Summary

181840013 陈超

Basic knowledge

Softmax：

$$p_i = Softmax(z_i) = \frac{e^{z_i}}{\sum_{c=1}^{C} e^{z_c}}$$

用交叉熵作为损失函数:

$$L = -\sum_{c=1}^{C} y_c \, log(p_c)$$

计算得到梯度为:

$$\frac{\partial L}{\partial z_i} = p_i - y_i$$

优点:

1. 归一化
2. 梯度计算简单
3. Softmax 训练的深度特征，会把整个超空间或者超球，按照分类个数进行划分，保证类别是可分的，这一点对多分类任务如 MNIST 和 ImageNet 非常合适，因为测试类别必定在训练类别中。

缺点:

Softmax 并不要求类内紧凑和类间分离，这一点非常不适合人脸识别任务，因为训练集的 1W 人数，相对测试集整个世界 70 亿人类来说，非常微不足道，而我们不可能拿到所有人的训练样本。

Loss function：

0-1 损失函数，绝对值损失函数，平方损失函数，交叉熵。

交叉熵函数使用来描述模型预测值和真实值的差距大小，越大代表越不相近；似然函数的本质就是衡量在某个参数下，整体的估计和真实的情况一样的概率，越大代表越相近。交叉熵函数可以由最大似然函数在伯努利分布的条件下推导出来，或者说最小化交叉熵函数的本质就是对数似然函数的最大化。

使用 sigmoid 作为激活函数的时候，常用交叉熵损失函数而不用均方误差损失函数，因为可以避免出现 sigmoid 导致的梯度过小，更新过慢的问题。

当然，针对特殊的任务也会有很多专用的损失函数，比如用于机器翻译的 BLEU，用于目标检测的 IOU 等。

Activation function：

1. sigmoid：

$$f(x) = sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)(1-f(x))$$

二分类问题时 sigmoid 和 softmax 是一样的，求的都是 cross entropy loss，而 softmax 可以用于多分类问题。

优点：

sigmoid 函数可以将实数映射到 (0, 1) 区间内。平滑、易于求导。

缺点：

1. 激活函数含有幂运算和除法，计算量大。
2. 反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络训练。
3. sigmoid 的输出不是 0 均值的,这会导致后一层的神经元将得到上一层输出的非 0 均值的信号作为输入。

2. tanh：

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$f'(x) = 1 - tanh^2(x)$$

优点：

tanh 激活函数是 0 均值的，tanh 激活函数相比 sigmoid 函数更'陡峭'了，对于有差异的特征区分得更开了

缺点：

tanh 也不能避免梯度消失问题。

3. ReLU：

$$f(x) = relu(x) = max(x, 0)$$

优点：

1. 计算量小。
2. 激活函数导数维持在 1，可以有效缓解梯度消失和梯度爆炸问题。
3. 使用 ReLU 会使部分神经元为 0，这样就造成了网络的稀疏性，并且减少了参数之间的相互依赖关系，缓解了过拟合问题的发生。

缺点：

输入激活函数值为负数的时候，会使得输出为 0，那么这个神经元在后面的训练迭代的梯度就永远是 0 了，参数 w 得不到更新，也就是这个神经元死掉了。这种

情况在你将学习率设得较大时很容易发生。

4. Leaky ReLU：、
   修改了负数部分，使其有一个较小的梯度，避免梯度消失。

防止过拟合的方法：

1. 提前终止（当验证集上的效果变差的时候）
2. L1 和 L2 正则化加权
3. soft weight sharing
4. dropout

常用小技巧：

1. Dropout：解决过拟合
2. Batch Normalization：

   BN 的本质就是利用优化变一下方差大小和均值位置，使得新的分布更切合数据的真实分布，保证模型的非线性表达能力。

   BN 在深层神经网络的作用非常明显：若神经网络训练时遇到收敛速度较慢，或者“梯度爆炸”等无法训练的情况发生时都可以尝试用 BN 来解决。同时，常规使用情况下同样可以加入 BN 来加速模型训练，甚至提升模型精度。
3. 如何解决高偏差（high bias）
   1. 使用更大的神经网络
   2. 尝试变换网络架构
   3. 训练更久
4. 如何解决高方差（high variance）
   1. 获取更多的训练数据
   2. 使用更简单的模型
   3. 正则化，drop-out（本质就是简化模型），early-stopping 等
   4. 尝试变化网络架构

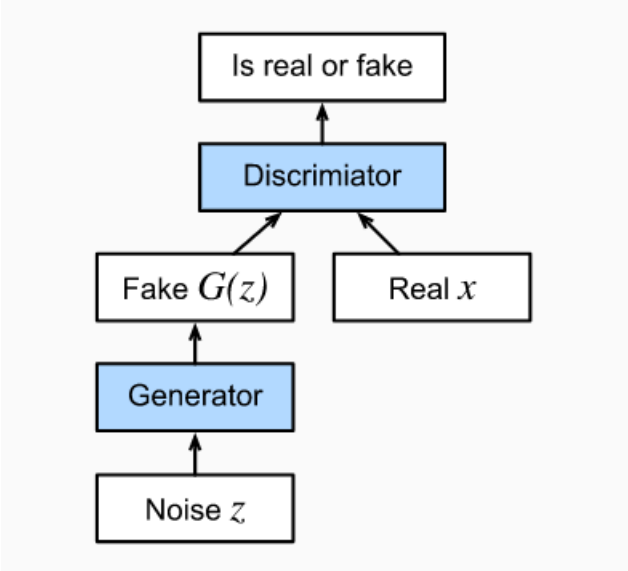
Optimization methods：

1. SGD：随机从每个 batch 种选择一个数据进行梯度更新，减少计算量，但是收敛速度慢，容易停留在局部最优点。
2. Minibatch Stochastic Gradient Descent：综合了 batch 梯度下降与 stochastic 梯度下降，在每次更新速度与更新次数中间取得一个平衡，其每次更新从训练集中随机选择 m, m<n 个样本进行学习。
3. SGD (with Momentum)：加了动量，收敛更快。
4. Adagrad：考虑了二阶动量，可以度量历史更新频率，可以解释为以往所有梯度值的平方和，即越大表示经常更新，需要放慢点，越小表示不经常更新，需要放快些。在稀疏数据场景下表现非常好，但是过于激进。
5. AdaDelta / RMSProp：针对 Adagrad 过于激进的改进策略：不累积全部历史梯度，而只关注过去一段时间窗口的下降梯度。这也就是 AdaDelta 名称中 Delta 的来历。
6. Adam：SGD + 一阶动量 + 二阶动量 = Adam = Adaptive + Momentum

综上，可以先 Adam 再 SGD。

Some structures

GAN：



我们有两个网络，G（Generator）和 D（Discriminator）。Generator 是一个生成图片的网络，它接收一个随机的噪声 z，通过这个噪声生成图片，记做 G(z)。Discriminator 是一个判别网络，判别一张图片是不是“真实的”。它的输入是 x，x 代表一张图片，输出 D（x）代表 x 为真实图片的概率，如果为 1，就代表 100%是真实的图片，而输出为 0，就代表不可能是真实的图片。

目标函数：

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

在这里，训练网络 D 使得最大概率地分对训练样本的标签（最大化 log D(x)和 [公式] ），训练网络 G 最小化 log(1 – D(G(z)))，即最大化 D 的损失。而训练过程中固定一方，更新另一个网络的参数，交替迭代，使得对方的错误最大化，最终，G 能估测出样本数据的分布，也就是生成的样本更加的真实。

优化 D：

$$\max_D V(D,G) = E_{x\sim p_{data}\ (x)}[log(D(x))] + E_{z\sim p_z\,(z)}[log(1 - D(G(z)))]$$

优化 G：

$$\min_G V(D,G) = E_{z\sim p_z\,(z)}[log(1 - D(G(z)))]$$

CNN:

如果用全连接神经网络处理大尺寸图像具有三个明显的缺点：

1. 将图像展开为向量会丢失空间信息。
2. 参数过多效率低下，训练困难。
3. 大量的参数也很快会导致网络过拟合。

而使用卷积神经网络可以很好地解决上面的三个问题。

卷积神经网络主要由这几类层构成：输入层、卷积层，激活函数层、池化层和全连接层。通过将这些层叠加起来，就可以构建一个完整的卷积神经网络。

其优点在于：

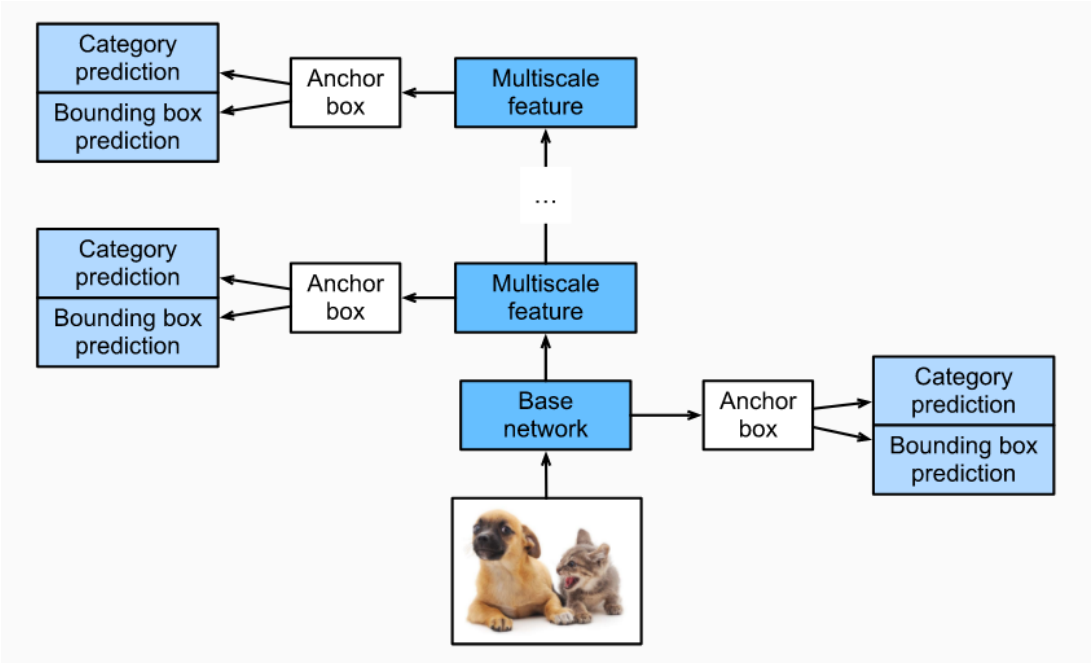
1. 有效降低了参数数量，加速了训练和推理速度。
2. 保留了图像原有的空间信息，能有效提取图片中的低维和高维信息。
3. 符合人类的视觉原理。

具体正向反向传播方法这里就不再赘述了。我这学期还选了一门通识课：基于 FPGA 的硬件加速原理，在硬件层面上实现了卷积层，对这方面的理解应该是比较透彻的。

经典架构包括 AlexNet，VGG，GoogLeNet，ResNet 等，这里也不再赘述了。

SSD：

就是简单的一层一层卷积，从低维到高维提取特征。



R-CNN：

1. 用选择性搜索 Selective Search 算法在图像中从下到上提取 2000 个左右的可能包含物体的候选区域 Region Proposal。
2. 因为取出的区域大小各自不同，所以需要将每个 Region Proposal 缩放成统一的大小并输入到 CNN，将 CNN 的 fc7 层的输出作为特征。
3. 将每个 Region Proposal 提取到的 CNN 特征输入到 SVM 进行分类。

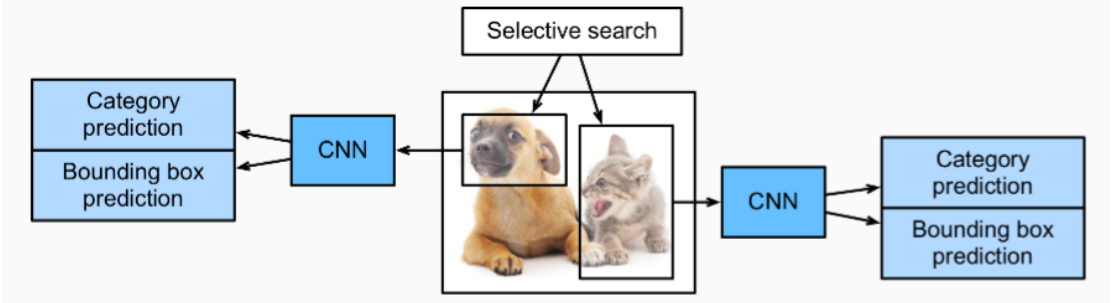


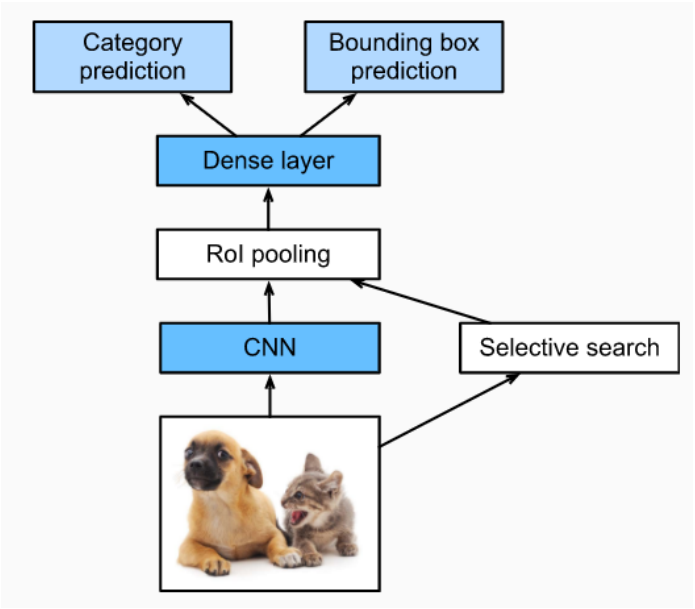
Fig. 13.8.1 R-CNN model.

Fast R-CNN:

不像 R-CNN 把每个候选区域给深度网络提特征，而是整张图提一次特征。

共享卷积层，现在不是每一个候选框都当做输入进入 CNN 了，而是输入一张完整的图片，在第五个卷积层再得到每个候选框的特征

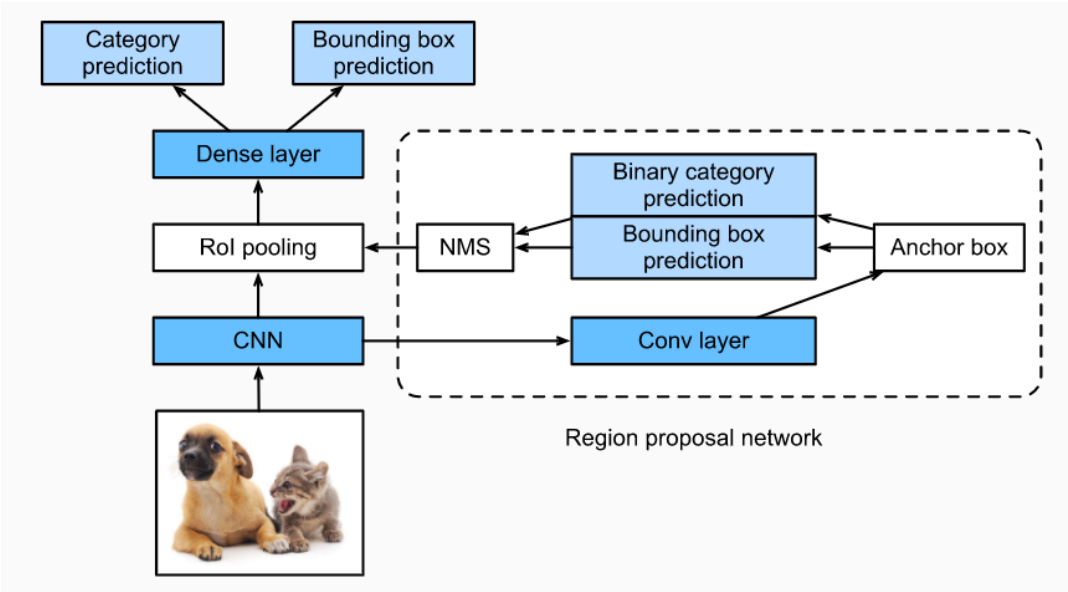
1. 在图像中确定约 1000-2000 个候选框 (使用选择性搜索)。
2. 对整张图片输进 CNN，得到 feature map。
3. 找到每个候选框在 feature map 上的映射 patch，将此 patch 作为每个候选框的卷积特征输入到 SPP layer 和之后的。



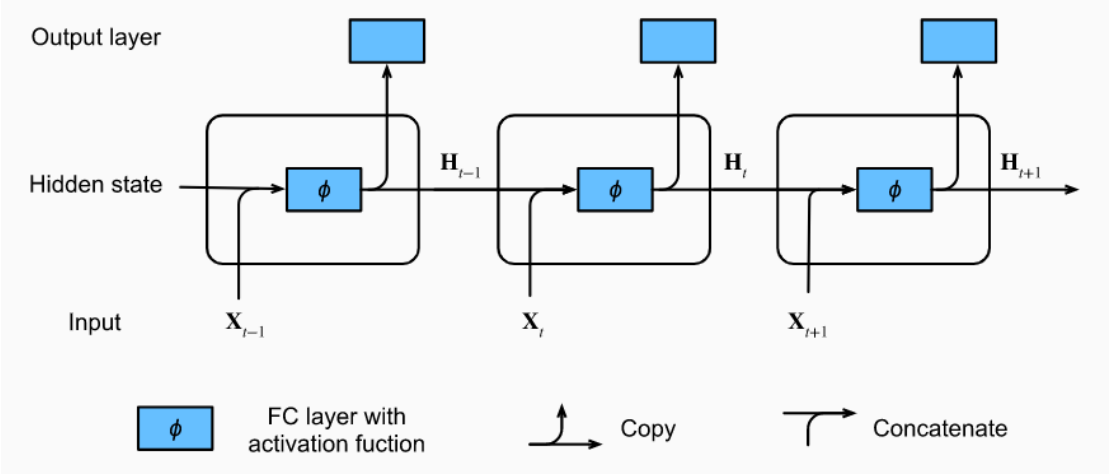
Faster R-CNN:

加入一个提取边缘的神经网络，也就说找到候选框的工作也交给神经网络来做了。

1. 对整张图片输进 CNN，得到 feature map。
2. 卷积特征输入到 RPN，得到候选框的特征信息。
3. 对候选框中提取出的特征，使用分类器判别是否属于一个特定类。
4. 对于属于某一类别的候选框，用回归器进一步调整其位置。



RNN：



使用隐藏状态递归计算的神经网络称为递归神经网络（RNN），RNN 的隐藏状态可以捕获序列到当前时间步长的历史信息。值得一提的是，RNN 的参数数量不会随着 timestep 增加而增加。具体每一个 timestep 的更新过程如下。

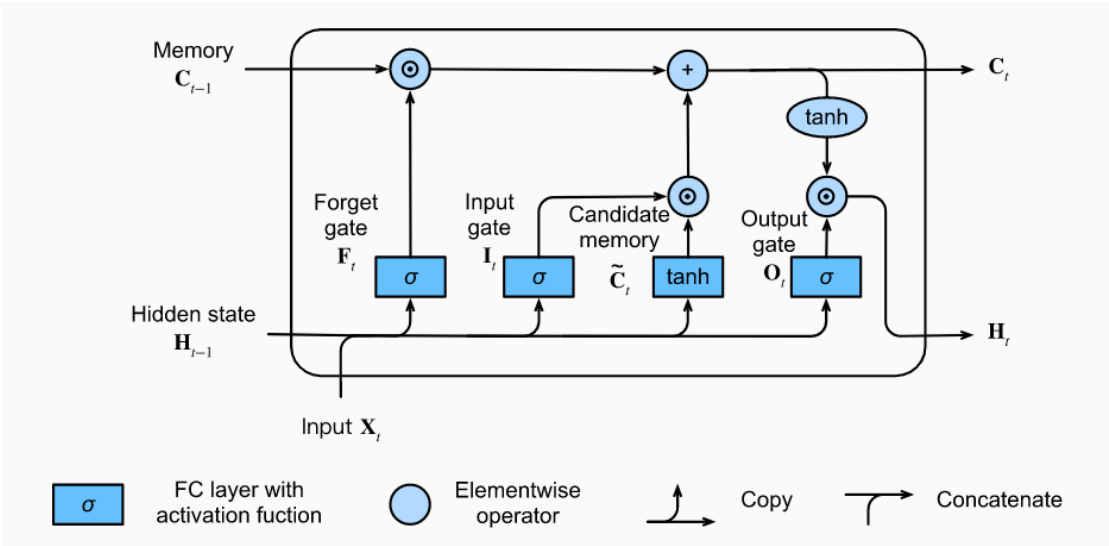
$$\mathbf{H}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h)$$

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q$$

由于梯度弥散，导致在序列长度很长时，无法在较后的时间步中，按照梯度更新较前时间步的 W，导致无法根据后续序列来修改前向序列的参数，使得前向序列无法很好地做特征提取，使得在长时间步过后，模型将无法再获取有效的前向序列记忆信息。

同样的，根据上述梯度的推导，梯度计算将会导致参数累乘，若初始参数较大时，则较大数相乘，将导致梯度爆炸，然而梯度爆炸相对于梯度弥散较容易解决，通常加入梯度裁剪即可一定程度缓解。

LSTM：



三门一状态：遗忘门、输入门、输出门和细胞状态。

1. 遗忘门会读取上一个输出和当前输入，做一个 sigmoid 的非线性映射，然后输出一

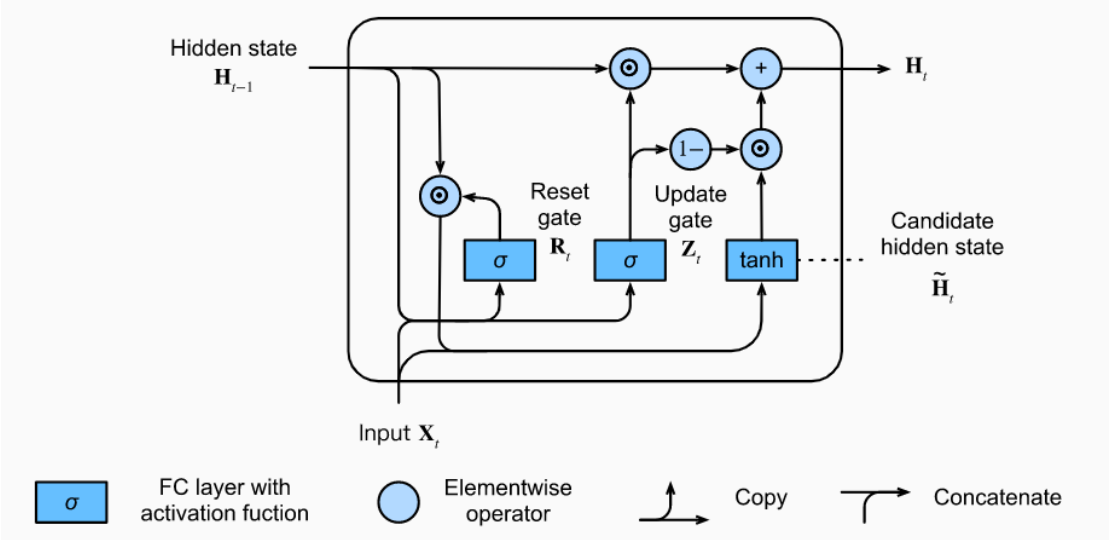
个向量 Ft，该向量每一个维度的值都在 0 到 1 之间，1 表示完全保留，0 表示完全舍弃，相当于记住了重要的，忘记了无关紧要的）

2. 输入门：1. sigmoid 层决定什么值我们将要更新。2. tanh 层创建一个新的候选值向量，会被加入到状态中。
3. 更新细胞状态：我们把旧状态与 Ft 相乘，丢弃掉我们确定需要丢弃的信息。接着加上 it*Ct，得到新的细胞状态。
4. 输出门：根据 Xt 和 Ct 得到新的 Ht，之后再根据隐藏状态 Ht 和输入 Xt 输出 Yt（图中没有画出 Yt）。

LSTM 和 RNN 的区别：

1. RNN 没有细胞状态；LSTM 通过细胞状态记忆信息。
2. RNN 激活函数只有 tanh；LSTM 通过输入门、遗忘门、输出门引入 sigmoid 函数并结合 tanh 函数，添加求和操作，减少梯度消失和梯度爆炸的可能性。
3. RNN 只能够处理短期依赖问题；LSTM 既能够处理短期依赖问题，又能够处理长期依赖问题。

GRU：



GRU 使用了更新门（update gate）与重置门（reset gate）。基本上，这两个门控向量决定了哪些信息最终能作为门控循环单元的输出。

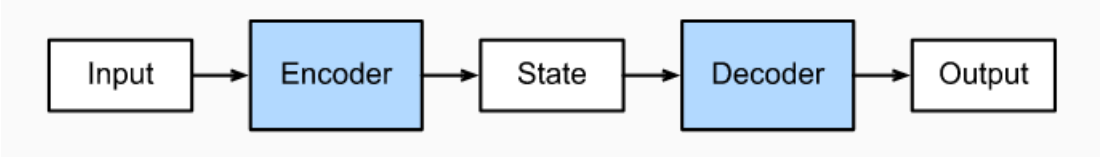
基本过程与 LSTM 很相似，不过做了简化，同时取消了细胞状态这个变量。具体过程见上图，这里就不再赘述了。

重置门有助于捕获序列中的短期依赖关系，而更新门有助于捕获序列中的长期依赖关系。

GRU 与 LSTM 的区别：

1. LSTM 单元的一个特征控制记忆内容的暴露，而 GRU 没有。在 LSTM 单元中，记忆内容的数量是由输出门控制的，但 GRU 是把所有内容都暴露出来，没有进行控制。
2. 另一个区别是输入门的位置，或者与之对应的重置门的位置。LSTM 单元计算新的记忆内容的时候，它没有控制从上一时间步传来的信息的数量。而是控制控制有多少新的内容被添加到记忆单元，记忆单元和遗忘门是分开的。
3. LSTM 有细胞状态，而 GRU 取消了细胞状态。

Encoder-Decoder：



seq2seq 任务（如机器翻译）的一个问题是其输入和输出都是可变长度序列。为了处理这种类型的输入和输出，我们可以设计一个包含两个主要组件的体系结构。第一个组件是一个编码器：它以一个可变长度的序列作为输入，并将其转换为具有固定形状的编码。第二个部分是解码器：它将固定形状的编码状态映射到可变长度序列。简单说来，编码就是将输入序列转化成一个固定长度的向量，解码就是将之前生成的固定向量再转化成输出序列。

Reinforcement learning：

从时间点 t 向后的总回报可以表达为：

$$R_t = r_t + \gamma(r_{t+1} + \gamma(r_{t+2}+\ldots)) = r_t + \gamma R_{t+1}$$

在 Q 学习中，我们定义一个函数 Q(s, a)，表示我们在状态 s 下执行动作 a ，未来能得到的最高分。

$$Q(s_t, a_t) = max\ R_{t+1}$$

有了 Q 函数，我们就可以计算Π函数来指导我们在各个状态下应该执行哪种操作。

$$\pi(s) = argmax_a\ Q(s,a)$$

深度学习的目标就是持续优化 Q 函数使它能接近真实情况。神经网络极其擅长处理高度结构化的数据。我们可以使用神经网络来表示我们的 Q 函数。

存在一个问题是，使用非线性函数得到近似的 Q 值是不稳定的。要使它真正收敛还需要加入很多技巧，而且要花费很长的时间。一个重要技巧就是经验回放。在游戏过程中，所有的经历 <s, a, r, s'> 都存储在一个可回顾的记忆模块中。训练神经网络的时候，会从记忆中取出随机小批量的记忆片段来训练，而不是直接使用最近期的经历。这样就切断了之后相似训练情况的连续性，这种相似性往往会使整个网络局限于一小块状态区域。此外，经验回放使训练任务更类似于通常的监督学习，这简化了调试和测试算法。而且也可以直接收集人类玩游戏的经验，在此基础上继续训练。

为了平衡探索与经验，还需要用到 ε-greedy 方法，用概率 ε 来选择是继续探索，还是直接根据经验做出决策。

Transfer learning：

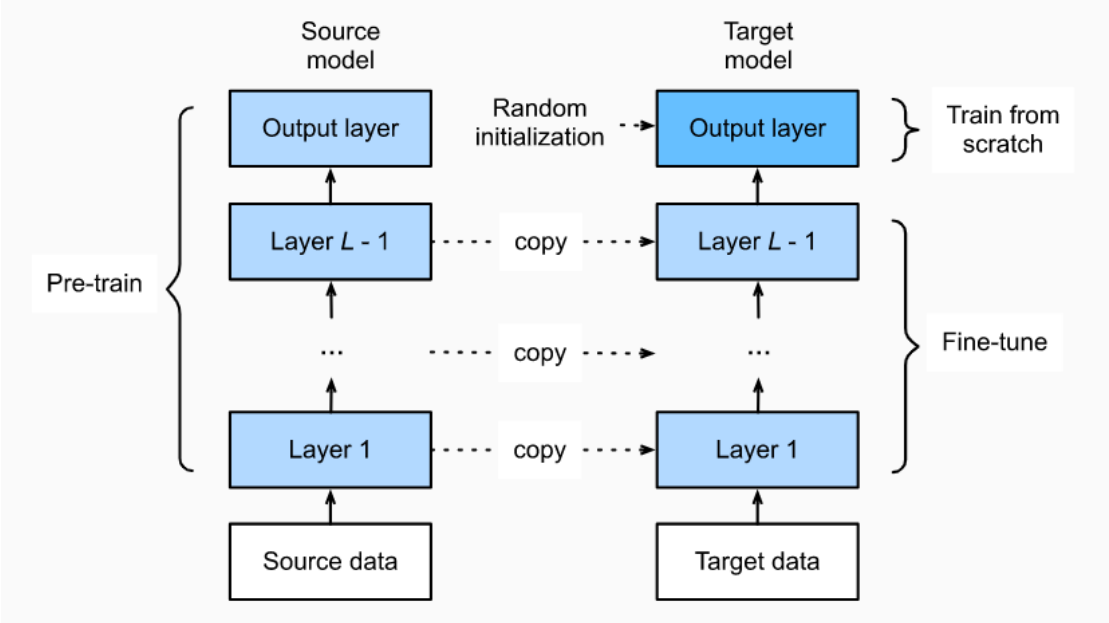
迁移学习将从源数据集学到的知识迁移到目标数据集。微调是迁移学习的常用方法。

微调包括以下四个步骤：

1. 在源数据集（例如 ImageNet 数据集）上预训练神经网络模型，即源模型。
2. 创建一个新的神经网络模型，即目标模型。这将复制源模型（输出层除外）上的所有模型设计及其参数。我们假设这些模型参数包含从源数据集学习到的知识，并且这些知识同样适用于目标数据集。我们还假设源模型的输出层与源数据集的标签密切相关，因此不在目标模型中使用。
3. 将输出大小为目标数据集类别数的输出层添加到目标模型中，并随机初始化该层的

模型参数。

4. 在目标数据集上训练目标模型。我们将从头开始训练输出层，而所有剩余层的参数将根据源模型的参数进行微调。通常，微调参数使用较小的学习速率，而从头开始训练输出层可以使用较大的学习速率。



Lifelong learning:

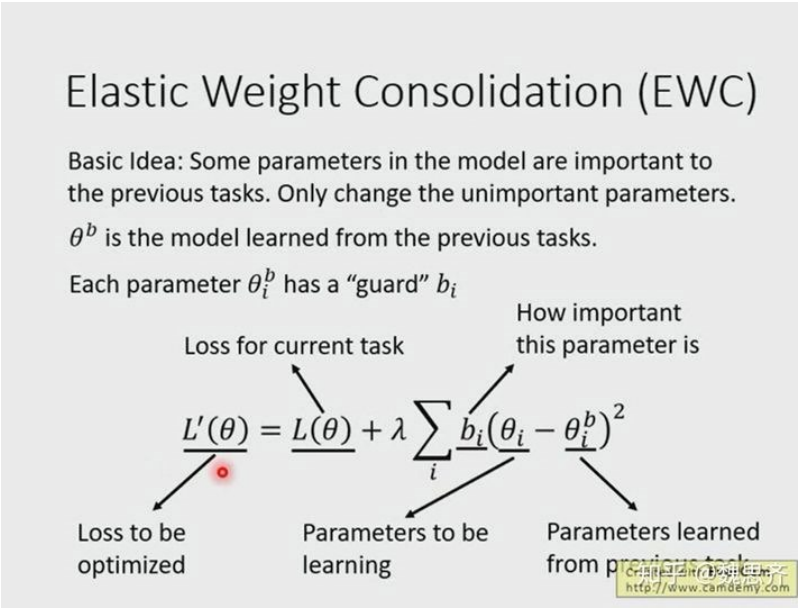
Transfer learning 和 Lifelong learning 的区别：Transfer learning 只考虑在当前任务上的效果，而 Lifelong learning 需要考虑在所有任务上的效果。

Multi-task learning 和 Lifelong learning 的区别: Lifelong learning 训练时只用当前任务的数据，而 Multi-task learning 会用到之前所有任务的数据。这带来了数据存储以及计算量不断增大的问题，Multi-task learning 可以看作是 Lifelong learning 的 upper bound。

第一个需要解决的问题是知识的保留。

在学完 task1 后学 task2，会导致模型忘记原有的 task1 上学到的东西（虽然 task1 和 task2 一同训练地结果表明模型是可以把两个 task 都学好的）。解决这个问题有两种很简单的方法，一是把所有任务和在一起训练，但这会造成很大的内存开销，二是在训练 task2 时用 GAN 网络生成 task1 中的数据，但这不能保证生成数据的可靠性。

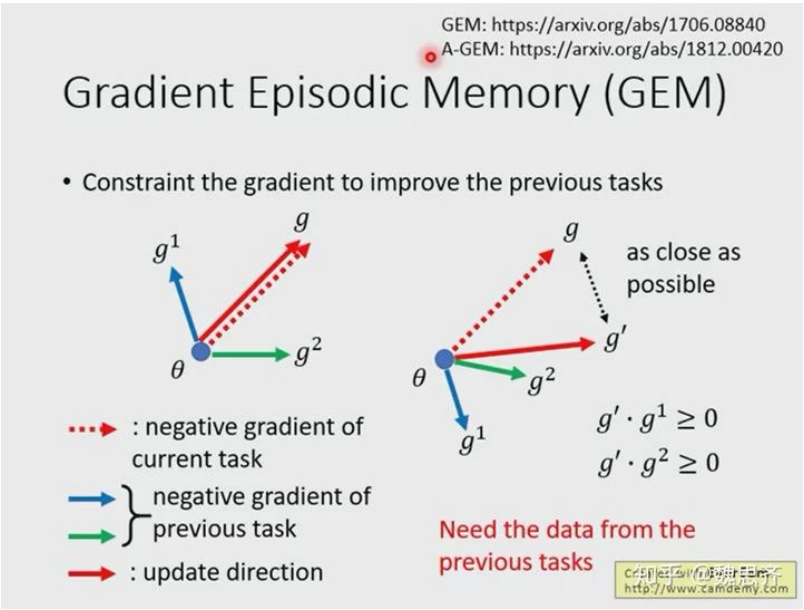
EWC 方法（elastic：弹性的，consolidation：巩固）：基础的想法就是：在训练的参数中，有的参数是不重要的（这个参数无论怎么改变，之前学学过的任务都不会忘记），但是有的参数是十分重要的（就是说如果这个参数改变了，那么之前学过的任务就忘记了），θb 是这个模型中所有的参数。每一个参数 θbi 第一位为 bi 表示参数是否是重要的，如果 bi 值大，就意味着这个参数是十分重要的，改变的话之前的任务学习就会忘记了。θi 反应的是我们更新后的参数，θib 是更新前的参数。原来的 loss function 仅仅是对参数 θ 进行更新，但是没有考虑这个参数是否是重要的。新的 loss function 就考虑了这一点，如果 bi 很大的话，该参数的变化就会对 loss 有很大影响，bi 很小则没有影响。



bi 的计算方法就是对 loss function 的二次微分。二次微分小，就意味着一阶微分较为平缓，重要性较低，可以进行较大改变。反之二次微分大，就意味着一阶微分较为陡峭，不能进行较大改变。

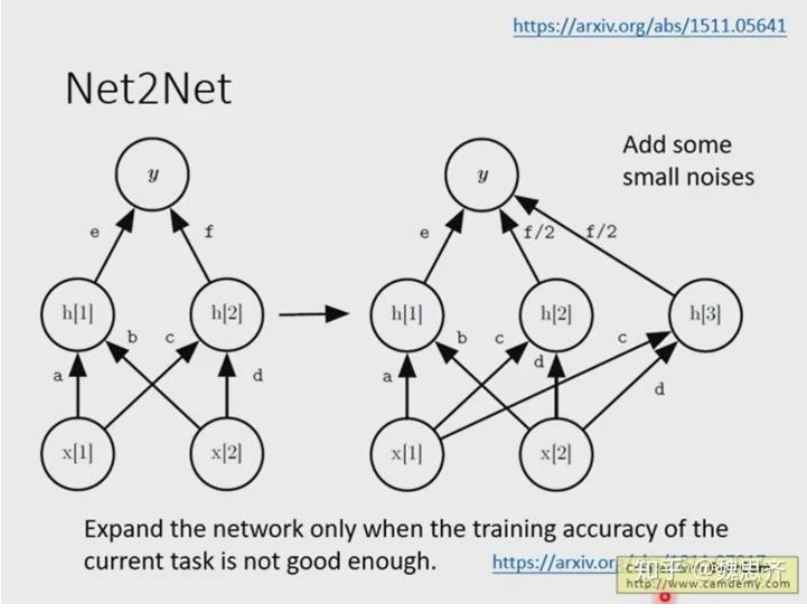
第二个需要解决的问题是，在训练不同任务的时候，如何将知识进行 transfer。

GEM 方法：假设我们有三个任务，task1，task2，task3。之前 1，2 都已经学习完毕，下面我们想学习 task3，我们计算出更新 task3 参数方向是红色虚线的方向，我们对这个 g 和之前第一个任务的 g1，第二个任务的 g2 做内积，发现这个内积都是正数，所以如果是按照虚线方向更新的话，对之前两个任务是没有任何影响的，我们就按照虚线方向更新参数。但是如果像右图一般，g 和 g1 的内积是负数的话，这个 g 就会对之前 task1 造成影响，我们就不可以这么做了。我们转换 g 到实线的方向（但是注意，g'的方向和 g 方向应该是差别不大的）。其实就是每一次更新 g 都和之前的 g1，g2 进行一个内积的运算，如果运算结果是正的话，我们就进行更新，如果运算结果是负的话，我们就调整一下 g，之后按照调整之后的 g 进行更新。



第三个要解决的问题是，模型的扩张。

net2net 方法：训练新任务时，原来模型的 loss 降不下去，或者准确率无法提升，就需要在原有模型中添加神经元。又因为不能改变原来参数，所以就采用分裂原有神经元的方法，h2 和 h3 就是分裂后的产物，都是输入是 c 和 d，输出是 f/2，但是这样的话，这两个分裂的神经元梯度下降更新的参数都是一样的，没什么用。所以我们在这两个神经元中添加一定的噪声，就让它们变成两个独立的神经元了。这个方法不是每添加一个 task 就加神经元，而是在模型的准确率不提高时候才添加神经元。



Semi-supervised learning：

利用大量的易于获取的无监督数据来进行学习的方法称作半监督学习

优点：

1. 提升性能：在几乎相同的训练代价（相同的有监督数据）下，可以利用大量的无监督数据来进一步提高搜索算法性能，以搜索出更好的网络结构。

2. 降低时间：在达到相同搜索精度的情况下，通过利用大量的无监督数据，可以大大减少有监督数据的数量，以降低训练耗时。

半监督学习算法可分为以下几种：

1. 简单自训练（simple self-training）：首先训练带有标记的数据（这一步也可以理解为监督训练），得到一个分类器。然后我们就可以使用这个分类器对未标识的数据进行分类。根据分类结果，我们将可信程度较高的未标记数据及其预测标记加入训练集，扩充训练集规模，重新学习以得到新的分类器。

2. 协同训练（co-training）：其实也是 self-training 的一种。假设每个数据可以从不同的角度进行分类，不同角度可以训练出不同的分类器，然后用这些从不同角度训练出来的分类器对无标签样本进行分类，再选出可信的无标签样本加入训练集中。由于这些分类器从不同角度训练出来的，可以形成一种互补，而提高分类精度。

3. 半监督支持向量机（S3VMs）：由直推学习支持向量机（TSVM）变化而来。S3VM 算法同时使用带有标记和不带标记的数据来寻找一个拥有最大类间距的分类面。

4. 基于图论的方法：首先从训练样本中构建图。两个顶点之间的无向边表示两个样本的相似性。根据图中的度量关系和相似程度，构造 k-聚类图。然后再根据已标记的数据信息去标记为未标记数据。

简单自训练方法 ICML 2013：

主要思想就是把网络对无标签数据的预测，作为无标签数据的标签（即 Pseudo label），用来对网络进行训练，其损失函数为：

$$L = \sum_{m=1}^{n} \sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t) \sum_{m=1}^{n'} \sum_{i=1}^{C} L(y'^m_i, f'^m_i)$$

损失函数的前面是有标签数据的损失，后面的无标签数据的损失，在无标签数据的损失中，无标签数据的 pseudo label，是直接取网络对无标签数据的预测的最大值为标签。

α(t) 决定着无标签数据的代价在网络更新的作用，太大性能退化，太小提升有限。在初始训练时，α要设为 0，然后再慢慢增加。

LadderNet：

LadderNet 是有监督算法和无监督算法的有机结合。很多半监督深度学习算法是用无监督预训练这种方式对无标签数据进行利用的，但事实上，这种把无监督学习强加在有监督学习上的方式有缺点：两种学习的目的不一致，其实并不能很好兼容。

无监督预训练一般是用重构样本进行训练，其编码（学习特征）的目的是尽可能地保留样本的信息。而有监督学习是用于分类，希望只保留其本质特征，去除不必要的特征。

LadderNet 通过 skip connection 解决这个问题，通过在每层的编码器和解码器之间添加跳跃连接（skip connection），减轻模型较高层表示细节的压力，使得无监督学习和有监督学习能结合在一起，并在最高层添加分类器，LadderNet 就成了一个半监督模型。

LadderNet 有机地结合了无监督学习和有监督学习，解决兼容性问题，发展出一个端对端的半监督深度模型。

Meta learning：

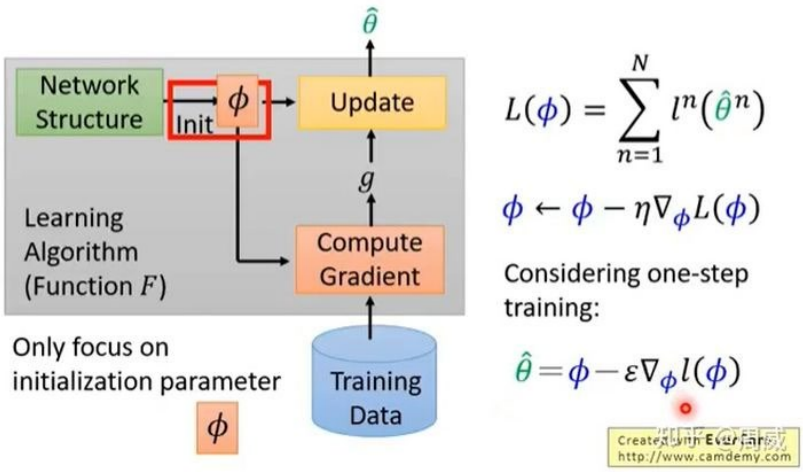
与 Lifelong learning 不同，Meta learning 是希望在不同任务上机器都能自己学会一个模型，而前者是希望学习到一个模型可以处理多个任务。如果机器能够自己的初始化参数，自己设定更新方法，就在一定程度上实现了 Meta learning 。

Machine learning 和 Meta learning 的区别：前者输入不同的图片或者文字数据，学习固定模型的参数，后者输入不同的任务，通过 F 产生的模型 f 在不同任务上的表现，更新 F 使其能产生更好的模型 f。

MAML：

MAML（Model-Agnostic Meta-Learning），意思就是模型无关的元学习。它是在一个固定的模型框架下学习一个最好的初始化的方法，有了这个初始化参数后，我们只需要少量的样本就可以快速在这个模型中进行收敛。

MAML 中是存在两种梯度下降的，也就是 gradient by gradient。第一种梯度下降是每个 task 都会执行的，用于更新经过Φ初始化后的网络参数，称为 inner loop update。而第二种梯度下降只有等 batch size 个 task 全部完成第一种梯度下降后才会执行，用于更新生成初始化参数Φ的网络，称为 outer loop update。经过这两种更新，就可以得到能生成初始化参数Φ的网络。具体过程如下：

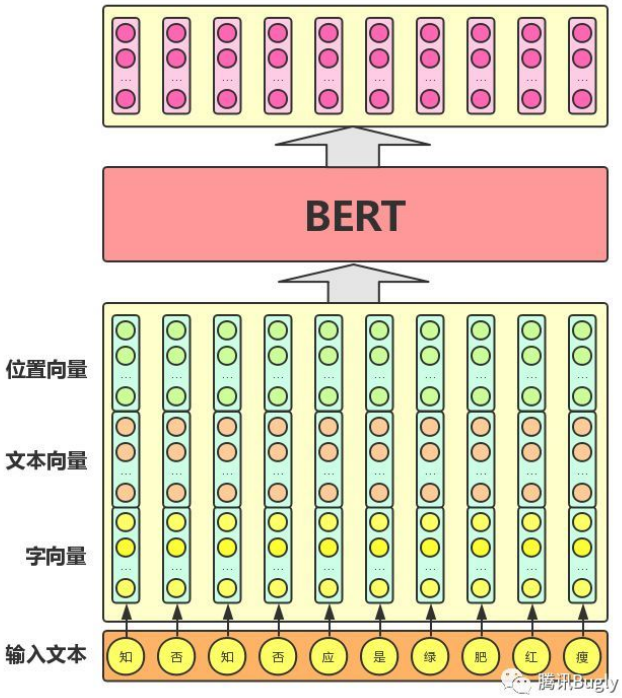


Reptile：

Reptile 对 MAML 进行了优化，训练过程中不断地用每个 task 的训练数据进行 inner loop update，那么当该 task 的 k 个子集全部训练结束后，网络初始化的参数Φ就更新为Φ‘。在 Reptile 中，更新 slow weight 的方向并不是像 MAML 中由总损失 Loss 对初始化参数的导数决定的，而是直接取Φ-Φ’的方向作为 slow weight 更新的方向，大大提高了效率。

Bert：

BERT= 基于 Transformer 的双向编码器表征，顾名思义，BERT 模型的根基就是 Transformer，来源于 attention is all you need。其中双向的意思表示它在处理一个词的时候，能考虑到该词前面和后面单词的信息，从而获取上下文的语义。



从上图中可以看出，BERT 模型通过查询字向量表将文本中的每个字转换为一维向量，作为模型输入；模型输出则是输入各字对应的融合全文语义信息后的向量表示。此外，模型输入除了字向量，还包含另外两个部分：

1. 文本向量：该向量的取值在模型训练过程中自动学习，用于刻画文本的全局语义信息，并与单字/词的语义信息相融合

2. 位置向量：由于出现在文本不同位置的字/词所携带的语义信息存在差异，因此，BERT 模型对不同位置的字/词分别附加一个不同的向量以作区分。

模型的预训练任务：

1. Masked Language Model (MLM)

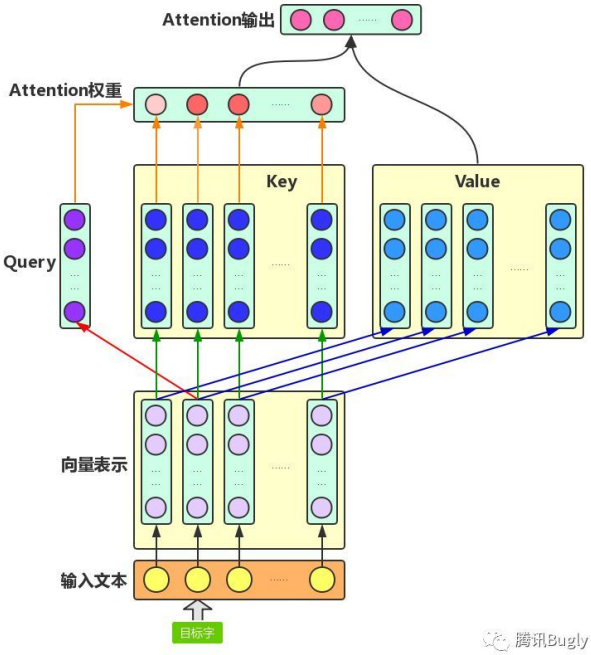
   为了实现深度的双向表示，使得双向的作用让每个单词能够在多层上下文中间接的看到自己。文中就采用了一种简单的策略，也就是 MLM。MLM：随机屏蔽掉部分输入 token，然后再去预测这些被屏蔽掉的 token。Masked LM 的任务描述为：给定一句话，随机抹去这句话中的一个或几个词，要求根据剩余词汇预测被抹去的几个词分别是什么。

2. Next Sentence Prediction

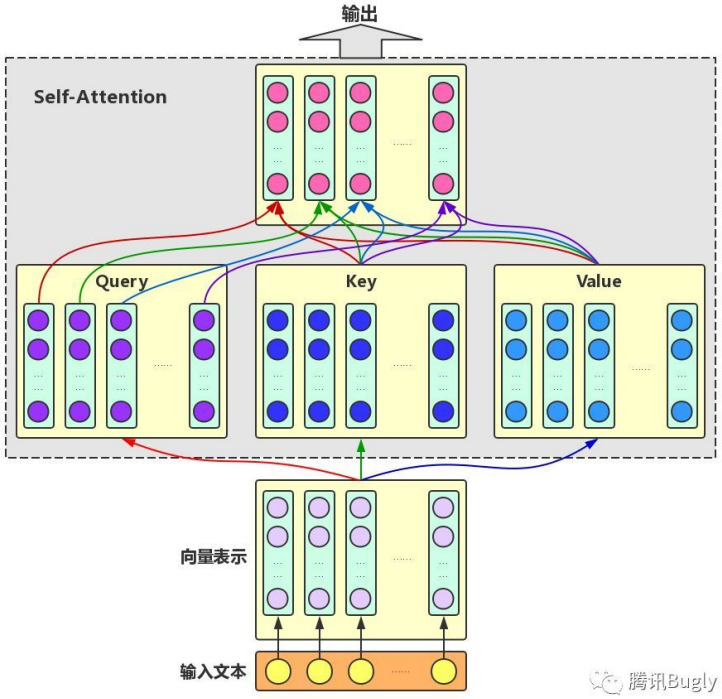
   Next Sentence Prediction 的任务描述为：给定一篇文章中的两句话，判断第二句话在文本中是否紧跟在第一句话之后，如下图所示。

模型架构：

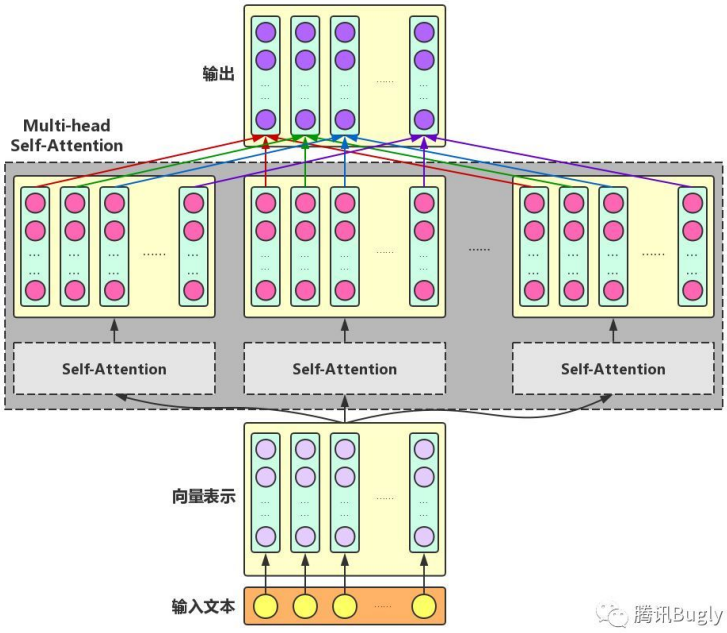
Attention 机制主要涉及到三个概念：Query、Key 和 Value。在上面增强字的语义表示这个应用场景中，目标字及其上下文的字都有各自的原始 Value，Attention 机制将目标字作为 Query、其上下文的各个字作为 Key，并将 Query 与各个 Key 的相似性作为权重，把上下文各个字的 Value 融入目标字的原始 Value 中。如下图所示，Attention 机制将目标字和上下文各个字的语义向量表示作为输入，首先通过线性变换获得目标字的 Query 向量表示、上下文各个字的 Key 向量表示以及目标字与上下文各个字的原始 Value 表示，然后计算 Query 向量与各个 Key 向量的相似度作为权重，加权融合目标字的 Value 向量和各个上下文字的 Value 向量，作为 Attention 的输出，即：目标字的增强语义向量表示。



Self-Attention: 对于输入文本，我们需要对其中的每个字分别增强语义向量表示，因此，我们分别将每个字作为 Query，加权融合文本中所有字的语义信息，得到各个字的增强语义向量，如下图所示。在这种情况下，Query、Key 和 Value 的向量表示均来自于同一输入文本，因此，该 Attention 机制也叫 Self-Attention。



Multi-head Self-Attention:为了增强 Attention 的多样性，文章作者进一步利用不同的 Self-Attention 模块获得文本中每个字在不同语义空间下的增强语义向量，并将每个字的多个增强语义向量进行线性组合，从而获得一个最终的与原始字向量长度相同的增强语义向量，如下图所示。

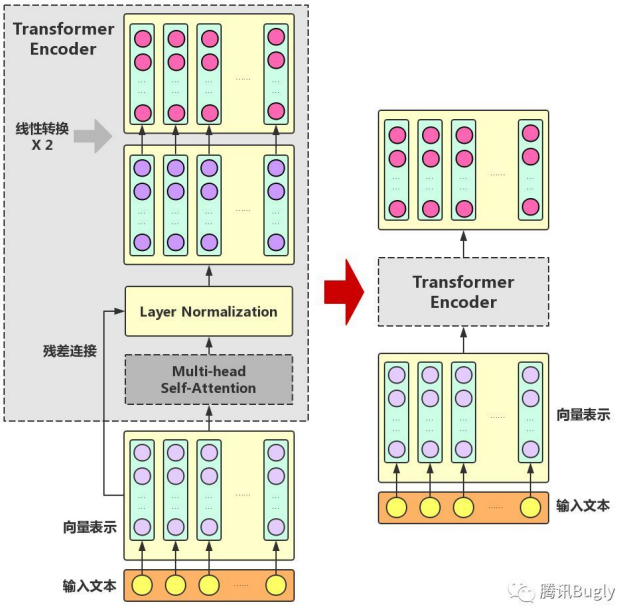


在 Multi-headSelf-Attention 的基础上再添加一些模块，就是 Transformer Encoder。实际上，Transformer 模型还包含一个 Decoder 模块用于生成文本，但由于 BERT 模型中并未使用到 Decoder 模块，因此这里对其不作详述。下图展示了 Transformer Encoder 的内部结构，可以看到，Transformer Encoder 在 Multi-head Self-Attention 之上又添加了三种关键操作：

残差连接（Residual Connection）：将模块的输入与输出直接相加，作为最后的输出。这种操作背后的一个基本考虑是：修改输入比重构整个输出更容易。

Layer Normalization：对某一层神经网络节点作 0 均值 1 方差的标准化。

线性转换：对每个字的增强语义向量再做两次线性变换，以增强整个模型的表达能力。这里，变换后的向量与原向量保持长度相同。



组装好 Transformer Encoder 之后，再把多个 Transformer Encoder 一层一层地堆叠起来，就得到了 BERT。