TechX 深度强化学习课程（三）

何舜成

宽带网数字媒体实验室
清华大学自动化系

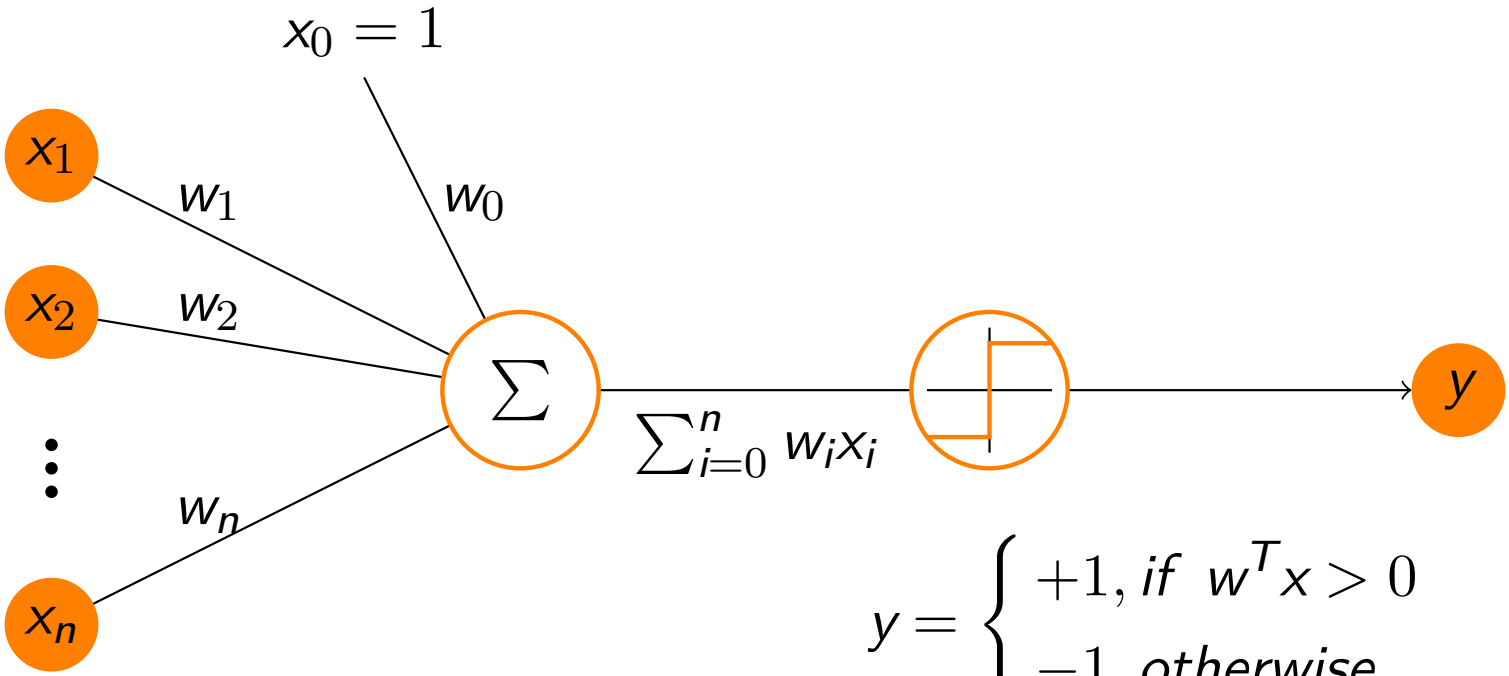
2018 年 7 月 27 日

目录

神经网络

感知机

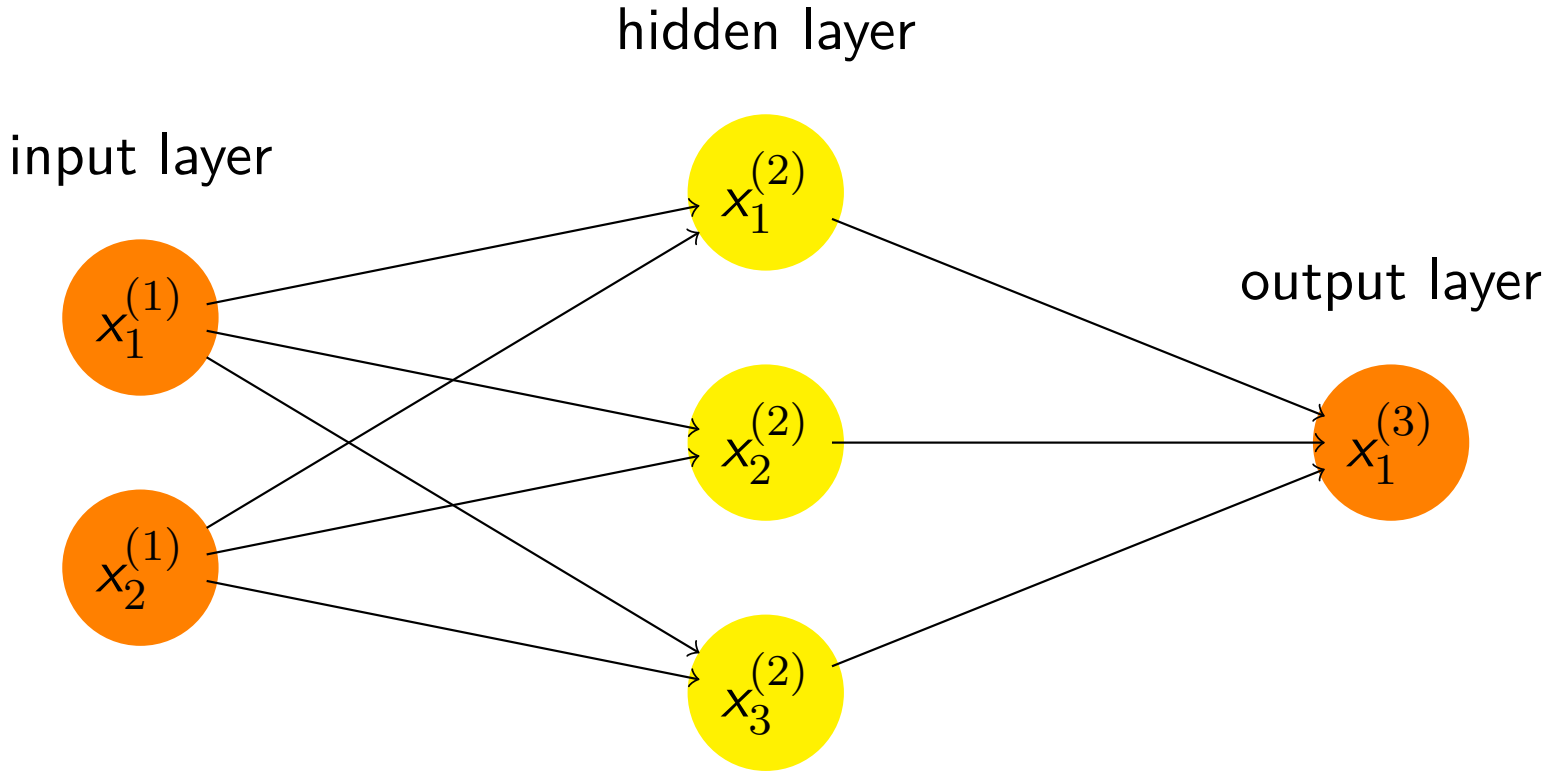
感知机（Perceptron）类似于以往的线性模型，只是在输出端加上了一个非线性单元。



$$y = \begin{cases} +1, if \ \ w^T x > 0 \\ -1, otherwise \end{cases}$$

人工神经网络

感知机可以作为一个基础单元，层层累加，构成更复杂的分类器，如人工神经网络（ANN）。



$$\boldsymbol{x}^{(l+1)} = h((\boldsymbol{W}^{(l)})^\top \boldsymbol{x}^{(l)}) \tag{1}$$

其中 $h(\cdot)$ 是非线性函数。

激活函数

每个神经元输出的所经过的 $h(\cdot)$ 称为**激活函数**，以前通常选择 Sigmoid 函数。由于 Sigmoid 可导，如此构成的神经网络将是端到端可导的。如此可以利用梯度下降法优化目标函数。

Sigmoid 之外，还有几种常用的激活函数：

1. 双曲正切函数（tanh）

$$h(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2}$$

2. 线性整流函数（ReLU）

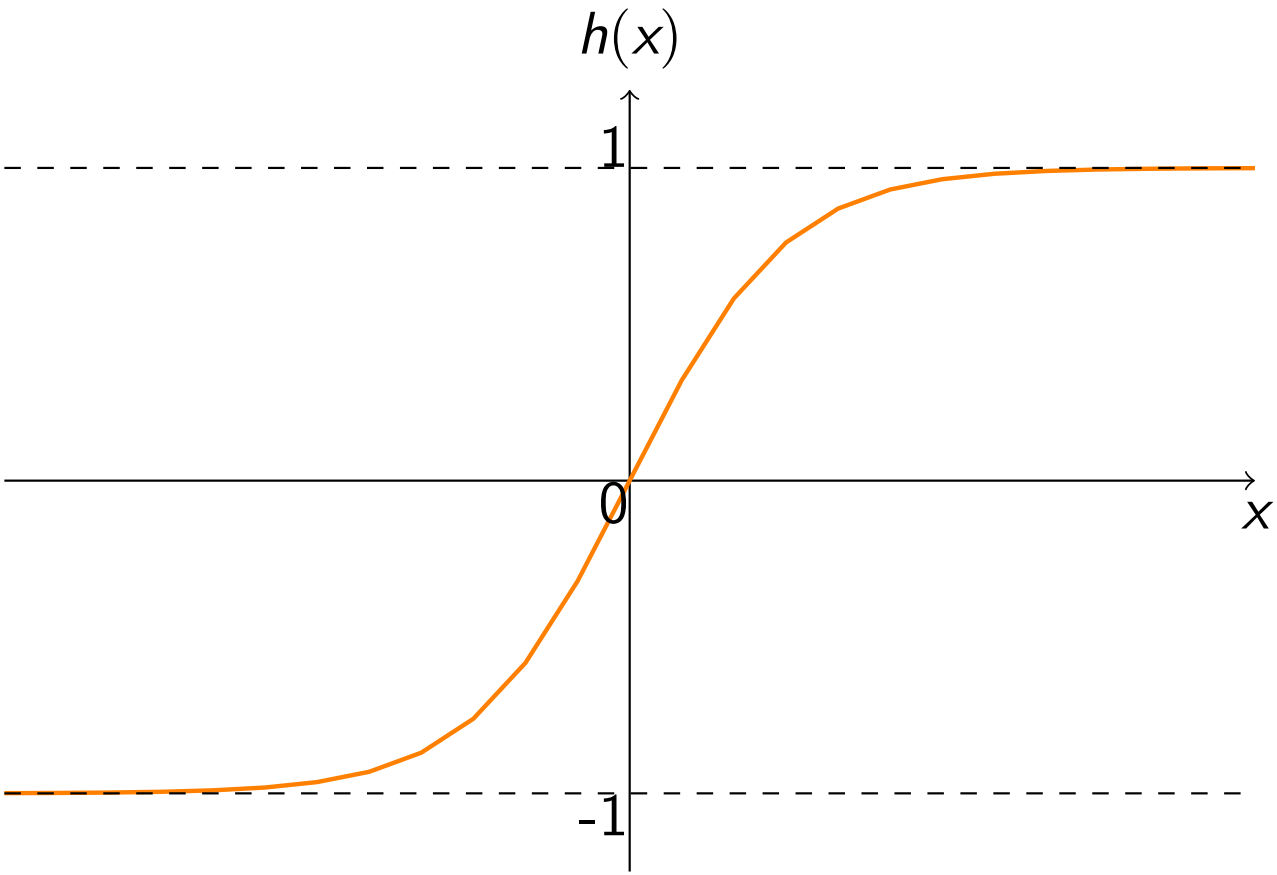
$$h(x) = \max(0, x) \tag{3}$$

3. Softmax 函数

$$h(\boldsymbol{x})_i = \sigma(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \tag{4}$$

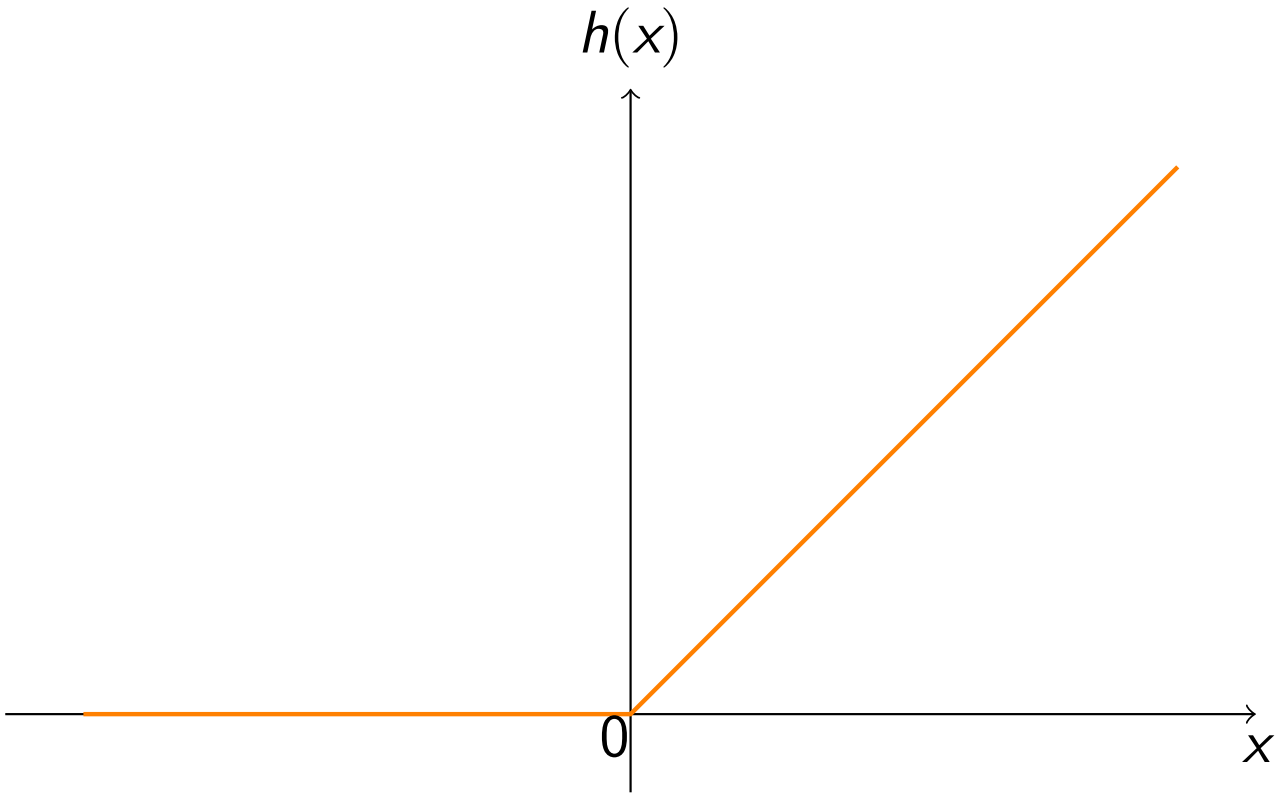
激活函数

双曲正切函数类似 Sigmoid 函数，将实值 $[-\infty, +\infty]$ 压缩到 $[-1, 1]$ 范围内。



激活函数

ReLU 函数非常简单，起到的作用是削平低于 0 的信号，通过高于 0 的信号。该函数不可导但次梯度存在。



激活函数

Softmax 函数由 Sigmoid 函数扩展而来。输入是 n 维向量 $(x_1, \ldots, x_n)^\top$，先进行指数运算得到 $(e^{x_1}, \ldots, e^{x_n})$，然后进行归一化，即同时除以 $\sum_{i=1}^{n} e^{x_i}$。当输入向量是 2 维时，退化为 Sigmoid 函数。

Softmax 有以下特性：

- 输入向量的最大分量对应的输出接近 1，其余分量接近 0，近似起到 arg max 的作用
- 函数可导
- 输出向量分量之和为 1，对应一个概率分布，通常用于多分类问题的输出

反向传播算法与 δ 规则

在深度学习快速发展之前，人工神经网络通常只有输入层、隐藏层和输出层。但这种神经元的堆叠方式显然是可以扩展到更多层的。给定一个 L 层的网络，前 $L-1$ 层都对应有权值矩阵 $\boldsymbol{W}^{(i)}, i = 1, \ldots, L-1$。层与层之间的递推关系如下

$$\boldsymbol{x}^{(l)} = h(\boldsymbol{u}^{(l)}), \boldsymbol{u}^{(l)} = (\boldsymbol{W}^{(l-1)})^{\top} \boldsymbol{x}^{(l-1)} \tag{5}$$

而 $\boldsymbol{x}^{(1)} = \boldsymbol{x}$ 即为网络的原始输入。

给定样本的 groundtruth 为 $\boldsymbol{t}$，记损失函数为 $E(\boldsymbol{x}^{(L)}, \boldsymbol{t})$。

反向传播算法与 δ 规则

将损失函数往前展开一项，得

$$E(\boldsymbol{x}^{(L)}, \boldsymbol{t}) = E(h((\boldsymbol{W}^{(L-1)})^\top \boldsymbol{x}^{(L-1)}), \boldsymbol{t}) \tag{6}$$

利用复合函数求导的链式规则，得

$$\frac{\partial E}{\partial \boldsymbol{W}^{(L-1)}} = \boldsymbol{x}^{(L-1)} (h'(\boldsymbol{u}^{(L)}) \star \frac{\partial E}{\partial \boldsymbol{x}^{(L)}})^\top \tag{7}$$

$\star$ 表示逐项相乘。如果定义

$$\boldsymbol{\delta}^{(L)} = h'(\boldsymbol{u}^{(L)}) \star \frac{\partial E}{\partial \boldsymbol{x}^{(L)}} \tag{8}$$

反向传播算法与 δ 规则

那么就有

$$\frac{\partial E}{\partial \boldsymbol{W}^{(L-1)}} = \boldsymbol{x}^{(L-1)}(\boldsymbol{\delta}^{(L)})^{\top} \tag{9}$$

如果接着展开，可以得到递推式

$$\boldsymbol{\delta}^{(l)} = h'(\boldsymbol{u}^{(l)}) \star ((\boldsymbol{W}^{(l)})^{\top}\boldsymbol{\delta}^{(l+1)}), l = L-1, \ldots, 2 \tag{10}$$

以及

$$\frac{\partial E}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{x}^{(l)}(\boldsymbol{\delta}^{(l+1)})^{\top}, l = L-2, \ldots, 1 \tag{11}$$

至此，我们可以利用 $\boldsymbol{\delta}$ 中间变量递推地求出每一层网络权值的梯度，这种算法称为**反向传播**。

多分类问题损失函数

多分类问题的分类器输出的是所有类别上的一个概率分布，而 groundtruth 可以视作某一类上概率为 1 的分布，衡量分类器的损失通常使用交叉熵（令 $\boldsymbol{y} = \text{softmax}(\boldsymbol{x}), \boldsymbol{x} = \boldsymbol{x}^{(L)}$）：

$$H(\boldsymbol{t}, \boldsymbol{y}) = -\sum_{i=1}^{n} t_i \log(y_i) \tag{12}$$

$0 \log 0$ 定义为 0。因此如果 groundtruth 标签为 j，那么交叉熵损失还可以写为

$$L = -\log(y_j) = -\log\left(\frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}\right) = \log\left(\sum_{k=1}^{n} e^{x_k}\right) \tag{13}$$

梯度为（当且仅当 $i = j$ 时 $\delta_{ij} = 1$，其余情况为 0）

$$\frac{\partial L}{\partial x_i} = y_i - \delta_{ij} \tag{14}$$

随机梯度下降

最速下降法的损失函数定义为分类器在所有样本上的误差（平方误差或交叉熵）之和。在神经网络的优化中，一般用随机梯度下降（Stochastic Gradient Descent，SGD）：

1. 从 N 个样本 $\mathbb{X}$ 中等概率随机抽取 m 个（$m \ll N$）样本 $\mathbb{X}_m$，称作 mini-batch

2. 计算这个 mini-batch 上的误差 $J(\mathbb{X}_m, \boldsymbol{W})$

3. 计算梯度并更新权值 $\boldsymbol{W}' = \boldsymbol{W} - \epsilon \nabla J(\mathbb{X}_m, \boldsymbol{W})$

重复以上步骤直至误差收敛。

随机梯度下降

由于神经网络激活函数的非线性，优化的损失函数极度非凸，会存在大量局部极小值。进行梯度下降的过程中需要避免陷入局部极小值。

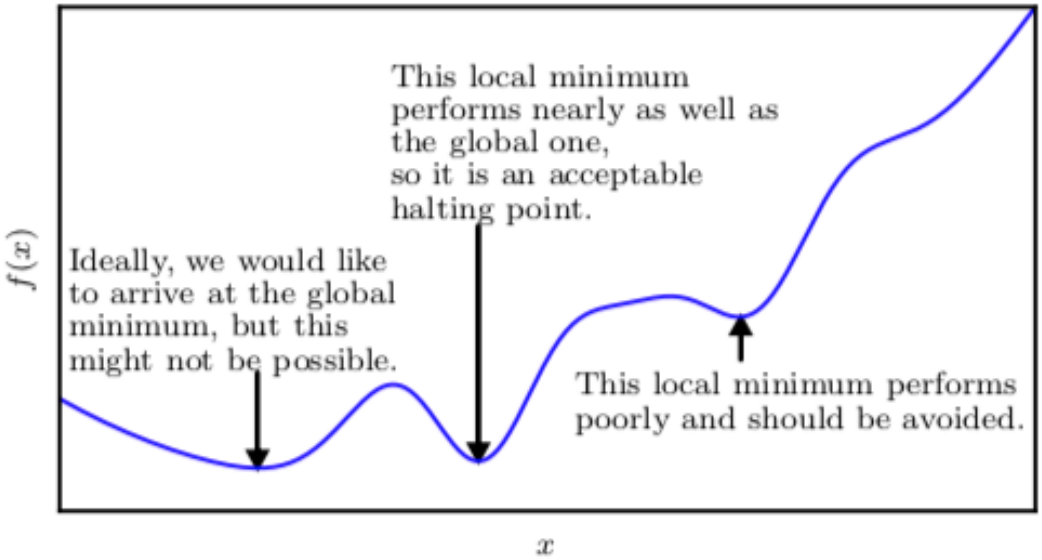


图 1: 多个局部极小值示意

随机梯度下降

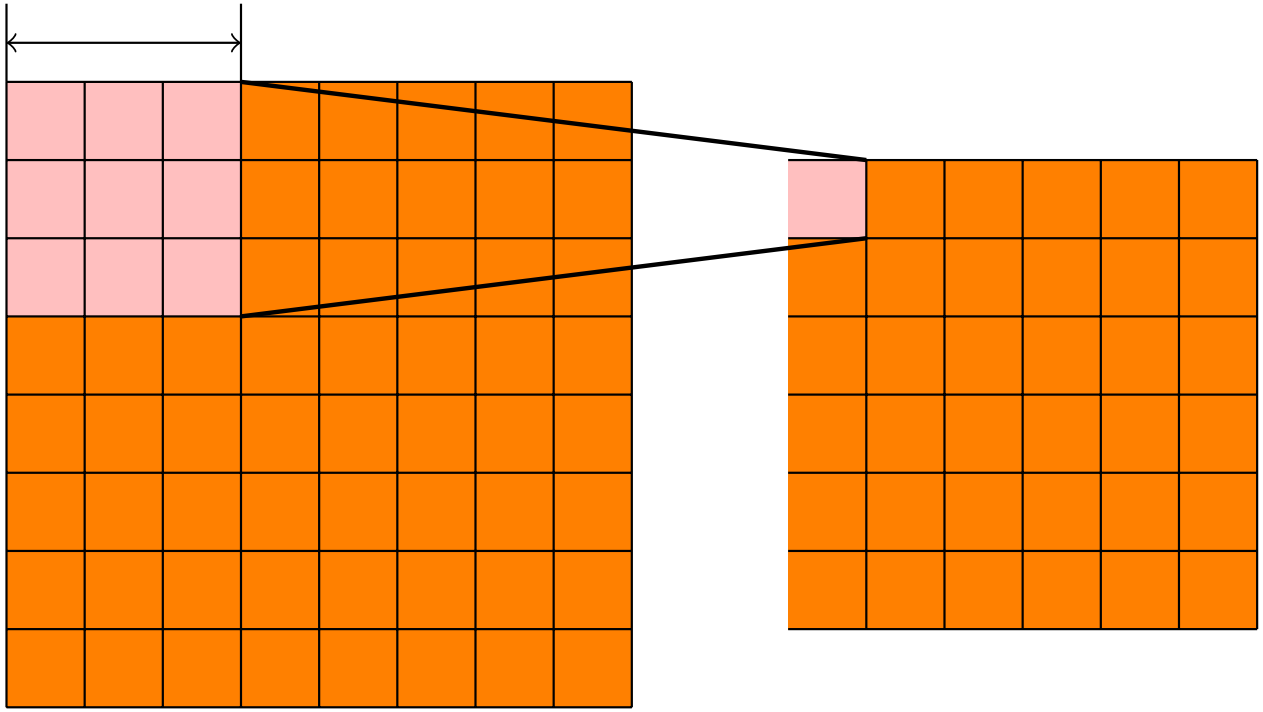
对比原始的梯度下降法，随机梯度下降会有更多优点：

- 样本抽样带来随机性，将改善陷入局部极小值的情况
- 在训练样本数庞大的情况下，将减小单次梯度计算和存储压力

卷积神经网络

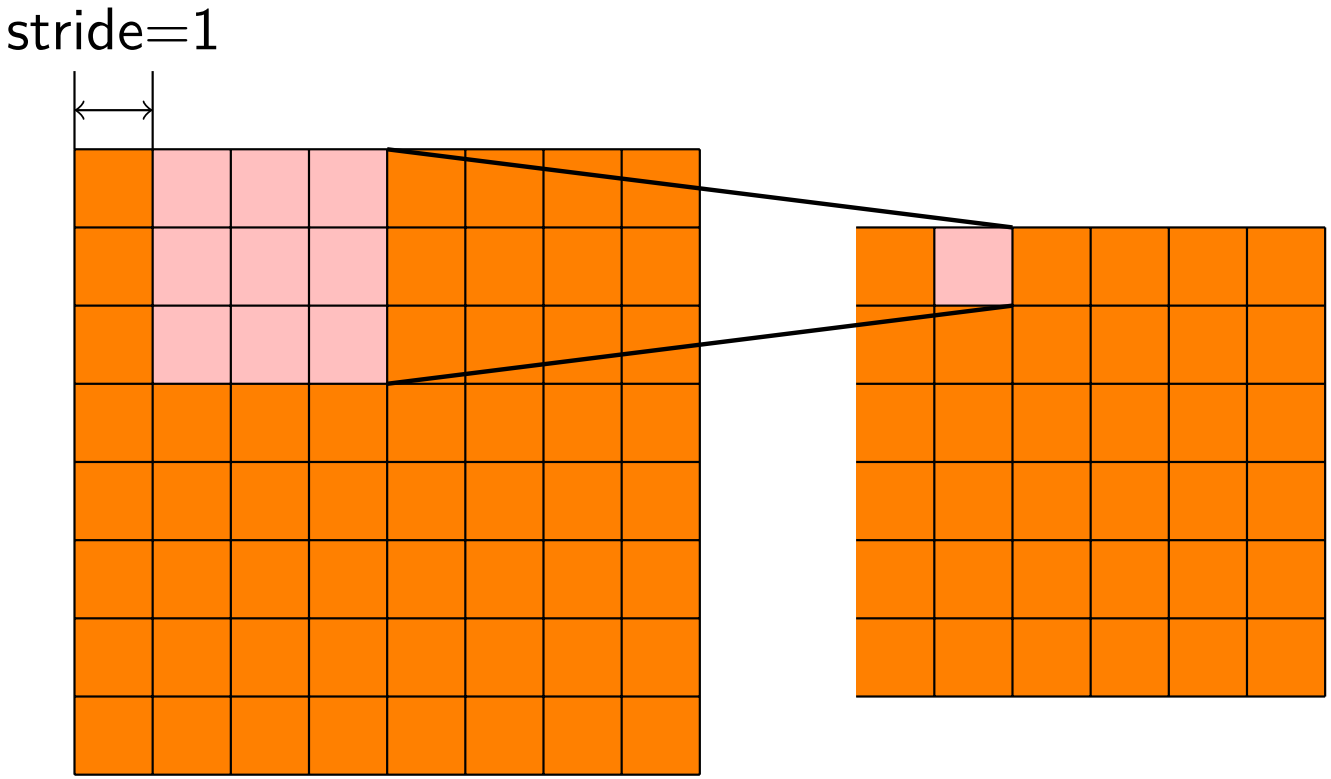
图像卷积

kernel size$=3 \times 3$



$$g_{ij} = \sum_{s=i}^{i+2} \sum_{t=j}^{j+2} h_{st} k_{st}$$

图像卷积



$$g_{ij} = \sum_{s=i}^{i+2} \sum_{t=j}^{j+2} h_{st} k_{st}$$

图像卷积

卷积作为特征提取的方法，广泛应用于传统图像处理领域。应用不同的卷积核，可以提取出不同的图像特征。

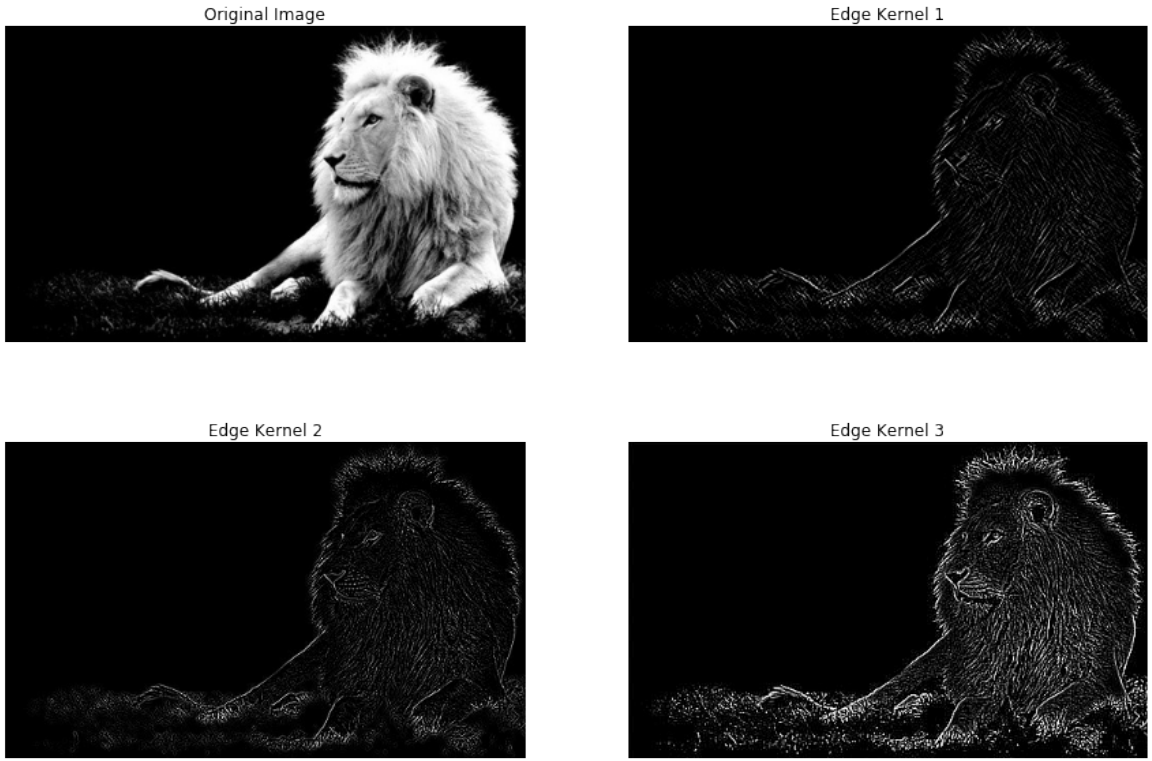
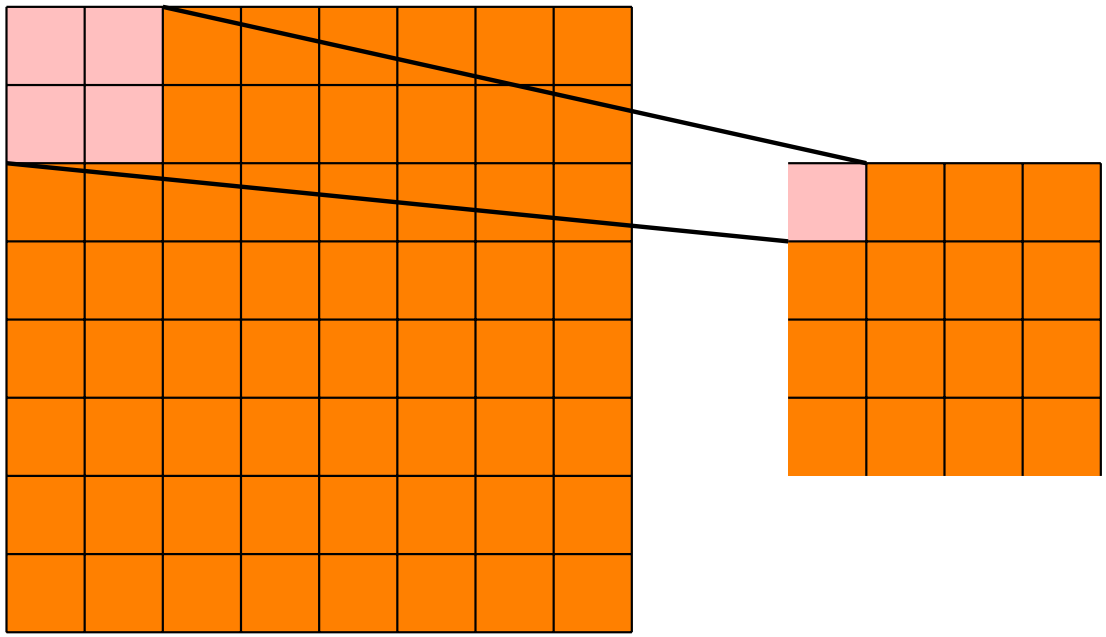


图 2: 在不同卷积核作用下的效果

图像池化

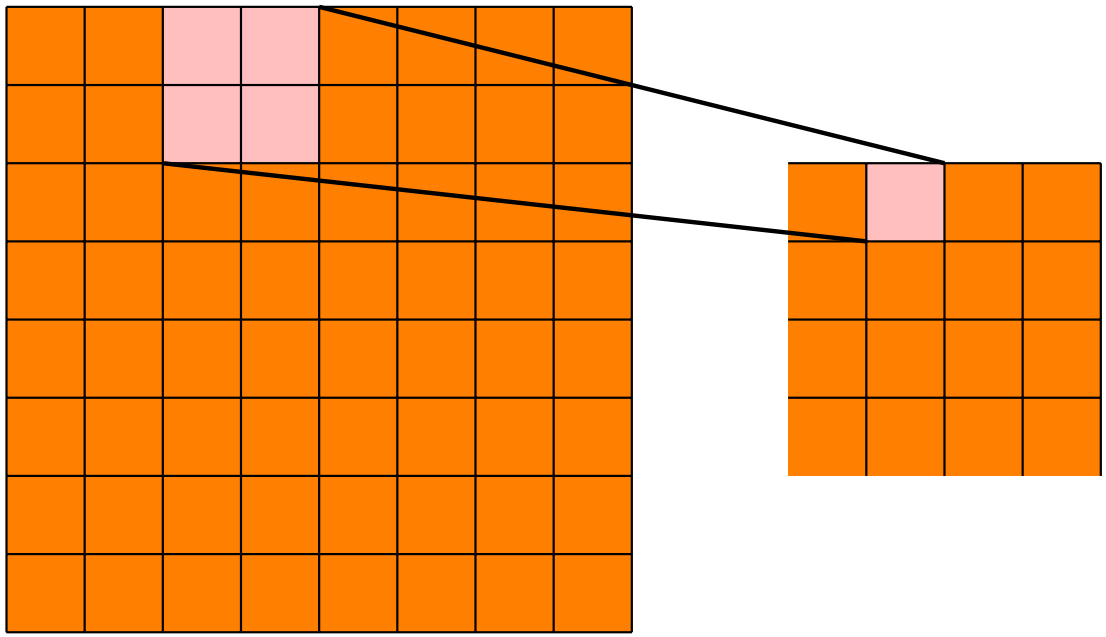
特征提取后，一般还需要用池化（压缩）的方法将特征集中起来。



$$g_{ij} = \max\{h_{2i,2j}, h_{2i+1,2j}, h_{2i,2j+1}, h_{2i+1,2j+1}\}$$

图像池化

特征提取后，一般还需要用池化（压缩）的方法将特征集中起来。



$$g_{ij} = \max\{h_{2i,2j}, h_{2i+1,2j}, h_{2i,2j+1}, h_{2i+1,2j+1}\}$$

卷积神经网络

将卷积的方法迁移到神经网络上来，就形成了为人熟知的卷积神经网络（Convolutional Neural Networks，CNN）。早在 1998 年，Yann LeCun 等人就提出一种 LeNet 的网络结构，用在 MNIST 数据集上进行数字分类。

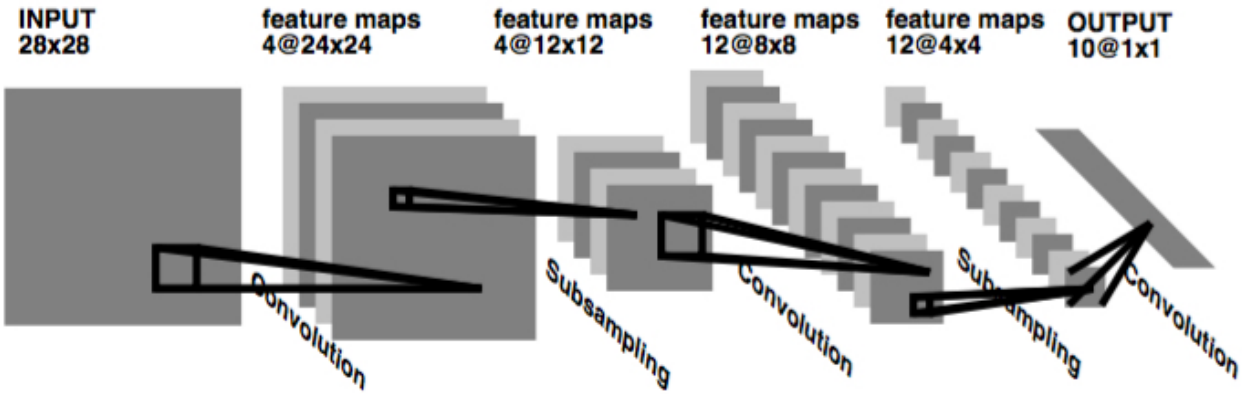


图 3: LeNet 网络结构

卷积神经网络

MNIST 是一个手写数字的数据库（Mixed National Institute of Standards and Technology ），共有 10 个类别。

图 4: 手写数字样例

卷积神经网络

以往利用图像卷积提取特征都是采用人工设计卷积核，来提取不同含义的特征。而 CNN 将卷积核的参数视为神经网络权值，通过训练来调整。池化层则将特征集中起来，一步步从区域特征集中到全局特征。

CNN 相比于全连接神经网络有独特的优点：

- 通过权值共享，极大减少网络参数量，提升泛化能力和计算速度
- 卷积操作关注局部关联，去除不必要的长程关联
- 网络结构具有物理含义，可进行可视化
- 平移不变性

实用技术

加入动量项

目前我们只学习了神经网络和卷积神经网络的理论原型，实际应用时有许多可以改进的技术，包括多种正则化手段以及优化技巧。

动量项的加入加速了损失函数的优化。基本想法是在计算参数更新方向时，保留一部分上一轮的更新方向。

设由 mini-batch 计算得到的梯度为 $\boldsymbol{g}$，学习率为 ϵ，那么以往的算法中，每一步的“速度” $\boldsymbol{v} = -\epsilon \boldsymbol{g}$，参数更新式为 $\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{v}$。

加入动量项后，$\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$。每一步新计算出来的梯度将影响之后所有的更新方向（速度项），影响因子呈指数衰减。

加入动量项

在某些情形下，动量项可以有助于保持一个稳定的更新方向，以及起到削减随机梯度下降法带来的高方差。

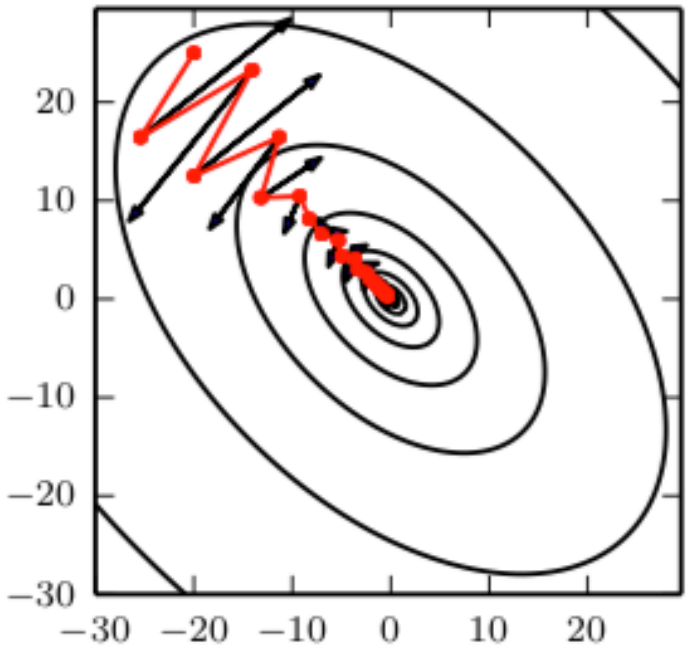


图 5: 在 Hessian 矩阵病态时动量项可以加快收敛

加入动量项

以往梯度下降更新的步长仅由梯度范数和学习率决定 $\epsilon \|\boldsymbol{g}\|$，加入动量项后，假设连续多步计算得到的梯度均接近 $\boldsymbol{g}$，那么更新速度将一直增大直至

$$\frac{\epsilon \|\boldsymbol{g}\|}{1-\alpha} \tag{15}$$

这样将加快收敛速度。

梯度截断

当目标函数出现类似悬崖的结构时，梯度往往会很大，以至于参数会因此跳到很远的区域。一般我们会对梯度的范数进行限制，截断那些过大的梯度。

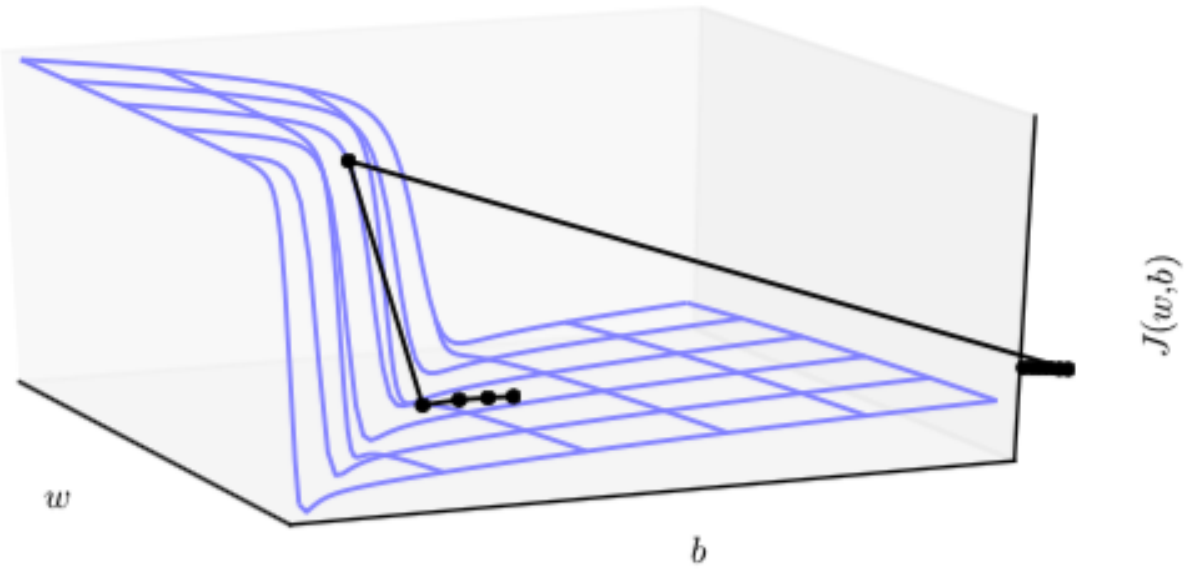


图 6: 梯度截断的效果示意

自适应学习率

神经网络中不同的参数往往并不等价，每个参数需要有自己的学习率。直觉上看，训练历史上拥有较大偏导的参数应当应用更小的学习率。

RMSProp 算法将梯度各分量按累积的平方梯度进行缩放。累积平方梯度初始值 $\boldsymbol{r} = \mathbf{0}$，累积时采用指数平滑平均

$$\boldsymbol{r} = \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \star \boldsymbol{g} \tag{16}$$

更新时按下式进行缩放

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\epsilon}{\delta + \boldsymbol{r}} \star \boldsymbol{g} \tag{17}$$

δ 是一个小的正数，ρ 决定指数平滑的记忆长度。

Dropout

Dropout 是一种防止过拟合的有效手段。在 CNN 中，大部分的参数并不是在靠前的卷积层，而是出现在靠后的全连接层。Dropout 会随机掩盖掉全连接层里 50% 的神经元以及对应的权值。实验证明 Dropout 对提高泛化能力有显著效果。

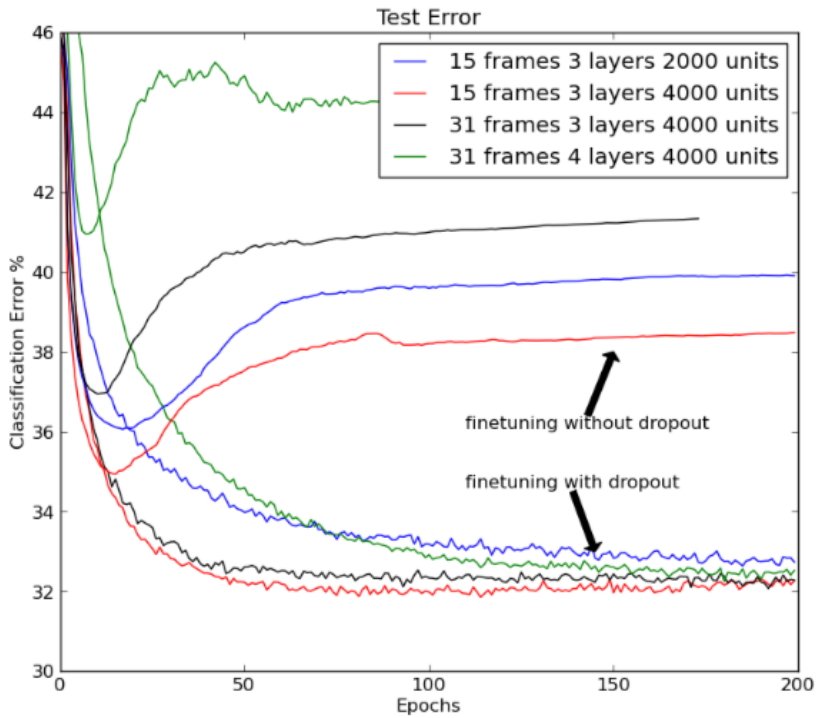


图 7: 使用与不使用 Dropout 效果对比

Batch Normalization

BN 算法对每个神经元在一个 mini-batch 上的响应做了归一化，避开了激活函数的饱和区域。

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

图 8: BN 算法

使用 TensorFlow

常用深度学习框架

科研与生产中常用的深度学习框架：

- Caffe 与 Caffe2——Facebook Research 主导，C++ 与 CUDA 编程，底层
- Torch 与 PyTorch——Facebook 主推，Lua 编程，偏顶层
- Theano——曾流行与科研界，Python 编程，偏底层
- TensorFlow——Google 主推，目前最为流行的框架，Python 编程，偏底层
- Keras——Python 编程，偏顶层

除此之外还有 CNTK 等深度学习框架。本课程将以 TensorFlow 为主。

TensorFlow 简介

TensorFlow（官网tensorflow.org）是目前使用最为广泛的开源的机器学习框架。

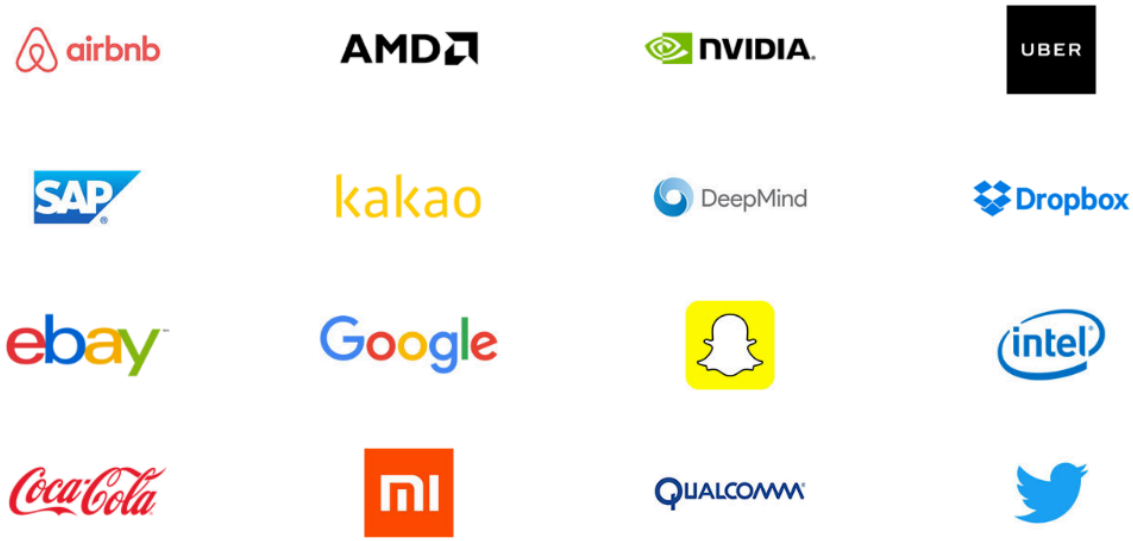


图 9: 使用 TensorFlow 的公司

TensorFlow 目前可在 mac OS、Ubuntu、Windows 系统上安装和运行。

使用虚拟环境

由于在不同的项目中可能使用不同的 Python、TensorFlow 及其他程序包，会导致冲突和不便，我们推荐使用虚拟环境如 virtualenv，但这里我们将使用 Anaconda。

Anaconda 是一个 Python 开发平台，预装有大部分 Python 开发会用到的工具包，并可以创建虚拟环境，指定 python 版本（`conda install python=x.x`）。

创建虚拟环境：`conda create -n $ENV_NAME`

进入虚拟环境：`source activate $ENV_NAME`

退出虚拟环境：`source deactivate`

在虚拟环境中仍然可以使用`pip`安装包，比如`pip install tensorflow==1.x.0`

TensorFlow 程序示例

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) =
    mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=[28, 28]),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

计算图

示例中的用法是 Keras 高度集成之后的，实际在科研和学习中，我们回使用更底层的 TensorFlow API，一方面熟悉深度神经网络的机制，另一方面为编程提供了更大的自由度。

神经网络中最基本的是网络结构。TensorFlow 中需要首先创建计算图（tf.Graph），然后提供所需的数据运行计算图（使用 tf.Session）。

图中的顶点是 Operations 或 ops，图的边是 Tensors（张量）。

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
sess = tf.Session()
print(sess.run(total))
```

计算图

计算图可以使用 TensorBoard 进行可视化，只要添加如下代码：

```
writer = tf.summary.FileWriter('.')
writer.add_graph(tf.get_default_graph())
```

writer 会将数据写入当前目录下的一个文件，只要在另外的终端中运行 tensorboard：

```
tensorboard --logdir .
```

就可以在 Chrome 浏览器中看到计算图。

数据输入

大多数情况下我们不会使用常量，而是使用不同的数据输入计算图。此时需要声明一些 placeholder，然后在运行时通过指定 feed_dict 的方式传入数据。注意：若不显式指定，placeholder 本身不限制数据的“形状”。

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y
print(sess.run(z, feed_dict={x: 3, y: 4.5}))
print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
```

层

神经网络中的 ops 通常会是“层”，如卷积层、池化层、全连接层等等。下面的代码应用了一个全连接层

```
x = tf.placeholder(tf.float32, shape=[None, 3])
linear_model = tf.layers.Dense(units=1)
y = linear_model(x)
```

参数`units=1`表明了这一层的输出是长为 1 的向量。placeholder 中`shape=[None, 3]`表示输入数据第一维不指定具体数目，第二维长度为 3。由此推断，y 的形状应为 [None, 1]。

层

全连接层参数需要初始化才能接着运行计算图。

```
init = tf.global_variables_initializer()
sess.run(init)
print(sess.run(y, {x: [[1, 2, 3],[4, 5, 6]]}))
```

训练

假设我们有一组数据要进行线性拟合：

```
x = tf.constant([[1], [2], [3], [4]],
    dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]],
    dtype=tf.float32)
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

print(sess.run(y_pred))
```

这样的结果是参数随机初始化的结果，我们需要进一步训练。

训练

首先要定义优化的目标函数，其次选定优化器，最后进行反复训练。

```
loss = tf.losses.mean_squared_error(labels=y_true,
    predictions=y_pred)
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)
```

注意：当sess.run()需要同时运行多个 ops 时，要用 tuple 的形式传参。

神经网络的生物学背景

神经元的结构

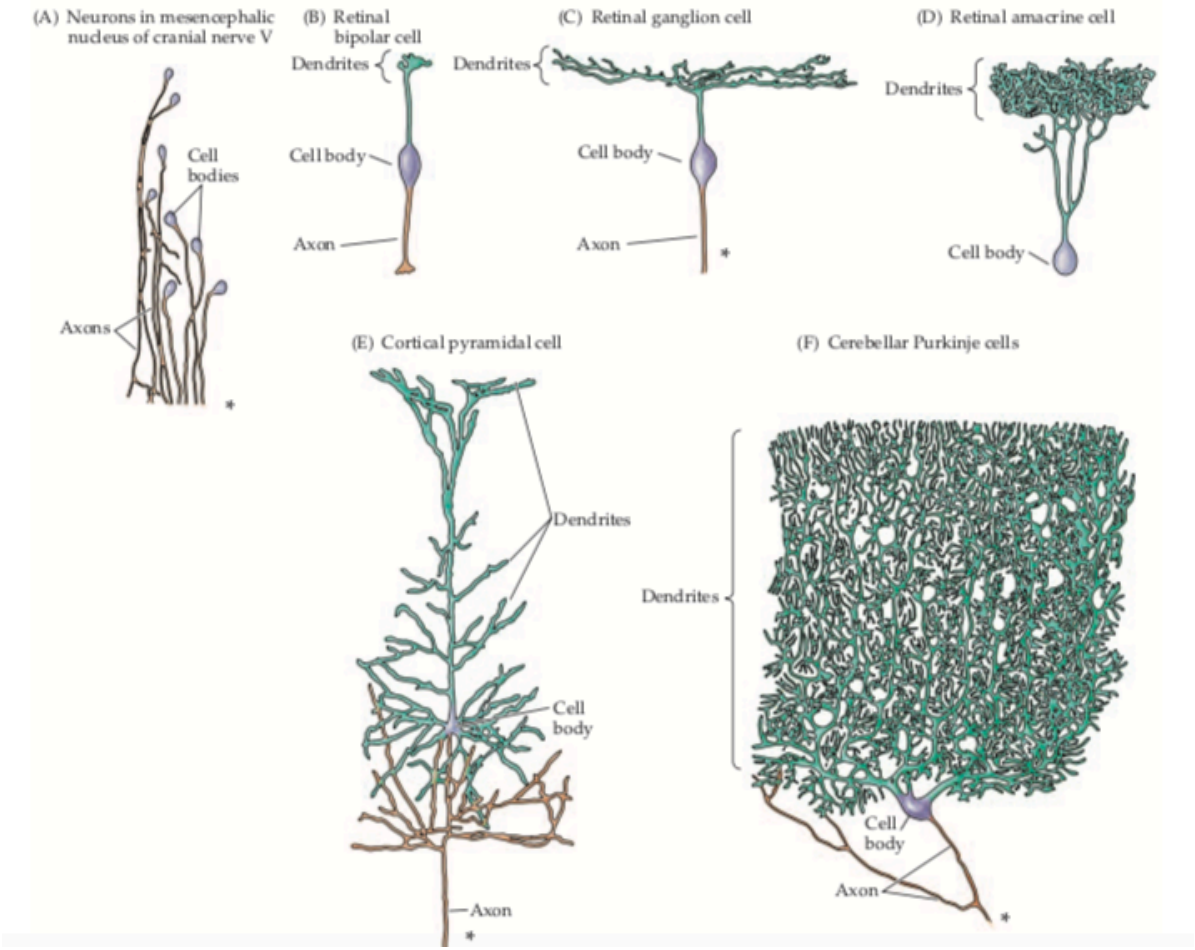


图 10: 不同的神经元形态

神经元的激活

静息电位、去极化、超极化与发放

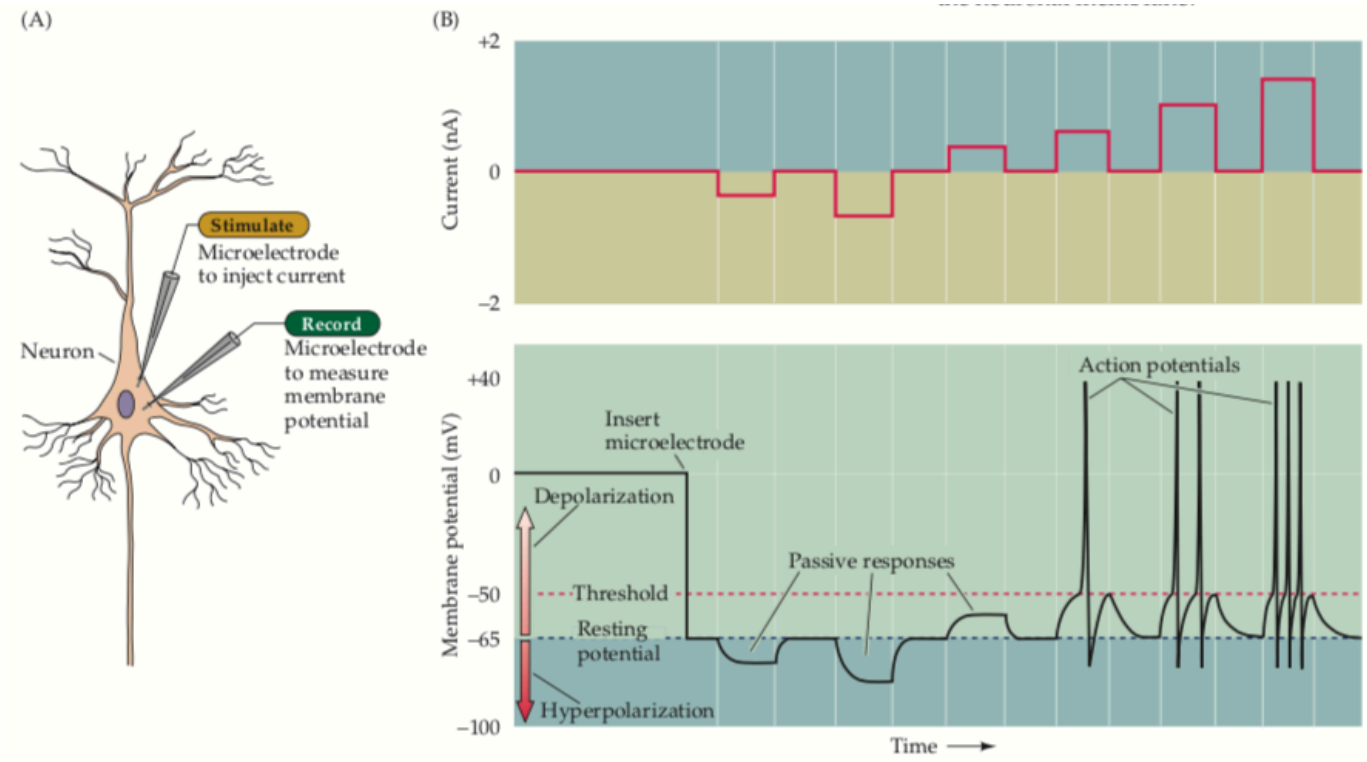


图 11: 神经元电信号变化

视网膜后的信号通路

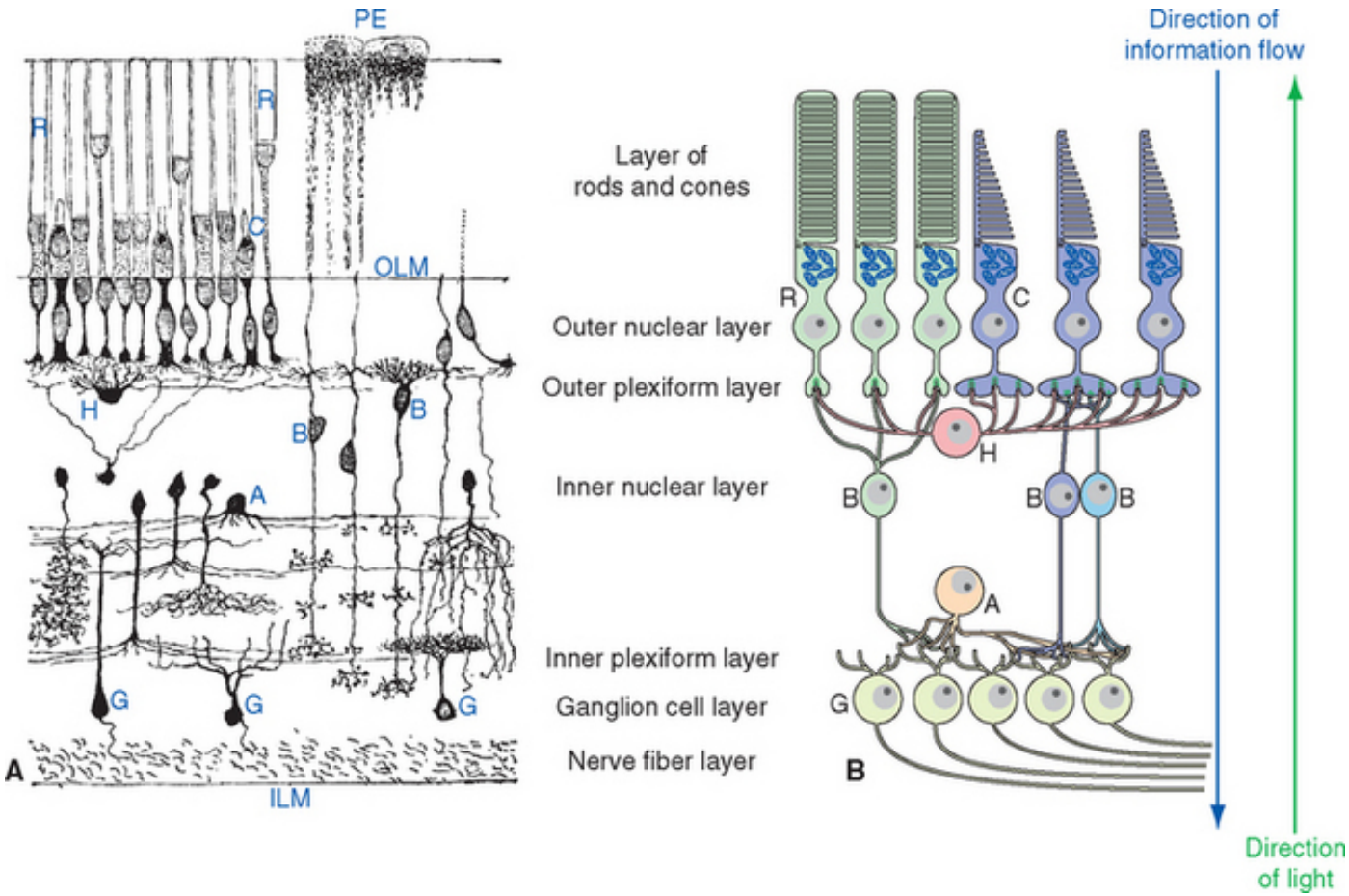


图 12: 视网膜下神经细胞分布

视觉皮层结构

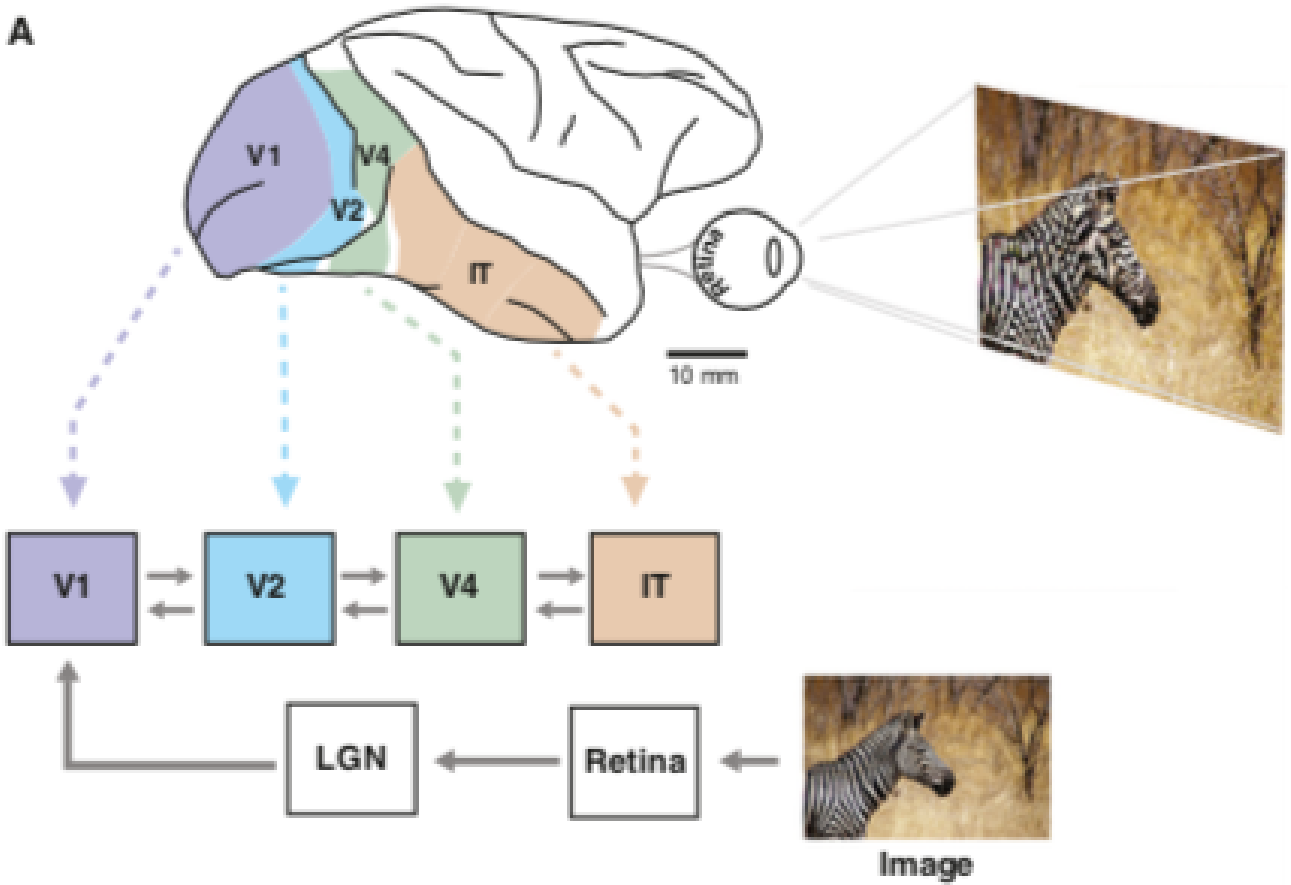


图 13: 视觉皮层的层次结构