大纲

1、线性层设计，SGD 优化器设计，损失函数设计，在 MNIST 上测试线性模型性能。
2、激活函数，Adam 优化器，在 MNIST 上测试神经网络性能。
3、卷积层和池化层，搭建能在 Cifar-10 上达到 70 测试集正确率的 CNN
4、RNN 模块设计，实现字符级 RNN 分类器
5、BatchNorm 和 Dropout 模块设计，收敛速度测试与比较
6、Embedding 模块和 Unpooling 模块设计，实现 Word2Vec 算法和卷积自编码器
7、神经网络模型的应用，CNN 可视化解释，GAN 图像生成和深度强化学习

仿射运算

多元函数线性变换，即对向量施行矩阵乘法和向量加法。在几何中，它是对一个向量空间进行一次线性变换并接上一个平移，变换为另一个向量空间。令 $\vec{x}$是d维向量，W是$d \times m$维矩阵，$\vec{b}$是m维偏置向量，则$\vec{y}$是仿射变换后得到的m维向量。

$$\vec{y} = \vec{x}W + \vec{b}$$

线性模型

令仿射运算的 m=1，则仿射运算模型对应从d维向量到标量y的映射。y可以是回归问题的输出，也可以是样本被分类为正样本或负样本的信念程度。根据定义的损失函数的不同将得到不同的线性模型，设 yhat 为模型输出，y 为样本的 ground truth

$$\hat{y} = \vec{x}\,\vec{w}^T + b$$

线性回归

$$loss(w, b, y) = (\hat{y} - y)^2$$

SVM $y \in \{-1,1\}$

$$loss(w, b, y) = max(1 - \hat{y} \cdot y, 0) + \lambda\frac{1}{2}\left|\left|w\right|\right|^2$$

Logitsic Regression $y \in \{-1,1\}$

$$loss(w, b, y) = log\big(1 + exp(-y \cdot \hat{y})\big)$$

神经网络

线性模型的表达能力受到限制，无法实现非线性回归和分类。为此引入多层仿射层+激活函数的神经网络。
单层的前向计算就是让向量经由仿射运算，再通过 pointwise 的激活函数。
单层神经网络

$$\vec{y} = \sigma(\vec{x}W + \vec{b})$$

多层神经网络

$$\overrightarrow{x_1} = \sigma(\overrightarrow{x_0}W_0 + \overrightarrow{b_0})$$
$$\overrightarrow{x_2} = \sigma(\overrightarrow{x_1}W_1 + \vec{b}_1)$$
$$\cdots$$
$$\vec{y} = \overrightarrow{x_{n-1}}W_n + \overrightarrow{b_n}$$

激活函数

除了 pointwise 的操作模式和非线性以外没有特别的要求，常用的激活函数有

sigmoid

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

ReLU

$$\sigma(x) = \max(0, x)$$

$$\sigma'(x) = int(x > 0)$$

Tanh

$$\sigma(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

$$\sigma'(x) = 1 - \sigma^2(x)$$

多分类

神经网络的多输出支持训练端到端的多分类模型，神经网络直接输出每个类别的概率。softmax 是有限项离散概率分布的梯度对数归一化，也被称为归一化指数函数。

$$p_j = \sigma(x)_j = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$$

函数的输入是从 K 个不同的线性函数得到的结果，而样本向量 x 属于第 j 个分类的概率为 softmax 的输出。它相当于让神经网络输出了一个离散条件概率分布

$$p(x \mid inputs, w)$$

即给定输入 X 和模型参数 W 下分类为 z 的概率，而我们在分类问题中希望优化的目标是:让训练集的真实条件概率分布和模型输出的概率分布尽可能相同。我们知道评估概率分布差异的方法是 KL 散度，而最优化 KL 散度的过程又可以简化为优化 Cross Entropy 的过程。离散概率分布的交叉熵写做

$$L(x, y) = -\sum_{i=1}^{n} log\left(p\left(x^{(i)}\right)_{y^{(i)}}\right)$$

它是模型接收第 i 个训练集后，我们预期的正确分类的概率的负对数的加和平均。我们假设训练集是从 p(x)的真实采样，则此方法获得的是真实交叉熵的蒙特卡洛估计。把 softmax 的输出代入，可以计算单个样本输入时，L 的导数。

$$L(x, y) = -log(p_y)$$

$$\frac{\partial L}{\partial p_i} = -\frac{1}{p_i} \qquad (i = y) \quad (1)$$

$$\frac{\partial L}{\partial p_i} = 0 \qquad (i \neq y) \quad (2)$$

计算p_i关于x_j的偏导数，要分为$i = j$和$i \neq j$讨论

$$\frac{\partial p_i}{\partial x_j} = \frac{-e^{x_i} e^{x_j}}{(\sum_{k=1}^{m} e^{x_k})^2} = -p_i p_j \qquad (i \neq j) \quad (3)$$

$$\frac{\partial p_i}{\partial x_i} = \frac{e^{x_i} \sum_{k=1}^{m} e^{x_k} - (e^{x_i})^2}{(\sum_{k=1}^{m} e^{x_k})^2} = p_i - p_i^2 \qquad (i = j) \quad (4)$$

组合上式(1),(2),(3),(4)则能得到 L 关于 x_j 的偏导数。

$$\frac{\partial L}{\partial x_j} = \frac{\partial L}{\partial p_y}\frac{\partial p_y}{\partial x_j}$$

$$\frac{\partial L}{\partial x_j} = -\frac{1}{p_y}\left(-p_y p_j\right) = p_j \qquad (y \neq j)$$

$$\frac{\partial L}{\partial x_j} = -\frac{1}{p_y}\left(p_j - p_j^2\right) = p_j - 1 \qquad (y = j)$$

如果把标签写成 one hot 的向量，即正确标签为 1，错误标签为 0，就能给出非常简单的向量梯度形式

$$\frac{\partial L}{\partial x} = p - one\_hot\_labels$$

优化

SGD with moentum

我们优化神经网络的一种有效方法是使用带动量的随机梯度下降，每次参数更新不是完全用当前的梯度，而是基于原本的下降方向作出调整。这样可以确定一个比较稳定的下降方向，本质上是共轭梯度法的简化版本。共轭梯度法希望通过方向修正，让所有的下降方向尽可能正交。而如果我们把修正系数设为常数，就是 SGD with moentum 算法。

$$g = gradient\ from\ SGD$$

$$v \leftarrow \beta v + (1-\beta)g$$
$$\theta \leftarrow \theta - \alpha v$$

RMSProp

一种自适应学习率的方法能帮助我们在训练时得到一个更合理的学习率，在训练的过程中，我们使用动量的思想维护并更新一个标量，这个标量是平方梯度的长期估计，即

$$g = gradient\ from\ SGD$$
$$r \leftarrow \rho r + (1-\rho) g \odot g$$

如果我们的学习率是α，则算法会用平方梯度开根号，即梯度 2 范数的长期估计来修正学习率

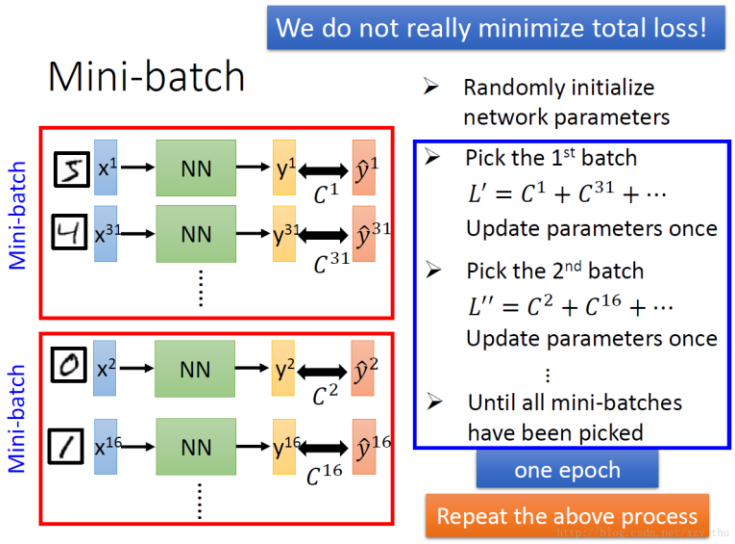
$$\epsilon \leftarrow \frac{\alpha}{\sqrt{r}}$$

要更新时，就使用ϵ作为真正的学习率

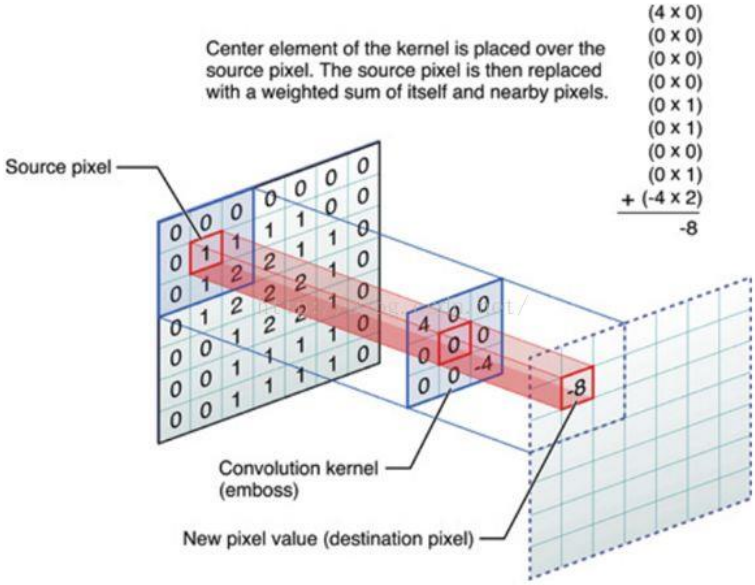
$$\theta \leftarrow \theta - \epsilon v$$

Mni-Batch

使用批量梯度下降算法是数据量较大时一种比较快速的计算和训练方法，我们上面神经网络的主要计算过程都是矩阵W和向量$\vec{x}$ 的乘法运算，如果我们把多个向量$\vec{x}$ 排成矩阵，运算就变成矩阵乘矩阵. 矩阵和矩阵的计算比起矩阵和向量进行多次计算更为有效率。



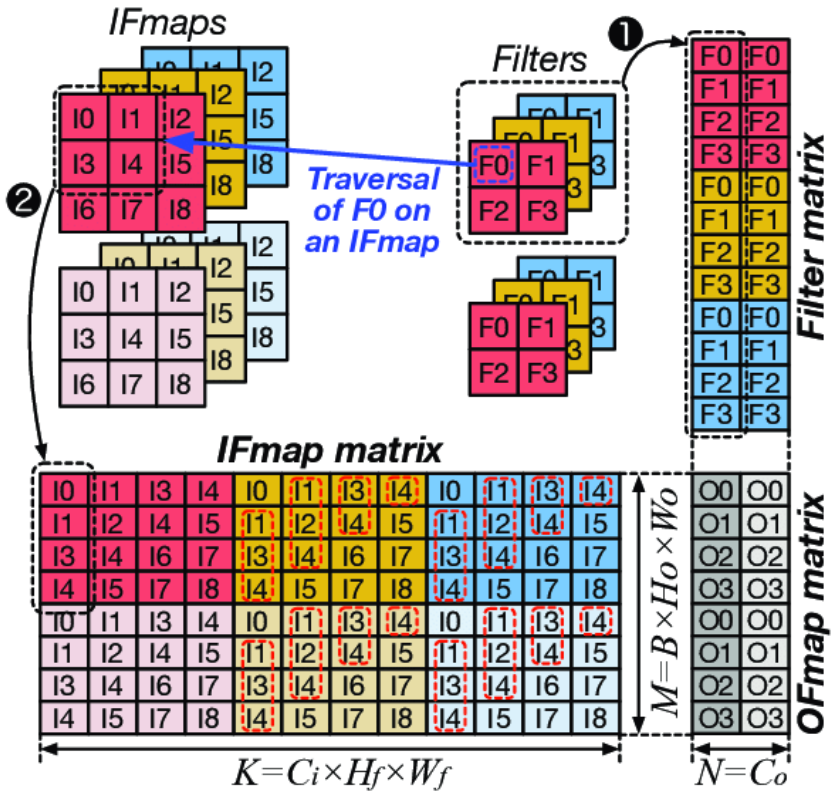
卷积



使用一个卷积核在图像上按照一定步长扫过整张图像，卷积核从左到右，从上到下扫过整张图片，并在每个位置让卷积核和局部的图片做点乘，然后把各像素点乘的结果加起来，得到一个输出。这种操作在图像处理时，比起仿射层更节省参数，从而更不容易过拟合。同时它保有了特征的平移不变性，这在图像上相当通用。

im2col

一种把要卷积的图像转为矩阵的方法如下所示



转换完成后，把卷积核展开，就能用矩阵乘法直接完成卷积。如果使用这种方法完成卷积，则 im2col 算法的质量直接影响了卷积的计算速度。

im2col 的接口写做

input: $im=(N,C,H,W)$，stride，kernel_size，pad。

$$out\_h=\frac{(H-size+2pad)}{stride}+1,$$

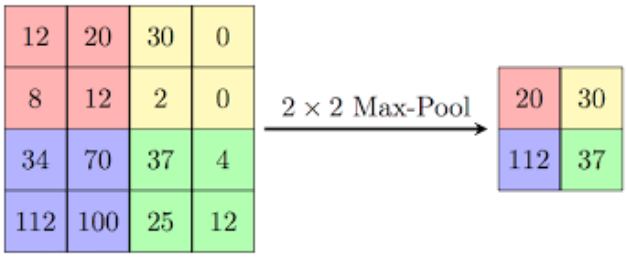
$$out\_w=\frac{(W-size+2pad)}{stride}+1$$

return: $col=(N\times out\_h\times out\_w, C\times size\times size)$

同时我们还需要 col2im 做反向传播，它接收$col=(N\times out_h\times out_w, C\times size\times size)$，输出$im=(N,C,H,W)$。这时 col 矩阵中的不同位置的元素可能映射到 im 中的同一个位置，这时我们把它们加起来。

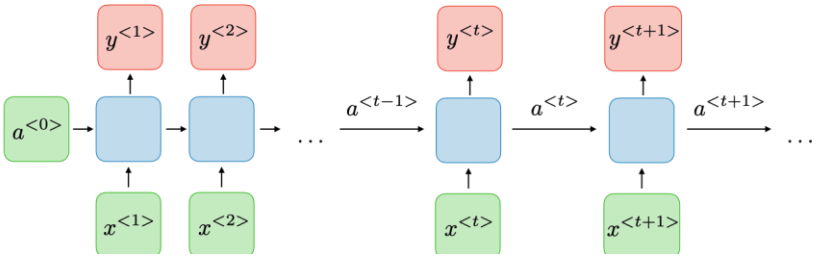
池化

池化的操作和卷积很类似，它也是用一个窗口在图像上扫描. 扫描的过程中，池化层做的不是用自带的参数去与图像点积，而是用 max 或者 average 等算术运算操作于当前窗口. 最常用的池化方法是 max pooling，它用一个窗口在图像上滑动，并输出每个窗口最大的像素值.



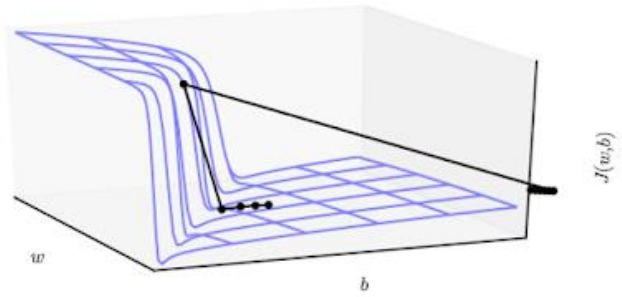
池化层实现 maxout 非线性和特征提取，能够把原图像快速压缩到比较小的尺寸，反复迭代则能让图像的尺寸变小，特征层次变高。

RNN



$$h_t = \sigma(x_t U + h_{t-1} W + b)$$

随时间推进，我们每个时间点都会接收一个输入x，并把它和之前的隐层状态仿射运算，经过激活函数生成新的隐层状态，传到下一个时间点。



RNN 中梯度会爆炸，所以要施加梯度截断来让 RNN 正常训练，或者使用 LSTM。

BPTT

随时间传播的过程和仿射的传播并无区别，我们把随时间传播过程中的中间变量都理解为独立的变量就能计算出随时间的反向传播梯度。

$$o_t = x_t U + h_{t-1} W + b$$
$$h_t = \sigma(o_t)$$

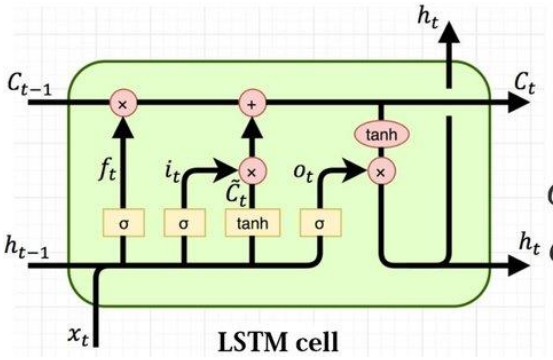
设激活函数为 tanh

$$\frac{\partial L}{\partial o_t} = 1 - \left(\frac{\partial L}{\partial h_t}\right)^2$$
$$\frac{\partial L}{\partial b}^{(t)} = SUMROW \frac{\partial L}{\partial o_t}$$
$$\frac{\partial L}{\partial U}^{(t)} = x^T \frac{\partial L}{\partial o_t}$$
$$\frac{\partial L}{\partial W}^{(t)} = {h_{t-1}}^T \frac{\partial L}{\partial o_t}$$
$$\frac{\partial L}{\partial h_{t-1}}^{(t)} = \frac{\partial L}{\partial o_t} W^T$$

对每个参数的总梯度，我们计算每个时间上梯度的和。如

$$\frac{\partial L}{\partial W} = \sum_t^T \frac{\partial L}{\partial W}^{(t)}$$

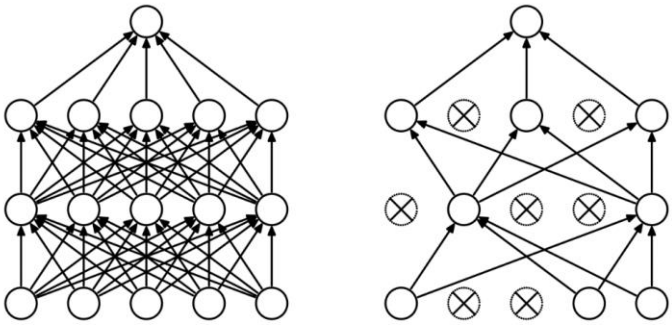
LSTM



$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\
C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
$$

使用门控单元实现比 RNN 更长期的记忆，从而不容易出现梯度问题，由于参数更多，表达能力也更强。

Dropout



(a) Standard Neural Net　　(b) After applying dropout.

在训练中，我们随机置零某些神经元，这样训练出来的网络将在失去一些神经元后仍然可用。从而我们的神经网络将是多个子模型混合的结果，子模型都能收敛，则总模型混合了它们之后将具有更好的抗干扰能力，降低模型方差，增强泛化能力。

BatchNorm

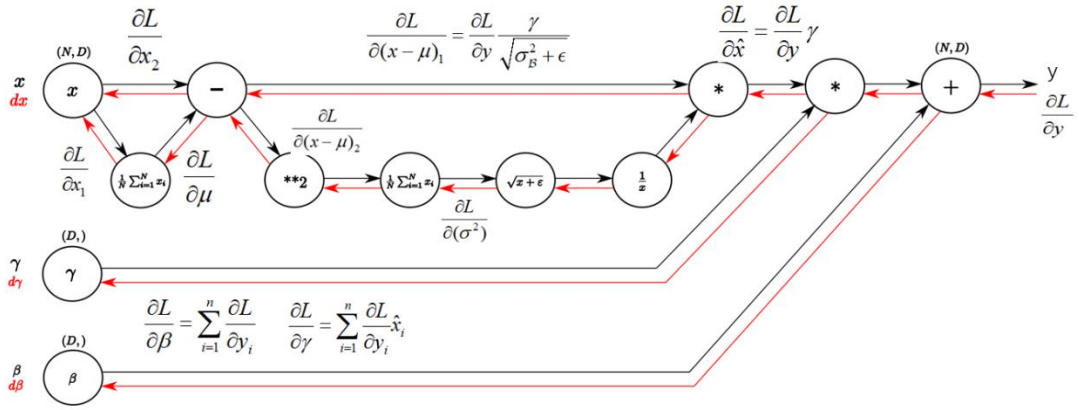
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$
\begin{aligned}
\mu_{\mathcal{B}} &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i && \text{// mini-batch mean} \\
\sigma^2_{\mathcal{B}} &\leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 && \text{// mini-batch variance} \\
\widehat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} && \text{// normalize} \\
y_i &\leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) && \text{// scale and shift}
\end{aligned}
$$

从数据标准化中得到启发，标准化的数据更能适应激活函数，网络训练更容易。而如果我们人为在网络中添加标准化操作，自然能帮助训练。
而只进行标准化又会让数据固有的特征丢失，因此我们设置一个反标准化操作以及一些可学习参数，允许网络在标准化后还原一部分的特征。这样即完成了标准化的目标又能不让数据

损失太多信息。

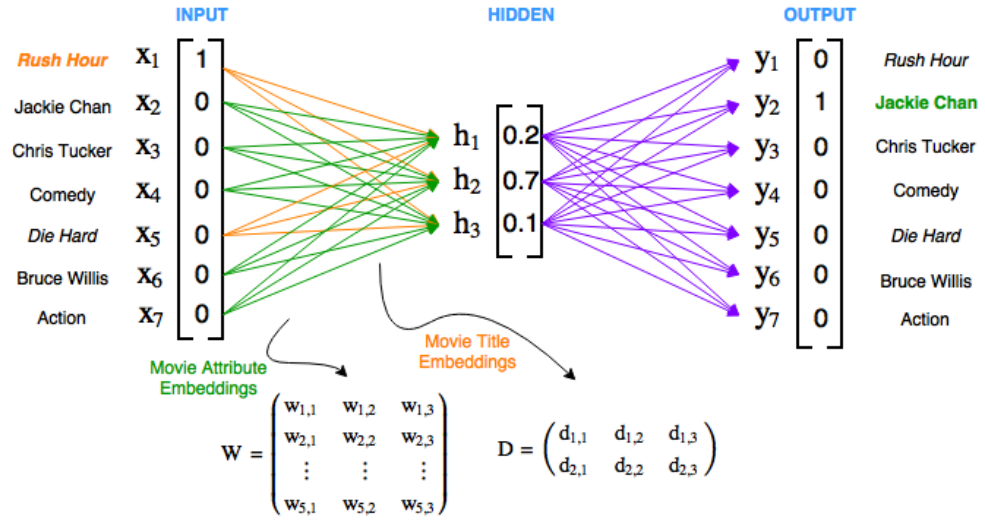
计算图模型



在 BatchNorm 中我们使用了多种 pointwise 的操作，每个操作对应了一种梯度计算方法。根据链式法则，我们能通过计算图和反向传播计算出这个过程中每个变量的梯度。这种方法在现代深度学习框架中极为常见。

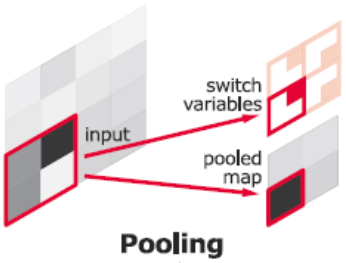
Embedding

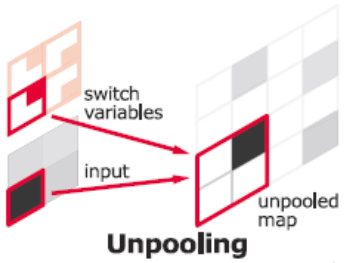
旨在完成稀疏到稠密的映射。我们构造字典，把输入的整形对应到字典中的一个连续型向量。字典中的全部向量都可以训练，我们可以基于 embedding 实现推荐系统和词嵌入等非常有用的分析工具。

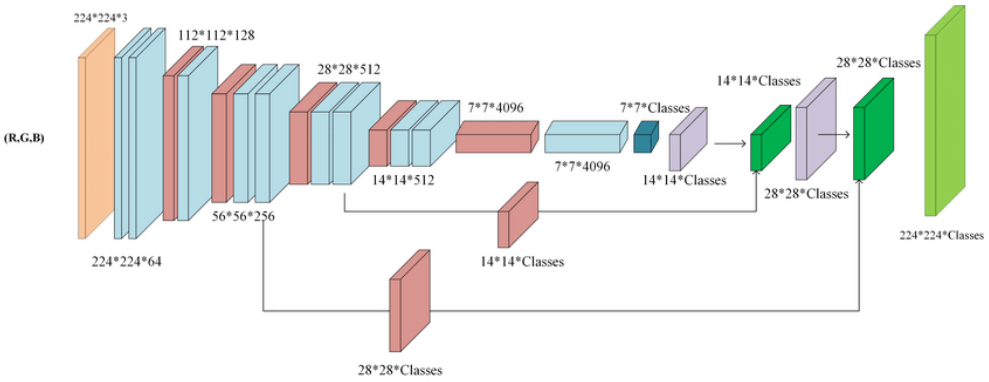


Unpooling

一种图像上采样的手段，常用于和卷积相对应的解码。经过上采样，我们可以得到比输入更大尺寸的图像，这个操作可以用于语义分割和高分辨率图像生成等等。


switch
variables
input
pooled
map
Pooling


switch
variables
input
unpooled
map
Unpooling


(R.G.B)
224*224*3
224*224*64
112*112*128
56*56*256
28*28*512
14*14*512
7*7*4096
7*7*4096
7*7*Classes
14*14*Classes
14*14*Classes
28*28*Classes
28*28*Classes
14*14*Classes
28*28*Classes
224*224*Classes