1 提交文件简要说明

本次实验使用实验室服务器进行训练，代码在超算中提供的代码框架基础上进行了更改。提交文件夹中，`checkpoint` 目录下是实验运行结果，包括各种模型在不同超参数下的实验日志，不同模型以不同的文件夹区分，而模型超参数则保存在`.txt` 文件名中。`tensorboard` 文件夹下则保存了不同模型、参数下 `SummaryWriter` 的运行日志，即 `events.out.tfevents.` 前缀文件。

2 各语言模型结构的理解

2.1 RNN

传统 RNN1的引入是为了解决时间序列中相邻的观察之间具有依赖关系的问题。时间序列数据和普通的数据有着一定的差异，表现为当前的数据依赖于之前的数据，或者说，观察之间不是相互独立的。然而，传统的神经网络却将每个观察视为独立的，而 RNN 通过包含数据点之间的依赖关系将记忆的概念引入神经网络，这样的一种假设和设计就使得 RNN 能够在具有依赖关系的数据集上表现显著优于传统神经网络。通过所谓的反馈回路，一个 RNN 单元的输出也被同一单元用作输入。因此可以认为，每个 RNN 都有两个输入，即来自过去和现在的信息。通过使用过去的信息，RNN 在某种程度上实现了“短期记忆”，RNN 的每个神经元都能同时接收当前时间步的输入、前一时间步的隐藏状态和偏置组合，然后仍通过激活函数，得到当前时间的状态。RNN 的这种实现不但能够通过短期记忆来处理顺序数据并识别历史数据中的模式，还能够处理不同长度的输入。

和传统神经网络一样，RNN 也存在梯度消失的问题。由于在反向传播期间更新权重的梯度变得非常小，将权重与接近于零的梯度相乘会阻止网络学习新的权重，梯度下降消失的问题随着网络层数的增加而增加。除此之外，RNN 的设计虽然能够保留一定程度的短期记忆，但是其参数在逐层传递的时候，来自于长期的记忆的影响自然也会受到削弱。所以 RNN 会忘记在较长序列中看到的内容，只有短期记忆，缺少长期记忆。

除此之外，RNN 在使用 ReLU 激活函数的时候会更倾向于受到死亡 ReLU 单元的影响，导致停止学习。在本次实验的数据集和配置环境下，我分别使用了 Tanh 激活函数和 ReLU 激活函数的 RNN 进行实验，发现在使用 ReLU 激活函数的情况下无法进行正常的训练，表现为训练时的 loss 和 ppl 都为 `nan`12。故而更推荐使用 Tanh 激活函数进行训练。

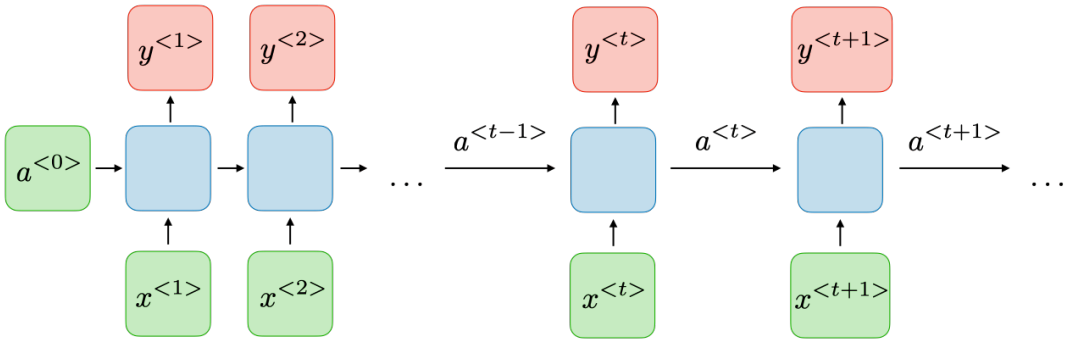


图 1: 传统 RNN 网络结构

2.2 LSTM

LSTM2是一种特殊类型的 RNN，它在一定程度上解决了 RNN 会梯度消失的问题，还能提供对长期记忆的理解和表达。LSTM 的关键通过被称为遗忘门、输入门和输出门的三个门来控制信息上下文在神经元之间的流动。遗忘门决定了应该保留多少长期记忆，可以使用 sigmoid 函数来控制信息的保留。通过 sigmoid 函数将输入映射到在 0 和 1 之间变化的输出，0 即不保留任何信息，1 则保留单元状态的所有信息。输入门决定将哪些信息添加到单元状态，从而添加到长期记忆中；输出门决定单元状态的哪些部分构建输出，从这个角度来讲，输出门负责短期记忆。总的来说，状态通过遗忘门和输入门更新。

LSTM 的关键是单元状态，在单元的输入传递到输出的过程中，单元状态允许信息沿着整个链流动，仅通过三个门进行较小的线性动作。通过三个门控，以过滤器的方式控制信息流并确定保留或忽略哪些信息。LSTM 的主要优点是它可以同时捕获序列的长期和短期模式，但由于结构更复杂，LSTM 的计算成本更高，也就意味着训练时间更长。

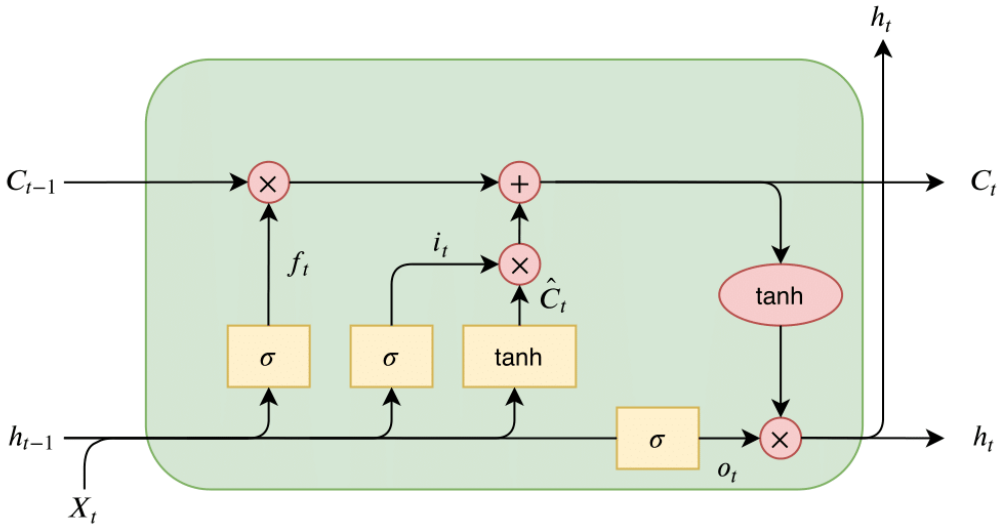


图 2: LSTM 网络结构

2.3 GRU

与 LSTM 类似，GRU3同样可以解决传统 RNN 的梯度消失问题。然而，与 LSTM 的不同之处在于 GRU 使用较少的门并且没有单独的内部存储器，或者说单元状态。从这个角度来讲，GRU 完全依赖隐藏状态作为记忆，这意味着 GRU 有着比 LSTM 更简单的架构。GRU 内部使用重置门和更新门来分别对长短期信息进行控制。重置门负责短期记忆，它决定保留和忽略多少过去的信息；更新门负责长期记忆，类似于 LSTM 的遗忘门。

当某一时间步需要更新隐藏状态时，他通过下述两个步骤来实现：首先，确定所选择的隐藏状态，即当前输入和前一时间步的隐藏状态以及激活函数的组合。前一个隐藏状态对当前所选择的隐藏状态的影响由重置门控制；然后，将所选择隐藏状态与上一时间步的隐藏状态相结合，生成当前隐藏状态。先前的隐藏状态和所选择隐藏状态如何组合由更新门决定。如果更新门给出的值为 0，则完全忽略先前的隐藏状态，这种情况下当前隐藏状态就是所选择的隐藏状态。如果更新门给出的值为 1，需要做出的处理则完全相反。由于与 LSTM

相比有着更简单的架构，GRU 的计算效率更高，训练速度更快，需要更少的内存。虽然 GRU 对过去的观察结果的考虑无法达到 LSTM 的程度，但是其已被证明对于较小的序列更有效。

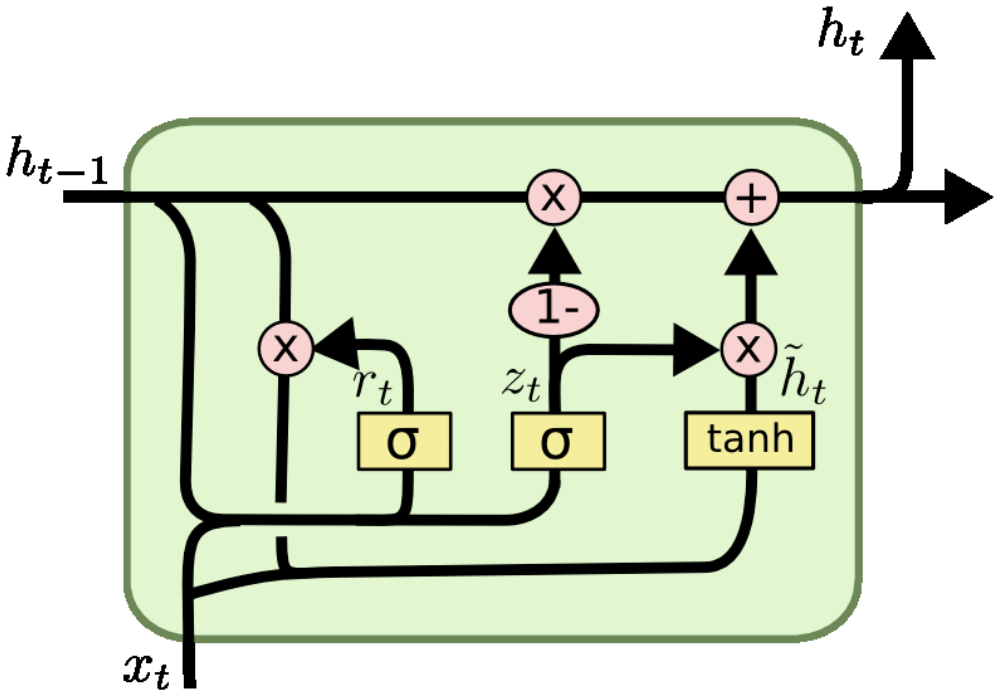


图 3: GRU 网络结构

2.4 Transformer

与 RNN 不同，Transformer4则采用了一种完全不同的范式解决问题。Transformer 是一种基于注意力机制的序列到序列模型，其主要优点是能够并行处理输入序列，而不是像 RNN 那样顺序处理。这使得 Transformer 比 RNN 更快，更容易训练。Transformer 中没有了序列这个概念，通过将输入的语句当作一个整体传入 embedding 层中，赋予模型并行计算的能力，因为不再强调输入序列次序，也就没有长依赖的问题。既然 Transformer 中不再使用循环结构，因此必须以不同的方式添加位置信息。为了能够捕获序列中的位置信息，Transformer 引入了 `positional_encoding` 即位置编码的概念，位置编码是一种向输入嵌入中添加位置信息的技术，它使模型能够理解输入中某些部分在整个输入中的位置。这种方式在解决了序列顺序依赖的同时，也能将位置信息纳入模型考虑。此外，Transformer 还引入了自注意力机制作为核心组件。与之前的机制不同的是，自注意力机制可以将序列中不同位置之间的依赖关系进行建模，不需要依赖于时间的顺序，因此可以更好地处理长序列。

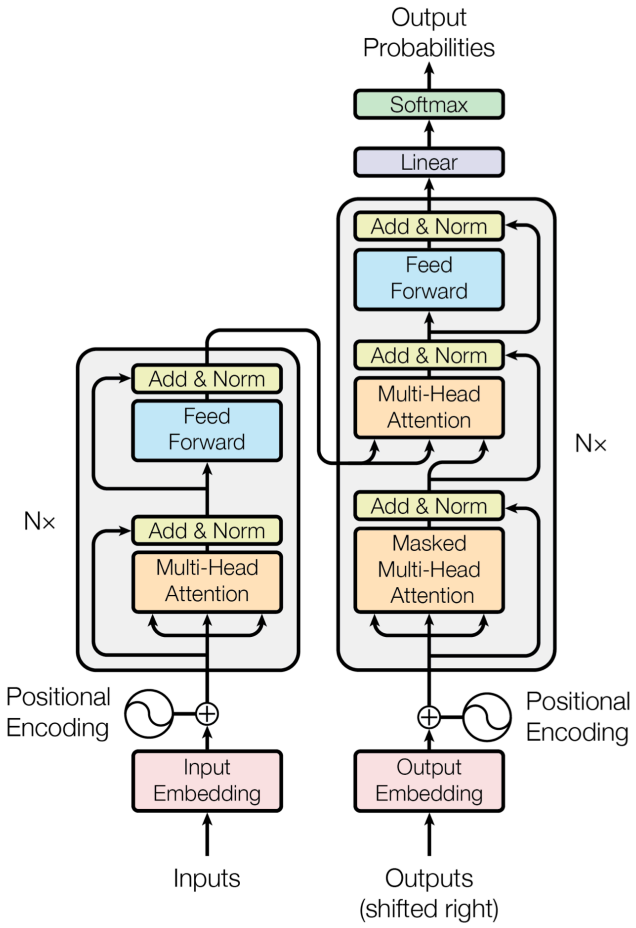


图 4: Transformer 结构

3 实验和超参数分析

本部分将从模型选取和超参数设定两个角度对实验过程进行分析。

3.1 同一参数下模型性能对比

由于本实验中我们使用了多种不同的语言模型，为了能够对比其性能的差异，应保持参数的一致性，这里列举了两种不同参数配置下的实验结果。

模型选取 / 评估指标	RNN(Relu)	RNN(Tanh)	LSTM	GRU	Transformer
Test Loss	NA	5.26	**4.95**	5.26	6.55
Perplexity	NA	192.27	**141.38**	192.27	700.61

表 1: Results with embedding size 200, hidden dimension 200

模型选取 / 评估指标	RNN(Relu)	RNN(Tanh)	LSTM	GRU	Transformer
Test Loss	NA	4.95	**4.90**	4.95	6.54
Perplexity	NA	141.47	**133.81**	141.47	694.68

表 2: Results with embedding size 400, hidden dimension 200

可以看到，在本次实验的数据集和配置环境下，LSTM 网络取得了最好的结果。虽然 Transformer 在大规模数据集和大多数自然语言处理的任务场景下优秀的性能已经得到了验证，但是当序列长度没有超过 RNN 的处理能力的时候，`positional_encoding` 对时序的建模能力还是不如 RNN 的。我们本次实验所使用 `gigaspeech` 数据集规模并不大，可能无法很好地发挥 Transformer 模型的优势。

3.2 不同超参数对模型性能的影响

在本次实验中，我取得的最优性能是在使用 LSTM 模型下得到的，即表格2中在测试集上的 test loss 4.90 和 perplexity 133.81。具体使用的参数配置为：embedding size: 400, hidden dimension: 200, number of layers: 2, initial learning rate: 20, bptt: 35, dropout rate: 0.2, batch_size: 20, clip: 0.25, epochs: 40, Vocabulary Size: 27710, Total number of model parameters: 17.46M.

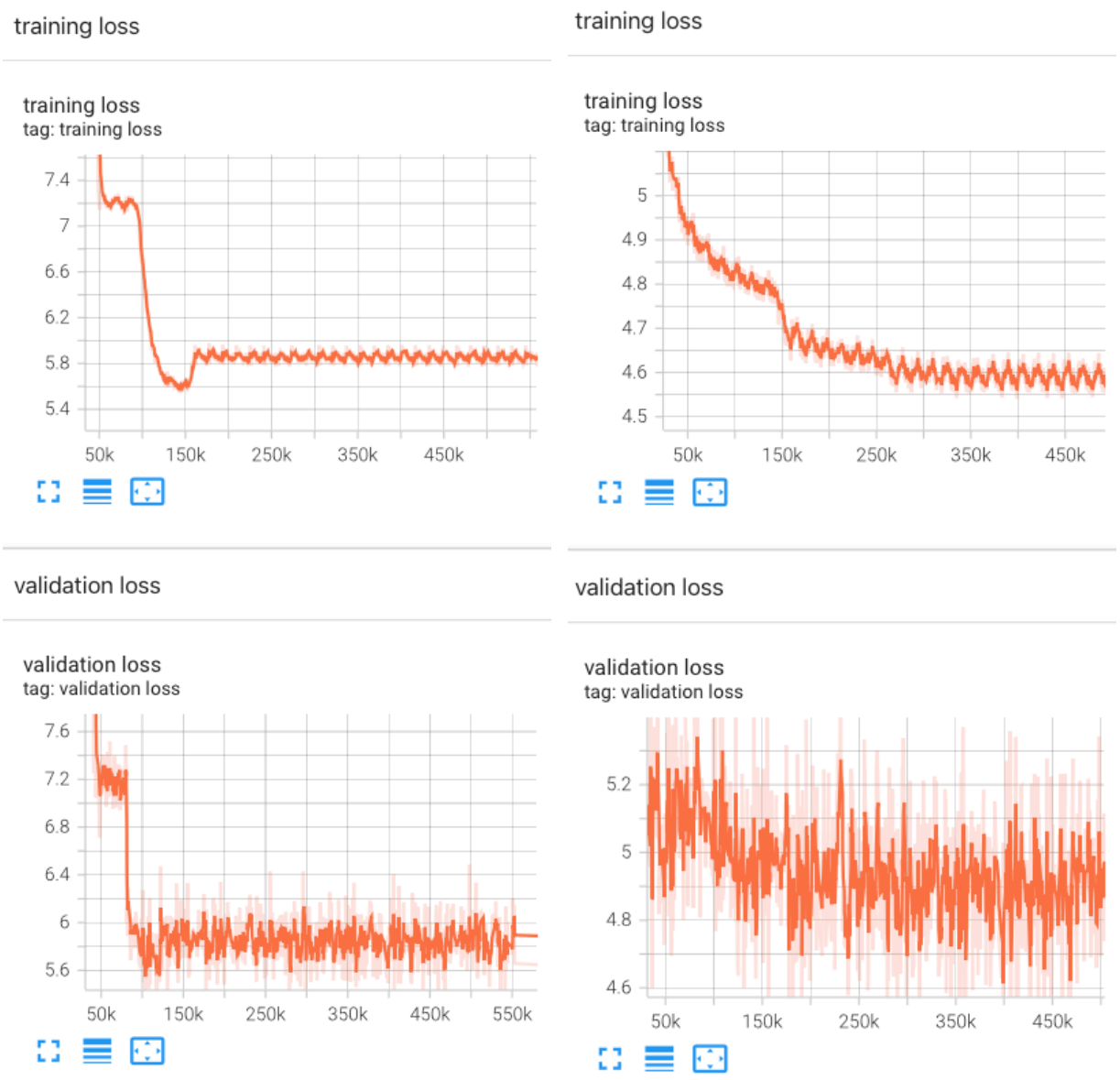
从语义信息的角度来讲，提高 embedding size 和 hidden dimension 的大小，都能够更好地表征确定数据集下的语义信息，即通过提升模型的参数量来提供更多的表征。在一定范围内，实验结果也确实表明更高的维度能够带来低的 perplexity。但是 embedding size 和 hidden dimension 也不是越大越好，过大的情况下，不但由于模型参数量的提升带来过多的训练开销，还会因为模型的参数规模并不能很好地适配数据集规模而带来过拟合等问题，导致模型并不能很好地学到属于特定数据集的特征。

在本次实验过程中，由于适配了动态学习率调整的方法，在训练的过程中会根据参数更新的情况动态减小学习率，所以初始 learning rate 并不会对最终模型的性能产生较大的影响。在模型训练的开始，可以使用比较大的学习率，帮助模型更快地参数更新。为了适当地减少过拟合的问题，设定了 drop out 几率，一定的 drop out rate 可以使神经元随机失活，从而减少过拟合，但是 drop out rate 也不能过高，需要根据经验和实际的使用场景进行设置。

4 Tensorboard 支持

为更好地观察和对比不同模型在统一参数下的运行结果，可以使用 Tensorboard 对 training 和 validation 过程中的 loss 进行绘图。使用 Tensorboard 不但能够画出训练、验证和测试过程中的 loss 趋势等信息，还能够提供实时观看的功能，而不必等到训练全部完成之后再整体查看训练图，从而能更早发现和定位训练过程出现的 bug。在使用服务器的情况下，由 tensorboard 的 `Summary Writer` 类记录结果，通过调用 `tensorboard --logdir` 命令并指定端口接收远端服务器传回的消息，并在 localhost 进行查看，所得到的训练图像见附录部分56。

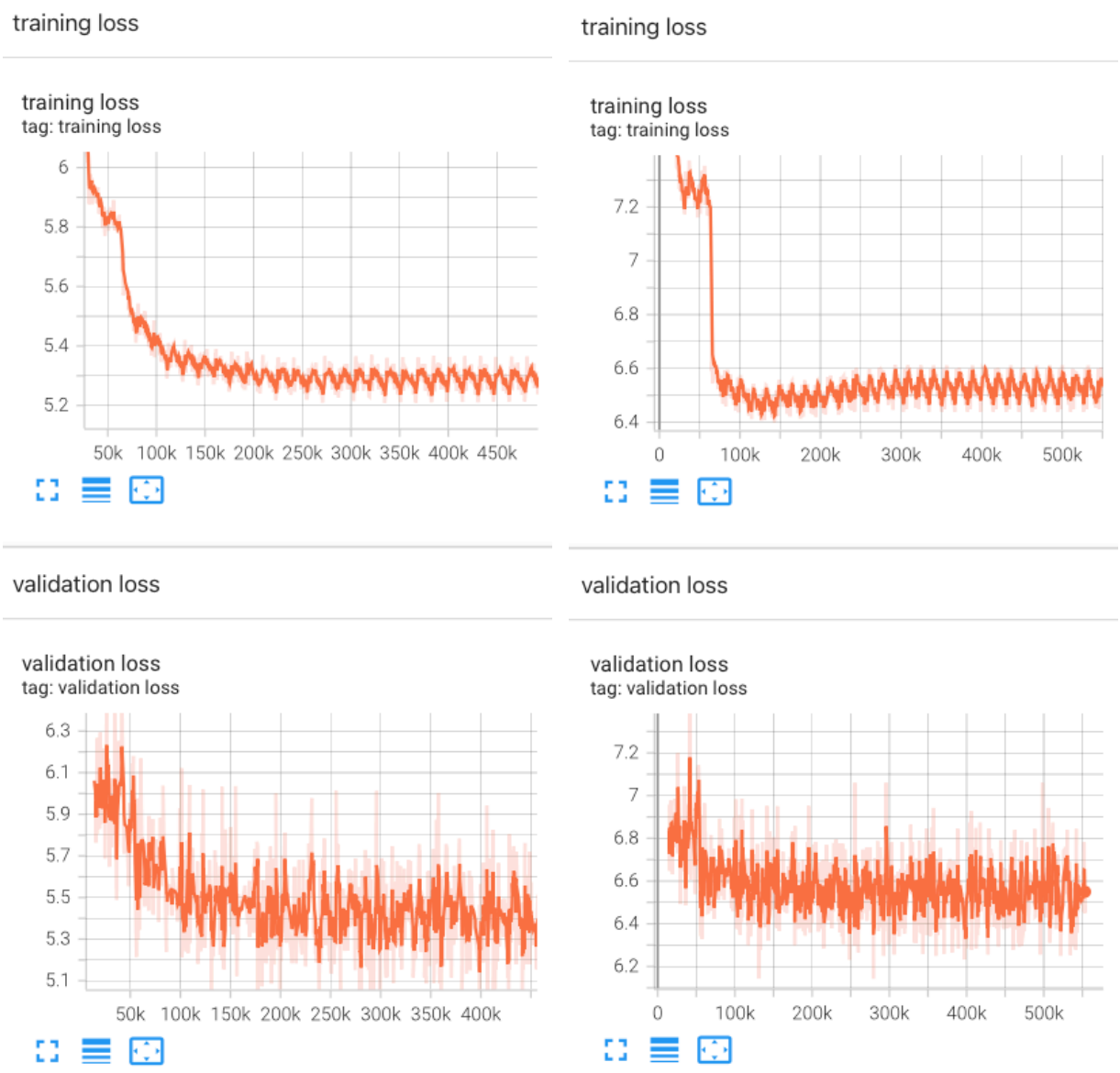
附录：Tensorboard 训练图



(a) RNN_Tanh 训练曲线

(b) LSTM 训练曲线

图 5: Tensorboard 结果展示



(a) GRU 训练曲线

(b) Transformer 训练曲线

图 6: Tensorboard 结果展示