人工神经网络

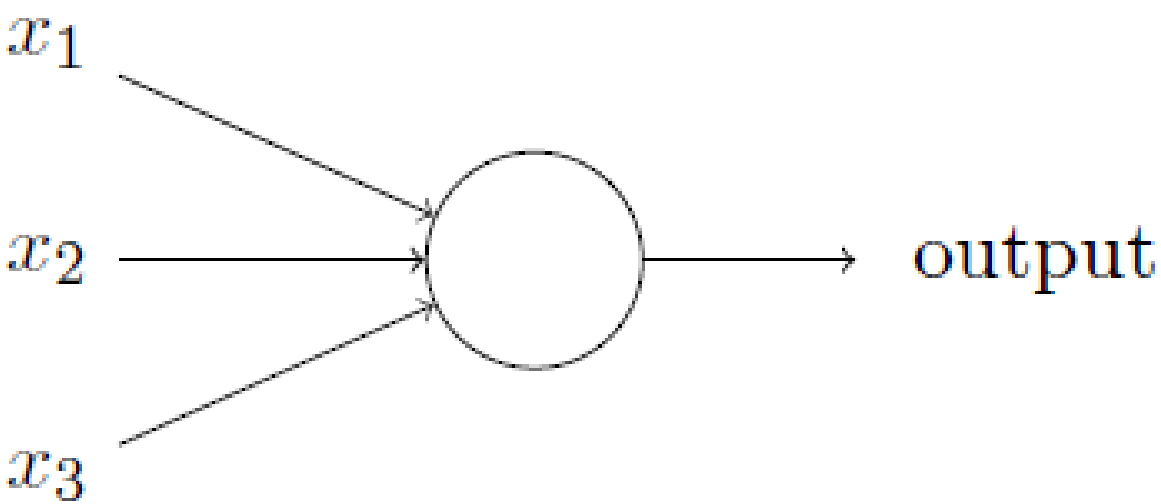
金曙松

北京聚睿智能科技有限公司

2017 年 7 月 10 日

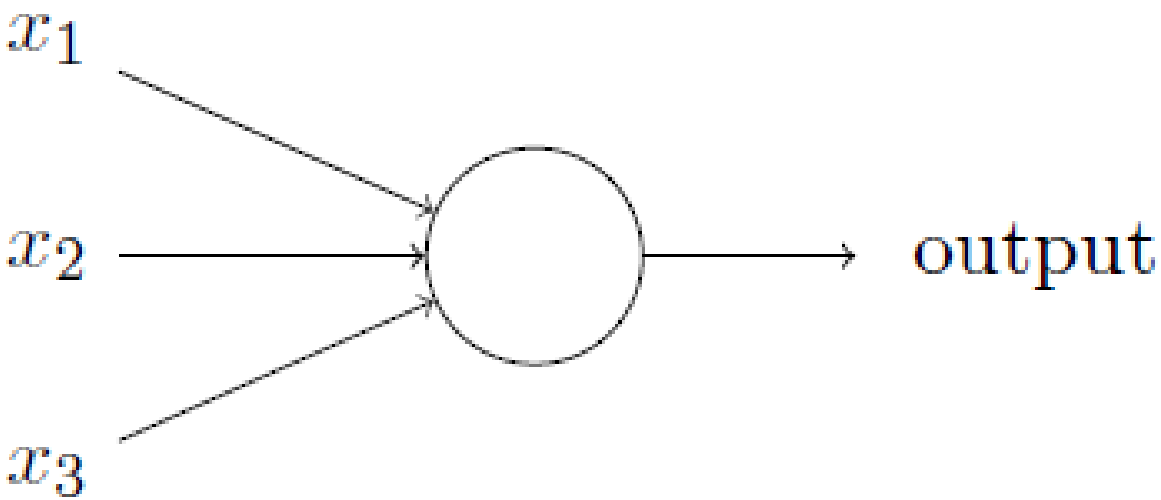
神经网络的雏形 — 感知器

- Frank Rosenblatt
- 感知器接受几个二进制输入 $x_1, x_2, \ldots$，并产生一个二进制输出：



神经网络的雏形 — 感知器

- Frank Rosenblatt
- 感知器接受几个二进制输入 $x_1, x_2, \ldots$，并产生一个二进制输出：



神经网络的雏形 — 感知器

- 引入权重 $w_1, w_2, \ldots,$ 表示相应输入对输出的重要性的实数
- 神经元的输出 0 或 1，由分配权重后的总和 $\sum_j w_j x_j$ 小于或者大于一些阈值决定
- 用公式表示：

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leqslant \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

神经网络的雏形 — 感知器

- 引入权重 $w_1, w_2, \ldots$，表示相应输入对输出的重要性的实数
- 神经元的输出 0 或 1，由分配权重后的总和 $\sum_j w_j x_j$ 小于或者大于一些阈值决定
- 用公式表示：

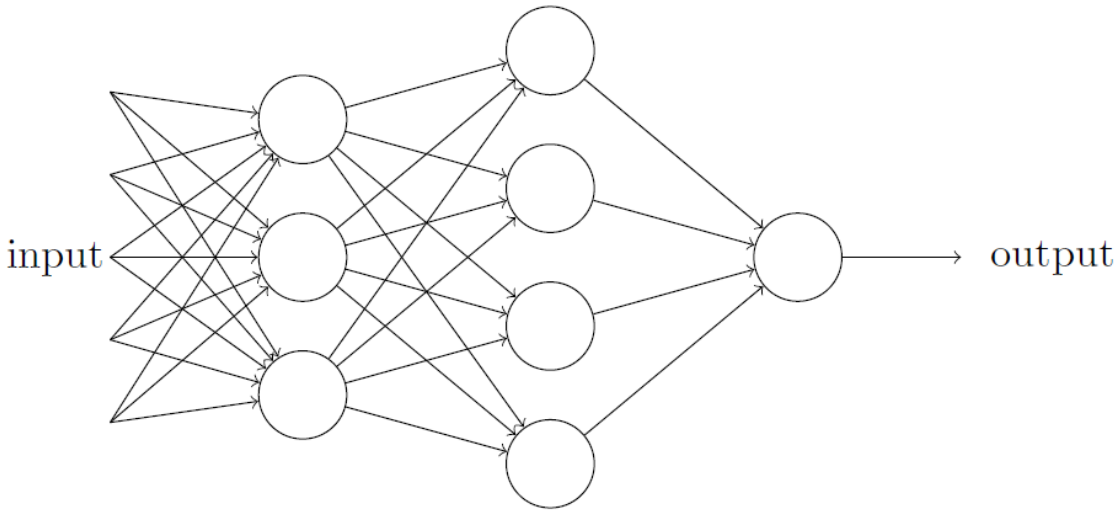
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leqslant \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

神经网络的雏形 — 感知器

- 引入权重 $w_1, w_2, \ldots$，表示相应输入对输出的重要性的实数
- 神经元的输出 0 或 1，由分配权重后的总和 $\sum_j w_j x_j$ 小于或者大于一些阈值决定
- 用公式表示：

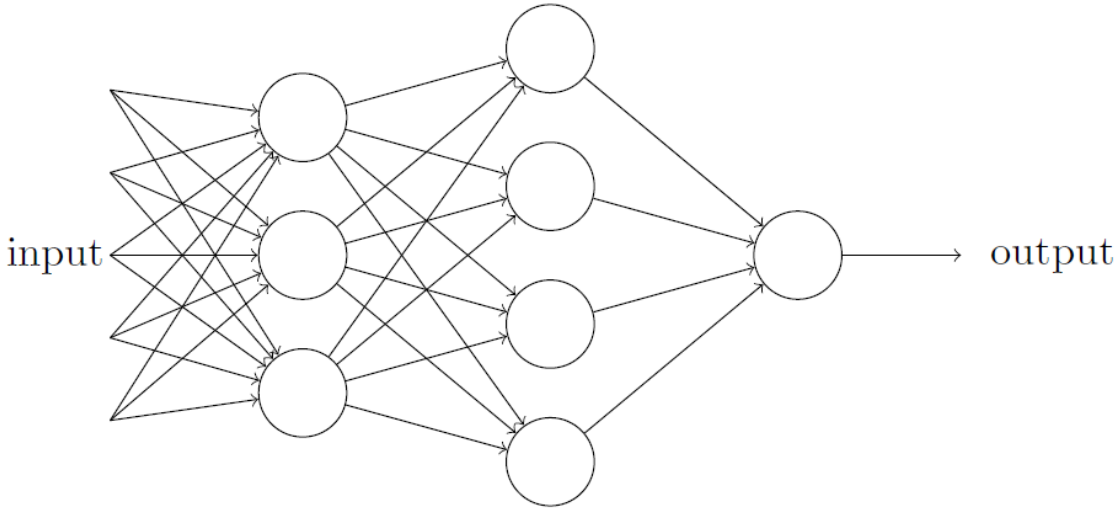
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leqslant \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

更为复杂的感知器



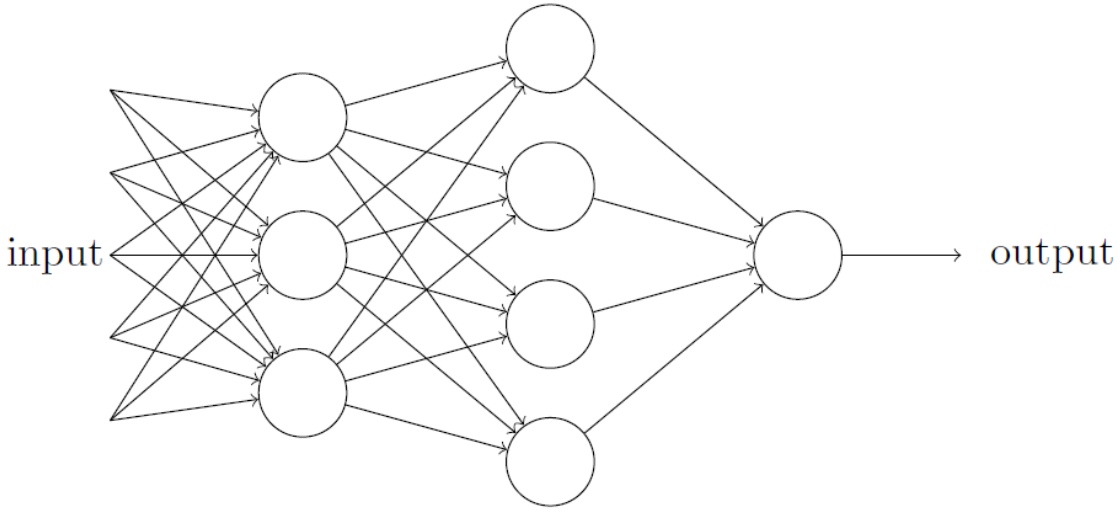
- 在这个网络中，第一列感知器——我们称其为第一层感知器——通过权衡输入依据做出三个非常简单的决定。
- 第二层的感知器每一个都在权衡第一层的决策结果并做出决定。一个第二层中的感知器可以比第一层中的做出更复杂和抽象的决策。
- 在第三层中的感知器甚至能进行更复杂的决策。
- 以这种方式，一个多层的感知器网络可以从事复杂巧妙的决策。

更为复杂的感知器



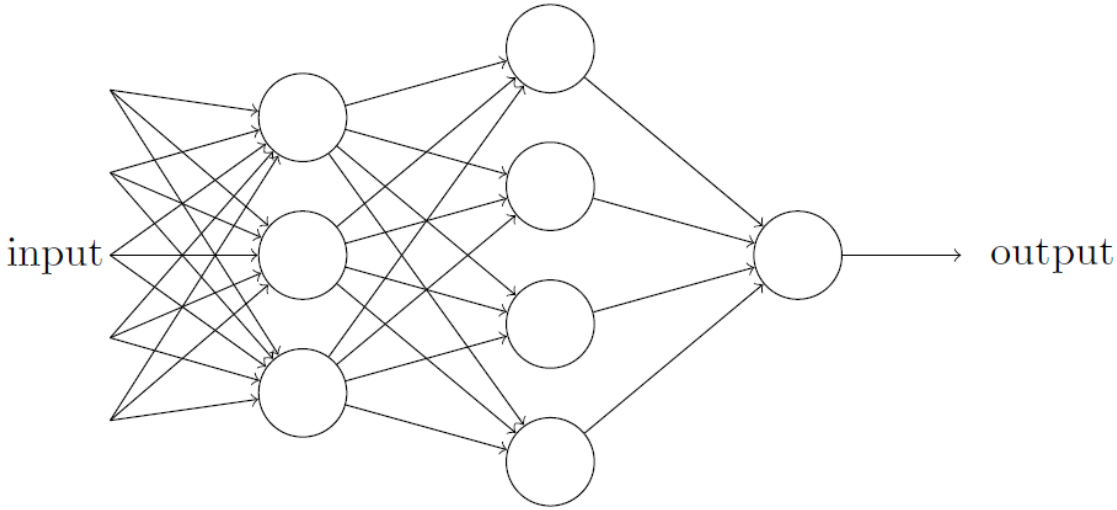
- 在这个网络中，第一列感知器——我们称其为第一层感知器——通过权衡输入依据做出三个非常简单的决定。
- 第二层的感知器每一个都在权衡第一层的决策结果并做出决定。一个第二层中的感知器可以比第一层中的做出更复杂和抽象的决策。
- 在第三层中的感知器甚至能进行更复杂的决策。
- 以这种方式，一个多层的感知器网络可以从事复杂巧妙的决策。

更为复杂的感知器



- 在这个网络中，第一列感知器——我们称其为第一层感知器——通过权衡输入依据做出三个非常简单的决定。
- 第二层的感知器每一个都在权衡第一层的决策结果并做出决定。一个第二层中的感知器可以比第一层中的做出更复杂和抽象的决策。
- 在第三层中的感知器甚至能进行更复杂的决策。
- 以这种方式，一个多层的感知器网络可以从事复杂巧妙的决策。

更为复杂的感知器



- 在这个网络中，第一列感知器——我们称其为第一层感知器——通过权衡输入依据做出三个非常简单的决定。
- 第二层的感知器每一个都在权衡第一层的决策结果并做出决定。一个第二层中的感知器可以比第一层中的做出更复杂和抽象的决策。
- 在第三层中的感知器甚至能进行更复杂的决策。
- 以这种方式，一个多层的感知器网络可以从事复杂巧妙的决策。

重写表达式

- 记 $w \cdot x \equiv \sum_j w_j x_j$, 其中 w 和 x 为权重和输入向量
- 感知器偏置: $b \equiv -\text{threshold}$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leqslant 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

重写表达式

- 记 $w \cdot x \equiv \sum_j w_j x_j$, 其中 w 和 x 为权重和输入向量
- 感知器偏置: $b \equiv -\text{threshold}$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leqslant 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

感知器的问题

- 网络中单个感知器上某个权重或偏置的微小改动有时候会引起那个感知器的输出完全翻转，如 0 变到 1。
- 那样的翻转可能接下来引起其余网络的行为以极其复杂的方式完全改变
- 我们可以引入一种称为 S 型神经元的新的人工神经元来克服这个问题。
- S 型神经元和感知器类似，但是被修改为权重和偏置的微小改动只引起输出的微小变化。

感知器的问题

- 网络中单个感知器上某个权重或偏置的微小改动有时候会引起那个感知器的输出完全翻转，如 0 变到 1。
- 那样的翻转可能接下来引起其余网络的行为以极其复杂的方式完全改变
- 我们可以引入一种称为 S 型神经元的新的人工神经元来克服这个问题。
- S 型神经元和感知器类似，但是被修改为权重和偏置的微小改动只引起输出的微小变化。

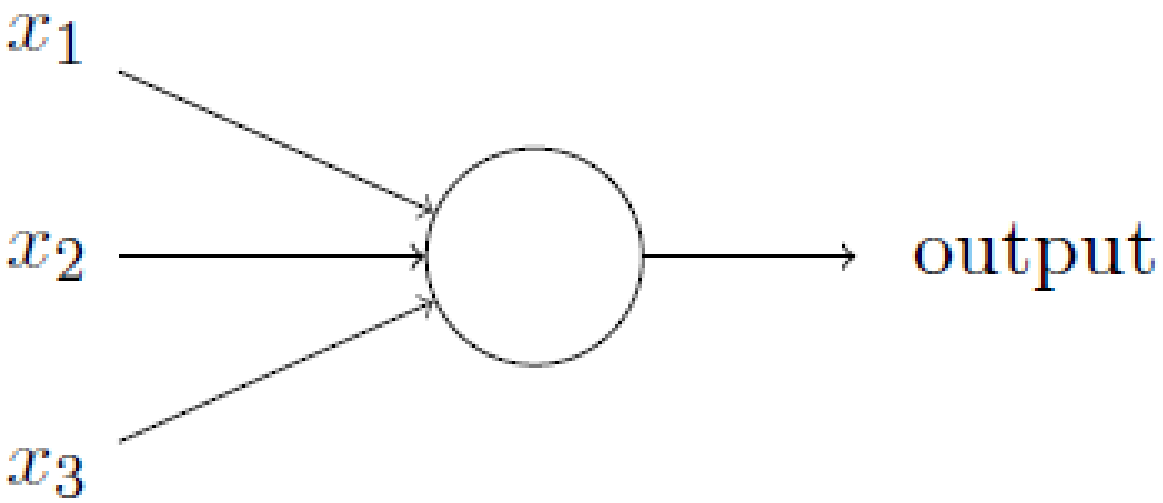
感知器的问题

- 网络中单个感知器上某个权重或偏置的微小改动有时候会引起那个感知器的输出完全翻转，如 0 变到 1。
- 那样的翻转可能接下来引起其余网络的行为以极其复杂的方式完全改变
- 我们可以引入一种称为 S 型神经元的新的人工神经元来克服这个问题。
- S 型神经元和感知器类似，但是被修改为权重和偏置的微小改动只引起输出的微小变化。

感知器的问题

- 网络中单个感知器上某个权重或偏置的微小改动有时候会引起那个感知器的输出完全翻转，如 0 变到 1。
- 那样的翻转可能接下来引起其余网络的行为以极其复杂的方式完全改变
- 我们可以引入一种称为 S 型神经元的新的人工神经元来克服这个问题。
- S 型神经元和感知器类似，但是被修改为权重和偏置的微小改动只引起输出的微小变化。

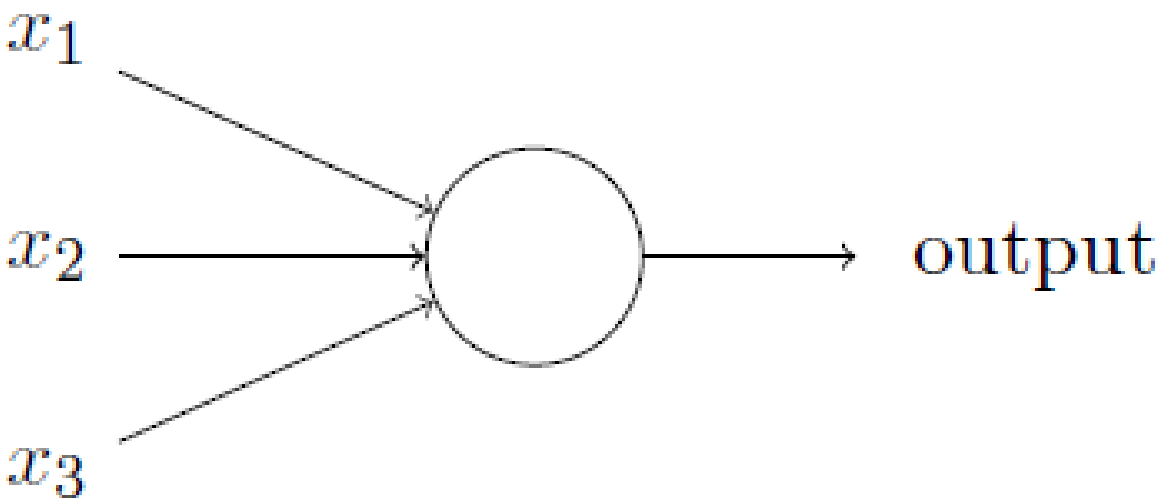
S 型神经元



- S 型神经元有多个输入，$x_1, x_2, \ldots$，但可以取 0 和 1 之间的任意值
- 同样，S 型神经元对每个输入有权重，$w_1, w_2, \ldots$，和一个总的偏置，b。
- 输出不是 0 或 1，而是 $\sigma(w \cdot x + b)$，其中 σ 是 S 型函数

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

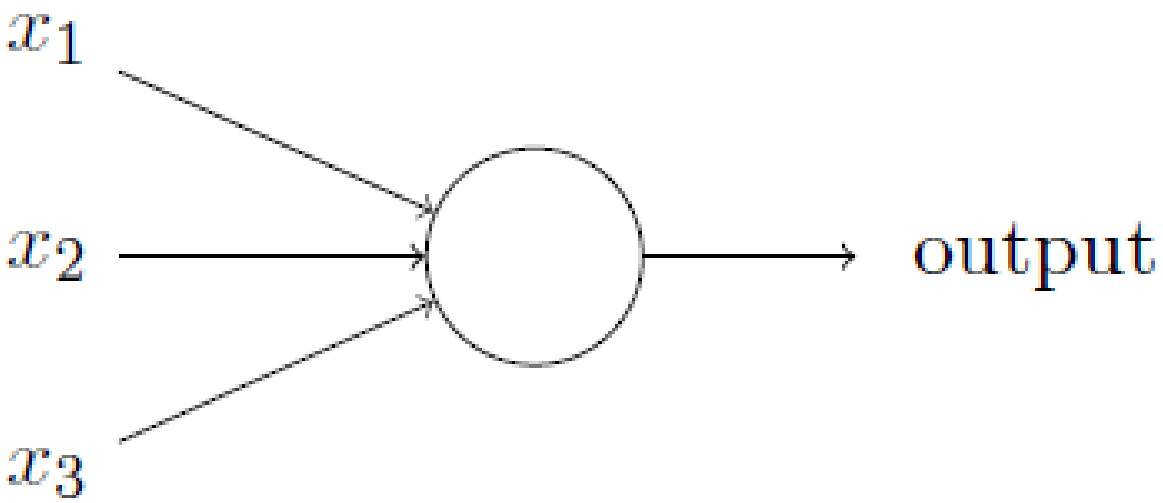
S 型神经元



- S 型神经元有多个输入，$x_1, x_2, \ldots$，但可以取 0 和 1 之间的任意值
- 同样，S 型神经元对每个输入有权重，$w_1, w_2, \ldots$，和一个总的偏置，b。
- 输出不是 0 或 1，而是 $\sigma(w \cdot x + b)$，其中 σ 是 S 型函数

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

S 型神经元



- S 型神经元有多个输入，$x_1, x_2, \ldots$，但可以取 0 和 1 之间的任意值
- 同样，S 型神经元对每个输入有权重，$w_1, w_2, \ldots$，和一个总的偏置，b。
- 输出不是 0 或 1，而是 $\sigma(w \cdot x + b)$，其中 σ 是 S 型函数

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

S 型神经元

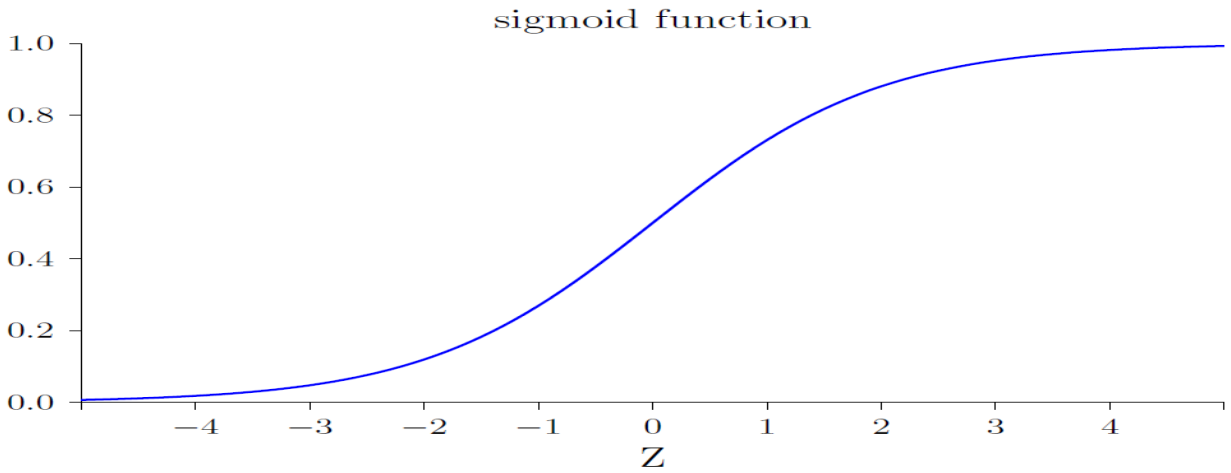
- 为了理解和感知器模型的相似性，假设 $z \equiv w \cdot x + b$ 是一个很大的正数。那么 $e^{-z} \approx 0$ 而 $\sigma(z) \approx 1$
- 即，当 $z = w \cdot x + b$ 很大并且为正，S 型神经元的输出近似为 1，正好和感知器一样。
- 反之，当 z 是一个很大的负数，则 $\sigma(z) \approx 0$，近似感知器。

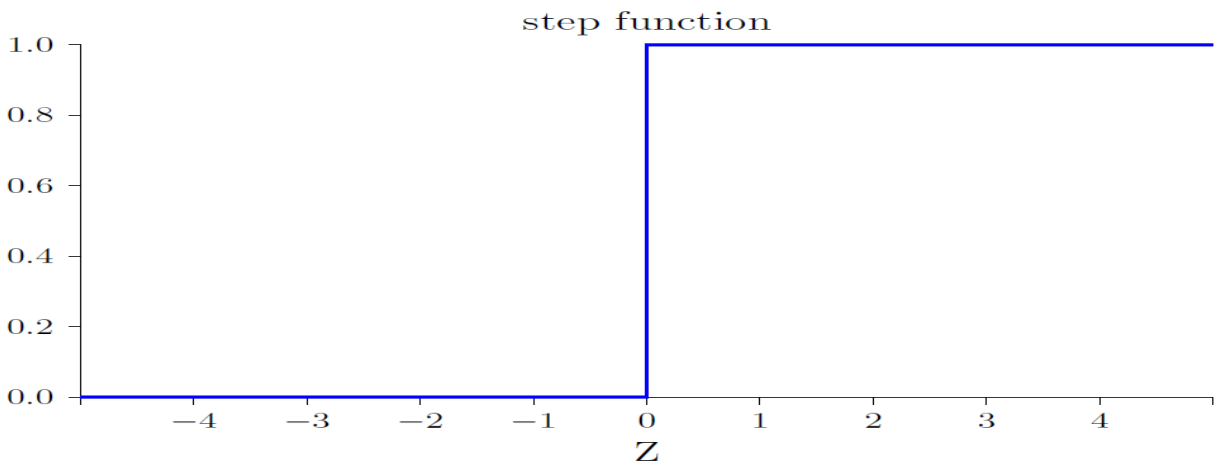
S 型神经元

- 为了理解和感知器模型的相似性，假设 $z \equiv w \cdot x + b$ 是一个很大的正数。那么 $e^{-z} \approx 0$ 而 $\sigma(z) \approx 1$
- 即，当 $z = w \cdot x + b$ 很大并且为正，S 型神经元的输出近似为 1，正好和感知器一样。
- 反之，当 z 是一个很大的负数，则 $\sigma(z) \approx 0$，近似感知器。

S 型神经元

- 为了理解和感知器模型的相似性，假设 $z \equiv w \cdot x + b$ 是一个很大的正数。那么 $e^{-z} \approx 0$ 而 $\sigma(z) \approx 1$
- 即，当 $z = w \cdot x + b$ 很大并且为正，S 型神经元的输出近似为 1，正好和感知器一样。
- 反之，当 z 是一个很大的负数，则 $\sigma(z) \approx 0$，近似感知器。


sigmoid function
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1
2
3
4
Z


step function
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1
2
3
4
Z

σ 函数

- σ 的平滑意味着权重和偏置的微小变化，即 Δw_j 和 Δb 会从神经元产生一个微小的输出变化 $\Delta output$

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b$$

- $\Delta output$ 是一个反映权重和偏置变化 (Δw_j 和 Δb) 的线性函数
- 决策标准：当输出大于某个给定值时 (例如 0.5) 输出 1，反之输出 0

σ 函数

- σ 的平滑意味着权重和偏置的微小变化，即 Δw_j 和 Δb 会从神经元产生一个微小的输出变化 Δoutput

$$\Delta \text{output} \approx \sum_j \frac{\partial \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial \, b} \Delta b$$

- Δoutput 是一个反映权重和偏置变化 (Δw_j 和 Δb) 的线性函数
- 决策标准：当输出大于某个给定值时 (例如 0.5) 输出 1，反之输出 0

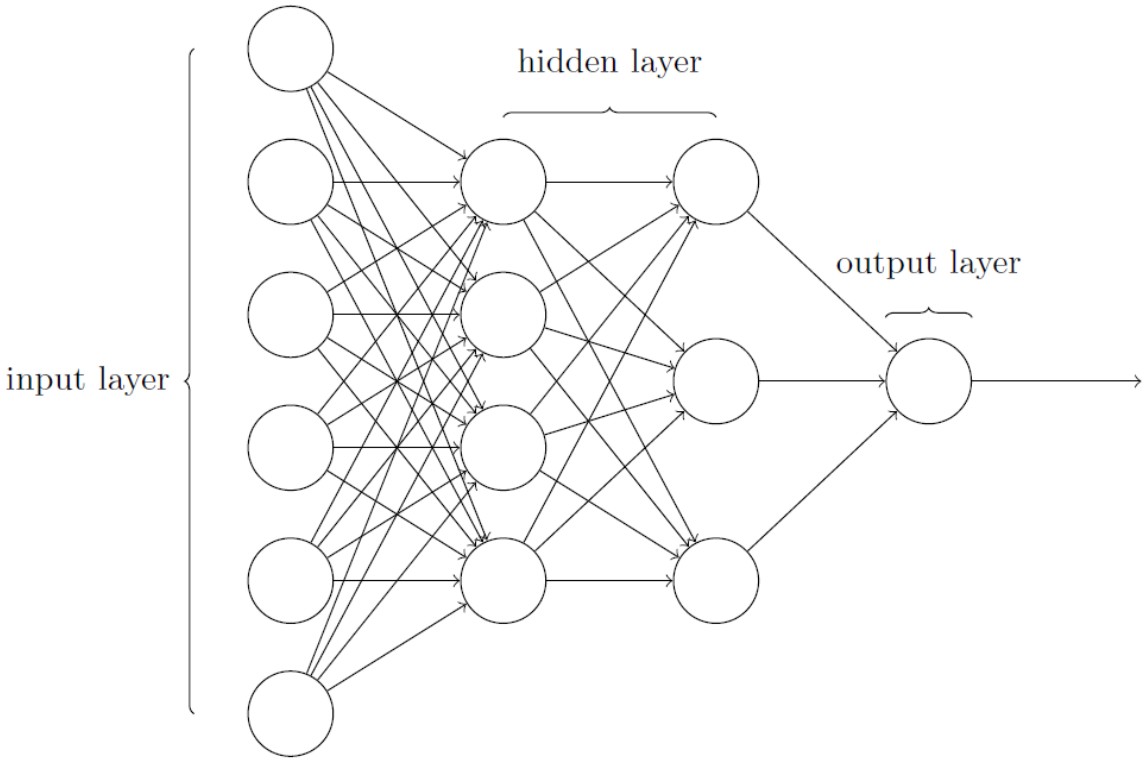
σ 函数

- σ 的平滑意味着权重和偏置的微小变化，即 Δw_j 和 Δb 会从神经元产生一个微小的输出变化 $\Delta output$

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b$$

- $\Delta output$ 是一个反映权重和偏置变化 (Δw_j 和 Δb) 的线性函数
- 决策标准：当输出大于某个给定值时 (例如 0.5) 输出 1，反之输出 0

神经网络的架构——前馈网络



例子

- 图像：28×28 的手写数字，所以输入层包含 784 个神经元
- 输入图像是灰度级别的，0.0 表示白色，1.0 表示黑色，中间数值表示深浅灰色
- 网络的第二层是一个隐藏层。我们用 n 来表示神经元的数量，我们将给 n 实验不同的数值。
- 网络的输出层包含有 10 个神经元。如果第一个神经元激活，即输出 ≈ 1，那么表明网络认为数字是一个 0。如果第二个神经元激活，就表明网络认为数字是一个 1。

例子

- 图像：28×28 的手写数字，所以输入层包含 784 个神经元
- 输入图像是灰度级别的，0.0 表示白色，1.0 表示黑色，中间数值表示深浅灰色
- 网络的第二层是一个隐藏层。我们用 n 来表示神经元的数量，我们将给 n 实验不同的数值。
- 网络的输出层包含有 10 个神经元。如果第一个神经元激活，即输出 ≈ 1，那么表明网络认为数字是一个 0。如果第二个神经元激活，就表明网络认为数字是一个 1。

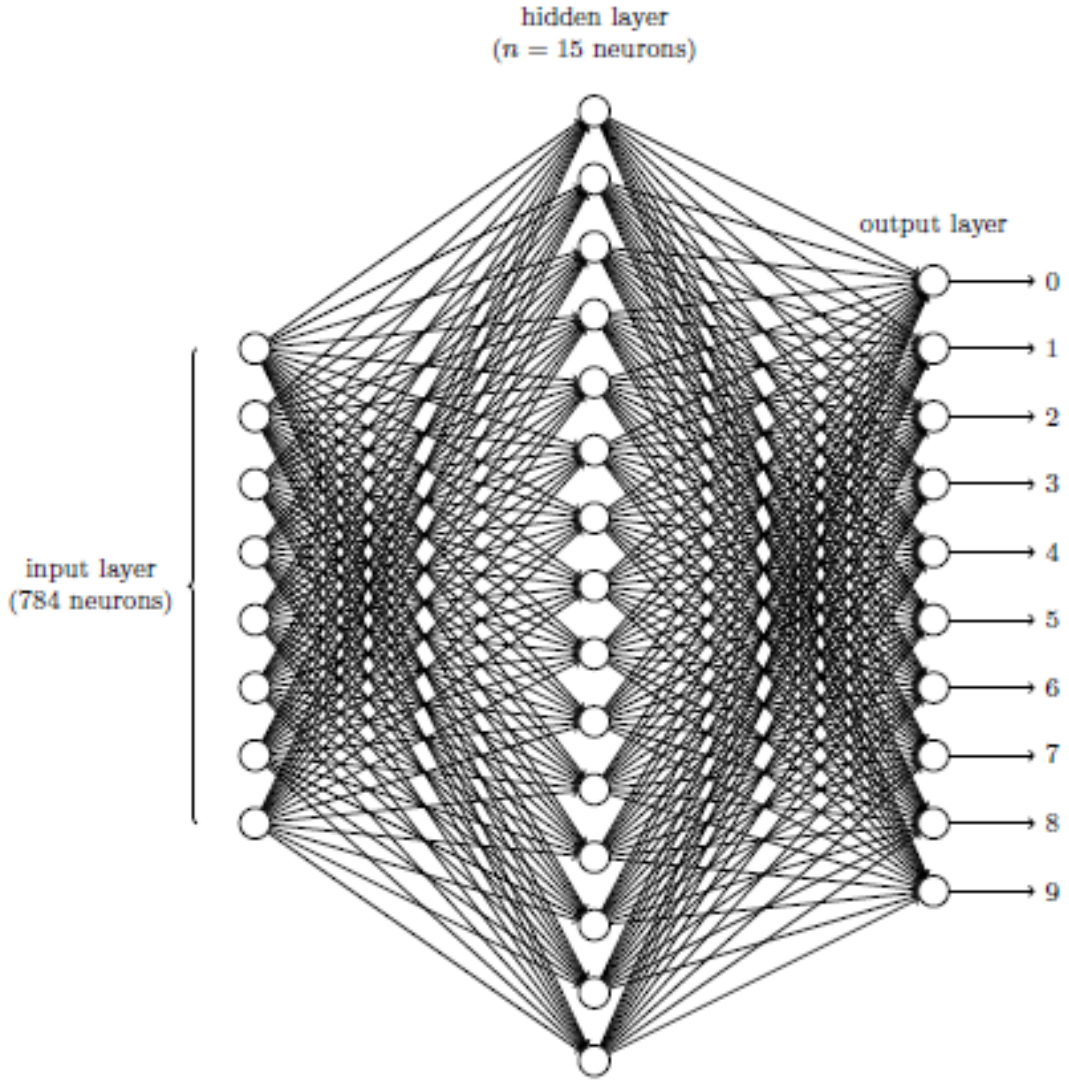
例子

- 图像：28×28 的手写数字，所以输入层包含 784 个神经元
- 输入图像是灰度级别的，0.0 表示白色，1.0 表示黑色，中间数值表示深浅灰色
- 网络的第二层是一个隐藏层。我们用 n 来表示神经元的数量，我们将给 n 实验不同的数值。
- 网络的输出层包含有 10 个神经元。如果第一个神经元激活，即输出 ≈ 1，那么表明网络认为数字是一个 0。如果第二个神经元激活，就表明网络认为数字是一个 1。

例子

- 图像：28×28 的手写数字，所以输入层包含 784 个神经元
- 输入图像是灰度级别的，0.0 表示白色，1.0 表示黑色，中间数值表示深浅灰色
- 网络的第二层是一个隐藏层。我们用 n 来表示神经元的数量，我们将给 n 实验不同的数值。
- 网络的输出层包含有 10 个神经元。如果第一个神经元激活，即输出 ≈ 1，那么表明网络认为数字是一个 0。如果第二个神经元激活，就表明网络认为数字是一个 1。

例子


hidden layer
($n = 15$ neurons)
output layer
input layer
(784 neurons)
0
1
2
3
4
5
6
7
8
9

使用梯度下降算法进行学习

- 数据分割：MNIST 数据集，第一部分 60,000 幅用于训练数据的图像；第二部分 10,000 用于测试
- 把每个训练样本 x 看成一个 $28 \times 28 = 784$ 维的向量，用 $y = y(x)$ 表示对应的期望输出，它是一个 10 维的向量，例如数字 3 的期望输出是 $y(x) = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)^T$
- 目的：找到权重和偏置，使得网络的输出 $y(x)$ 能拟合所有的训练输入 x
- 代价 (损失) 函数：

$$C(w, b) \equiv \frac{1}{2n} \sum_{2} ||y(x) - a||^2$$

其中 w: 所有的网络中权重的集合，b: 所有的偏置，n: 训练输入数据的个数，a: 当输入为 x 时输出的向量。这是均方误差或 MSE。

使用梯度下降算法进行学习

- 数据分割：MNIST 数据集，第一部分 60,000 幅用于训练数据的图像；第二部分 10,000 用于测试
- 把每个训练样本 x 看成一个 $28 \times 28 = 784$ 维的向量，用 $y = y(x)$ 表示对应的期望输出，它是一个 10 维的向量，例如数字 3 的期望输出是
  $y(x) = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)^T$
- 目的：找到权重和偏置，使得网络的输出 $y(x)$ 能拟合所有的训练输入 x
- 代价 (损失) 函数：

$$C(w, b) \equiv \frac{1}{2n} \sum_{2} ||y(x) - a||^2$$

其中 w: 所有的网络中权重的集合，b: 所有的偏置，n: 训练输入数据的个数，a: 当输入为 x 时输出的向量。这是均方误差或 MSE。

使用梯度下降算法进行学习

- 数据分割：MNIST 数据集，第一部分 60,000 幅用于训练数据的图像；第二部分 10,000 用于测试
- 把每个训练样本 x 看成一个 $28 \times 28 = 784$ 维的向量，用 $y = y(x)$ 表示对应的期望输出，它是一个 10 维的向量，例如数字 3 的期望输出是 $y(x) = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)^T$
- 目的：找到权重和偏置，使得网络的输出 $y(x)$ 能拟合所有的训练输入 x
- 代价 (损失) 函数：

$$C(w, b) \equiv \frac{1}{2n} \sum_{2} ||y(x) - a||^2$$

其中 w: 所有的网络中权重的集合，b: 所有的偏置，n: 训练输入数据的个数，a: 当输入为 x 时输出的向量。这是均方误差或 MSE。

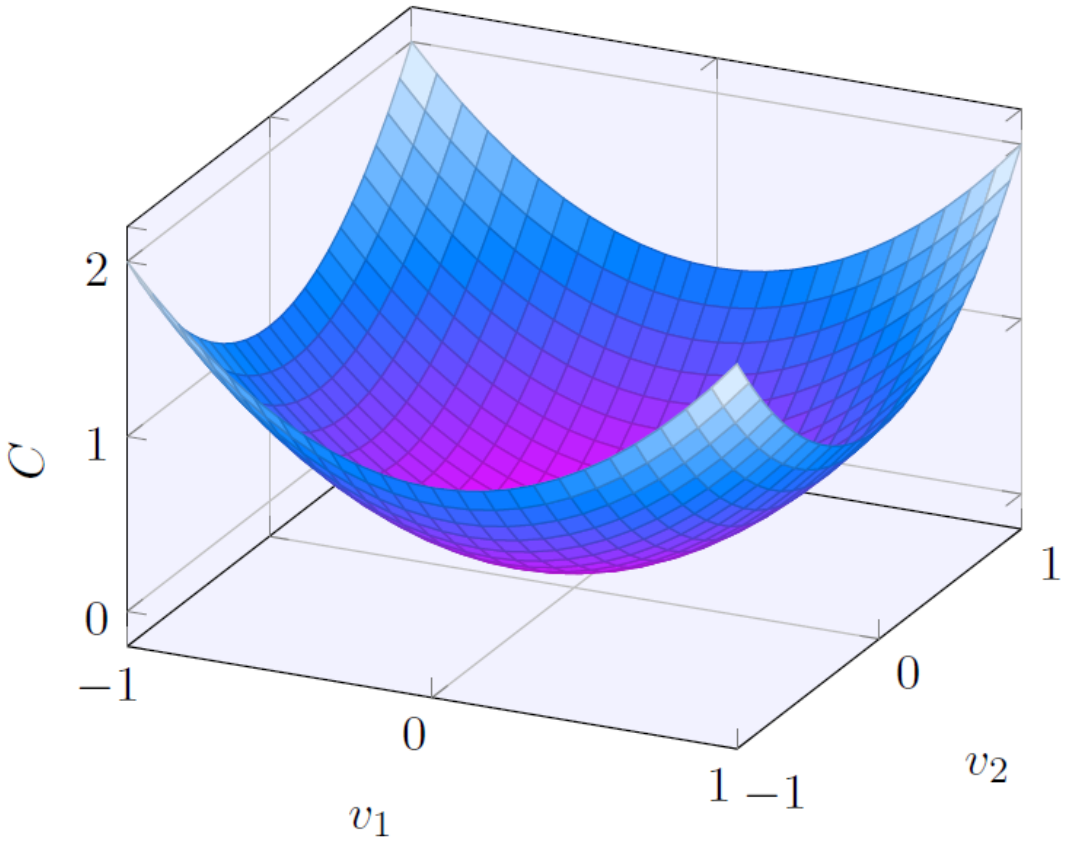
使用梯度下降算法进行学习

- 数据分割：MNIST 数据集，第一部分 60,000 幅用于训练数据的图像；第二部分 10,000 用于测试
- 把每个训练样本 x 看成一个 $28 \times 28 = 784$ 维的向量，用 $y = y(x)$ 表示对应的期望输出，它是一个 10 维的向量，例如数字 3 的期望输出是
  $y(x) = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)^T$
- 目的：找到权重和偏置，使得网络的输出 $y(x)$ 能拟合所有的训练输入 x
- 代价 (损失) 函数：

$$C(w, b) \equiv \frac{1}{2n} \sum_{2} ||y(x) - a||^2$$

其中 w: 所有的网络中权重的集合，b: 所有的偏置，n: 训练输入数据的个数，a: 当输入为 x 时输出的向量。这是均方误差或 MSE。

使用梯度下降算法进行学习



使用梯度下降算法进行学习

- 当我们在 v1 和 v2 方向分别将球体移动一个很小的量，即 $\Delta v1$ 和 $\Delta v2$ 时，

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 = \nabla C \cdot \Delta v$$

其中 $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$ 而 $\nabla C \equiv \partial C/\partial v_1, \partial C/\partial v_2)^T$

- 我们要寻找一种选择 Δv_1 和 Δv_2 的方法使得 ΔC 为负，使得球体滚落
- 令 $\Delta v = -\eta \nabla C$，则 ΔC 将成为负数。其中 η 是一个很小的正数 (称为学习速率)
- 总结一下，梯度下降算法工作的方式就是重复计算梯度 ∇C，然后沿着相反的方向移动，沿着山谷“滚落”。

使用梯度下降算法进行学习

- 当我们在 v1 和 v2 方向分别将球体移动一个很小的量，即 $\Delta v1$ 和 $\Delta v2$ 时，

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 = \nabla C \cdot \Delta v$$

  其中 $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$ 而 $\nabla C \equiv \partial C/\partial v_1, \partial C/\partial v_2)^T$
- 我们要寻找一种选择 Δv_1 和 Δv_2 的方法使得 ΔC 为负，使得球体滚落
- 令 $\Delta v = -\eta \nabla C$，则 ΔC 将成为负数。其中 η 是一个很小的正数 (称为学习速率)
- 总结一下，梯度下降算法工作的方式就是重复计算梯度 ∇C，然后沿着相反的方向移动，沿着山谷“滚落”。

使用梯度下降算法进行学习

- 当我们在 v1 和 v2 方向分别将球体移动一个很小的量，即 $\Delta v1$ 和 $\Delta v2$ 时，

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 = \nabla C \cdot \Delta v$$

  其中 $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$ 而 $\nabla C \equiv \partial C/\partial v_1, \partial C/\partial v_2)^T$
- 我们要寻找一种选择 Δv_1 和 Δv_2 的方法使得 ΔC 为负，使得球体滚落
- 令 $\Delta v = -\eta \nabla C$，则 ΔC 将成为负数。其中 η 是一个很小的正数 (称为学习速率)
- 总结一下，梯度下降算法工作的方式就是重复计算梯度 ∇C，然后沿着相反的方向移动，沿着山谷“滚落”。

使用梯度下降算法进行学习

- 当我们在 v1 和 v2 方向分别将球体移动一个很小的量，即 $\Delta v1$ 和 $\Delta v2$ 时，

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 = \nabla C \cdot \Delta v$$

  其中 $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$ 而 $\nabla C \equiv \partial C/\partial v_1, \partial C/\partial v_2)^T$
- 我们要寻找一种选择 Δv_1 和 Δv_2 的方法使得 ΔC 为负，使得球体滚落
- 令 $\Delta v = -\eta \nabla C$，则 ΔC 将成为负数。其中 η 是一个很小的正数 (称为学习速率)
- 总结一下，梯度下降算法工作的方式就是重复计算梯度 ∇C，然后沿着相反的方向移动，沿着山谷“滚落”。

随机梯度下降

- 其思想就是通过随机选取少量训练输入样本来计算 ∇C。
- 通过计算少量样本的平均值，快速得到一个对于实际梯度 ∇C 的估算
- 随机梯度下降通过随机选取少量 m 个训练输入来工作。
- 将这些随机的训练输入记为 $X_1, X_2, \ldots, X_m$ 把它们称为一个小批量数据 (mini-batch)。假设样本量 m 足够大，

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$$

- 算法变为：

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_\ell \to b_\ell' = b_\ell - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

随机梯度下降

- 其思想就是通过随机选取少量训练输入样本来计算 ∇C。
- 通过计算少量样本的平均值, 快速得到一个对于实际梯度 ∇C 的估算
- 随机梯度下降通过随机选取少量 m 个训练输入来工作。
- 将这些随机的训练输入记为 $X_1, X_2, \ldots, X_m$ 把它们称为一个小批量数据 (mini-batch)。假设样本量 m 足够大,

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$$

- 算法变为:

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_\ell \to b_\ell' = b_\ell - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

随机梯度下降

- 其思想就是通过随机选取少量训练输入样本来计算 ∇C。
- 通过计算少量样本的平均值, 快速得到一个对于实际梯度 ∇C 的估算
- 随机梯度下降通过随机选取少量 m 个训练输入来工作。
- 将这些随机的训练输入记为 $X_1, X_2, \ldots, X_m$ 把它们称为一个小批量数据 (mini-batch)。假设样本量 m 足够大,

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$$

- 算法变为:

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_\ell \to b_\ell' = b_\ell - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

随机梯度下降

- 其思想就是通过随机选取少量训练输入样本来计算 ∇C。
- 通过计算少量样本的平均值, 快速得到一个对于实际梯度 ∇C 的估算
- 随机梯度下降通过随机选取少量 m 个训练输入来工作。
- 将这些随机的训练输入记为 $X_1, X_2, \ldots, X_m$ 把它们称为一个小批量数据 (mini-batch)。假设样本量 m 足够大,

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$$

- 算法变为:

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_\ell \to b_\ell' = b_\ell - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

随机梯度下降

- 其思想就是通过随机选取少量训练输入样本来计算 ∇C。
- 通过计算少量样本的平均值, 快速得到一个对于实际梯度 ∇C 的估算
- 随机梯度下降通过随机选取少量 m 个训练输入来工作。
- 将这些随机的训练输入记为 $X_1, X_2, \ldots, X_m$ 把它们称为一个小批量数据 (mini-batch)。假设样本量 m 足够大,

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$$

- 算法变为:

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_\ell \to b_\ell' = b_\ell - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

反向传播

- 反向传播的目的是计算代价函数 C 分别关于 w 和 b 的偏导数 $\partial C/\partial w$ 和 $\partial C/\partial b$。
- 设代价函数

$$C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2$$

其中 n 是训练样本总数，L 表示网络层数，$a^L = a^L(x)$ 是当输入为 x 时的网络输出的激活值向量。

- 两个假设：
  1. 代价函数可以被写成一个在每个训练样本 x 上的代价函数 C_x 的均值 $C = \frac{1}{n}\sum_x C_x$
  2. 代价可以写成神经网络输出的函数

反向传播

- 反向传播的目的是计算代价函数 C 分别关于 w 和 b 的偏导数 $\partial C/\partial w$ 和 $\partial C/\partial b$。
- 设代价函数

$$C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2$$

其中 n 是训练样本总数，L 表示网络层数，$a^L = a^L(x)$ 是当输入为 x 时的网络输出的激活值向量。

- 两个假设：
  1. 代价函数可以被写成一个在每个训练样本 x 上的代价函数 C_x 的均值 $C = \frac{1}{n}\sum_x C_x$
  2. 代价可以写成神经网络输出的函数

反向传播

- 反向传播的目的是计算代价函数 C 分别关于 w 和 b 的偏导数 $\partial C/\partial w$ 和 $\partial C/\partial b$。
- 设代价函数

$$C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2$$

其中 n 是训练样本总数，L 表示网络层数，$a^L = a^L(x)$ 是当输入为 x 时的网络输出的激活值向量。

- 两个假设：
  1. 代价函数可以被写成一个在每个训练样本 x 上的代价函数 C_x 的均值 $C = \frac{1}{n}\sum_x C_x$
  2. 代价可以写成神经网络输出的函数

反向传播

- 反向传播的目的是计算代价函数 C 分别关于 w 和 b 的偏导数 $\partial C/\partial w$ 和 $\partial C/\partial b$。
- 设代价函数

$$C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2$$

其中 n 是训练样本总数，L 表示网络层数，$a^L = a^L(x)$ 是当输入为 x 时的网络输出的激活值向量。

- 两个假设：
  1. 代价函数可以被写成一个在每个训练样本 x 上的代价函数 C_x 的均值 $C = \frac{1}{n}\sum_x C_x$
  2. 代价可以写成神经网络输出的函数

反向传播

- 反向传播的目的是计算代价函数 C 分别关于 w 和 b 的偏导数 $\partial C/\partial w$ 和 $\partial C/\partial b$。
- 设代价函数

$$C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2$$

其中 n 是训练样本总数，L 表示网络层数，$a^L = a^L(x)$ 是当输入为 x 时的网络输出的激活值向量。

- 两个假设：
  1. 代价函数可以被写成一个在每个训练样本 x 上的代价函数 C_x 的均值 $C = \frac{1}{n}\sum_x C_x$
  2. 代价可以写成神经网络输出的函数

Hadamard 乘积, $s \odot t$

假设 s 和 t 是两个同样维度的向量。那么我们使用 $s \odot t$ 来表示按元素的乘积。所以 $s \odot t$ 的元素就是 $(s \odot t)_j = s_j t_j$。例如：

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \odot \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 \times 3 \\ 2 \times 4 \end{pmatrix} = \begin{pmatrix} 3 \\ 8 \end{pmatrix}$$

反向传播的四个方程

- 引入中间量 $\delta_{\mathrm{j}}^{\ell}$ 为在 ℓ^{th} 层第 j^{th} 个神经元上的计算误差

$$\delta_{\mathrm{j}}^{\ell} \equiv \frac{\partial \mathrm{C}}{\partial \mathrm{z}_{\mathrm{j}}^{\ell}}$$

- 输出层误差方程：

$$\delta_{\mathrm{j}}^{\mathrm{L}} = \frac{\partial \mathrm{C}}{\partial \mathrm{a}_{\mathrm{j}}^{\mathrm{L}}} \sigma'(\mathrm{z}_{\mathrm{j}}^{\mathrm{L}}) \tag{1}$$

用矩阵形式 $\delta^{\mathrm{L}} = \nabla_{\mathrm{a}} \mathrm{C} \odot \sigma'(\mathrm{z}^{\mathrm{L}})$。在二次代价函数时，$\nabla_{\mathrm{a}} \mathrm{C} = (\mathrm{a}^{\mathrm{L}} - \mathrm{y})$，所以 $\delta^{\mathrm{L}} = (\mathrm{a}^{\mathrm{L}} - \mathrm{y}) \odot \sigma'(\mathrm{z}^{\mathrm{L}})$

- 使用下一层误差 $\delta^{\ell+1}$ 表示当前层的误差 δ^{ℓ}

$$\delta^{\ell} = ((\mathrm{w}^{\ell+1})^{\mathrm{T}} \delta^{\ell+1}) \odot \sigma'(\mathrm{z}^{\ell}) \tag{2}$$

其中 $(\mathrm{w}^{\ell+1})^{\mathrm{T}}$ 是 $(\ell+1)^{\mathrm{th}}$ 层权重矩阵 $\mathrm{w}^{\ell+1}$ 的转置。

反向传播的四个方程

- 引入中间量 δ_j^ℓ 为在 ℓ^{th} 层第 j^{th} 个神经元上的计算误差

$$\delta_j^\ell \equiv \frac{\partial C}{\partial z_j^\ell}$$

- 输出层误差方程：

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{1}$$

用矩阵形式 $\delta^L = \nabla_a C \odot \sigma'(z^L)$。在二次代价函数时，$\nabla_a C = (a^L - y)$，所以 $\delta^L = (a^L - y) \odot \sigma'(z^L)$

- 使用下一层误差 $\delta^{\ell+1}$ 表示当前层的误差 δ^ℓ

$$\delta^\ell = ((w^{\ell+1})^T \delta^{\ell+1}) \odot \sigma'(z^\ell) \tag{2}$$

其中 $(w^{\ell+1})^T$ 是 $(\ell+1)^{th}$ 层权重矩阵 $w^{\ell+1}$ 的转置。

反向传播的四个方程

- 引入中间量 $\delta_{\mathrm{j}}^{\ell}$ 为在 ℓ^{th} 层第 j^{th} 个神经元上的计算误差

$$\delta_{\mathrm{j}}^{\ell} \equiv \frac{\partial \mathrm{C}}{\partial \mathrm{z}_{\mathrm{j}}^{\ell}}$$

- 输出层误差方程：

$$\delta_{\mathrm{j}}^{\mathrm{L}} = \frac{\partial \mathrm{C}}{\partial \mathrm{a}_{\mathrm{j}}^{\mathrm{L}}} \sigma'(\mathrm{z}_{\mathrm{j}}^{\mathrm{L}}) \tag{1}$$

用矩阵形式 $\delta^{\mathrm{L}} = \nabla_{\mathrm{a}}\mathrm{C} \odot \sigma'(\mathrm{z}^{\mathrm{L}})$。在二次代价函数时，$\nabla_{\mathrm{a}}\mathrm{C} = (\mathrm{a}^{\mathrm{L}} - \mathrm{y})$，所以 $\delta^{\mathrm{L}} = (\mathrm{a}^{\mathrm{L}} - \mathrm{y}) \odot \sigma'(\mathrm{z}^{\mathrm{L}})$

- 使用下一层误差 $\delta^{\ell+1}$ 表示当前层的误差 δ^{ℓ}

$$\delta^{\ell} = ((\mathrm{w}^{\ell+1})^{\mathrm{T}}\delta^{\ell+1}) \odot \sigma'(\mathrm{z}^{\ell}) \tag{2}$$

其中 $(\mathrm{w}^{\ell+1})^{\mathrm{T}}$ 是 $(\ell+1)^{\mathrm{th}}$ 层权重矩阵 $\mathrm{w}^{\ell+1}$ 的转置。

反向传播的四个方程

- 利用公式(1)和(2)，我们首先计算 δ^{L}，然后计算 $\delta^{\mathrm{L}-1}$，然后 $\delta^{\mathrm{L}-2}$，如此反向传播完整个网络。
- 代价函数关于网络中任意偏置的改变率

$$\frac{\partial \mathrm{C}}{\partial \mathrm{b}_{\mathrm{j}}^{\ell}} = \delta_{\mathrm{j}}^{\ell} \tag{3}$$

即误差 $\delta_{\mathrm{j}}^{\ell}$ 和偏导数 $\partial \mathrm{C}/\partial \mathrm{b}_{\mathrm{j}}^{\ell}$ 完全一致

- 代价函数关于任何一个权重的改变率:

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}_{\mathrm{jk}}^{\ell}} = \mathrm{a}_{\mathrm{k}}^{\ell-1}\delta_{\mathrm{j}}^{\mathrm{l}} \tag{4}$$

这告诉我们如何计算 $\partial \mathrm{C}/\partial \mathrm{w}_{\mathrm{jk}}^{\ell}$，即 $\partial \mathrm{C}/\partial \mathrm{w} = \mathrm{a}_{\mathrm{in}}\delta_{\mathrm{out}}$

反向传播的四个方程

- 利用公式(1)和(2)，我们首先计算 δ^{L}，然后计算 $\delta^{\mathrm{L}-1}$，然后 $\delta^{\mathrm{L}-2}$，如此反向传播完整个网络。
- 代价函数关于网络中任意偏置的改变率

$$\frac{\partial \mathrm{C}}{\partial \mathrm{b}_{\mathrm{j}}^{\ell}} = \delta_{\mathrm{j}}^{\ell} \tag{3}$$

即误差 $\delta_{\mathrm{j}}^{\ell}$ 和偏导数 $\partial \mathrm{C}/\partial \mathrm{b}_{\mathrm{j}}^{\ell}$ 完全一致

- 代价函数关于任何一个权重的改变率:

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}_{\mathrm{jk}}^{\ell}} = \mathrm{a}_{\mathrm{k}}^{\ell-1}\delta_{\mathrm{j}}^{\mathrm{l}} \tag{4}$$

这告诉我们如何计算 $\partial \mathrm{C}/\partial \mathrm{w}_{\mathrm{jk}}^{\ell}$，即 $\partial \mathrm{C}/\partial \mathrm{w} = \mathrm{a}_{\mathrm{in}}\delta_{\mathrm{out}}$

反向传播的四个方程

- 利用公式(1)和(2)，我们首先计算 δ^{L}，然后计算 $\delta^{\mathrm{L}-1}$，然后 $\delta^{\mathrm{L}-2}$，如此反向传播完整个网络。
- 代价函数关于网络中任意偏置的改变率

$$\frac{\partial \mathrm{C}}{\partial \mathrm{b}_{\mathrm{j}}^{\ell}} = \delta_{\mathrm{j}}^{\ell} \tag{3}$$

即误差 $\delta_{\mathrm{j}}^{\ell}$ 和偏导数 $\partial \mathrm{C}/\partial \mathrm{b}_{\mathrm{j}}^{\ell}$ 完全一致

- 代价函数关于任何一个权重的改变率:

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}_{\mathrm{jk}}^{\ell}} = \mathrm{a}_{\mathrm{k}}^{\ell-1}\delta_{\mathrm{j}}^{\mathrm{l}} \tag{4}$$

这告诉我们如何计算 $\partial \mathrm{C}/\partial \mathrm{w}_{\mathrm{jk}}^{\ell}$，即 $\partial \mathrm{C}/\partial \mathrm{w} = \mathrm{a}_{\mathrm{in}}\delta_{\mathrm{out}}$

反向传播算法

1. **输入** x：为输入层设置对应的激活值 a^1
2. **向前传播**：对每个 $\ell = 2, 3, \ldots, L$ 计算相应的 $z^\ell = w^\ell a^{\ell-1} + b^\ell$ 和 $a^\ell = \sigma(z^\ell)$
3. **输出层误差** δ^L：计算向量 $\delta^L = \nabla_a C \odot \sigma'(z^L)$
4. **反向误差传播**：对每个 $\ell = L-1, L-2, \ldots, 2$，计算 $\delta^\ell = ((w^{\ell+1})^T \delta^{\ell+1}) \odot \sigma'(z^\ell)$
5. **输出**：代价函数的梯度由

$$\frac{\partial C}{\partial w^\ell_{jk}} = a^{\ell-1}_k \delta^\ell_j \text{ 和 } \frac{\partial C}{\partial b^\ell_j} = \delta^\ell_j$$

得出。

反向传播算法

1. **输入** x：为输入层设置对应的激活值 a^1
2. **向前传播：** 对每个 $\ell = 2, 3, \ldots, \mathrm{L}$ 计算相应的 $\mathrm{z}^\ell = \mathrm{w}^\ell \mathrm{a}^{\ell-1} + \mathrm{b}^\ell$ 和 $\mathrm{a}^\ell = \sigma(\mathrm{z}^\ell)$
3. **输出层误差** δ^{L}：计算向量 $\delta^{\mathrm{L}} = \nabla_{\mathrm{a}} \mathrm{C} \odot \sigma'(\mathrm{z}^{\mathrm{L}})$
4. **反向误差传播：** 对每个 $\ell = \mathrm{L}-1, \mathrm{L}-2, \ldots, 2$，计算 $\delta^\ell = ((\mathrm{w}^{\ell+1})^{\mathrm{T}} \delta^{\ell+1}) \odot \sigma'(\mathrm{z}^\ell)$
5. **输出：** 代价函数的梯度由

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}^\ell_{\mathrm{jk}}} = \mathrm{a}^{\ell-1}_{\mathrm{k}} \delta^\ell_{\mathrm{j}} \text{ 和 } \frac{\partial \mathrm{C}}{\partial \mathrm{b}^\ell_{\mathrm{j}}} = \delta^\ell_{\mathrm{j}}$$

得出。

反向传播算法

1. **输入** x：为输入层设置对应的激活值 a^1
2. **向前传播**：对每个 $\ell = 2, 3, \ldots, \mathrm{L}$ 计算相应的 $\mathrm{z}^\ell = \mathrm{w}^\ell \mathrm{a}^{\ell-1} + \mathrm{b}^\ell$ 和 $\mathrm{a}^\ell = \sigma(\mathrm{z}^\ell)$
3. **输出层误差** δ^{L}：计算向量 $\delta^{\mathrm{L}} = \nabla_{\mathrm{a}} \mathrm{C} \odot \sigma'(\mathrm{z}^{\mathrm{L}})$
4. **反向误差传播**：对每个 $\ell = \mathrm{L}-1, \mathrm{L}-2, \ldots, 2$，计算 $\delta^\ell = ((\mathrm{w}^{\ell+1})^{\mathrm{T}} \delta^{\ell+1}) \odot \sigma'(\mathrm{z}^\ell)$
5. **输出**：代价函数的梯度由

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}^\ell_{\mathrm{jk}}} = \mathrm{a}^{\ell-1}_{\mathrm{k}} \delta^\ell_{\mathrm{j}} \text{ 和 } \frac{\partial \mathrm{C}}{\partial \mathrm{b}^\ell_{\mathrm{j}}} = \delta^\ell_{\mathrm{j}}$$

得出。

反向传播算法

1. **输入** x：为输入层设置对应的激活值 a^1
2. **向前传播**：对每个 $\ell = 2, 3, \ldots, \mathrm{L}$ 计算相应的 $\mathrm{z}^\ell = \mathrm{w}^\ell \mathrm{a}^{\ell-1} + \mathrm{b}^\ell$ 和 $\mathrm{a}^\ell = \sigma(\mathrm{z}^\ell)$
3. **输出层误差** δ^{L}：计算向量 $\delta^{\mathrm{L}} = \nabla_{\mathrm{a}} \mathrm{C} \odot \sigma'(\mathrm{z}^{\mathrm{L}})$
4. **反向误差传播**：对每个 $\ell = \mathrm{L}-1, \mathrm{L}-2, \ldots, 2$，计算 $\delta^\ell = ((\mathrm{w}^{\ell+1})^{\mathrm{T}} \delta^{\ell+1}) \odot \sigma'(\mathrm{z}^\ell)$
5. **输出**：代价函数的梯度由

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}^\ell_{\mathrm{jk}}} = \mathrm{a}^{\ell-1}_{\mathrm{k}} \delta^\ell_{\mathrm{j}} \text{ 和 } \frac{\partial \mathrm{C}}{\partial \mathrm{b}^\ell_{\mathrm{j}}} = \delta^\ell_{\mathrm{j}}$$

得出。

反向传播算法

1. **输入** x：为输入层设置对应的激活值 a^1
2. **向前传播**：对每个 $\ell = 2, 3, \ldots, \mathrm{L}$ 计算相应的 $\mathrm{z}^\ell = \mathrm{w}^\ell \mathrm{a}^{\ell-1} + \mathrm{b}^\ell$ 和 $\mathrm{a}^\ell = \sigma(\mathrm{z}^\ell)$
3. **输出层误差** δ^{L}：计算向量 $\delta^{\mathrm{L}} = \nabla_{\mathrm{a}} \mathrm{C} \odot \sigma'(\mathrm{z}^{\mathrm{L}})$
4. **反向误差传播**：对每个 $\ell = \mathrm{L}-1, \mathrm{L}-2, \ldots, 2$，计算 $\delta^\ell = ((\mathrm{w}^{\ell+1})^{\mathrm{T}} \delta^{\ell+1}) \odot \sigma'(\mathrm{z}^\ell)$
5. **输出**：代价函数的梯度由

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}^\ell_{\mathrm{jk}}} = \mathrm{a}^{\ell-1}_{\mathrm{k}} \delta^\ell_{\mathrm{j}} \text{ 和 } \frac{\partial \mathrm{C}}{\partial \mathrm{b}^\ell_{\mathrm{j}}} = \delta^\ell_{\mathrm{j}}$$

得出。