人工智能学习笔记

1. 概念梳理

1.1. 预训练模型

在一个原始任务上预先训练一个初始模型，然后在目标任务上使用该模型。这本质上是一种迁移学习的方法。

在自己的目标任务上使用别人训练好的模型，针对目标任务的特性，对该初始模型进行精调，从而达到提高目标任务的目的。

1.1.1. 自编码语言模型（Autoencoder Language Model）

根据上文内容预测下一个可能的单词，就是常说的自左向右的语言模型任务，或者反过来也行，就是根据下文预测前面的单词。GPT 就是典型的自回归语言模型。

优点

其实跟下游NLP任务有关，比如生成类NLP任务，比如文本摘要，机器翻译等，在实际生成内容的时候，就是从左向右的，自回归语言模型天然匹配这个过程。

缺点

只能利用上文或者下文的信息，不能同时利用上文和下文的信息。

1.1.1.1. BERT

即双向Transformer的Encoder，bert并没有使用到transformer的decoder。模型的主要创新点集中在pre-train方法上，可以使用大量的无监督数据，即用了Masked LM和Next Sentence Prediction两种方法分别捕捉词语和句子级别的representation。

1.1.1.1.1. transformer

1.1.1.1.2. Masked LM（Masked Language Model）

此任务就是**随机遮盖或替换**一句话里面的任意字或词，然后让模型通过上下文预测那一个被遮盖或替换的部分，之后**做 Loss 的时候也只计算被遮盖部分的 Loss**，实际操作如下：

1. 随机把一句话中 15% 的 token（字或词）替换成以下内容：
2. 1. 这些 token 有 80% 的几率被替换成 `[MASK]`，例如 my dog is hairy→my dog is [MASK]
   2. 有 10% 的几率被替换成任意一个其它的 token，例如 my dog is hairy→my dog is apple
   3. 有 10% 的几率原封不动，例如 my dog is hairy→my dog is hairy
3. 之后让模型**预测和还原**被遮盖掉或替换掉的部分，计算损失的时候，只计算在第 1 步里被**随机遮盖或替换**的部分，其余部分不做损失。

这样做的好处是，BERT 并不知道 [MASK] 替换的是哪一个词，而且任何一个词都有可能是被替换掉的，比如它看到的 apple 可能是被替换的词。这样强迫模型在编码当前时刻词的时候不能太依赖当前的词，而要考虑它的上下文，甚至根据上下文进行 "纠错"。比如上面的例子中，模型在编码 apple 时，根据上下文 my dog is，应该把 apple 编码成 hairy 的语义而不是 apple 的语义。

1.1.1.1.3. Next Sentence Prediction

判断两个句子（A和B）是否为前后句关系，之后要在这两个句子中加一些特殊的 token：[CLS]A[SEP]B[SEP]。也就是在句子开头加一个 [CLS]，在两句话之间和句末加 [SEP]。拼接起来后，如果句子超出长度，需要从头部或者尾部截取，不能从中间截取，保证句子的完整性。最终归结为二分类问题，需要让正负例的两种情况出现的数量为 **1:1**。

- 正例：属于上下文的一对句子A和B。
- 负例：第一个句子选择A，第二个句子从其他段落中选择一个句子。

1.1.1.2. ALBert

1.1.1.3. RoBERTa

1.1.2. 自回归语言模型（Autoregressive Language Model）

自编码语言模型是对输入的句子随机Mask其中的单词，然后预训练过程的主要任务之一是根据上下文单词来预测这些被Mask掉的单词，那些被Mask掉的单词就是在输入侧加入的噪音。BERT就是典型的自编码类语言模型。

优点

它能比较自然地融入双向语言模型，同时看到被预测单词的上文和下文。

缺点

主要在输入侧引入[Mask]标记，导致预训练阶段和Fine-tuning阶段不一致的问题，因为Fine-tuning阶段是看不到[Mask]标记的。而Bert这种自编码模式，在生成类NLP任务中，就面临训练过程和应用过程不一致的问题，导致生成类的NLP任务到目前为止都做不太好。

1.1.2.1. ELMO

1.1.2.2. XLnet

1.1.2.3. GPT1

2. 机器学习的分类

2.1. 回归

例子：

1. 资产定价（Asset Pricing）
2. 多资产趋势跟踪策略（Multi-Asset Trend Following Strategies）
3. 隐含波动率估计（Implied-Volatility Estimation）

2.1.1. 岭回归（Ridge Regression）

2.1.2. Lasso回归（Lasso Regression）

2.1.3. 决策树回归（Decision Tree Regression）

2.1.4. 随机森林回归（Random Forest Regression）

2.2. 分类

例子：

1. 线性分类（Linear Classification）
2. 基于距离的分类器（Distance-based Classifiers）
3. 支持向量机（ Support Vector Machines）

3. 技术框架

3.1. Pytorch

3.1.1. Pytorch的优点

PyTorch不仅是最受欢迎的深度学习框架之一，而且也是最强大的深度学习框架之一。它有许多独特的优点，使其在学术界和工业界都受到广泛的关注和使用。接下来我们就来详细地探讨一下PyTorch的优点。

1. 动态计算图

PyTorch最突出的优点之一就是它使用了动态计算图（Dynamic Computation Graphs，DCGs），与TensorFlow和其他框架使用的静态计算图不同。动态计算图允许你在运行时更改图的行为。这使得PyTorch非常灵活，在处理不确定性或复杂性时具有优势，因此非常适合研究和原型设计。

2. 易用性

PyTorch被设计成易于理解和使用。其API设计的直观性使得学习和使用PyTorch成为一件非常愉快的事情。此外，由于PyTorch与Python的深度集成，它在Python程序员中非常流行。

3. 易于调试

由于PyTorch的动态性和Python性质，调试PyTorch程序变得相当直接。你可以使用Python的标准调试工具，如PDB或PyCharm，直接查看每个操作的结果和中间变量的状态。

4. 强大的社区支持

PyTorch的社区非常活跃和支持。官方论坛、GitHub、Stack Overflow等平台上有大量的PyTorch用户和开发者，你可以从中找到大量的资源和帮助。

5. 广泛的预训练模型

PyTorch提供了大量的预训练模型，包括但不限于ResNet，VGG，Inception，SqueezeNet，EfficientNet等等。这些预训练模型可以帮助你快速开始新的项目。

6. 高效的GPU利用

PyTorch可以非常高效地利用NVIDIA的CUDA库来进行GPU计算。同时，它还支持分布式计算，让你可以在多个GPU或服务器上训练模型。

综上所述，PyTorch因其易用性、灵活性、丰富的功能以及强大的社区支持，在深度学习领域中备受欢迎。

3.1.2. 3 Pytorch的使用场景

PyTorch的强大功能和灵活性使其在许多深度学习应用场景中都能够发挥重要作用。以下是PyTorch在各种应用中的一些典型用例：

1. 计算机视觉

在计算机视觉方面，PyTorch提供了许多预训练模型（如ResNet，VGG，Inception等）和工具（如TorchVision），可以用于图像分类、物体检测、语义分割和图像生成等任务。这些预训练模型和工具大大简化了开发计算机视觉应用的过程。

2. 自然语言处理

在自然语言处理（NLP）领域，PyTorch的动态计算图特性使得其非常适合处理变长输入，这对于许多NLP任务来说是非常重要的。同时，PyTorch也提供了一系列的NLP工具和预训练模型（如Transformer，BERT等），可以帮助我们处理文本分类、情感分析、命名实体识别、机器翻译和问答系统等任务。

3. 生成对抗网络

生成对抗网络（GANs）是一种强大的深度学习模型，被广泛应用于图像生成、图像到图像的转换、样式迁移和数据增强等任务。PyTorch的灵活性使得其非常适合开发和训练GAN模型。

4. 强化学习

强化学习是一种学习方法，其中智能体通过与环境的交互来学习如何执行任务。PyTorch的动态计算图和易于使用的API使得其在实现强化学习算法时表现出极高的效率。

5. 时序数据分析

在处理时序数据的任务中，如语音识别、时间序列预测等，PyTorch的动态计算图为处理可变长度的序列数据提供了便利。同时，PyTorch提供了包括RNN、LSTM、GRU在内的各种循环神经网络模型。

总的来说，PyTorch凭借其强大的功能和极高的灵活性，在许多深度学习的应用场景中都能够发挥重要作用。无论你是在研究新的深度学习模型，还是在开发实际的深度学习应用，PyTorch都能够提供强大的支持。