

法律声明

□ 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

□ 课程详情请咨询

■ 微信公众号：大数据分析挖掘

■ 新浪微博：ChinaHadoop



传统神经网络

主讲人： 李伟

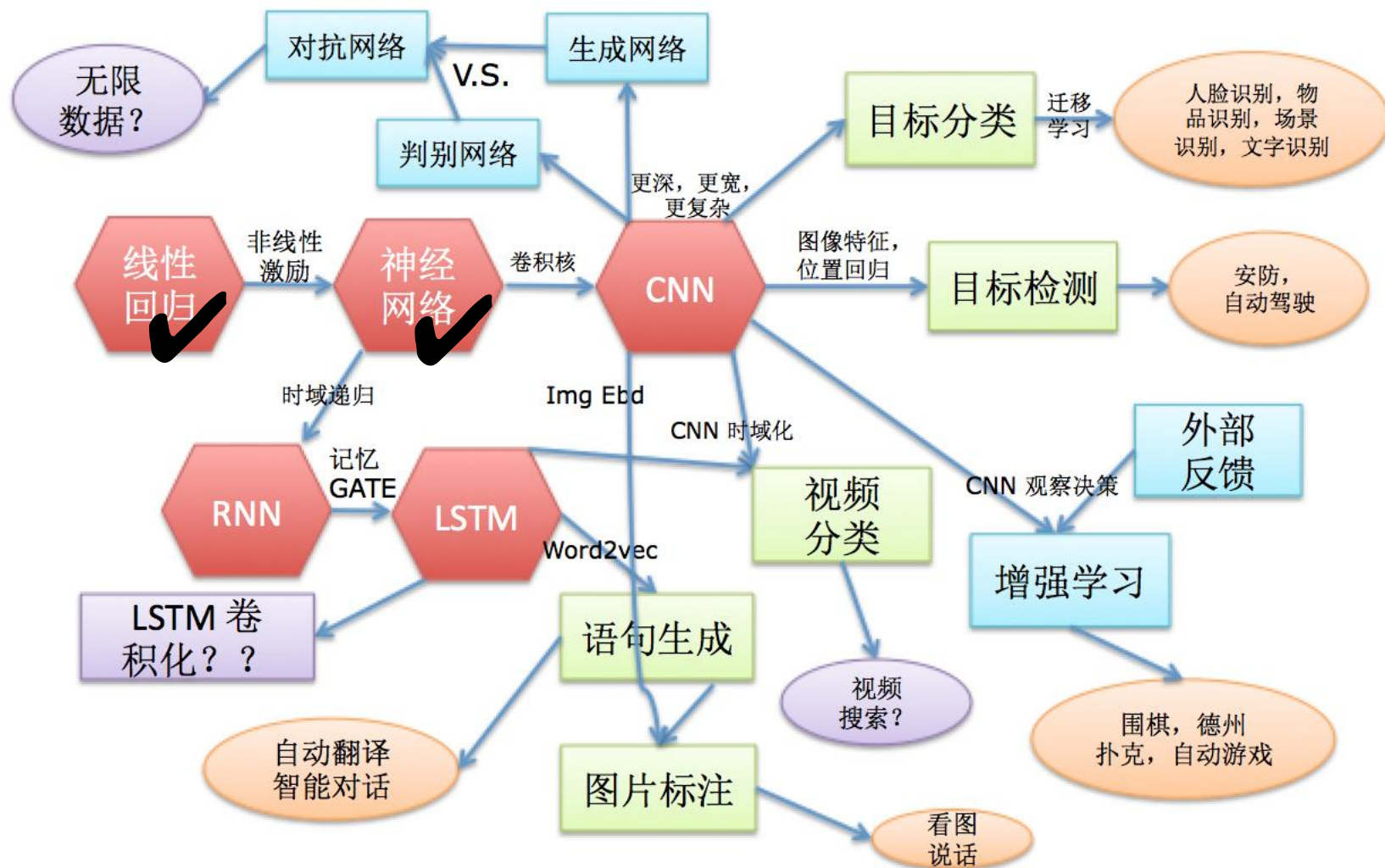
纽约城市大学博士

主要研究深度学习，计算机视觉，人脸计算
多篇重要研究文章作者，重要会议期刊审稿人

微博ID: weightlee03 (相关资料分享)

GitHub ID: wiibrew (课程代码发布)

结构



提纲

- 1. 神经网络起源：线性回归
- 2. 从线性到非线性
- 3. 神经网络的构建
- 4. 神经网络的“配件”

期待目标

- 1. 了解从线性到非线性回归的转化
- 2. 明白如何构建神经网络，了解不同激励函数的区别联系
- 3. 掌握“配件”对神经网络性能的影响（损失函数，学习率，动量，过拟合），会“调参”
- 4. 明白本节所有的[面试题]

提纲

- 1. 神经网络起源：线性回归
- 2. 从线性到非线性
- 3. 神经网络的构建
- 4. 神经网络的“配件”

线性回归

□ 概念: 线性关系来描述输入到输出的映射关系

□ 应用场景:

网络分析

银行风险分析

基金股价预测

天气预报

线性回归

□ 一个线性回归问题

目标方程: $y = ax_1 + bx_2 + cx_3 + d$

参数: $m = [a, b, c, d]$

数据: $[(x_{1,1}, x_{2,1}, x_{3,1}), (x_{1,2}, x_{2,2}, x_{3,2}), \dots, (x_{1,n}, x_{2,n}, x_{3,n})]$
 $[y_1, y_2, \dots, y_n]$

预测: $\hat{y}_t = ax_{1,t} + bx_{2,t} + cx_{3,t} + d$

目标: minimize $(\hat{y}_t - y_t)$

线性回归

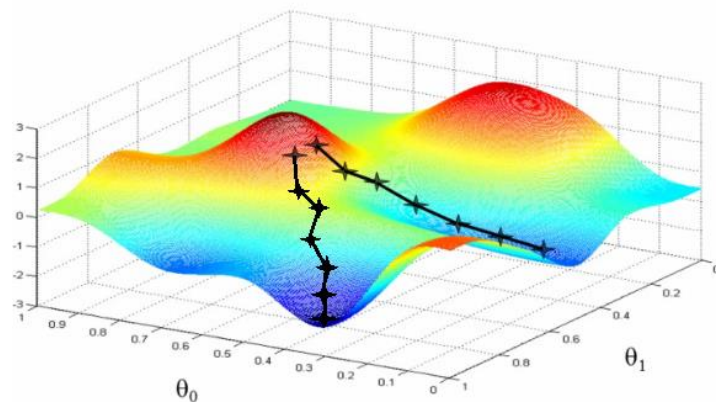
□ 优化方法：梯度下降

□ 模型参数

当前 $m_0 = [a_0, b_0, c_0, d_0]$

每一步 Δm ?

参数： $m = [a, b, c, d]$



山坡高度：Loss

地面位置：参数

山坡最低点：Loss minimal

最低点位置：目标参数

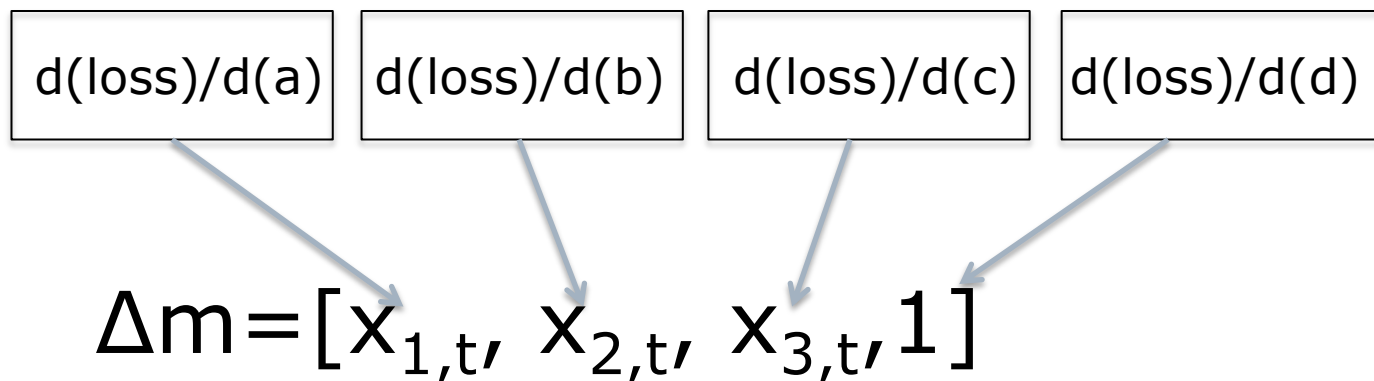
怎么到达：下坡方向，梯度下降

怎么找方向：高度对地面关系导数

线性回归

- 梯度下降：梯度计算

$$\text{Loss} = ax_{1,t} + bx_{2,t} + cx_{3,t} + d - y$$



- 梯度下降：参数更新 $m := m - \eta \Delta m$

线性回归

□ 梯度下降法总结：

随机初始化参数

开启循环： $t = 0, 1, 2, \dots$

 带入数据求出结果 \hat{y}_t

 与真值比较得到 $\text{loss} = y - \hat{y}_t$

 对各个变量求导得到 Δm

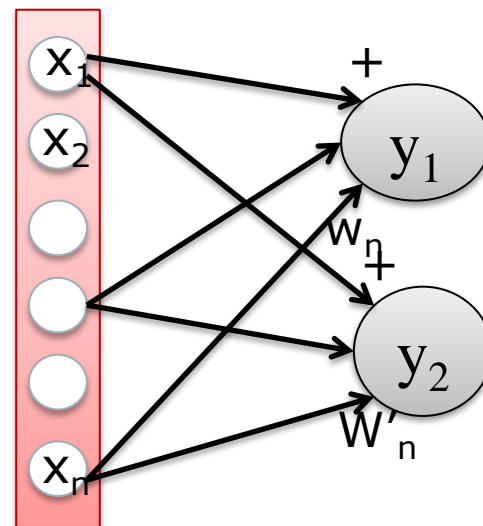
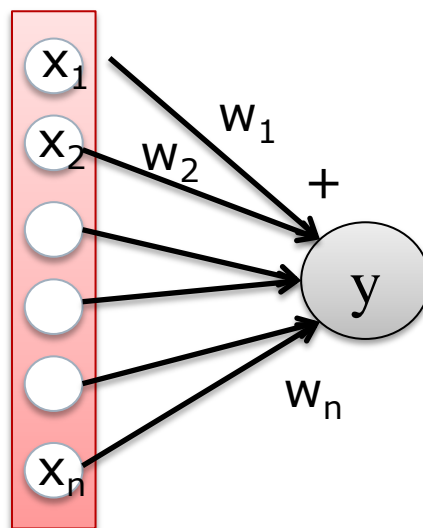
 更新变量 m

 如果 loss 足够小或 t 循环结束，停止

线性回归

□ 输出 +

□ 能否同时预测
多个目标？

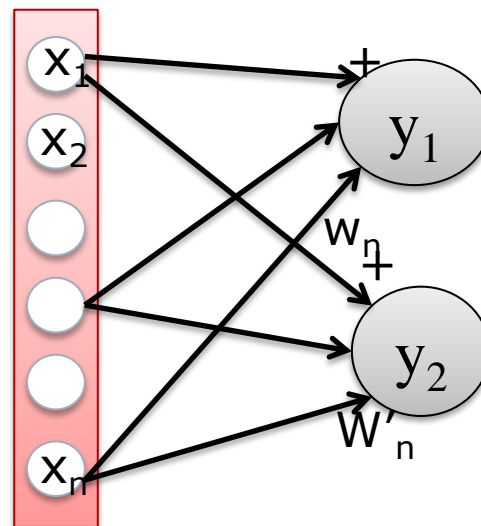
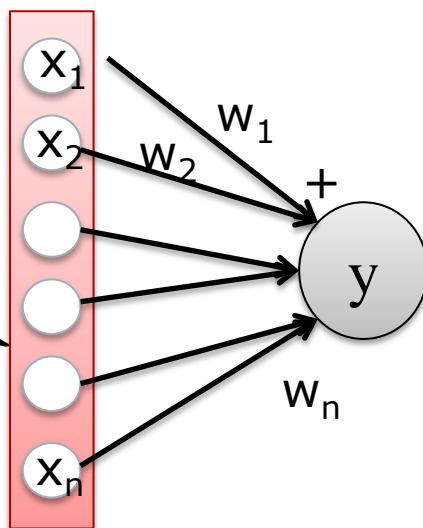


线性回归

□ 输出 +

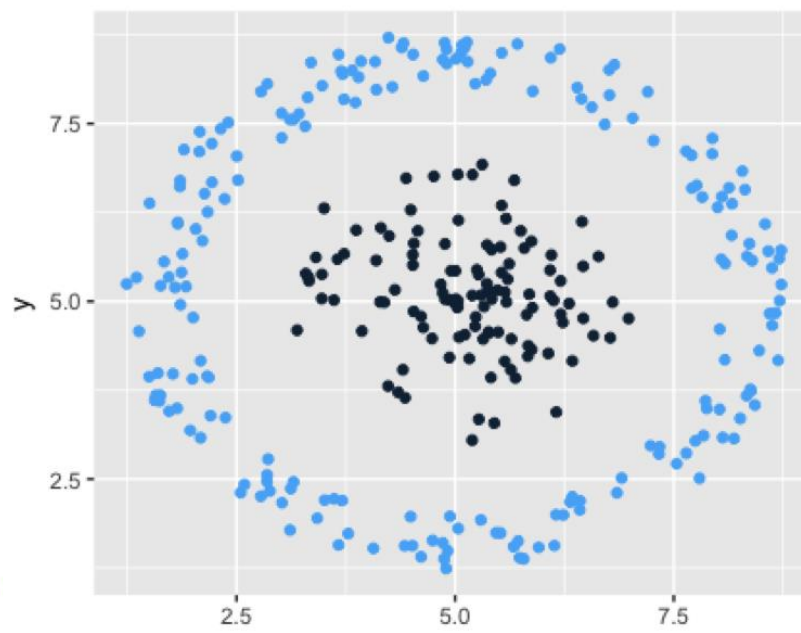
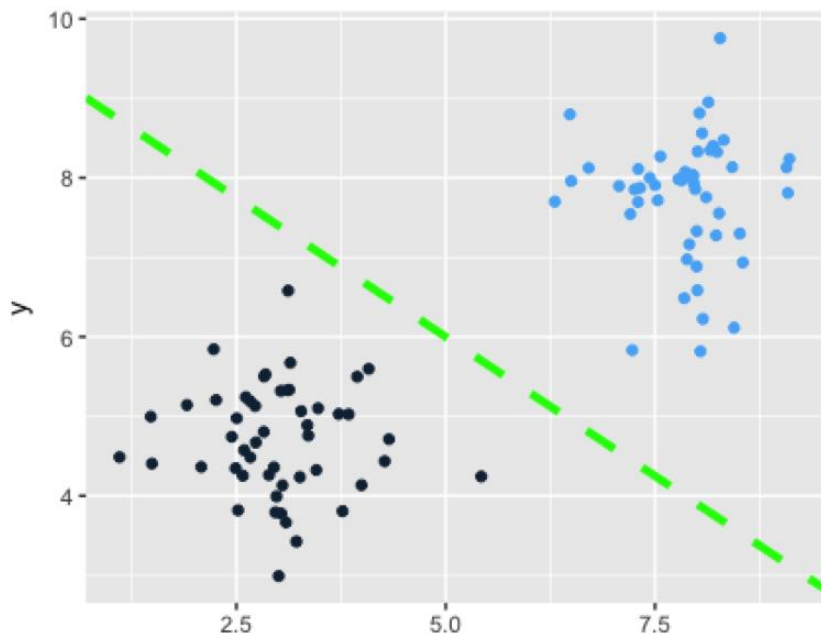
□ 能否同时预测
多个目标？

多目标学习，通过合并多个
任务loss，一般能够产生比
单个模型更好的效果。



线性回归

□ 局限



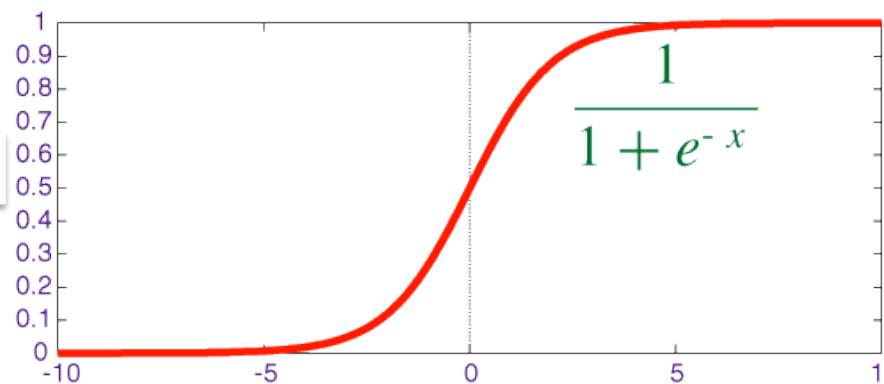
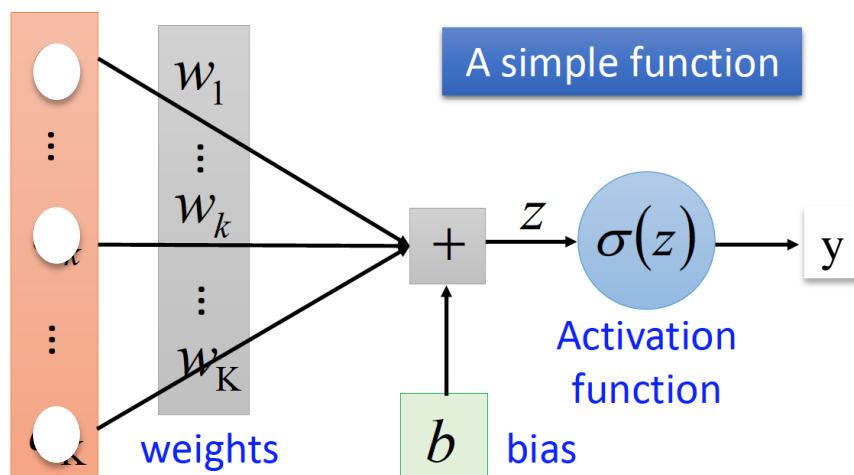
线性回归能够清楚的描述分割线性分布的数据，对非线性分布的数据描述较弱

提纲

- 1. 神经网络起源：线性回归
- 2. 从线性到非线性
- 3. 神经网络的构建
- 4. 神经网络的“配件”

从线性到非线性

□ 非线性激励



□ 考量标准:

- 1.正向对输入的调整
- 2.反向梯度损失

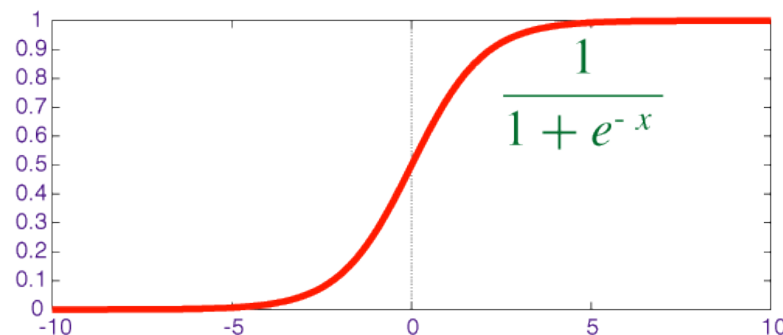
从线性到非线性

□ 常用的非线性激励函数

□ Sigmoid,

□ 函数效果，导数

□ 优点，缺点



$$y(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$y(x)' = y(x)(1 - y(x))$$

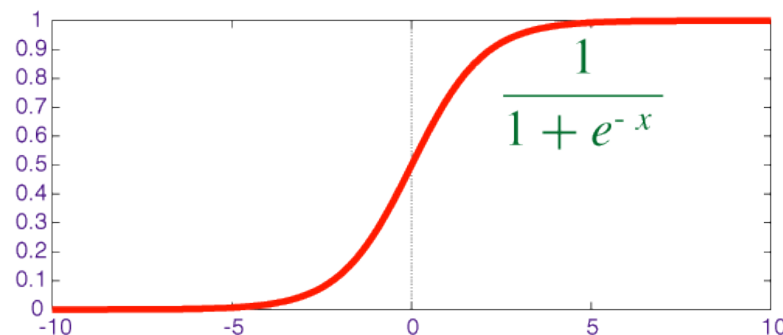
从线性到非线性

□ 常用的非线性激励函数

□ Sigmoid,

□ 函数效果，导数

□ 优点，缺点



$$y(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

将输入数据映射到 $[0, 1]$

$$y(x)' = y(x)(1 - y(x))$$

梯度下降非常明显，至少减少75%

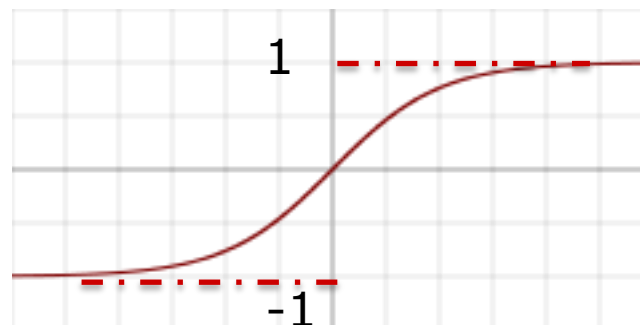
从线性到非线性

□ 常用的非线性激励函数

□ tahn,

□ 函数效果，导数

□ 优点，缺点



$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

$$f'(x) = 1 - f(x)^2$$

从线性到非线性

□ 常用的非线性激励函数

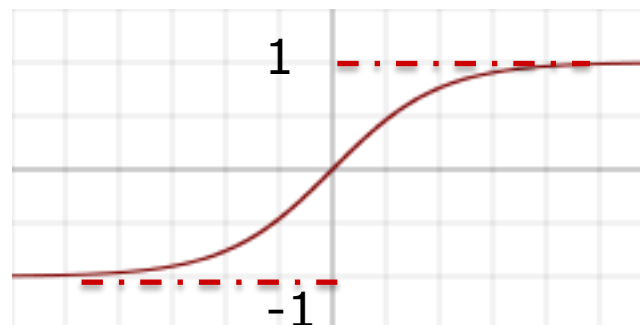
□ tahn,

□ 函数效果，导数

□ 优点，缺点

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

$$f'(x) = 1 - f(x)^2$$



将输入数据映射到 $[-1, 1]$

梯度损失明显

从线性到非线性

□ 常用的非线性激励函数

□ ReLU(Rectified linear unit)

□ 函数效果，导数

□ 优点，缺点



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

从线性到非线性

□ 常用的非线性激励函数

□ ReLU(Rectified linear unit)

□ 函数效果，导数

□ 优点，缺点



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

正向截断负值，损失大量特征

反向梯度没有损失

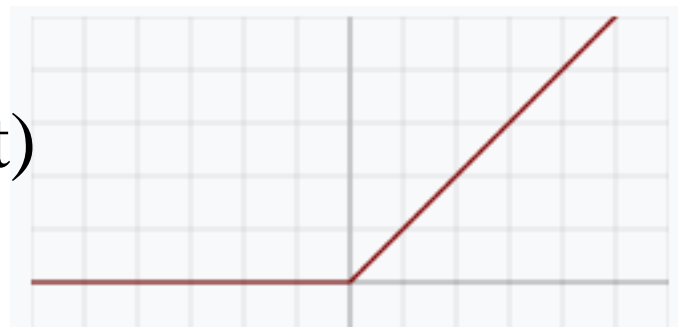
从线性到非线性

□ 常用的非线性激励函数

□ ReLU(Rectified linear unit)

□ 函数效果，导数

□ 优点，缺点



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

正向截断负值，损失大量特征
为什么还用？

反向梯度没有损失

从线性到非线性

□ 常用的非线性激励函数

□ ReLU(Rectified linear unit)

□ 函数效果，导数

□ 优点，缺点



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

正向截断负值，损失大量特征
为什么还用？特征足够多

反向梯度没有损失

从线性到非线性

□ 常用的非线性激励函数

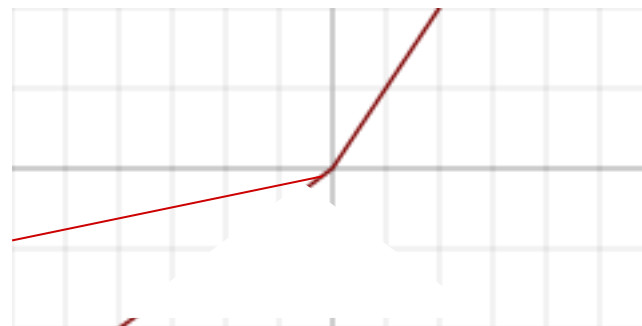
□ Leaky ReLU(Rectified linear unit)

□ 函数效果，导数

□ 优点，缺点

$$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$



保留更多参数，少量梯度反向传播

为什么不变成 $y=x$ ？

从线性到非线性

□ 常用的非线性激励函数

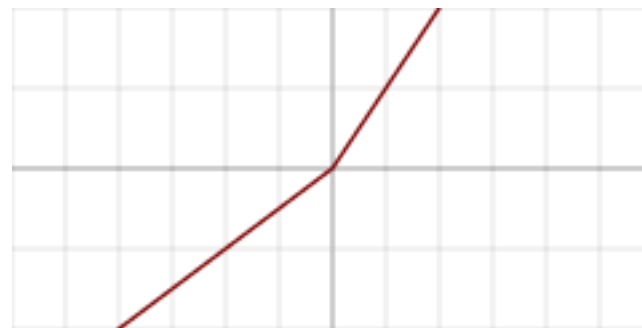
□ Leaky ReLU(Rectified linear unit)

□ 函数效果，导数

□ 优点，缺点

$$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

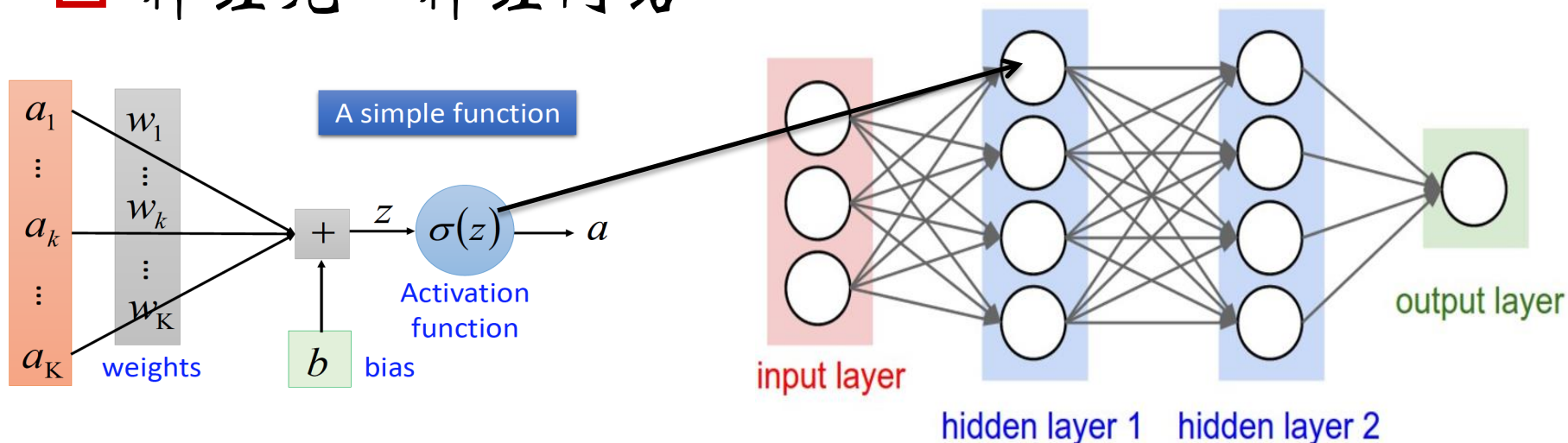


保留更多参数，少量梯度反向传播

为什么不变成 $y=x$? 线性了

从线性到非线性

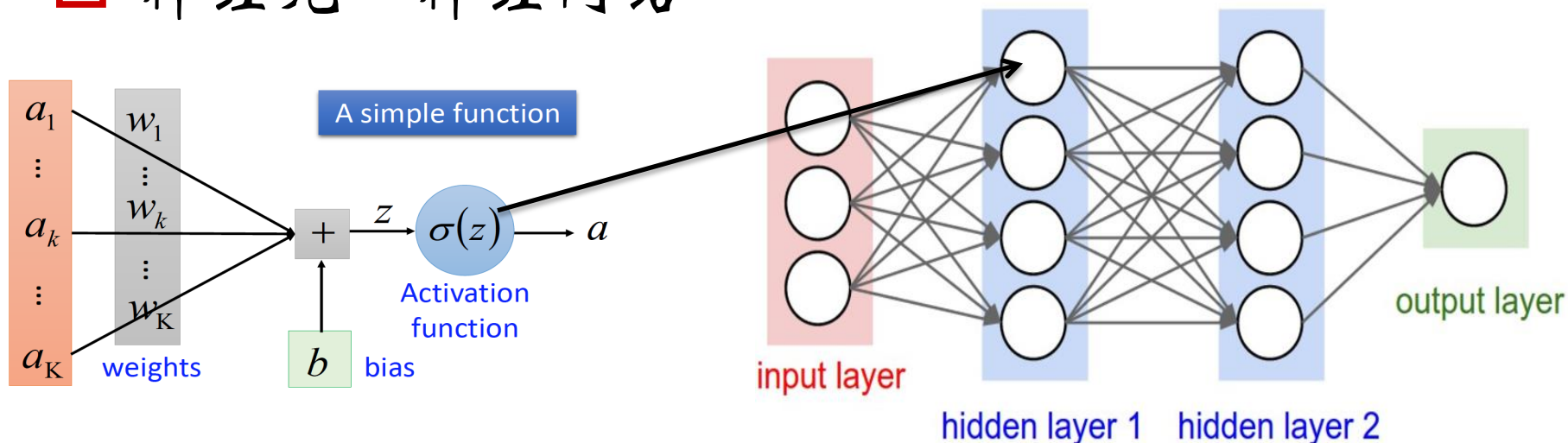
□ 神经元-神经网络



□ [面试题] 有线性回归网络吗?

从线性到非线性

□ 神经元-神经网络



□ [面试题] 有线性回归网络吗?

并没有。。

$$X_1 = W_0 \cdot X_0, X_2 = W_1 \cdot X_1, Y = W_2 \cdot X_2$$

$$Y = W_2 \cdot W_1 \cdot W_0 \cdot X_0 = W_3 \cdot X_0$$

提纲

- 1. 神经网络起源：线性回归
- 2. 从线性到非线性
- 3. 神经网络的构建
- 4. 神经网络的“配件”

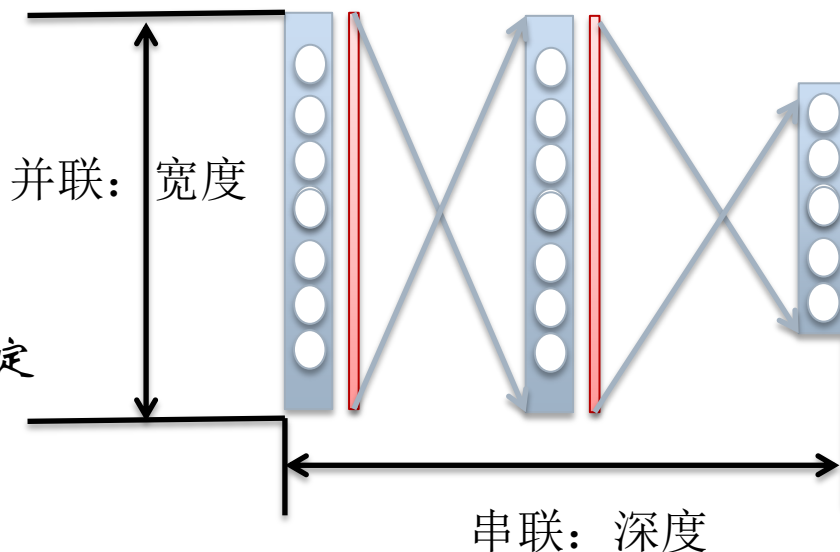
神经网络构建

□ 神经元的“并联”和“串联”

- 从第一层神经网络到最终输出，每一个神经元的数值由前一层神经元数值，神经元参数W，b以及激励函数共同决定第n+1层第k个神经元的方程可由公式表示为：

$$z_{n+1,k} = \sum_{i=1}^m W_{n,k,i} \cdot x_{n,i} + b_{n,k}$$

$$y_{n+1,k} = \frac{1}{1 + e^{-z_{n+1,k}}}$$



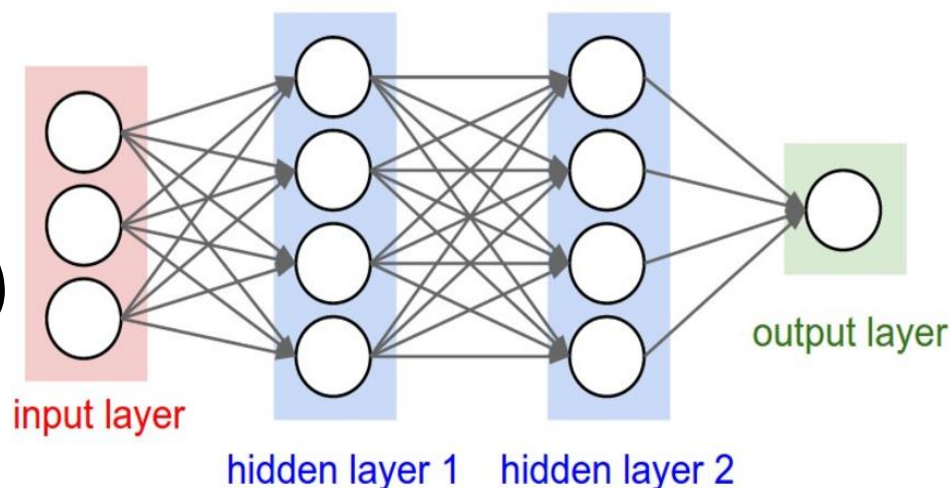
在这里，m表示第n层神经网络的宽度，n为当前神经网络的深度

神经网络构建

神经网络优化

□ 链式法则

(下一讲主要内容)



□ 计算梯度

output->last layer Loss--> Δy_n

layer->layer Δy_n --> Δx_n

layer->parameter Δy_n --> Δw_n

怎么计算 Δw_i ? $\Delta y_i, \Delta x_{i+1}, \dots, \Delta y_n$,

神经网络构建

神经网络求导 - TensorFlow 实现

```
data = tf.placeholder(tf.float32)
var = tf.Variable(...)
loss = some_function_of(var, data)
var_grad = tf.gradients(loss, [var])
```

```
sess = tf.Session()
var_grad_val = sess.run(var_grad, feed_dict={data: ...})
```


神经网络构建

神经网络实例分析：MINIST 神经网络分类

见：course_2_tf_nn.py

结构变化影响

- ☐ “并联” 宽度影响
- ☐ “串联” 层数影响
- ☐ Dropout
- ☐ Learning rate

提纲

- 1. 神经网络起源：线性回归
- 2. 从线性到非线性
- 3. 神经网络的构建
- 4. 神经网络的“配件”

神经网络的“配件”

□ 1. 损失函数-Loss

- 影响深度学习性能最重要因素之一。是外部世界对神经网络模型训练的直接指导。
- 合适的损失函数能够确保深度学习模型收敛
- 设计合适的损失函数是研究工作的主要内容之一

神经网络的“配件”

□ 1. 损失函数

□ Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

□ Loss影响？

神经网络的“配件”

□ 1. 损失函数

□ Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

□ Loss影响？

$[1, 2, 3, 4, 1, 2, 3] \longrightarrow [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]$

神经网络的“配件”

□ 1. 损失函数

□ Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

□ Loss影响？

$[1, 2, 3, 4, 1, 2, 3] \longrightarrow [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]$

□ Softmax的好处？

神经网络的“配件”

□ 1. 损失函数

□ Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

□ Loss影响？

$[1, 2, 3, 4, 1, 2, 3] \longrightarrow [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]$

□ Softmax的好处？

分类问题的预测结果更明显

神经网络的“配件”

□ 1. 损失函数

□ Cross entropy

$$L(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N H(p_n, q_n) = -\frac{1}{N} \sum_{n=1}^N \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]$$

□ Explode?->

$$Loss = -\sum (l \cdot \log(\frac{p + 0.05}{1.05}) + (1 - l) \cdot \log(\frac{1.05 - p}{1.05}))$$

W. Li, F. Abtahi, Z. Zhu, Action Unit Detection with Region Adaptation, Multi-labeling Learning and Optimal Temporal Fusing. CVPR 2017.

□ 用途?

神经网络的“配件”

□ 1. 损失函数

□ Cross entropy

$$L(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N H(p_n, q_n) = -\frac{1}{N} \sum_{n=1}^N \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]$$

□ Explode?->

$$Loss = -\sum (l \cdot \log(\frac{p + 0.05}{1.05}) + (1 - l) \cdot \log(\frac{1.05 - p}{1.05}))$$

W. Li, F. Abtahi, Z. Zhu, Action Unit Detection with Region Adaptation, Multi-labeling Learning and Optimal Temporal Fusing. CVPR 2017.

□ 用途?

目标为 $[0, 1]$ 区间的回归问题，以及生成

神经网络的“配件”

□ 1. 损失函数

自定义

□ a. 看中某一个属性

单独将某一些预测值取出或赋予不同大小的参数

□ b. 合并多个loss

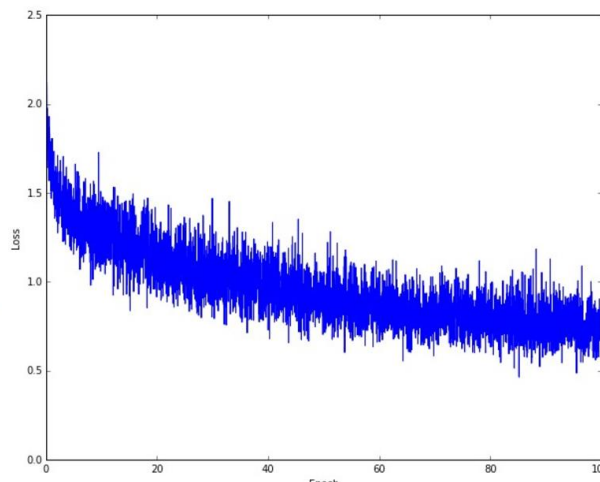
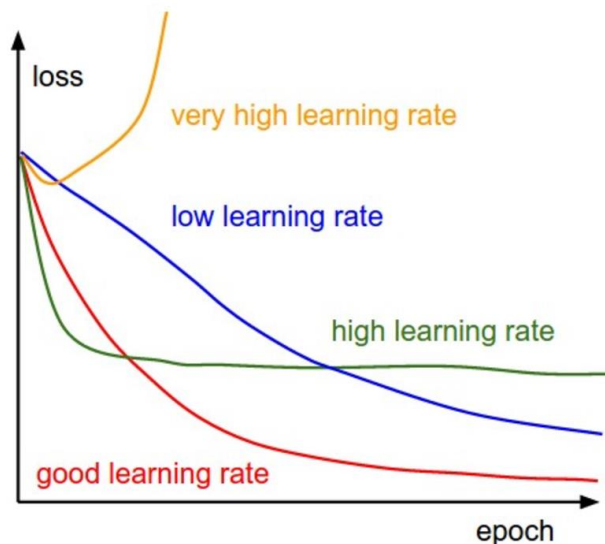
多目标训练任务，设置合理的loss结合方式(各种运算)

□ c. 神经网络融和

不同神经网络loss结合，共同loss对网络进行训练指导

神经网络的“配件”

□ 2. 学习率 Learning rate



□ 数值大：收敛速度快

□ 数值小：精度高

神经网络的“配件”

□ 2. 学习率 Learning rate

□ 如何选用合适的学习率？

1. Fixed; 2. Step;

3. Adagrad (知道定义即可)

```
# Assume the gradient dx and parameter vector x  
cache += dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

4. RMSprop

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

神经网络的“配件”

□ 3. 动量

□ 正常 `x += - learning_rate * dx`

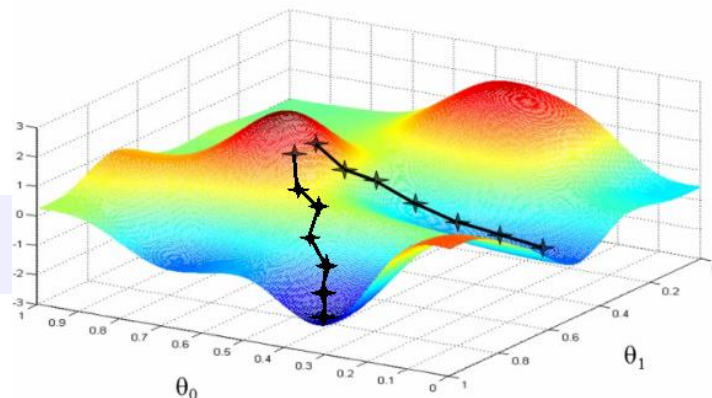
□ Why?

沿着已经得到的优化方向前进，不用重新找方向，只需微调

□ How?

$v = 0$

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```



神经网络的“配件”

□ 3. 动量

□ 正常 `x += - learning_rate * dx`

□ Why?

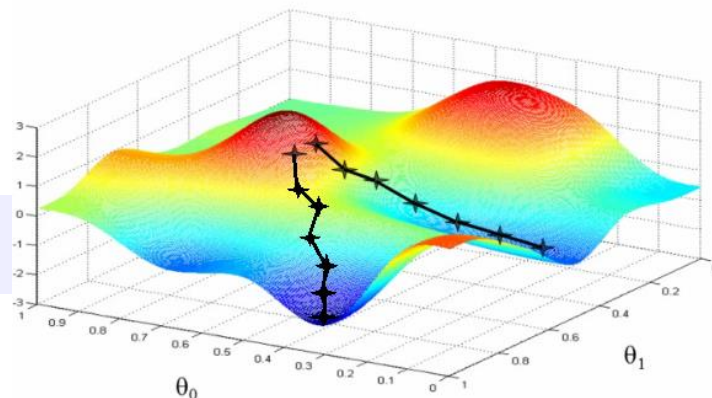
沿着已经得到的优化方向前进，不用重新找方向，只需微调

□ How?

$v = 0$

```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

[面试题] 用动量和直接调大学习率有什么区别？



神经网络的“配件”

□ 3. 动量

□ 正常 `x += - learning_rate * dx`

□ Why?

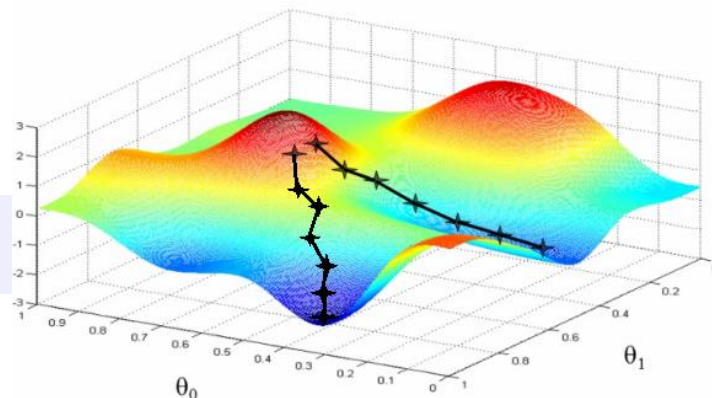
沿着已经得到的优化方向前进，不用重新找方向，只需微调

□ How?

```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

[面试题] 和直接调大学习率有什么区别？

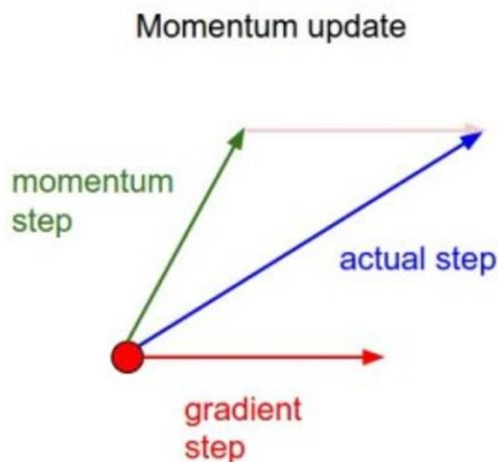
方向不同找的更准确



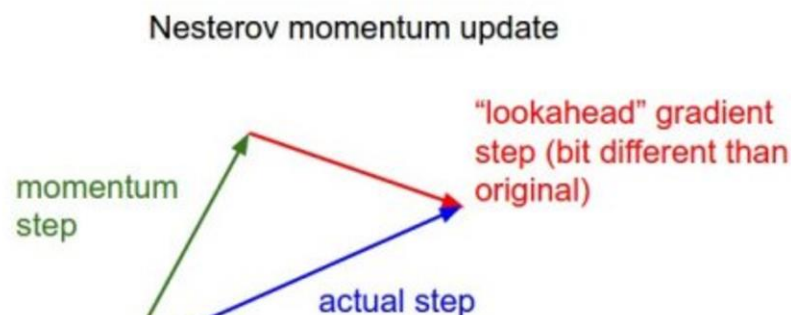
神经网络的“配件”

□ 3. 动量

Nesterov 动量



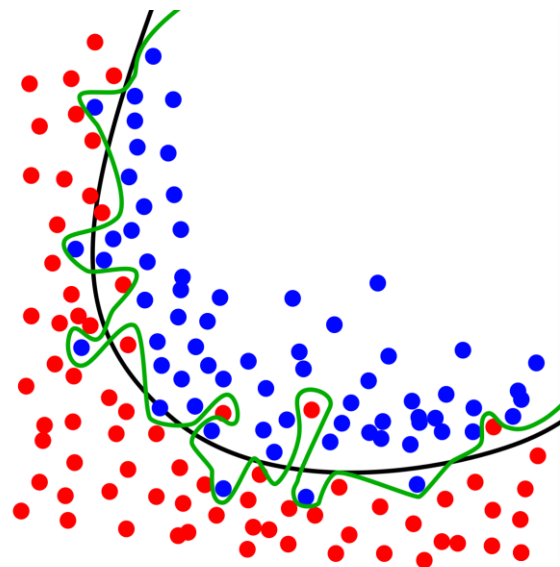
理解：新的梯度更新是在动量投射的基础上



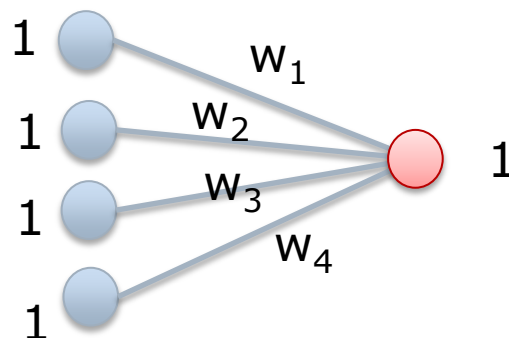
```
x_ahead = x + mu * v
# evaluate dx_ahead (the gradient at x_ahead)
v = mu * v - learning_rate * dx_ahead
x += v
```


神经网络的“配件”

4. 过拟合(Overfitting):



哪个参数更好？

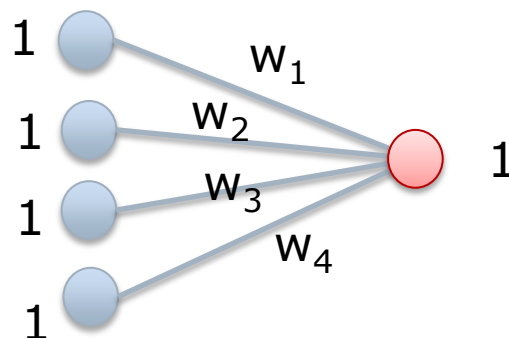
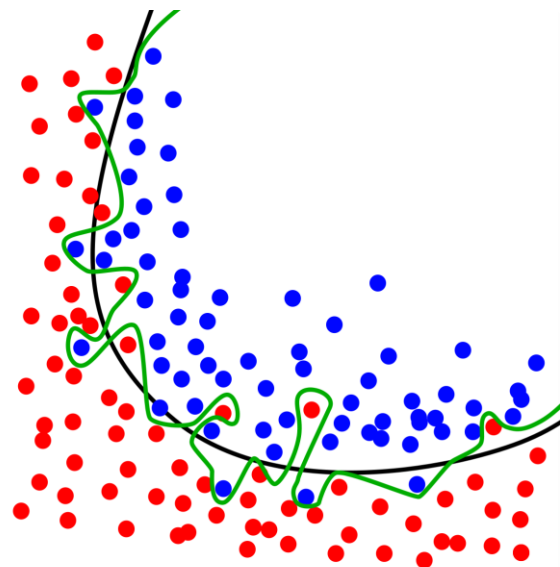


$$W' = [0.25, 0.25, 0.25, 0.25]$$

$$W'' = [1, 0, 0, 0]$$

神经网络的“配件”

4. 过拟合(Overfitting):



$$W' = [0.25, 0.25, 0.25, 0.25]$$

$$W'' = [1, 0, 0, 0]$$

哪个参数更好？

如果更多的参数能够参与决策，会对输入有更高的适应性， w'' 更好

神经网络的“配件”

4. 过拟合---应对:

☐ Regularization

Before

After

☐ $\text{Loss} = \hat{y} - y;$

☐ $\Delta w = d(\text{Loss})/d(w);$

☐ $w := w - \eta \Delta w;$

神经网络的“配件”

4. 过拟合---应对:

□ Regularization

Before

After

□ $Loss = \hat{y} - y;$

$$Loss' = \hat{y} - y + \lambda \cdot ||w^2||$$

□ $\Delta w = d(Loss)/d(w);$ $\Delta w = d(Loss)/d(w) + 2\lambda \cdot w$

□ $w := w - \eta \Delta w;$ $w := w - \eta \Delta w - 2\eta \lambda w$

Regularization 对参数 w 有什么影响?

神经网络的“配件”

4. 过拟合---应对:

□ Regularization

Before

After

□ $Loss = \hat{y} - y;$

$$Loss' = \hat{y} - y + \lambda \cdot ||w^2||$$

□ $\Delta w = d(Loss)/d(w);$ $\Delta w = d(Loss)/d(w) + 2\lambda \cdot w$

□ $w := w - \eta \Delta w;$ $w := w - \eta \Delta w - 2\eta \lambda w$

Regularization 对参数 w 有什么影响?

为了使Loss ‘最小， w^2 部分要求 w 的值尽量平衡（why），
和Loss共同影响 w 变化

神经网络的“配件”

4. 过拟合---应对:

□ Regularization

Before

□ $Loss = \hat{y} - y;$

□ $\Delta w = d(Loss)/d(w);$

□ $w := w - \eta \Delta w;$

After

$$Loss' = \hat{y} - y + \lambda \cdot ||w^2||$$

□ $\Delta w = d(Loss)/d(w) + 2\lambda \cdot w$

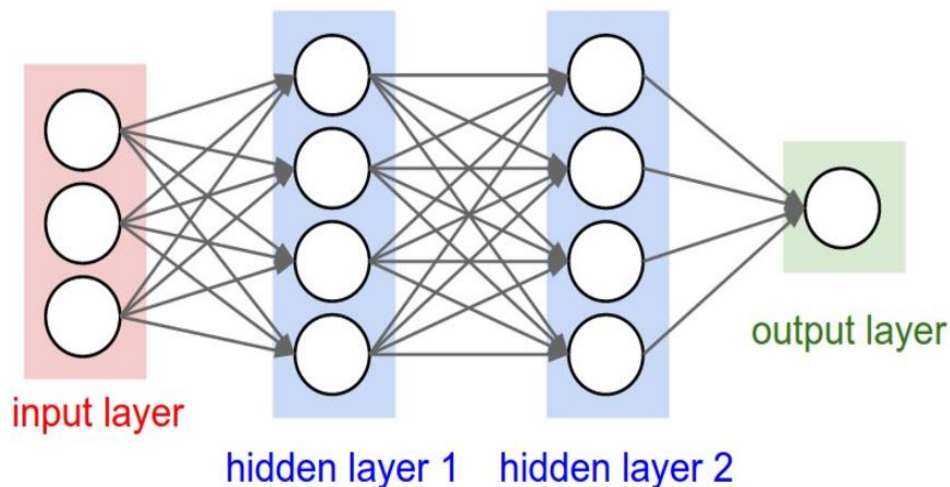
□ $w := w - \eta \Delta w - 2\eta \lambda w$

[面试题] 什么叫做weight decay,与Regularization有何联系?

神经网络的“配件”

4. 过拟合---应对:

□ Dropout

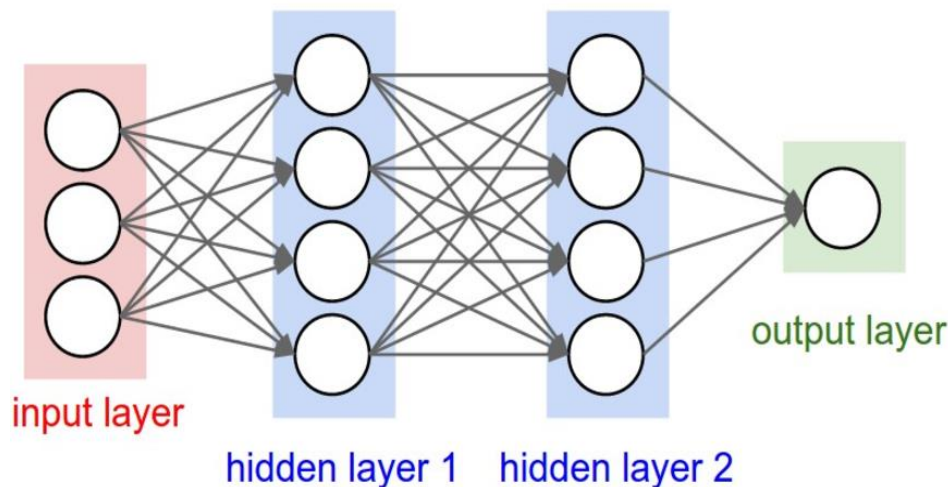


□ [面试题] Dropout, Pooling 区别?

神经网络的“配件”

4. 过拟合---应对:

□ Dropout



□ [面试题] Dropout, Pooling 区别?

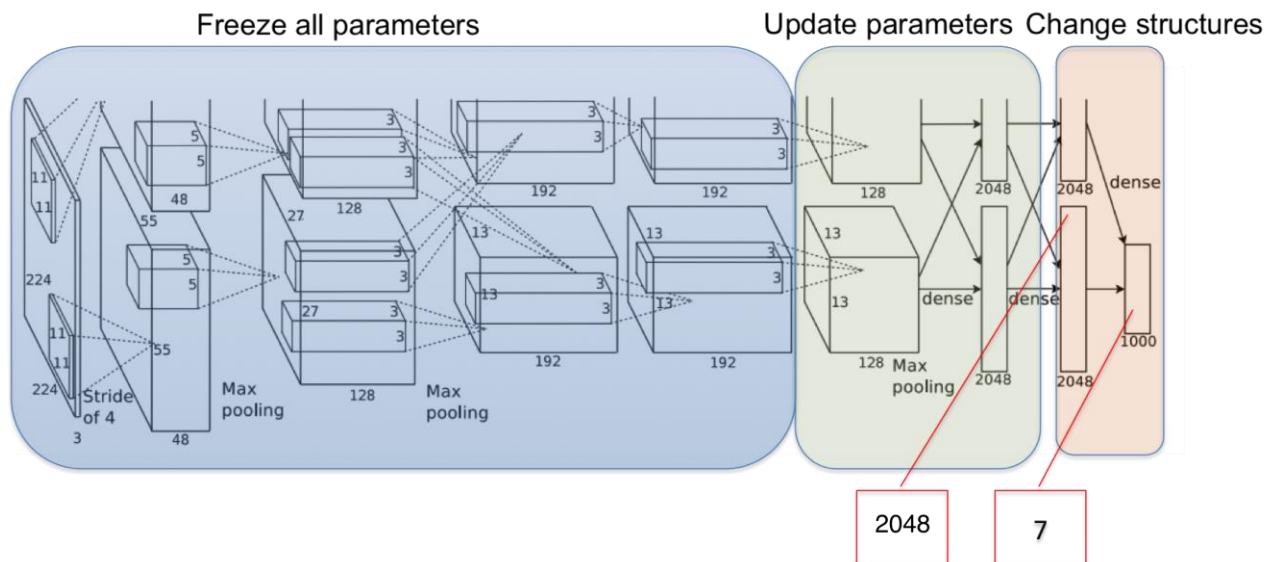
Pooling的本质是降维,

Dropout的本质是Regularization (为什么?)

神经网络的“配件”

4. 过拟合---应对:

□ Fine-tuning



□ 大部分的参数不用更新，实际的参数大量减少

总结

- 1. 神经网络起源：线性回归
 - 2. 从线性到非线性
 - 3. 神经网络的构建
 - 4. 神经网络的“配件”
-
- 下节课预告：链式规则反向求导，SGD优化原理，卷积神经网络(CNN)各个layers介绍

总结

□ 有问题请到课后交流区

□ 问题答疑：<http://www.xxwenda.com/>

■ 可邀请老师或者其他回答人回答问题

□ 课堂QQ群452946104，微信群

□ 讲师微博：weightlee03，每周不定期分享DL资料

□ GitHub ID：wiibrew（课程代码发布）