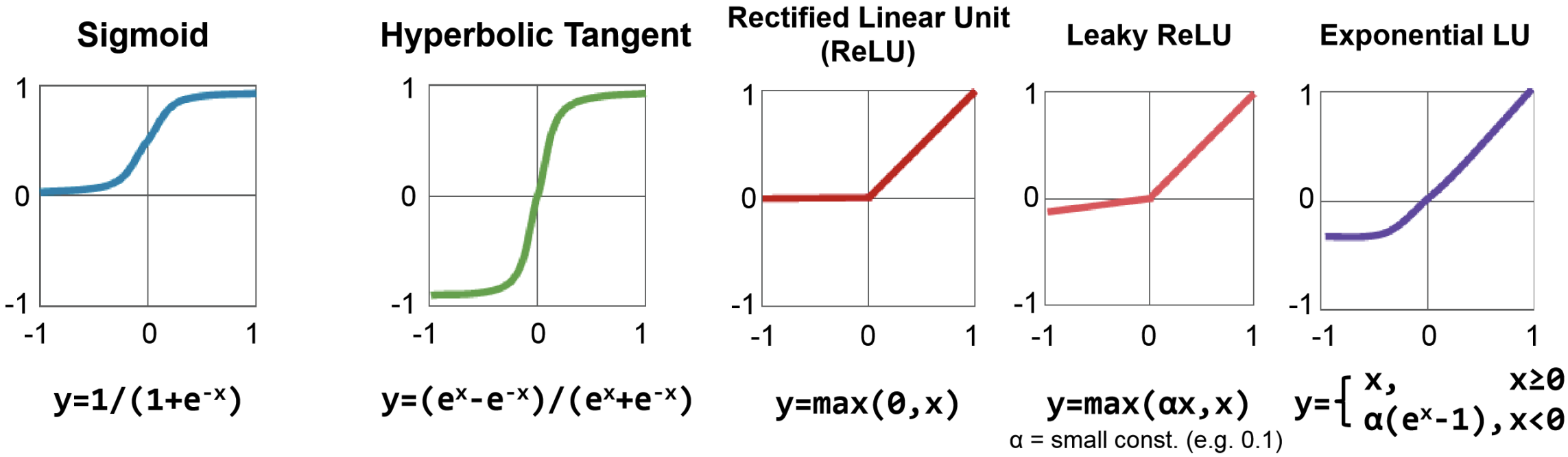



- 1：按照能量大小决定剪枝的顺序 2：按照参数大小进行第一步剪枝
- 3：按照一定的策略存储一些对神经网络准确率比较有用的参数
- 4：保存好参数之后进行独层训练 5：对整个网络进行训练优化

$$\tilde{A}_i = \arg\min_{\hat{A}_i} \left\| \hat{Y}_i - X_i \hat{A}_i \right\|_p^p, \quad \text{subject to } \left\| \hat{A} \right\|_0 \leqslant q,$$

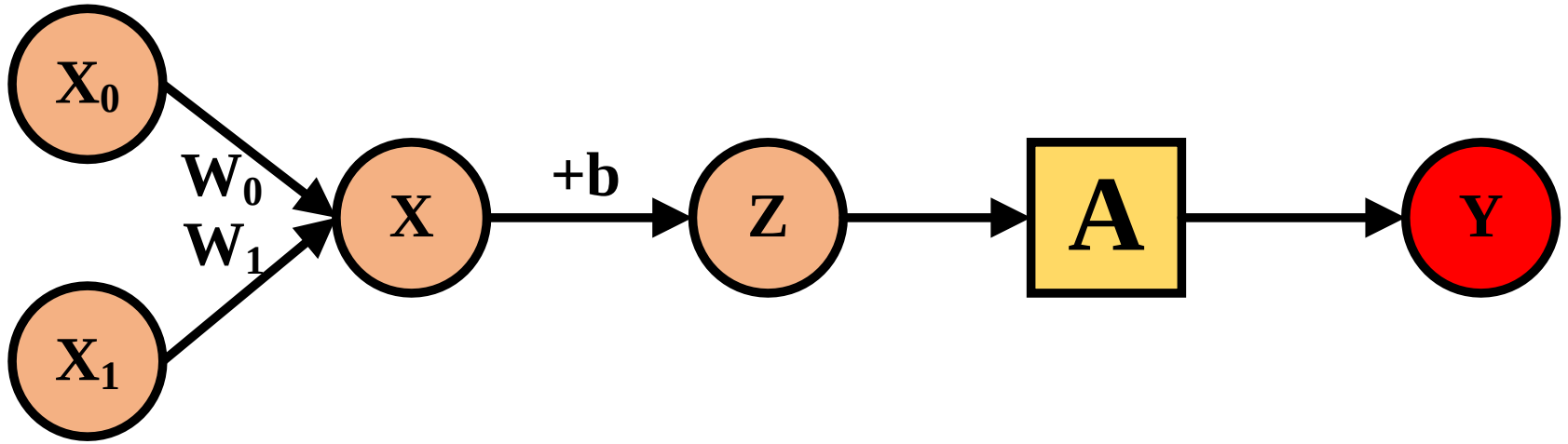
# 4.1为什么sigmoid函数被替代




**激活函数的种类多种多样，从刚开始的Sigmoid的，到后面的Tanh，ReLU，Leaky ReLU。目前激活函数方面最新的进展是SeLU(Self-Normalizing Neural Networks),SeLU可以使得训练出来的神经网络参数符合标准的正态分布。**

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

# 4.1为什么sigmoid函数被替代




**输出求解方式：**

$$X = X_0 W_0 + X_1 W_1$$

$$Z = X + b$$

$$Y = A(Z)$$

$$C = C(Y)$$

**梯度求解公式：** $$\frac{\partial C}{\partial W_0} = \frac{\partial C}{\partial Y} \times \frac{\partial Y}{\partial Z} \times \frac{\partial Z}{\partial X} \times \frac{\partial X}{\partial W_0}$$

**如果激活函数是sigmoid，由导数的计算公式得到Dsigmoid(x)=Sigmoid(x)*(1-Sigmoid(x)),那么在x比较大或者比较小的时候，返回的梯度都接近于0，这导致神经网络收敛速度变慢，并且容易陷入局部最小值**

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \left( 1 + \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i) \right)$$

**第二个原因是Sigmoid函数导数的最大值也比较小，更容易造成梯度消失**

# 5.1三值化原理推导

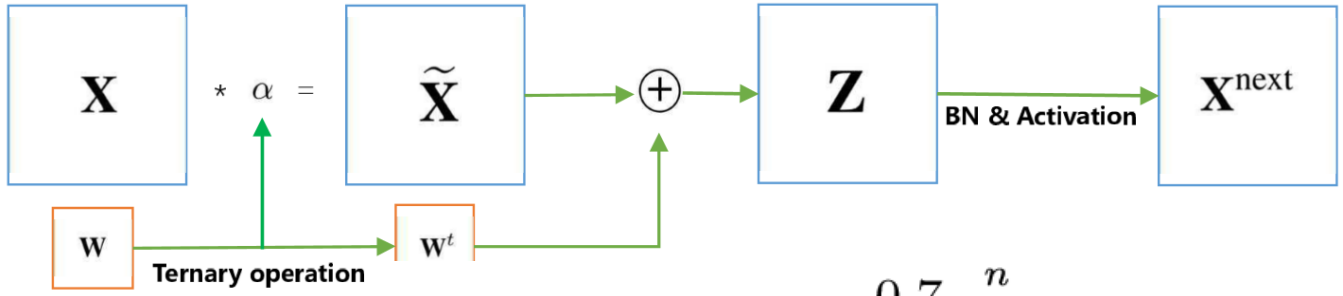
$$\begin{cases} \alpha^*, \mathbf{W}^{t*} = \underset{\alpha, \mathbf{W}^t}{\operatorname{argmin}} J(\alpha, \mathbf{W}^t) = ||\mathbf{W} - \alpha \mathbf{W}^t||_2^2 \\ s.t. \qquad \alpha \geq 0,\ \mathrm{W}_i^t \in \{-1, 0, 1\},\ i = 1, 2, \ldots, n. \end{cases}$$

$$\mathrm{W}_i^t = f_t(\mathrm{W}_i|\Delta) = \begin{cases} +1, & \text{if } \mathrm{W}_i > \Delta \\ 0, & \text{if } |\mathrm{W}_i| \leq \Delta \\ -1, & \text{if } \mathrm{W}_i < -\Delta \end{cases}$$

$$\begin{aligned} \alpha^*, \Delta^* &= \underset{\alpha \geq 0, \Delta > 0}{\operatorname{argmin}} ||\mathbf{W} - \alpha \mathbf{W}^t||_2^2 \\ &= \underset{\alpha \geq 0, \Delta > 0}{\operatorname{argmin}} \Big( \sum_{i:|\mathrm{W}_i| > \Delta} ||\mathrm{W}_i| - \alpha|^2 + \sum_{i:|\mathrm{W}_i| \leq \Delta} \mathrm{W}_i^2 \Big) \\ &= \underset{\alpha \geq 0, \Delta > 0}{\operatorname{argmin}} \big( |\mathrm{I}_\Delta| \alpha^2 - 2 (\sum_{i \in \mathrm{I}_\Delta} |\mathrm{W}_i|) \alpha + c \big) \end{aligned}$$

$$\alpha_\Delta^* = \frac{1}{|\mathrm{I}_\Delta|} \sum_{i \in \mathrm{I}_\Delta} |\mathrm{W}_i|.$$

$$\begin{aligned} \Delta^* &= \underset{\Delta > 0}{\operatorname{argmin}} ||\mathbf{W} - \alpha_\Delta^* \mathbf{W}^t||_2^2 \\ &= \underset{\Delta > 0}{\operatorname{argmin}} -\frac{1}{|\mathrm{I}_\Delta|} \Big( \sum_{i \in \mathrm{I}_\Delta} |\mathrm{W}_i| \Big)^2 \\ &= \underset{\Delta > 0}{\operatorname{argmax}} \frac{1}{|\mathrm{I}_\Delta|} \Big( \sum_{i \in \mathrm{I}_\Delta} |\mathrm{W}_i| \Big)^2 \end{aligned}$$




➢ **经验公式：** $\Delta^* = 0.7 \cdot E(|\mathbf{W}|) \approx \frac{0.7}{n} \sum_{i=1}^{n} |\mathrm{W}_i|$

**Forward propagation:**
**for** $l$ = 1 to $L$ **do**
  Get the threshold value $\Delta_l^*$ by (16) with the filters in the $l$th layer.
  Compute the ternary weights $\mathbf{W}_l^t$ by (6) for the $l$th layer.
  Get the scaling factor $\alpha_l^*$ by (8).
  Rescale the inputs of $i$th layer $\mathbf{X}^{l-1}$ with scaling factor $\alpha_l^*$, thus, $\widetilde{\mathbf{X}}^{l-1} = \alpha^* \mathbf{X}^{l-1}$.
  Compute $\mathbf{Z}^l$ with input $\widetilde{\mathbf{X}}^{l-1}$ and the ternary-valued weights $\mathbf{W}_l^t$.
  Perform Batch Normalization and non-linear activation to get $\mathbf{X}^l$.
**end for**
Compute the cost $C$ with forward outputs $\mathbf{X}^L$ and the targets $\mathbf{Y}$.
**Backward propagation:**
Initialize output layer's activation gradient $\frac{\partial C}{\partial \mathbf{X}^L}$.
**for** $l$ = $L$ to 2 **do**
  Compute $\frac{\partial C}{\partial \mathbf{X}^{l-1}}$ knowning $\frac{\partial C}{\partial \mathbf{X}^l}$ and $\mathbf{W}_l^t$.
**end for**
**Update parameters and learning rate:**
**for** $l$ = 1 to $L$ **do**
  Compute $\frac{\partial C}{\partial \mathbf{W}_l^t}$ according to $\frac{\partial C}{\partial \mathbf{X}^{l+1}}$ and $\mathbf{X}^l$.
  $\mathbf{W}_l^{new} \leftarrow \mathbf{W}_l - \eta \frac{\partial C}{\partial \mathbf{W}_l^t}$
**end for**
Update learning rate $\eta^{new}$ according to any learning rate scaling method.

# 6.1多值化推导以及结果展示

$$W_i^t = f_t(W_i \mid \Delta_1, \Delta_2) = \begin{cases} +2 & W_i > \Delta_2 \\ +1 & \Delta_2 > W_i > \Delta_1 \\ 0 & |W_i| < \Delta_1 \\ -1 & -\Delta_2 < W_i < -\Delta_1 \\ -2 & W_i < -\Delta_2 \end{cases}$$

$$\alpha^*, \Delta_1^*, \Delta_2^* = \arg\min_{\alpha \geq 0, \Delta_1 > 0, \Delta_2 > 0} \| W - \alpha W^t \|_2^2$$

$$= \arg\min_{\alpha \geq 0, \Delta_1 > 0, \Delta_2 > 0} \left( \sum_{i:|W_i| \leq \Delta_1} W_i^2 + \sum_{i:|W_i| \in (\Delta_1, \Delta_2)} \| W_i \| - \alpha |^2 + \sum_{i:|W_i| \geq \Delta_2} \| W_i \| - 2\alpha |^2 \right)$$

$$= \arg\min_{\alpha \geq 0, \Delta_1 > 0, \Delta_2 > 0} \left( | I_{\Delta_1} | \alpha^2 - 2\left( \sum_{i \in I_{\Delta_1}} | W_i | \alpha \right) + 4 | I_{\Delta_2} | \alpha^2 - 4\left( \sum_{i \in I_{\Delta_2}} | W_i | \alpha \right) + c \right)$$

$$\alpha_\Delta^* = \frac{1}{(I_{\Delta_1} + 4 I_{\Delta_2})} \left( \sum_{i \in I_{\Delta_1}} | W_i | + 2 \sum_{i \in I_{\Delta_2}} | W_i | \right)$$

delta1 和 delta2 大小的选取：关于 delta1 和 delta2 大小的选取，采用假设

均匀分布：

$$\Delta_1^* = 0.2a$$

$$\Delta_1^* = 0.6a$$

高斯分布：

$$\Delta_1^* = 0.43\sigma$$

$$\Delta_1^* = 1.28\sigma$$

经验公式：

按照均匀分布来算的话：scale1=0.4，scale2=1.2。按照高斯分布来算的话，scale1=0.54，scale2=1.6，所以最终如果 w 的分布不能确定的话，我们可以取经验公式：

Scale1=0.5，scale2=1.4；

$$\Delta_1^* = scale_1 E(|W|) \approx \frac{0.5}{n} \sum_{i=1}^{n} |W_i|$$

$$\Delta_2^* = scale_2 E(|W|) \approx \frac{1.4}{n} \sum_{i=1}^{n} |W_i|$$

- **基于以上的推导，在MNIST里面做的实现，主要是复现了三值化还有实现了多值化，下面是目前实现的结果（因为没有加入BN可能网络的准确率没有高于原来的准确率稍有遗憾）**

原网络：

12@5x5CONV(Sigmoid)+2x2MAXP+12@5x5CONV(Sigmoid)+2x2MAXP+512(Sigmoid)+SVM(不收敛)

三值化：

12@5x5CONV(Sigmoid)+2x2MAXP+12@5x5CONV(Sigmoid)+2x2MAXP+512(Sigmoid)+SVM(96.25%)

原网络全连接激活函数换成 ReLU：

12@5x5CONV(Sigmoid)+2x2MAXP+12@5x5CONV(Sigmoid)+2x2MAXP+512(ReLU)+SVM(98.45%)

三值化：

12@5x5CONV(Sigmoid)+2x2MAXP+12@5x5CONV(Sigmoid)+2x2MAXP+512(ReLU)+SVM(97.90%)

五值化：

12@5x5CONV(Sigmoid)+2x2MAXP+12@5x5CONV(Sigmoid)+2x2MAXP+512(ReLU)+SVM(98.22%)

- **多值化网络具有的表达能力更强，拥有比Binary还有Ternary更好的泛化能力，我们未来可以基于Densely Connect Network 做多值化和压缩。**

# 谢谢！

人工智能与机器人研究所

2017年6月