自然语言处理

杨海钦

2025-2026-1学期

基于Junjie, Graham, Diyi等课件更新

面试问题

- 请解释反向传播的核心原理，以及链式法则在其中的作用。
- 激活函数的核心作用是什么？列举3种常用的激活函数，并说明各自的适合场景与优缺点。
- 什么是过拟合？在神经网络中，有哪些常用的防止过拟合的方法？

大纲

- 张量和数值计算/Tensors and Numerical Computation
- **模型**: 定义神经网络架构
- **前向计算**: 计算预测值与损失
- **反向传播/Backpropagation**: 计算梯度
- **优化/Optimization**: 参数更新
- 训练技巧/Training Tricks

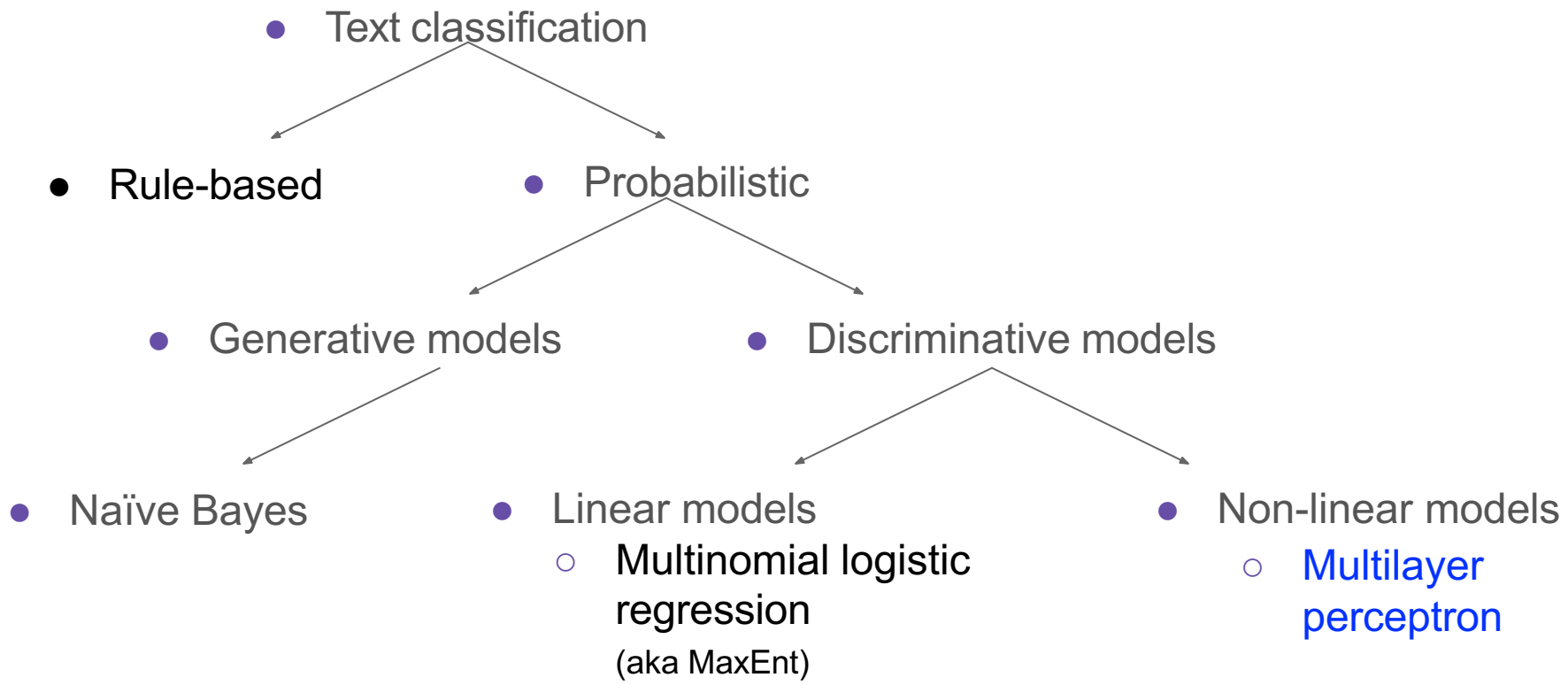
后续课程，我们将探讨

1. 如何将文本“吸收/digest”成函数可用的形式*F*?
   (关键词：特征、特征提取、特征选择、表征)
   feature, extraction/selection, representation

2. 我们可以用什么样的策略来创建决策函数*f*?
   (关键词：建模 models)

3. 如何评估决策函数*s*?
   (关键词：评估 evaluation)

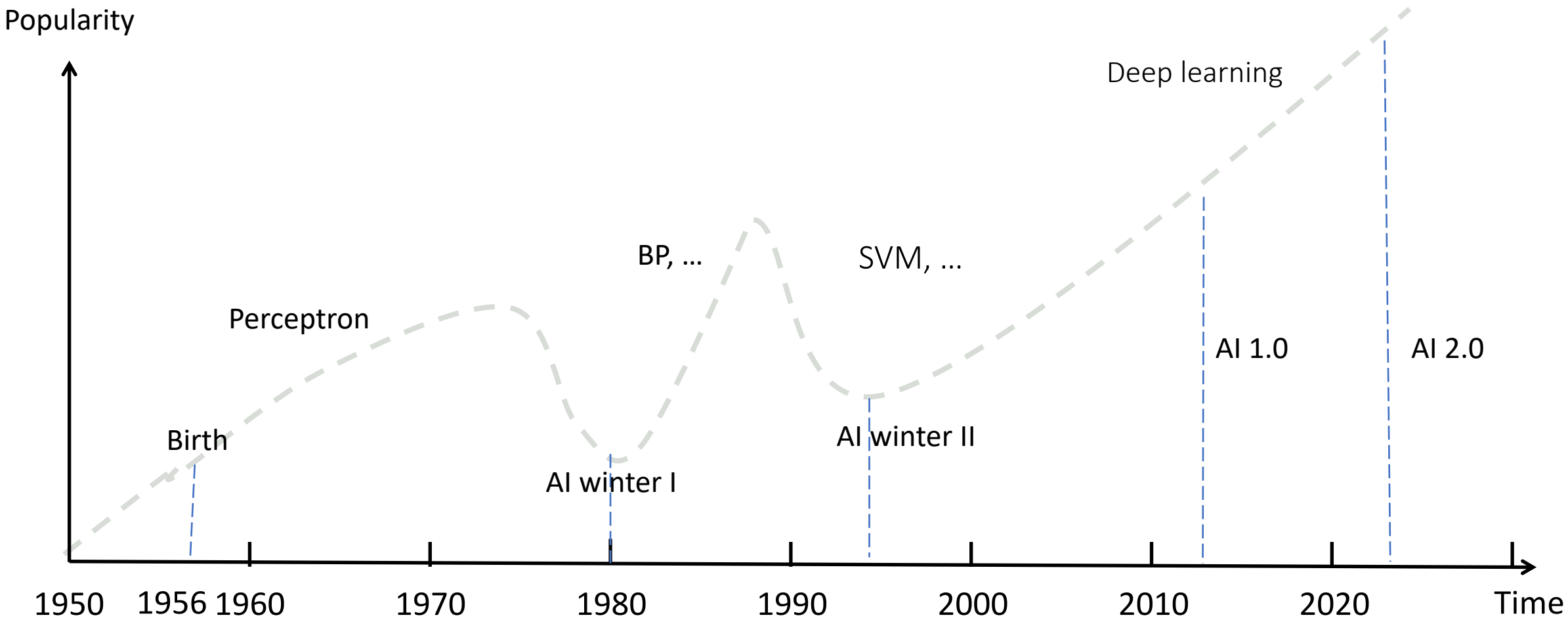
判定式线性模型: 感知器/Perceptron



概率机器学习分类器的组成部分

- 给定N个样本(输入/输出对): $\left(x^{(i)}, y^{(i)}\right)$
  1. 计算输入的特征表示: 对于每个输入观测值$x^{(i)}$ ，获得其对应的n维特征表示$\left[x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right]$, 通常其特征$j$亦可记为$F_j(x)$
  2. 通过分类函数计算其预测值$\hat{y} = P(y|x)$, 分类函数类似于sigmoid或softmax函数
  3. 学习的目标函数，比如交叉熵损失
  4. 优化目标函数的算法：如随机梯度下降

历史


Popularity
Deep learning
BP, …
SVM, …
Perceptron
AI 1.0
AI 2.0
Birth
AI winter I
AI winter II
1950
1956
1960
1970
1980
1990
2000
2010
2020
Time

神经网络框架

神经网络算法梗概

- 创建一个模型并定义其损失（即构造一个计算图）
- 对于每个样本
  - 前向/Forward：计算结果（预测和损失）
  - 如果训练
    - 进行反向传播/back propagation
    - 更新参数

大纲

- 张量和数值计算/Tensors and Numerical Computation
- 模型: 定义神经网络架构
- 前向计算: 计算预测值与损失
- 反向传播/Backpropagation: 计算梯度
- 优化/Optimization: 参数更新
- 训练技巧/Training Tricks

数值计算后端

- 大多数神经网络库使用后端/backend进行数值计算
- PyTorch/Tensorflow: MKL, cuDNN, CUDA, OpenMP，自定义编写内核
- 支持张量上的许多数值函数

```
import torch
import torch.nn as nn
print(*torch.__config__.show().split("\n"), sep="\n")
```

```
PyTorch built with:
  - GCC 7.3
  - C++ Version: 201402
  - Intel(R) Math Kernel Library Version 2020.0.0 Pro
  - Intel(R) MKL-DNN v2.2.3 (Git Hash 7336ca9f055cf1b
  - OpenMP 201511 (a.k.a. OpenMP 4.5)
  - LAPACK is enabled (usually provided by MKL)
  - NNPACK is enabled
  - CPU capability usage: AVX2
  - CUDA Runtime 11.1
  - NVCC architecture flags: -gencode;arch=compute_37
  - CuDNN 8.0.5
  - Magma 2.5.2
  - Build settings: BLAS_INFO=mkl, BUILD_TYPE=Release
```

张量/Tensors

- N维数组: 广泛应用于神经网络

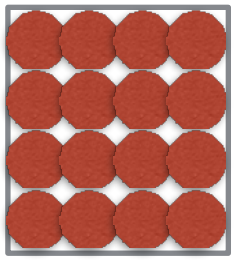
标量/ Scalar

$x \in \mathbb{R}$

矢量/ Vector

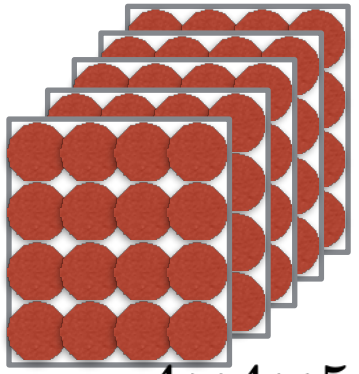
$\mathbf{x} \in \mathbb{R}^4$

矩阵/ Matrix



$\mathbf{X} \in \mathbb{R}^{4\times4}$

3维张量/ 3-dim Tensor



$\mathbf{X} \in \mathbb{R}^{4\times4\times5}$

- 神经网络中的参数由不同形状的张量组成，张量存储了它们的值和梯度

张量计算

- PyTorch/Tensorflow：支持许多不同的矩阵运算：矩阵-乘法

```
import torch
import torch.nn as nn

x = torch.randn((4, 2))
W = torch.randn((3, 4))
print(x)
print(W)
```

```
tensor([[-1.5372,  0.0845],
        [ 0.5752,  0.7634],
        [ 0.4265, -0.1287],
        [-1.8629, -0.8520]])
tensor([[-1.1272, -1.1810,  0.0867,  0.0676],
        [-0.1070,  3.2586,  0.7446, -1.2094],
        [-1.7670,  0.2900, -1.0881, -1.5555]])
```

```
[10] torch.matmul(W, x)  # results in a [3,2] matrix
```

```
tensor([[ 0.9646, -1.0656],
        [ 4.6092,  3.4132],
        [ 5.3167,  1.5373]])
```

matrix multiply

```
import numpy as np
x = torch.Tensor([[2, 3], [1, 2]])
x = torch.Tensor(np.array([[-1, 1], [2, 4]]))
x = torch.zeros([2, 3], dtype=torch.int32)
```

create tensors from list, numpy.array

```
x = torch.randn((4,2))
z = torch.randn((4,2))
print(x)
print(z)
```

```
tensor([[ 0.0762,  1.5145],
        [-0.4747, -0.9141],
        [ 0.7106,  0.4888],
        [ 0.6959, -0.5305]])
tensor([[-0.1766,  0.6187],
        [ 0.9254, -0.5931],
        [-0.9162,  0.3209],
        [ 0.0216, -0.7116]])
```

```
[13] x * z   # results in a [4,2] matrix
```

```
tensor([[-0.0135,  0.9371],
        [-0.4392,  0.5421],
        [-0.6510,  0.1569],
        [ 0.0151,  0.3775]])
```

Element-wise matrix multiply

大纲

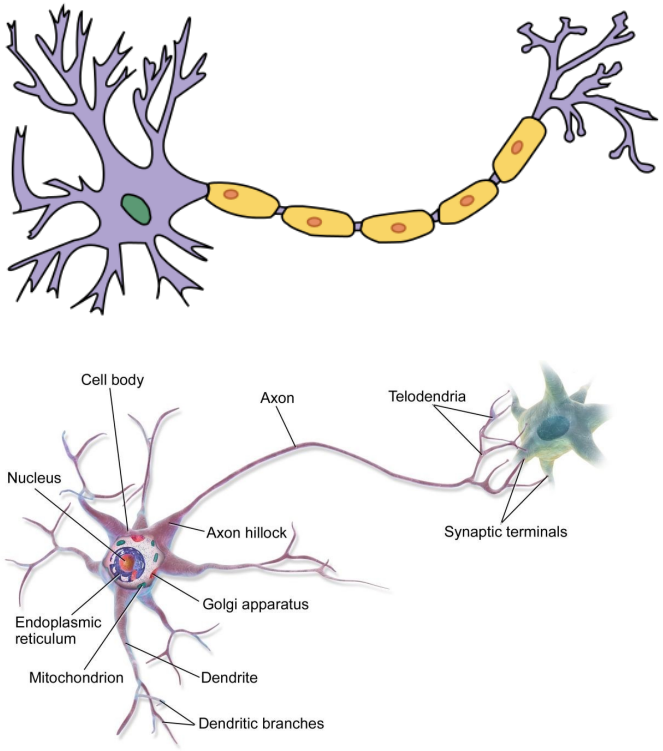
- 张量和数值计算/Tensors and Numerical Computation
- **模型**: 定义神经网络架构
- 前向计算: 计算预测值与损失
- 反向传播/Backpropagation: 计算梯度
- 优化/Optimization: 参数更新
- 训练技巧/Training Tricks

神经网络算法梗概

- 创建一个模型并定义其损失（即构造一个计算图）
- 对于每个样本
  - **前向/Forward计算**：**获得结果**（预测和损失）
  - 如果训练
    - 进行**反向传播/back propagation**
    - **更新**参数

“神经”网络/“Neural” Nets

- 原始动机：大脑中的神经元



By BruceBlaus - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=28761830

- 当前概念: 计算图

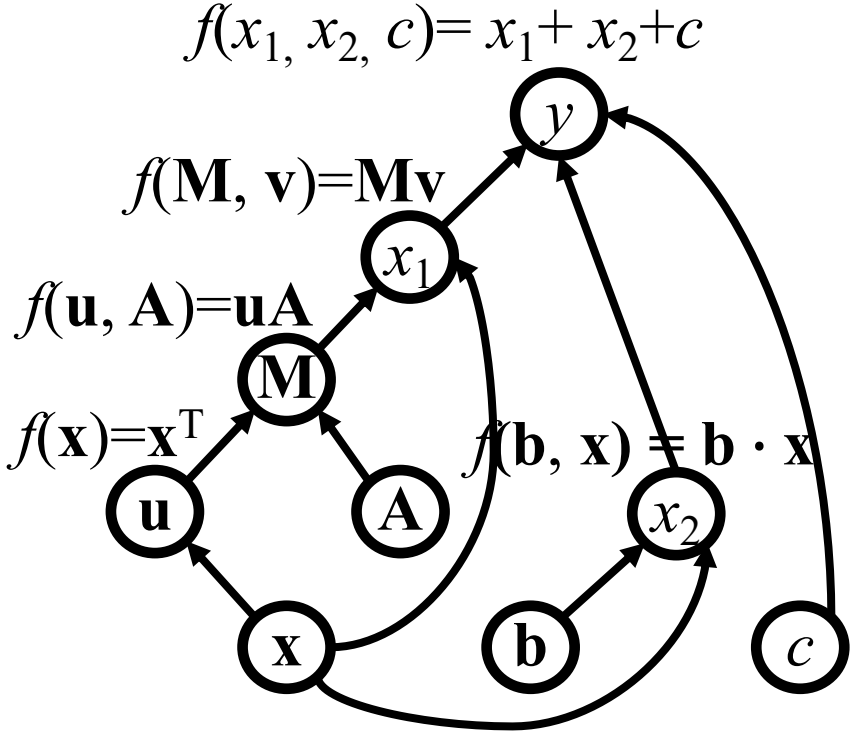
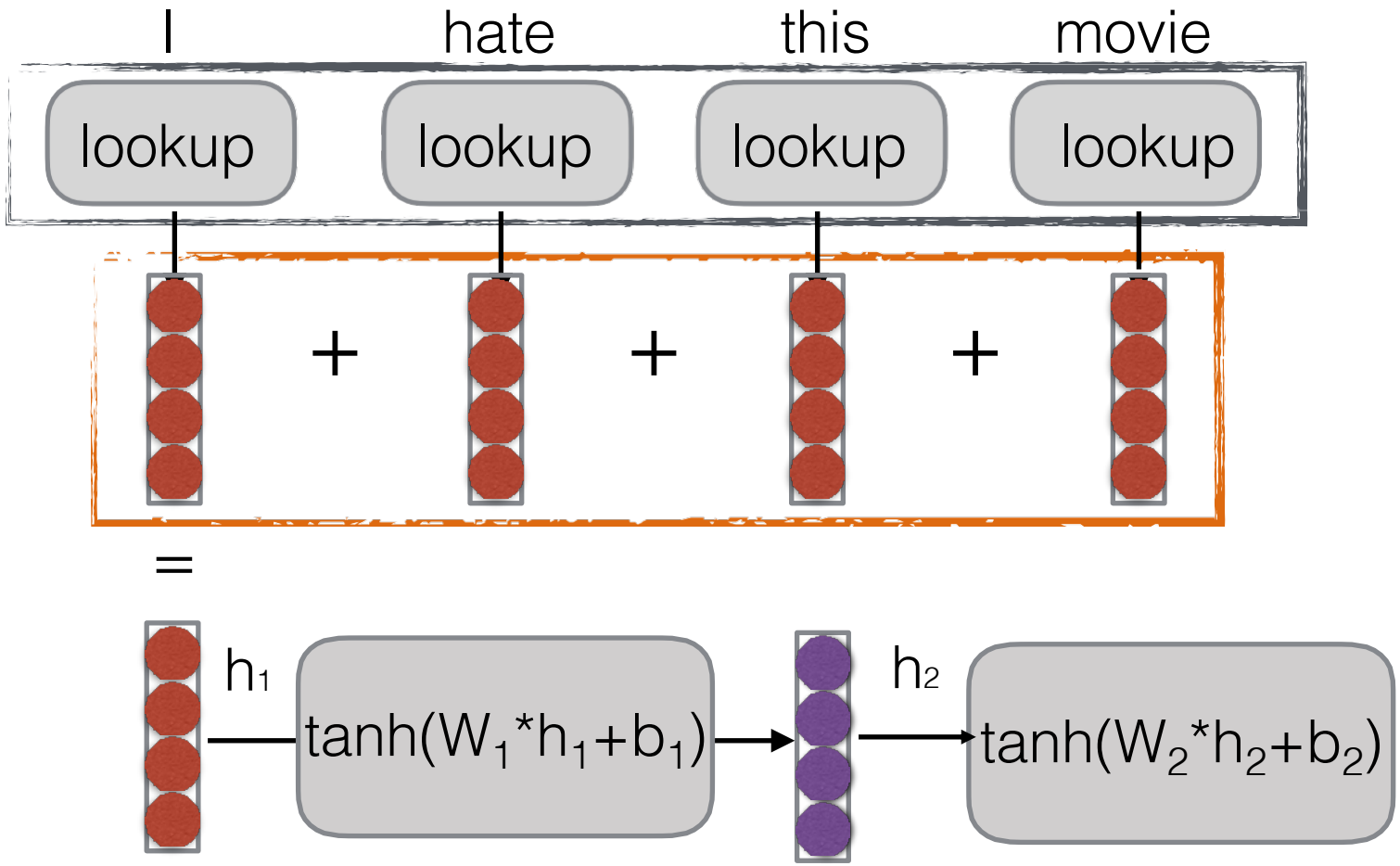
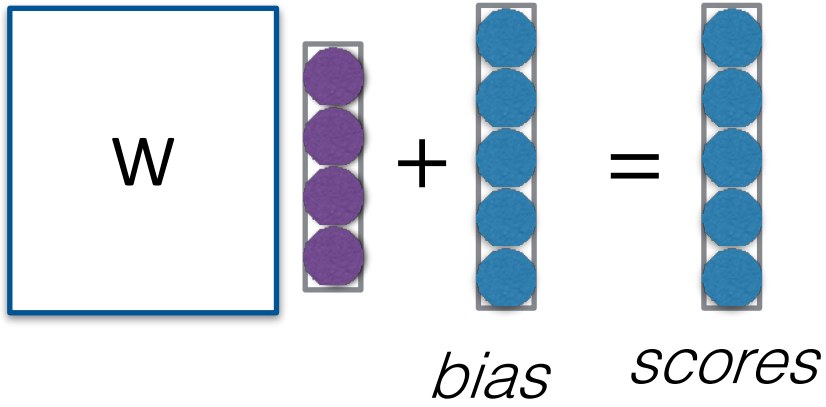


Image credit: Wikipedia

神经网络中的张量



把参数表示为中间值



例子: 模型构建

- 定义模型参数
  - 输入嵌入层的权重矩阵W0
  - 前馈层的权重矩阵W1， W2和偏置b1， b2
  - 其他更多层…

```python
class TryModel(BaseModel):
    def __init__(self, config):
        super(TryModel, self).__init__(config)
        self.define_model_parameters()
        self.init_model_parameters()

        if config.emb_file is not None:
            self.load_pretrained_embeddings(config.emb_file)

    def define_model_parameters(self):
        """
        Define the model's parameters here. e.g., embedding layers, etc.
        """
        raise NotImplementedError
```

参数初始化

- 神经网络可以用不相同的权值学习不同的特征
- 统一初始化/Uniform initialization: 在一定范围内初始化权重，例如$[-\mathrm{v}，\ \mathrm{v}], \mathrm{v} = 0.01$
  - 问题! 根据网络的大小，下游节点的输入可能非常大

- **Glorot (Xavier) Initialization, He Initialization:** 基于矩阵大小进行初始化

Glorot Init: $v = \sqrt{\dfrac{6}{d_{\text{in}} + d_{\text{out}}}}$

Xavier Glorot, Yoshua Bengio: Understanding the difficulty of training deep feedforward neural networks. AISTATS 2010: 249-256

例子: 模型构建

- 使用Glorot或其他方法初始化模型参数：
  - 在训练前为w0， w1, w2, b1， b2等张量在[-v， v]内创建初始随机值

```
def init_model_parameters(self):
    """
    Initialize the model's parameters by uniform sampling from a range [-v, v]
    """
    raise NotImplementedError()
```

损失函数

- 给定一个有标签的样本 $(\boldsymbol{x}, y^*)$ ，使用神经网络 $f(\boldsymbol{x})$ 估计条件概率，并预测其标签为$y = \mathrm{argm}axP_\theta(y|\boldsymbol{x})$:

$$P_\theta(y|\boldsymbol{x}) = \mathrm{Softmax}\big(f(\boldsymbol{x})\big)$$

- 可以通过损失函数计算预测值与真实标签的接近程度
  - 分类任务：交叉熵
    - $L(\boldsymbol{x}, y^*) = -\log P_\theta(y = y^*|\boldsymbol{x})$
  - 回归任务：L1损失、L2损失(均方误差)
    - $L(\boldsymbol{x}, y^*) = \|f(\boldsymbol{x}) - y^*\|_1$
    - $L(\boldsymbol{x}, y^*) = \|f(\boldsymbol{x}) - y^*\|_2$

神经网络算法梗概

- 创建一个模型并定义其损失（即构造一个计算图）
- 对于每个样本
  - **前向/Forward**：**计算结果**（预测和损失）
  - 如果训练
    - 进行**反向传播/back propagation**
    - **更新**参数

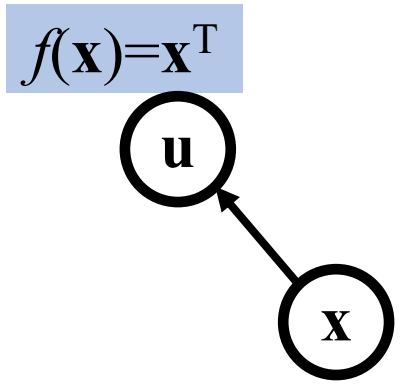
图表示

- 表达式: x
- 节点是一个{张量，矩阵，向量，标量}值
- 图例

图表示

- 边/edge代表函数参数 (即: 数据依赖项)
  - 它们只是指向节点的指针
- 具有传入边的节点
  - 该边的尾节点的函数

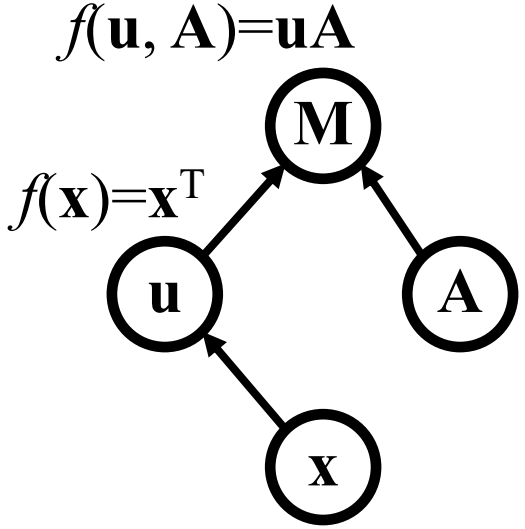
- 图例



图表示

- 表达式: $\mathbf{x}^{\mathrm{T}}\mathbf{A}$
- 函数可以是一元、二元、…、n元(输入的个数)。
  - 通常是一元或二元的

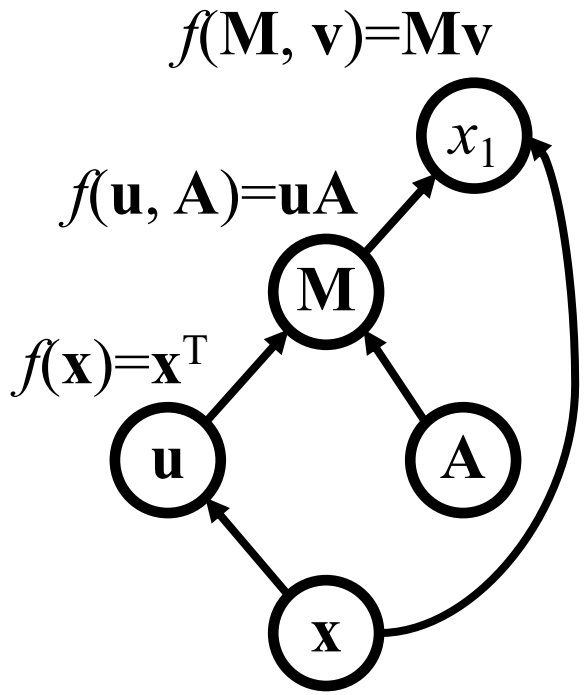
- 图例



图表示

- 表达式: $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$

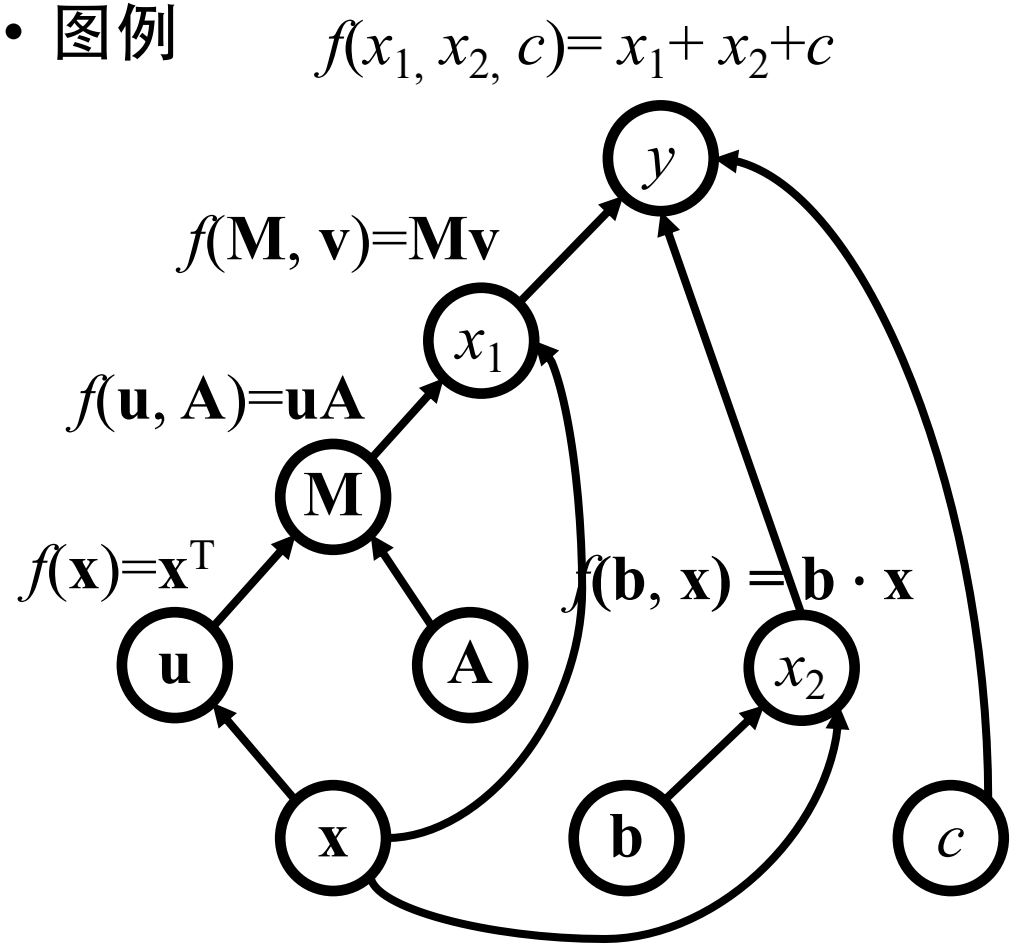
- 图例



图表示

- 表达式: $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$

- 图例

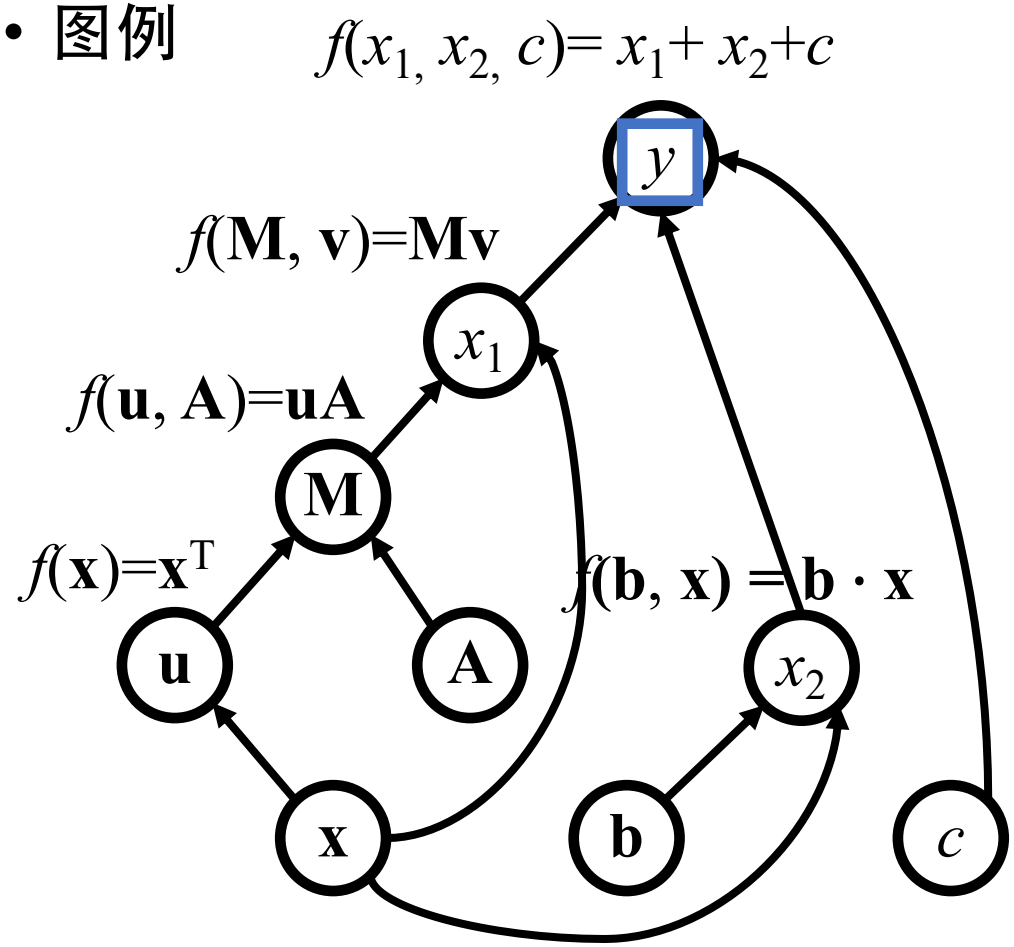


图表示

- 表达式：$y = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$

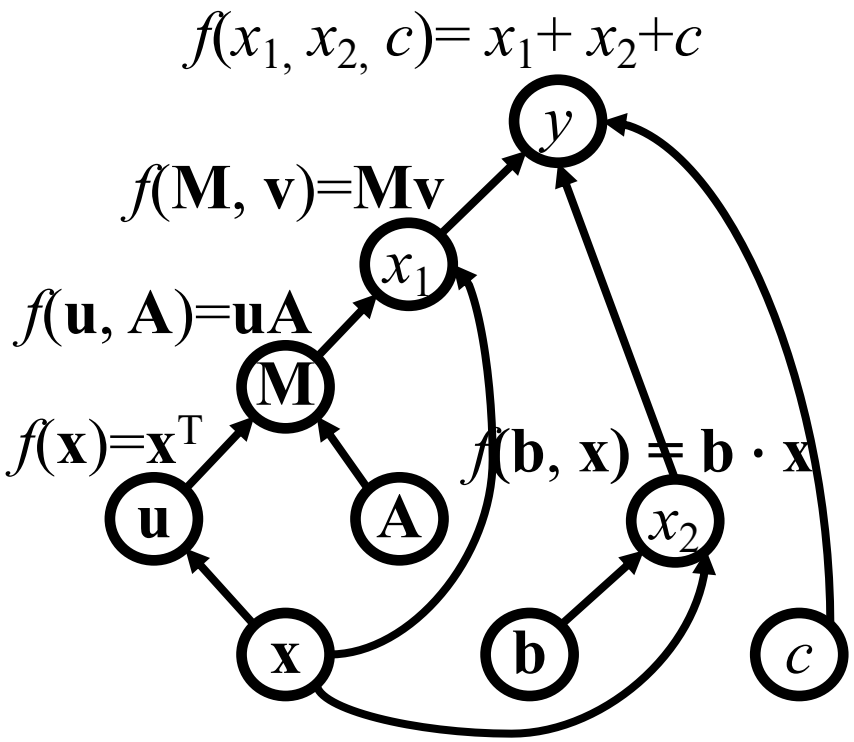
- 注意：计算图通常是有向且无循环

- 图例



图表示

- 表达式: $y = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$




```
# x: 5-dimensional vector
# Model's parameters contain:
#    A: 5x5 matrix
#    b: 5-dimensional vector
#    c: scalar

class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.define_model_parameters()

    def define_model_parameters(self):
        self.A = nn.Parameter(torch.randn((5,5)))
        self.b = nn.Parameter(torch.randn((5)))
        self.c = nn.Parameter(torch.randn((1)))

    def forward(self, x):
        U = x.T
        print('U', U)
        M = torch.matmul(U, self.A)
        print('M', M)
        Mv = torch.matmul(M, x)
        print('Mv', Mv)
        bx = torch.dot(self.b, x)
        print('bx', bx)
        y = Mv + bx + self.c
        return y

x = torch.randn((5))
model = SimpleModel()
y = model(x)
print(y)
```

图表示

- 表达式: $y = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$

- Code: https://github.com/hqyang/nlp-codes/blob/main/01-simpleclassifier/train_NN.ipynb

```
# x: 5-dimensional vector
# Model's parameters contain:
#    A: 5x5 matrix
#    b: 5-dimensional vector
#    c: scalar

class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.define_model_parameters()

    def define_model_parameters(self):
        self.A = nn.Parameter(torch.randn((5,5)))
        self.b = nn.Parameter(torch.randn((5)))
        self.c = nn.Parameter(torch.randn((1)))

    def forward(self, x):
        U = x.T
        print('U', U)
        M = torch.matmul(U, self.A)
        print('M', M)
        Mv = torch.matmul(M, x)
        print('Mv', Mv)
        bx = torch.dot(self.b, x)
        print('bx', bx)
        y = Mv + bx + self.c
        return y

x = torch.randn((5))
model = SimpleModel()
y = model(x)
print(y)
```

Operations 运算

- 前向/Forward: 给定输入，计算预测值/prediction或损失值/loss
- 回归/Regression
  - 预测值/Prediction:
    $$y = f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$
  - 损失值/Loss: $L = (y\text{-}y^*)^2$
- 分类/Classification
  - 激活函数/Activation:
    - S型函数/Sigmoid, 双曲正切函数/Tanh,
    - 线性整流函数/ReLU (Rectified Linear Unit), Leaky ReLU, ELU, Swish, GELU
    - 柔性最大值传输函数/Softmax
  - 损失函数: 交叉熵损失

- 后向/backward: 计算模型参数对损失的导数

$$\frac{\partial L}{\partial \mathbf{A}}, \frac{\partial L}{\partial \mathbf{b}}, \frac{\partial L}{\partial c}$$

大纲

- 张量和数值计算/Tensors and Numerical Computation
- 模型: 定义神经网络架构
- 前向计算: 计算预测值与损失
- **反向传播/Backpropagation: 计算梯度**
- 优化/Optimization: 参数更新
- 训练技巧/Training Tricks

神经网络算法梗概

- 创建一个模型并定义其损失（即构造一个计算图）
- 对于每个样本
  - **前向/Forward**：**计算结果**（预测和损失）
  - 如果训练
    - 进行**反向传播/back propagation**
    - **更新**参数

反向传播/Back Propagation

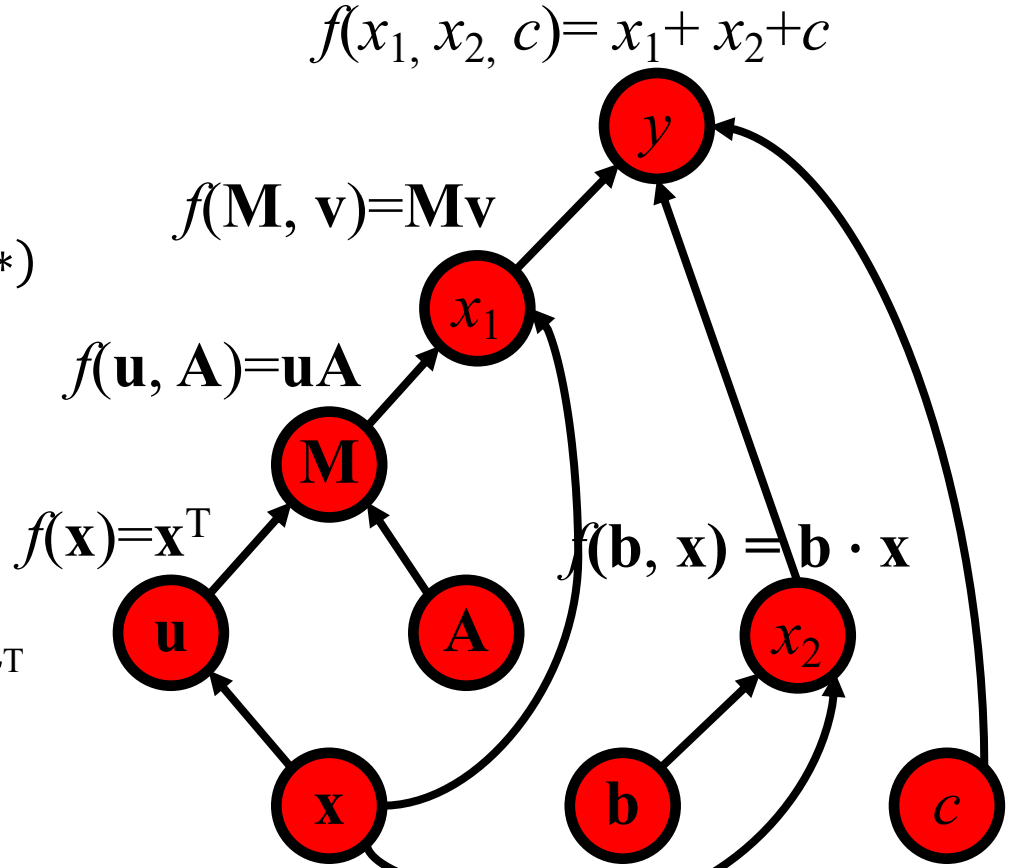
- 模型参数: $\mathbf{A}$, $\mathbf{b}$, c
- 均方误差/mean-square error: $L = (y\text{-}y^*)^2$

$$\frac{\partial L}{\partial y} = 2(y-y^*)$$

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x_1} = 2(y-y^*)$$

$$\frac{\partial L}{\partial \mathbf{M}} = \frac{\partial L}{\partial x_1} \cdot \frac{\partial x_1}{\partial \mathbf{M}} = 2(y-y^*)\boldsymbol{x}$$

$$\frac{\partial L}{\partial \mathbf{A}} = \frac{\partial L}{\partial \mathbf{M}} \cdot \frac{\partial \mathbf{M}}{\partial \mathbf{A}} = 2(y-y^*)\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}$$



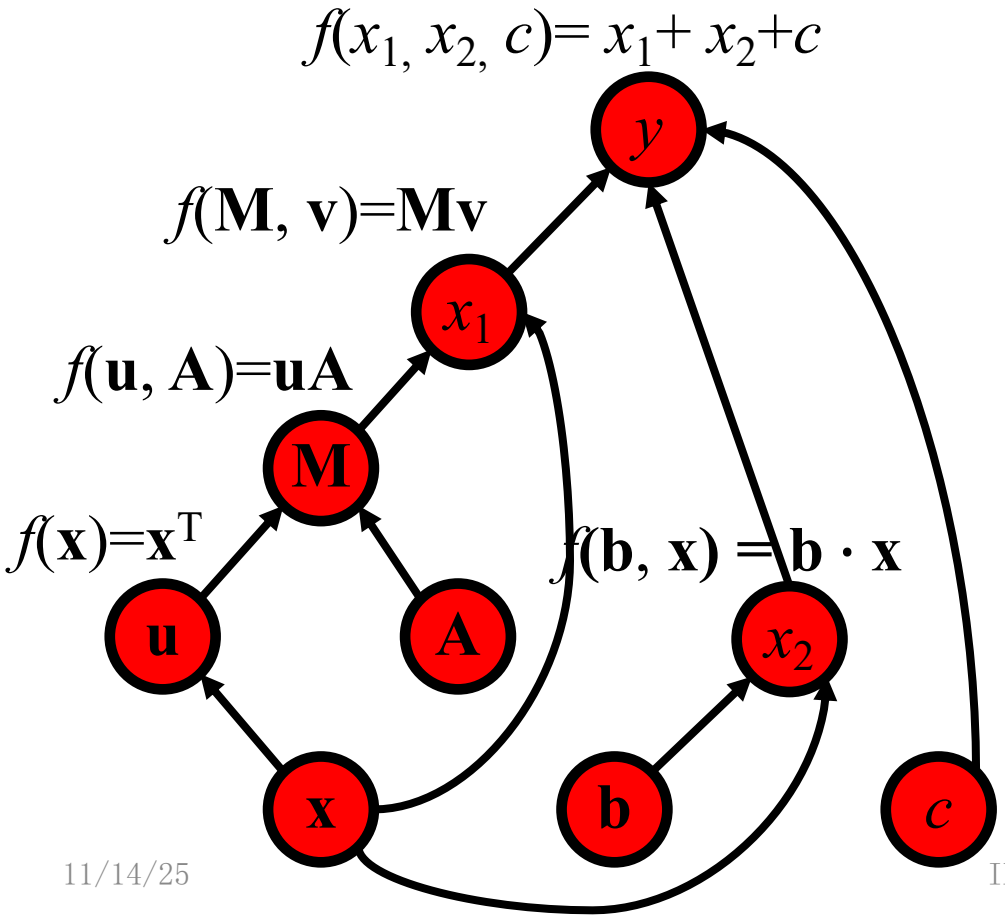
$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x_2} = 2(y-y^*)$$

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial \mathbf{b}} = 2(y-y^*)\boldsymbol{x}$$

$$\frac{\partial L}{\partial \boldsymbol{c}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial \boldsymbol{c}} = 2(y-y^*)$$

反向传播/Back Propagation

- 模型参数: **A**, **b**, c



- 均方误差/mean-square error: $L = (y\text{-}y^*)^2$

```
optimizer = torch.optim.Adagrad(model.parameters(), lr=0.01)
loss_func = nn.MSELoss()

y_true = torch.FloatTensor([1.0]) # ground true
loss = loss_func(y, y_true)
print(loss)
```

```
tensor(34.0528, grad_fn=<MseLossBackward0>)
```

- 使用工具自动计算梯度

```
optimizer.zero_grad()
loss.backward(retain_graph=True)
for name, param in model.named_parameters():
  print(name, param.grad)
```

```
A tensor([[ -0.0313,  -0.0386,  -0.8571,   0.7265,   0.9405],
        [ -0.0386,  -0.0476,  -1.0572,   0.8961,   1.1601],
        [ -0.8571,  -1.0572, -23.4711,  19.8953,  25.7557],
        [  0.7265,   0.8961,  19.8953, -16.8643, -21.8319],
        [  0.9405,   1.1601,  25.7557, -21.8319, -28.2627]])
b tensor([  0.6044,   0.7455,  16.5508, -14.0294, -18.1619])
c tensor([-11.6710])
```

In-class Practice

大纲

- 张量和数值计算/Tensors and Numerical Computation
- 模型: 定义神经网络架构
- 前向计算: 计算预测值与损失
- 反向传播/Backpropagation: 计算梯度
- 优化/Optimization: 参数更新
- 训练技巧/Training Tricks

神经网络算法梗概

- 创建一个模型并定义其损失（即构造一个计算图）
- 对于每个样本
  - **前向/Forward**：**计算结果**（预测和损失）
  - 如果训练
    - 进行**反向传播/back propagation**
    - **更新**参数

优化器更新/ Optimizer Update

- 大多数深度学习工具包通过调用optimizer.step()函数实现参数更新

更新前

```
[59] # Before gradient update
     for name, param in model.named_parameters():
         print(name, param)
```

```
A Parameter containing:
tensor([[-0.2267,  0.6521, -0.8193,  0.7723, -0.6456],
        [ 1.2410, -2.4380, -0.5612, -0.1144, -0.2687],
        [ 0.4792,  0.4543, -1.3530,  1.2934, -0.9943],
        [ 0.7565,  0.9449,  0.2796,  0.4703,  0.2926],
        [ 0.4143,  0.5891,  0.4370,  0.6060,  0.0161]]
b Parameter containing:
tensor([ 0.3592,  0.3455, -0.2517, -0.5678, -0.6016],
c Parameter containing:
tensor([-1.5490], requires_grad=True)
```

更新后

```
[61] # Gradient update:
     optimizer.step()
     # After gradient update
     for name, param in model.named_parameters():
         print(name, param)
```

```
A Parameter containing:
tensor([[-0.2167,  0.6621, -0.8093,  0.7623, -0.6556],
        [ 1.2510, -2.4280, -0.5512, -0.1244, -0.2787],
        [ 0.4892,  0.4643, -1.3430,  1.2834, -1.0043],
        [ 0.7465,  0.9349,  0.2696,  0.4803,  0.3026],
        [ 0.4043,  0.5791,  0.4270,  0.6160,  0.0261]]
b Parameter containing:
tensor([ 0.3492,  0.3355, -0.2617, -0.5578, -0.5916],
c Parameter containing:
tensor([-1.5390], requires_grad=True)
```

许多不同优化器

- SGD:
  - 仅依据当前梯度更新权重
- Momentum/动量:
  - 结合梯度的累积平均值更新权重
- Adagrad:
  - 通过梯度方差自适应调整，对高方差权重进行降权 /downweighting
- Adam:
  - 结合梯度的累积平均值更新权重，同时利用梯度方差的累积平均值自适应调整权重

标准SGD

- 标准随机梯度下降(Stochastic Gradient Descent, SGD)法

$$g_t = \nabla_\theta L(\theta_{t-1})$$

损失梯度

$$\theta_t = \theta_{t-1} - \eta g_t$$

学习率

- 大量变种，见[Ruder 2016]

Sebastian Ruder: An overview of gradient descent optimization algorithms. CoRR abs/1609.04747 (2016)

SGD With Momentum

- 记住过去的梯度

$$v_t = \gamma v_{t-1} + \eta g_t$$

当前动量　　之前动量

动量保留参数

$$\theta_t = \theta_{t-1} - v_t$$

- 直觉: 防止突然变化造成的不稳定

Sebastian Ruder: An overview of gradient descent optimization algorithms. CoRR abs/1609.04747 (2016)

Adagrad

- 基于梯度累积方差自适应降低学习率

$$G_t = G_{t-1} + g_t \odot g_t$$

当前梯度平方

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

微小常数

- 直觉: 频繁更新的参数（如常用词嵌入）应该少更新
- 问题：学习率持续下降，训练可能会停滞——通过在AdaDelta和RMSProp中使用滚动平均来修复

Sebastian Ruder: An overview of gradient descent optimization algorithms. CoRR abs/1609.04747 (2016)

Adam

- 大多数选择的优化器
- 考虑梯度和动量的滚动平均值

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

动量

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t$$

梯度滚动平均值

- 在训练早期纠正偏差(因为是$E[v_t]$方差的有偏估计: $E[v_t] = E[g_t^2](1-\beta_2^t)$)

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- 最终更新

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Sebastian Ruder: An overview of gradient descent optimization algorithms. CoRR abs/1609.04747 (2016)

大纲

- 张量和数值计算/Tensors and Numerical Computation
- 模型: 定义神经网络架构
- 前向计算: 计算预测值与损失
- 反向传播/Backpropagation: 计算梯度
- 优化/Optimization: 参数更新
- 训练技巧/Training Tricks

重排训练数据/ Shuffling the Training Data

- 随机梯度法每次更新一点参数
  - 比如: 训练数据最后出现50次“i love this sentence so much!”

- 为了训练正确，我们应该随机打乱每次的数据顺序

防止过拟合的简单方法

- 神经网络有大量的参数：我们希望防止过拟合

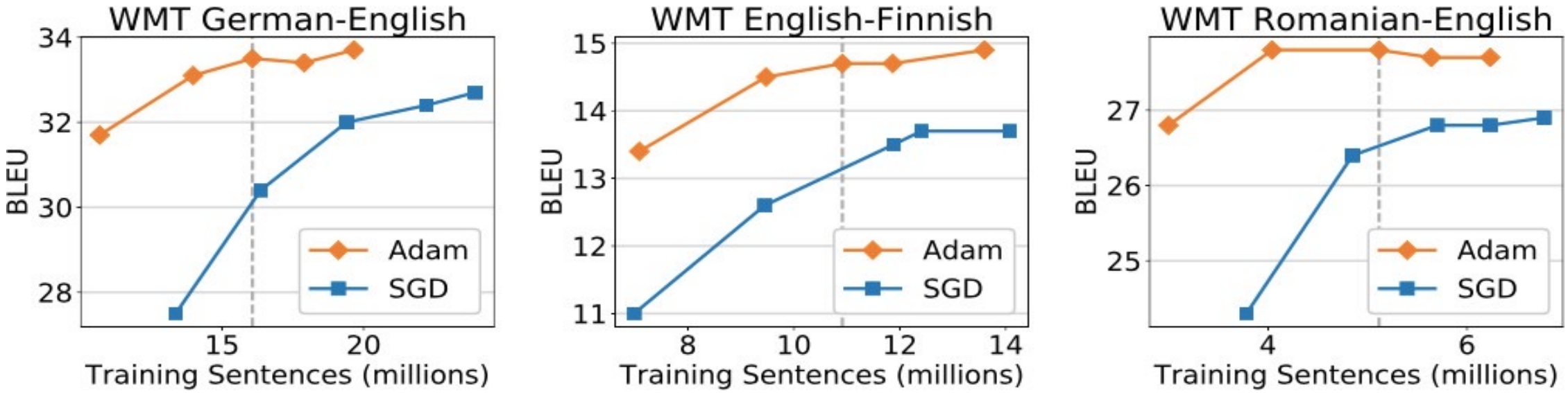
- **提早停止/Early stop**:
  - 监控保留的开发集的性能，并在性能恶化时停止训练
- **学习率衰减/Learning decay**：
  - 随着训练的继续，逐渐降低学习率，或者
  - 在开发集的表现停滞时降低学习率
- **耐心**:
  - 学习可能不稳定，所以有时要等开发集性能变差n次后，才停止训练或降低学习率

应该选哪个优化器呢？

- Adam通常收敛得很快，也很稳定
- 简单的SGD在泛化方面通常做得不错(见Wilson et al. 2017)
- 有时候需要使用学习率衰减(见Denkowski & Neubig 2017机器翻译的结果)

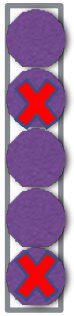
A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht: The Marginal Value of Adaptive Gradient Methods in Machine Learning. NIPS 2017

M. J. Denkowski, G. Neubig: Stronger Baselines for Trustable Results in Neural Machine Translation. NMT@ACL 2017



Dropout

- 神经网络有很多参数，容易出现过拟合
- Dropout：仅在**训练**时以p概率随机将隐藏层中的节点归零



- 因为训练/测试时，节点的数量不同的，需要进行缩放/scaling:
  - **标准Dropout**：在测试时按1-p缩放
  - **反向Dropout**：在训练时按1/(1-p)进行缩放

- 其他选择：使用DropConnect ，而不是将神经网络中的权重归零

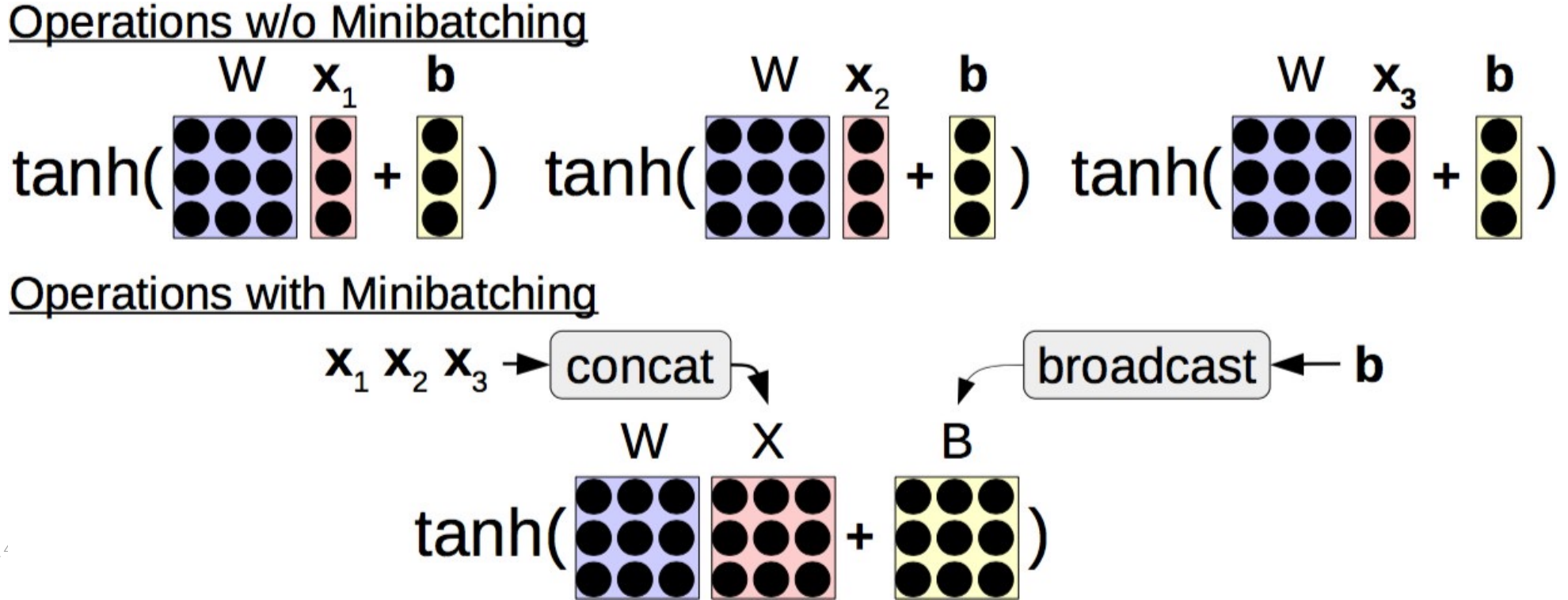
Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus: Regularization of Neural Networks using DropConnect. ICML (3) 2013

大纲

- 张量和数值计算/Tensors and Numerical Computation
- 模型: 定义神经网络架构
- 前向计算: 计算预测值与损失
- 反向传播/Backpropagation: 计算梯度
- 优化/Optimization: 参数更新
- 训练技巧/Training Tricks
  - 训练效率技巧：批处理操作

训练效率技巧：小批量/Mini-batching

- 在现代硬件上，进行10次样本大小为1的操作比进行1次样本大小为10的操作慢得多
- 小批处理将较小的操作组合成一个大操作



小批量训练过程

- 将类似的操作组合在一起（例如单个单词的损失计算）并一起执行它们
  - 在前馈语言模型的情况下，句子中的每个单词预测都可以批处理
  - 对于递归神经网络等，更为复杂

- 如何实现取决于工具包
  - 大多数工具包都要求添加一个表示批大小的额外维度
  - 一些工具包有显式的工具来实现小批量训练

变长序列的小批量训练

- 在句子的末尾添加填充词元(padding tokens)，以确保小批量训练的所有句子都具有相同的长度

- 例子: 假设序列最大长度为5
  - X1 = “ I love this interesting movie” # 5个真实单词
  - X2 = “ great movie [pad] [pad] [pad]” # 2个真实单词
  - X3 = “This movie is bad [pad] ” # 4个真实单词

- 在计算平均单词嵌入时，我们应该
  1) 将填充词元归零
  2) 将单词上的嵌入求和
  3) 将总和除以真实单词的数量

苦涩的教训/The Bitter Lesson

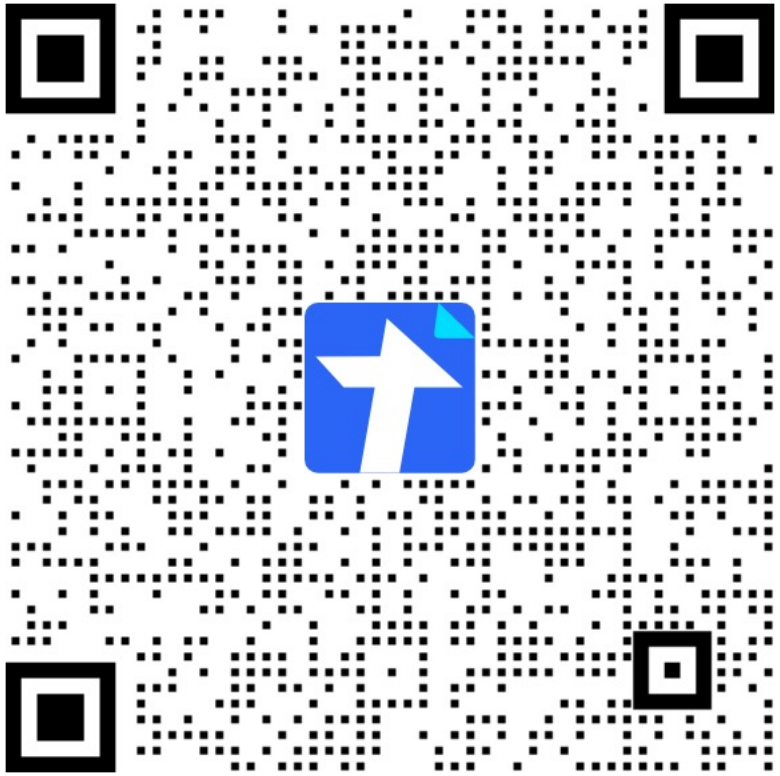
理查德・萨顿 (Richard S. Sutton), 2019.3.13

- … 70 years of AI research…
  - general methods that leverage computation are ultimately the most effective, and by a large margin.
  - The ultimate reason … is Moore‘s law, or ...

1. 能充分发挥算力价值的**通用方法**，最终…最有效
   - 原因: 摩尔定律 (单位算力成本持续呈指数级下降)
   - 例子:
     - 1997年: 深蓝击败国际象棋冠军
     - 2015年: AlphaGo
     - 语音识别、计算机视觉
     - 2023年: ChatGPT
2. 人类意识所承载的内容极为复杂，且…难以简化
   - 只需为系统搭建能自主发现并捕捉这种复杂特征的元方法即可

一句话总结

- 张量
- 神经网络: 网络结构、前向计算、后向反馈/backpropagation
- 优化器: SDG, momentum, Adagrad, ADAM等
- 训练技巧:
  - 重排数据，
  - 防止过拟合: early stop, learning rate decay, drop out
  - mini-batching
- 苦涩的教训
- 课外阅读
  - [SLP3] Ch. 7



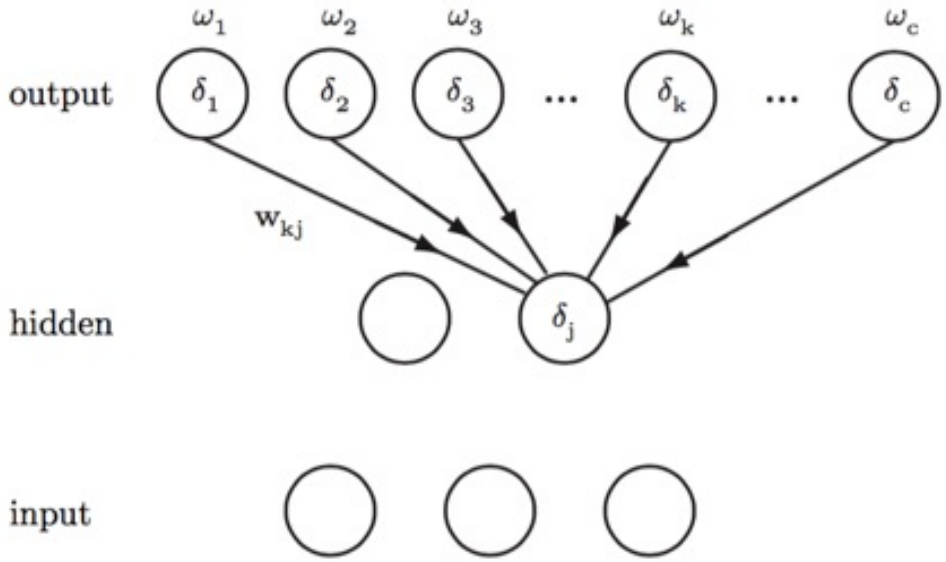
附录: 神经网络推导/Neural Networks Derivatives

Constructing a Network Topology

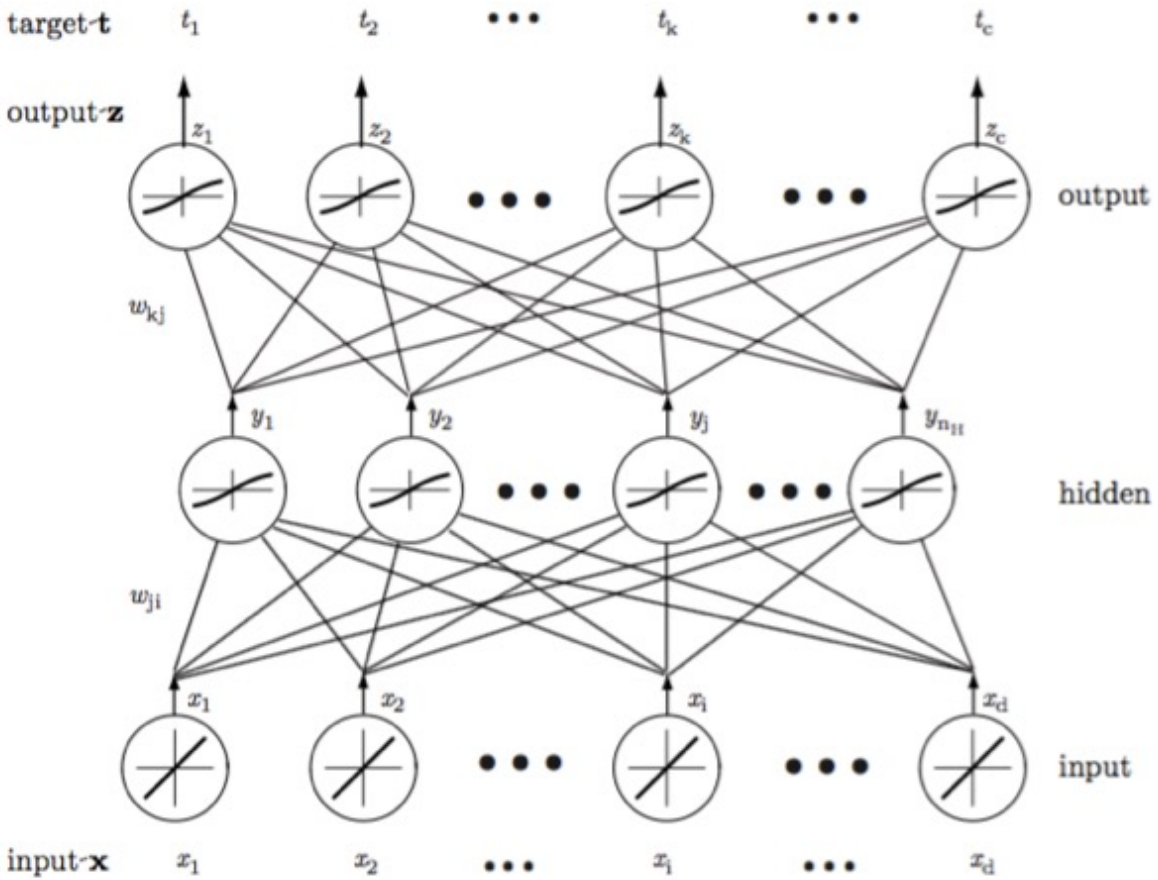
- First decide the **network topology** (fully-connected network):
  - # of units in the input layer,
  - # of hidden layers (if > 1),
  - # of units in each hidden layer, and
  - # of units in the output layer
- **Input:** weight initialization; feature normalization
- **Output**:
  - if binary/multiple classification, applying one-hot encoding

Backpropagation

- Two phases
  - Forward pass phase: computes 'functional signal', feed forward propagation of input pattern signals through network
  - Backward pass phase: computes 'error signal', ***propagates*** the error ***backwards*** through network starting at output units (where the error is the difference between actual and desired output values)



Backpropagation (BP)



$$z_k = g(net_k)$$

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} \equiv \mathbf{w}_k^T \mathbf{y}$$

$$y_j = g(net_j)$$

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{w}_j^T \mathbf{x}$$

Input data: $\{(\mathbf{x}_i, t_i)\}$, $i=1, \ldots, n$

$\mathbf{x}_i \in \mathcal{X} = \mathbb{R}^d$, $t_i \in \mathcal{Y} = \{1, \ldots, c\}$

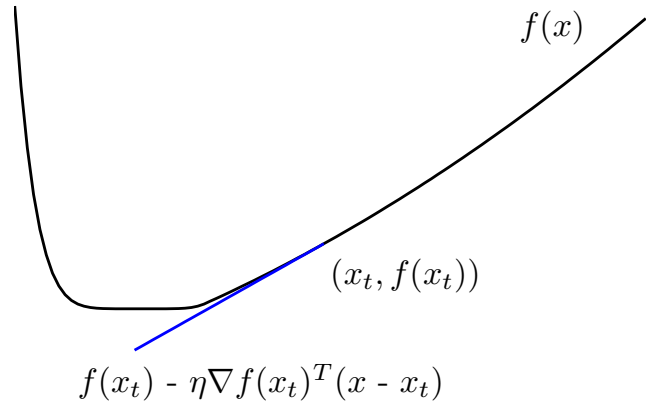
Stochastic vs. Batch

Algorithm 1 (Stochastic backpropagation)

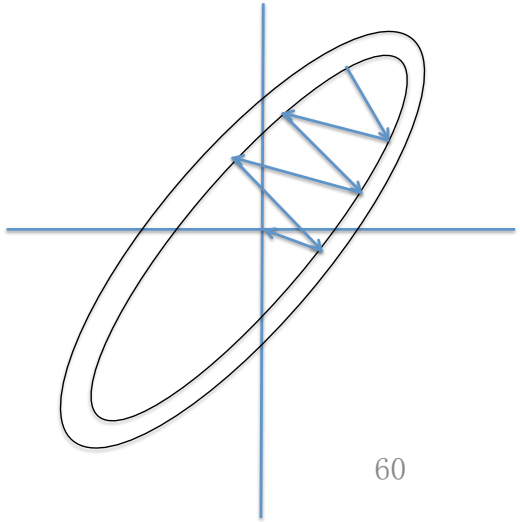
```
1  begin initialize  network topology (# hidden units), w, criterion θ, η, m ← 0
2     do m ← m + 1
3        x^m ← randomly chosen pattern
4        w_ij ← w_ij + ηδ_j x_i;   w_jk ← w_jk + ηδ_k y_j
5     until ∇J(w) < θ
6  return w
7  end
```

Algorithm 2 (Batch backpropagation)

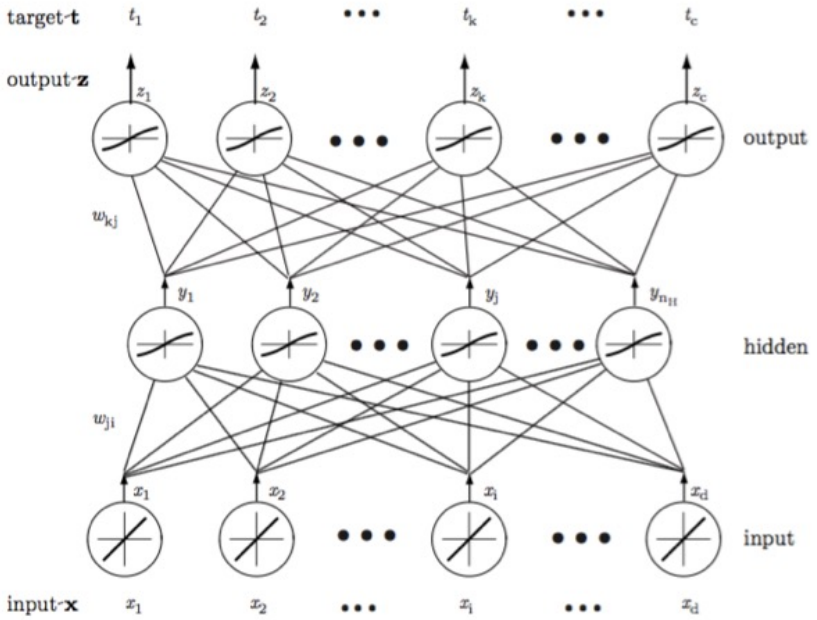
```
1  begin initialize  network topology (# hidden units), w, criterion θ, η, r ← 0
2     do r ← r + 1 (increment epoch)
3        m ← 0; Δw_ij ← 0; Δw_jk ← 0
4        do m ← m + 1
5           x^m ← select pattern
6           Δw_ij ← Δw_ij + ηδ_j x_i;   Δw_jk ← Δw_jk + ηδ_k y_j
7        until m = n
8        w_ij ← w_ij + Δw_ij;   w_jk ← w_jk + Δw_jk
9     until ∇J(w) < θ
10 return w
11 end
```



$$x_{t+1} = x_t - \eta \nabla f(x_t)$$



BP: Updating Rule



$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} \equiv \mathbf{w}_k^T \mathbf{y}$$

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{w}_j^T \mathbf{x}$$

$$y_j = g(net_j) \qquad z_k = g(net_k)$$

- Least-Mean-Square (LMS)

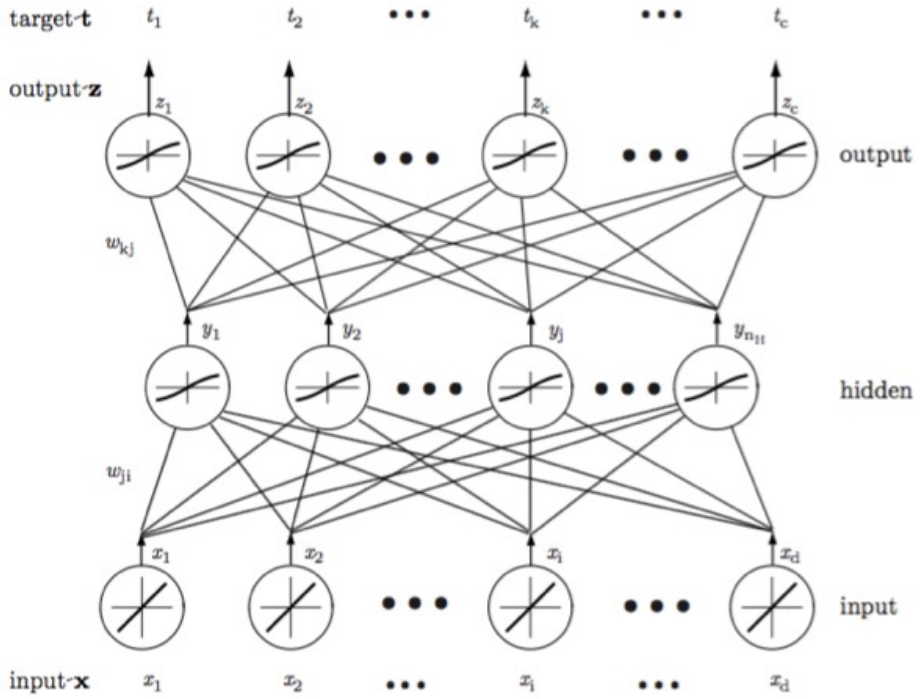
$$J(\mathbf{w}) \equiv \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|\mathbf{t} - \mathbf{z}\|^2$$

- Updating rule

$$\mathbf{w}_{m+1} = \mathbf{w}_m + \triangle \mathbf{w}_m$$

$$\triangle \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

BP: Hidden-to-output Weights



$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} \equiv \mathbf{w}_k^T \mathbf{y}$$

$$z_k = g(net_k)$$

$$\mathbf{w}_{m+1} = \mathbf{w}_m + \triangle \mathbf{w}_m$$

- Derivative on weight $\mathbf{w}_{kj}$

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

- Specifically,

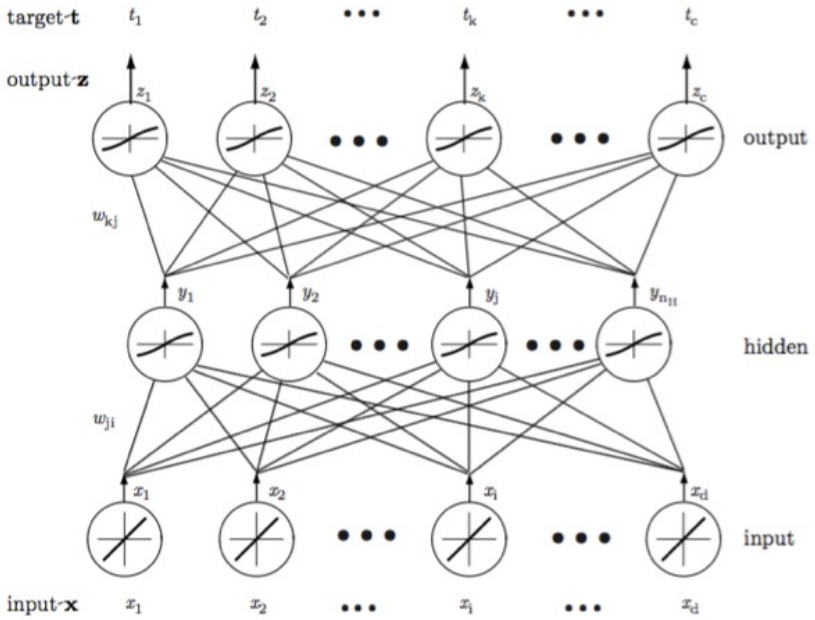
$$\delta_k = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k)g'(net_k)$$

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

- Updating rule

$$\triangle w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) g'(net_k) y_j$$

BP: Input-to-Hidden Weight



$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} \equiv \mathbf{w}_k^T \mathbf{y}$$

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{w}_j^T \mathbf{x}$$

$$y_j = g(net_j) \qquad z_k = g(net_k)$$

$$\mathbf{w}_{m+1} = \mathbf{w}_m + \triangle \mathbf{w}_m$$

- Derivative on weight $\mathbf{w}_{ji}$

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

- Specifically,

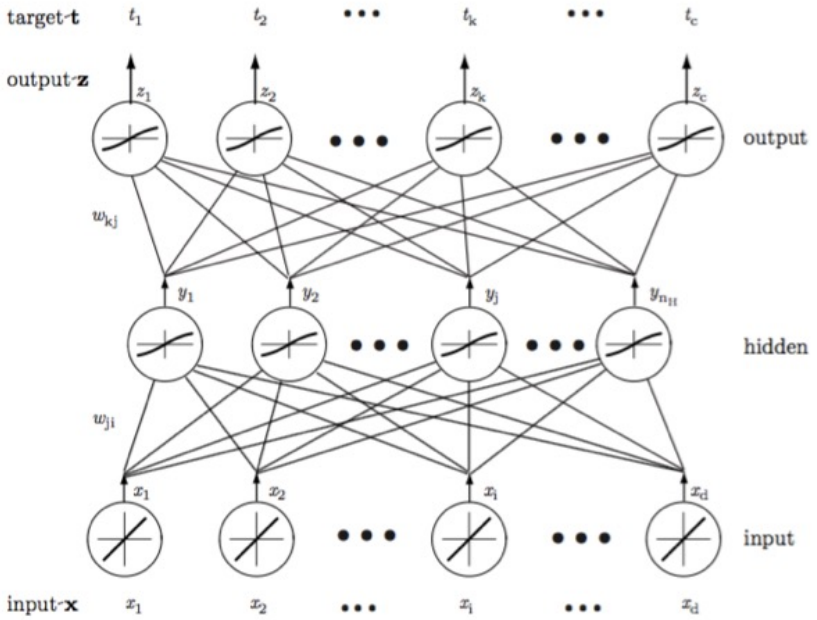
$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)g'(net_k)w_{kj}
\end{aligned}$$

δ_k

$$\delta_j \equiv g'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

$$\triangle w_{ji} = \eta \delta_j x_i = \eta \left[\sum_{k=1}^{c} w_{kj}\delta_k\right] g'(net_j) x_i$$

BP: Formulation Summary



$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} \equiv \mathbf{w}_k^T \mathbf{y}$$

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{w}_j^T \mathbf{x}$$

$$y_j = g(net_j) \qquad z_k = g(net_k)$$

$$\mathbf{w}_{m+1} = \mathbf{w}_m + \triangle \mathbf{w}_m$$

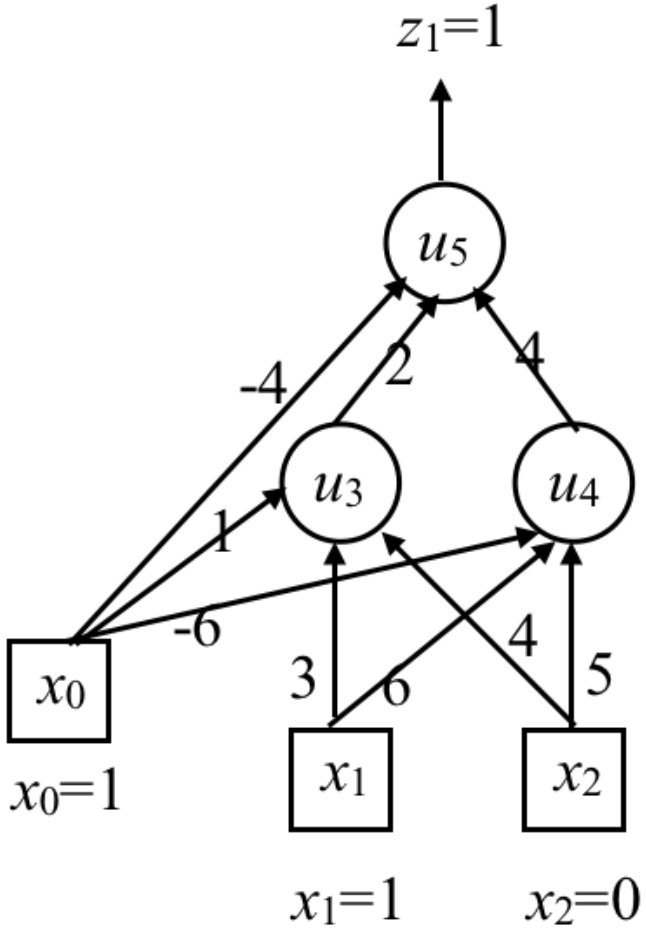
- Derivative on weight $\mathbf{w}_{ji}$

$$\triangle w_{ji} = \eta \delta_j x_i = \eta \left[\sum_{k=1}^{c} w_{kj} \delta_k \right] g'(net_j) x_i$$

- Derivative on weight $\mathbf{w}_{kj}$

$$\triangle w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) g'(net_k) y_j$$

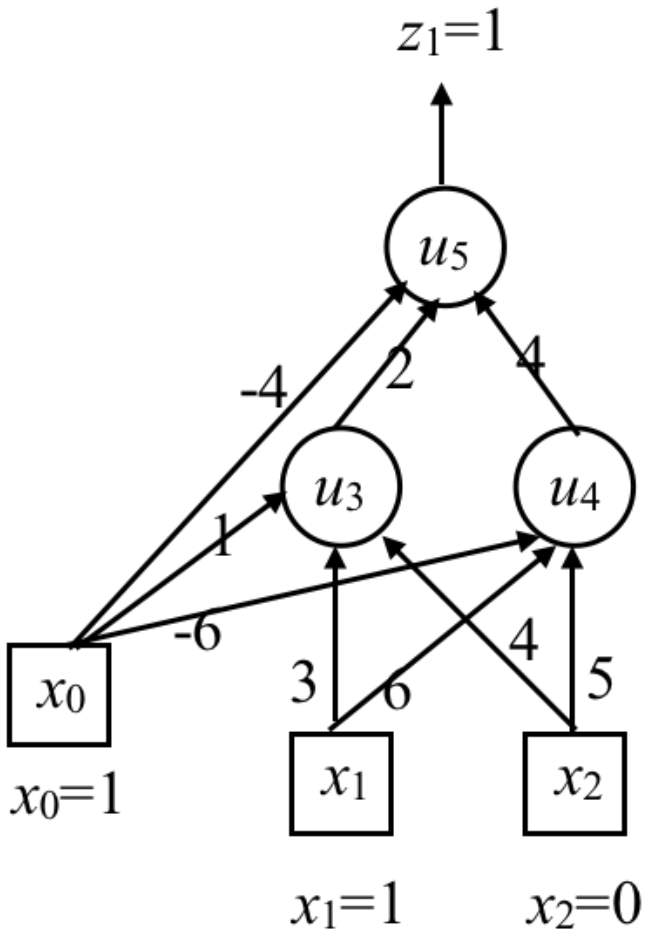
In-class Practice: MLP/BP



假设你用带偏置的2-2-1网络及均方损失解决异或XOR问题:

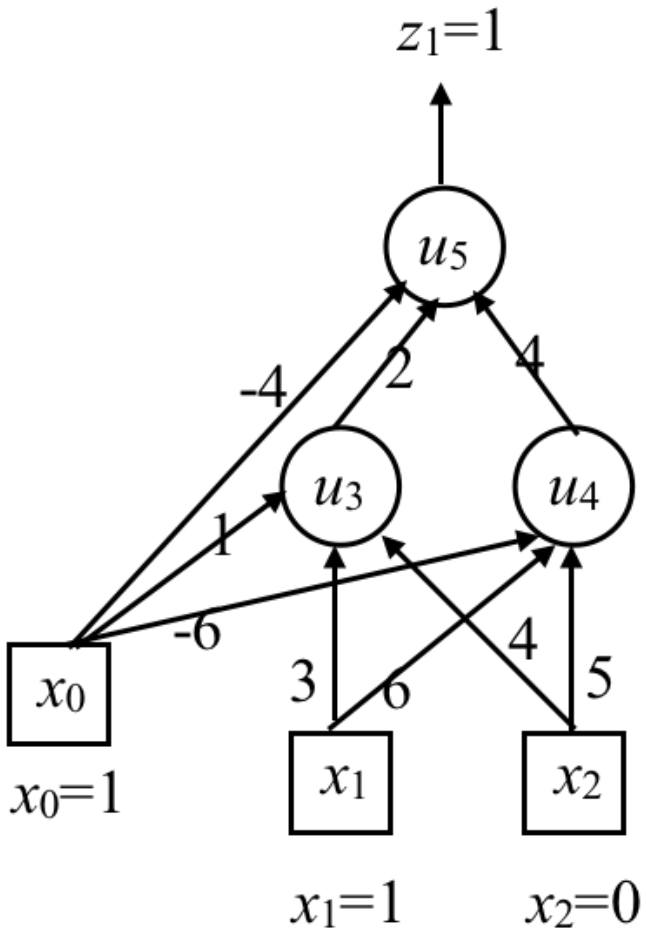
- 初始参数如下:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 数据
  - 输入特征: (x_1=1,x_2=0)
  - 目标输出: z_1=1
- 考虑以下两个激活函数
  1) ReLU: $g(x) = \max\{x, 0\}$,
  2) Sigmoid : $g(x) = 1/(1+e^{-x})$
- 计算下述值
  - u_3, u_4, u_5的前向计算值
  - 对应的损失值
  - 权重更新梯度(假设学习率$\eta = 1$)

Solution to In-class Practice: MLP/BP



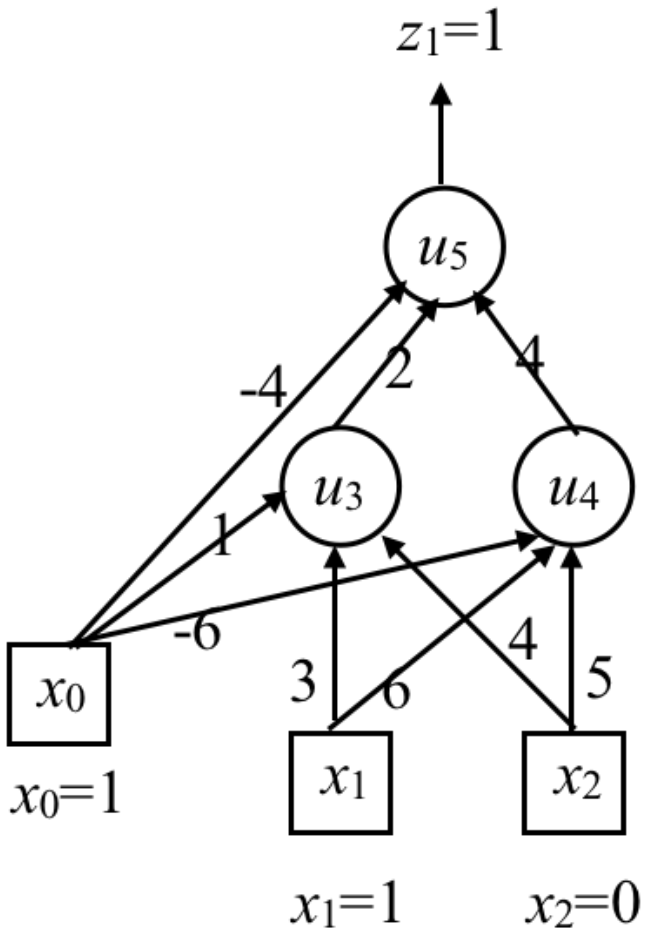
- 初始参数:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 数据
  - 输入特征: (x_1=1,x_2=0)
  - 目标输出: z_1=1
- 使用上述参数及给定数据，当使用ReLU激活函数，可得:
  - $net_3 = 1\times3 + 0\times4 + 1 = 4.\text{So } u_3 = g(net_3) = 4$
  - $net_4 = 1\times6 + 0\times5 - 6 = 0.\text{ So } u_4 = g(net_4) = 0$
  - $net_5 = 4\times2 + 0\times4 - 4 = 4.\text{ So } u_5 = g(net_5) = 4$
  - Loss: $L = \frac{1}{2}(z_1 - u_5)^2 = \frac{1}{2}(1-4)^2 = 4.5$

Solution to In-class Practice: MLP/BP



- 初始参数:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 数据
  - 输入特征: (x_1=1,x_2=0)
  - 目标输出: z_1=1
- 前向计算可得: $u_3 = 4, u_4 = 0, u_5 = 4$
- 权重更新值(假设学习率$\eta = 1$)
  - 输出层权重梯度:
    - $\delta_5 = -(z_1 - u_5) \cdot g'(net_5) = -(1-4)\times 1 = 3$
    - $\Delta w_{05} = -\eta \cdot \delta_5 \cdot x_0 = -1 \times 3 \times 1 = -3,$
    - $\Delta w_{35} = -\eta \cdot \delta_5 \cdot u_3 = -1 \times 3 \times 4 = -12,$
    - $\Delta w_{45} = -\eta \cdot \delta_5 \cdot u_4 = -1 \times 3 \times 0 = 0$

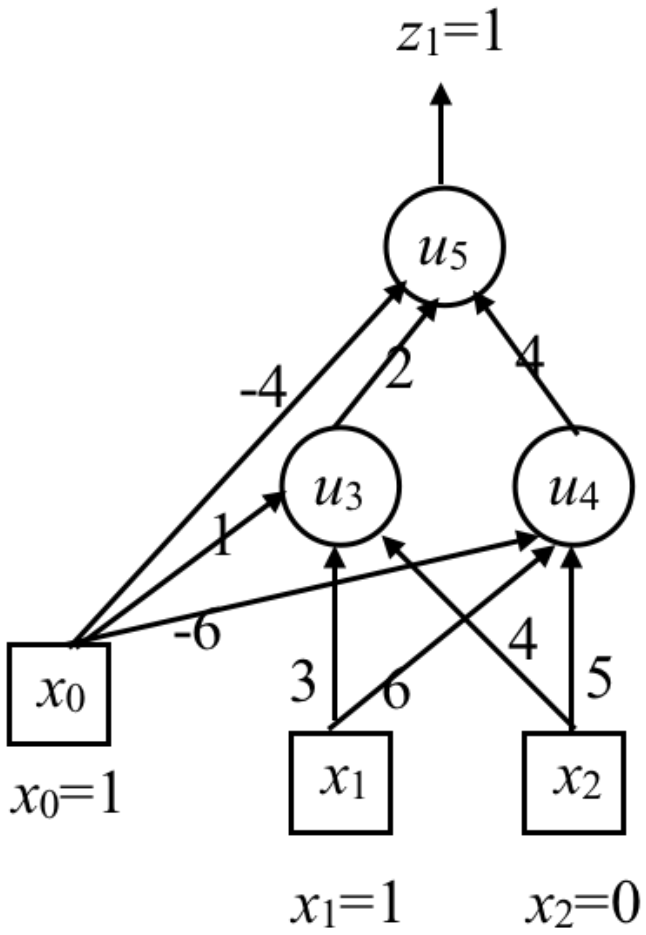
Solution to In-class Practice: MLP/BP



- 初始参数:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 前向计算: $u_3 = 4, u_4 = 0, u_5 = 4$
- 权重更新值(假设学习率$\eta = 1$)
  - 输出层权重: $\delta_5 = 3$
  - 输入层权重梯度:
    - $\delta_3 = \delta_5 \cdot w_{35} \cdot g'(net_3) = 3\times2\times1 = 6,$
    - $\delta_4 = \delta_5 \cdot w_{45} \cdot g'(net_5) = 3\times4\times0 = 0$
    - $\Delta w_{03} = -\eta \cdot \delta_3 \cdot x_0 = -1 \times6 \times1 = -6,$
    - $\Delta w_{13} = -\eta \cdot \delta_3 \cdot x_1 = -1 \times6 \times1 = -6,$
    - $\Delta w_{23} = -\eta \cdot \delta_3 \cdot x_2 = -1 \times6 \times0 = 0$
    - $\Delta w_{04} = -\eta \cdot \delta_4 \cdot x_0 = -1 \times0 \times1 = 0, \Delta w_{14} = -\eta \cdot \delta_4 \cdot x_1 = 0,$ $\Delta w_{24} = -\eta \cdot \delta_4 \cdot x_2 = 0$

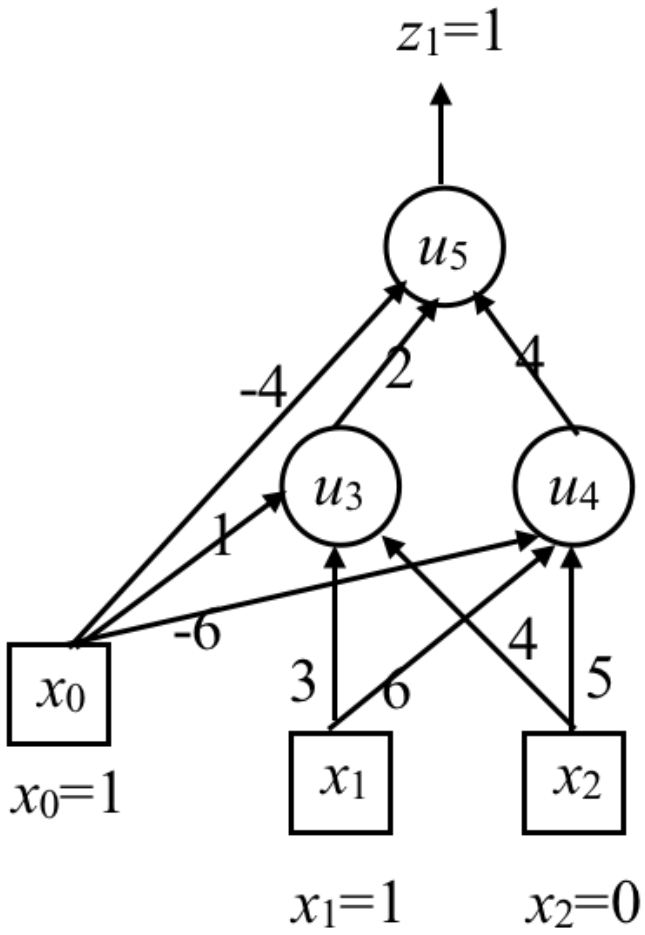
- 数据
  - 输入特征: $(x_1$=1,x_2=0)
  - 目标输出: z_1=1

Solution to In-class Practice: MLP/BP



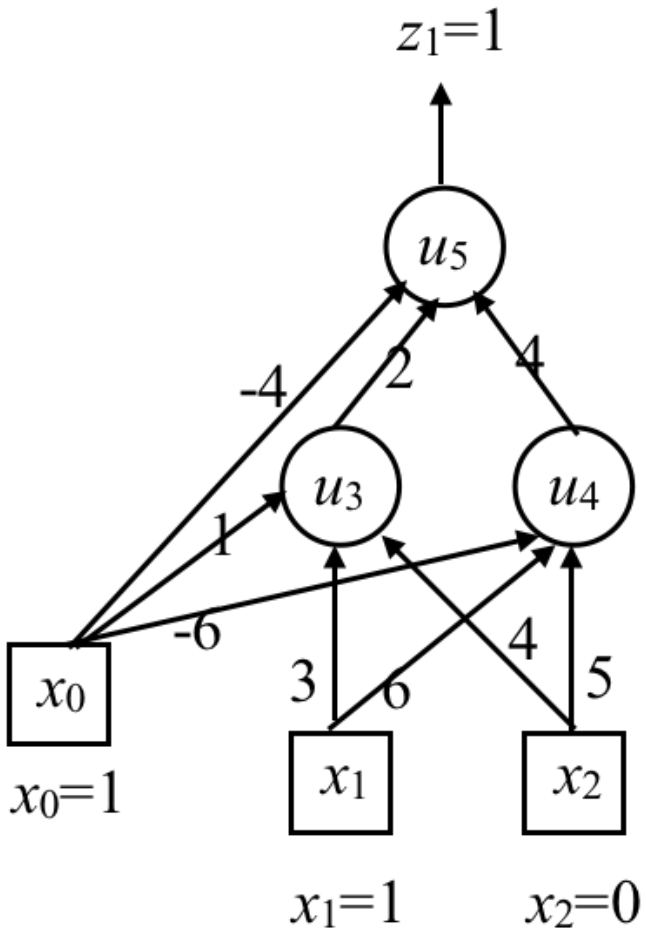
- 初始参数:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 数据
  - 输入特征: (x_1=1,x_2=0)
  - 目标输出: z_1=1
- 使用上述参数及给定数据，当使用Sigmoid激活函数，可得:
  - $net_3 = 1\times3 + 0\times4 + 1 = 4. So\ u_3 = g(net_3) = \frac{1}{1+e^{-4}} \approx 0.9820$
  - $net_4 = 1\times6 + 0\times5 - 6 = 0\ . So\ u_4 = g(net_4) = \frac{1}{1+e^{-0}} = 0.5$
  - $net_5 \approx 0.98\times2 + 0.5\times4 - 4 = -0.04. So\ u_5 = g(net_5) \approx \frac{1}{1+e^{-0.04}} = 0.491$
  - Loss: $L = \frac{1}{2}(z_1 - u_5)^2 = \frac{1}{2}(1-0.491)^2 = 0.1295$

Solution to In-class Practice: MLP/BP



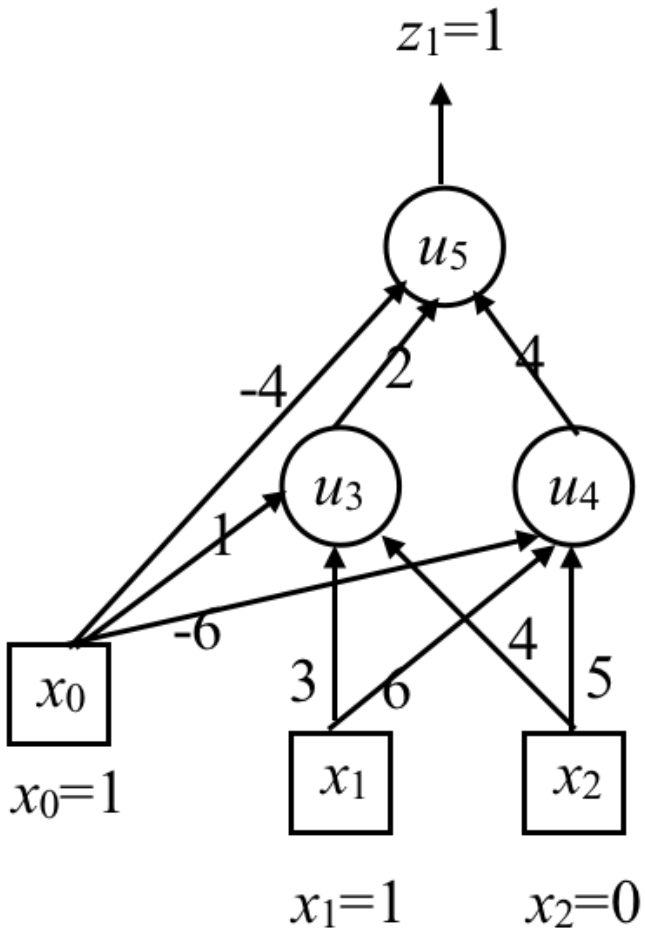
- 初始参数:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 数据
  - 输入特征: (x_1=1,x_2=0)
  - 目标输出: z_1=1
- 前向计算: $u_3 \approx 0.9820$, $u_4 = 0.5$, $u_5 \approx 0.4910$
- 权重更新值(假设学习率$\eta = 1$)
- 输出层权重: (Sigmoid导数: $g'(x) = g(x)(1 - g(x))$
  - $g'(net_5) = u_5 \times (1 - u_5) \approx 0.491 \times (1 - 0.491) \approx 0.2499$
  - $\delta_5 = -(z_1 - u_5) \cdot g'(net_5) \approx -(1 - 0.491) \times 0.2499 = -0.1275$
  - $\Delta w_{05} = -\eta \cdot \delta_5 \cdot x_0 \approx -1 \times (-0.1275) \times 1 = 0.1275,$
  - $\Delta w_{35} = -\eta \cdot \delta_5 \cdot u_3 \approx -1 \times (-0.1275) \times 0.9820 \approx 0.1252,$
  - $\Delta w_{45} = -\eta \cdot \delta_5 \cdot u_4 \approx -1 \times (-0.1275) \times 0.5 \approx 0.0638$

Solution to In-class Practice: MLP/BP



- 初始参数:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 数据
  - 输入特征: (x_1=1,x_2=0)
  - 目标输出: z_1=1
- 前向计算: $u_3 \approx 0.9820$, $u_4 = 0.5$, $u_5 \approx 0.4910$
- 权重更新值(假设学习率$\eta = 1$)
- 输出层权重: $\delta_5 \approx -0.1275$
- 输入层权重梯度:
  - $g'(net_3) = u_3 \times (1 - u_3) \approx 0.9820 \times (1 - 0.9820) \approx 0.0177$; $g'(net_4) = u_4 \times (1 - u_4) = 0.5 \times (1 - 0.5) = 0.25$
  - $\delta_3 = \delta_5 \cdot w_{35} \cdot g'(net_3) \approx -0.1275 \times 2 \times 0.0177 \approx -0.0045$,
  - $\delta_4 = \delta_5 \cdot w_{45} \cdot g'(net_4) \approx -0.1275 \times 4 \times 0.25 = -0.1275$
  - $\Delta w_{03} = -\eta \cdot \delta_3 \cdot x_0 \approx -1 \times (-0.0045) \times 1 = 0.0045$,
  - $\Delta w_{13} = -\eta \cdot \delta_3 \cdot x_1 \approx -1 \times (-0.0045) \times 1 = -0.0045$,
  - $\Delta w_{23} = -\eta \cdot \delta_3 \cdot x_2 \approx -1 \times (-0.0045) \times 0 = 0$

Solution to In-class Practice: MLP/BP



- 初始参数:
  - w_{03}=1, w_{04}=-6, w_{05}=-4
  - w_{13}=3, w_{14}=6, w_{23}=4, w_{24}=5
  - w_{35}=2, w_{45}=4
- 数据
  - 输入特征: (x_1=1,x_2=0)
  - 目标输出: z_1=1
- 前向计算: $u_3 \approx 0.9820$, $u_4 = 0.5$, $u_5 \approx 0.4910$
- 权重更新值(假设学习率$\eta = 1$)
- 输出层权重: $\delta_5 \approx -0.1275$
- 输入层权重梯度:
  - $g'(net_3) \approx 0.0177$, $g'(net_4) = 0.25$
  - $\delta_3 \approx -0.0045$, $\delta_4 \approx -0.1275$
  - $\Delta w_{04} = -\eta \cdot \delta_4 \cdot x_0 \approx -1 \times(-0.1275) \times 1 = 0.1275$,
  - $\Delta w_{14} = -\eta \cdot \delta_4 \cdot x_1 \approx -1 \times(-0.1275) \times 1 = 0.1275$,
  - $\Delta w_{24} = -\eta \cdot \delta_4 \cdot x_2 \approx -1 \times(-0.1275) \times 0 = 0$