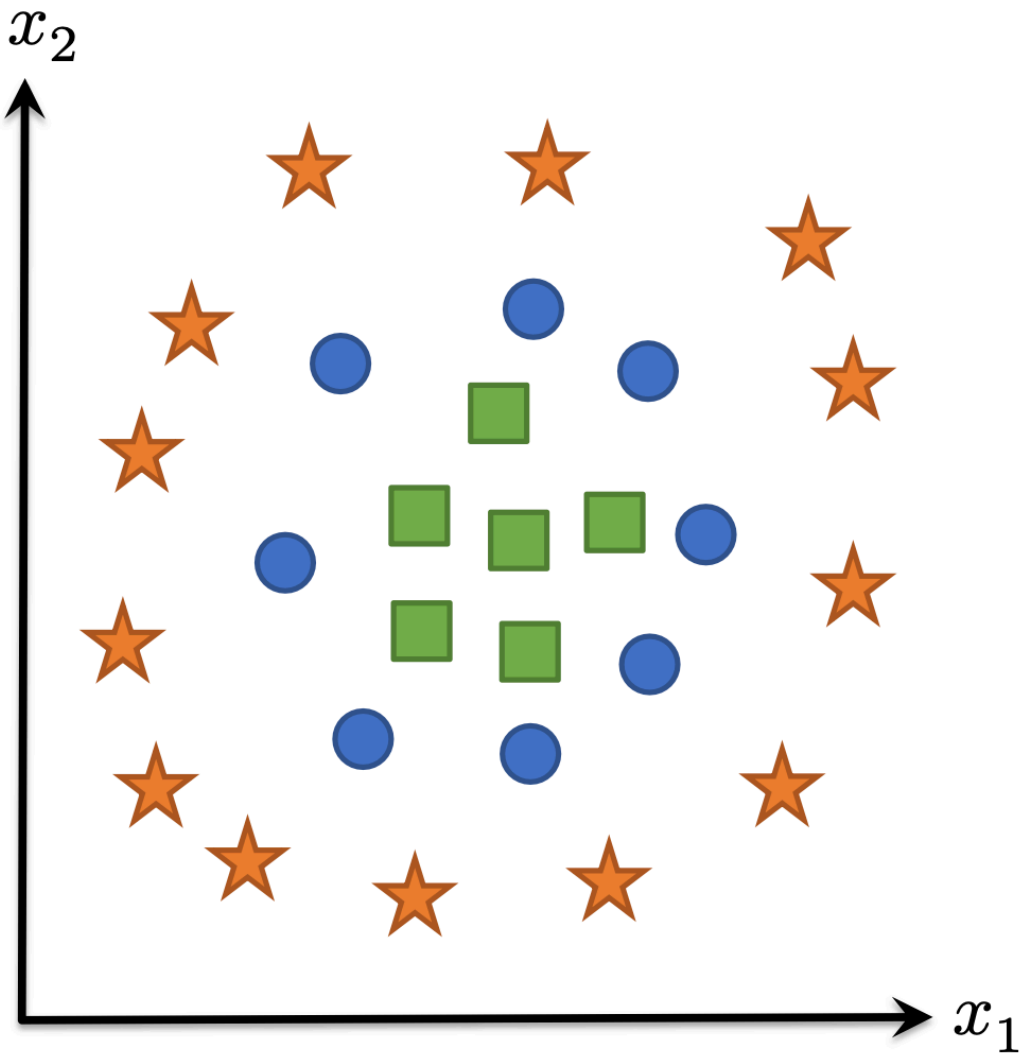
Manual Neural Networks / Backprop

Trouble with Linear Hypothesis Classes

基于线性假设集的分类器实际上相当于将输入划分为 k 个线性区域，每个区域对应一个类别。

但数据很有可能无法被划分为若干个线性区域，因此需要非线性的划分方法。



解决方法：获取输入 x 的特征，再将特征输入到线性分类器中

$$h_\theta(x) = \theta^T \phi(x)$$
$$\theta \in \mathbb{R}^{d \times k}, \phi: \mathbb{R}^n \to \mathbb{R}^d$$

ϕ 被称为特征函数，将输入 x 映射到特征向量 $\phi(x)$

建立特征函数 ϕ 的方法：

1. 老方法：根据对应用场景的理解，人工设置特征函数
2. 新方法：从数据中学习到相应的特征函数（神经网络兴起的主要原因）

但是，ϕ 不能是线性函数，否则仍然是线性假设集

Nonlinear Features

要建立非线性的特征函数，可以在线性化特征的基础上套一个非线性函数

$$\phi(x) = \sigma(W^T x) \tag{2}$$

其中 $W \in R^{n \times d}, \sigma : R^d \to R^d$

进而

$$h_\theta(x) = \theta^T \sigma(W^T x) \tag{3}$$

有更多的参数可以优化

Neural Networks

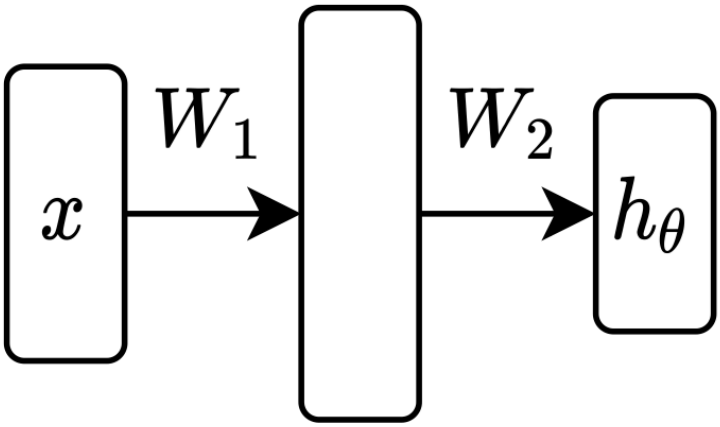
神经网络是一类特殊的假设集，由多个参数化且可微的函数（也称为“层”）组成以产生输出。

Two Layer Neural Network

最简单的神经网络

$$\begin{aligned} h_\theta(x) &= W_2^T \sigma(W_1^T x) \\ \theta &= \{W_1 \in R^{n \times d}, W_2 \in R^{d \times k}\} \end{aligned} \tag{4}$$

其中 θ 包含网络中所有参数



Batch matrix 形式：

$$h_\theta(X) = \sigma(XW_1)W_2 \tag{5}$$

Universal Function Approximation

定理 (1D case): 给定一个光滑函数 $f: R \to R$, 闭区域 $D \subset R, \epsilon > 0$, 可以构建一个浅层神经网络 (one-hidden-layer neural network) $\hat{f}$ 使得

$$\max_{x \in D} \left| f(x) - \hat{f}(x) \right| \leq \epsilon \tag{6}$$

即，一个浅层（两层）神经网络可以拟合闭区域上的任意一个光滑函数。这样的性质称为 Universal Function Approximation.

证明：在 f 上采样若干个点，构建一个神经网络 $\hat{f}$ 使得经过这些点。神经网络可以形成 linear spline, 例如使用 ReLu 的网络。采样点足够多的时候，就能拟合该函数。

这个性质很好，但是实际中需要很多采样点，相应地，在 batch matrix 中，W_1 的维度要和采样点数量一样大，这不现实。

例如一个使用 ReLu 的浅层神经网络：

$$\hat{f}(x) = \sum_{i=1}^{d} \pm \max\{0, w_i x + b_i\} \tag{7}$$

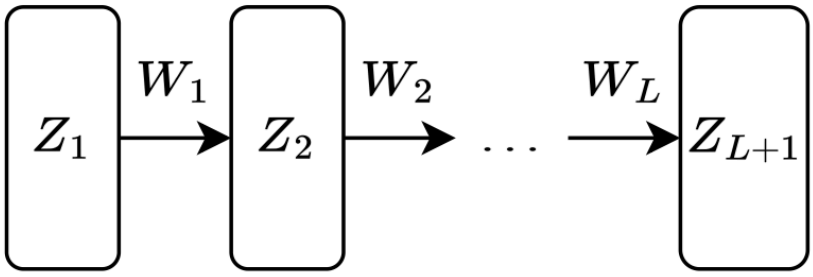
Fully-connected Deep Networks

更一般地，考虑 L 层神经网络，也被称为 Multi-layer perceptron (MLP), feedforward network, fully-connected network.

Batch 形式如下：

$$\begin{aligned}
&Z_{i+1} = \sigma_i(Z_i W_i), i = 1, \ldots, L \\
&Z_1 = X, \\
&h_\theta(X) = Z_{L+1} \\
&\left[Z_i \in \mathbb{R}^{m \times n_i},\ W_i \in \mathbb{R}^{n_i \times n_{i+1}}\right]
\end{aligned}$$

其中 $\theta = \{W_1, \ldots, W_L\}$，中间特征 $Z_i (i > 1)$ 被称为 layer / activation / neuron / hidden layer.



（也可以在其中加入 bias, $Z_i W_i + b_i$）

Backpropagation

如何用梯度下降法训练神经网络

Loss function: cross-entropy

Optimization: SGD

Gradient of a Two-layer Network

以 batch matrix 形式推导：

$$\nabla_{\{W_1,W_2\}} l_{ce}(\sigma(XW_1)W_2, y) \tag{8}$$

对于 W_2 而言，梯度计算如下：

$$\begin{aligned}\frac{\partial l_{ce}(\sigma(XW_1)W_2, y)}{\partial W_2} &= \frac{\partial l_{ce}(\sigma(XW_1)W_2, y)}{\partial \sigma(XW_1)W_2} \cdot \frac{\partial \sigma(XW_1)W_2}{\partial W_2} \\ &= (S - I_y) \cdot \sigma(XW_1)\end{aligned} \tag{9}$$

其中 $S = normalize(exp(\sigma(XW_1)W_2))$

在 matching sizes 后梯度为

$$\nabla_{W_2} l_{ce}(\sigma(XW_1)W_2, y) = \sigma(XW_1)^T(S - I_y) \tag{10}$$

对于 W_1 而言，梯度计算同理：

$$\begin{aligned}\frac{\partial \ell_{ce}(\sigma(XW_1)W_2, y)}{\partial W_1} &= \frac{\partial \ell_{ce}(\sigma(XW_1)W_2, y)}{\partial \sigma(XW_1)W_2} \cdot \frac{\partial \sigma(XW_1)W_2}{\partial \sigma(XW_1)} \cdot \frac{\partial \sigma(XW_1)}{\partial XW_1} \cdot \frac{\partial XW_1}{\partial W_1} \\ &= (S - I_y) \cdot (W_2) \cdot \left(\sigma'(XW_1)\right) \cdot (X)\end{aligned}$$

在 matching sizes 后梯度为

$$\nabla_{W_1} l_{ce}(\sigma(XW_1)W_2, y) = X^T((S - I_y)W_2^T \circ \sigma'(XW_1)) \tag{11}$$

其中 ∘ 代表 elementwise 相乘。

Backpropagation in General

考虑一般情况，对于 L-layer 神经网络：

$$Z_{i+1} = \sigma_i(Z_i W_i),\ i = 1, \ldots, L \tag{12}$$

那么概念上的梯度计算为：

$$\frac{\partial \ell(Z_{L+1}, y)}{\partial W_i} = \underbrace{\frac{\partial \ell}{\partial Z_{L+1}} \cdot \frac{\partial Z_{L+1}}{\partial Z_L} \cdot \frac{\partial Z_{L-1}}{\partial Z_{L-2}} \cdot \ldots \cdot \frac{\partial Z_{i+2}}{\partial Z_{i+1}}}_{G_{i+1} = \frac{\partial \ell(Z_{L+1}, y)}{\partial Z_{i+1}}} \cdot \frac{\partial Z_{i+1}}{\partial W_i}$$

其中红色部分在对 $W_j(j<i)$ 求梯度时也会出现，记为 G_{i+1}

由此可推出 G_i 概念上的迭代公式：

$$G_i = G_{i+1} \cdot \frac{\partial Z_{i+1}}{\partial Z_i} = G_{i+1} \cdot \frac{\partial \sigma_i(Z_i W_i)}{\partial Z_i W_i} \cdot \frac{\partial Z_i W_i}{\partial Z_i} = G_{i+1} \cdot \sigma_i'(Z_i W_i) \cdot W_i \tag{13}$$

又由于 $G_i = \frac{\partial l(Z_{L+1}, y)}{\partial Z_i} = \nabla_{Z_i} l(Z_{L+1}, y) \in R^{m \times n_i}$

故经过 matching sizes 可得迭代公式：

$$G_i = (G_{i+1} \circ \sigma_i'(Z_i W_i)) W_i^T \tag{14}$$

回到所需的梯度计算，由于 $\nabla_{W_i} l(Z_{L+1}, y) \in R^{n_i \times n_{i+1}}$, 故有：

$$\nabla_{W_i} l(Z_{L+1}, y) = Z_i^T (G_{i+1} \circ \sigma_i'(Z_i W_i)) \tag{15}$$

Backpropagation: Forward and Backward Passes

实际上，整个梯度计算过程分为 forward pass 和 backward pass 两部分，其中 forward pass 计算实际的输出，backward pass 根据 loss 反向传递计算梯度所必需的 G_i

1. Initialize: $Z_1 = X$
   Iterate: $Z_{i+1} = \sigma_i(Z_i W_i), \quad i = 1, \dots, L$ } Forward pass
2. Initialize: $G_{L+1} = \nabla_{Z_{L+1}} \ell(Z_{L+1}, y) = S - I_y$
   Iterate: $G_i = \left(G_{i+1} \circ \sigma_i'(Z_i W_i)\right) W_i^T, \quad i = L, \dots, 1$ } Backward pass

经过两次 pass, 最终可以计算所需的梯度

$$\nabla_{W_i} l(Z_{L+1}, y) = Z_i^T (G_{i+1} \circ \sigma_i'(Z_i W_i)) \tag{16}$$

从而进行学习过程。

事实上，反向传播就是链式法则 + 对中间结果的缓存，并且需要对 Z_i 和 G_i 都进行缓存。因此反向传播的计算效率高，但内存开销大。

进一步观察，每一层都需要做 vector Jacobian product 操作，即反向传播得到的 G_i 乘上该层的微分 $\frac{\partial Z_{i+1}}{\partial W_i}$. 因此可以进一步模块化，即接下来将介绍的自微分。