

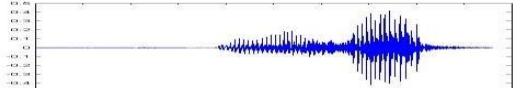
# 8. PyTorch 入门

- 机器学习与深度学习
- PyTorch 深度学习训练

# 机器学习

- 机器学习≈寻找一个函数

- 语音识别

$f(\text{  }) = \text{"How are you"}$

- 图片识别

$f(\text{  }) = \text{"Cat"}$

- 下围棋

$f(\text{  }) = \text{"5-5"} \quad (\text{下一步棋})$

- 对话系统

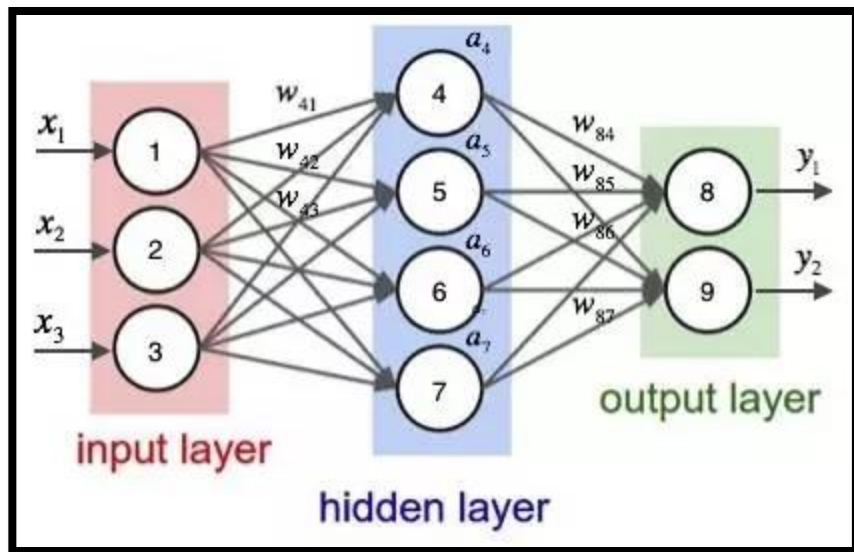
$f(\text{ "Hi" } \text{ (你说的内容) }) = \text{ "Hello" } \text{ (系统的回答)}$

# 机器学习

- 机器学习的核心是寻找  $f$ 。
- 这里的  $f$  可以是简单的闭式表达：
$$f(x) = x^2 + 2x + 1$$

- 也可以是一个复杂的神经网络：

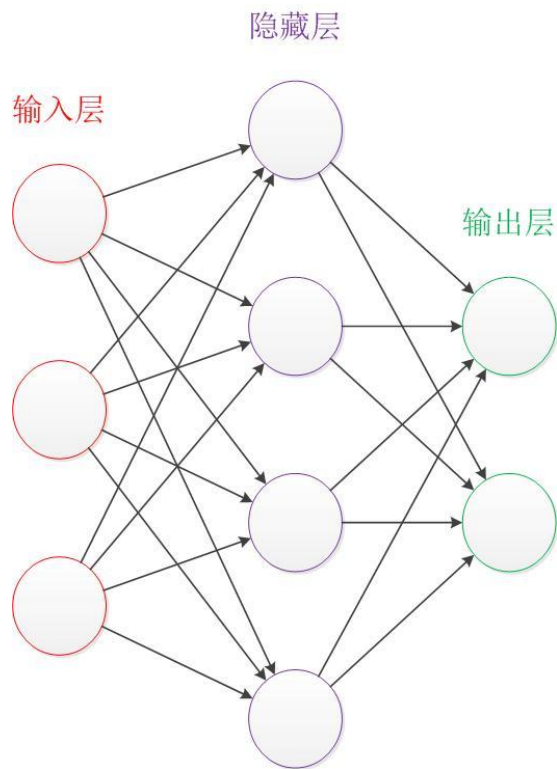
- $f(x_1, x_2, x_3) =$



神经网络事实上是一个复杂的 $f$ 函数。接下来将以一种简单的, 循序的方式讲解神经网络。

# 神经网络

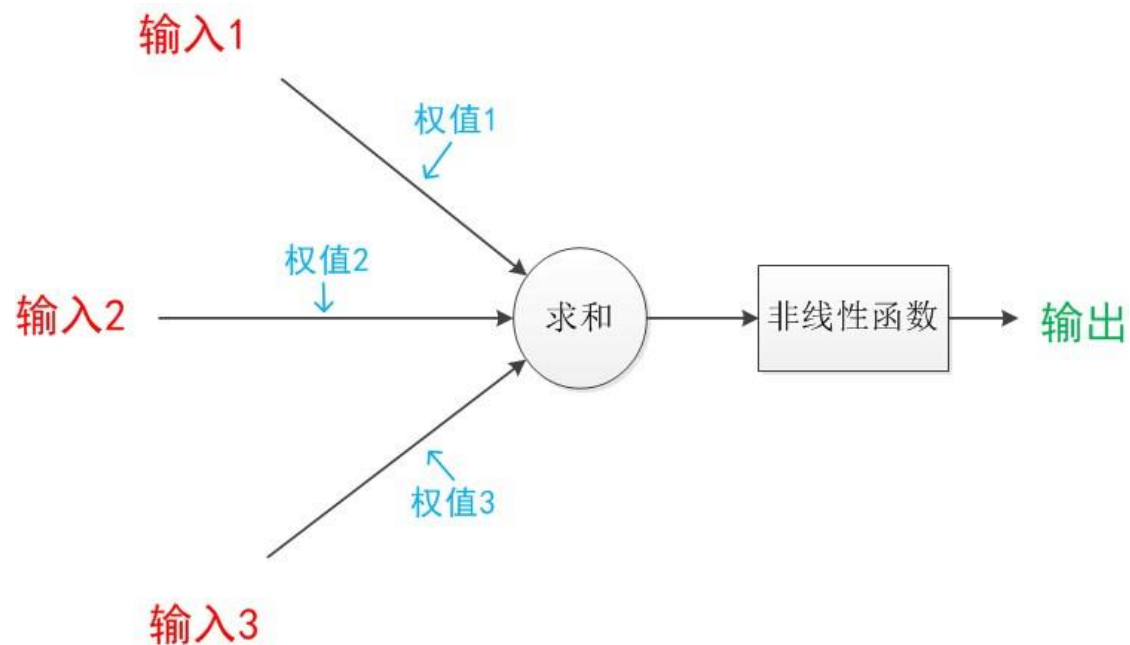
- 让我们来看一个经典的神经网络。这是一个包含三个层次的神经网络。红色的是**输入层**，绿色的是**输出层**，紫色的是**中间层**（也叫**隐藏层**）。输入层有3个输入单元，隐藏层有4个单元，输出层有2个单元。后文中，我们统一使用这种颜色来表达神经网络的结构。



1. 设计一个神经网络时，输入层与输出层的节点数往往是固定的，中间层则可以自由指定；
2. 神经网络结构图中的拓扑与箭头代表着预测过程时数据的流向。
3. 结构图里的关键不是圆圈（代表“神经元”），而是连接线（代表“神经元”之间的连接）。每个连接线对应一个不同的**权重**（其值称为**权值**），这是需要训练得到的。

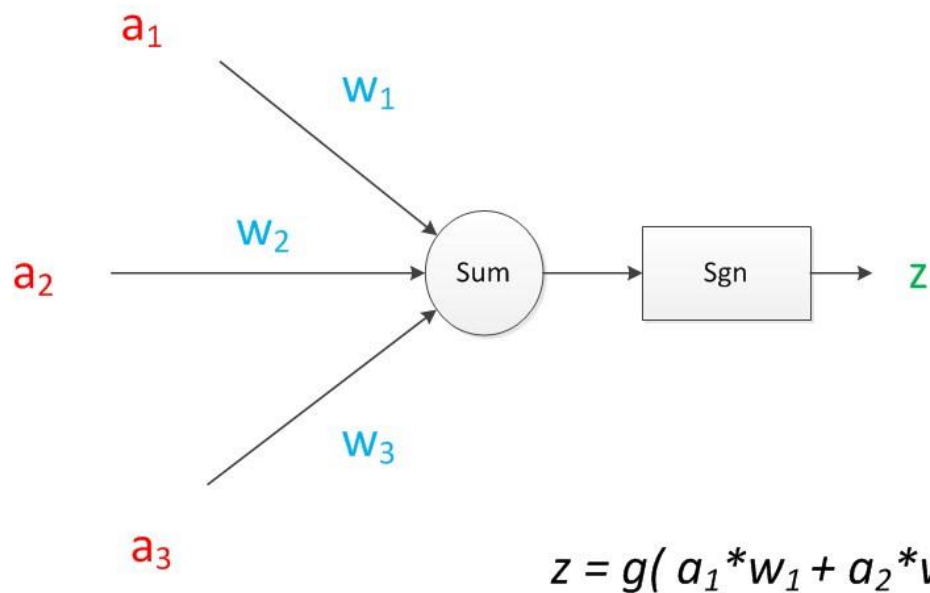
# 神经元

- 神经元模型是一个包含输入，输出与计算功能的模型。
- 下图是一个典型的神经元模型：包含有3个输入，1个输出，以及2个计算功能。注意中间的箭头线。这些线称为“连接”。每个上有一个“权值”。



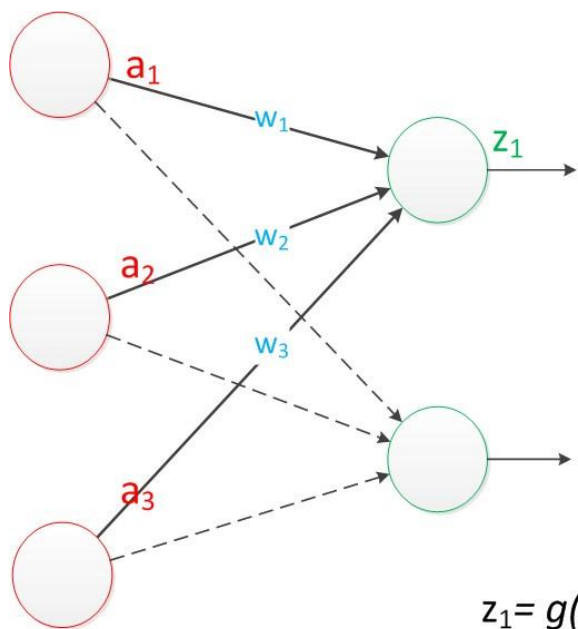
# 神经元

- 如果我们将神经元图中的所有变量用符号表示，并且写出输出的计算公式的话，就是下图。
- 可见  $z$  是在输入和权值的线性加权和叠加了一个函数  $g$  的值。 $g$  的作用是对  $\text{sum}$  进行取值范围的调整（本文不做具体介绍），后文将省略。

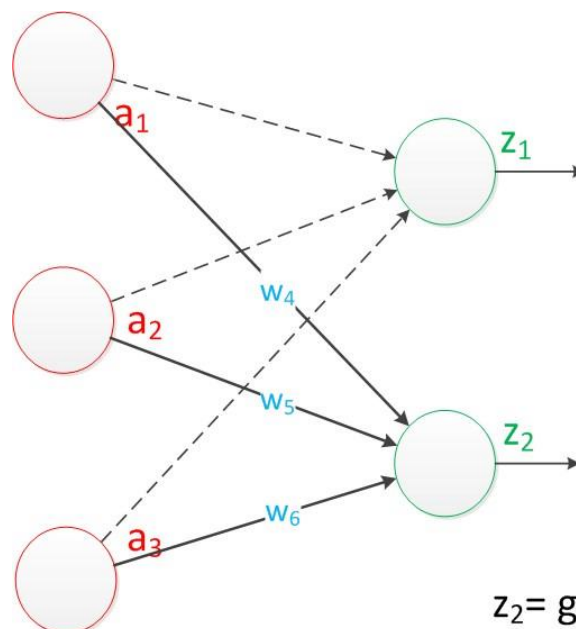


# 神经元

- 假如我们要输出不再是一个值  $z$ ，而是一个向量，例如  $(z_1, z_2)$ 。那么可以在输出层再增加一个“输出单元”。
- $z_1$  和  $z_2$  的计算方法和上一页类似。



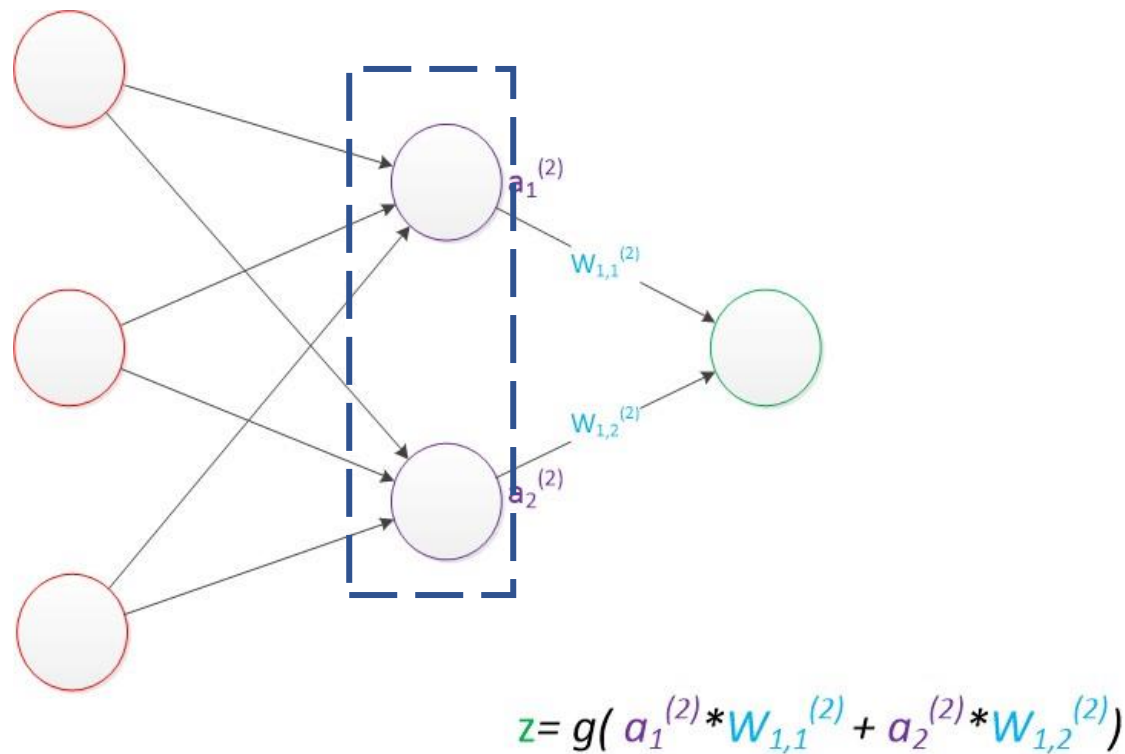
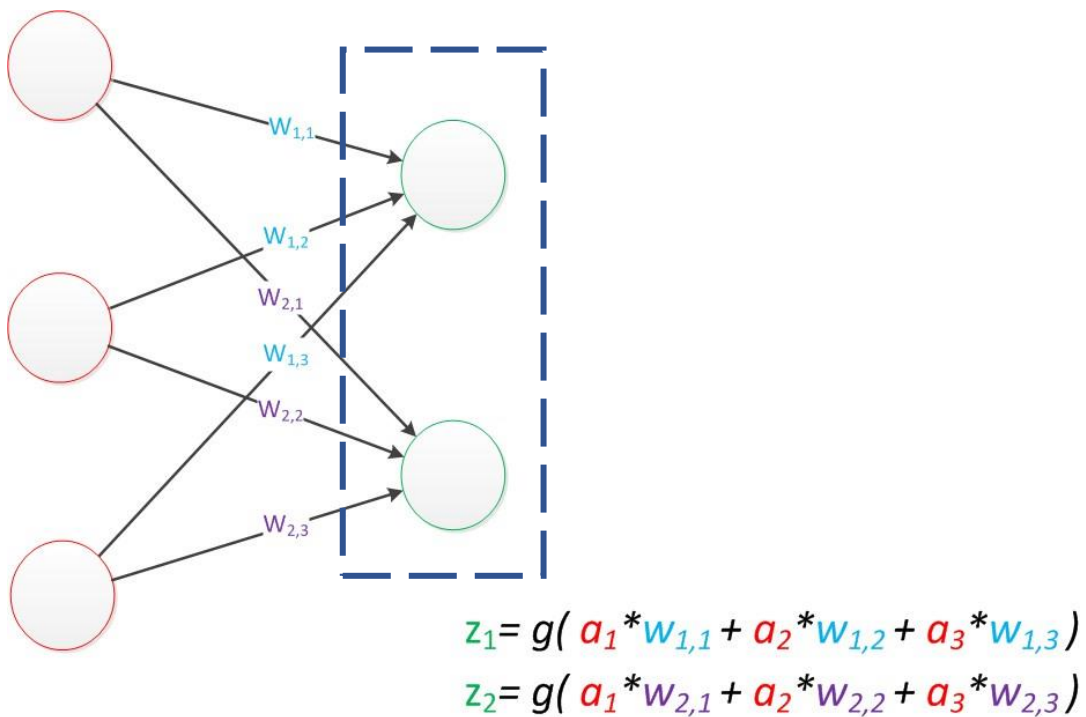
$$z_1 = g(a_1 * w_1 + a_2 * w_2 + a_3 * w_3)$$



$$z_2 = g(a_1 * w_4 + a_2 * w_5 + a_3 * w_6)$$

# 神经元

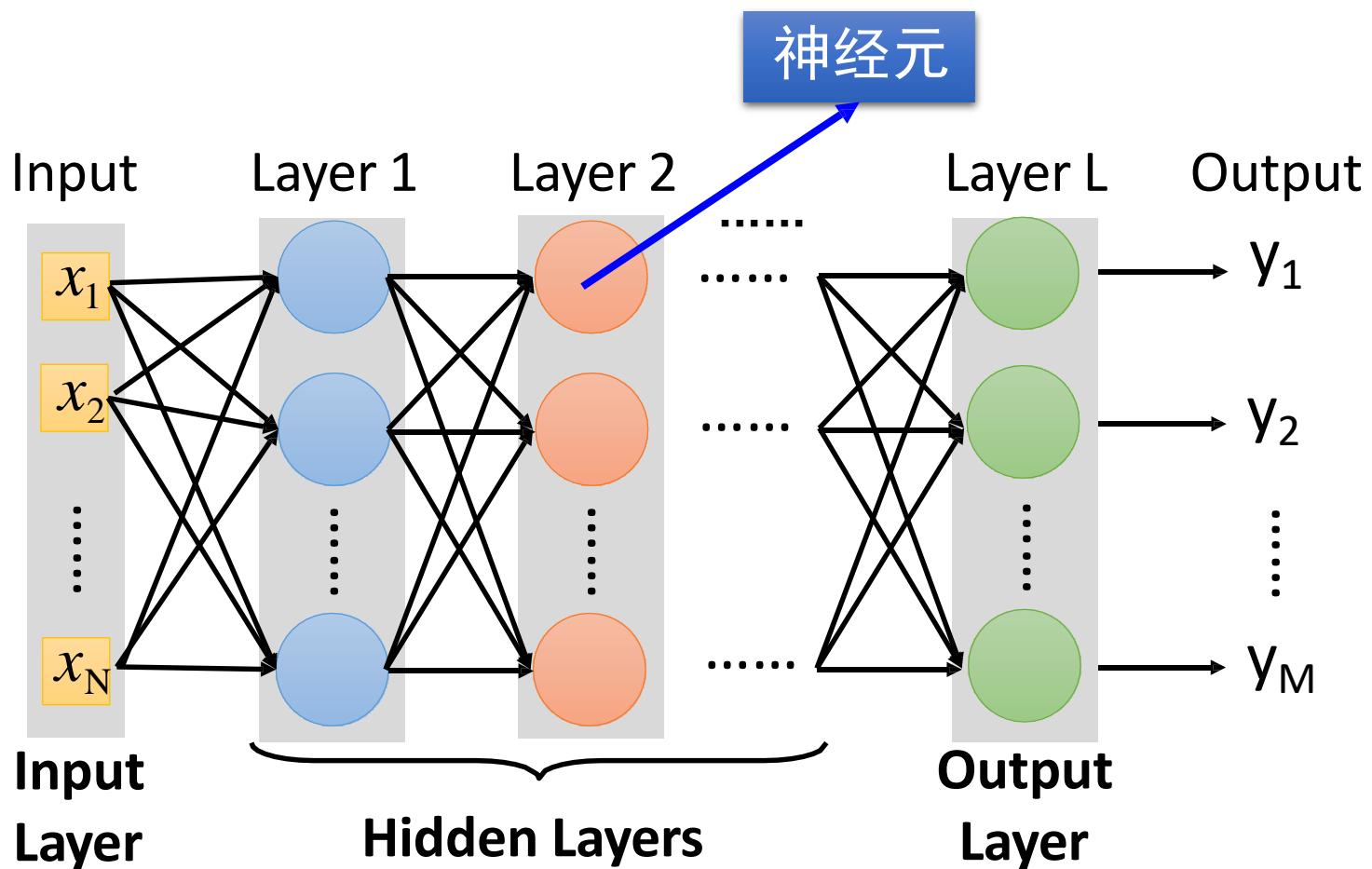
- 上文计算得到的  $z_1$  和  $z_2$  又可作为下一层的输入。





# 深度学习

- 现代神经网络的层数非常多（可能有50层甚至几百层）
- 对这种多层神经网络的训练，可以认为就是深度学习

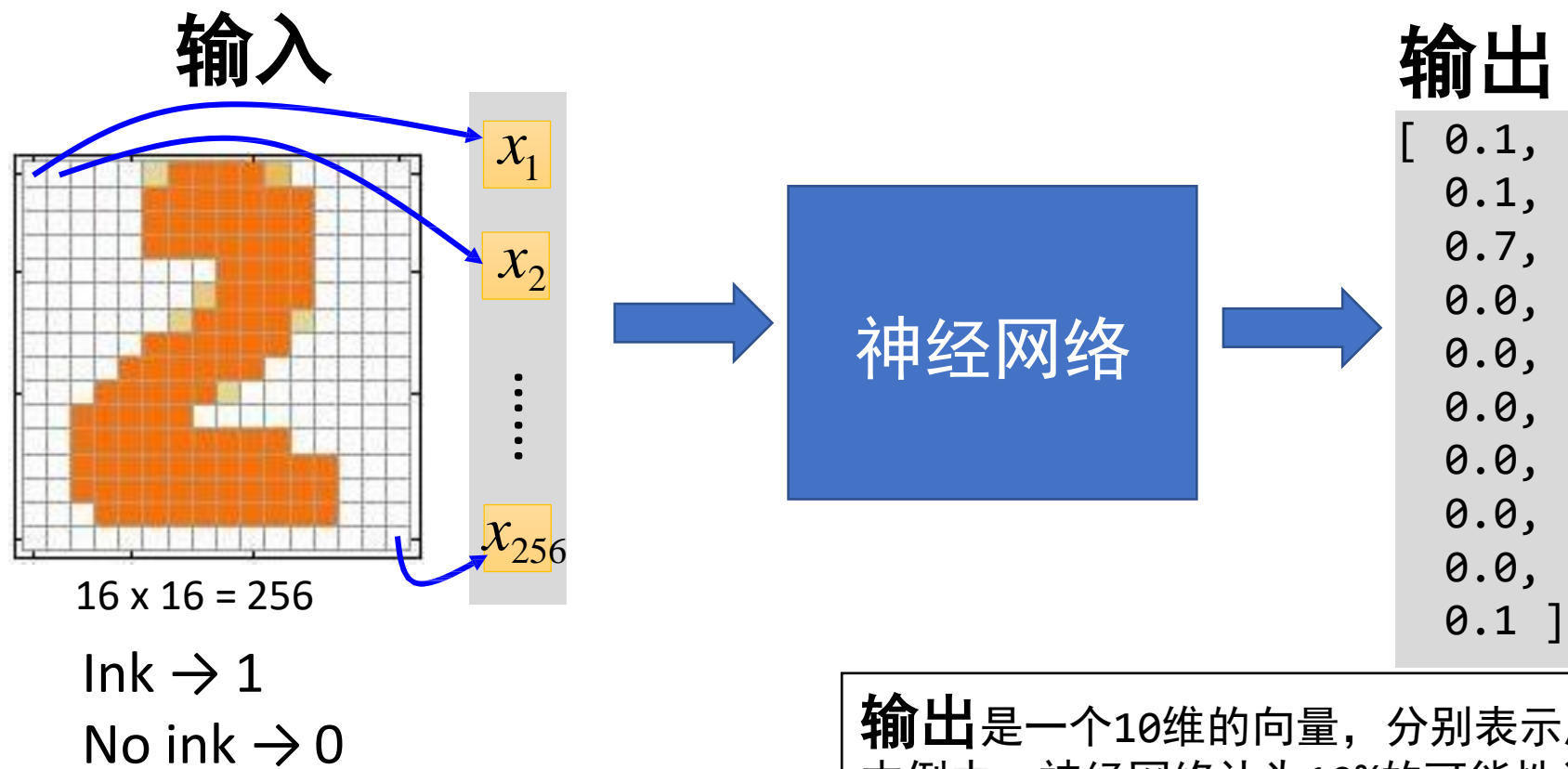


深度意味着很多隐藏层

# 深度学习

可以看出：深度学习是机器学习的一个子领域。

以手写数字的图像分类为例：输入图片可以看作一个256维的向量



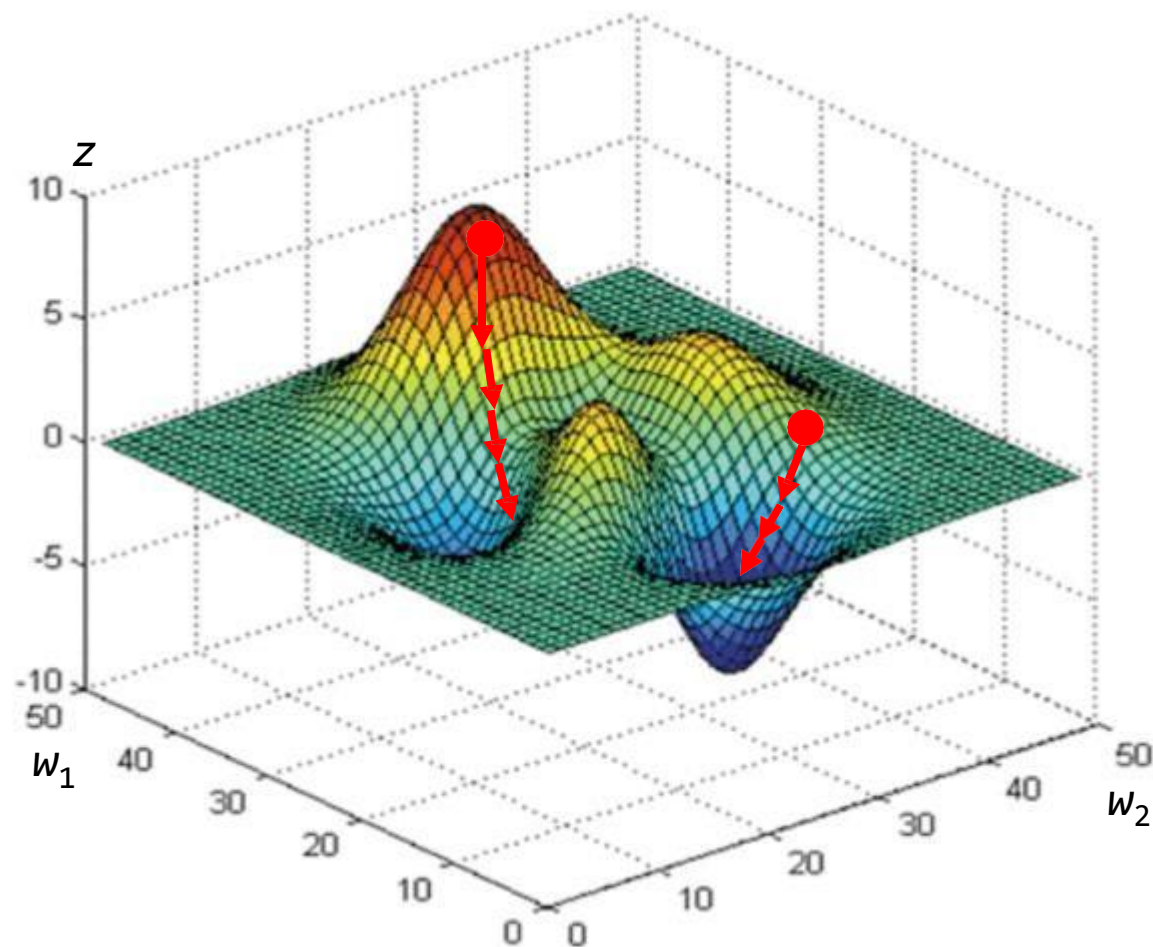
**输出**是一个10维的向量，分别表示属于第  $i$  类的概率。  
本例中，神经网络认为10%的可能性为“0”，10%的可能性为“1”，70%的可能性为“2”，10%的可能性为“9”。即，模型认为输入图片最有可能表示“2”。

# 模型训练

- 损失 (loss)：神经网络输出和目标之间的距离
- 以上文数字“2”为例子，  
其输出是 `output = [0.1, 0.1, 0.7, 0, 0, 0, 0, 0, 0.1]`;  
目标是 `target = [0, 0, 1, 0, 0, 0, 0, 0, 0]` # 以100%的概率认为是“2”
- 计算这两者之间的  $l_2$  损失函数：  
$$\text{Loss} = \sum_{i=0}^9 (\text{output}_i - \text{target}_i)^2$$
。显然， $\text{loss} = 0$  最佳。
- 此外，还有 `CrossEntropyLoss`、`FocalLoss` 等，计算方式有所不同，在此不做介绍。

# 模型训练

- 损失的计算，可以指导模型参数的修正方向（红色箭头）：
- 右图表示了  $\text{loss}$  与模型参数  $w_1$ 、 $w_2$  取值之间的关系。 $z$  轴表示损失函数的值， $w_1$ 、 $w_2$  轴表示模型的参数。
- 我们期望损失函数最小，则需要告知模型参数是该增大还是减小（图中红色箭头），使得  $\text{loss}$  尽可能降低。
- 学习率（learning rate, LR）：规定了参数修正的幅度。若损失函数要求  $w_1$ 、 $w_2$  变小，则 LR 可调节  $w_1$ 、 $w_2$  是应该减小0.1、1.0、还是10，即“迈的步子该多大”。LR 过小会导致  $\text{loss}$  下降不显著，LR过大会导致“走过头”，无法到达谷底。



# 深度学习框架

- 有很多工具可以帮助你快速实现深度学习



theano

libdnn

台大周伯威  
同學開發

Caffe



- 本实验采用目前最火热的 PyTorch。PyTorch 已经成为科研人员的首选深度学习框架之一，在人工智能、机器学习、计算机视觉、自然语言处理相关领域学术论文中的使用率不断增长。

PYTORCH

# 实验准备

- 到这里，我们初步了解了神经网络和深度学习。更多参考资料：

<https://www.cnblogs.com/subconscious/p/5058741.html>

<https://zhuanlan.zhihu.com/p/88399471>

<https://www.zhihu.com/question/26006703/answer/536169538>

- 接下来我们将尝试使用 PyTorch 进行深度学习。
- 别担心，如果你对上述内容还有疑问，应该不会对接下来的实验有太多影响。只需记住：深度学习并不神秘，神经网络也只是大量节点的加权求和。
- Training set: 训练集，所有的训练数据
- Epoch: 训练轮次。深度学习需要对 Training set 遍历多遍，每一遍叫做一个训练轮次。
- Batch: 批。在**每个训练轮次**中，由于训练集可能很大，无法一次性放入神经网络计算得到 Prediction，因此需要分批输入。在每个 Epoch 中，若每次将128张图片输入网络，则认为 Batch size = 128。每128张图片构成一个 Batch。

# 练习1：初等函数拟合

- 利用深度学习实现对非线性函数  $f(x) = x^2 + 2\sin(x) + \cos(x-1) - 5$  的拟合（见 exp1.py）。本实验将帮助你理解深度学习的基本流程：
  1. 随机读入一个 Batch 的数据 (x, target\_y)。  
`[len(x) == len(target_y) == batch size, 即 x[i] 对应 target_y[i]]`。
  2. 把 x 输入模型，得到 predicted\_y。 `[predicted_y == model(x)]`
  3. 比较 predicted\_y 和 target\_y，计算误差 (mse)。
  4. 根据误差，告知模型各参数的修正趋势（各参数是该变大还是减小） `[loss.backward()]`，并依据此趋势更新参数 `[optimizer.step()]`。
  5. 回到 1。

# 练习1：初等函数拟合

- 注：以下实验不要对 `NUM_TRAIN_SAMPLES` 进行改动。
- 1. 不对代码其他部分进行改动，更改参数 `NUM_TRAIN_EPOCHS` 为 100, 1000, 10000, 50000, 你发现拟合得到的曲线有什么变化？
- 2. 固定 `NUM_TRAIN_EPOCHS = 1000`，更改参数 `LEARNING_RATE` 为 1, 0.1, 0.01, 0.001, 你发现拟合得到的曲线有什么变化？
- 3. 自定义一个函数  $f(x)$ ，调整合适的参数，使得模型拟合效果尽可能好。

## 提交要求：

1. 自定义  $f(x)$  后的 `exp1.py`，注意设定合适的参数；并提交对应的 `model.pth` 文件。
2. 报告中附上对前三小问的实验结果与分析。



## 练习2: CIFAR-10 图片分类

- 利用深度学习对 CIFAR-10 (<http://www.cs.toronto.edu/~kriz/cifar.html>) 图片进行分类。  
(见exp2.py)
- CIFAR-10数据集由10个类的60000个32x32彩色图像组成, 每个类有6000个图像。有50000个训练图像和10000个测试图像。
- 注意: 训练时, 我们只能使用50000个训练图像。这样, 在测试时模型没见过这10000张测试图片, 其准确率评估才准确。
- 本实验中, 我们采用简单的 ResNet20 (一个20层的神经网络) 作为模型, 在50000个训练图片上对其参数进行更新。经过10轮训练后, 在10000张测试图片上的准确率可以达到70%以上。
- 事实上, 现在最好的模型 (不是使用 ResNet20) 可以达到95%以上的准确率, 这一数字远远好于随机猜测的准确率——10%。当然, 这些模型训练的轮数远多于10轮, 模型也要比Resnet20 复杂。
- 一个多年前的排行榜:  
<https://www.kaggle.com/c/cifar-10/leaderboard>

飞机

汽车

鸟

猫

鹿

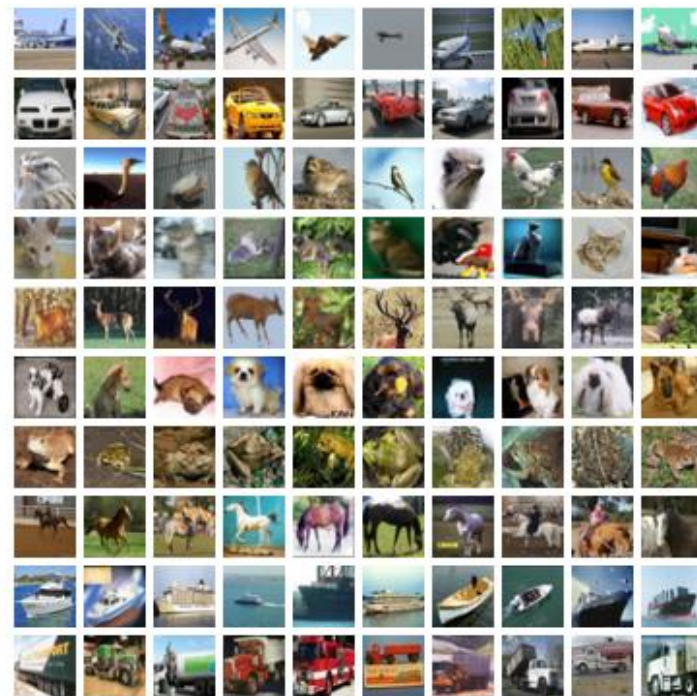
狗

青蛙

马

船

卡车



## 练习2: CIFAR-10 图片分类

- 补充exp2.py的代码,使得程序可以完整运行。
- 使用 Resnet20 模型,训练一个 CIFAR-10 的分类器。  
推荐训练策略: 以0.1的 LR 训练5个 Epoch, 再以0.01的 LR 训练5个 Epoch。
- 提示: 使用 `model.load_state_dict(torch.load(restore_model_path)['net'])` 可以加载已保存的模型,继续训练。训练可能会需要很长的时间,请耐心等待。  
若笔记本性能有限,可以加载我们已经预训练了5轮的 `pretrain_model.pth`, 这样可以跳过LR = 0.1 的训练阶段, 直接进行 LR = 0.01 的5轮训练。
- 思考: Train acc 和 Test acc 有什么关联和不同? 在 LR 从 0.1 变到 0.01 后, acc 发生了什么变化? 为什么?

### 提交要求:

- 补充完整的 exp2.py (使得算法可以按照给定的推荐训练策略,从第0个 Epoch 开始自动完成训练), 以及最终模型 final.pth。
- 在报告中展示10个 Epoch 的 Test acc 变化趋势。