



## 第七讲：预训练、激活函数、权重初始化、块归一化

Unsupervised Pre-training, Better activation functions, Better weight initialization  
methods, Batch Normalization, Dropout

张盛平

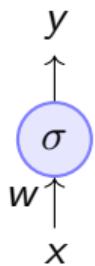
s.zhang@hit.edu.cn

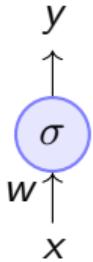
计算学部  
哈尔滨工业大学

2021 年秋季学期

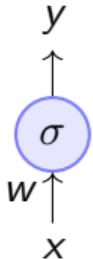


# 快速回忆如何训练深度神经网络



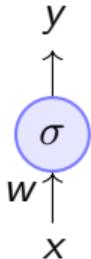


- 只有一个输入的 Sigmoid 神经元网络



- 只有一个输入的 Sigmoid 神经元网络

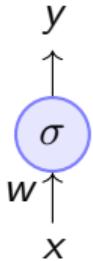
$$w = w - \eta \nabla w \quad \text{where,}$$



- 只有一个输入的 Sigmoid 神经元网络

$$w = w - \eta \nabla w \quad \text{where,}$$

$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$

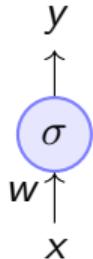


▪ 只有一个输入的 Sigmoid 神经元网络

$$w = w - \eta \nabla w \quad \text{where,}$$

$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$

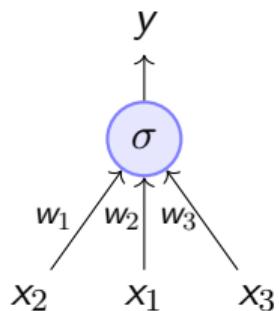


- 只有一个输入的 Sigmoid 神经元网络

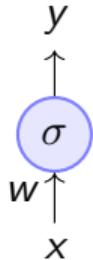
$$w = w - \eta \nabla w \quad \text{where,}$$

$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



- 有多个输入的更宽的 Sigmoid 神经元网络:

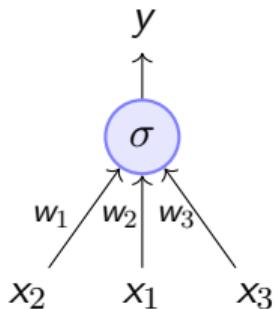


- 只有一个输入的 Sigmoid 神经元网络

$$w = w - \eta \nabla w \quad \text{where,}$$

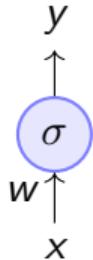
$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



- 有多个输入的更宽的 Sigmoid 神经元网络:

$$w_1 = w_1 - \eta \nabla w_1$$

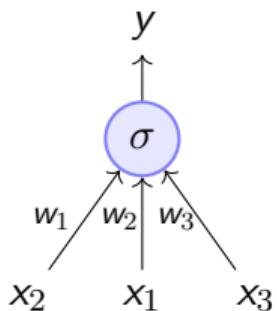


- 只有一个输入的 Sigmoid 神经元网络

$$w = w - \eta \nabla w \quad \text{where,}$$

$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$

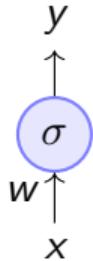
$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



- 有多个输入的更宽的 Sigmoid 神经元网络:

$$w_1 = w_1 - \eta \nabla w_1$$

$$w_2 = w_2 - \eta \nabla w_2$$

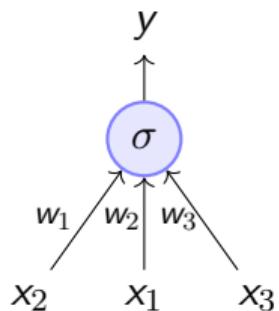


- 只有一个输入的 Sigmoid 神经元网络

$$w = w - \eta \nabla w \quad \text{where,}$$

$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$

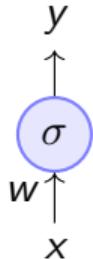


- 有多个输入的更宽的 Sigmoid 神经元网络:

$$w_1 = w_1 - \eta \nabla w_1$$

$$w_2 = w_2 - \eta \nabla w_2$$

$$w_3 = w_3 - \eta \nabla w_3$$

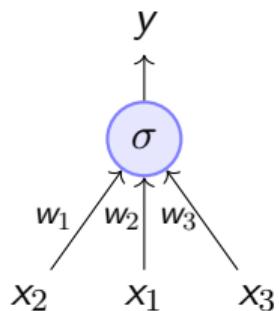


▪ 只有一个输入的 Sigmoid 神经元网络

$$w = w - \eta \nabla w \quad \text{where,}$$

$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$

$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$



▪ 有多个输入的更宽的 Sigmoid 神经元网络:

$$w_1 = w_1 - \eta \nabla w_1$$

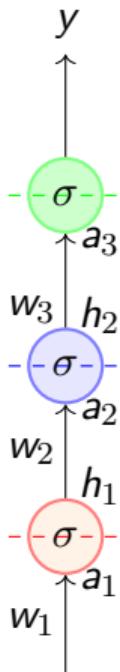
$$w_2 = w_2 - \eta \nabla w_2$$

$$w_3 = w_3 - \eta \nabla w_3$$

$$, \nabla w_i = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x_i$$



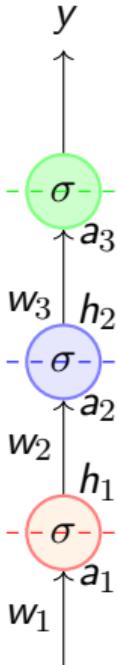
▪ 如何训练一个有更多层的更深的网络?



$$x = h_0$$

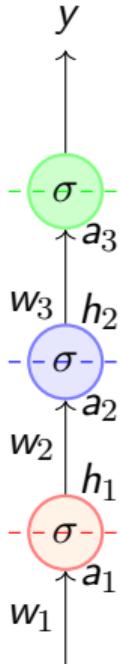
$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$
$$a_1 = w_1 * x = w_1 * h_0$$

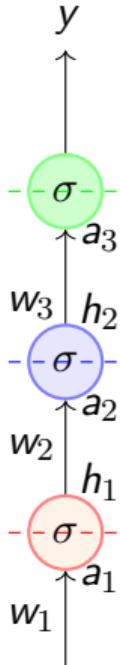
- 如何训练一个有更多层的更深的网络?
- 使用链式法则来计算梯度  $\nabla w_1$  :



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$
$$a_1 = w_1 * x = w_1 * h_0$$

- 如何训练一个有更多层的更深的网络?
- 使用链式法则来计算梯度  $\nabla w_1$  :

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

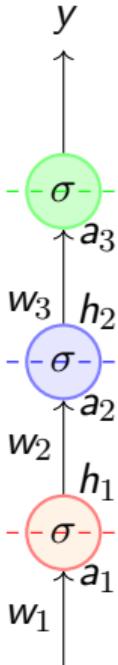


$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

- 如何训练一个有更多层的更深的网络?
- 使用链式法则来计算梯度  $\nabla w_1$  :

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1} &= \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} \\ &= \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} * \dots * h_0\end{aligned}$$



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

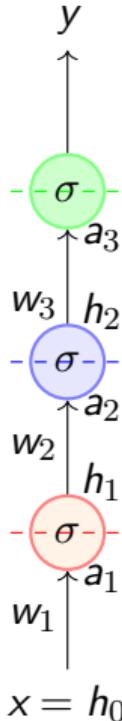
$$a_1 = w_1 * x = w_1 * h_0$$

- 如何训练一个有更多层的更深的网络?
- 使用链式法则来计算梯度  $\nabla w_1$  :

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1} &= \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} \\ &= \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} * \dots * h_0\end{aligned}$$

- 一般而言,

$$\nabla w_i = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} * \dots * h_{i-1}$$



$$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$$

$$a_1 = w_1 * x = w_1 * h_0$$

- 如何训练一个有更多层的更深的网络?
- 使用链式法则来计算梯度  $\nabla w_1$ :

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1} &= \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} \\ &= \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} * \dots * h_0\end{aligned}$$

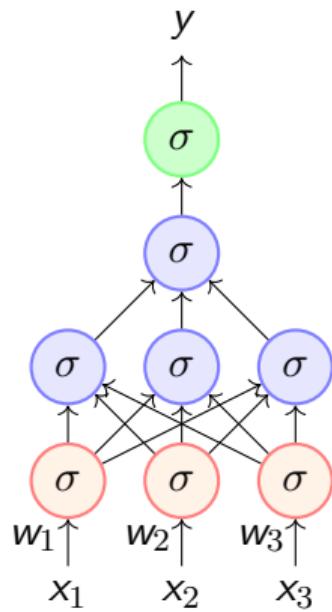
- 一般而言,

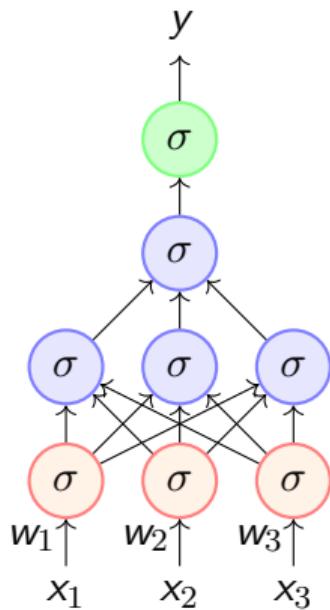
$$\nabla w_i = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} * \dots * h_{i-1}$$

- 注意  $\nabla w_i$  正比于其对应的输入  $h_{i-1}$

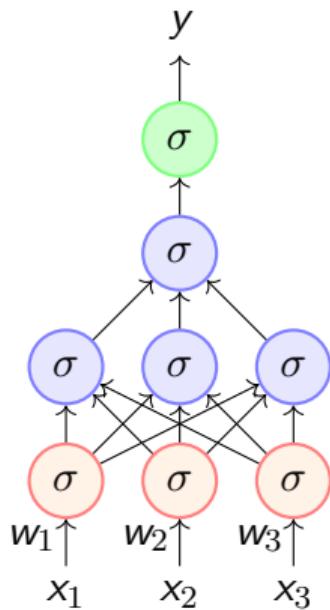


- 如何训练一个又深又宽的神经网络?

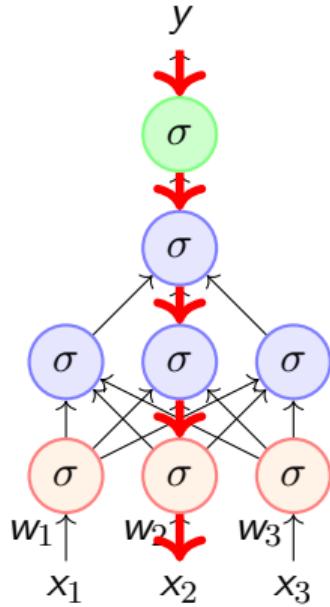




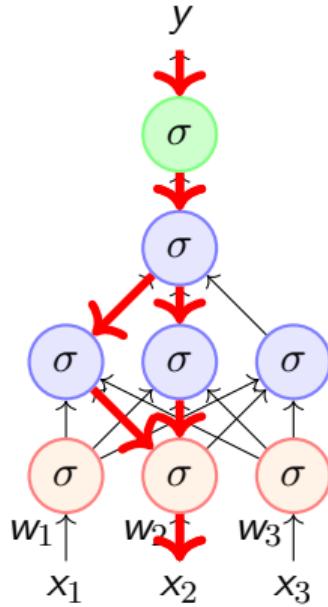
- 如何训练一个又深又宽的神经网络?
- 如何计算  $\nabla w_2 = ?$



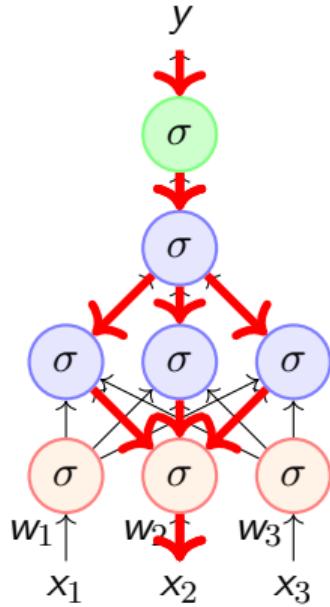
- 如何训练一个又深又宽的神经网络?
- 如何计算  $\nabla w_2 = ?$
- 需要在多个路径上使用链式法则



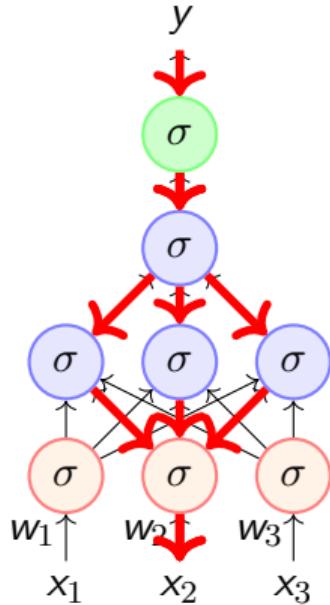
- 如何训练一个又深又宽的神经网络?
- 如何计算  $\nabla w_2 = ?$
- 需要在多个路径上使用链式法则



- 如何训练一个又深又宽的神经网络?
- 如何计算  $\nabla w_2 = ?$
- 需要在多个路径上使用链式法则



- 如何训练一个又深又宽的神经网络?
- 如何计算  $\nabla w_2 = ?$
- 需要在多个路径上使用链式法则



- 如何训练一个又深又宽的神经网络?
- 如何计算  $\nabla w_2 = ?$
- 需要在多个路径上使用链式法则 (在讲解 反向传播时提供过更多细节)



## Things to remember

- 训练神经网络是 *Game of Gradients* (使用现有的基于梯度的方法)



## Things to remember

- 训练神经网络是 *Game of Gradients* (使用现有的基于梯度的方法)
- 梯度描述了每个参数的变化对最小化损失的贡献



## Things to remember

- 训练神经网络是 *Game of Gradients* (使用现有的基于梯度的方法)
- 梯度描述了每个参数的变化对最小化损失的贡献
- 梯度正比于这个参数对应的输入 (在  $\nabla w_i$  的计算公式中的 “ $\dots * x$ ” 或 “ $\dots * h_i$ ” 项)



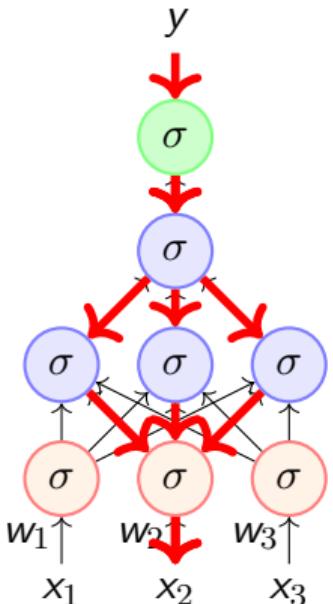
Learning representations  
by back-propagating errors

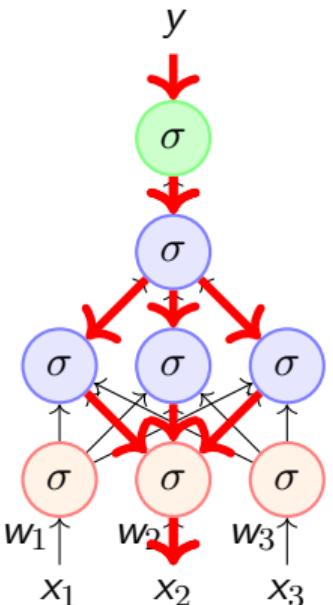
David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure<sup>1</sup>.





Learning representations  
by back-propagating errors

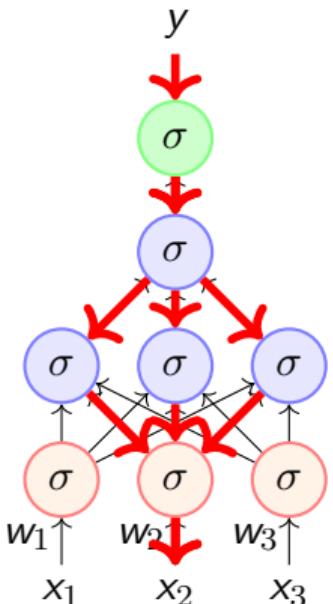
David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure<sup>1</sup>.

- 反向传播算法由 Rumelhart et.al 在 1986 年推广



Learning representations  
by back-propagating errors

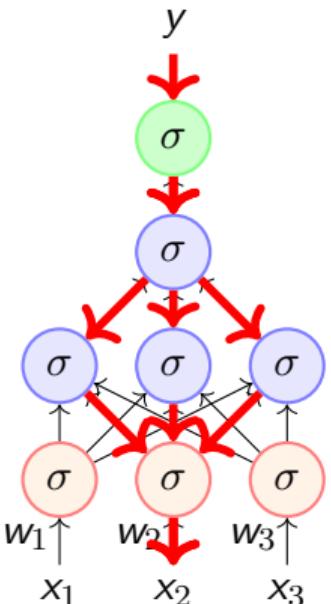
David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure\*.

- 反向传播算法由 Rumelhart et.al 在 1986 年推广
- 对于很深的网络，表现并不好



Learning representations  
by back-propagating errors

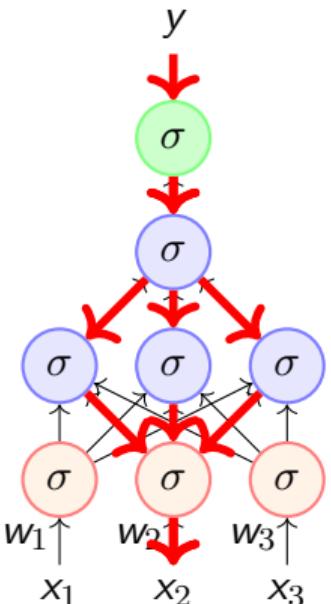
David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure\*.

- 反向传播算法由 Rumelhart et.al 在 1986 年推广
- 对于很深的网络，表现并不好
- 2006 年之前，训练很深的网络仍然很困难



Learning representations  
by back-propagating errors

David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure\*.

- 反向传播算法由 Rumelhart et.al 在 1986 年推广
- 对于很深的网络，表现并不好
- 2006 年之前，训练很深的网络仍然很困难
- 即使是训练很多 epochs 后，可能仍然不收敛



# Unsupervised pre-training



- 2006 年发生了什么?

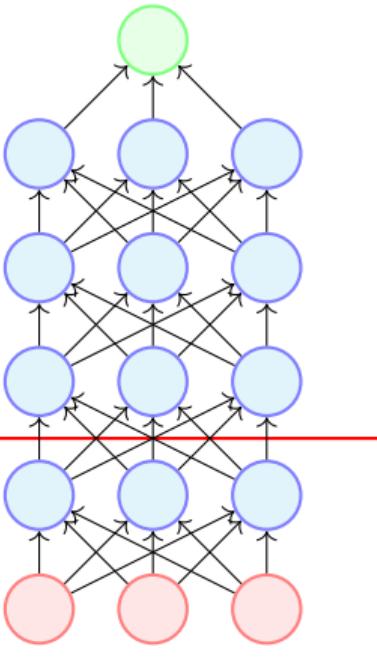
\*G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, July 2006.

- 2006 年发生了什么?
- 得益于 Hinton and Salakhutdinov 在 2006 年的开创性工作— 无监督预训练 (unsupervised pre-training)

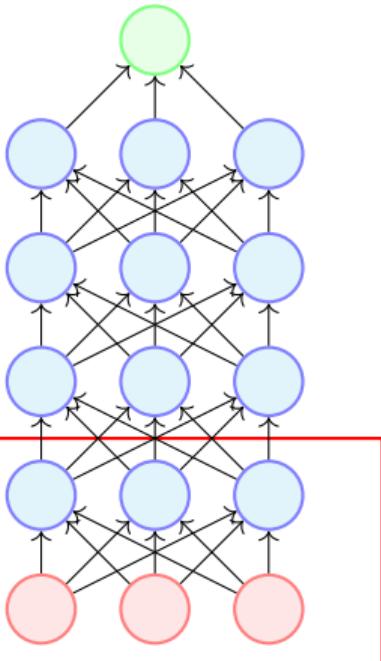
\*G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, July 2006.

- 2006 年发生了什么?
- 得益于 Hinton and Salakhutdinov 在 2006 年的开创性工作— 无监督预训练 (unsupervised pre-training)
- 在这篇论文中, 他们在 RBMs 中引入无监督预训练, 下面我们将在 Autoencoders 中讲解这一方法

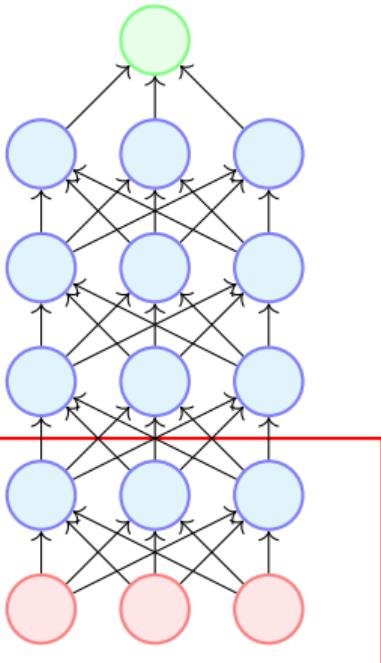
\*G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, July 2006.



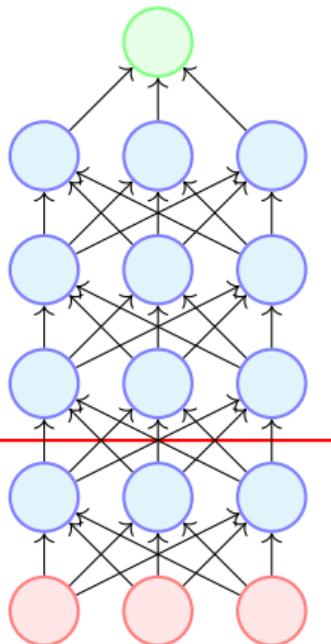
- 图中的深度神经网络



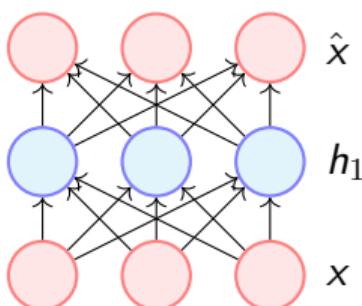
- 图中的深度神经网络
- 考虑网络的前两层 ( $x$  和  $h_1$ )



- 图中的深度神经网络
- 考虑网络的前两层 ( $x$  和  $h_1$ )
- 首先使用一个 **无监督的目标** 来训练这两层之间的权重

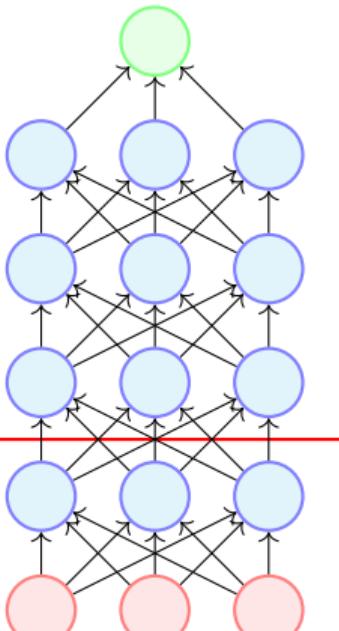


reconstruct  $x$

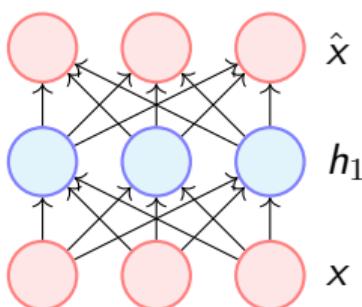


$$\min \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

- 图中的深度神经网络
- 考虑网络的前两层 ( $x$  和  $h_1$ )
- 首先使用一个 **无监督的目标** 来训练这两层之间的权重
- 目标是要从隐含表示 ( $h_1$ ) 中重构出输入 ( $x$ )

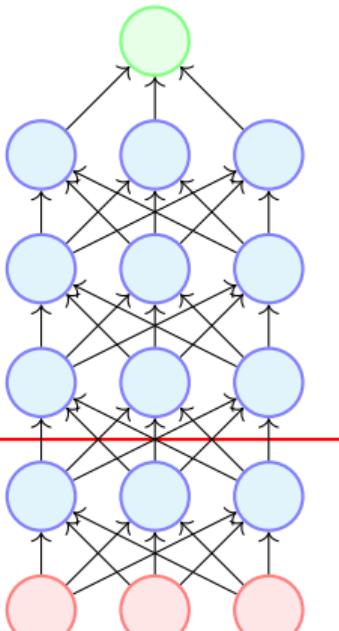


reconstruct  $x$

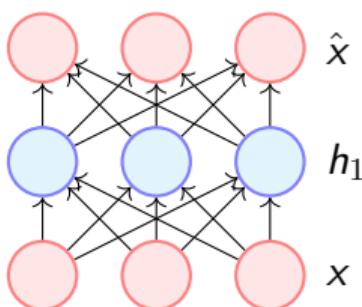


$$\min \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

- 图中的深度神经网络
- 考虑网络的前两层 ( $x$  和  $h_1$ )
- 首先使用一个 **无监督的目标**来训练这两层之间的权重
- 目标是要从隐含表示 ( $h_1$ ) 中重构出输入 ( $x$ )
- 之所以称它是无监督的目标，是因为不涉及到输入的标签 ( $y$ )，而仅仅使用输入数据 ( $x$ )

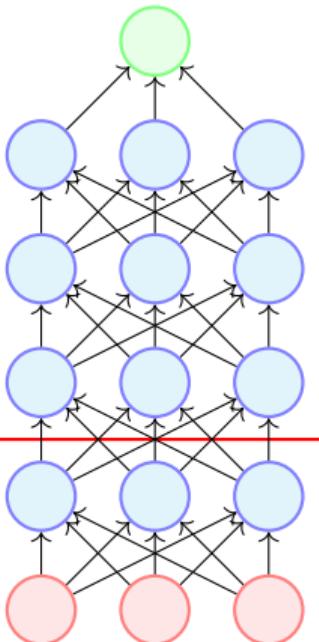


reconstruct  $x$

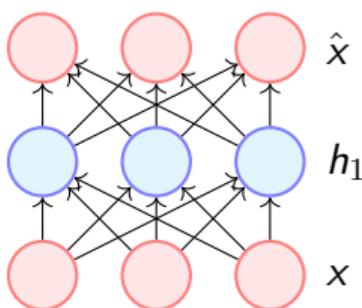


$$\min \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

- 图中的深度神经网络
- 考虑网络的前两层 ( $x$  和  $h_1$ )
- 首先使用一个 **无监督的目标**来训练这两层之间的权重
- 目标是要从隐含表示 ( $h_1$ ) 中重构出输入 ( $x$ )
- 之所以称它是无监督的目标，是因为不涉及到输入的标签 ( $y$ )，而仅仅使用输入数据 ( $x$ )

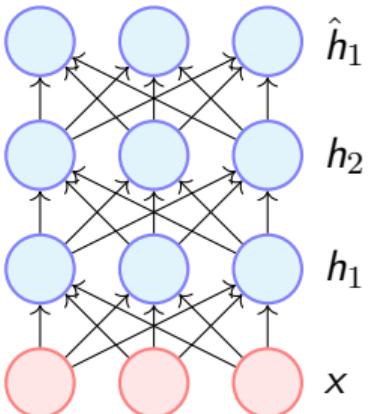
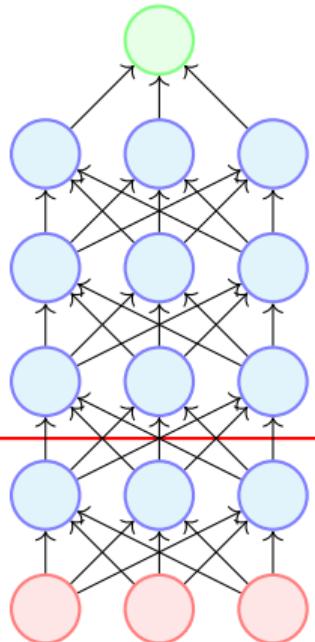


reconstruct  $x$



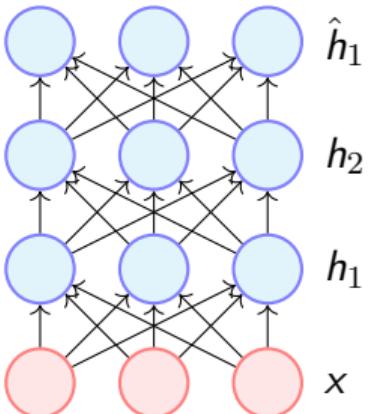
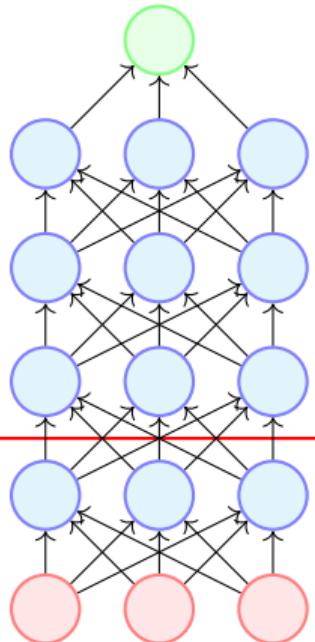
$$\min \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

- 经过这一步后，第一层的权重被训练，使得  $h_1$  捕获输入  $x$  的重要信息



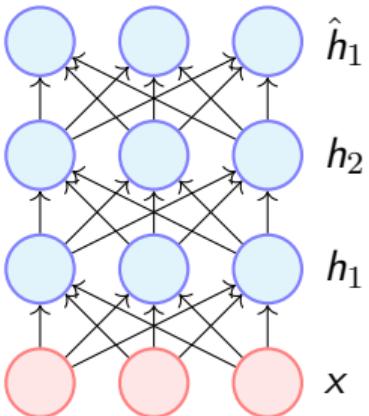
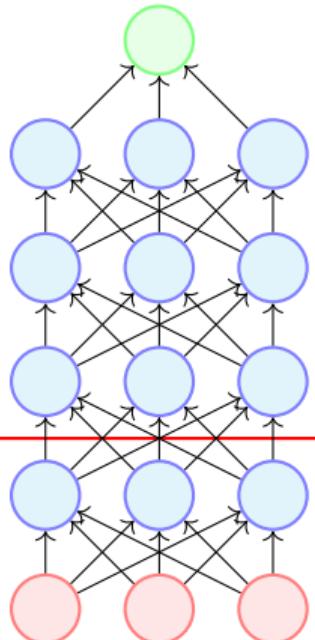
$$\min \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{h}_{1ij} - h_{1ij})^2$$

- 经过这一步后，第一层的权重被训练，使得  $h_1$  捕获输入  $x$  的重要信息
- 然后，将第一层的权重固定，在第二层上重复这一过程



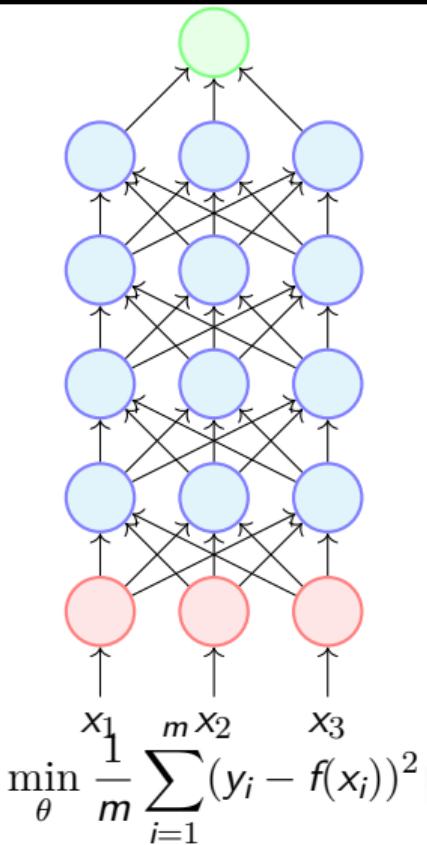
$$\min \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{h}_{1ij} - h_{1ij})^2$$

- 经过这一步后，第一层的权重被训练，使得  $h_1$  捕获输入  $x$  的重要信息
- 然后，将第一层的权重固定，在第二层上重复这一过程
- 经过这一步后，第二层的权重被训练，使得  $h_2$  捕获  $h_1$  的重要信息

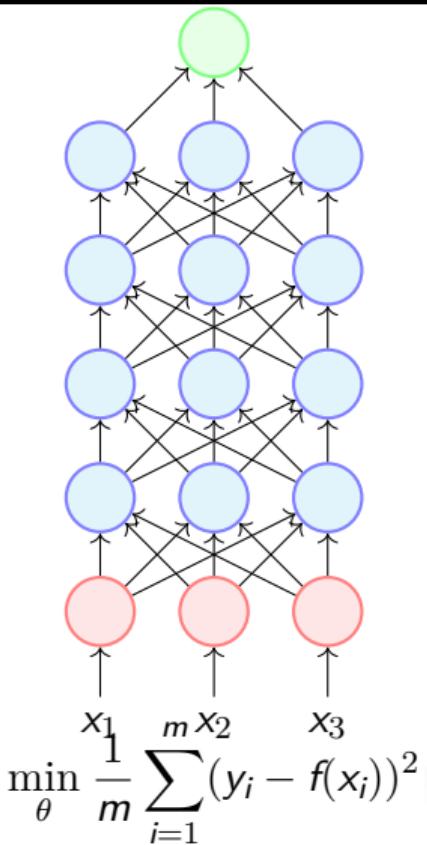


$$\min \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{h}_{1ij} - h_{1ij})^2$$

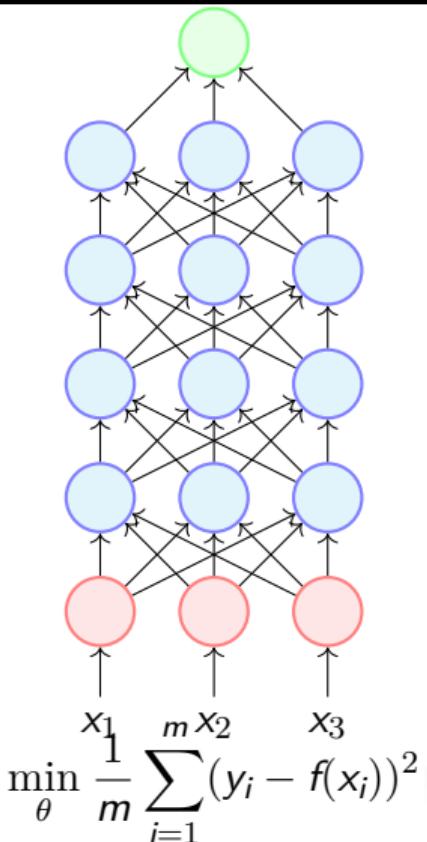
- 经过这一步后，第一层的权重被训练，使得  $h_1$  捕获输入  $x$  的重要信息
- 然后，将第一层的权重固定，在第二层上重复这一过程
- 经过这一步后，第二层的权重被训练，使得  $h_2$  捕获  $h_1$  的重要信息
- 继续这一过程，直到最后一个隐含层 (*i.e.*, 输出层的前一层)



- 预训练结束后，使用训练出的权重来初始化隐含层的权重。所得到的网络能够学习到输入数据类别独立的特征表示 (class independent 因为没有使用到数据的标签  $y$ )



- 预训练结束后，使用训练出的权重来初始化隐含层的权重。所得到的网络能够学习到输入数据类别独立的特征表示 (class independent 因为没有使用到数据的标签  $y$ )
- 预训练结束后，再在网络上增加输出层，使用特定的目标 (或损失函数) 来训练整个网络



- 预训练结束后，使用训练出的权重来初始化隐含层的权重。所得到的网络能够学习到输入数据类别独立的特征表示 (class independent 因为没有使用到数据的标签  $y$ )
- 预训练结束后，再在网络上增加输出层，使用特定的目标 (或损失函数) 来训练整个网络
- 整个过程可以理解为：先使用无监督的预训练（无监督的目标）来初始化网络权重，再使用特定有监督的目标来 fine tune 整个网络



为什么这种无监督的预训练-有监督的 fine-tune 能工作的更好？

---

\*The difficulty of training deep architectures and effect of unsupervised pre-training - Erhan et al,2009

\*Exploring Strategies for Training Deep Neural Networks, Larocelle et al,2009

## 为什么这种无监督的预训练-有监督的 fine-tune 能工作的更好？

- better optimization?

---

\*The difficulty of training deep architectures and effect of unsupervised pre-training - Erhan et al,2009

\*Exploring Strategies for Training Deep Neural Networks, Larochelle et al,2009

## 为什么这种无监督的预训练-有监督的 fine-tune 能工作的更好？

- better optimization?
- better regularization?

---

\*The difficulty of training deep architectures and effect of unsupervised pre-training - Erhan et al,2009

\*Exploring Strategies for Training Deep Neural Networks, Larochelle et al,2009



为什么这种无监督的预训练-有监督的 fine-tune 能工作的更好？

- better optimization?
- better regularization?

先搞清楚这两个问题意味着什么，再基于现有的研究成果<sup>1,2</sup> 来回答这一问题

---

\*The difficulty of training deep architectures and effect of unsupervised pre-training - Erhan et al,2009

\*Exploring Strategies for Training Deep Neural Networks, Larochelle et al,2009



Why does this work better?

- Is it because of better optimization?
- Is it because of better regularization?



- 我们正在解决什么样的优化问题？



- 我们正在解决什么样的优化问题？

$$\text{minimize } \mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^m (y_i - f(x_i))^2$$



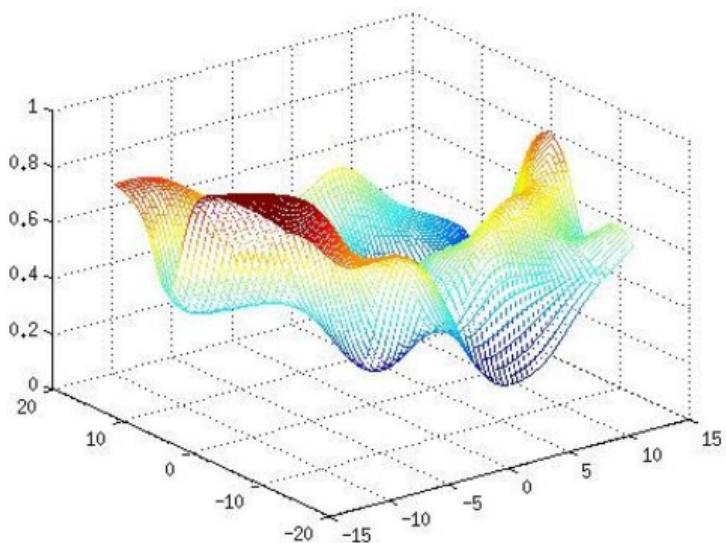
- 我们正在解决什么样的优化问题？

$$\text{minimize } \mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^m (y_i - f(x_i))^2$$

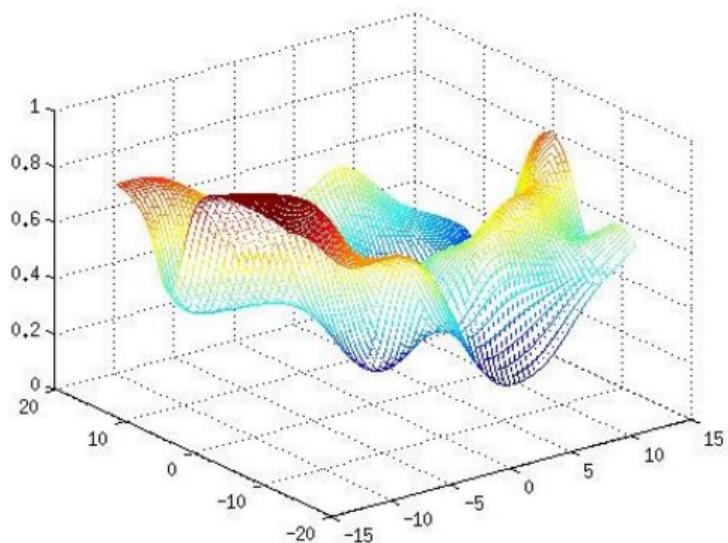
- 如果没有无监督的预训练，现有的优化方法不能够使得训练集上的损失函数  $\mathcal{L}(\theta)$  降为 0 吗？(因此 poor optimization) ?



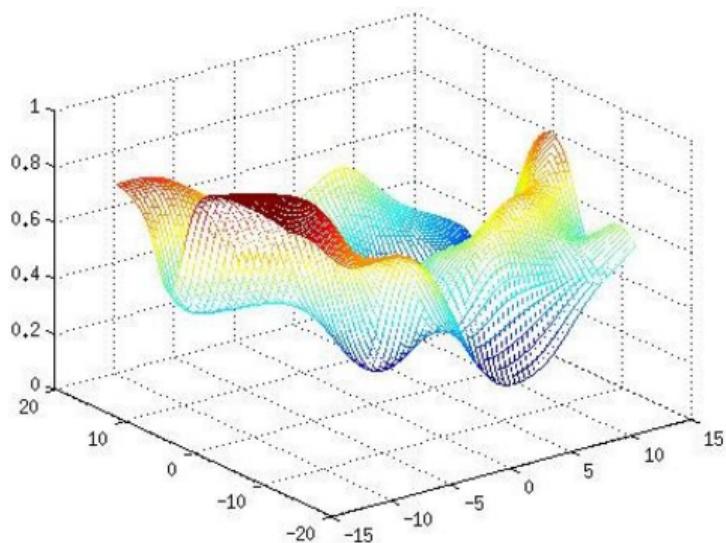
- 一个深度神经网络的有监督目标的误差曲面是高度非凸的



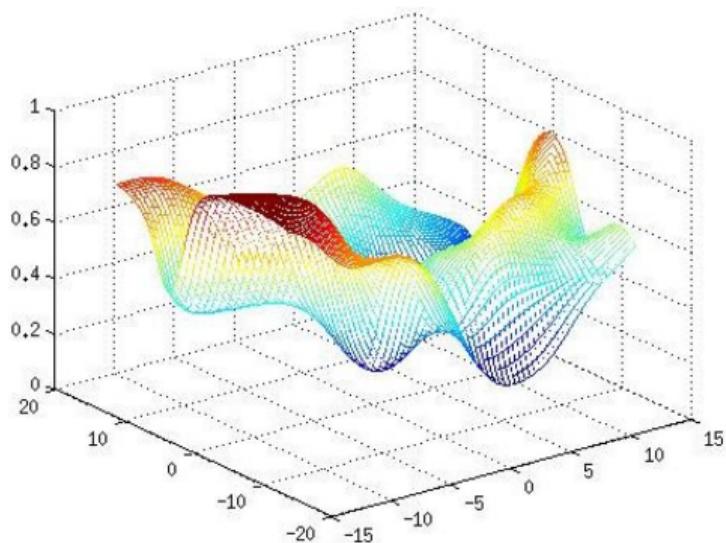
\*Exploring Strategies for Training Deep Neural Networks, Larochelle et al, 2009



- 一个深度神经网络的有监督目标的误差曲面是高度非凸的
- 有很多丘陵、高原和山谷

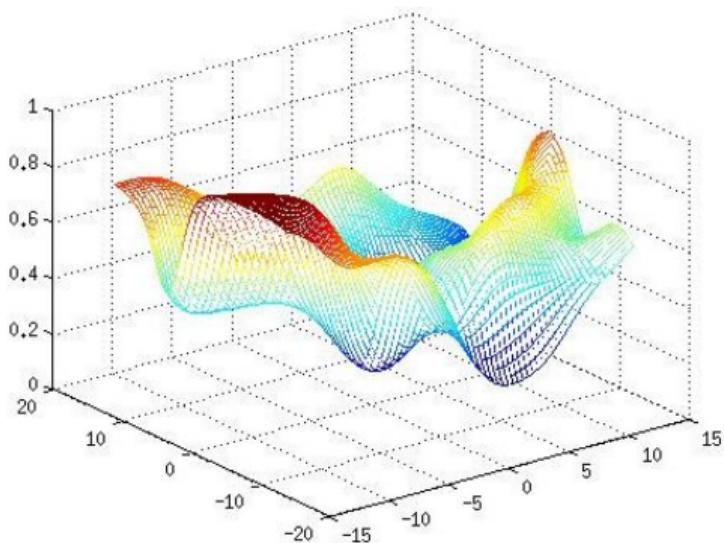


- 一个深度神经网络的有监督目标的误差曲面是高度非凸的
- 有很多丘陵、高原和山谷
- 如果 DNN 有 large capacity, 很容易落到这些损失函数为 0 的区域



- 一个深度神经网络的有监督目标的误差曲面是高度非凸的
- 有很多丘陵、高原和山谷
- 如果 DNN 有 large capacity, 很容易落到这些损失函数为 0 的区域
- 实际上, Larochelle et.al.<sup>1</sup> 表明, 如果网络的最后一层有 large capacity, 即使没有预训练, 损失函数  $\mathcal{L}(\theta)$  也会降到 0

\*Exploring Strategies for Training Deep Neural Networks, Larocelle et al,2009



- 一个深度神经网络的有监督目标的误差曲面是高度非凸的
- 有很多丘陵、高原和山谷
- 如果 DNN 有 large capacity, 很容易落到这些损失函数为 0 的区域
- 实际上, Larochelle et.al.<sup>1</sup> 表明, 如果网络的最后一层有 large capacity, 即使没有预训练, 损失函数  $\mathcal{L}(\theta)$  也会降到 0
- 然而, 如果网络的容量很小, 无监督的预训练会起到帮助

\*Exploring Strategies for Training Deep Neural Networks, Larocelle et al,2009



Why does this work better?

- Is it because of better optimization?
- Is it because of better regularization?



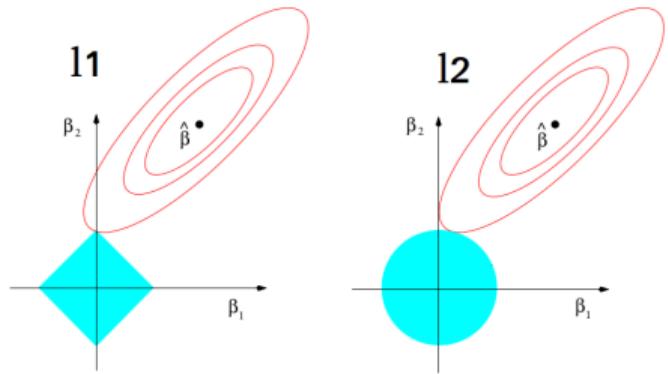
## ▪ 正则化起到什么作用？

\*Image Source: The Elements of Statistical Learning-T. Hastie, R. Tibshirani, and J. Friedman, Pg



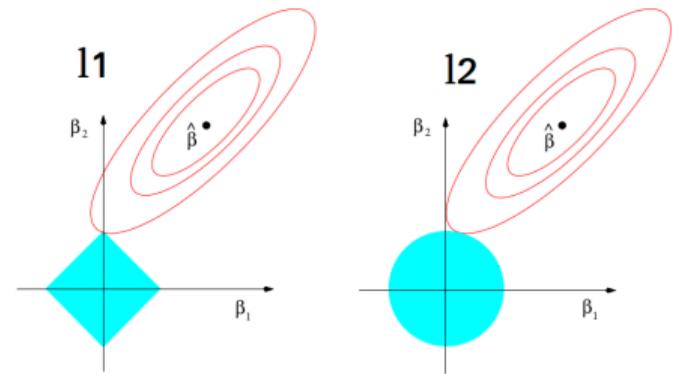
- 正则化起到什么作用？它将权重约束到参数空间中的特定区域

\*Image Source: The Elements of Statistical Learning-T. Hastie, R. Tibshirani, and J. Friedman, Pg



- 正则化起到什么作用？它将权重约束到参数空间中的特定区域
- L-1 regularization: 约束大多数权重为 0

\*Image Source: The Elements of Statistical Learning-T. Hastie, R. Tibshirani, and J. Friedman, Pg



- 正则化起到什么作用？它将权重约束到参数空间中的特定区域
- L-1 regularization: 约束大多数权重为0
- L-2 regularization: 阻止大多数权重取值较大

\*Image Source: The Elements of Statistical Learning-T. Hastie, R. Tibshirani, and J. Friedman, Pg



- 实际上，预训练将权重约束到参数空间的特定区域



- 实际上，预训练将权重约束到参数空间的特定区域
- 特别地，它将权重约束到能很好捕获数据特性的区域（无监督的预训练优化的目标是为了重构输入）



- **Unsupervised objective:**

$$\Omega(\theta) = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - \hat{x}_{ij})^2$$

- 实际上，预训练将权重约束到参数空间的特定区域
- 特别地，它将权重约束到能很好捕获数据特性的区域（无监督的预训练优化的目标是为了重构输入）



- **Unsupervised objective:**

$$\Omega(\theta) = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - \hat{x}_{ij})^2$$

- 实际上，预训练将权重约束到参数空间的特定区域
- 特别地，它将权重约束到能很好捕获数据特性的区域（无监督的预训练优化的目标是为了重构输入）

- 无监督的目标可以作为优化问题的额外约束



- **Unsupervised objective:**

$$\Omega(\theta) = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - \hat{x}_{ij})^2$$

- 无监督的目标可以作为优化问题的额外约束
- **Supervised objective:**

$$\mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^m (y_i - f(x_i))^2$$

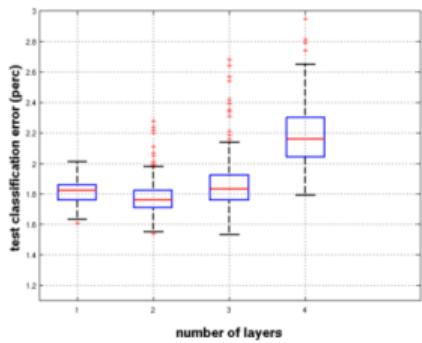
- 实际上，预训练将权重约束到参数空间的特定区域
- 特别地，它将权重约束到能很好捕获数据特性的区域（无监督的预训练优化的目标是为了重构输入）
- 这种无监督的优化目标确保无监督的学习不像有监督学习那样贪婪



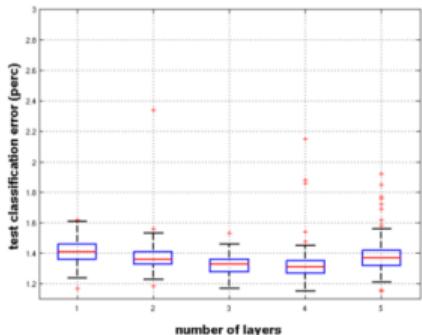
- 实验结果也表明预训练比随机初始化权重更鲁棒

---

\* The difficulty of training deep architectures and effect of unsupervised pre-training - Erhan et al 2009



- 实验结果也表明预训练比随机初始化权重更鲁棒
- 因为使用预训练时，隐含层能更好的捕获数据的内在特性



\* The difficulty of training deep architectures and effect of unsupervised pre-training - Erhan et al 2009



2006-2009 年发生了什么?



## Better activation functions

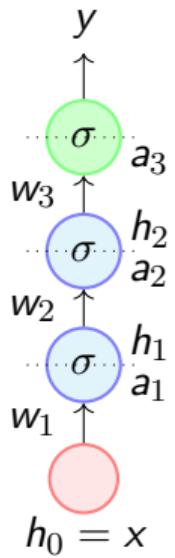
## Deep Learning has evolved

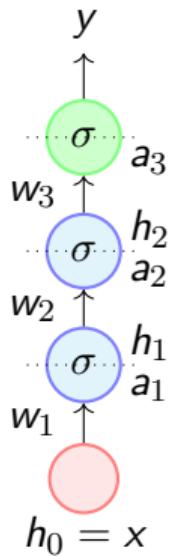
- Better optimization algorithms
- Better regularization methods
- **Better activation functions**
- Better weight initialization strategies



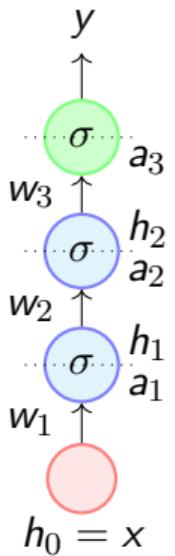
- 在学习激活函数之前，先回答：是什么让 Deep Neural Networks 如此给力？

▪ 考虑这个 deep neural network



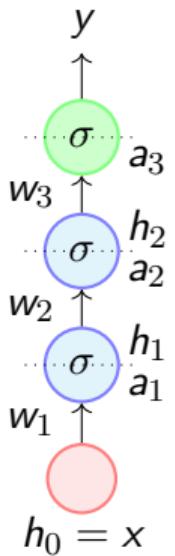


- 考虑这个 deep neural network
- 想象一下, 如果将每一层的 sigmoid 神经元替换成一个简单的线性变换



- 考虑这个 deep neural network
- 想象一下, 如果将每一层的 sigmoid 神经元替换成一个简单的线性变换

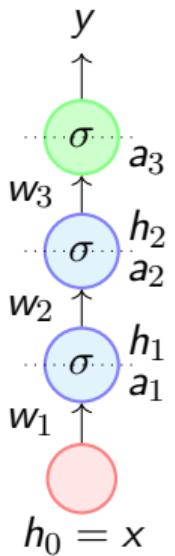
$$y = (w_4 * (w_3 * (w_2 * (w_1 * (w_1 x)))))$$



- 考虑这个 deep neural network
- 想象一下, 如果将每一层的 sigmoid 神经元替换成一个简单的线性变换

$$y = (w_4 * (w_3 * (w_2 * (w_1 * x))))$$

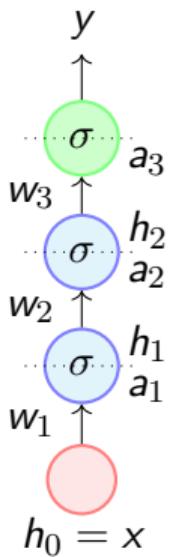
- 最终只能学习到  $y$  是  $x$  的线性变换



- 考虑这个 deep neural network
- 想象一下, 如果将每一层的 sigmoid 神经元替换成一个简单的线性变换

$$y = (w_4 * (w_3 * (w_2 * (w_1 * x))))$$

- 最终只能学习到  $y$  是  $x$  的线性变换
- 只能学习到线性的决策函数



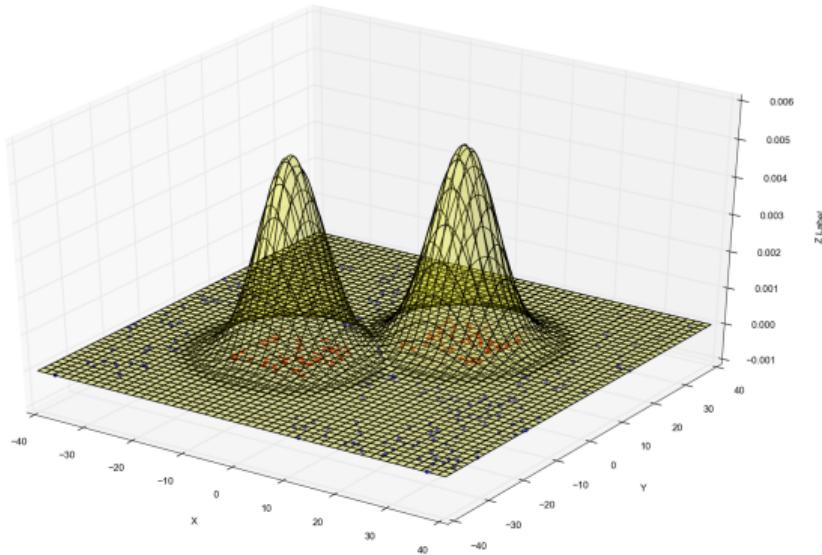
- 考虑这个 deep neural network
- 想象一下, 如果将每一层的 sigmoid 神经元替换成一个简单的线性变换

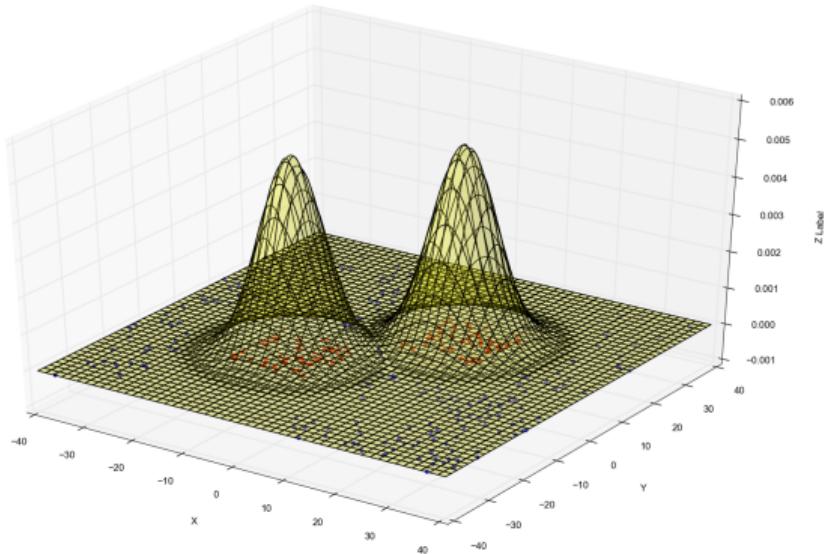
$$y = (w_4 * (w_3 * (w_2 * (w_1 x))))$$

- 最终只能学习到  $y$  是  $x$  的线性变换
- 只能学习到线性的决策函数
- 不能学习到任意的决策函数



- 一个深度线性神经网络不能学习到这样的决策函数





- 一个深度线性神经网络不能学习到这样的决策函数
- 实际上，一个深度的非线性神经网络能够学习到这样的决策函数 (recall Universal Approximation Theorem)



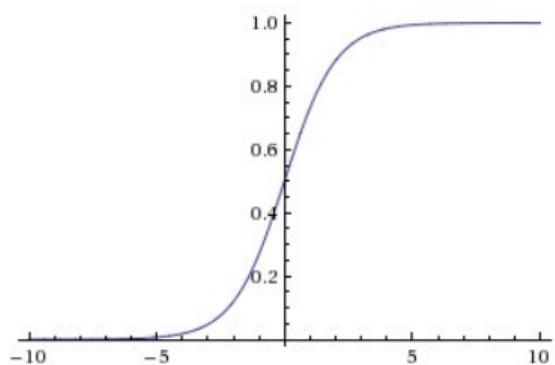
- 深度神经网络中经常使用的非线性激活函数有哪些 ? (\*)

---

\*<http://cs231n.github.io>

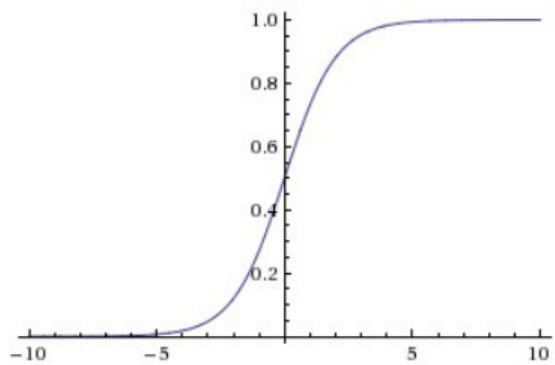


- $\sigma(x) = \frac{1}{1+e^{-x}}$



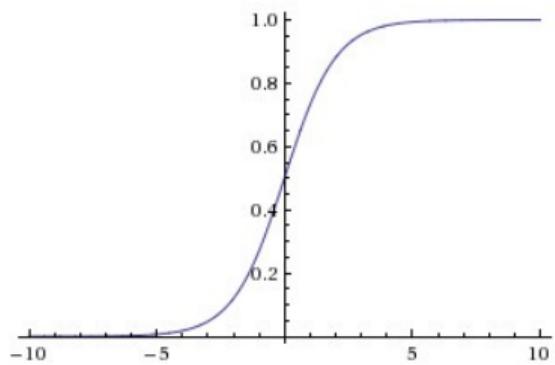
Sigmoid

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- sigmoid 函数将输入约束到 [0,1]



Sigmoid

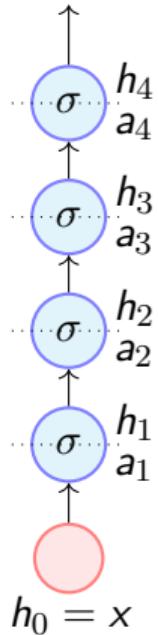
- $\sigma(x) = \frac{1}{1+e^{-x}}$
- sigmoid 函数将输入约束到 [0,1]
- 它的梯度计算如下：



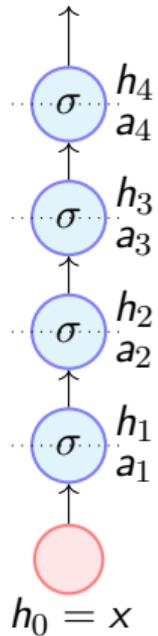
Sigmoid

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- sigmoid 函数将输入约束到 [0,1]
- 它的梯度计算如下：

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$



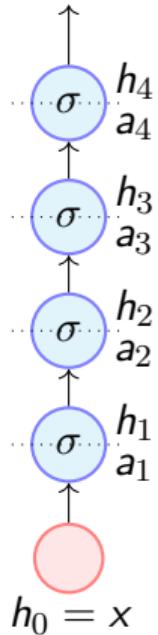
$$a_3 = w_2 h_2 \\ h_3 = \sigma(a_3)$$



$$a_3 = w_2 h_2 \\ h_3 = \sigma(a_3)$$

▪ 计算  $\nabla w_2$  时涉及,

$$\frac{\partial h_3}{\partial a_3} = \frac{\partial \sigma(a_3)}{\partial a_3} = \sigma(a_3)(1 - \sigma(a_3))$$

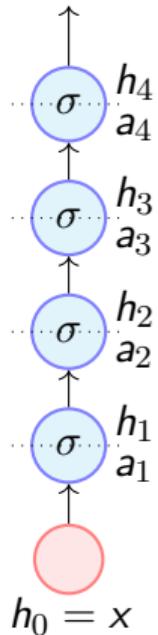


$$a_3 = w_2 h_2$$
$$h_3 = \sigma(a_3)$$

- 计算  $\nabla w_2$  时涉及,

$$\frac{\partial h_3}{\partial a_3} = \frac{\partial \sigma(a_3)}{\partial a_3} = \sigma(a_3)(1 - \sigma(a_3))$$

- 这会导致什么结果?

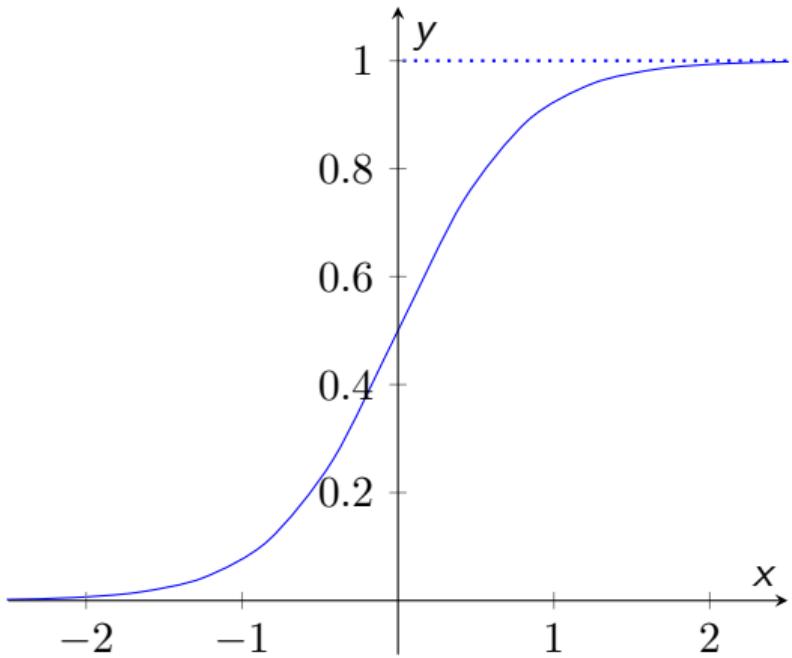


$$a_3 = w_2 h_2 \\ h_3 = \sigma(a_3)$$

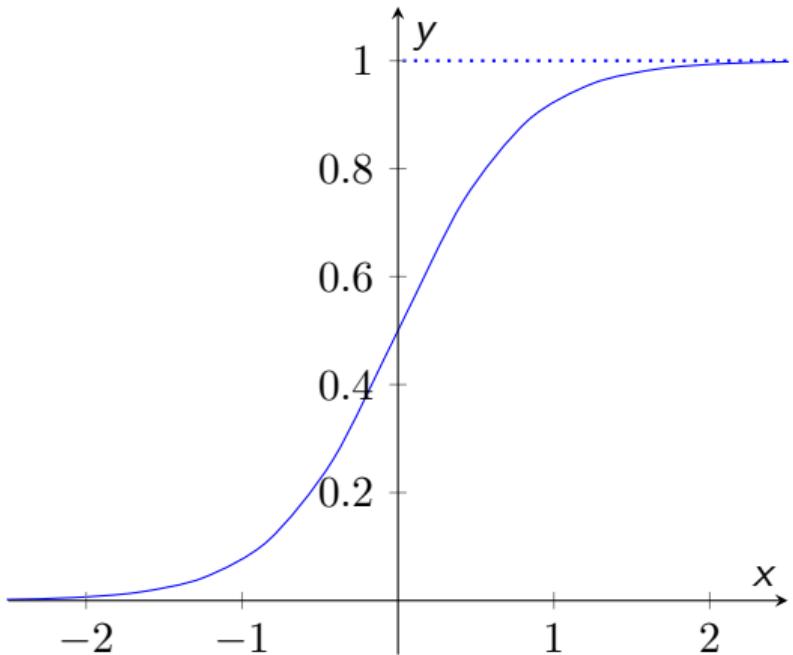
- 计算  $\nabla w_2$  时涉及,

$$\frac{\partial h_3}{\partial a_3} = \frac{\partial \sigma(a_3)}{\partial a_3} = \sigma(a_3)(1 - \sigma(a_3))$$

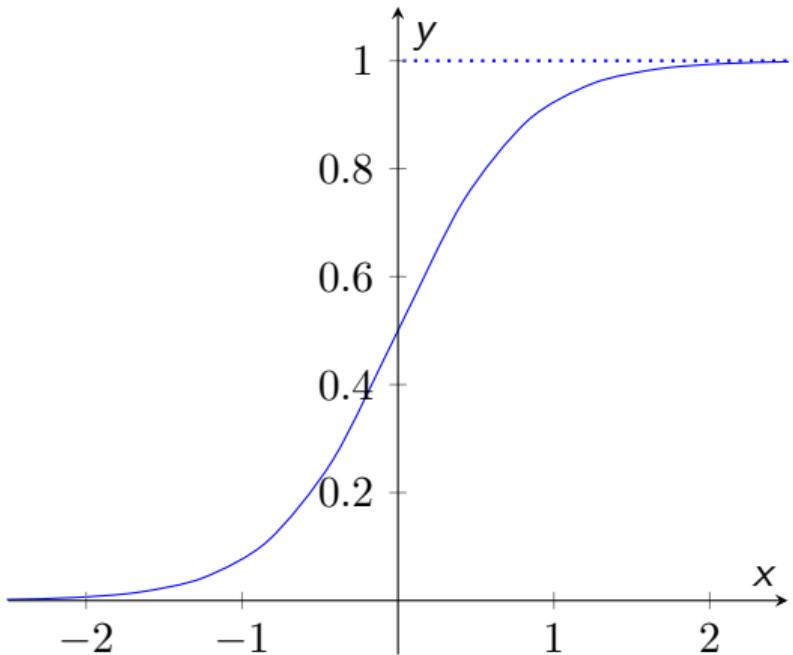
- 这会导致什么结果?
- 回答这个问题, 先理解什么是 saturated neuron ?



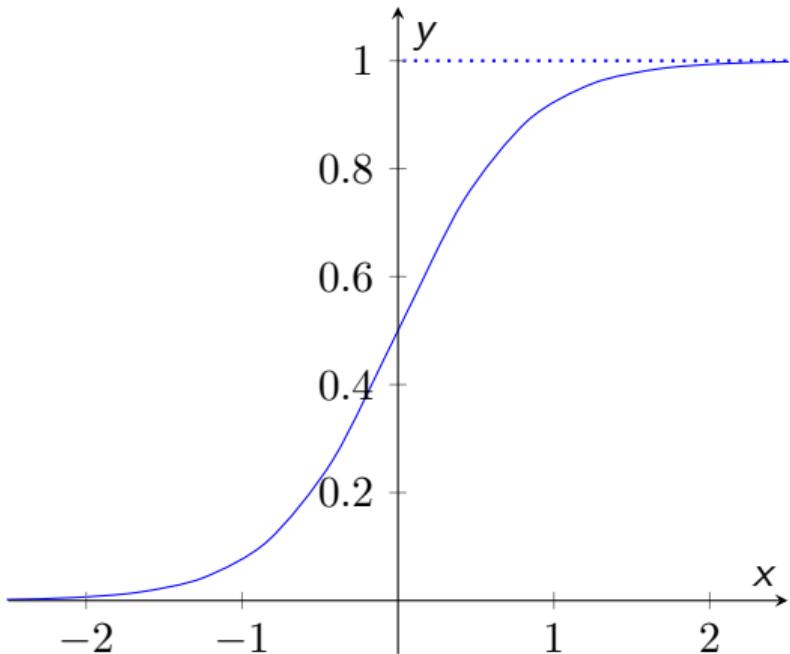
- 一个 sigmoid neuron 被称为饱和了, 当  $\sigma(x) = 1$  或  $\sigma(x) = 0$



- 一个 sigmoid neuron 被称为饱和了, 当  $\sigma(x) = 1$  或  $\sigma(x) = 0$
- 当在饱和处, sigmoid 函数的梯度为 0



- 一个 sigmoid neuron 被称为饱和了, 当  $\sigma(x) = 1$  或  $\sigma(x) = 0$
- 当在饱和处, sigmoid 函数的梯度为 0

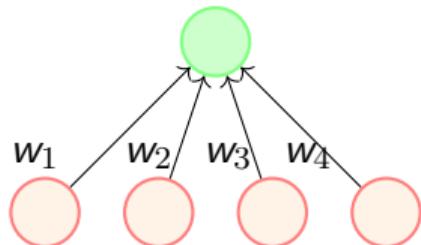


- 一个 sigmoid neuron 被称为饱和了, 当  $\sigma(x) = 1$  或  $\sigma(x) = 0$
- 当在饱和处, sigmoid 函数的梯度为 0

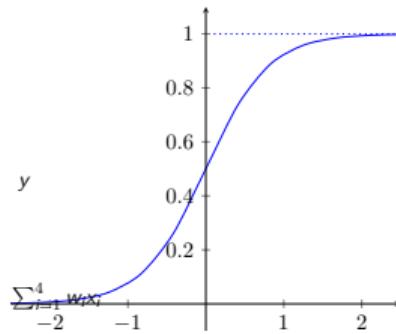
饱和的神经元导致梯度消失



## 饱和的神经元导致梯度消失



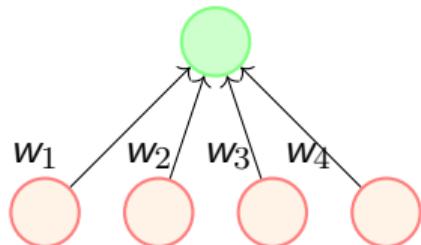
$$\sigma(\sum_{i=1}^4 w_i x_i)$$



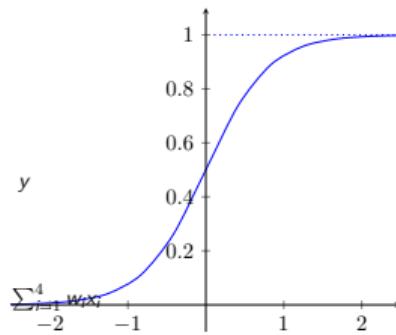
- 如果给 sigmoid 神经元的权重赋很大的初值



## 饱和的神经元导致梯度消失



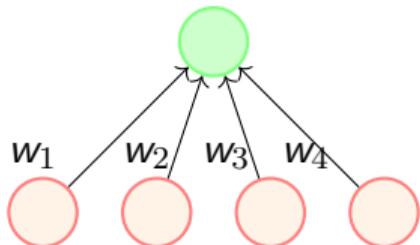
$$\sigma(\sum_{i=1}^4 w_i x_i)$$



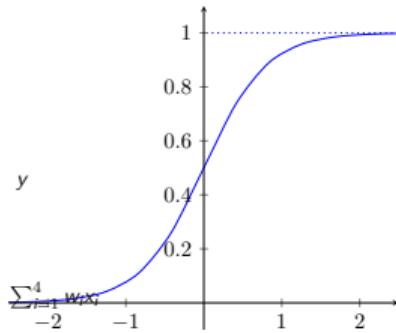
- 如果给 sigmoid 神经元的权重赋很大的初值
- 神经元将很快达到饱和



## 饱和的神经元导致梯度消失



$$\sigma(\sum_{i=1}^4 w_i x_i)$$



- 如果给 sigmoid 神经元的权重赋很大的初值
- 神经元将很快达到饱和
- 梯度将消失，训练将会停止



- 饱和的神经元将导致梯度消失

- 饱和的神经元将导致梯度消失
- Sigmoids 不是 0 中心化的 (Zero centered)

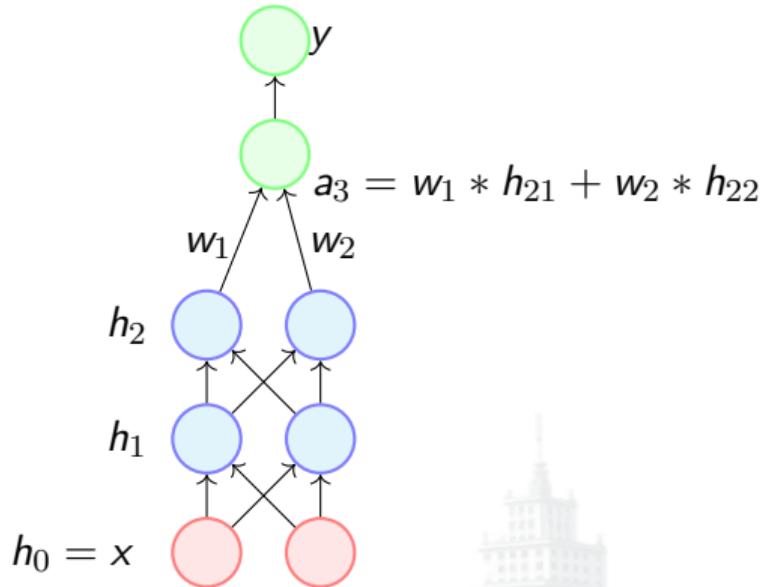


- 饱和的神经元将导致梯度消失
- Why 非 0 中心化会导致什么问题 ?
- Sigmoids 不是 0 中心化的 (Zero centered)



- 饱和的神经元将导致梯度消失
- Sigmoids 不是 0 中心化的 (Zero centered)

- Why 非 0 中心化会导致什么问题 ?



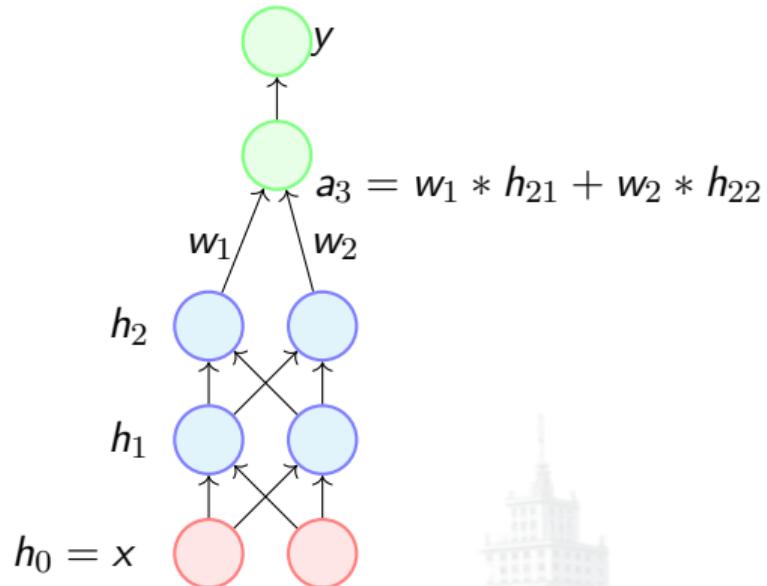


- 饱和的神经元将导致梯度消失
- Sigmoids 不是 0 中心化的 (Zero centered)
- 考虑损失函数分别对  $w_1$  和  $w_2$  的梯度

$$\nabla w_1 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} \frac{\partial a_3}{\partial w_1}$$

$$\nabla w_2 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} \frac{\partial a_3}{\partial w_2}$$

- Why 非 0 中心化会导致什么问题 ?





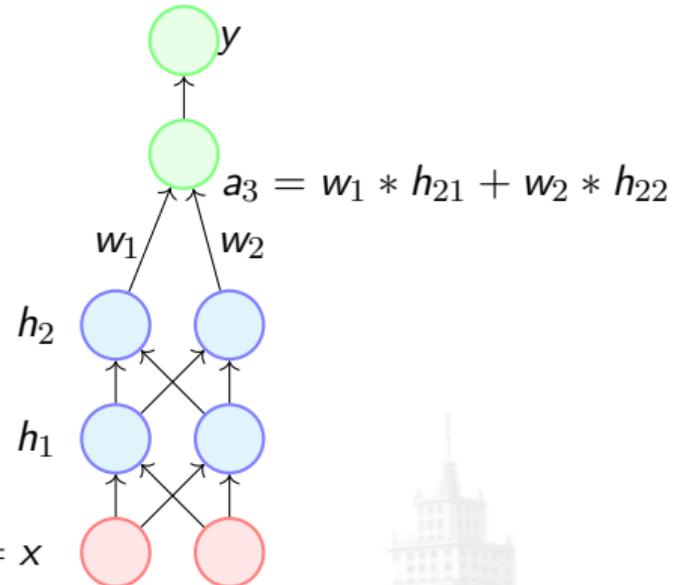
- 饱和的神经元将导致梯度消失
- Sigmoids 不是 0 中心化的 (Zero centered)
- 考虑损失函数分别对  $w_1$  和  $w_2$  的梯度

$$\nabla w_1 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} h_{21}$$

$$\nabla w_2 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} h_{22}$$

- 注意  $h_{21}$  和  $h_{22}$  取值在  $[0, 1]$  (*i.e.*, 它们都是正数)

- Why 非 0 中心化会导致什么问题 ?





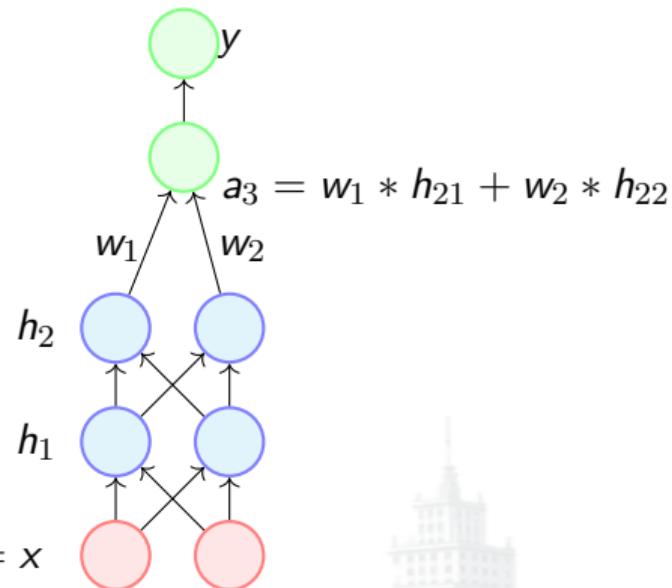
- 饱和的神经元将导致梯度消失
- Sigmoids 不是 0 中心化的 (Zero centered)
- 考虑损失函数分别对  $w_1$  和  $w_2$  的梯度

$$\nabla w_1 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} h_{21}$$

$$\nabla w_2 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} h_{22}$$

- 注意  $h_{21}$  和  $h_{22}$  取值在  $[0, 1]$  (*i.e.*, 它们都是正数)
- 因此, 第一个共同的项 (红色) 是正 (负), 则  $\nabla w_1$  和  $\nabla w_2$  将是正 (负)

- Why 非 0 中心化会导致什么问题?





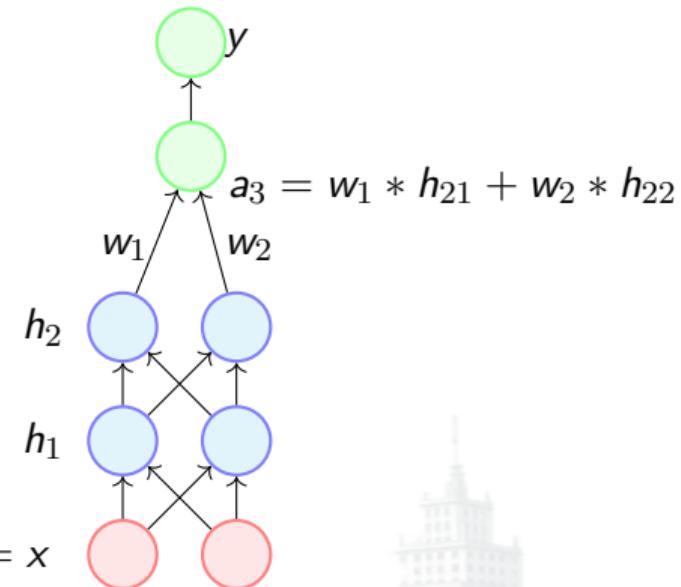
- 饱和的神经元将导致梯度消失
- Sigmoids 不是 0 中心化的 (Zero centered)
- 考虑损失函数分别对  $w_1$  和  $w_2$  的梯度

$$\nabla w_1 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} h_{21}$$

$$\nabla w_2 = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_3} h_{22}$$

- 注意  $h_{21}$  和  $h_{22}$  取值在  $[0, 1]$  (*i.e.*, 它们都是正数)
- 因此, 第一个共同的项 (红色) 是正 (负), 则  $\nabla w_1$  和  $\nabla w_2$  将是正 (负)

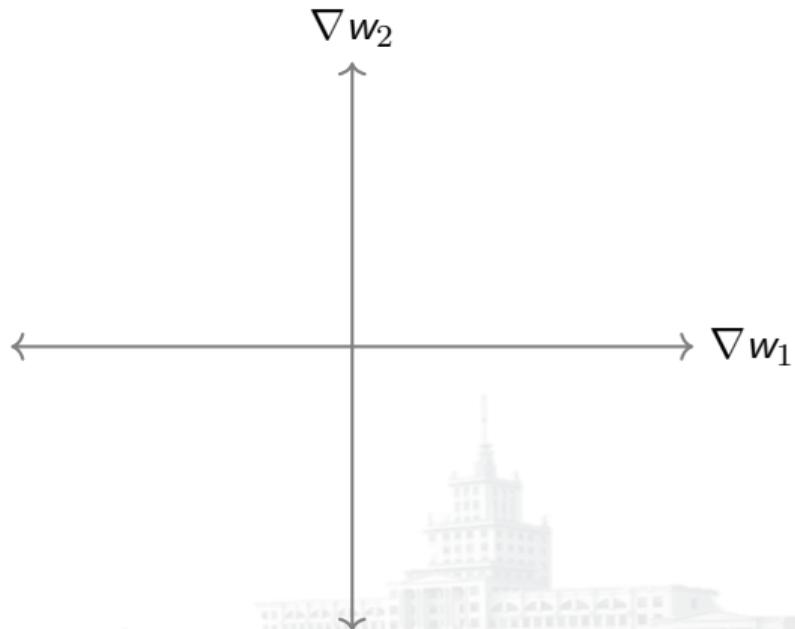
- Why 非 0 中心化会导致什么问题?



- 本质上, 某一层的梯度要么全是正, 要么全是负

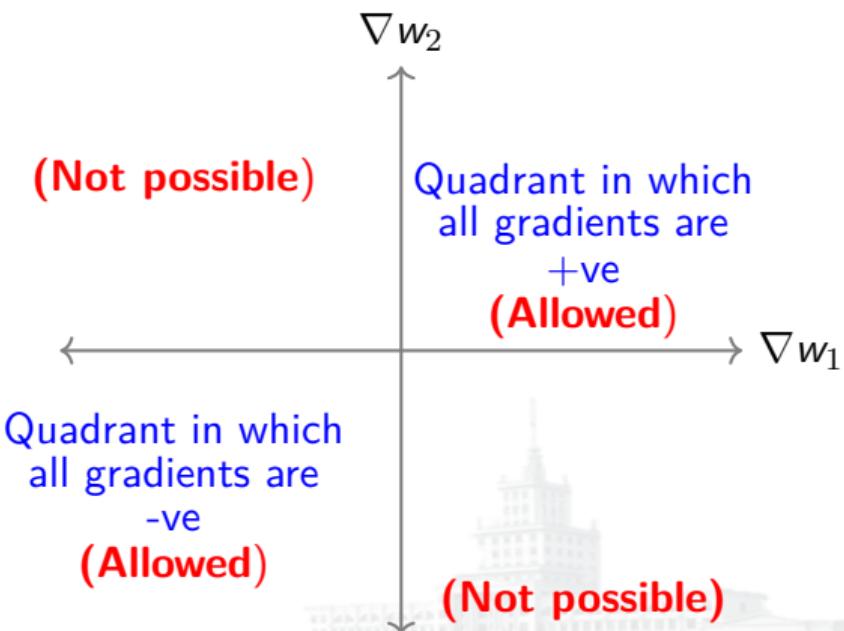


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered
- 这限制了参数可能的更新方向



- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered

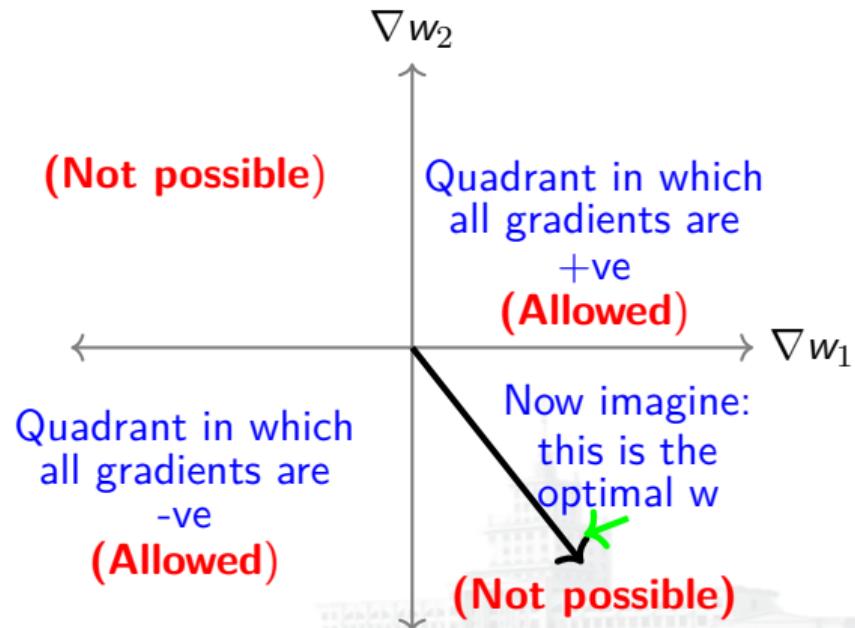
- 这限制了参数可能的更新方向





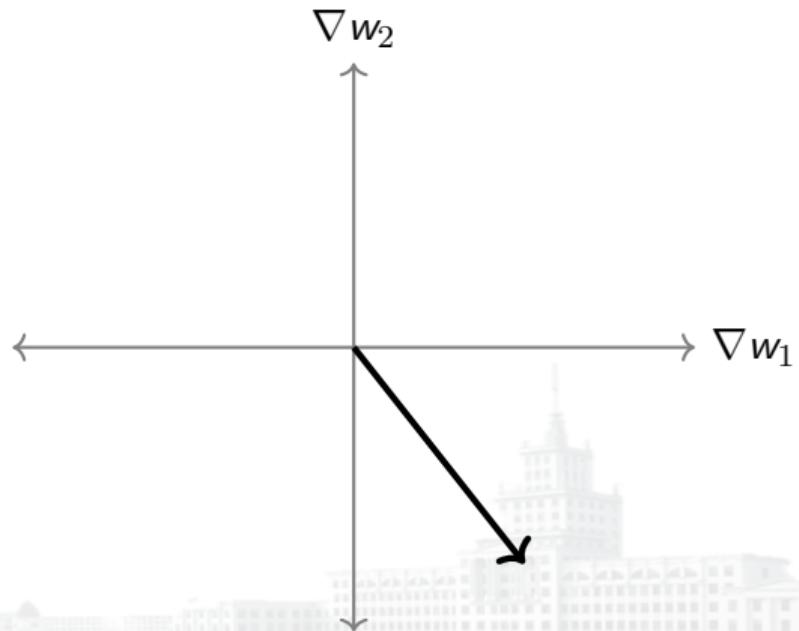
- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered

- 这限制了参数可能的更新方向



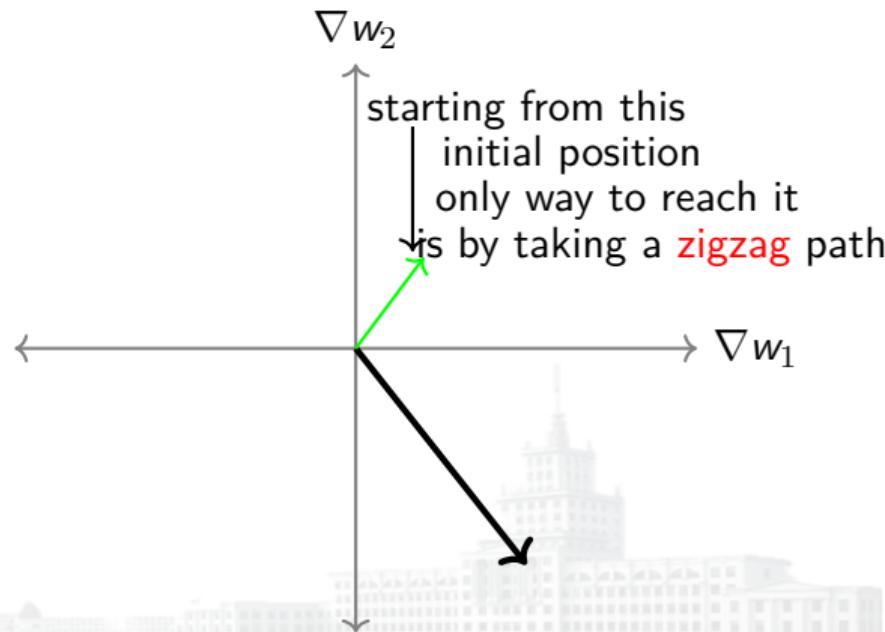


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered



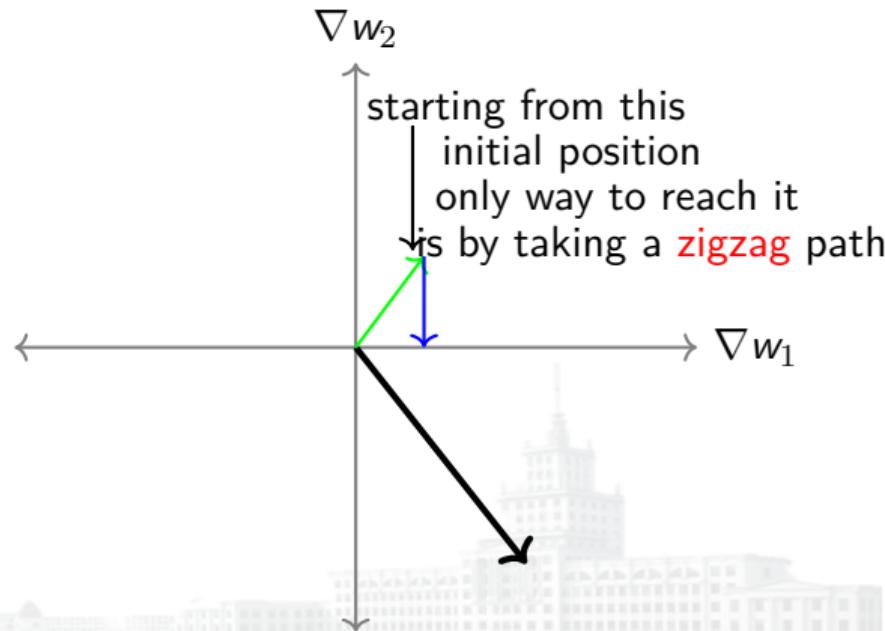


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered



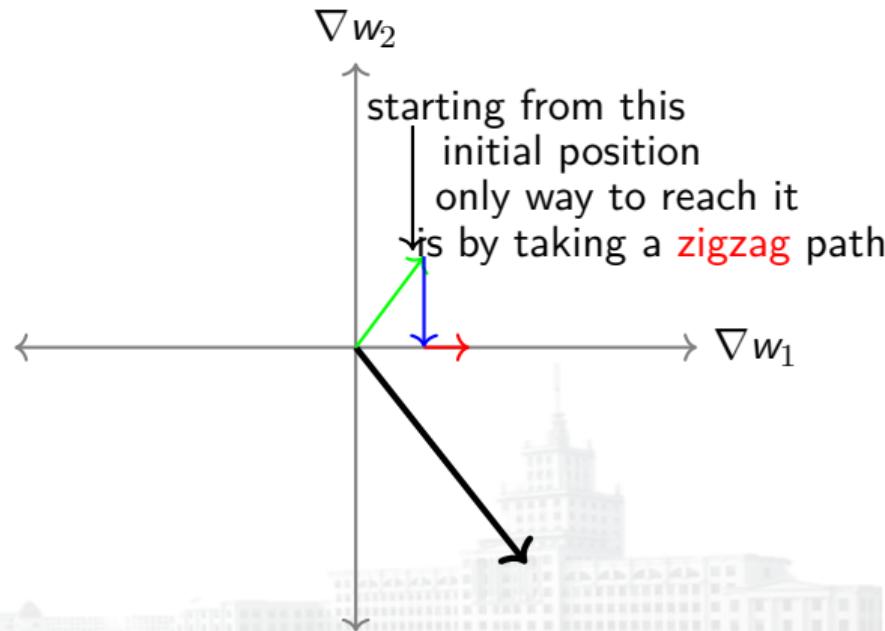


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered



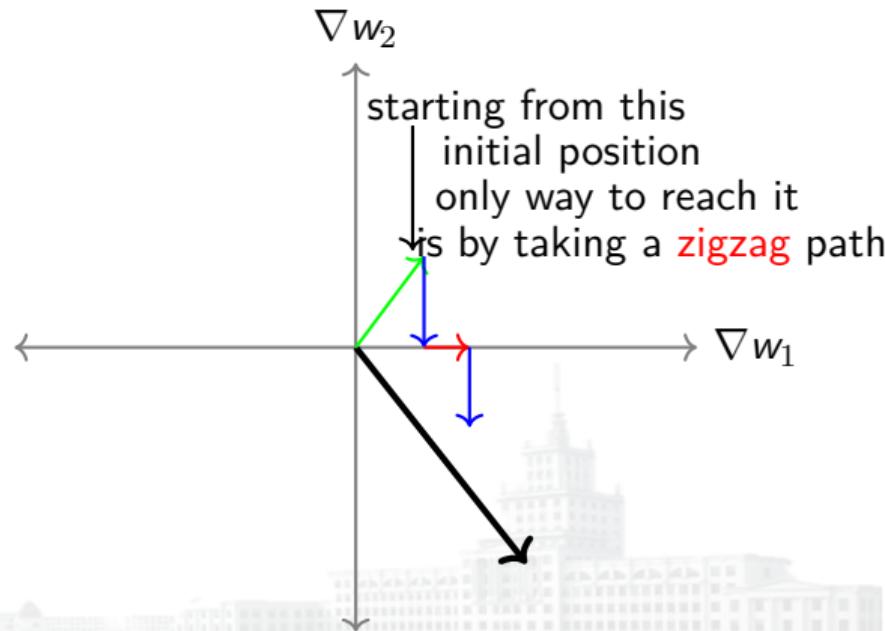


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered



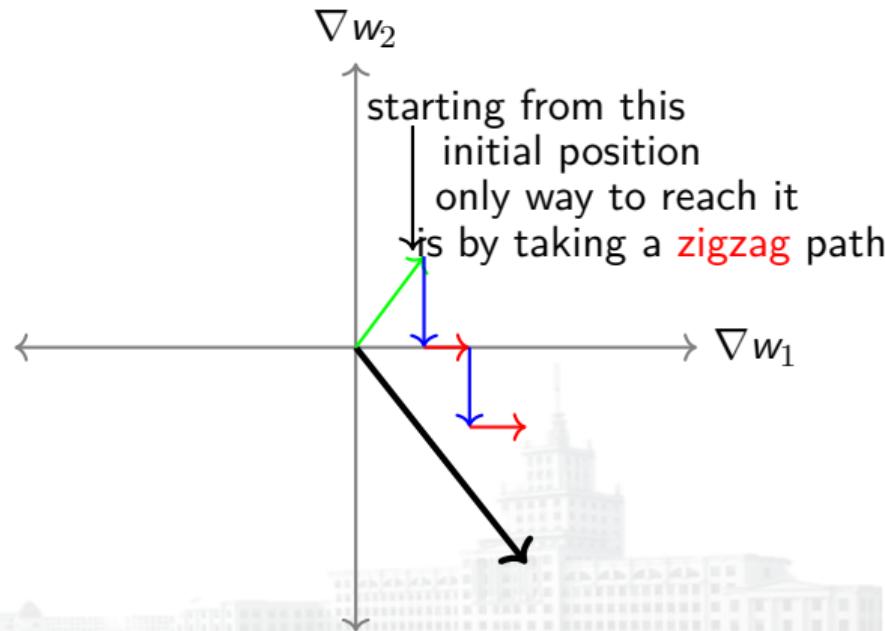


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered



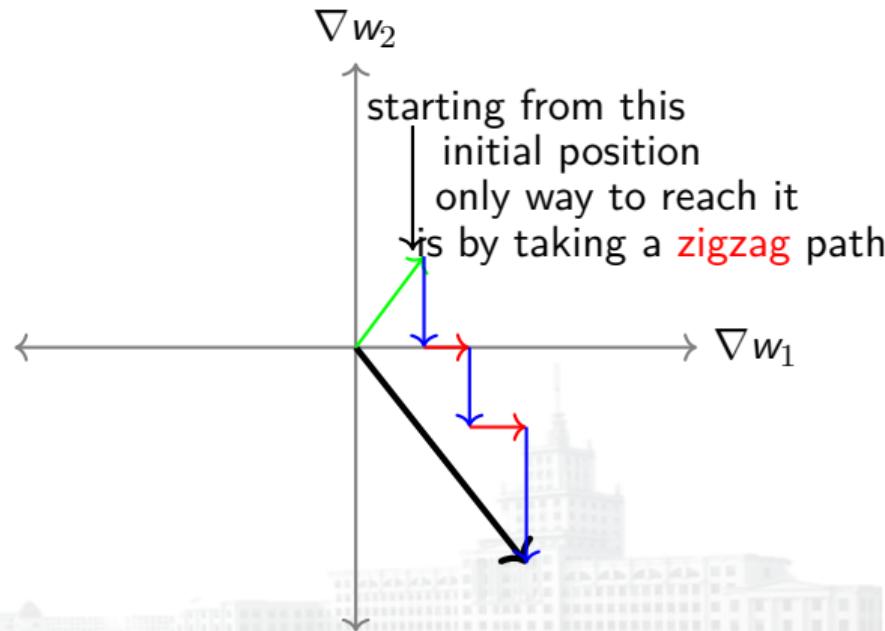


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered



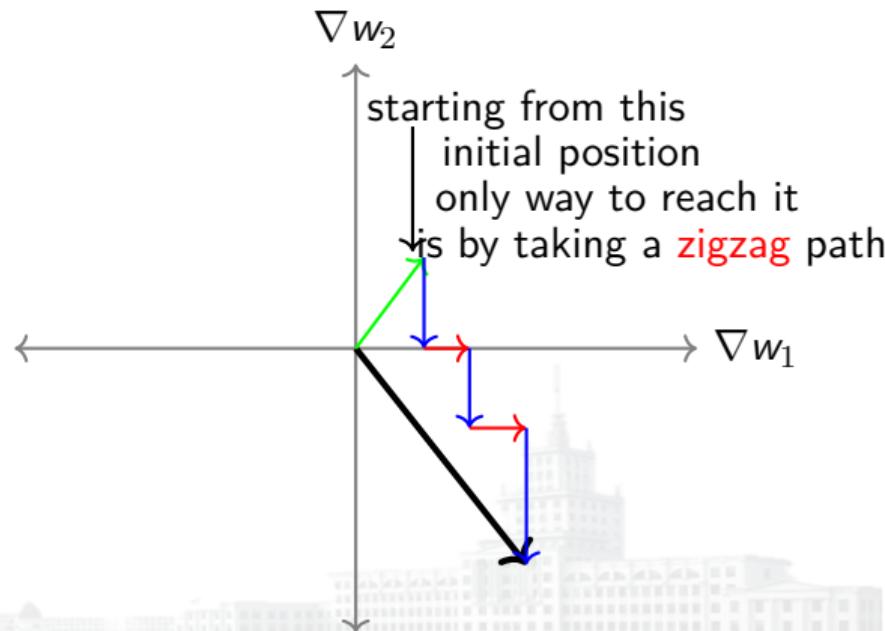


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered



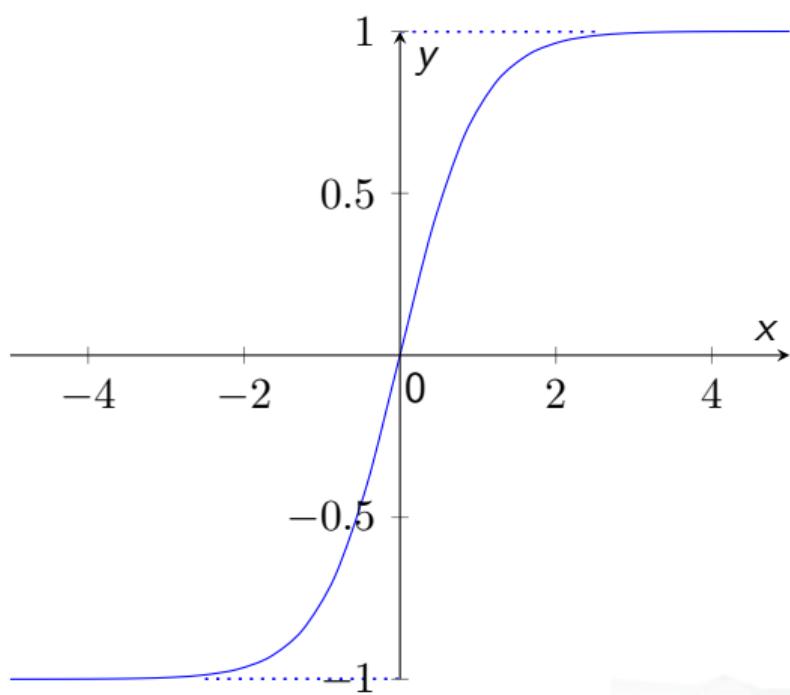


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered
- 最后, sigmoids 需要花费大量的计算 (因为  $\exp(x)$ )



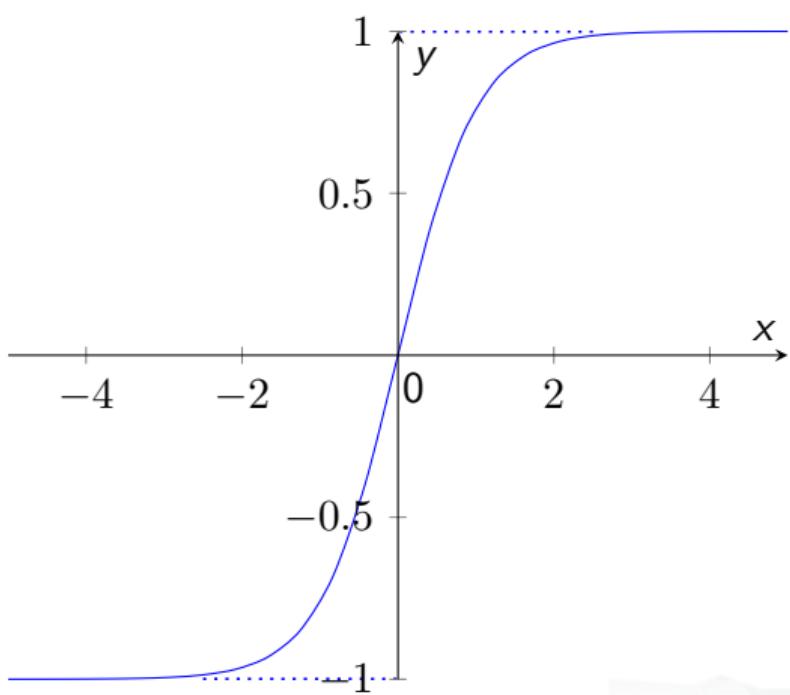


$\tanh(x)$



- 将输入变换到范围  $[-1,1]$

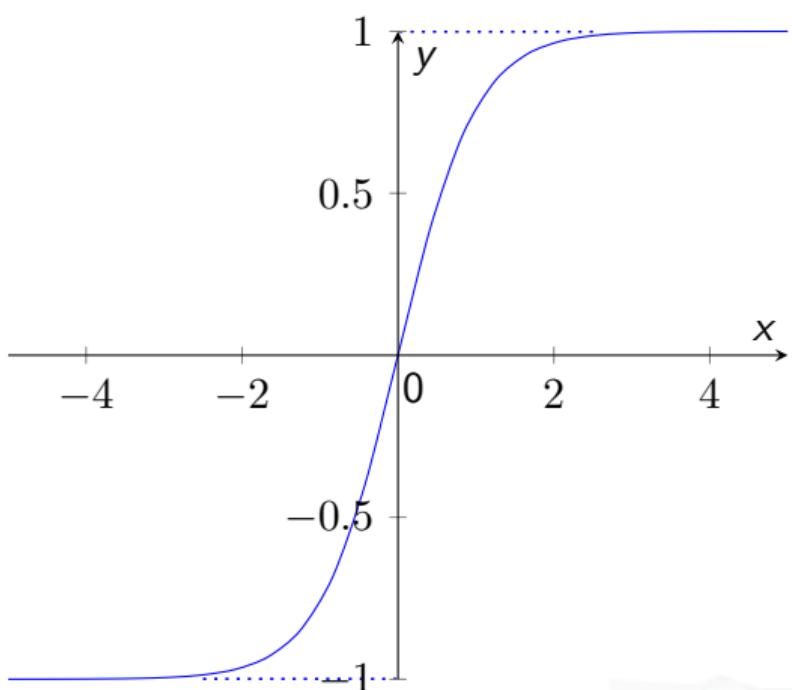
$\tanh(x)$



- 将输入变换到范围 [-1,1]
- 0 中心化的



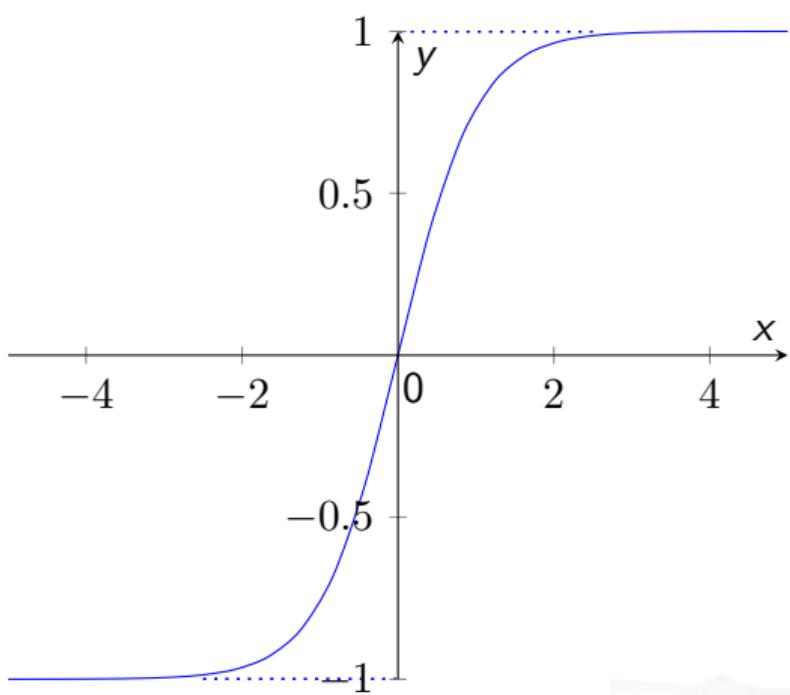
$\tanh(x)$



- 将输入变换到范围 [-1,1]
- 0 中心化的
- 梯度计算如下



tanh(x)

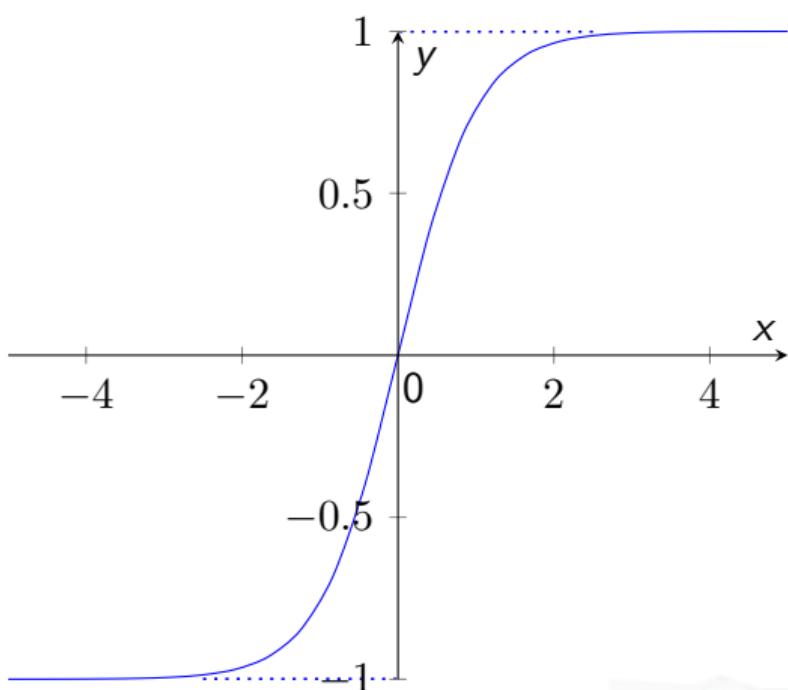


- 将输入变换到范围 [-1,1]
- 0 中心化的
- 梯度计算如下

$$\frac{\partial \tanh(x)}{\partial x} = (1 - \tanh^2(x))$$



tanh(x)



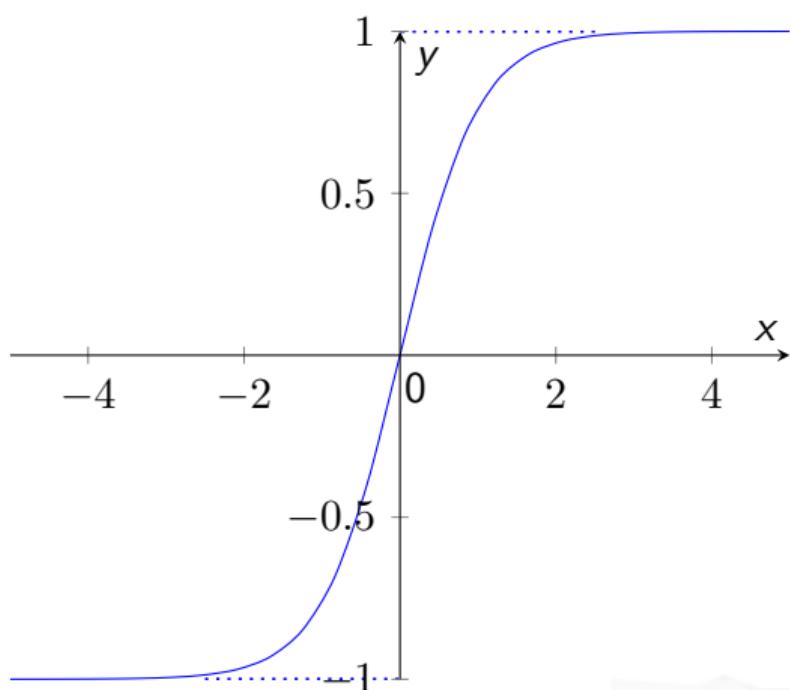
- 将输入变换到范围 [-1,1]
- 0 中心化的
- 梯度计算如下

$$\frac{\partial \tanh(x)}{\partial x} = (1 - \tanh^2(x))$$

- 在饱和点处，梯度也会消失



tanh(x)



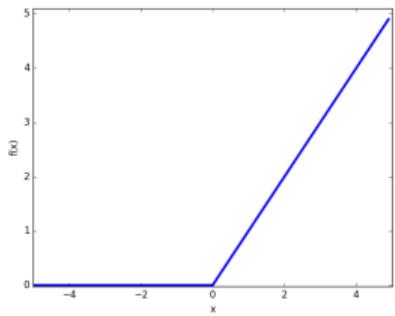
- 将输入变换到范围 [-1,1]
- 0 中心化的
- 梯度计算如下

$$\frac{\partial \tanh(x)}{\partial x} = (1 - \tanh^2(x))$$

- 在饱和点处，梯度也会消失
- 也需要大量的计算



## ReLU

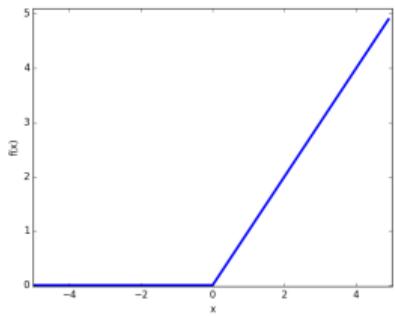


- 这是一个非线性函数

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



## ReLU

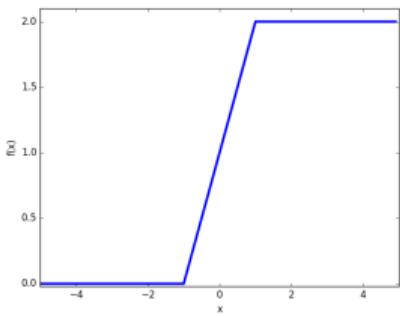


- 这是一个非线性函数
- 实际上，可以使用两个 ReLU 单元来近似一个 sigmoid 函数

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



## ReLU

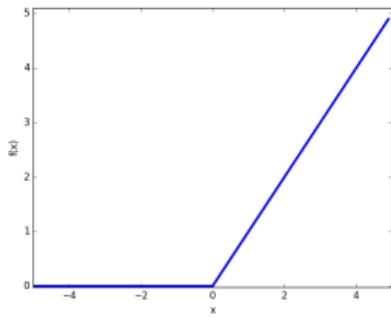


- 这是一个非线性函数
- 实际上，可以使用两个 ReLU 单元来近似一个 sigmoid 函数

$$f(x) = \max(0, x + 1) - \max(0, x - 1)$$



## ReLU



ReLU 的优点

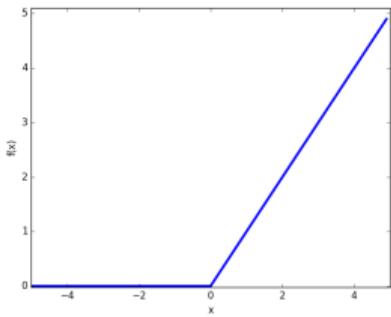
- 在正的区域，不会饱和

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

\*ImageNet Classification with Deep Convolutional Neural Networks- Alex Krizhevsky Ilya Sutskever, Geoffrey E. Hinton, 2012



## ReLU



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

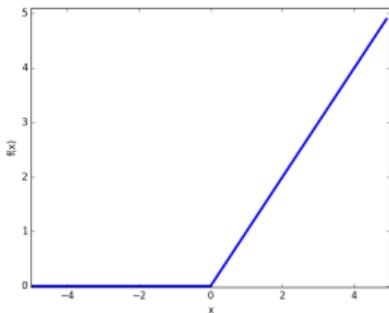
### ReLU 的优点

- 在正的区域，不会饱和
- 计算有效

\*ImageNet Classification with Deep Convolutional Neural Networks- Alex Krizhevsky Ilya Sutskever, Geoffrey E. Hinton, 2012



## ReLU



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

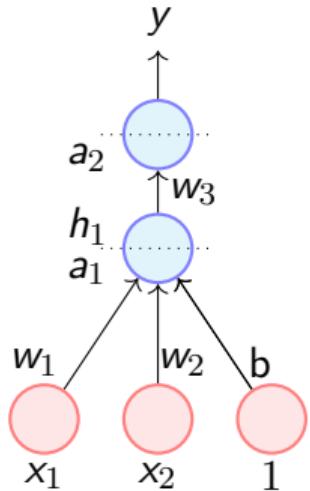
### ReLU 的优点

- 在正的区域，不会饱和
- 计算有效
- 比  $\text{sigmoid}/\tanh^1$  收敛的更快

\*ImageNet Classification with Deep Convolutional Neural Networks- Alex Krizhevsky Ilya Sutskever, Geoffrey E. Hinton, 2012

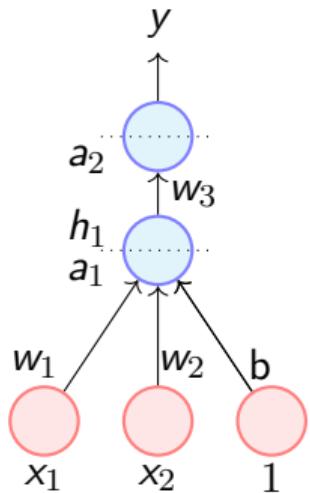


- 在实际使用中，有一个警告





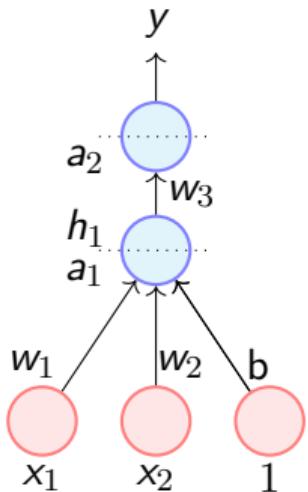
- 在实际使用中，有一个警告
- ReLU( $x$ ) 的微分



$$\begin{aligned}\frac{\partial \text{ReLU}(x)}{\partial x} &= 0 \quad \text{if } x < 0 \\ &= 1 \quad \text{if } x > 0\end{aligned}$$



- 在实际使用中，有一个警告
- ReLU( $x$ ) 的微分

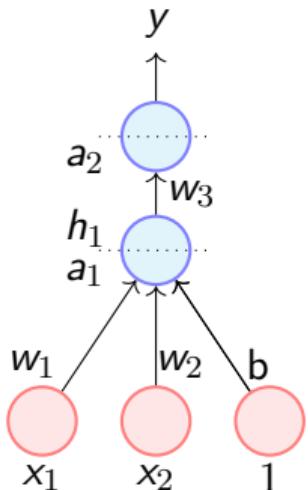


$$\begin{aligned}\frac{\partial \text{ReLU}(x)}{\partial x} &= 0 \quad \text{if } x < 0 \\ &= 1 \quad \text{if } x > 0\end{aligned}$$

- 考虑左边的网络



- 在实际使用中，有一个警告
- ReLU( $x$ ) 的微分



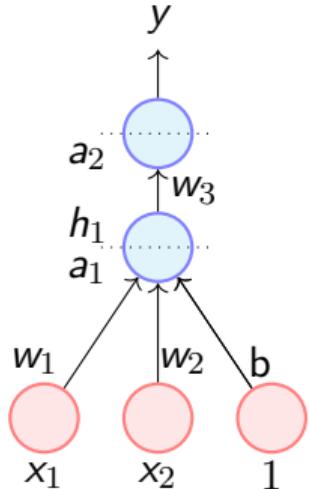
$$\begin{aligned}\frac{\partial \text{ReLU}(x)}{\partial x} &= 0 \quad \text{if } x < 0 \\ &= 1 \quad \text{if } x > 0\end{aligned}$$

- 考虑左边的网络
- 如果在某个点，一个大的梯度导致 bias  $b$  更新到一个大的负值



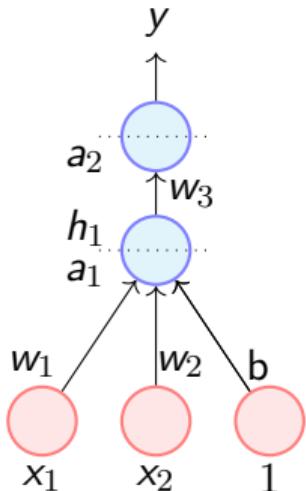
$$w_1x_1 + w_2x_2 + b < 0 \quad [if \quad b << 0]$$

- 神经元将输出 0 [dead neuron]





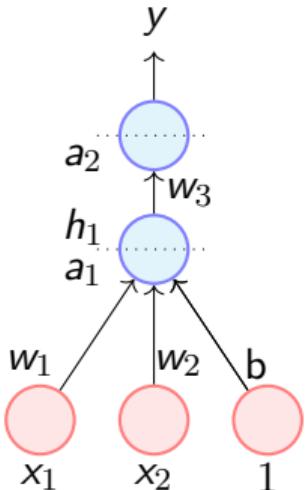
$$w_1x_1 + w_2x_2 + b < 0 \quad [if \quad b << 0]$$



- 神经元将输出 0 [dead neuron]
- 不仅仅是神经元的输出为 0, 在反向传播时, 梯度  $\frac{\partial h_1}{\partial a_1}$  也将为 0



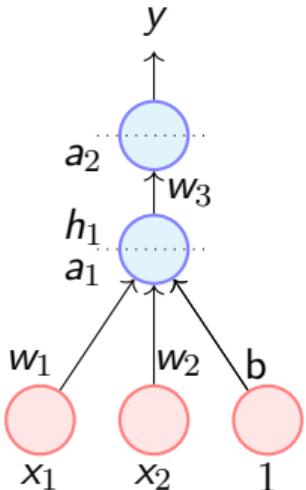
$$w_1x_1 + w_2x_2 + b < 0 \quad [if \quad b << 0]$$



- 神经元将输出 0 [dead neuron]
- 不仅仅是神经元的输出为 0, 在反向传播时, 梯度  $\frac{\partial h_1}{\partial a_1}$  也将为 0
- 权重  $w_1, w_2$  和  $b$  将不会得到更新

$$\nabla w_1 = \frac{\partial \mathcal{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

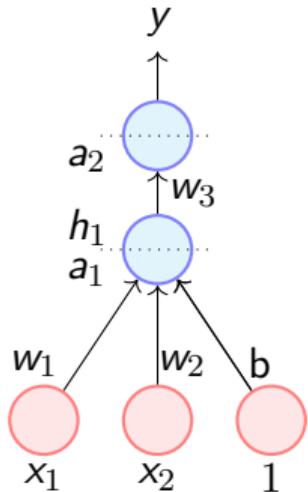
$$w_1x_1 + w_2x_2 + b < 0 \quad [if \quad b << 0]$$



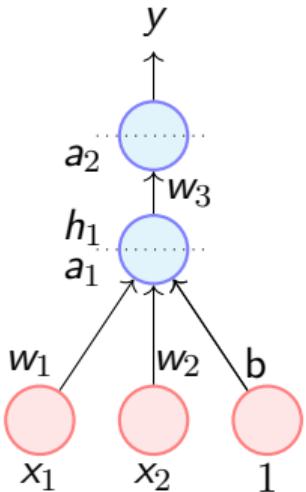
- 神经元将输出 0 [dead neuron]
- 不仅仅是神经元的输出为 0, 在反向传播时, 梯度  $\frac{\partial h_1}{\partial a_1}$  也将为 0
- 权重  $w_1, w_2$  和  $b$  将不会得到更新

$$\nabla w_1 = \frac{\partial \mathcal{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

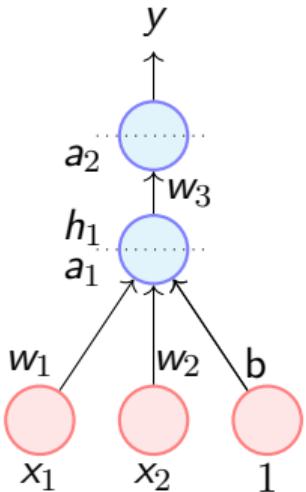
- 神经元将一直输出 0 (一直是死亡状态) !!



- 在实际中，如果学习率设的过高，会导致大部分 ReLU 神经元死亡



- 在实际中，如果学习率设的过高，会导致大部分 ReLU 神经元死亡
- 因此，建议将 bias 初始化为一个正值（例如，0.01）

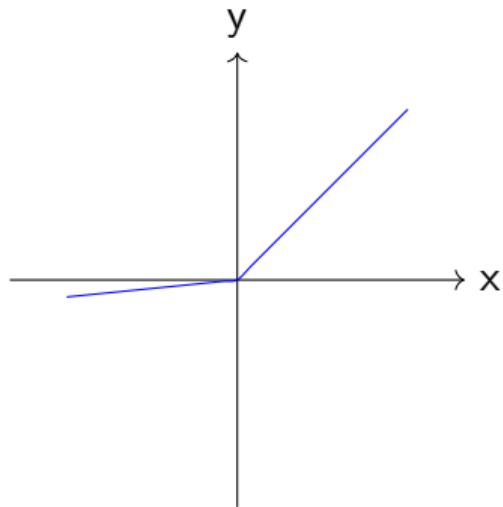


- 在实际中，如果学习率设的过高，会导致大部分 ReLU 神经元死亡
- 因此，建议将 bias 初始化为一个正值（例如，0.01）
- 使用 ReLU 的其他变体



## Leaky ReLU

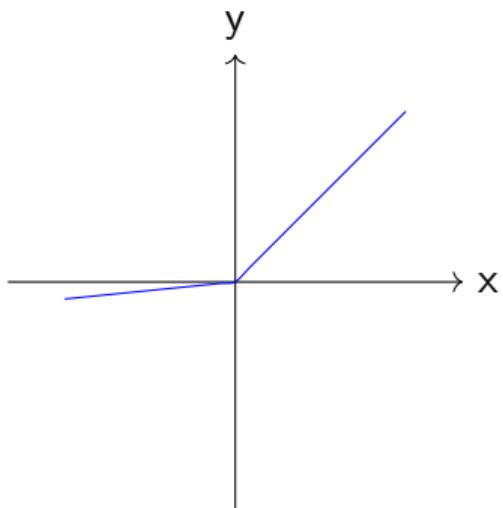
▪ 没有饱和



$$f(x) = \max(0.01x, x)$$



## Leaky ReLU

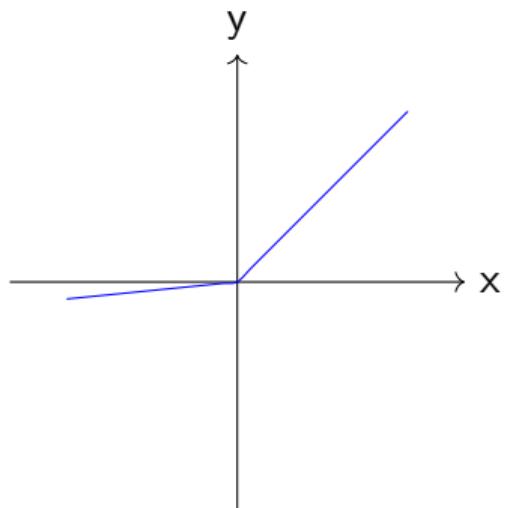


- 没有饱和
- 不会死亡 (0.01x 确保至少一个小的梯度将会传递)

$$f(x) = \max(0.01x, x)$$



## Leaky ReLU

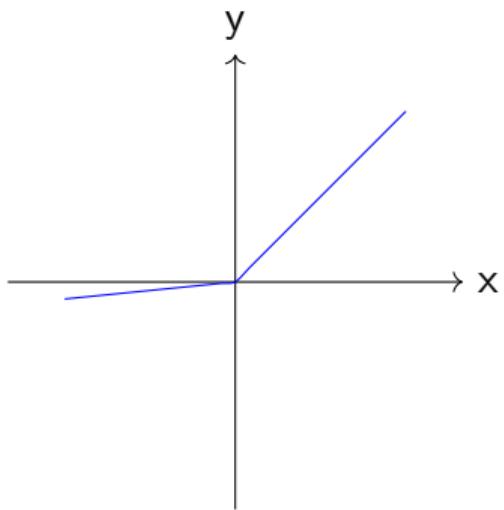


$$f(x) = \max(0.01x, x)$$

- 没有饱和
- 不会死亡 (0.01x 确保至少一个小的梯度将会传递)
- 计算有效



## Leaky ReLU

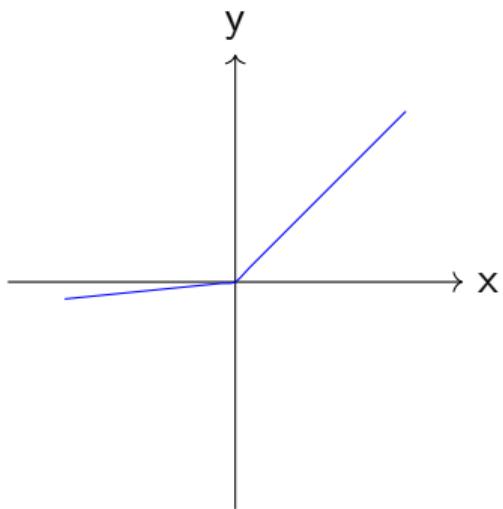


$$f(x) = \max(0.01x, x)$$

- 没有饱和
- 不会死亡 (0.01x 确保至少一个小的梯度将会传递)
- 计算有效
- 接近 0 中心输出



## Leaky ReLU



$$f(x) = \max(0.01x, x)$$

- 没有饱和
- 不会死亡 (0.01x 确保至少一个小的梯度将会传递)
- 计算有效
- 接近 0 中心输出

## Parametric ReLU

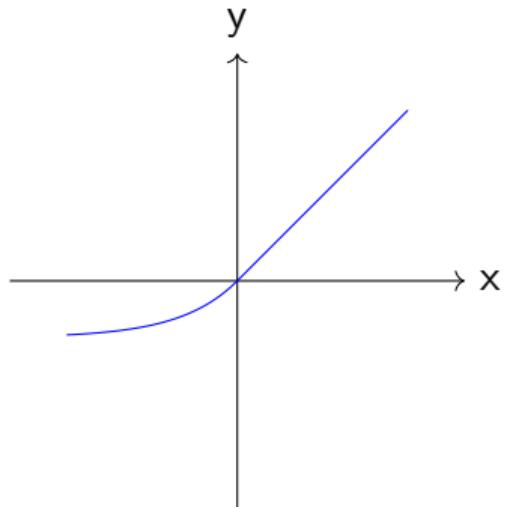
$$f(x) = \max(\alpha x, x)$$

$\alpha$  是模型的参数

$\alpha$  将在反向传播中更新



## Exponential Linear Unit

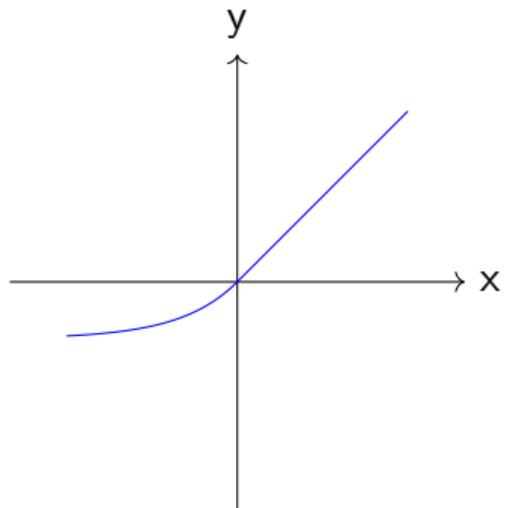


- 拥有 ReLU 的所有优点

$$\begin{aligned}f(x) &= x \quad \text{if } x > 0 \\&= ae^x - 1 \quad \text{if } x \leq 0\end{aligned}$$



## Exponential Linear Unit

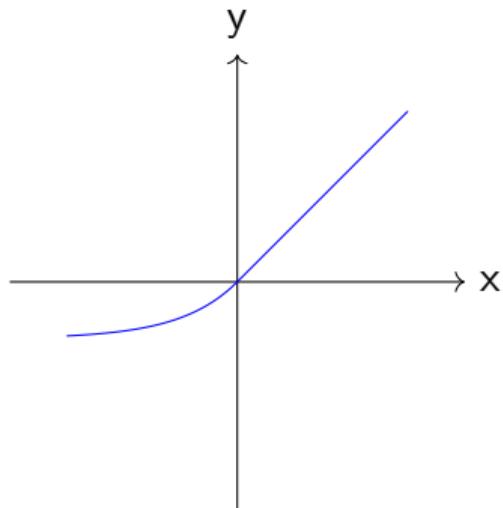


- 拥有 ReLU 的所有优点
- $ae^x - 1$  确保至少一个小的梯度将会传递

$$\begin{aligned}f(x) &= x \quad \text{if } x > 0 \\&= ae^x - 1 \quad \text{if } x \leq 0\end{aligned}$$



## Exponential Linear Unit

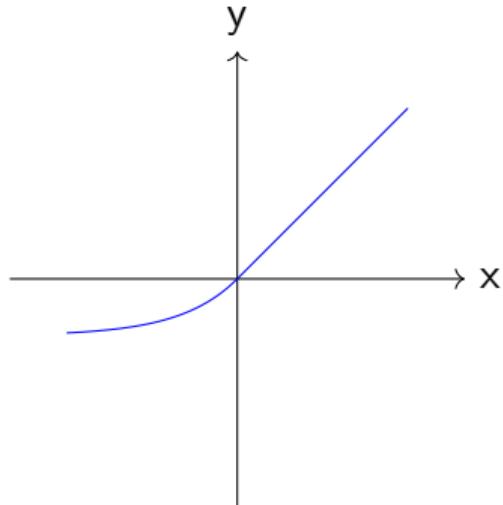


- 拥有 ReLU 的所有优点
- $ae^x - 1$  确保至少一个小的梯度将会传递
- 接近 0 中心输出

$$\begin{aligned}f(x) &= x \quad \text{if } x > 0 \\&= ae^x - 1 \quad \text{if } x \leq 0\end{aligned}$$



## Exponential Linear Unit



$$\begin{aligned}f(x) &= x \quad \text{if } x > 0 \\&= ae^x - 1 \quad \text{if } x \leq 0\end{aligned}$$

- 拥有 ReLU 的所有优点
- $ae^x - 1$  确保至少一个小的梯度将会传递
- 接近 0 中心输出
- 计算昂贵 (需要计算  $\exp(x)$ )



## Maxout Neuron

- ReLU 和 Leaky ReLU 是 Maxout Neuron 的特殊情况

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$



## Maxout Neuron

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- ReLU 和 Leaky ReLU 是 Maxout Neuron 的特殊情况
- No saturation! No death!



## Maxout Neuron

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- ReLU 和 Leaky ReLU 是 Maxout Neuron 的特殊情况
- No saturation! No death!
- Doubles the number of parameters



## Things to Remember

- Sigmoid 激活函数不好



## Things to Remember

- Sigmoid 激活函数不好
- ReLU 是 CNN 的标准激活函数



## Things to Remember

- Sigmoid 激活函数不好
- ReLU 是 CNN 的标准激活函数
- 可以尝试 Leaky ReLU/Maxout/ELU



## Things to Remember

- Sigmoid 激活函数不好
- ReLU 是 CNN 的标准激活函数
- 可以尝试 Leaky ReLU/Maxout/ELU
- tanh 和 sigmoids 仍然在 LSTMs/RNNs 中使用



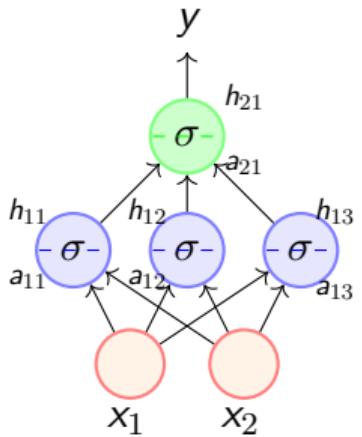
# 更好的初始化策略

## Deep Learning has evolved

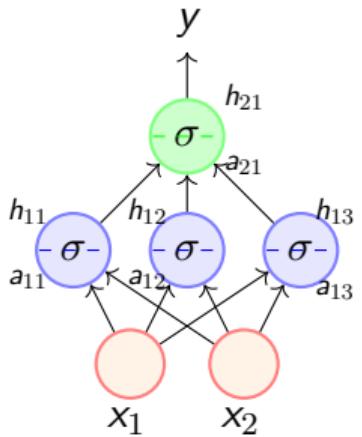
- Better optimization algorithms
- Better regularization methods
- Better activation functions
- **Better weight initialization strategies**



- 如果把所有的权重都初始化为 0，会怎样？

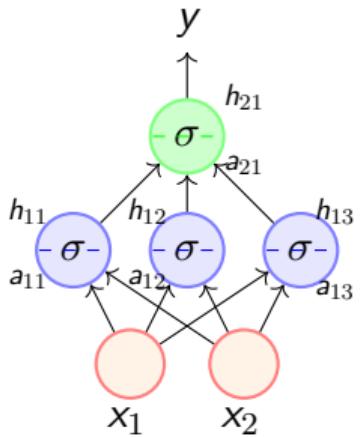


- 如果把所有的权重都初始化为 0，会怎样？



$$a_{11} = w_{11}x_1 + w_{12}x_2$$

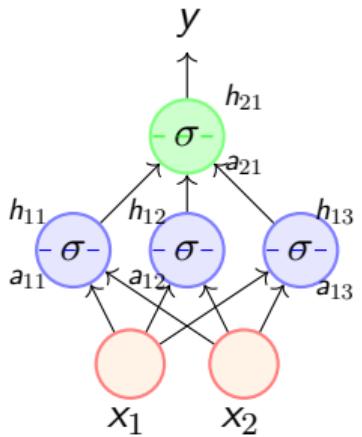
- 如果把所有的权重都初始化为 0，会怎样？



$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

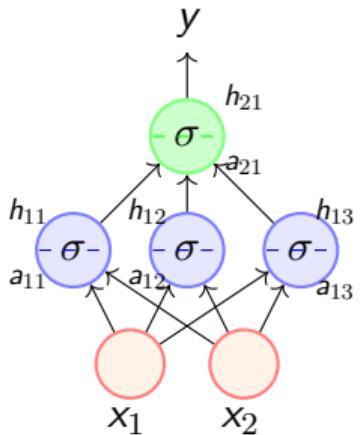
- 如果把所有的权重都初始化为 0，会怎样？



$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$



- 如果把所有的权重都初始化为 0，会怎样？
- 第一层的所有神经元将得到同样的激活

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

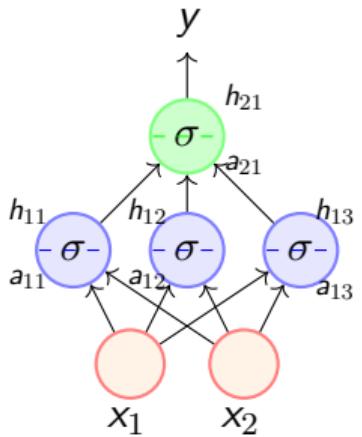
$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



▪ 反向传播会发生什么事？

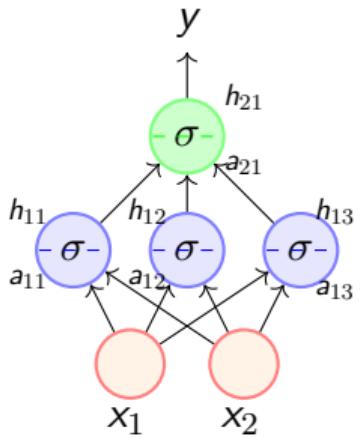


$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



▪ 反向传播会发生什么事 ?

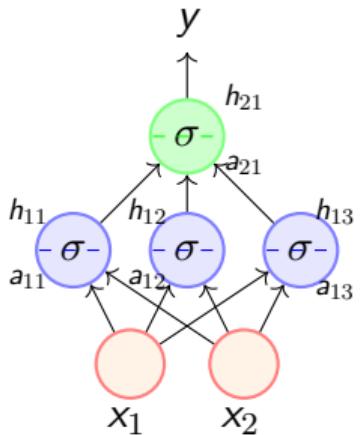
$$\nabla w_{11} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} \cdot x_1$$

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



▪ 反向传播会发生什么事？

$$\nabla w_{11} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} \cdot x_1$$

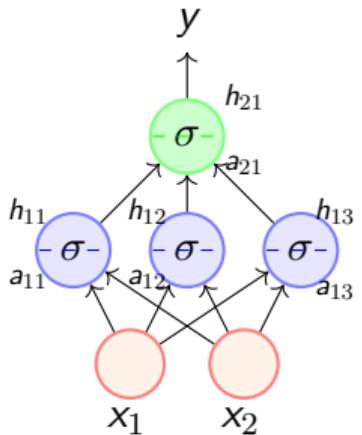
$$\nabla w_{21} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{12}} \cdot \frac{\partial h_{12}}{\partial a_{12}} \cdot x_1$$

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



▪ 反向传播会发生什么事 ?

$$\nabla w_{11} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} \cdot x_1$$

$$\nabla w_{21} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{12}} \cdot \frac{\partial h_{12}}{\partial a_{12}} \cdot x_1$$

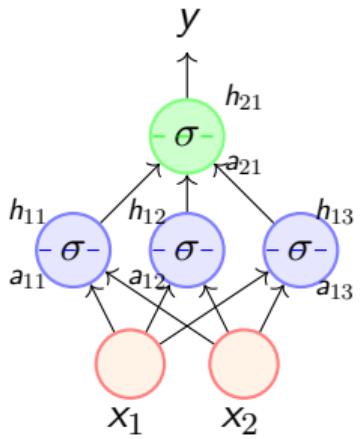
but  $h_{11} = h_{12}$

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



- 反向传播会发生什么事？

$$\nabla w_{11} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} \cdot x_1$$

$$\nabla w_{21} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{12}} \cdot \frac{\partial h_{12}}{\partial a_{12}} \cdot x_1$$

but  $h_{11} = h_{12}$

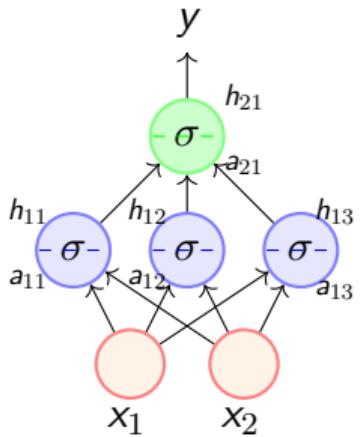
and  $a_{12} = a_{11}$

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$

▪ 反向传播会发生什么事？

$$\nabla w_{11} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{11}} \cdot \frac{\partial h_{11}}{\partial a_{11}} \cdot x_1$$

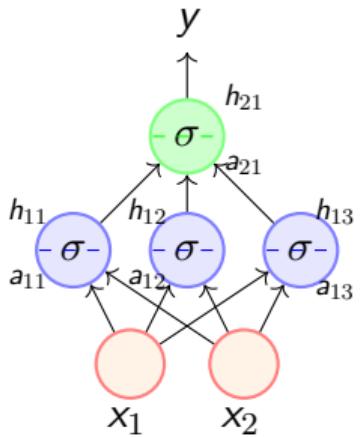
$$\nabla w_{21} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial h_{12}} \cdot \frac{\partial h_{12}}{\partial a_{12}} \cdot x_1$$

$$but \quad h_{11} = h_{12}$$

$$and \quad a_{12} = a_{11}$$

$$\therefore \nabla w_{11} = \nabla w_{21}$$

- 因此，这两个权重将得到同样的更新，保持相同

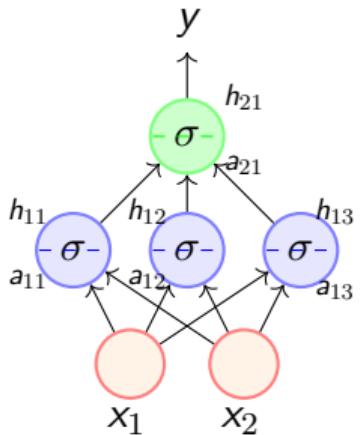


$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



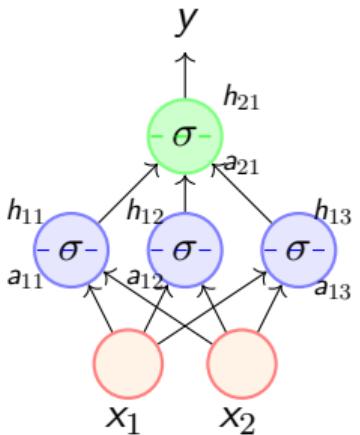
- 因此，这两个权重将得到同样的更新，保持相同
- 事实上，这两个权重将在整个训练过程中保持一样

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



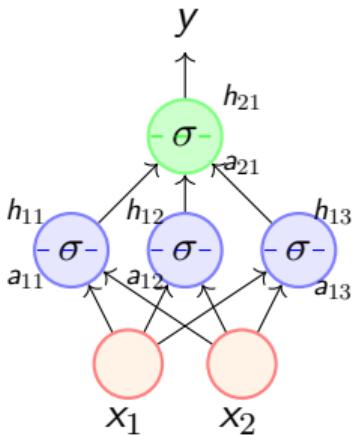
- 因此，这两个权重将得到同样的更新，保持相同
- 事实上，这两个权重将在整个训练过程中保持一样
- $w_{12}$  和  $w_{22}$  也是类似

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



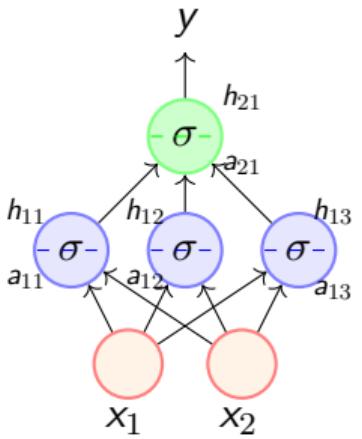
- 因此，这两个权重将得到同样的更新，保持相同
- 事实上，这两个权重将在整个训练过程中保持一样
- $w_{12}$  和  $w_{22}$  也是类似
- 第二层的权重也将保持相同

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



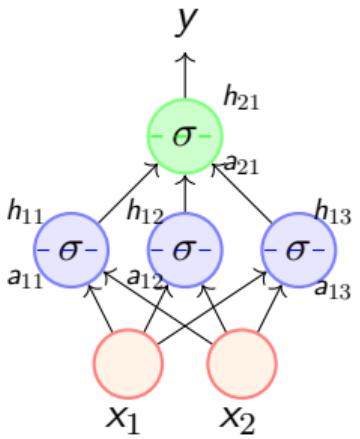
- 因此，这两个权重将得到同样的更新，保持相同
- 事实上，这两个权重将在整个训练过程中保持一样
- $w_{12}$  和  $w_{22}$  也是类似
- 第二层的权重也将保持相同
- 这就是 **symmetry breaking problem**

$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$



$$a_{11} = w_{11}x_1 + w_{12}x_2$$

$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12} = 0$$

$$\therefore h_{11} = h_{12}$$

- 因此，这两个权重将得到同样的更新，保持相同
- 事实上，这两个权重将在整个训练过程中保持一样
- $w_{12}$  和  $w_{22}$  也是类似
- 第二层的权重也将保持相同
- 这就是 **symmetry breaking problem**
- 当一个神经网络的所有权重被初始化同样的值时，就会发生这个问题

## 考虑一个前馈神经网络：

```
D = np.random.randn(1000,500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1]
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```



```
D = np.random.randn(1000,500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1]
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

## 考虑一个前馈神经网络：

- input: 1000 points, each  $\in R^{500}$



```
D = np.random.randn(1000,500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

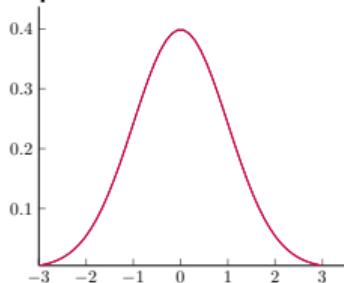
act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1]
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

## 考虑一个前馈神经网络：

- input: 1000 points, each  $\in R^{500}$
- input data is drawn from unit Gaussian





```
D = np.random.randn(1000,500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

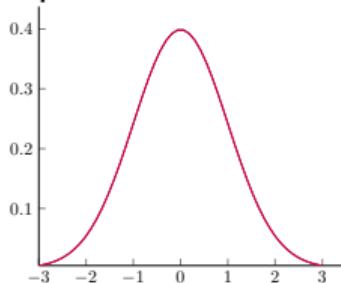
act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1]
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

## 考虑一个前馈神经网络：

- input: 1000 points, each  $\in R^{500}$
- input data is drawn from unit Gaussian



- the network has 5 layers



```
D = np.random.randn(1000,500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

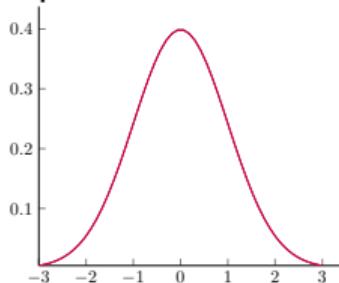
act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1]
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

## 考虑一个前馈神经网络：

- input: 1000 points, each  $\in R^{500}$
- input data is drawn from unit Gaussian



- the network has 5 layers
- each layer has 500 neurons



```
D = np.random.randn(1000,500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

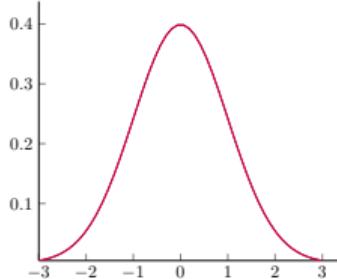
act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1]
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

## 考虑一个前馈神经网络：

- input: 1000 points, each  $\in R^{500}$
- input data is drawn from unit Gaussian



- the network has 5 layers
- each layer has 500 neurons
- 使用不同的权重初始化，计算网络的前向传播

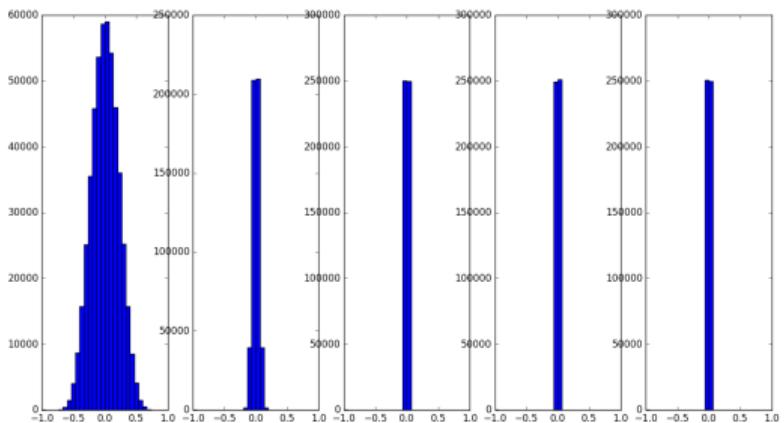
```
W = np.random.randn(fan_in, fan_out) * 0.01
```

- 将权重初始化为小的随机数



```
W = np.random.randn(fan_in, fan_out) * 0.01
```

- 将权重初始化为小的随机数
- 看看不同层的激活情况

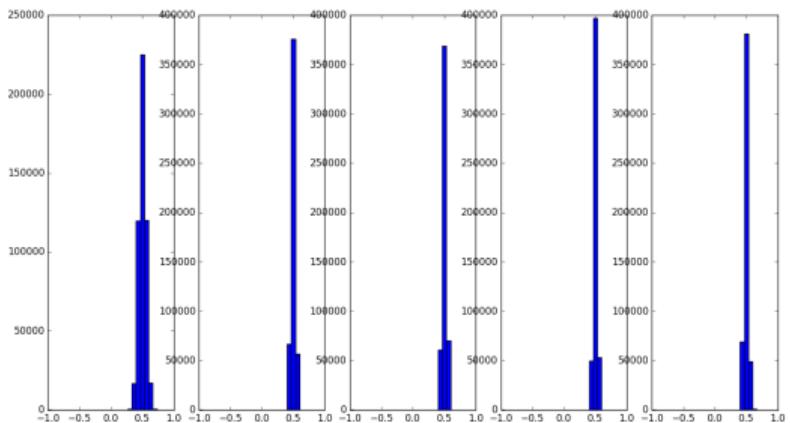


tanh activation functions



```
W = np.random.randn(fan_in, fan_out) * 0.01
```

- 将权重初始化为小的随机数
- 看看不同层的激活情况



sigmoid activation functions



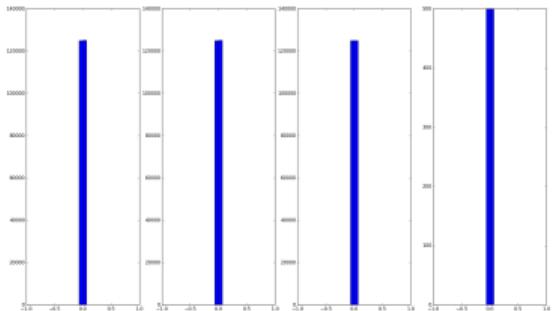
- 反向传播会发生什么?



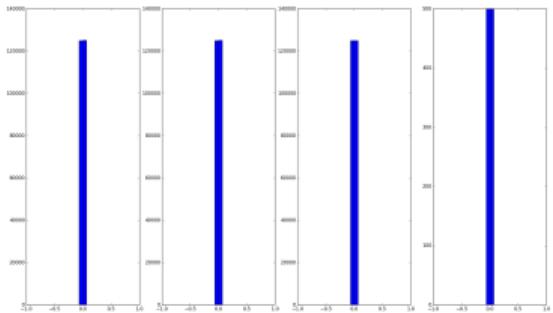
- 反向传播会发生什么?
- 记住  $\nabla w_1$  正比于由它传递的激活



- 反向传播会发生什么?
- 记住  $\nabla w_1$  正比于由它传递的激活
- 如果某一层的所有激活都非常接近 0, 连接这一层和下一层的权重的梯度会是什么?



- 反向传播会发生什么?
- 记住  $\nabla w_1$  正比于由它传递的激活
- 如果某一层的所有激活都非常接近 0, 连接这一层和下一层的权重的梯度会是什么?



- 反向传播会发生什么?
- 记住  $\nabla w_1$  正比于由它传递的激活
- 如果某一层的所有激活都非常接近 0, 连接这一层和下一层的权重的梯度会是什么?
- 这些梯度也会接近 0 (梯度消失问题)



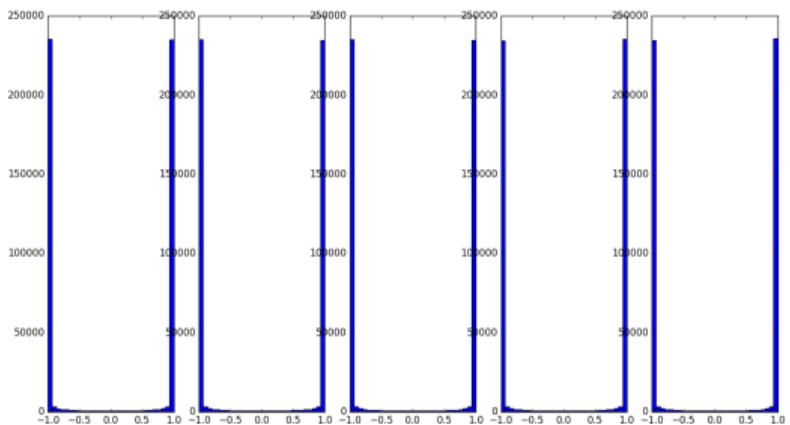
```
W = np.random.randn(fan_in, fan_out)
```

- 将权重初始化为大的随机数



```
W = np.random.randn(fan_in, fan_out)
```

- 将权重初始化为大的随机数

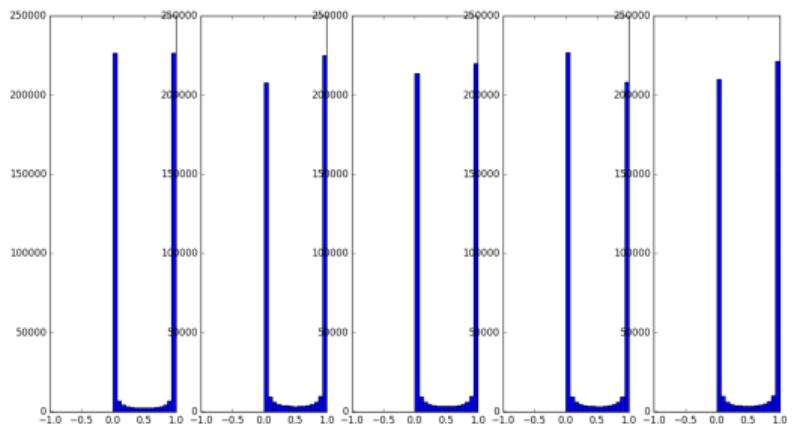


tanh activation with large weights



```
W = np.random.randn(fan_in, fan_out)
```

- 将权重初始化为大的随机数

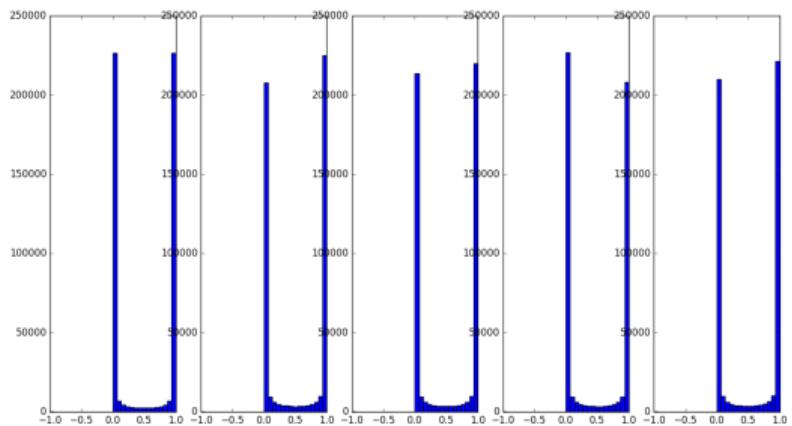


sigmoid activations with large weights



```
W = np.random.randn(fan_in, fan_out)
```

- 将权重初始化为大的随机数
- 大多数激活将达到饱和

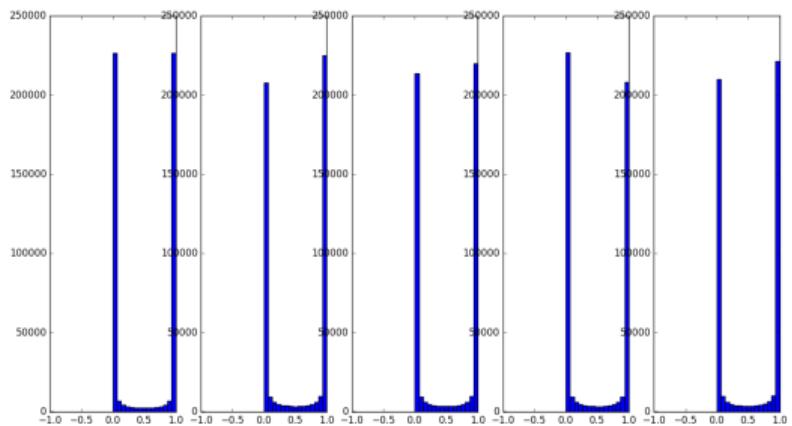


sigmoid activations with large weights



```
W = np.random.randn(fan_in, fan_out)
```

- 将权重初始化为大的随机数
- 大多数激活将达到饱和
- 在饱和的地方，梯度是多少？

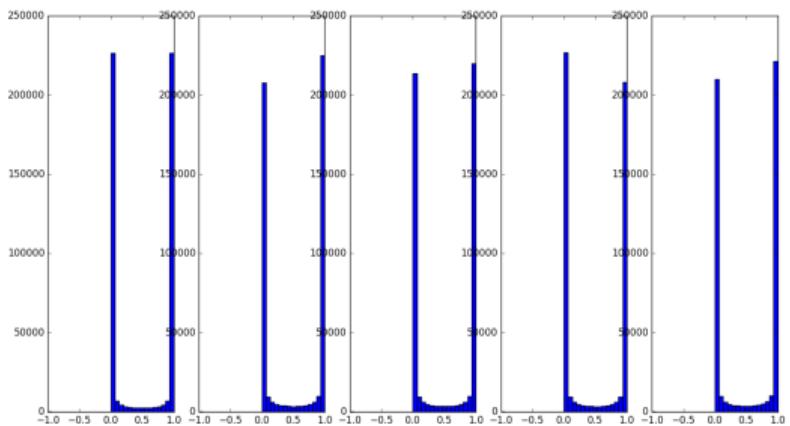


sigmoid activations with large weights



```
W = np.random.randn(fan_in, fan_out)
```

- 将权重初始化为大的随机数
- 大多数激活将达到饱和
- 在饱和的地方，梯度是多少？
- 梯度将接近 0 (梯度消失问题)



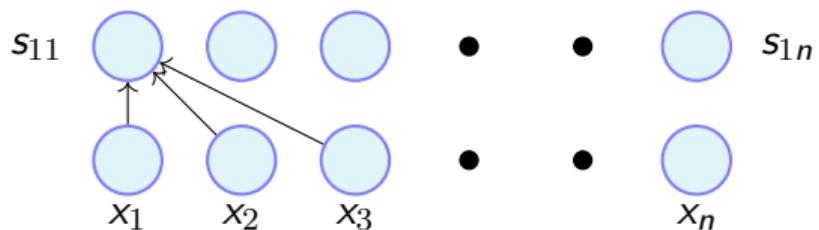
sigmoid activations with large weights



- 看一个更高级的权重初始化方法

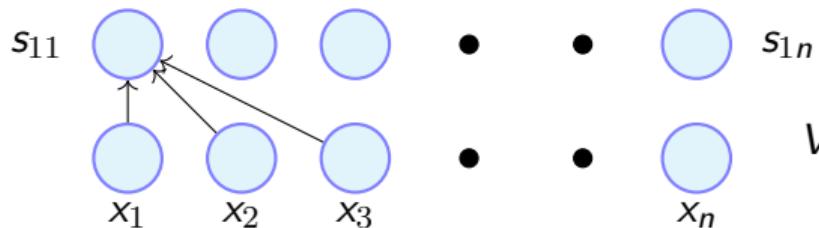


■ 看一个更高级的权重初始化方法



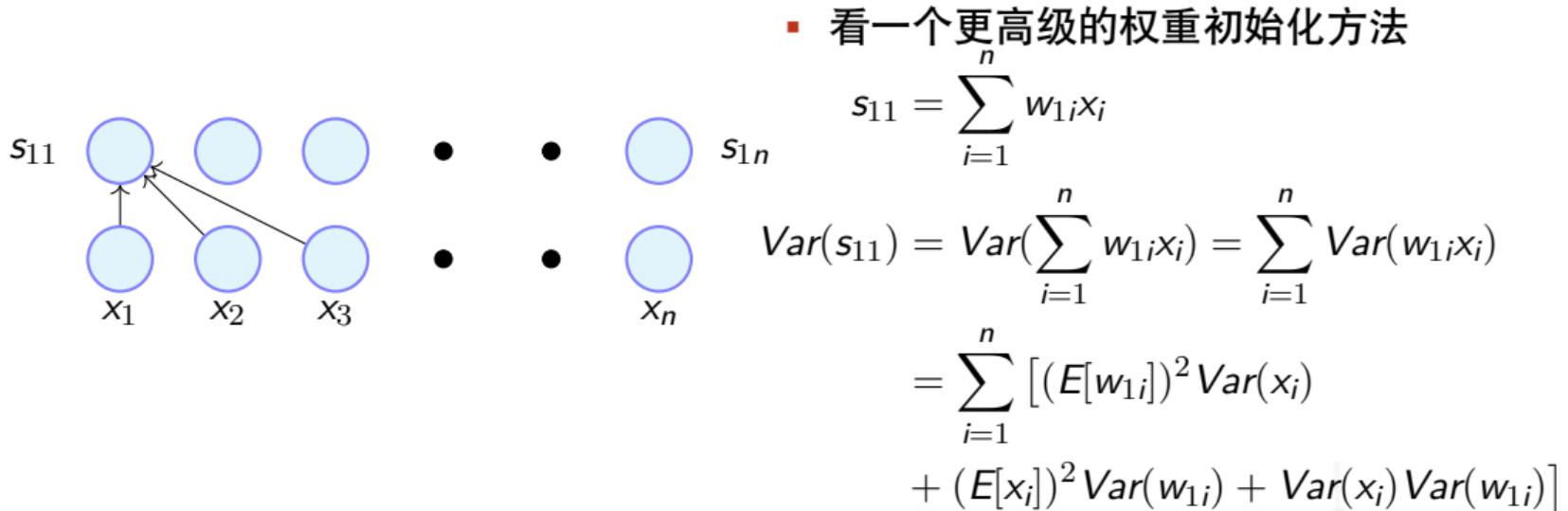
$$s_{11} = \sum_{i=1}^n w_{1i}x_i$$

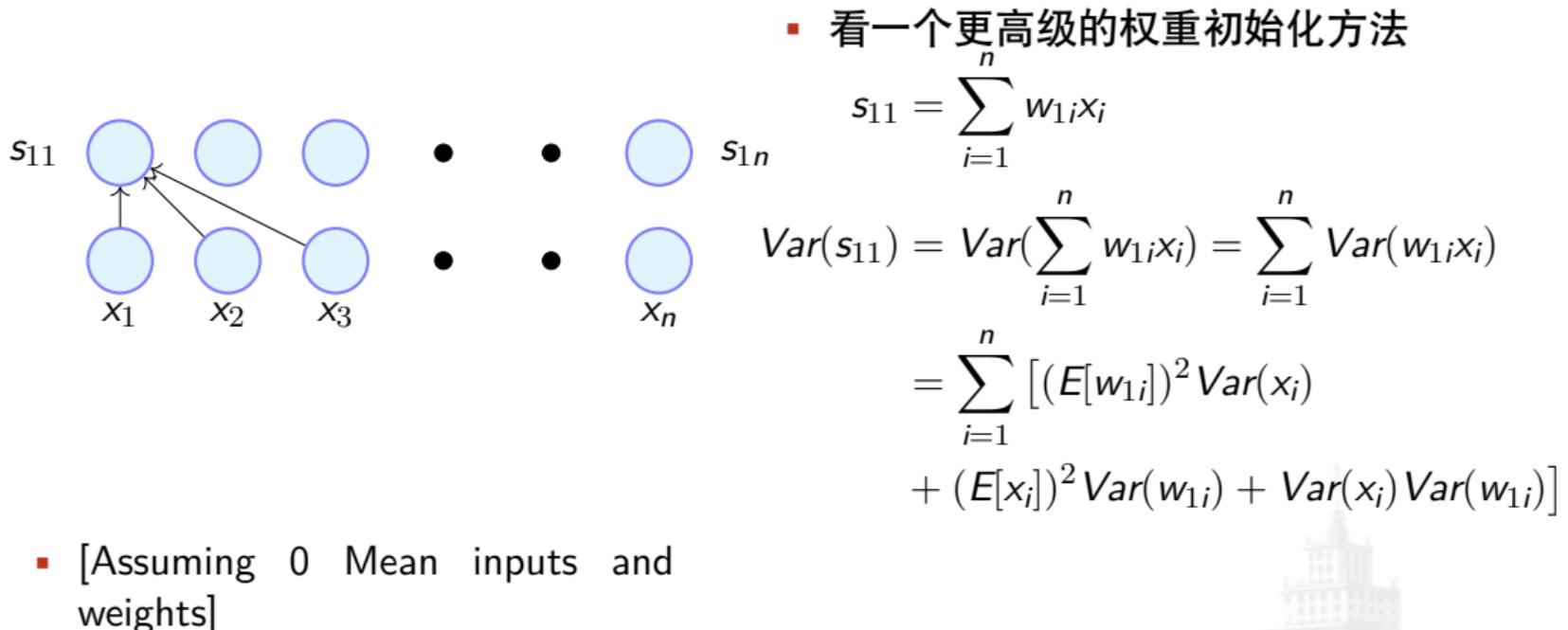
▪ 看一个更高级的权重初始化方法

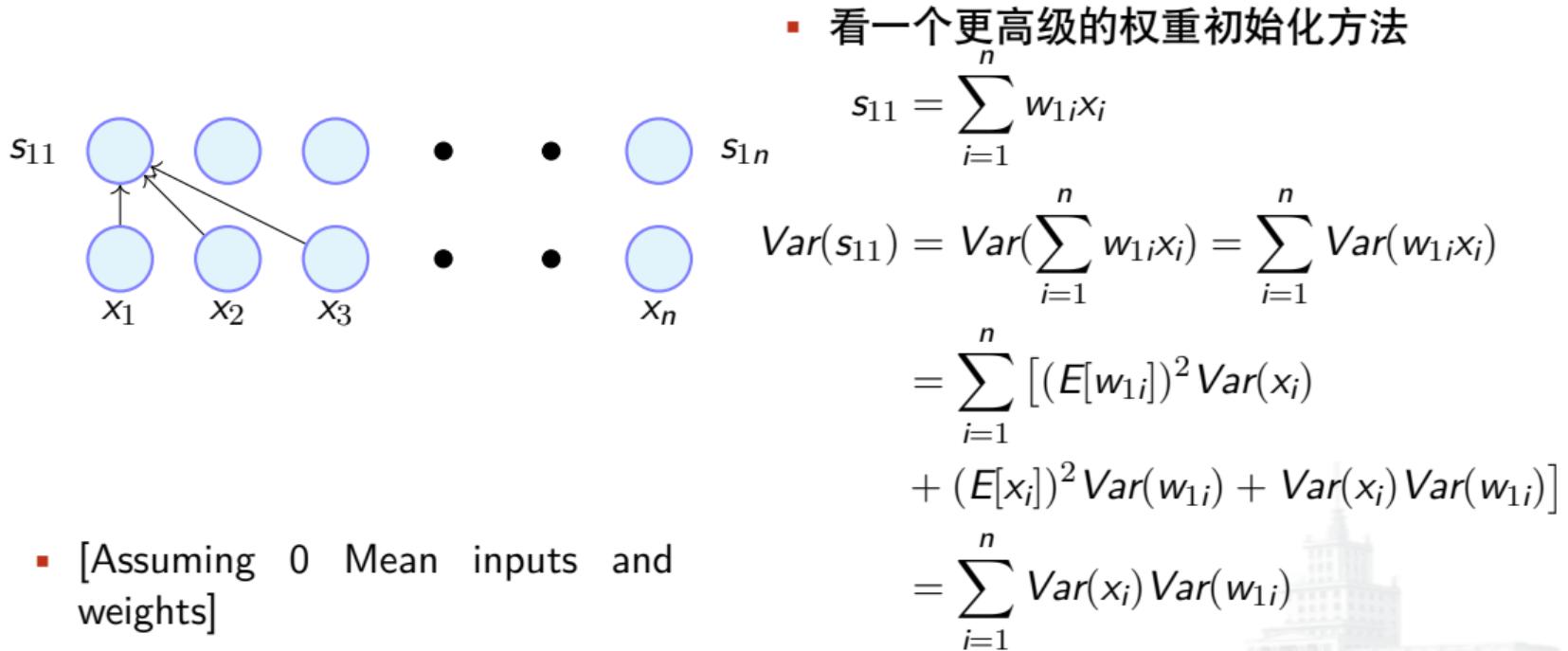


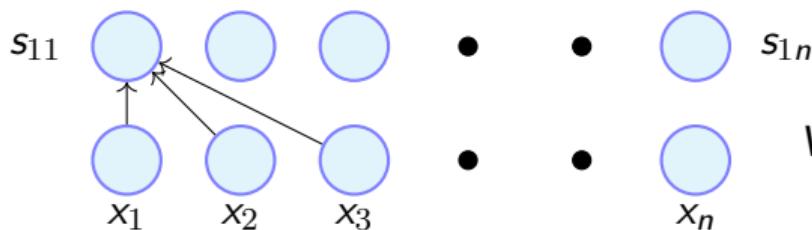
$$s_{11} = \sum_{i=1}^n w_{1i}x_i$$

$$\text{Var}(s_{11}) = \text{Var}\left(\sum_{i=1}^n w_{1i}x_i\right) = \sum_{i=1}^n \text{Var}(w_{1i}x_i)$$









■ 看一个更高级的权重初始化方法

$$s_{11} = \sum_{i=1}^n w_{1i}x_i$$

$$\text{Var}(s_{11}) = \text{Var}\left(\sum_{i=1}^n w_{1i}x_i\right) = \sum_{i=1}^n \text{Var}(w_{1i}x_i)$$

$$= \sum_{i=1}^n [(E[w_{1i}])^2 \text{Var}(x_i)$$

$$+ (E[x_i])^2 \text{Var}(w_{1i}) + \text{Var}(x_i) \text{Var}(w_{1i})]$$

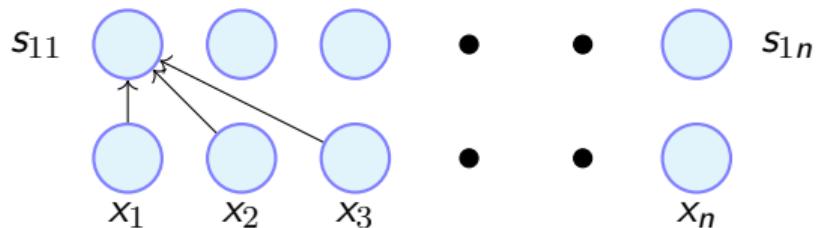
$$= \sum_{i=1}^n \text{Var}(x_i) \text{Var}(w_{1i})$$

$$= (n \text{Var}(w)) (\text{Var}(x))$$

- [Assuming 0 Mean inputs and weights]
- [Assuming  $\text{Var}(x_i) = \text{Var}(x) \forall i$ ]
- [Assuming  $\text{Var}(w_{1i}) = \text{Var}(w) \forall i$ ]



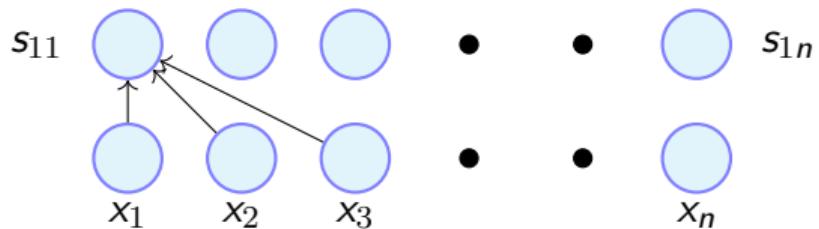
- In general,



$$Var(S_{1i}) = (nVar(w))(Var(x))$$



- In general,

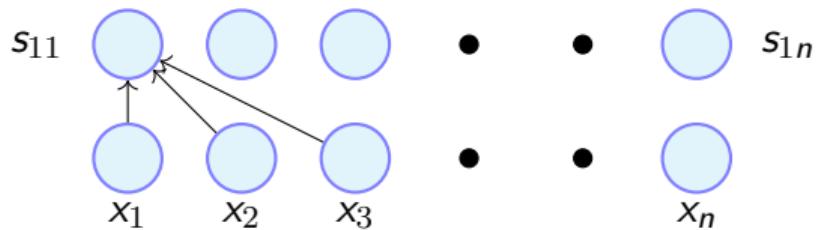


$$\text{Var}(S_{1i}) = (n\text{Var}(w))(\text{Var}(x))$$

- What would happen if  $n\text{Var}(w) \gg 1$ ?  
?



- In general,

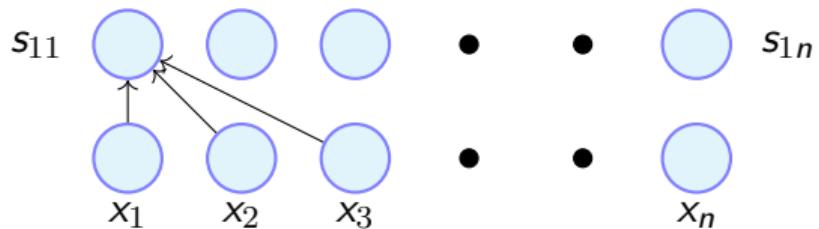


$$Var(S_{1i}) = (nVar(w))(Var(x))$$

- What would happen if  $nVar(w) \gg 1$  ?
- The variance of  $S_{1i}$  will be large



- In general,

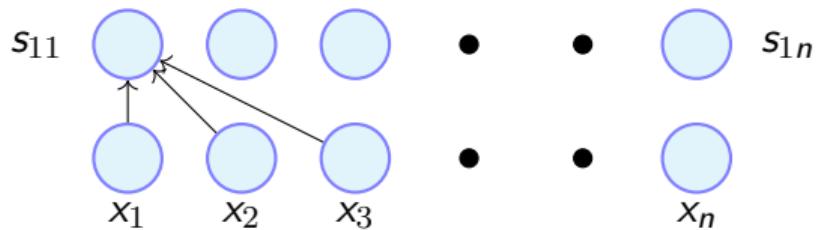


$$\text{Var}(S_{1i}) = (n\text{Var}(w))(\text{Var}(x))$$

- What would happen if  $n\text{Var}(w) \gg 1$  ?
- The variance of  $S_{1i}$  will be large
- What would happen if  $n\text{Var}(w) \rightarrow 0$ ?



- In general,

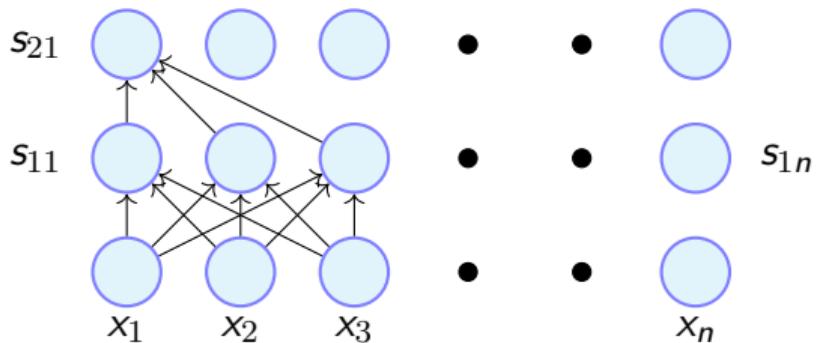


$$\text{Var}(S_{1i}) = (n\text{Var}(w))(\text{Var}(x))$$

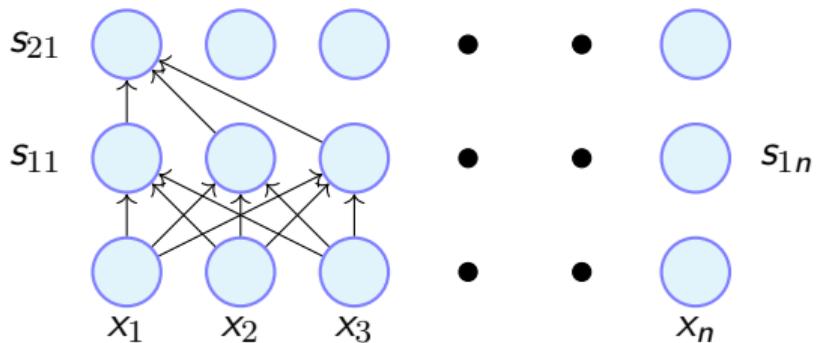
- What would happen if  $n\text{Var}(w) \gg 1$  ?
- The variance of  $S_{1i}$  will be large
- What would happen if  $n\text{Var}(w) \rightarrow 0$ ?
- The variance of  $S_{1i}$  will be small



- Let us see what happens if we add one more layer

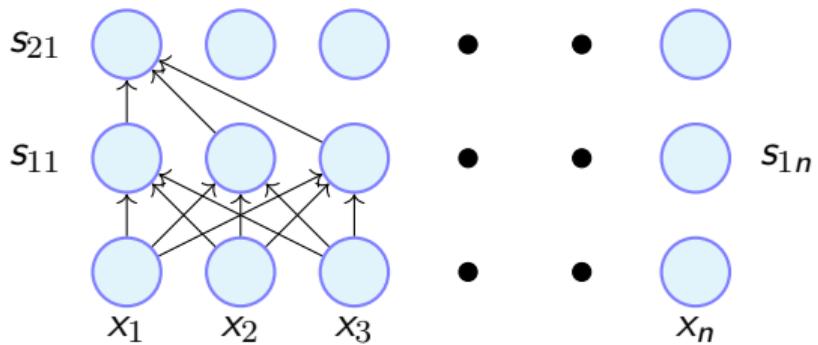


- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at



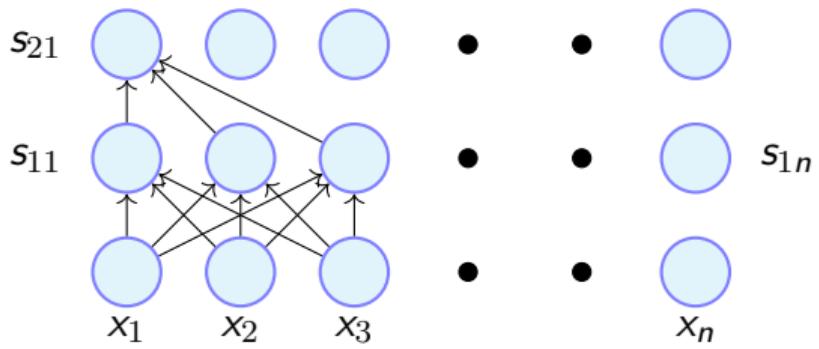
- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

$$Var(s_{21}) = \sum_{i=1}^n Var(s_{1i}) Var(w_{2i})$$



- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

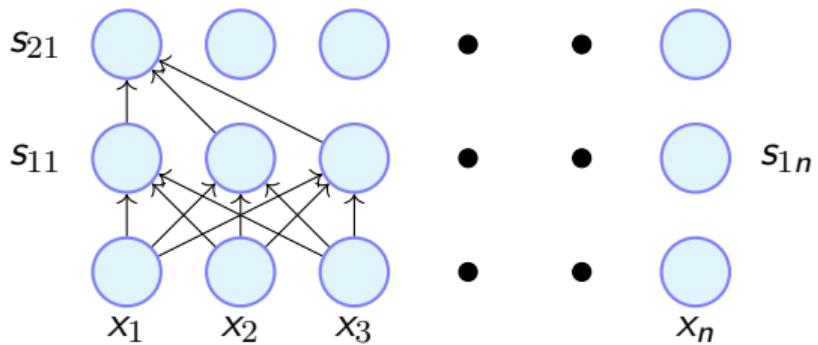
$$\begin{aligned}Var(s_{21}) &= \sum_{i=1}^n Var(s_{1i}) Var(w_{2i}) \\&= n Var(s_{1i}) Var(w_2)\end{aligned}$$



- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

$$\begin{aligned}Var(s_{21}) &= \sum_{i=1}^n Var(s_{1i}) Var(w_{2i}) \\&= n Var(s_{1i}) Var(w_2)\end{aligned}$$

$$Var(S_{i1}) = n Var(w_1) Var(x)$$



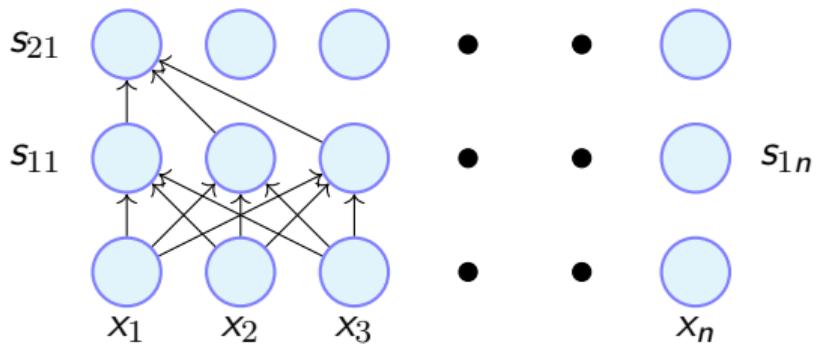
$$Var(S_{i1}) = nVar(w_1)Var(x)$$

- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

$$Var(s_{21}) = \sum_{i=1}^n Var(s_{1i}) Var(w_{2i})$$

$$= nVar(s_{1i}) Var(w_2)$$

$$Var(s_{21}) \propto [nVar(w_2)][nVar(w_1)]Var(x)$$



- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

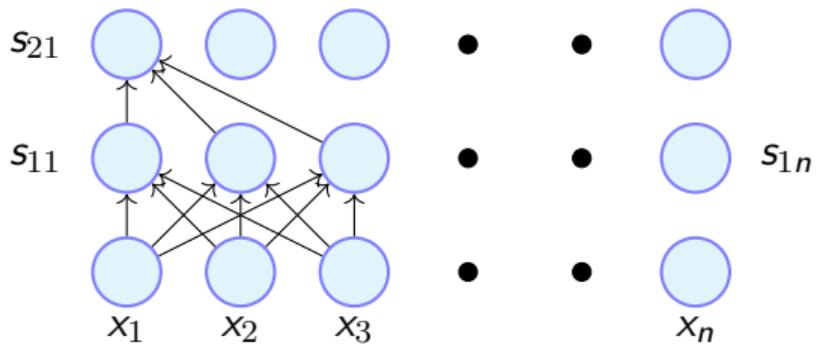
$$Var(s_{21}) = \sum_{i=1}^n Var(s_{1i}) Var(w_{2i})$$

$$= nVar(s_{1i}) Var(w_2)$$

$$Var(S_{i1}) = nVar(w_1) Var(x)$$

$$Var(s_{21}) \propto [nVar(w_2)][nVar(w_1)] Var(x)$$

$$\propto [nVar(w)]^2 Var(x)$$



- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

$$Var(s_{21}) = \sum_{i=1}^n Var(s_{1i}) Var(w_{2i})$$

$$= nVar(s_{1i}) Var(w_2)$$

$$Var(S_{i1}) = nVar(w_1) Var(x)$$

$$Var(s_{21}) \propto [nVar(w_2)][nVar(w_1)] Var(x)$$

$$\propto [nVar(w)]^2 Var(x)$$

Assuming weights across all layers



- In general,



- In general,

$$Var(s_{ki}) = [nVar(w)]^k Var(x)$$



- In general,

$$Var(s_{ki}) = [nVar(w)]^k Var(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

$$nVar(w) = 1$$



- In general,

$$\text{Var}(s_{ki}) = [n \text{Var}(w)]^k \text{Var}(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

$$n \text{Var}(w) = 1$$

- If we draw the weights from a unit Gaussian and scale them by  $\frac{1}{\sqrt{n}}$  then, we have :



- In general,

$$Var(s_{ki}) = [nVar(w)]^k Var(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

$$nVar(w) = 1$$

- If we draw the weights from a unit Gaussian and scale them by  $\frac{1}{\sqrt{n}}$  then, we have :

$$nVar(w) = nVar\left(\frac{z}{\sqrt{n}}\right)$$



- In general,

$$\text{Var}(s_{ki}) = [n \text{Var}(w)]^k \text{Var}(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

$$\boxed{\text{Var}(az) = a^2(\text{Var}(z))}$$

$$n \text{Var}(w) = 1$$

- If we draw the weights from a unit Gaussian and scale them by  $\frac{1}{\sqrt{n}}$  then, we have :

$$\begin{aligned} n \text{Var}(w) &= n \text{Var}\left(\frac{z}{\sqrt{n}}\right) \\ &= n * \frac{1}{n} \text{Var}(z) = 1 \end{aligned}$$



- In general,

$$\text{Var}(s_{ki}) = [n \text{Var}(w)]^k \text{Var}(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

$$\boxed{\text{Var}(az) = a^2(\text{Var}(z))}$$

$$n \text{Var}(w) = 1$$

- If we draw the weights from a unit Gaussian and scale them by  $\frac{1}{\sqrt{n}}$  then, we have :

$$n \text{Var}(w) = n \text{Var}\left(\frac{z}{\sqrt{n}}\right)$$

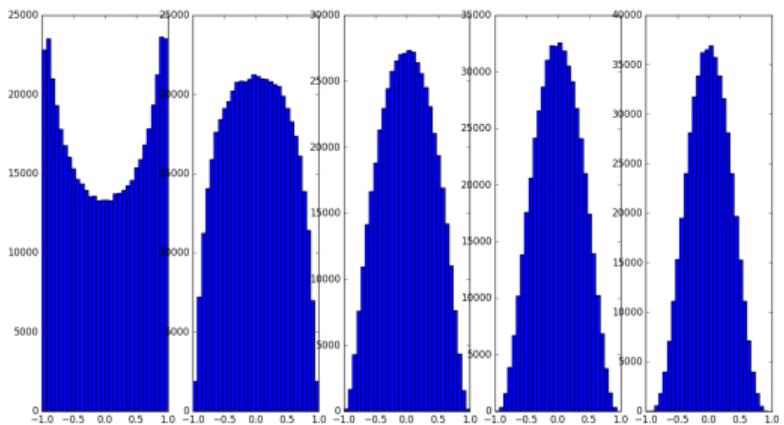
$$= n * \frac{1}{n} \text{Var}(z) = 1 \leftarrow (\text{Unit Gaussian})$$

```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in)
```

- Let's see what happens if we use this initialization

```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in)
```

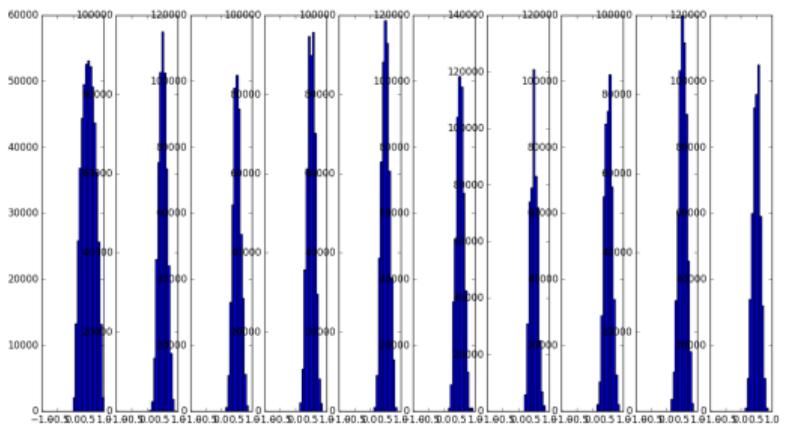
- Let's see what happens if we use this initialization



tanh activation

```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in)
```

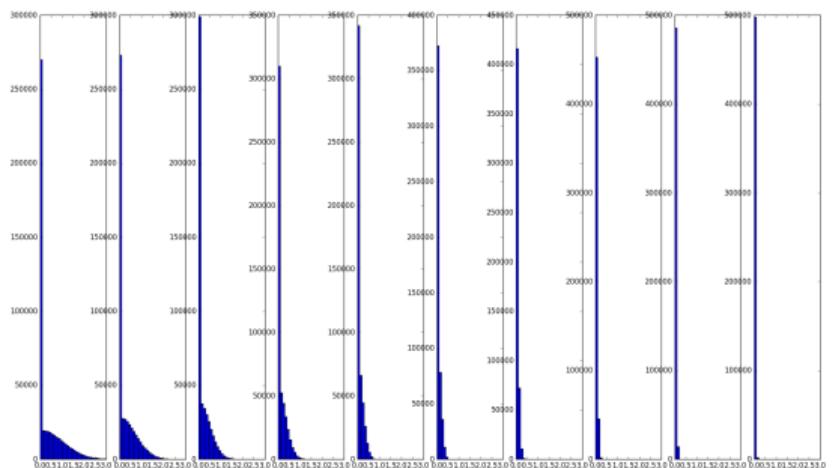
- Let's see what happens if we use this initialization

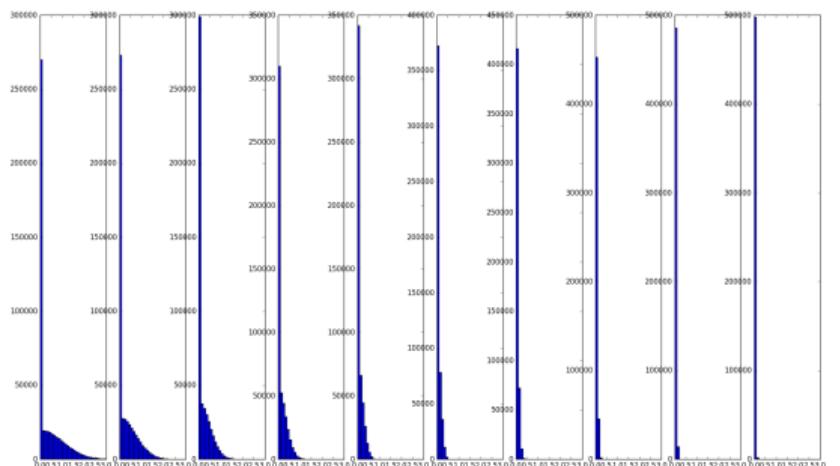


sigmoid activations

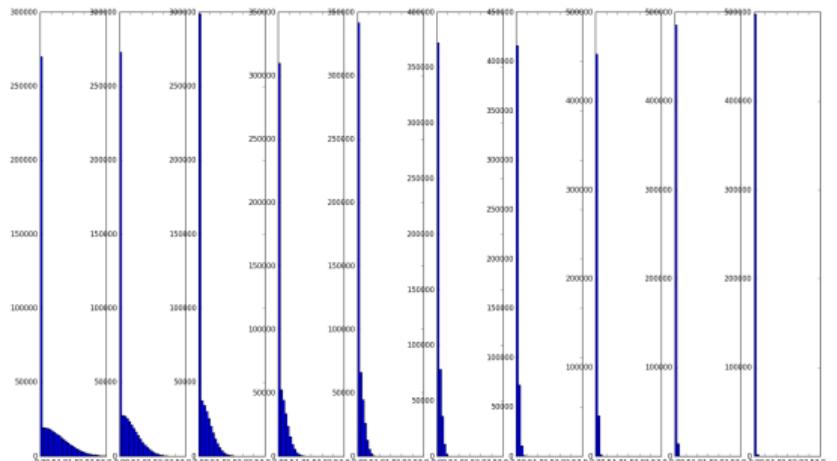


- However this does not work for ReLU neurons

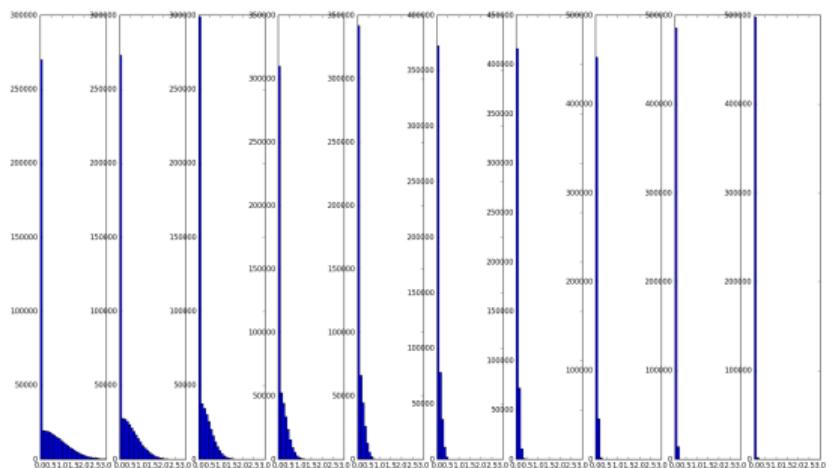




- However this does not work for ReLU neurons
- Why ?



- However this does not work for ReLU neurons
- Why ?
- Intuition: *He et.al.* argue that a factor of 2 is needed when dealing with ReLU Neurons



- However this does not work for ReLU neurons
- Why ?
- Intuition: *He et.al.* argue that a factor of 2 is needed when dealing with ReLU Neurons
- Intuitively this happens because the range of ReLU neurons is restricted only to the positive half of the space



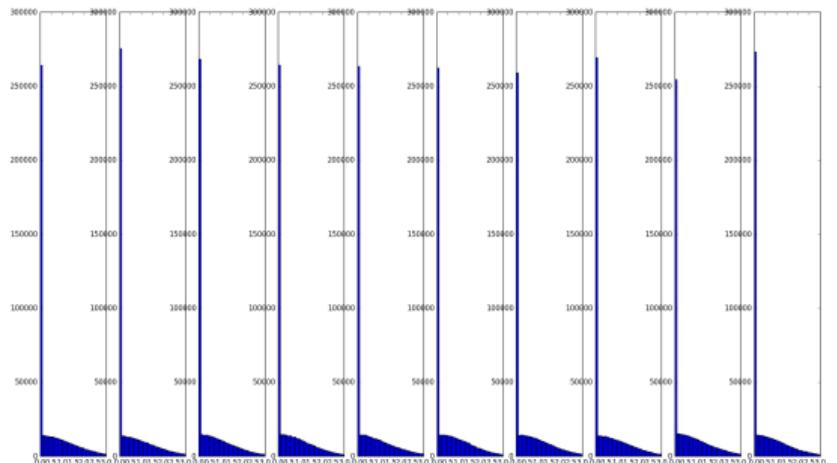
```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in/2)
```

- Indeed when we account for this factor of 2 we see better performance



```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in/2)
```

- Indeed when we account for this factor of 2 we see better performance





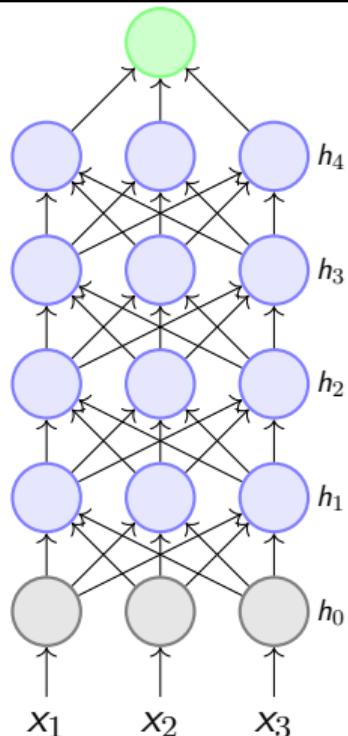
# 批归一化 (Batch Normalization)



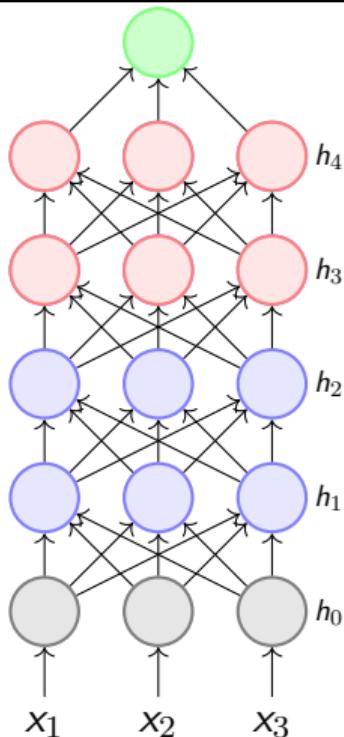
批归一化可以让模型训练对参数初始化不太敏感



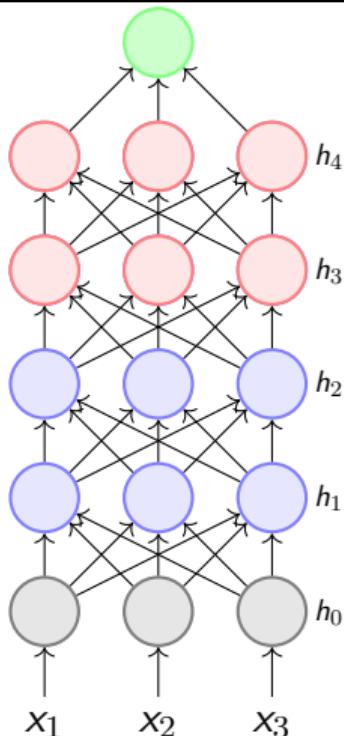
- 考虑一个深度网络



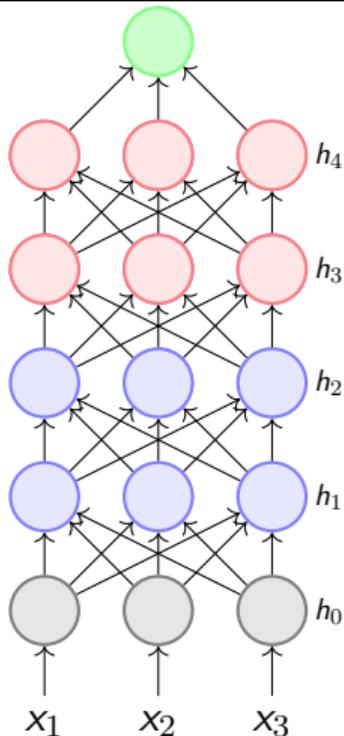
▪ 考虑一个深度网络



- 考虑一个深度网络
- 如果要学习这两层之间的权重



- 考虑一个深度网络
- 如果要学习这两层之间的权重
- 通常使用 mini-batch 梯度下降算法



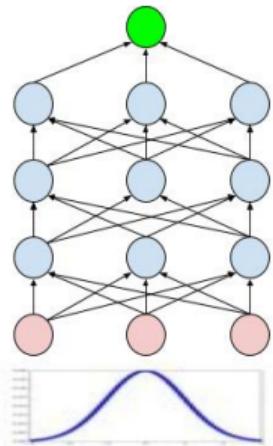
- 考虑一个深度网络
- 如果要学习这两层之间的权重
- 通常使用 mini-batch 梯度下降算法
- 如果输入发生变化，导致  $h_3$  相应变化，学习算法会不会有问题？



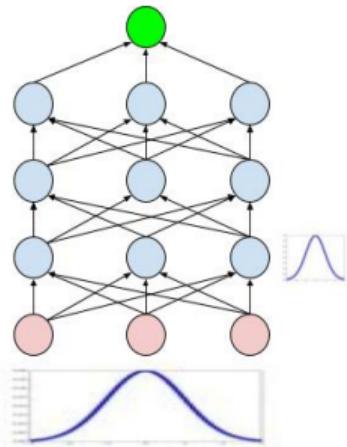
- 考虑一个深度网络
- 如果要学习这两层之间的权重
- 通常使用 mini-batch 梯度下降算法
- 如果输入发生变化，导致  $h_3$  相应变化，学习算法会不会有问题？



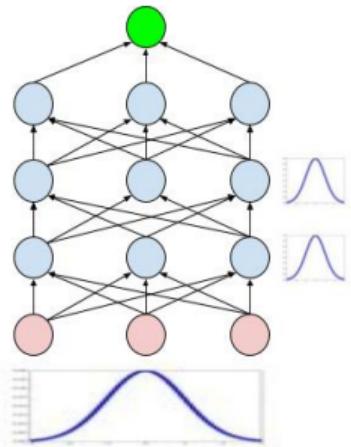
- 考虑一个深度网络
- 如果要学习这两层之间的权重
- 通常使用 mini-batch 梯度下降算法
- 如果输入发生变化，导致  $h_3$  相应变化，学习算法会不会有问题？



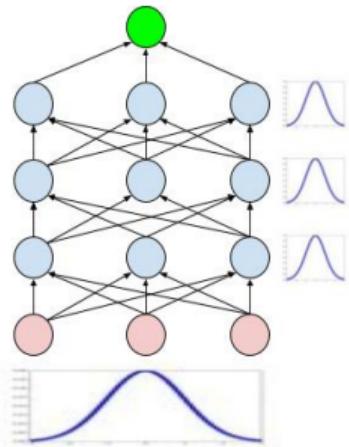
- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒



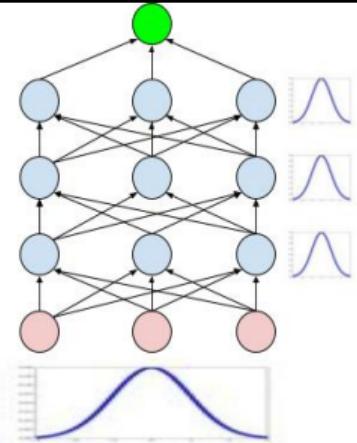
- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒



- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒

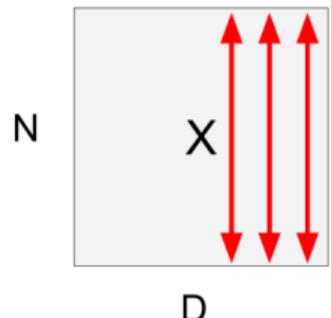


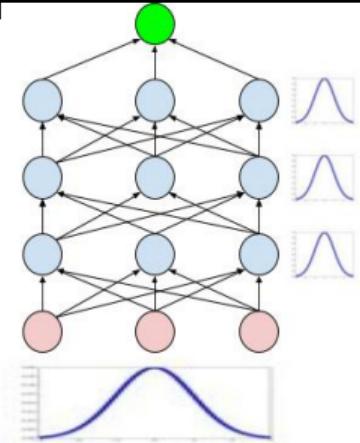
- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒



- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒
- 因此，可以对每一层的预激活进行如下“标准化”操作

$$\hat{s}_{ik} = \frac{s_{ik} - E[s_{ik}]}{\sqrt{\text{var}(s_{ik})}}$$

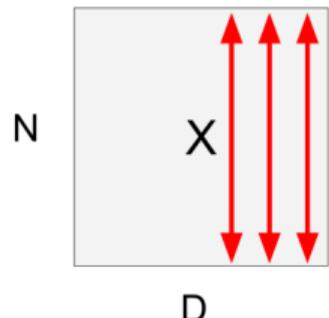


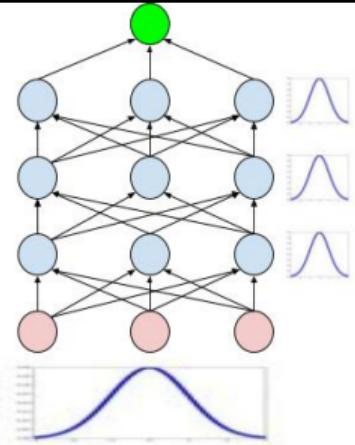


- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒
- 因此，可以对每一层的预激活进行如下“标准化”操作

$$\hat{s}_{ik} = \frac{s_{ik} - E[s_{ik}]}{\sqrt{\text{var}(s_{ik})}}$$

- 如何计算  $E[s_{ik}]$  和  $\text{Var}[s_{ik}]$ ?

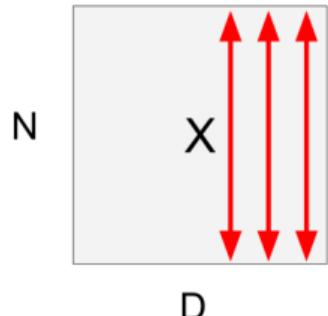


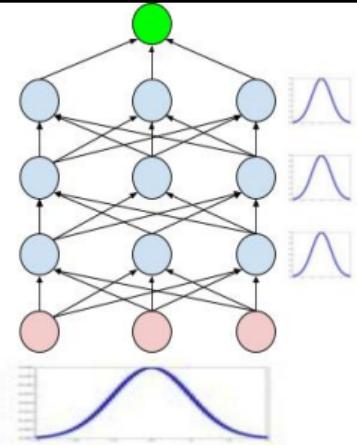


- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒
- 因此，可以对每一层的预激活进行如下“标准化”操作

$$\hat{s}_{ik} = \frac{s_{ik} - E[s_{ik}]}{\sqrt{\text{var}(s_{ik})}}$$

- 如何计算  $E[s_{ik}]$  和  $\text{Var}[s_{ik}]$ ?
- 使用小批量的数据来计算它们

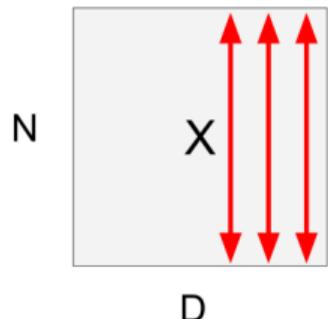




- 如果每一层的预激活服从单位高斯分布，对输入的变化会更鲁棒
- 因此，可以对每一层的预激活进行如下“标准化”操作

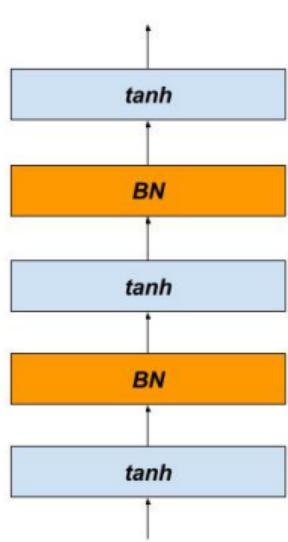
$$\hat{s}_{ik} = \frac{s_{ik} - E[s_{ik}]}{\sqrt{\text{var}(s_{ik})}}$$

- 如何计算  $E[s_{ik}]$  和  $\text{Var}[s_{ik}]$ ?
- 使用小批量的数据来计算它们
- 因此，确保不同层的输入的分布不会随着 batch 的不同而改变

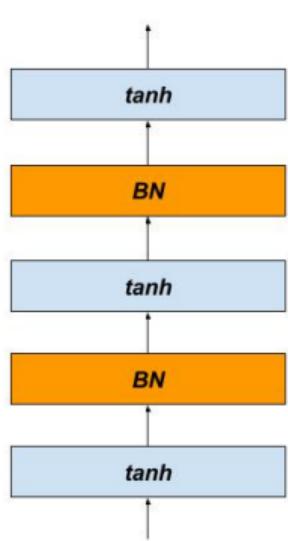




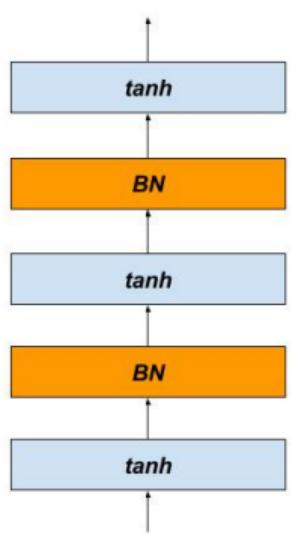
- 这样的操作称为 Batch Normalization



- 这样的操作称为 Batch Normalization
- 可微吗?



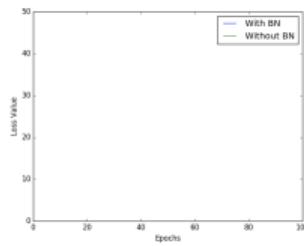
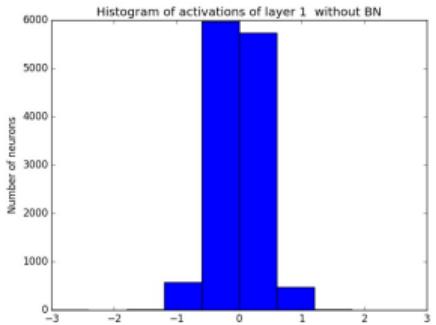
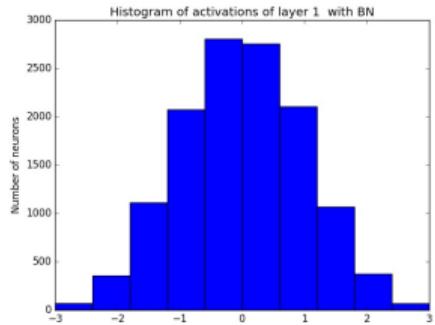
- 这样的操作称为 Batch Normalization
- 可微吗?
- 像  $tanh$  一样, Batch Normalization 也是可微的

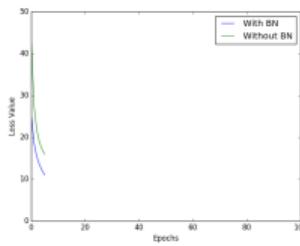
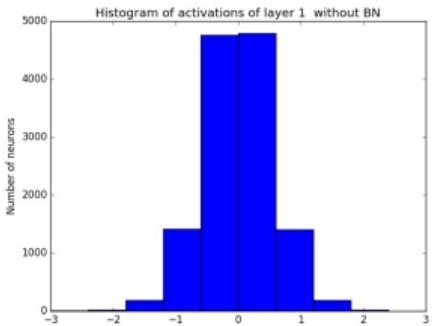
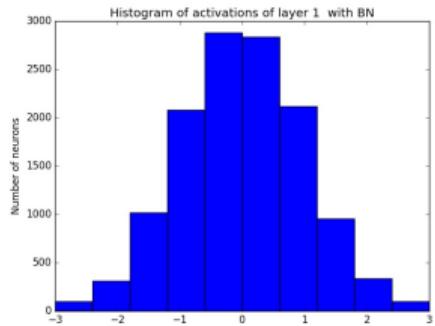


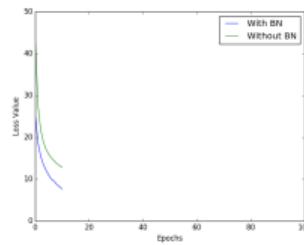
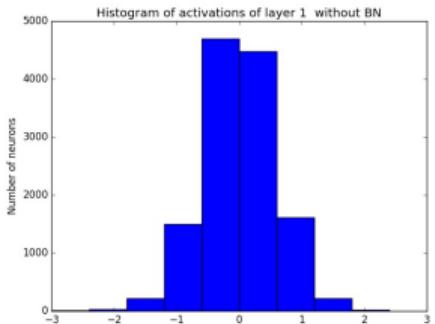
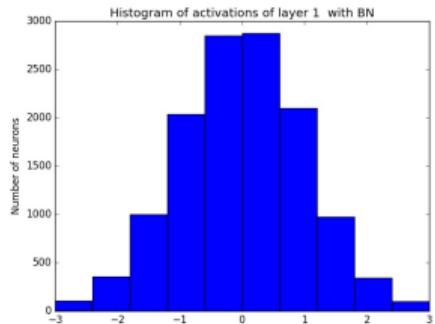
- 这样的操作称为 Batch Normalization
- 可微吗?
- 像  $tanh$  一样, Batch Normalization 也是可微的
- 因此, 梯度可以反向传播

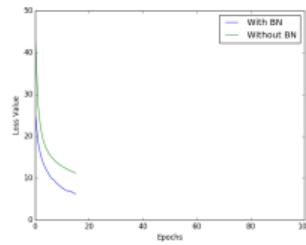
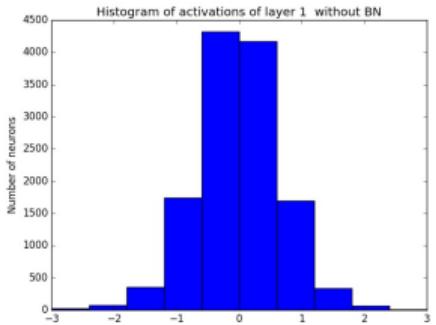
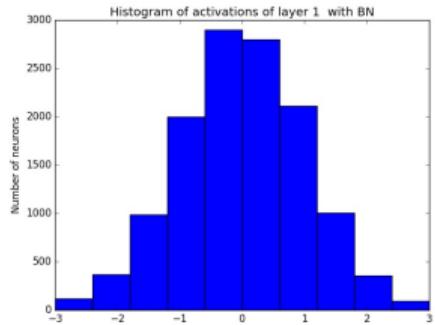


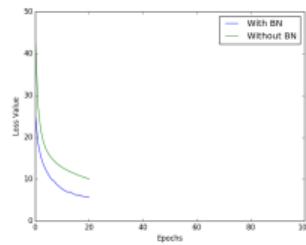
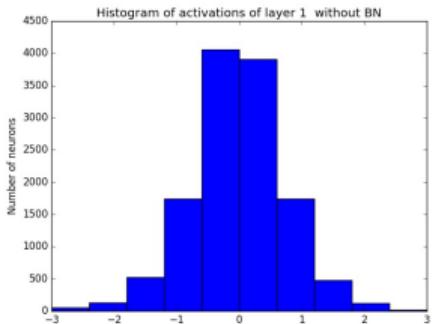
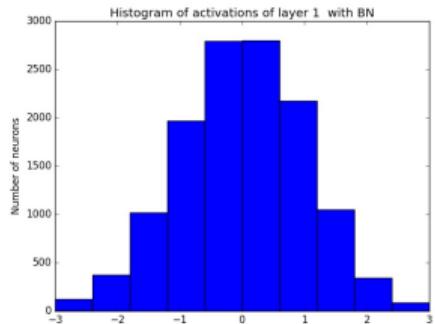
在 MNIST 数据集上，设计一个 2 层的网络，比较使用和不使用 batch normalization 的差异

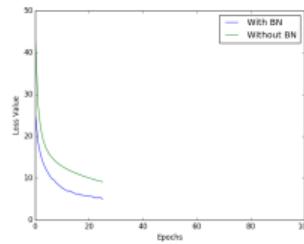
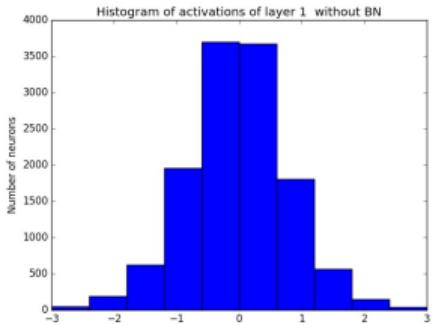
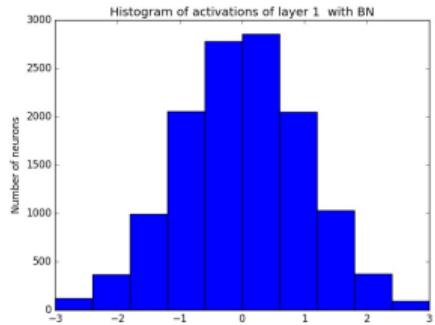


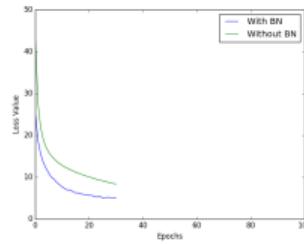
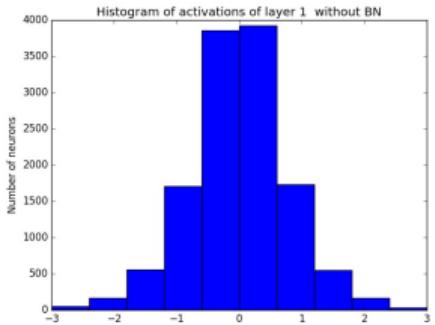
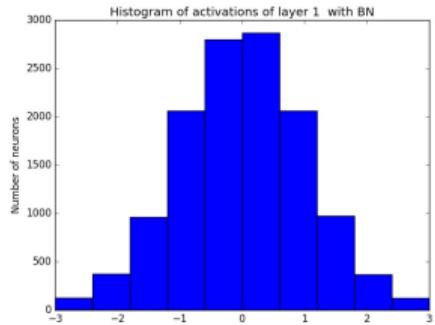


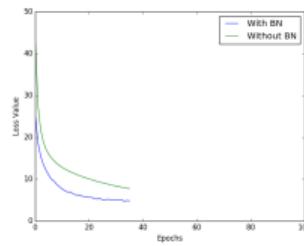
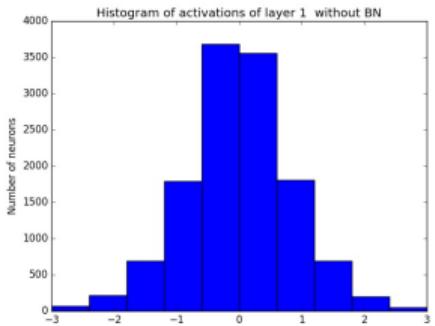
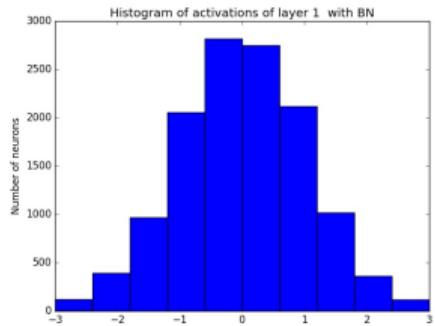


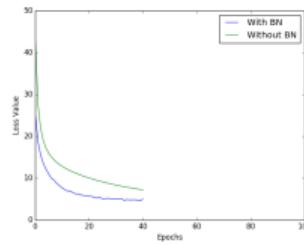
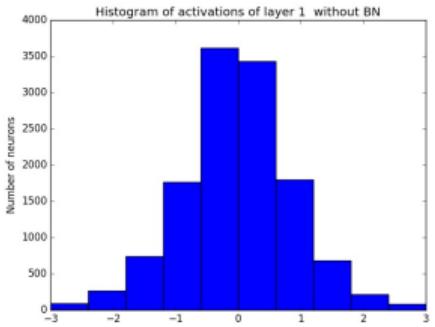
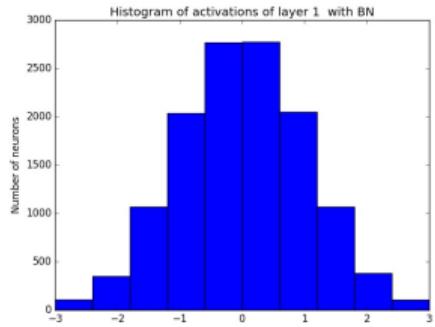


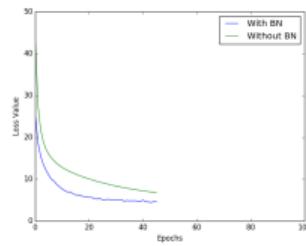
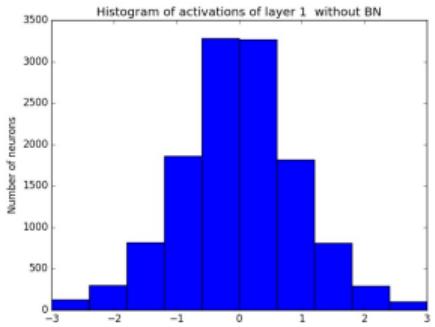
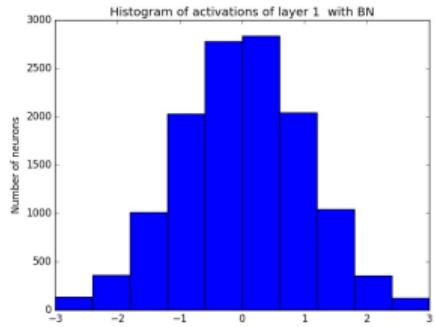


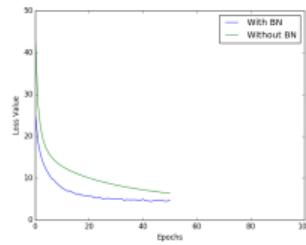
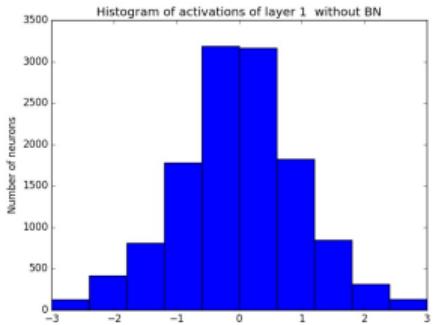
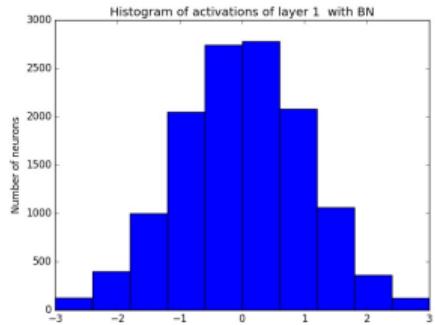


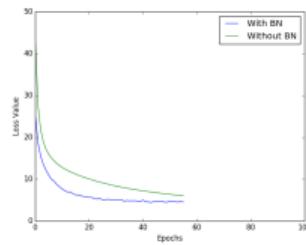
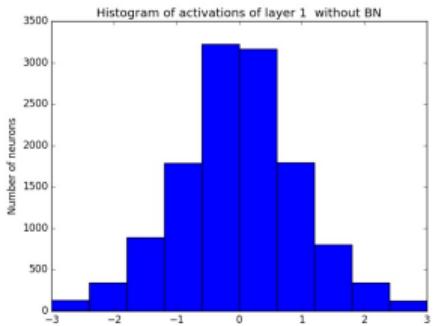
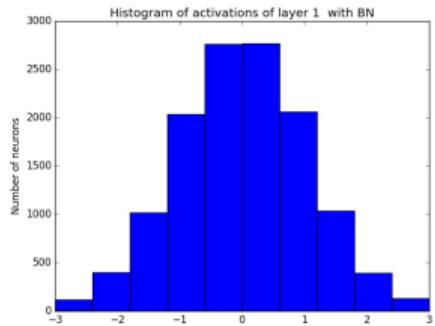


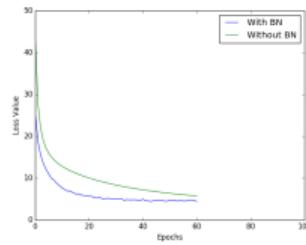
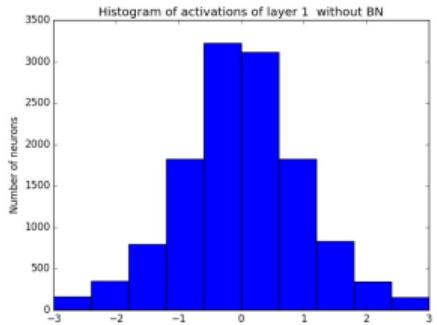
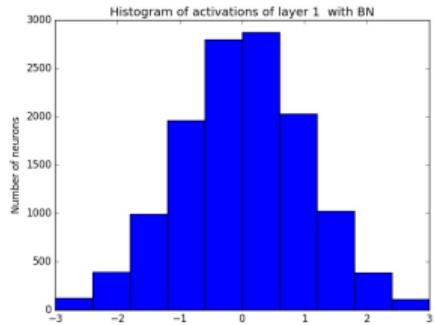


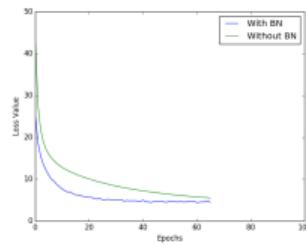
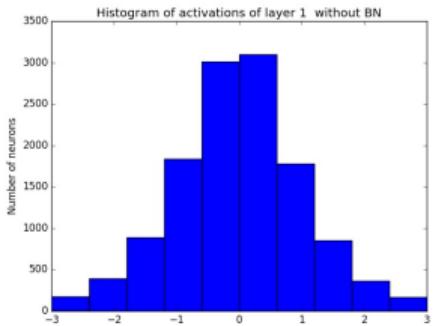
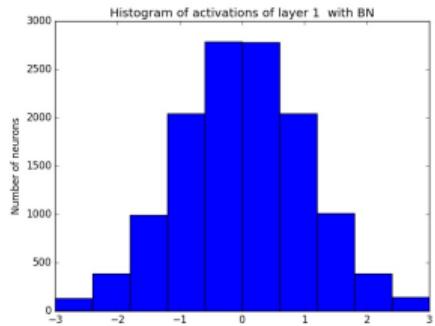


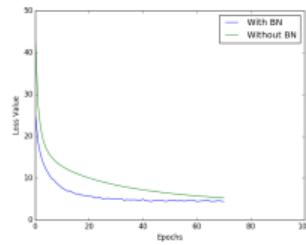
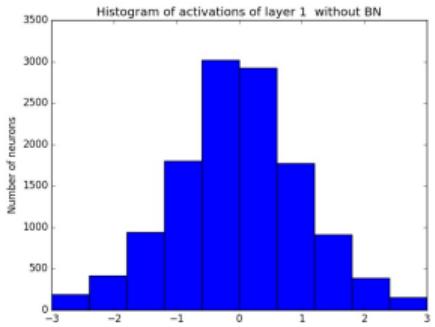
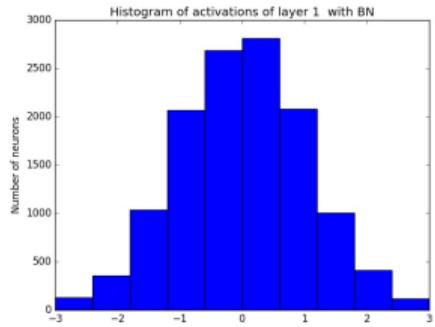


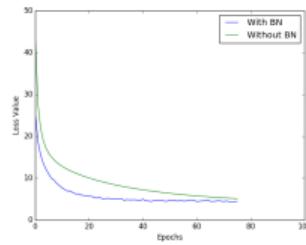
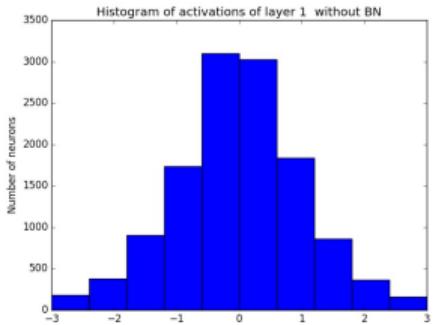
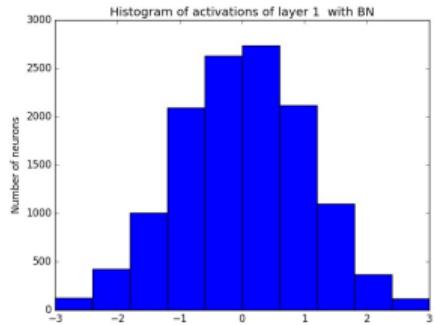


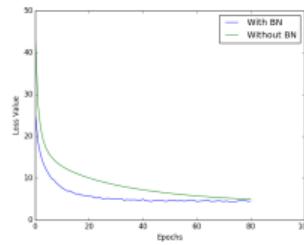
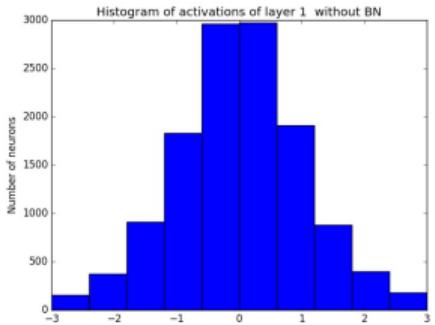
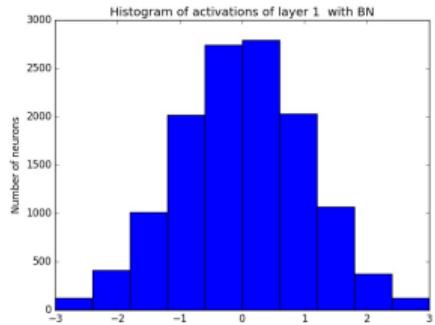


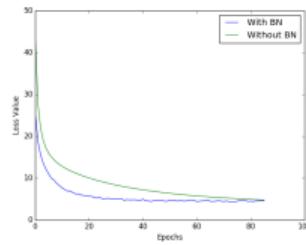
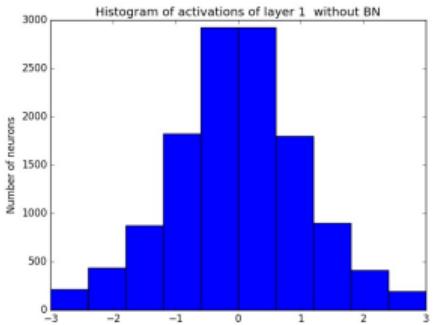
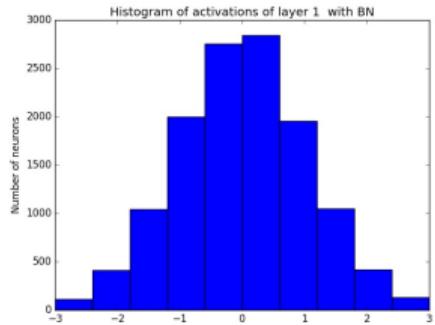


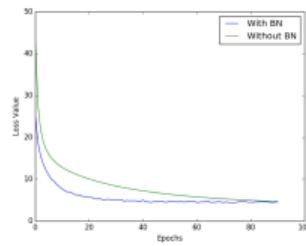
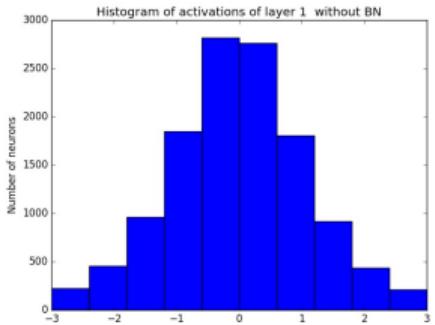
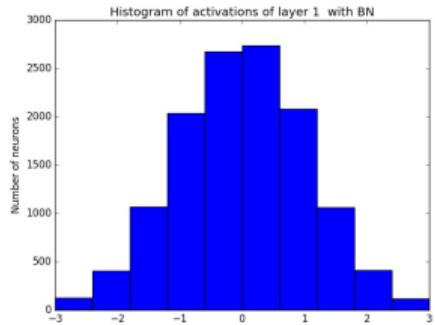


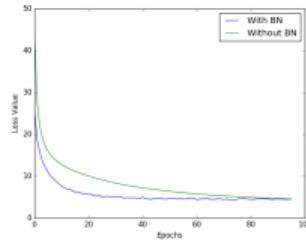
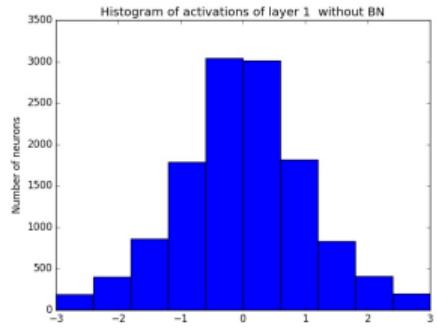
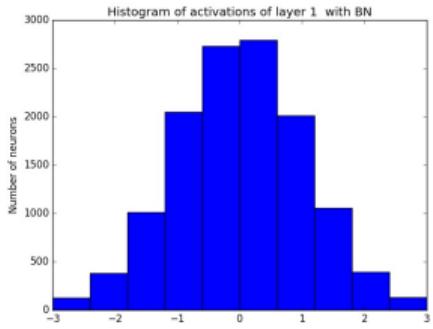














# Dataset augmentation



label = 2



label = 2

[训练样本]



label = 2

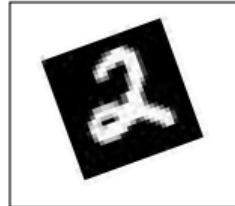
[训练样本]





label = 2

[训练样本]



rotated by 20°

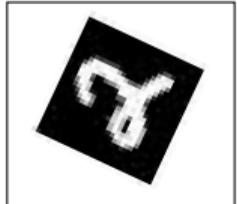


label = 2

[训练样本]



rotated by  $20^\circ$



rotated by  $65^\circ$



label = 2

[训练样本]



rotated by  $20^\circ$



rotated by  $65^\circ$

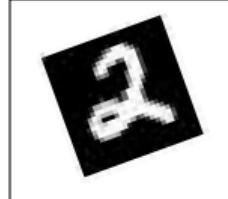


shifted vertically



label = 2

[训练样本]



rotated by  $20^\circ$



rotated by  $65^\circ$



shifted vertically

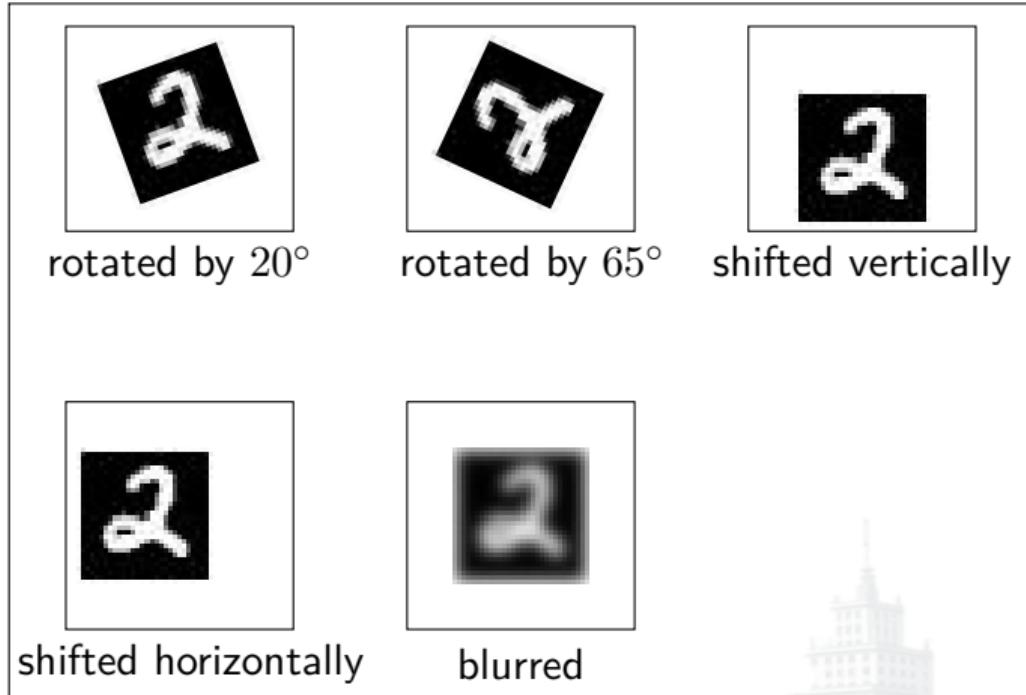


shifted horizontally



label = 2

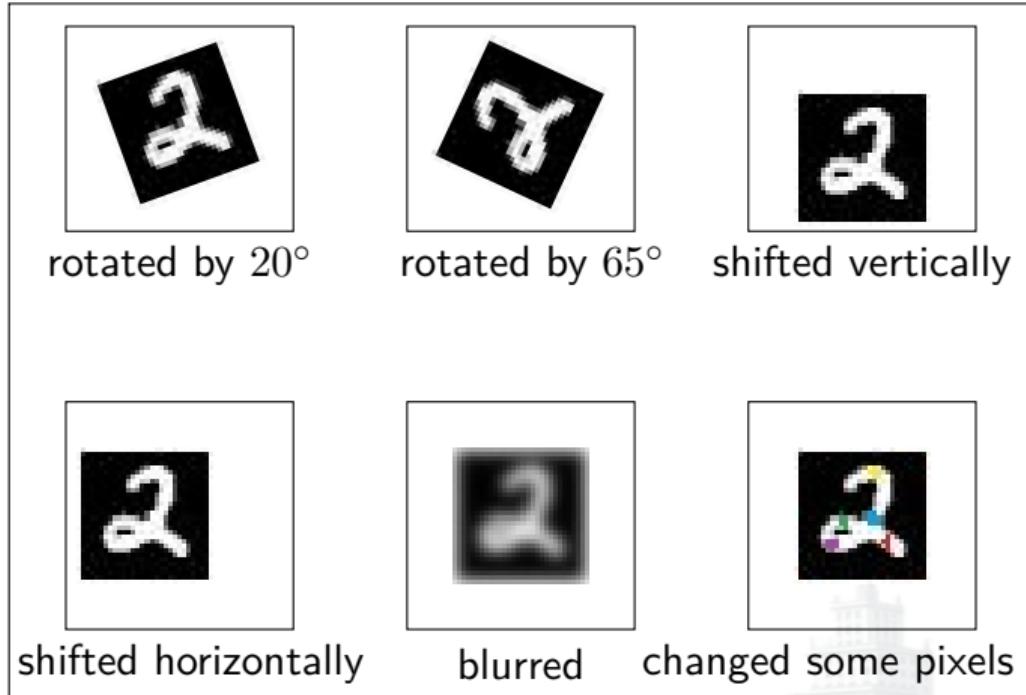
[训练样本]





label = 2

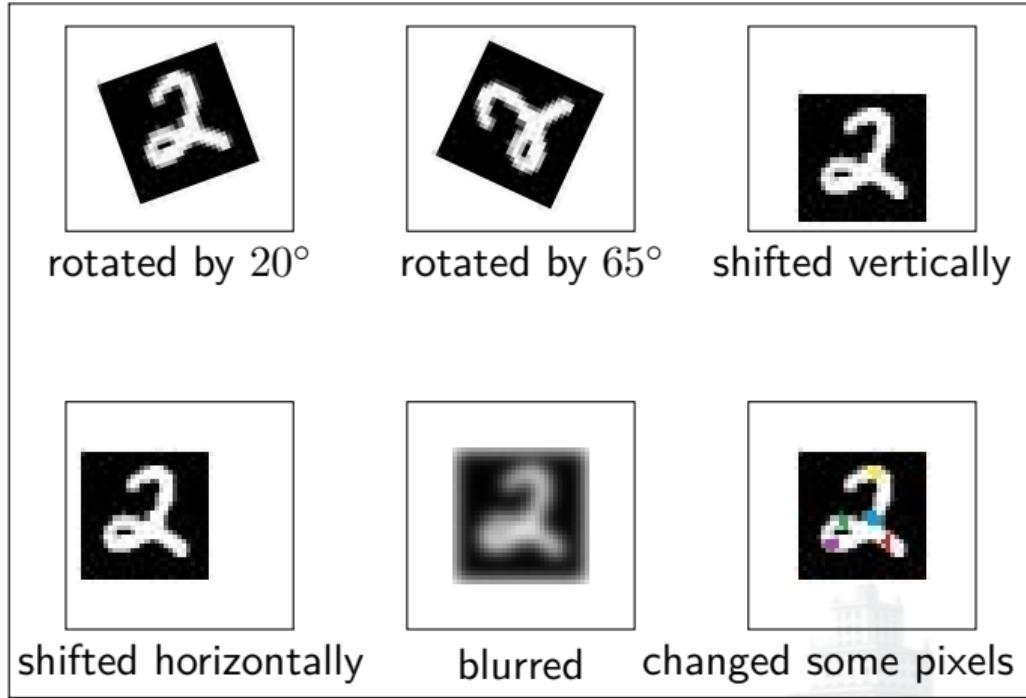
[训练样本]





label = 2

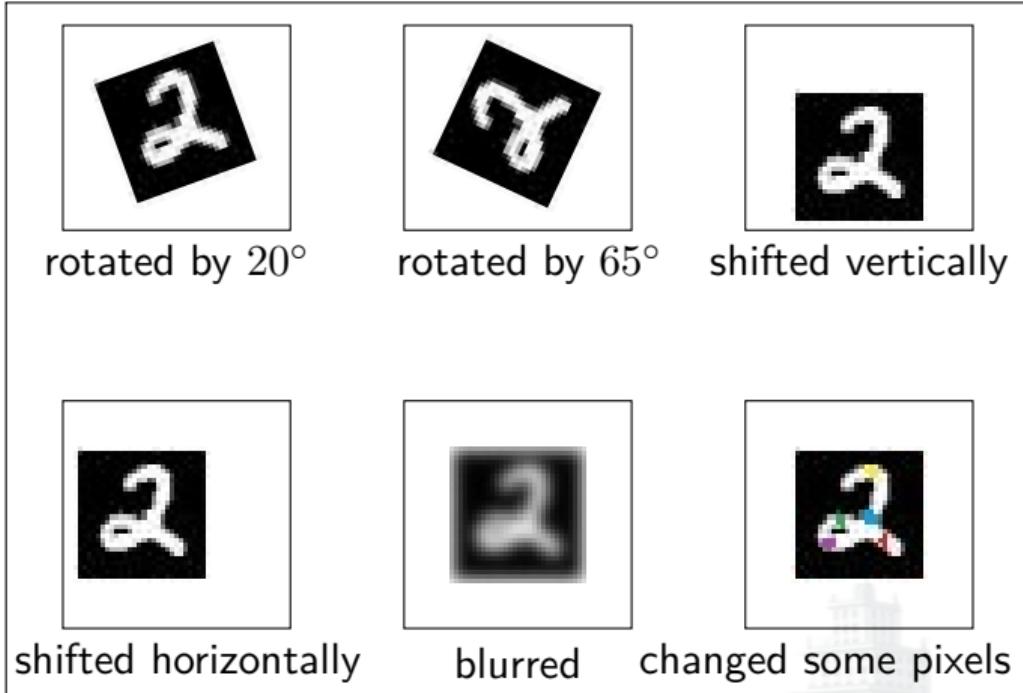
[训练样本]





label = 2

[训练样本]



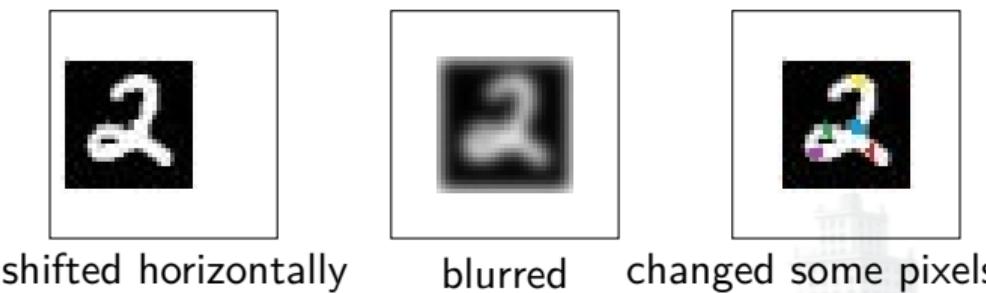
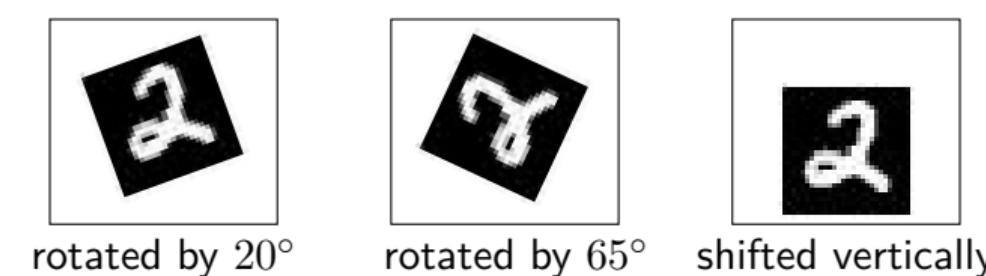
label = 2

[任  
务相关的先验知识]



label = 2

[训练样本]  
对图像进行特定的变换，  
不会改变它的 label.



label = 2

务相关的先验知识]



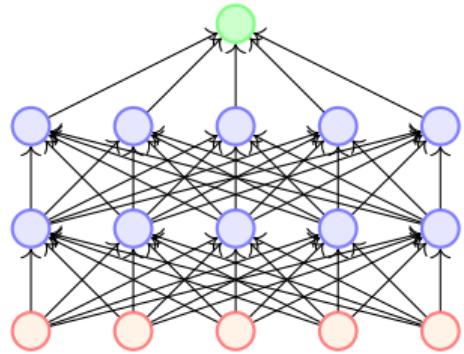
- 通常, More data = better learning



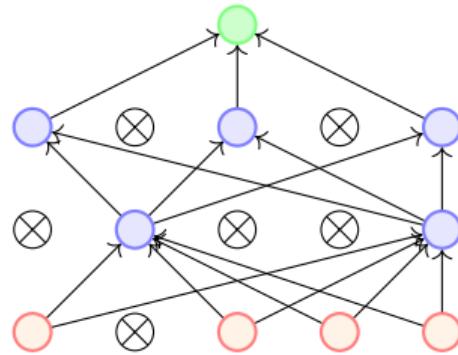
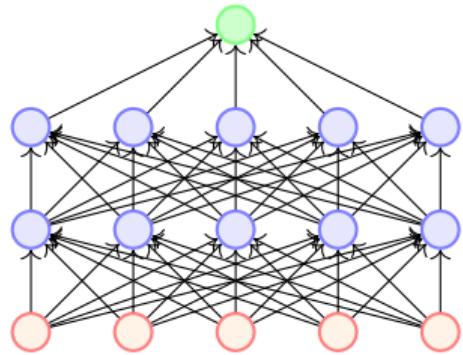
- 通常, More data = better learning
- 对于图像分类、物体识别、语音识别等任务, 很容易执行数据增广



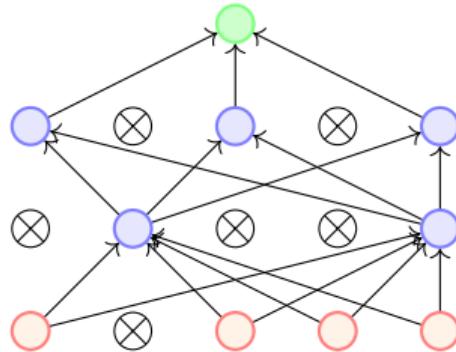
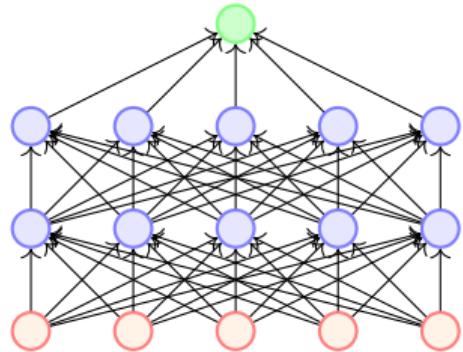
# Dropout



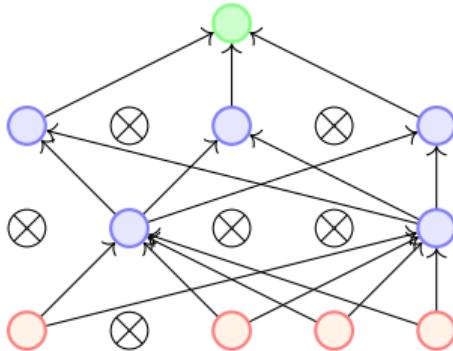
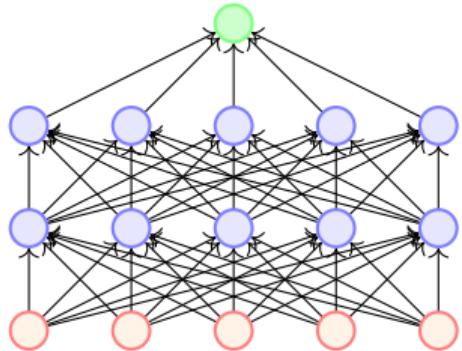
- Dropout refers to dropping out units

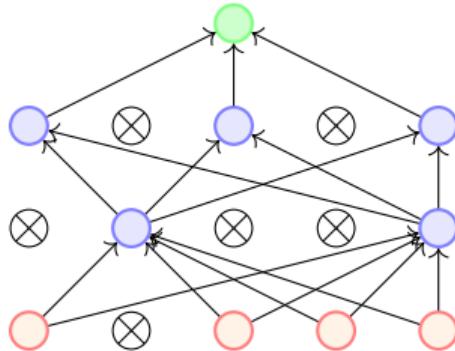
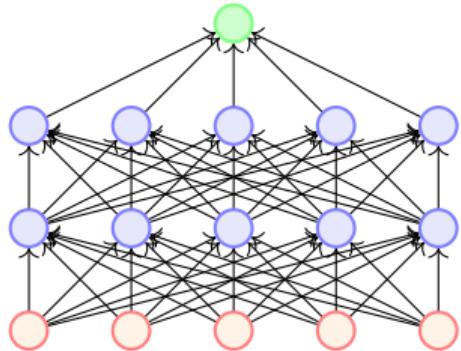


- Dropout refers to dropping out units
- 在神经网络中，移除一个结点和与之相连的边，会得到一个『瘦身』之后的网络

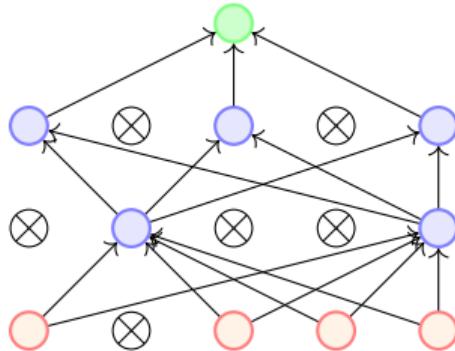
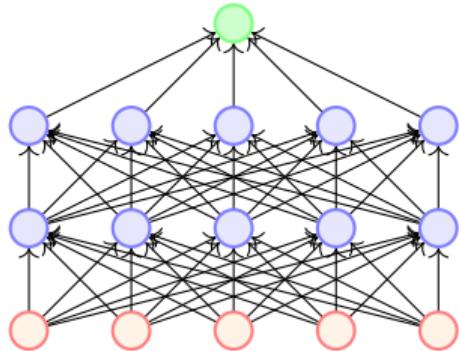


- Dropout refers to dropping out units
- 在神经网络中，移除一个结点和与之相连的边，会得到一个『瘦身』之后的网络
- 神经网络中的每一个结点，以一个固定概率移除（通常对于隐含结点  $p = 0.5$ ，对于可见结点  $p = 0.8$ ）

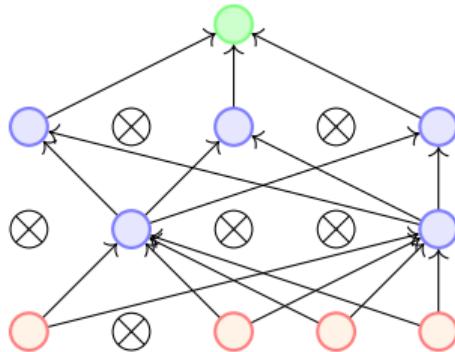
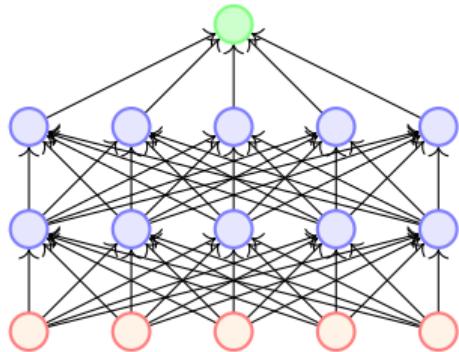




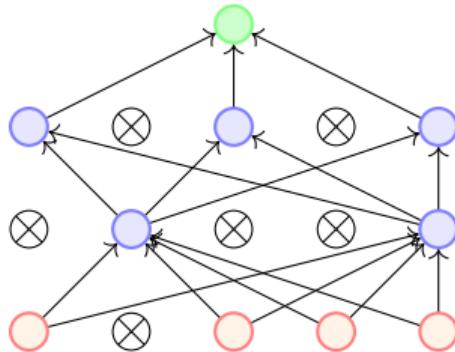
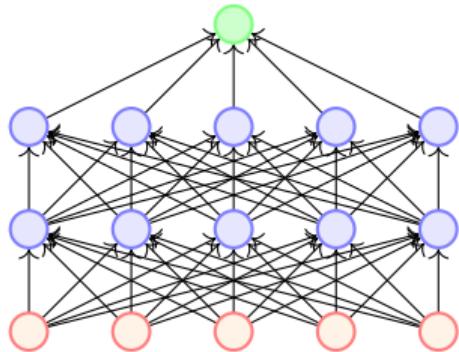
- 假定一个神经网络有  $n$  个结点



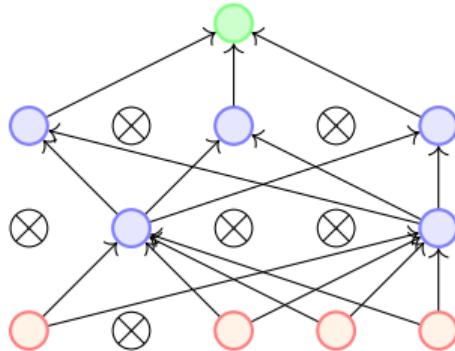
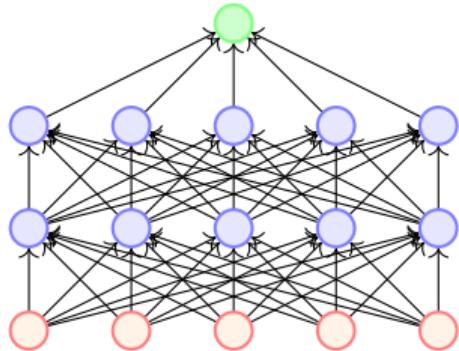
- 假定一个神经网络有  $n$  个结点
- 当使用 dropout，每一个结点都有可能被移除或保留



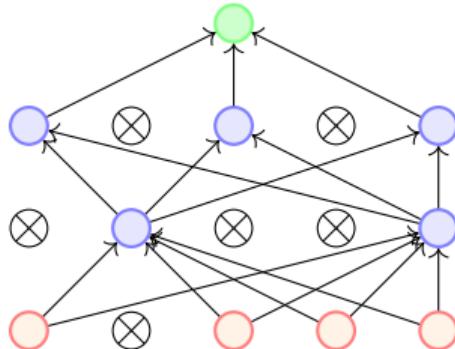
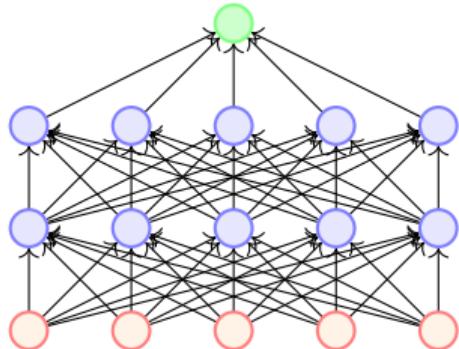
- 假定一个神经网络有  $n$  个结点
- 当使用 dropout，每一个结点都有可能被移除或保留
- 给定一个包含  $n$  个结点的神经网络，能够得到多少个『瘦身』的网络？



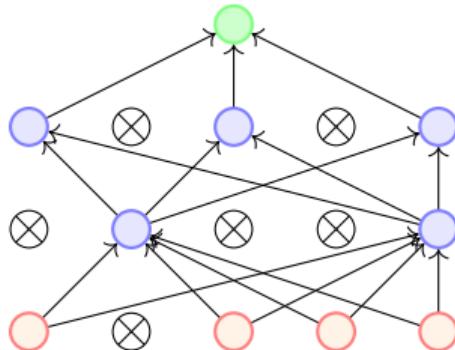
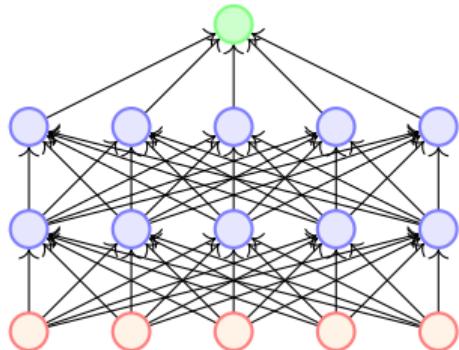
- 假定一个神经网络有  $n$  个结点
- 当使用 dropout，每一个结点都有可能被移除或保留
- 给定一个包含  $n$  个结点的神经网络，能够得到多少个『瘦身』的网络？ $2^n$



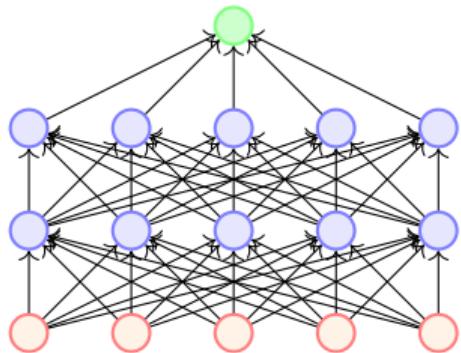
- 假定一个神经网络有  $n$  个结点
- 当使用 dropout，每一个结点都有可能被移除或保留
- 给定一个包含  $n$  个结点的神经网络，能够得到多少个『瘦身』的网络？ $2^n$
- 这么多『瘦身』的网络，不可能对它们都训练

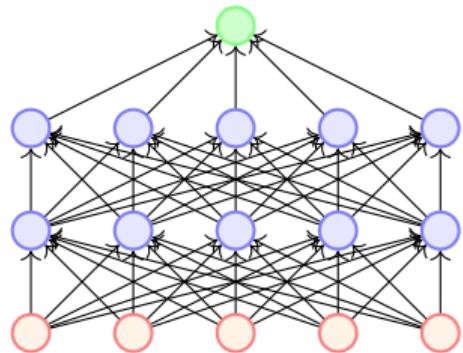


- 假定一个神经网络有  $n$  个结点
- 当使用 dropout，每一个结点都有可能被移除或保留
- 给定一个包含  $n$  个结点的神经网络，能够得到多少个『瘦身』的网络？ $2^n$
- 这么多『瘦身』的网络，不可能对它们都训练
- Trick: (1) 不同网络之间共享权重

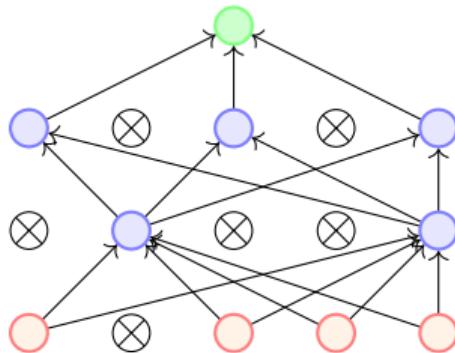
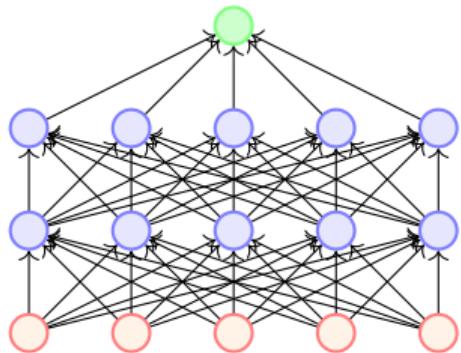


- 假定一个神经网络有  $n$  个结点
- 当使用 dropout，每一个结点都有可能被移除或保留
- 给定一个包含  $n$  个结点的神经网络，能够得到多少个『瘦身』的网络？ $2^n$
- 这么多『瘦身』的网络，不可能对它们都训练
- Trick: (1) 不同网络之间共享权重  
(2) 对于每一个训练实例（一个训练样本或一个 mini-batch），采样一个『瘦身』的网络

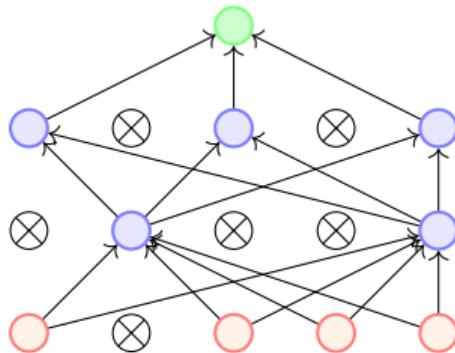
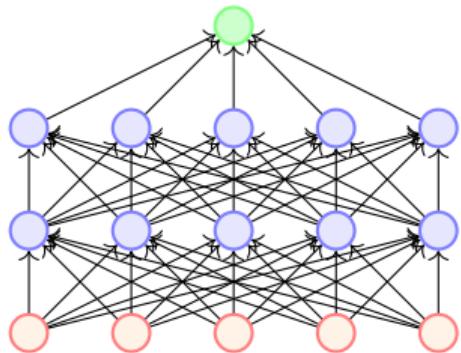




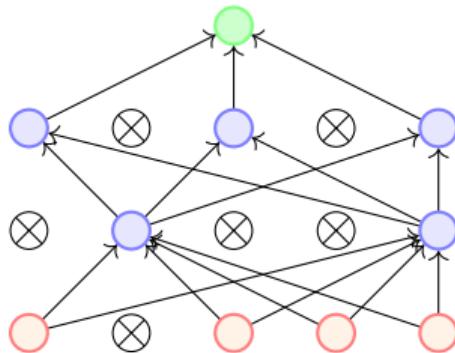
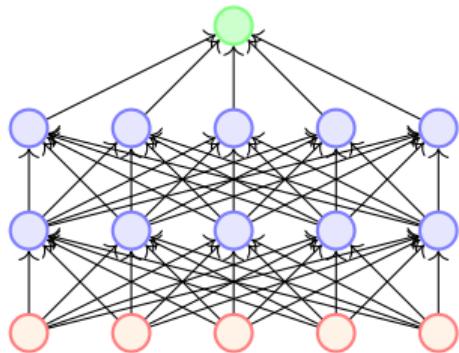
- 初始网络的所有参数（权重），开始训练



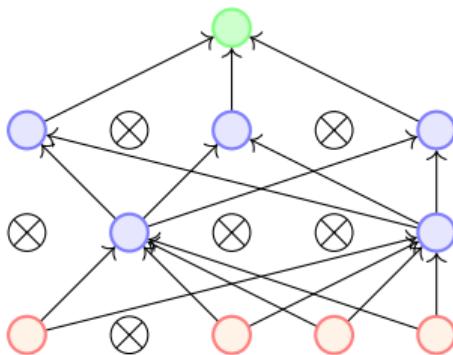
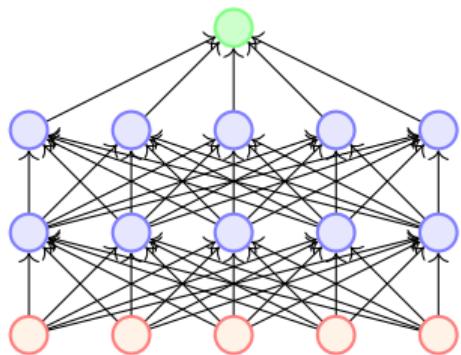
- 初始网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络



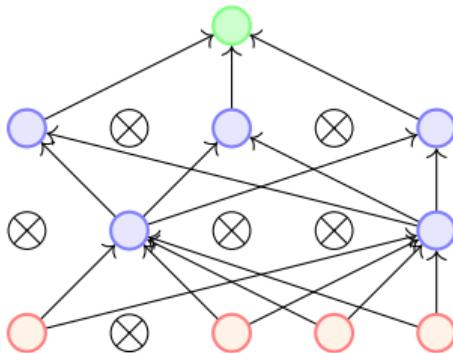
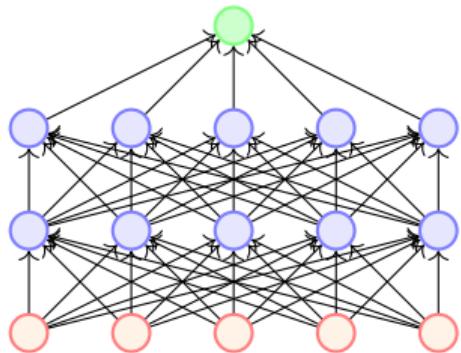
- 初始化网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络
- 计算损失函数



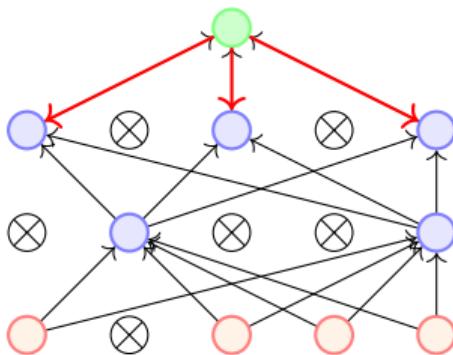
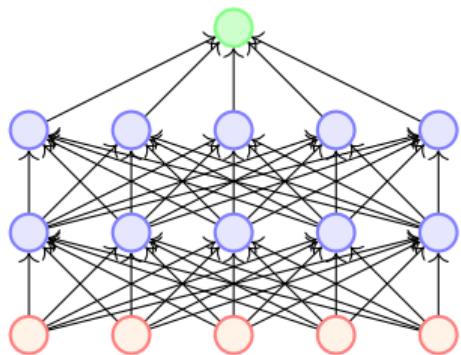
- 初始化网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络
- 计算损失函数
- 需要更新哪些参数？



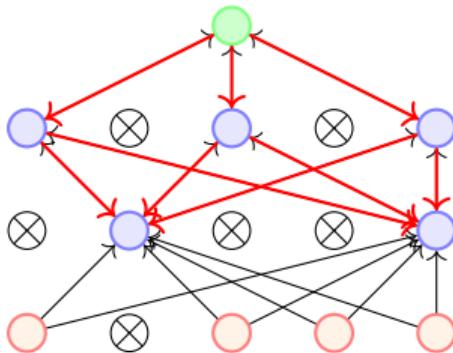
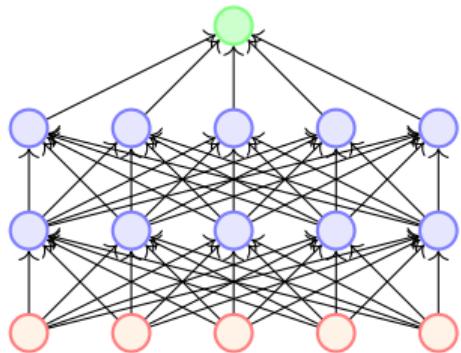
- 初始化网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络
- 计算损失函数
- 需要更新哪些参数？只更新那些『瘦身』网络中激活的参数，使用反向传播计算它们的梯度



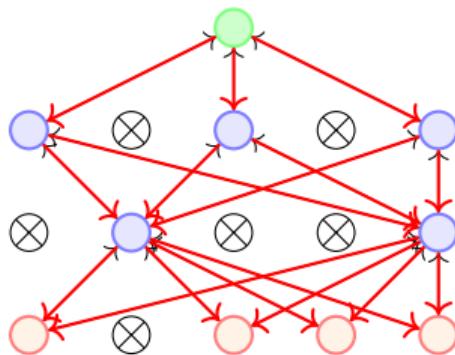
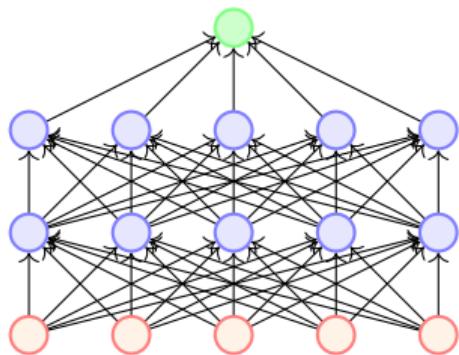
- 初始化网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络
- 计算损失函数
- 需要更新哪些参数？只更新那些『瘦身』网络中激活的参数，使用反向传播计算它们的梯度



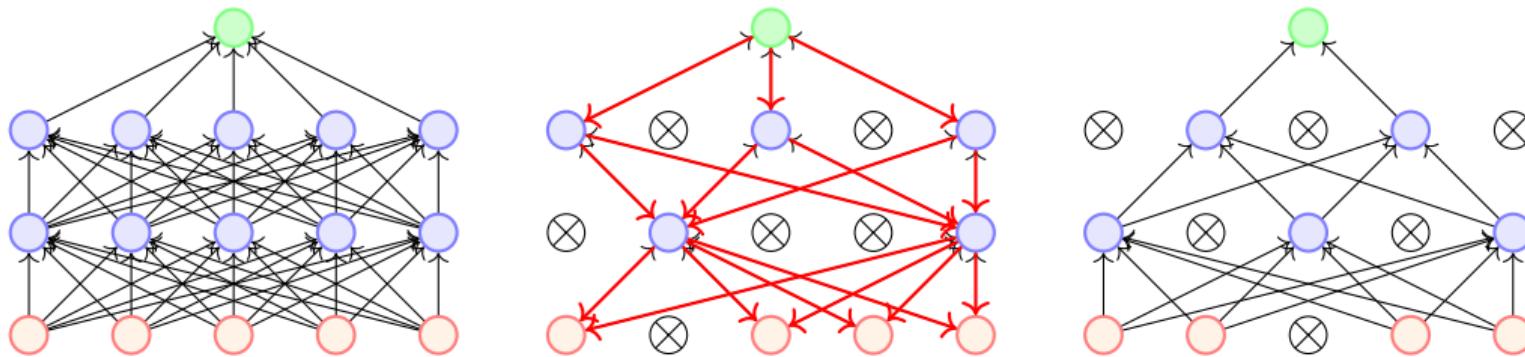
- 初始化网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络
- 计算损失函数
- 需要更新哪些参数？只更新那些『瘦身』网络中激活的参数，使用反向传播计算它们的梯度



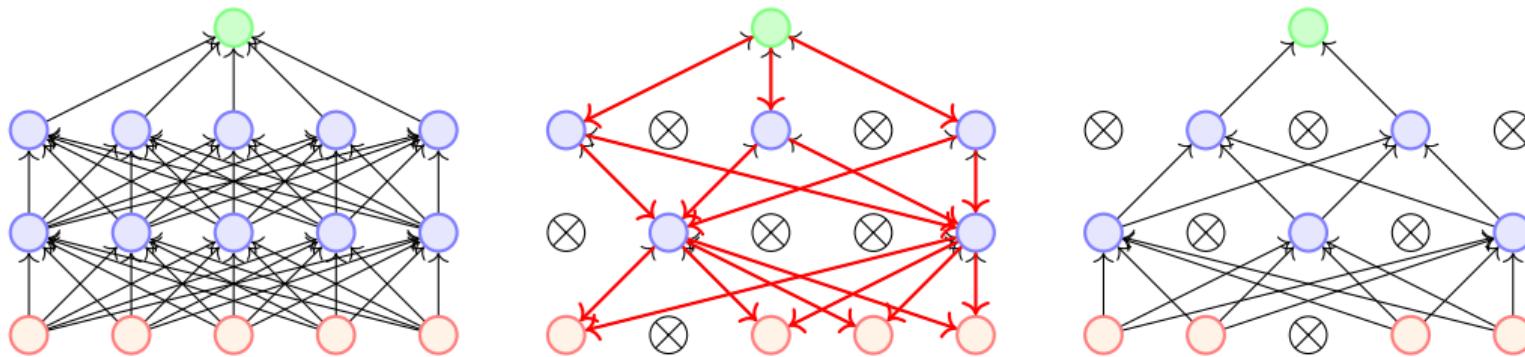
- 初始化网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络
- 计算损失函数
- 需要更新哪些参数？只更新那些『瘦身』网络中激活的参数，使用反向传播计算它们的梯度



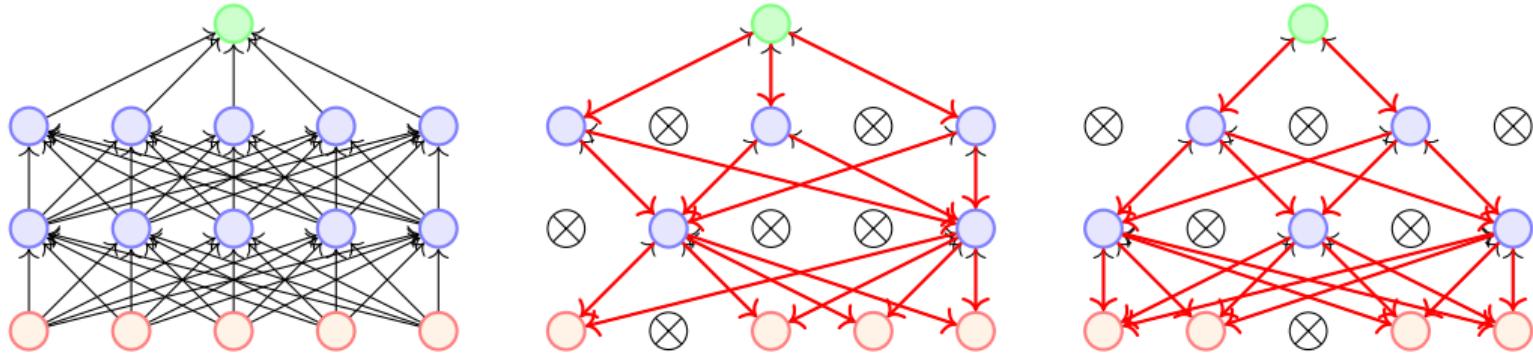
- 初始化网络的所有参数（权重），开始训练
- 对第一个训练实例（一个训练样本或一个 mini-batch），使用 Dropout 得到一个『瘦身』的网络
- 计算损失函数
- 需要更新哪些参数？只更新那些『瘦身』网络中激活的参数，使用反向传播计算它们的梯度



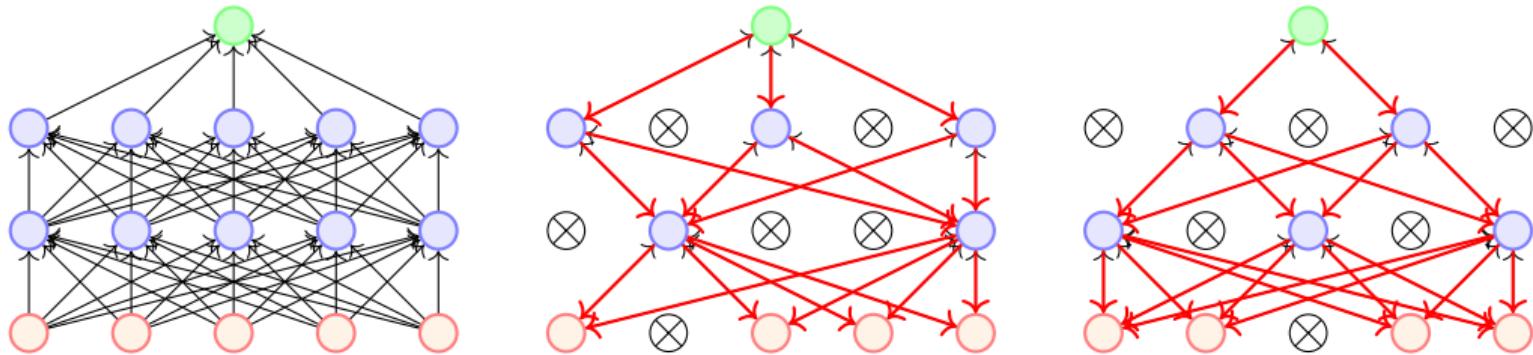
- 对第二个训练实例（一个训练样本或一个 mini-batch）(or mini-batch)，使用 Dropout 得到一个『瘦身』的网络



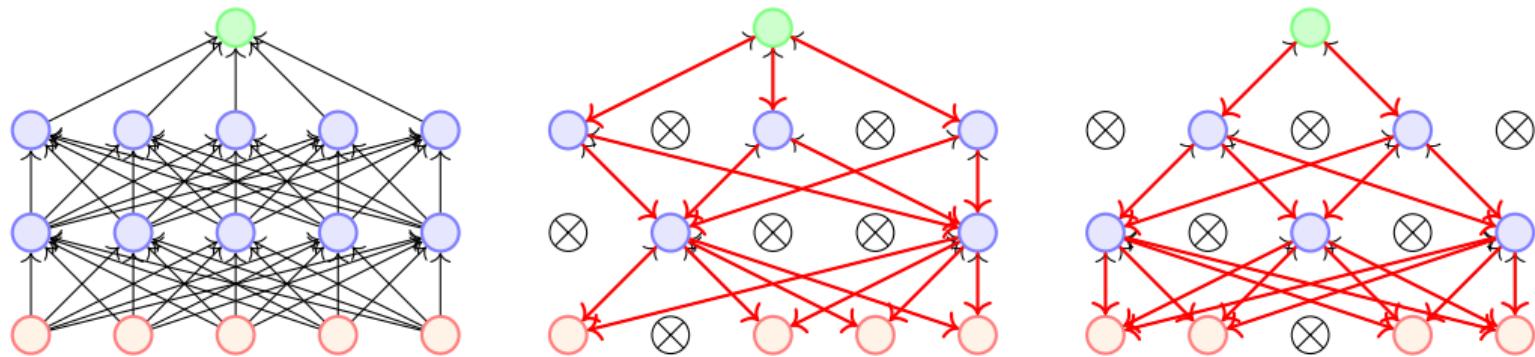
- 对第二个训练实例（一个训练样本或一个 mini-batch）(or mini-batch), 使用 Dropout 得到一个『瘦身』的网络
- 计算损失, 对激活的权重使用反向传播计算梯度



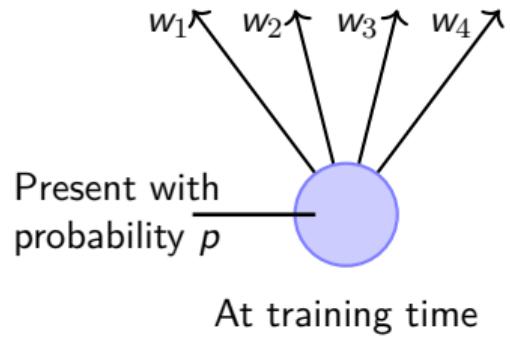
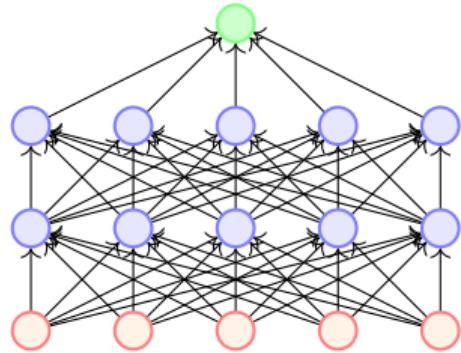
- 对第二个训练实例（一个训练样本或一个 mini-batch）(or mini-batch), 使用 Dropout 得到一个『瘦身』的网络
- 计算损失, 对激活的权重使用反向传播计算梯度

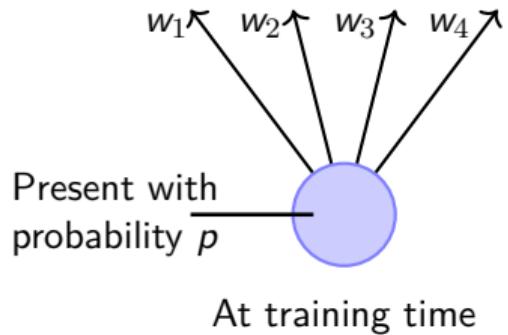
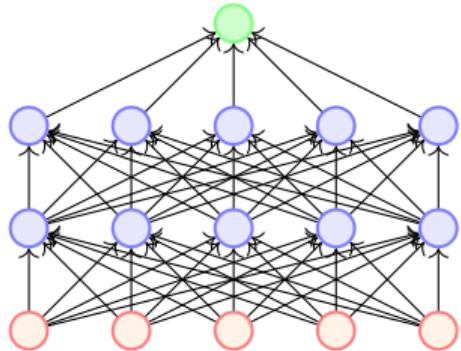


- 对第二个训练实例（一个训练样本或一个 mini-batch）(or mini-batch)，使用 Dropout 得到一个『瘦身』的网络
- 计算损失，对激活的权重使用反向传播计算梯度
- 如果某个权重在两次『瘦身』网络中都激活，它将会得到两次更新

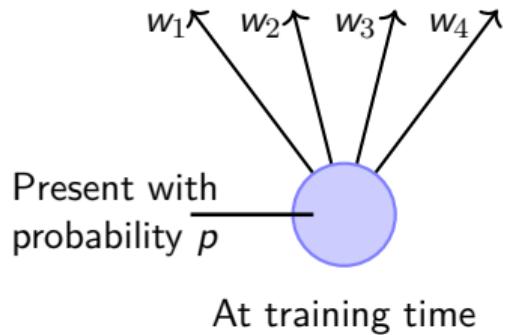
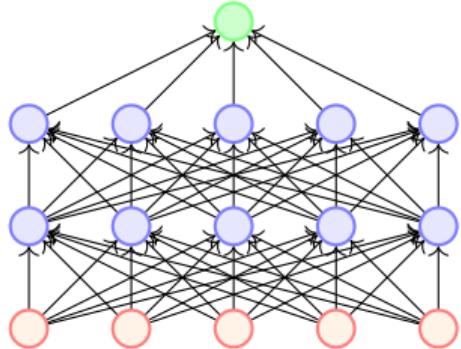


- 对第二个训练实例（一个训练样本或一个 mini-batch）(or mini-batch)，使用 Dropout 得到一个『瘦身』的网络
- 计算损失，对激活的权重使用反向传播计算梯度
- 如果某个权重在两次『瘦身』网络中都激活，它将会得到两次更新
- 如果某个权重在一次『瘦身』网络中激活，它将只会得到一次更新

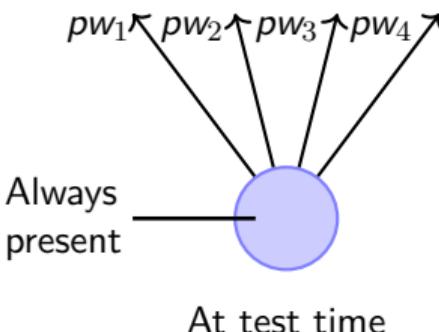
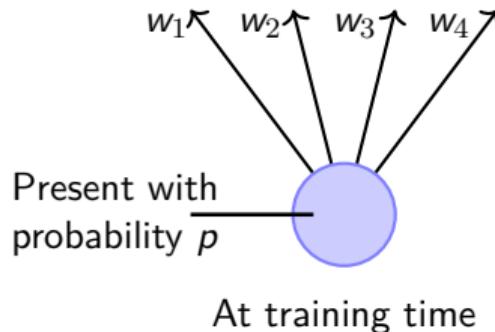
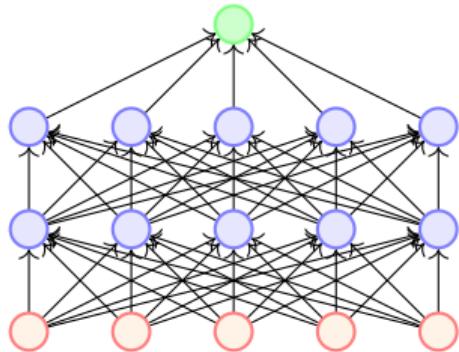




- 测试的时候如何执行？



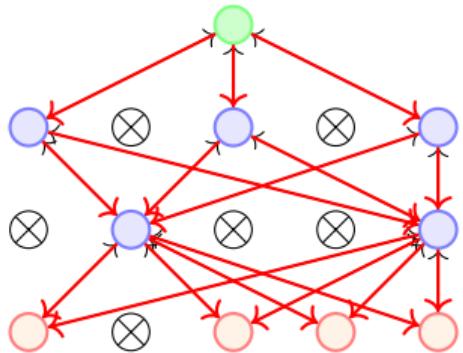
- 测试的时候如何执行？
- 不可能执行每一个瘦身的网络，再把所有  $2^n$  瘦身的网络的输出以某种方式结合起来



- 测试的时候如何执行？
- 不可能执行每一个瘦身的网络，再把所有  $2^n$  瘦身的网络的输出以某种方式结合起来
- 相反，使用没有瘦身的全网络，将每个结点的输出，根据它在训练过程中激活的频率进行缩放

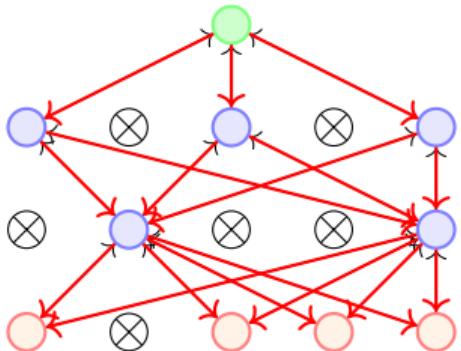


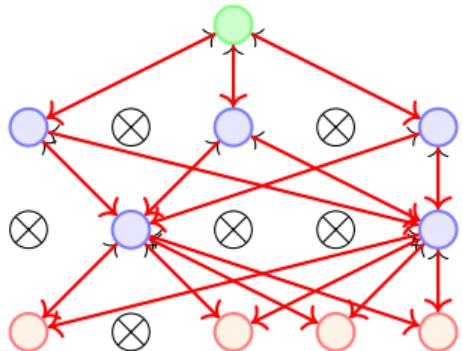
- Dropout 本质上可以看作是对隐含单元施加噪声



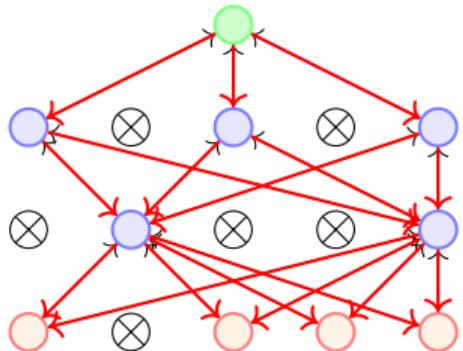


- Dropout 本质上可以看作是对隐含单元施加噪声
- 阻止隐含单元互相适 (co-adapting)

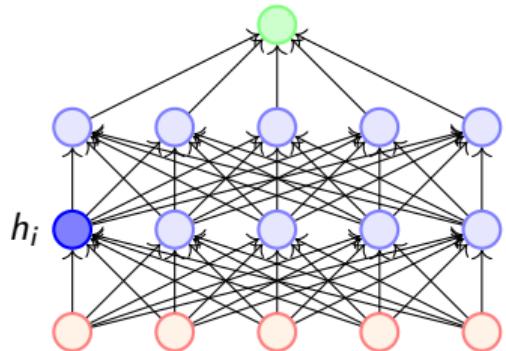




- Dropout 本质上可以看作是对隐含单元施加噪声
- 阻止隐含单元互相适 (co-adapting)
- 一个隐含单元不能过于依赖其他隐含单元 (因为其他隐含单元可能随时被移除)

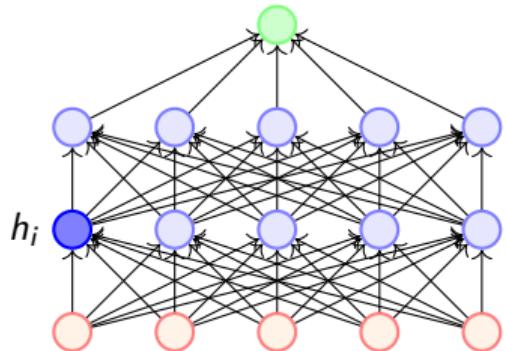


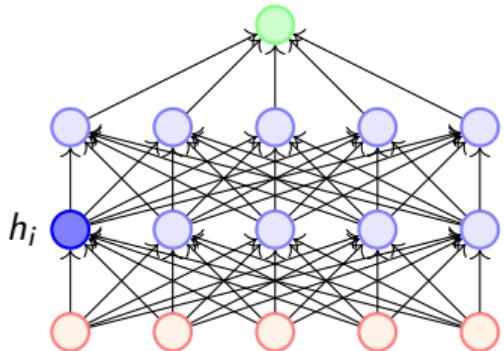
- Dropout 本质上可以看作是对隐含单元施加噪声
- 阻止隐含单元互相适 (co-adapting)
- 一个隐含单元不能过于依赖其他隐含单元 (因为其他隐含单元可能随时被移除)
- 每个隐含单元必须学习到对其他隐含单元的移除足够鲁棒



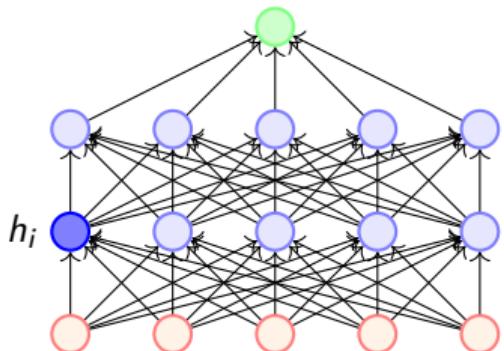


- dropout 提高了网络的冗余性和鲁棒性

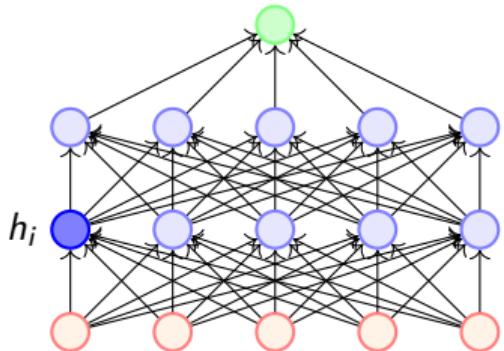




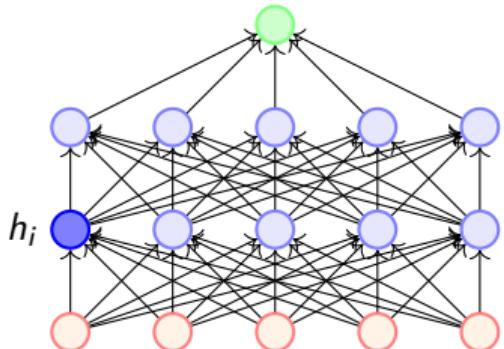
- dropout 提高了网络的冗余性和鲁棒性
- 假设  $h_i$  学习通过检测一个鼻子来判断是否输入图像是一张人脸



- dropout 提高了网络的冗余性和鲁棒性
- 假设  $h_i$  学习通过检测一个鼻子来判断是否输入图像是一张人脸
- 把  $h_i$  移除，等价于拿掉了一个能通过图像中是否有鼻子来判断图像是否是人脸图像的结点



- dropout 提高了网络的冗余性和鲁棒性
- 假设  $h_i$  学习通过检测一个鼻子来判断是否输入图像是一张人脸
- 把  $h_i$  移除，等价于拿掉了一个能通过图像中是否有鼻子来判断图像是否是人脸图像的结点
- 网络的其他  $h_i$  将学习编码图像中是否有鼻子这个信息，因此这些结点编码的信息会存在冗余



- dropout 提高了网络的冗余性和鲁棒性
- 假设  $h_i$  学习通过检测一个鼻子来判断是否输入图像是一张人脸
- 把  $h_i$  移除，等价于拿掉了一个能通过图像中是否有鼻子来判断图像是否是人脸图像的结点
- 网络的其他  $h_i$  将学习编码图像中是否有鼻子这个信息，因此这些结点编码的信息会存在冗余
- 或者是网络会学习其他的特征来进行人脸检测