**学习进度**

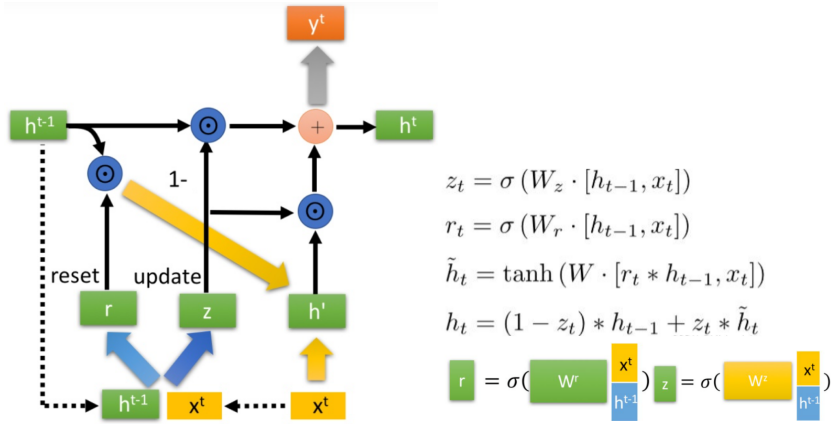
本周学习任务：

学习李沐老师《动手学深度学习（PyTorch版）》，关于**循环神经网络**部分的内容。

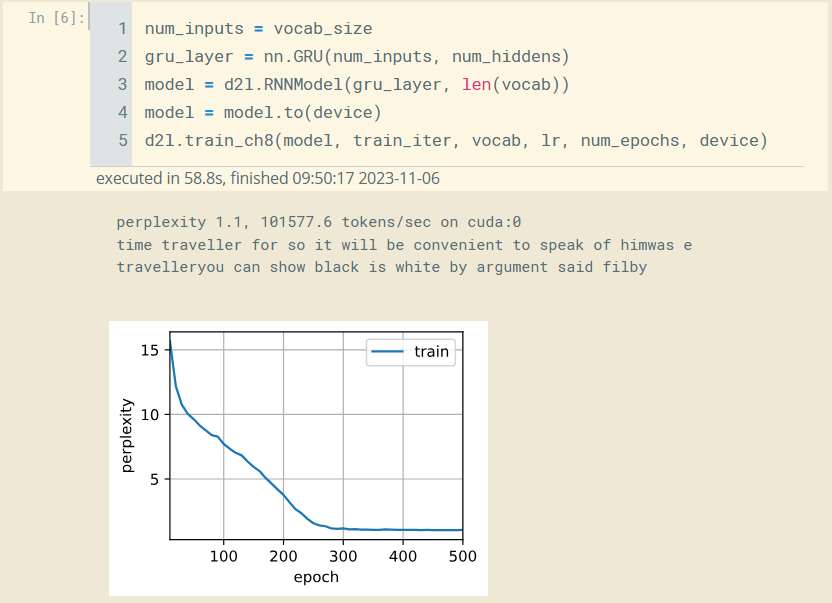
**一、GRU**

1. 应知必会

门控循环单元（GRU）由重置门及更新门构成，前者控制前层隐藏状态与当前输入的结合程度，后者控制本层记忆传递到未来的程度。由此，GRU凭借可学习门控机制，用于控制隐藏状态的更新及重置，缓解了重要历史信息的长程依赖问题及序列内部逻辑中断问题，亦或者说缓解了传统RNN模型架构的梯度消失或爆炸问题。



1. 实验结果【代码省略，详见：李沐《动手学深度学习（PyTorch版）》P343】

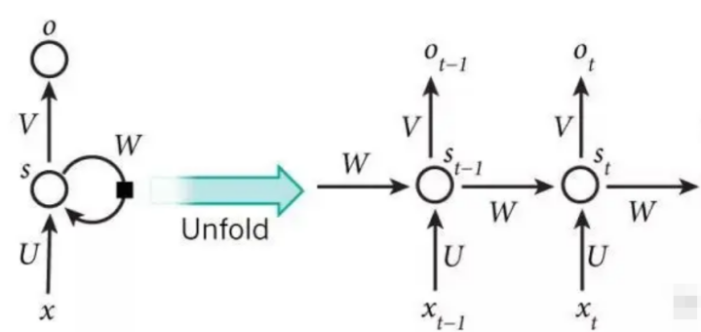


1. 相关思考
2. 如何理解隐藏状态？

隐藏状态是历史信息的编码表示。在分时间步处理过程中，隐藏状态负责记忆输入序列已知部分的重要信息，并参与新隐藏状态的计算及当前步的输出，从而提高了模型理解序列数据的能力。其中，隐藏层单元数决定隐藏状态的长度，批次大小决定隐藏状态的宽度。

1. **如何理解参数共享？**

由于深度网络参数较多，依赖参数共享来减少参数数量，是较为常见的策略。在CNN中，依赖卷积核完成了空间上的参数共享；在RNN中，依赖计算单元的分步滑动，完成了时序上的参数共享。对于RNN模型架构而言，在分步处理输入序列的过程中，计算单元中的映射权重U、V、W是共享的。这种方式即能减少网络的参数量级，又能够使得模型适应不同长度的序列样本，从而极大提高泛化能力。此外，计算单元本身能有效处理序列分词间的位置关系，且不同的序列分词亦遵循共同的特定语法逻辑。由此，通过参数共享数据逻辑模式的方式，似乎是合理的。



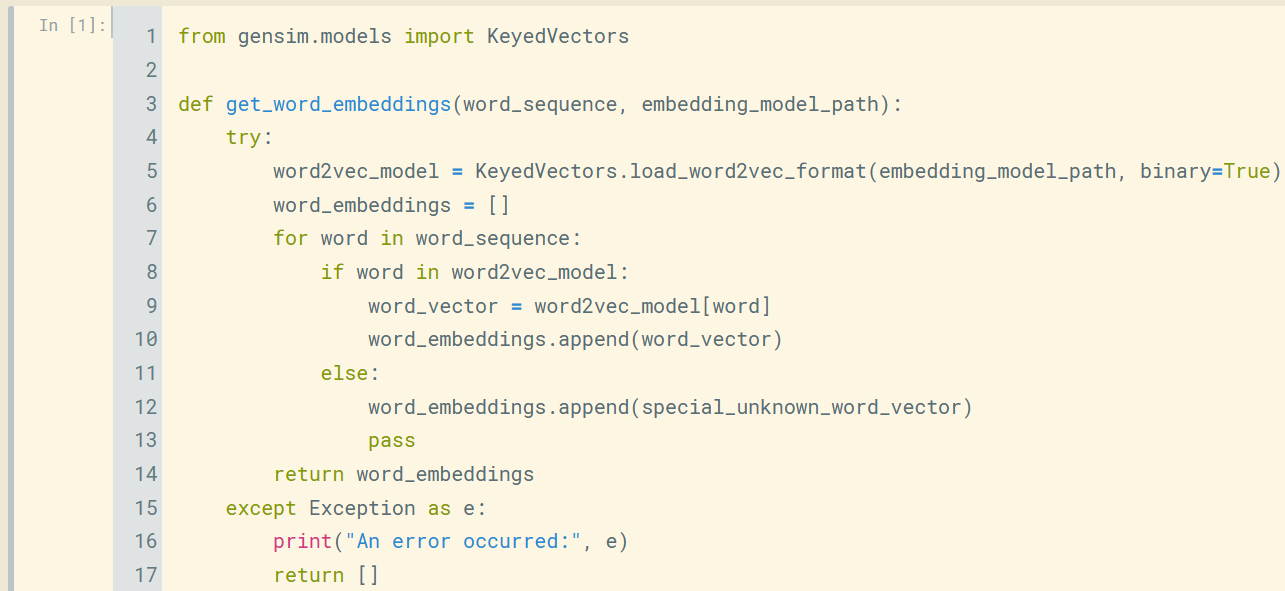
1. 如何理解输入序列的维度？

若将同批次的所有子序列理解为一个三维的输入序列，每个子序列都可看作“时间步数\*词嵌入向量长度”维数的特征图，通道数则指该批次的子序列数。故此，输入序列的维度指：批次大小\*时间步数\*词嵌入向量长度。其中，时间步数指输入序列的长度（比如句子中的单词数量），亦为分步滑动的次数。而词嵌入向量的长度作为超参，决定了每个单词在词嵌入空间中的表示维度。较大的嵌入维度可容纳更多的语义信息，代价是增加了模型的复杂度及计算成本；较小的嵌入维度，则又会损失一些语义信息。

1. **文本向量化表示**

在代码实践的文本向量化环节，需完成如下工作：构建专门的词汇映射表、截取文档序列数据、子序列的独热编码表示。对于构建词汇映射表而言：首先将每条文本行看作字符串，按序存储到列表中；然后依赖词元化方法（分词器或N-gram分词等），将每个字符串拆解到词元列表中；最后基于词频将所有非重词元映射到数字索引，从而获得额定词频范围（低频词元可拆分为字符词元，亦可通过<unknow>标识）内的词汇映射表。对于截取文档序列数据而言，需要将任意长的文档序列划分为具有相同步数的子序列，从而截取出用于训练的输入序列及对应标签。具体地说，可采用随机采样或顺序分区的方法，从文本序列中截取出划分批次的子序列集合及对应的标签序列集。其中，子序列数等于⌊(序列总长−1) /步长 ⌋。对于子序列的独热编码表示而言，需要依赖事先构建的词汇映射表，获得所有子序列分词的词元索引，从而完成基于独热编码的分词向量化表示。

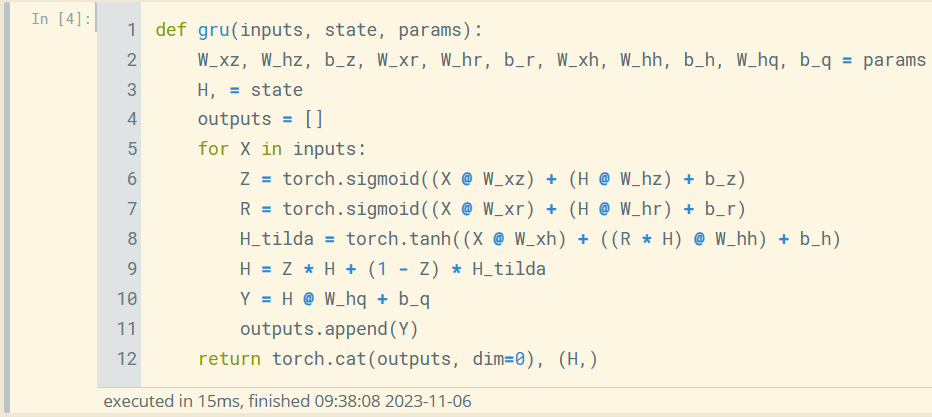
然而，若词汇映射表过长，基于独热编码的分词向量化表示，存在稀疏性问题。为此，可以简便地使用预训练的词嵌入模型，直接检索完成分词的嵌入表示，代码如下：



然而，预训练的词嵌入模型方法，缺乏在训练过程中微调词嵌入矩阵的灵活性。为此，可以在网络中引入嵌入层，从头学习词嵌入矩阵或继承预训练的词嵌入矩阵进行微调学习，将独热编码表示的分词同训练好的词嵌入矩阵相乘，从而获得分词对应的词嵌入向量表示。

1. 如何自定义GRU计算单元？

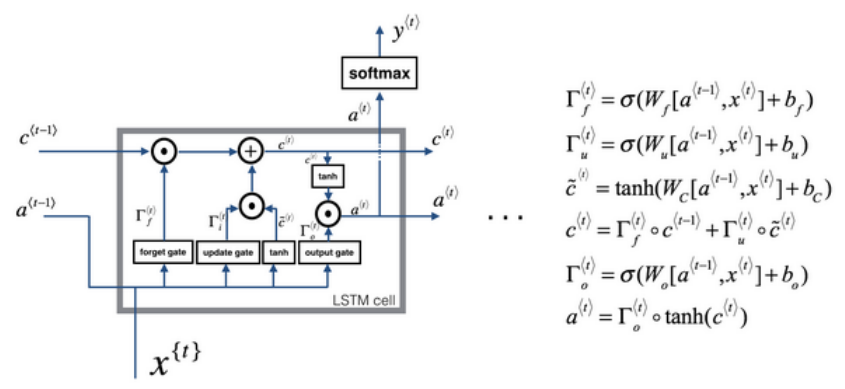
如下代码，实现了GRU计算单元。其中，输入序列inputs逐步被处理，输出序列outputs被逐步构建。在每个时间步中：首先，通过输入X和前步隐藏状态H的线性组合及偏置项，计算更新门Z和重置门R；然后，使用重置门R来控制计算候选记忆H\_tilda，并使用双曲正切函数作为激活函数；最后，使用更新门Z和候选记忆H\_tilda计算出新的隐藏状态H，并计算输出Y，输出被添加到outputs中。



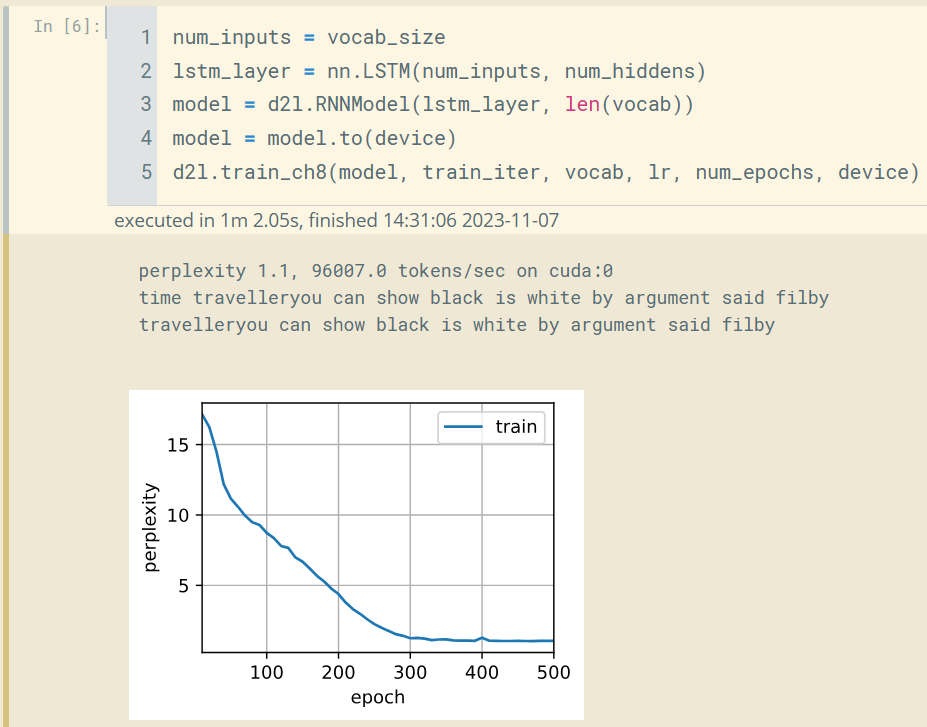
**二、LSTM**

1. 应知必会

长短期记忆（LSTM）单元主要由遗忘门、输入门及输出门构成，接收并维护隐藏状态（输出单元）及细胞状态（记忆单元）。其中，遗忘门控制前层细胞状态的遗忘程度，输入门控制候选细胞状态的记忆程度，输出门控制当前细胞状态作为新隐藏状态的传递程度。相较于GRU，LSTM通过细胞状态和隐藏状态间的交互，弥补了单靠隐藏状态难以保持和传递长期记忆的缺陷，从而更灵活地捕捉并维持了序列的长期依赖关系。



1. 实验结果【代码省略，详见：李沐《动手学深度学习（PyTorch版）》P350】



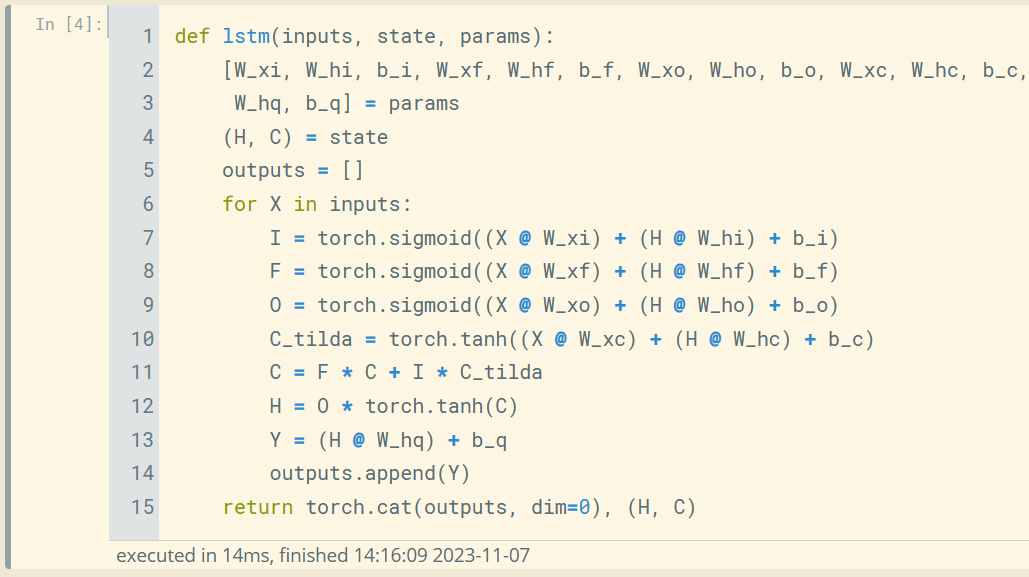
1. 相关思考
2. **如何理解GRU与LSTM的区别？**

GRU不存在细胞状态，仅靠通过接收并维护隐藏状态的方式，同时捕获长、短期记忆。LSTM则额外引入细胞状态作为长期记忆单元，通过细胞状态与隐藏状态的交互，更灵活地捕获和传递了长、短期记忆，弥补了单一隐藏状态缺乏有效传递长期记忆的缺陷。

相较于GRU，由于LSTM存在更多的门控组件且额外维护细胞状态，使得LSTM通常具有更多的参数，能够更灵活地控制信息流，从而具有更强的表达能力，更擅长处理需要长期依赖关系的序列数据。当然，由于LSTM模型架构更为复杂，需要更多的计算资源，使得LSTM计算效率往往低于GRU。

1. 如何自定义LSTM计算单元？

如下代码，实现了LSTM计算单元，用于处理输入序列inputs。在每个时间步中：首先，依赖输入X和前步隐藏状态H，计算输入门I、遗忘门F、输出门O及候选细胞状态C\_tilda；然后，依赖前述门控组件，处理前层细胞状态及本层候选细胞状态，更新出当前的新细胞状态C及新隐藏状态H（新细胞状态C通过输入门I来增加新信息，并通过遗忘门F来保留或遗忘旧信息；新隐藏状态H是新细胞状态的非线性变换，并通过输出门O来控制输出）；最后，使用新隐藏状态H计算输出Y，并将结果添加到outputs中。



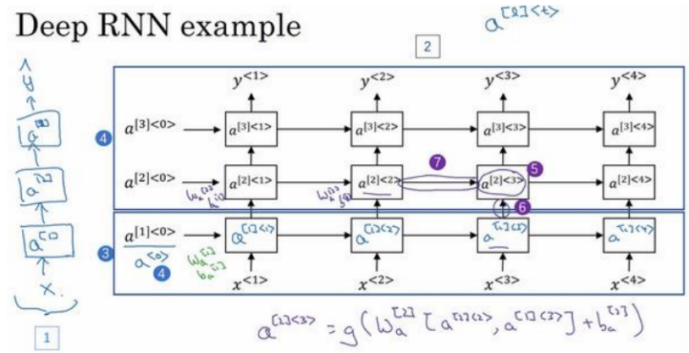
1. **如何理解“只有隐藏状态会传递到输出层，细胞状态完全属于内部信息”？**

细胞状态用于处理序列的长期记忆；隐藏状态用于处理序列的短期记忆及当前步的输入，以便于向外部输出或向后续时间步传递信息。这种分离式设计思想，使得LSTM能够更好地处理序列数据中的长期依赖关系，且保持了对当前信息的敏感性。细胞状态所谓的内部性质，意味着它不直接参与网络输出层的计算。相反，如自定义LSTM单元的代码实现所示，隐藏状态H直接参与了输出运算。显而易见的是，内、外部信息的分离，有助于LSTM在各种序列建模任务中表现出色。

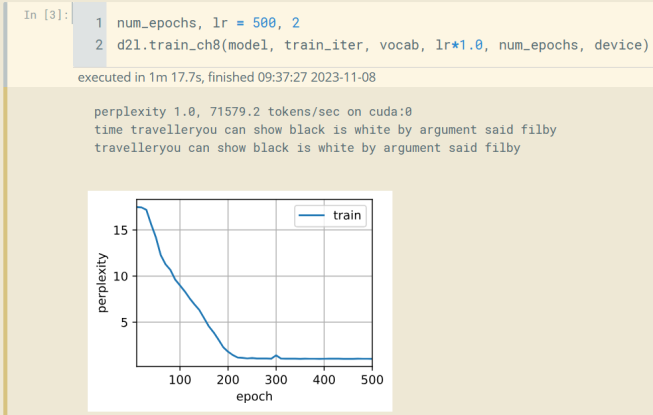
**三、Deep RNNs**

1. 应知必会

深层循环神经网络（Deep RNNs）通过堆叠隐藏层的方式，尝试提高模型的学习能力。值得注意的是，Deep RNNs中的隐藏状态，需要被传递到当前层的下⼀时间步和下⼀层的当前时间步。此外，考虑到模型的复杂程度且防止过拟合问题，很少堆叠太多有水平连接的RNN隐藏层，更常见的做法是在更深处的隐藏层上挂载无水平连接的纵向单向网络。



1. 实验结果【代码省略，详见：李沐《动手学深度学习（PyTorch版）》P355】



1. 相关思考
2. 同等实验条件下，为什么Deep RNNs收敛所需的迭代周期更少？

相较于GRU及LSTM，在同等实验条件下，Deep RNNs仅需迭代训练约200周期就能接近收敛。由于Deep RNNs具有更深的隐藏层数，提供了更多的计算单元及非线性变换容量，能够更好地捕捉和表示数据特征，网络收敛所需的迭代周期自然更少。

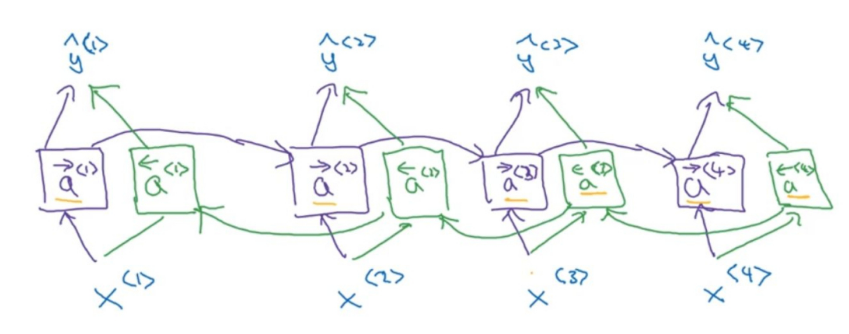
1. **如何理解隐藏层单元数与隐藏层数？**

隐藏层数决定了网络深度，表示网络中堆叠的隐藏层数量。隐藏层数越大，意味着每个时间步需要连接更多的计算单元处理输入及前步隐藏状态。而在Deep RNNs网络中，不同的隐藏层可以存在不同数量的神经单元用于处理隐藏状态，隐藏层单元数（超参）决定了隐藏层在处理当前时间步输入数据时的学习能力，越多则模型表征能力越强。此外，某层隐藏层的隐藏状态长度，在数值上等于该层隐藏层中的神经单元数。

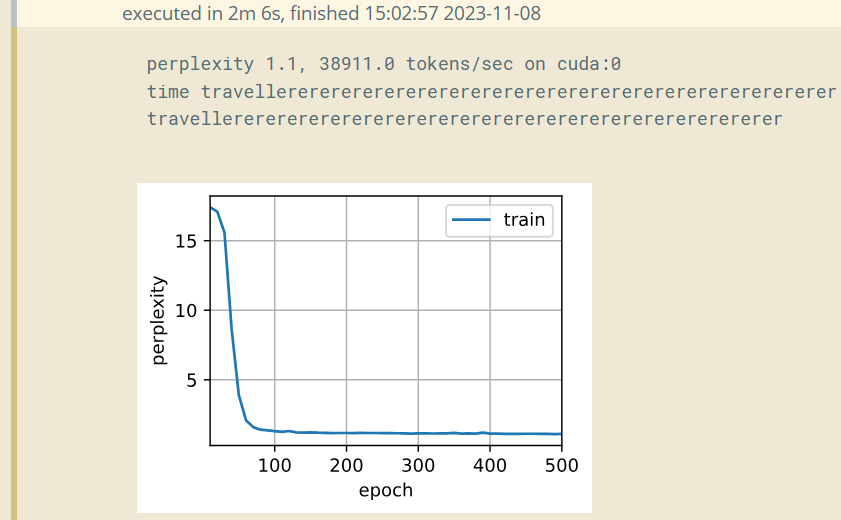
**四、Bidirectional RNN**

1. 应知必会

双向循环神经网络（Bidirectional RNN，BiRNN）通过同时考虑正、逆向激活值的方式，在每个时间步中有效捕捉了上下文信息，弥补了传统RNN网络缺乏考虑后文信息的缺陷。值得注意的是，BiRNN需要反向计算出所有的逆向激活值后，才能开始分步给出预测结果，这需要一次性给模型投喂完整的输入序列。此外，BiRNN也可以使用RNN、GRU及LSTM等基本计算单元，完成模型网络架构的搭建。



1. 实验结果【代码省略，详见：李沐《动手学深度学习（PyTorch版）》P361】



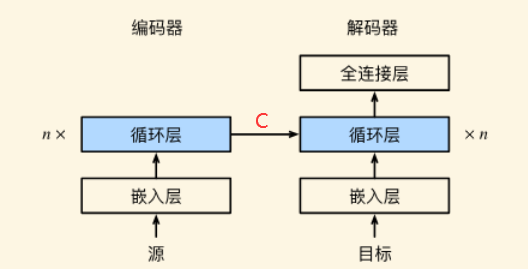
1. 相关思考
2. BiRNN的缺陷

由于BiRNN复杂的网络架构，存在计算复杂度较高及调参优化工作较为困难的问题。此外，由于BiRNN需要等待逆向计算完成才能得到全部的逆向信息，不太适用于诸如实时语音识别等需要即时响应的任务。而且，由实验结果可知，当投喂训练的文本数据不足时，BiRNN网络模型将更容易过拟合数据中的噪声信息，从而导致表象上困惑度指标通过训练在逐渐下降，然而实际生成的文本却是乱序无意义的。

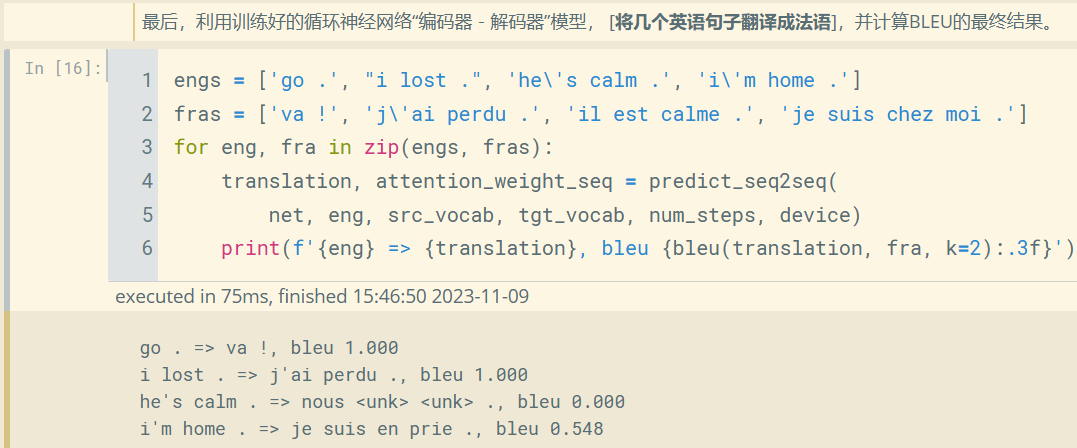
**五、seq2seq**

1. 应知必会

采用编码-解码器架构的seq2seq模型，主要用于应对输入、出都是不定长序列的情况，比如机器翻译等任务。如下图所示，编码器可以把不定长的输入序列，变换成定长的上下文变量C（包含输入序列的全局语义信息）；解码器可分时间步地接收上下文变量C及目标序列分词，从而依次给出对应的输出分量，以完成模型训练。值得注意的是，在解码器的每个时间步中，由于都依赖相同的上下文变量C，使得模型无法集中注意力关注对于当前时间步而言，相对更为重要的局部上下文信息，天然存在性能瓶颈。



1. 实验结果【代码省略，详见：李沐《动手学深度学习（PyTorch版）》P372】

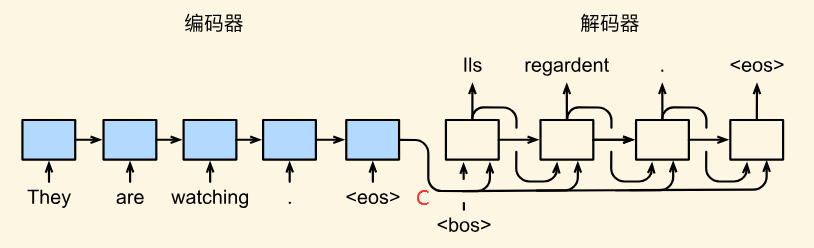


1. 相关思考
2. **如何获取及使用上下文变量C**

若使用传统的RNN网络分别实现编码器及解码器，需要通过指定时间步数的方式，规定输入序列的长度。如果某个输入序列太长，则会被截距到等同于时间步数的长度；如果某个输入序列太短，则会使用标记<pad>将序列填充至等同于时间步数的长度。解决不定长输入序列的问题后，可按照惯例运行基于RNN网络架构的编码器，并额外记忆每个时间步的隐藏状态。当记录最后一个时间步的隐藏状态后，可通过自定义函数将各个时间步的隐藏状态变换为上下文变量C，公式为：。最简单的一种变换方式是：将最终时间步的隐藏状态直接作为上下文变量的值，即：。获得编码器输出的上下文变量C后，在解码器中引用上下文变量C的方式如下：其一，将上下文变量作为解码器的初始化隐藏状态，参与解码器分时间步的运算，要求上下文变量的维数同解码器的隐藏状态适配。其二，将上下文变量和输入张量（维度：时间步数, 批次大小, 特征数）在特征维度上拼接起来，一起分时间步输入到解码器的计算单元中。

1. Decoder工作机理

针对于机器翻译任务而言。在模型训练时，采用传统RNN网络架构的解码器，将分时间步接收对应的标签序列分量，只不过第一个时间步接收的输入是开始标记<bos>，第二个时间步接收的输入才是标签序列中第一个分量。而在模型测试中，从解码器的第二个时间步开始，每个时间步接收的输入都是上一个时间步的预测输出。值得注意的是，解码器无论是在训练还是测试阶段，当某时间步接收的输入是结束标记<eos>时，或者在最终时间步完成计算后，解码器将直接结束序列生成工作。



1. 评估指标BLEU

BLEU最初作为机器翻译任务的评估指标，目前已经推广到许多应用场景中，用于简单评估输出序列与标签序列间的相似度。BLEU计算输出序列质量分数的过程是：首先，将输出及标签序列划分为若干个N-Grams对象，并分别计算分词数量；然后，循环检索输出序列所有N-Grams对象，同标签序列N-Grams对象的匹配比例；最后，根据匹配比例及分词对象数量进行加权计算，从而求出BLEU Score（BLEU Score取值介于0-1之间，越高则表示输出序列同标签序列越相似，质量则越好）。