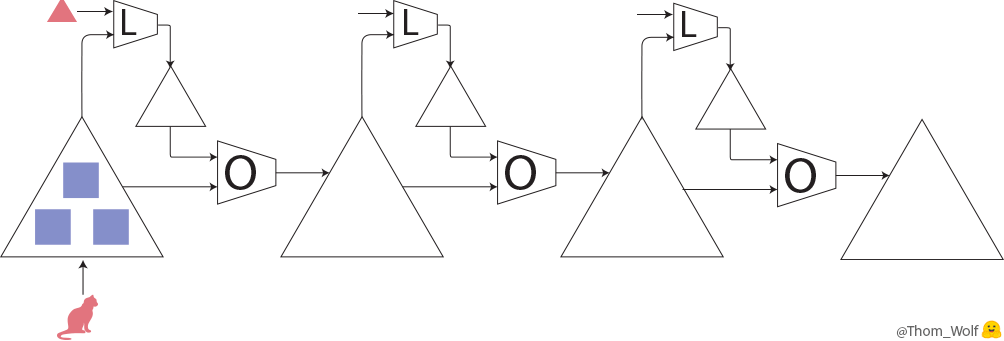
# 🐣从零到研究 - 元学习导论



元学习是机器学习领域的一个令人兴奋的研究趋势，它解决了学习的问题。

传统的机器学习研究范式是在特定的任务上获取一个庞大的数据集，并利用这个数据集从头开始训练一个模型。很明显，这与人类如何利用过去的经验快速地从一个例子手机中学习新任务相去甚远。

那是因为人类学会学习。

在过去的几个月里，我一直在用自然语言处理的元学习模型进行大量的游戏和实验，并将在🇨🇦 - 来打个招呼！👋

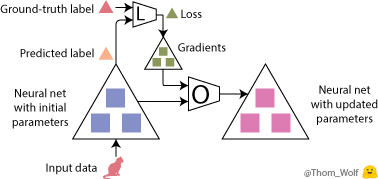
在这篇文章中，我将以一种非常直观的方式来解释什么是元学习。然后，我们将在PyTorch中编写一个元学习模型，我将分享在这个项目中学到的一些经验教训。

### 首先要学的是什么？

让我们快速了解一下当我们训练一个简单的神经网络来对猫和狗的图像进行分类时会发生什么。假设我们有一个猫的训练图像，还有一个标签，表明这个图像代表一只猫。我快速地动画了一个训练步骤，为我们节省了几千句话。

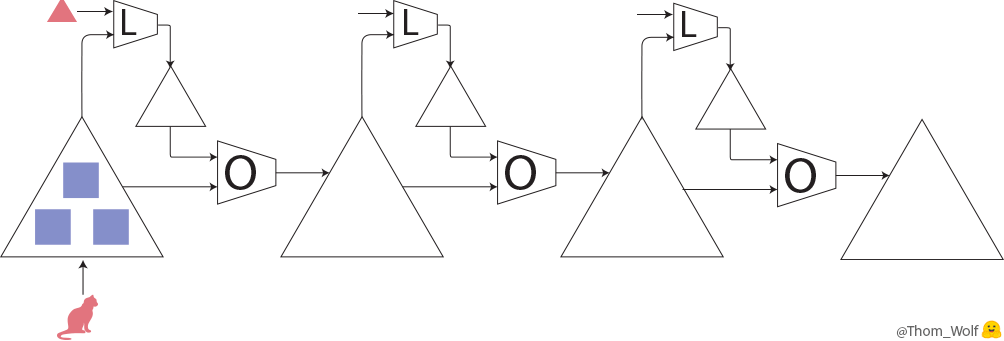
反向传球（backprop）是训练神经网络的关键步骤。由于神经网络的计算和损失是可微函数，我们可以计算应应用于神经网络的每个参数的梯度，以减少神经网络当前预测的标签和实际/目标标签之间的差异（这种差异是由损失函数测量的）。反向传播之后是优化器，它计算模型的更新参数。在这里，训练神经网络变得更像是一门艺术而不是一门科学，因为有太多可能的优化器和优化设置（超参数）。

让我们用更简洁的方式来表示我们的单一训练步骤



训练图像现在是a🐈，表示图片代表猫的标签是a🔺。大的△s是我们的神经网络，具有参数和梯度。损失函数是L-box，优化器是O-box。

然后，学习过程简单地包括重复应用优化步骤，直到我们收敛到神经网络的良好参数。

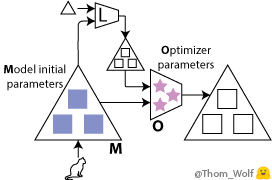


### 让我们转向元学习

元学习的思想是学习过程。

有几种方法可以实现元学习，但我想在这里描述的两种方法是学习一个类似于我们刚刚看到的学习过程。

在我们的培训过程中，我们尤其可以学到两件事：



* 神经网络的初始参数（蓝色■）和
* 优化器的参数（粉红色★）。

我将描述两个案例的组合，但每个案例本身也非常有趣，可以导致简化、加速和合理的理论结果。

现在，我们有两个模块要训练：

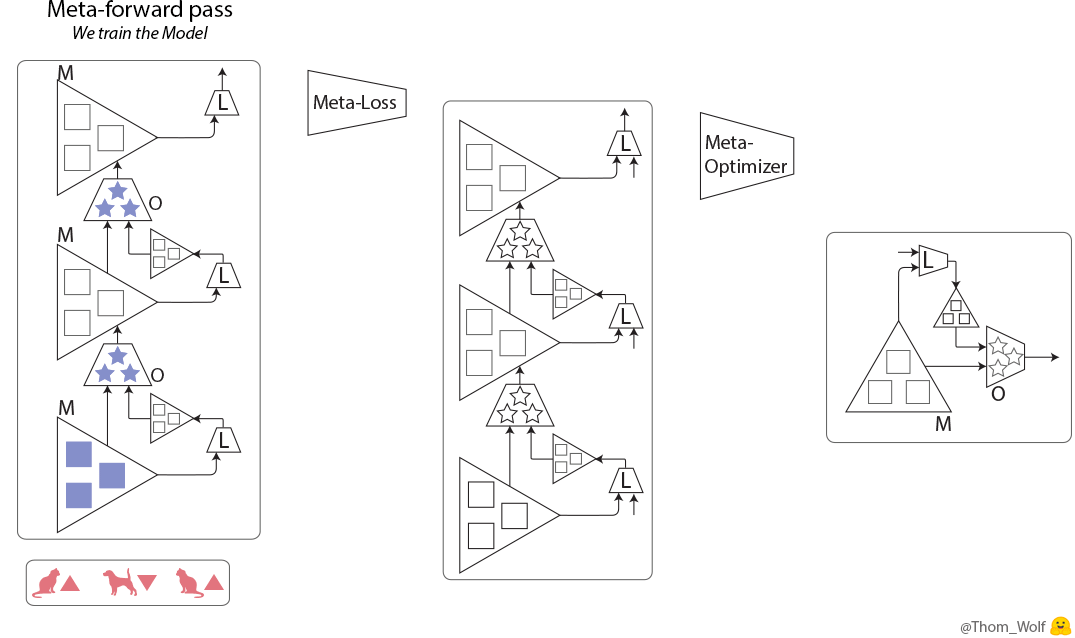
* 我称之为模型（M），这是我们以前的神经网络。它现在可以被看作是一个低级网络。它有时被称为或。模型的重量为图纸上的··。
* 优化器（O）或元学习器是更新较低层网络（模型）权值的更高层模型。优化器的权重是图纸上的★。

#### 我们如何学习这些元参数？

结果证明，我们可以沿着训练过程本身反向传播一个meta-loss梯度，回到模型的初始权重和/或优化器的参数。

我们现在有两个嵌套的训练过程：优化器/元学习者的元训练过程，其中（元-）向前传递包括模型的几个训练步骤（前面看到的向前、向后和优化步骤）。

让我们看看元训练步骤：



这里，元训练过程的一个步骤是水平表示的。它包括模型训练过程的两个步骤（垂直于元前向和元后向框）。模型的训练过程和我们刚才看到的完全一样。

如我们所见，meta forward pass的输入是一个示例/标签列表（或批处理列表），这些示例/标签在模型训练pass期间连续使用。



现在我们可以用什么样的元损失来训练元学习者呢？在模型训练的情况下，我们可以简单地将模型预测与目标标签进行比较，以获得错误信号。

在元学习者的情况下，我们希望元损失能够表明元学习者执行其任务的情况：训练模型。

一种可能是在一些训练数据上计算模型的损失，损失越小，训练效果越好。我们可以在训练结束时计算出一个元损失，甚至可以将我们在训练过程中已经计算出的模型损失合并起来（例如，通过求和）。

我们还需要一个元优化器来更新优化器的权重。在这里，它开始变得非常元，因为我们可以使用另一个元学习者来优化元学习者等等，但最终我们将需要一个手工定义的优化器，如SGD或ADAM（不可能）。

关于执行情况，我们现在也可以讨论几点重要的意见：

* 二阶导数：通过模型的梯度反向传播元损耗涉及到计算导数的导数，即二阶导数（当上一个动画的元反向过程中绿色¨穿过绿色¨时）。我们可以在像Tensorflow或Py火炬这样的现代框架中计算，但在实践中，我们经常丢弃第二导数，并且只通过模型权重（Meta向后传递的黄色）来反向传播，以降低复杂性。
* 坐标共享：最近的一个深度学习模型可以有非常多的参数（在NLP中很容易达到3-2亿）。使用当前的GPU内存，不可能有这么多参数作为优化器的单独输入。我们通常所做的是所谓的坐标共享，这意味着我们为模型的单个参数设计优化器，并为所有参数复制它（即沿与模型参数相关联的输入维度共享它的权重）。这样，元学习者的参数个数不是模型参数个数的函数。当元学习者是一个具有类似RNN的记忆的网络时，我们仍然可以允许每个模型参数有一个单独的隐藏状态，以保持每个模型参数演化的单独记忆。

### PyTorch中的元学习

让我们尝试一些代码来看看这在实践中是什么样子的。

因此，我们有一个带有一组权重的模型，我们希望对其进行训练，并将其用于两个任务：

* 在元转发过程中：我们使用我们的模型计算梯度（从损失中），这些梯度作为优化器的输入来更新模型参数，并且
* 在元后向传递过程中：我们使用我们的模型作为反向传播优化器参数梯度（从元损失计算）的路径。

在PyTorch中，最简单的方法是有两个表示模型的重复模块，每个任务一个。让我们调用forward model负责存储meta forward pass期间使用的模型梯度的模块和backward model负责将参数保持为meta backward pass期间反向传播优化器梯度的连续路径的模块。

这两个模块将共享它们的张量，以避免重复记忆（张量是记忆中真正的肉），但将保留单独的变量，以清晰地分离模型的梯度和元学习者使用的梯度。

#### PyTorch中的一个简单的元学习课程

在PyTorch中共享张量是相当直接的：我们只需要更新变量类中的指针来指向相同的张量。当我们的模型已经是一个内存优化模型时，一个困难就来了，就像一个具有共享张量（输入和输出嵌入）的模型。然后在更新这两个模块的模型参数时，我们需要注意保持正确的指针。

一种方法是设置一个简单的助手来处理参数循环的任务，发送回更新参数指针（不仅仅是张量）所需的所有信息，并保持共享参数同步。这里有这样一个函数：

使用此函数，我们可以插入任何模型，并在元学习器中以干净的方式循环模型参数。

现在让我们起草一个简单的元学习者课程。我们的优化器是一个模块，它将在前向传递期间作为输入，前向模型（带有梯度）和后向模型将循环它们的参数，以允许元梯度反向传播的方式更新后向模型参数（通过更新参数指针而不仅仅是张量）。

我们现在可以像在第一部分中看到的那样训练这个优化器。以下是一个简单的要点，说明了我们一直在描述的元训练过程：

#### 避免记忆爆炸 - 隐藏状态记忆

有时我们想学习一种优化器，它可以在具有数千万个参数的非常大的模型上运行，同时我们想通过大量步骤展开元训练，以获得像我们在中所做的那样的高质量梯度。

在实践中，这意味着我们要在元前向传球过程中包括一个长时间的训练过程，包括许多时间步，并且我们必须记住用于元后向传球的每一步的参数（黄色■）和梯度（绿色■）数据。

我们怎么能在不破坏GPU记忆的情况下做到这一点呢？

一种方法是使用梯度检查点（也称为隐藏状态记忆）来交换一些内存进行计算。在我们的例子中，梯度检查点是将元前向和元后向通道分割成我们依次计算的段。

OpenAI的博客文章很好地介绍了梯度检查点。如果你对此感兴趣，你应该去看看：

[**Fitting larger networks into memory.** TLDR; we release the python/Tensorflow package openai/gradient-checkpointing, that lets you fit 10x larger neural nets…medium.com](https://medium.com/@yaroslavvb/fitting-larger-networks-into-memory-583e3c758ff9)

这篇文章已经很长了，所以我不会包括梯度检查点代码的全部要点。我更愿意把你带到TSHadley的漂亮的Pythorch，并在Pythorch中加入梯度检查点。

#### 元学习的其他方法

在元学习的研究中，还有另外两个趋势我没有时间去讨论，但它们也很有希望。我只给你几点建议，这样你就可以自己去检查一下，现在你知道了总的想法：

* 递归网络：我们建立在神经网络的标准训练过程之上。另一种方法是将任务的连续性视为一系列连续的输入，并构建一个递归模型，该模型可以接收并为新任务构建此序列的表示。在这种情况下，我们通常有一个单一的训练过程和一个具有记忆或注意力的循环网络。这种方法也给出了很好的结果，特别是当嵌入为任务进行了充分的设计时。一个很好的例子是最近的。
* 强化学习：优化器在元前向过程中进行的计算与递归网络的计算非常相似：在输入序列上重复应用相同的参数（学习期间模型的权重和梯度的连续性）。在实践中，这意味着我们遇到了一个经常性的问题：当模型犯错误时，它们很难返回到安全路径，因为它们没有经过训练从训练错误中恢复过来，而且模型很难推广到比元训练中使用的序列更长的序列。要解决这些问题，可以转向模型学习与当前培训状态相关联的操作策略的位置。

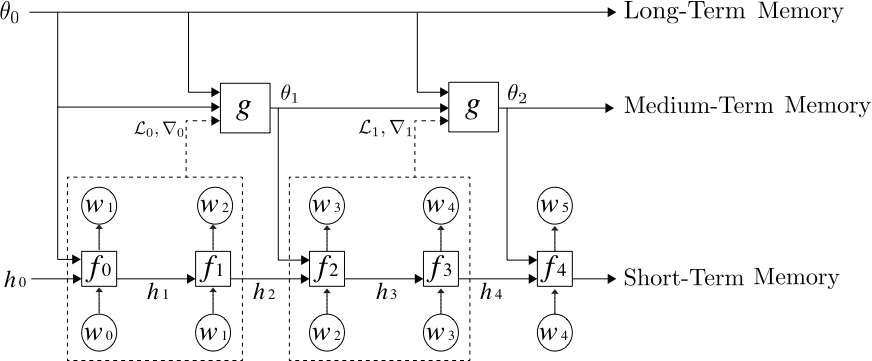
### 自然语言处理中的元学习

在元学习和自然语言处理（NLP）中使用的神经网络模型（如递归神经网络（RNN））之间有一个有趣的平行关系，我们在前一段中刚刚开始提到：

优化神经网络模型的元学习者的行为类似于递归神经网络。

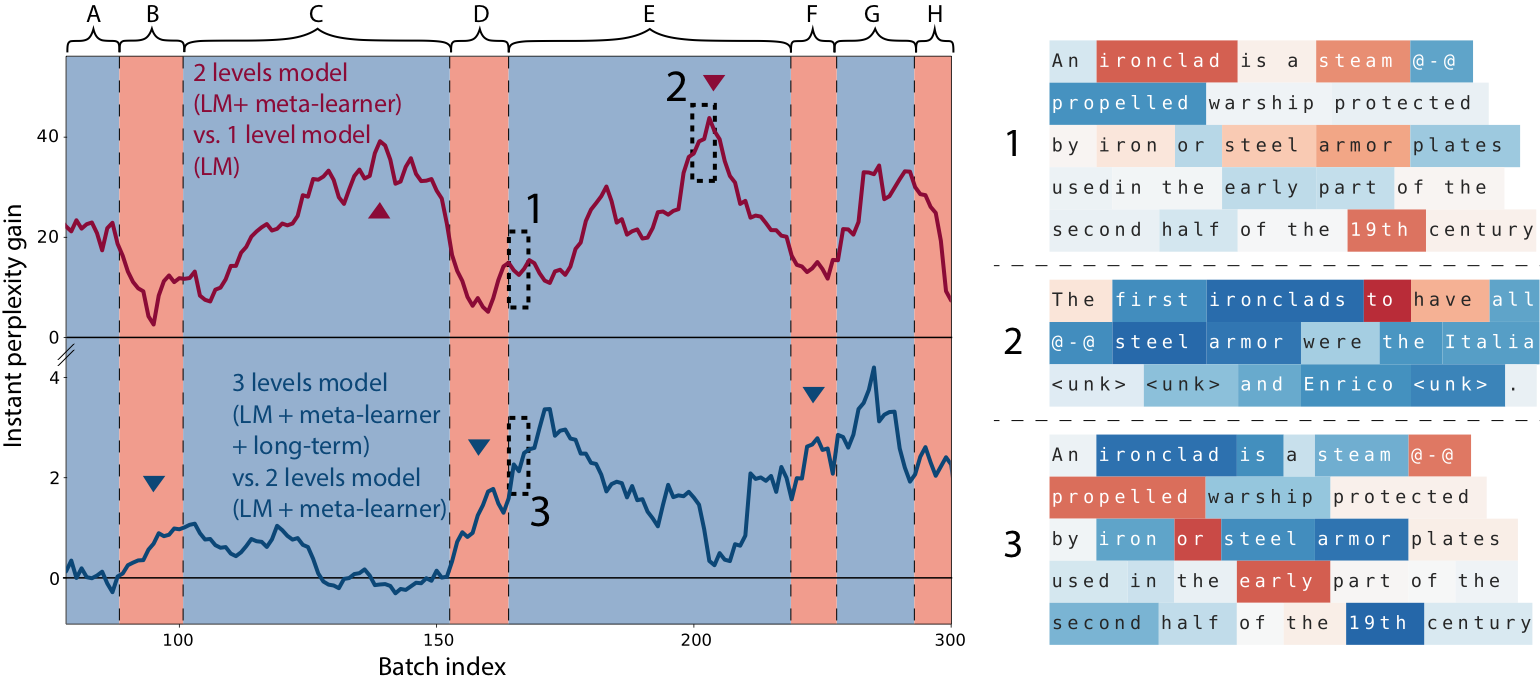
与RNN一样，元学习者在训练期间将模型的一系列参数和梯度作为输入序列摄取，并从该输入序列计算序列输出（更新的模型参数序列）。

我们在中发展了这一类比，并研究了元学习者如何在神经网络语言模型中实现中期记忆：元学习者学习将中期记忆编码为标准RNN的权重，就像LSTM一样（除了短期记忆在LSTM的隐藏状态中的常规编码方式之外）。



我们的元学习语言模型有三个层次的记忆，从下到上：一个标准的LSTM，一个元学习者更新LSTM的权重以存储中期记忆和长期静态记忆。

我们发现元学习语言模型可以被训练成对最近输入的记忆进行编码，比如维基百科文章的开头，这将有助于预测文章的结尾。



好吧，我想你现在已经准备好看更多关于这个故事的细节了。

我对元学习的介绍到此结束。恭喜你完成了这项漫长的任务！

希望你喜欢！

如果你想要更多这样的内容，别忘了给我们鼓掌！

1. [^]（#afeb）因此，元学习可以看作是“转移学习”的一种推广，它与任务模型的微调技术以及超参数优化技术有关。去年12月，NIPS 2017举办了一场有趣的活动。
2. [^]（#dc5a）当然，在真正的培训中，我们将使用一小批示例。
3. [^]（#e0bb）更准确地说：“大多数”这些操作是可微的。
4. [^]（#d640）介绍相关文献的好博客文章有：Chelsea Finn和Ke Li的文章。
5. [^]（#930c）学习模型初始参数的好例子是UC Berkeley及其OpenAI。DeepMind的论文是学习优化器参数的一个很好的例子。萨钦·拉维和雨果·拉罗谢尔的作品将两者结合在一起。可以在中找到一个很好的最近的概述。
6. [^]（#d094）类似于我们在展开的递归网络中通过时间反向传播的方式。
7. [^]（#725d）最初在DeepMind的论文中描述。
8. [^]（#4e23）如前所述，我们在元学习者中使用坐标共享。实际上，这意味着我们只需在模型参数上迭代，并在每个参数上应用我们的优化器广播（例如，不需要像在L-BFGS中那样展平和收集参数）。
9. [^](#d029) There is a surprising under-statement of how important back-propagating over very long sequence can be to get good results. The recent paper [An Analysis of Neural Language Modeling at Multiple Scales](https://arxiv.org/abs/1803.08240) from Salesforce research is a good pointer in that direction.
10. [^](#6c6f) Gradient checkpointing is described for example in [Memory-Efficient Backpropagation Through Time](https://arxiv.org/abs/1606.03401) and the nice [blog post](https://medium.com/@yaroslavvb/fitting-larger-networks-into-memory-583e3c758ff9) of Yaroslav Bulatov.