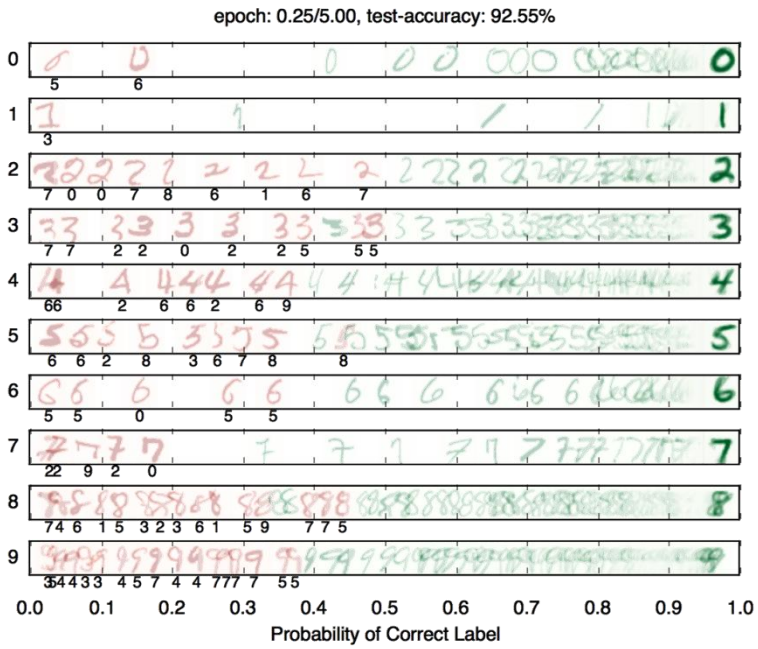
萍乡学院 PINGXIANG UNIVERSITY

信息与计算机工程学院

人工智能导论

课程项目 4：机器学习



1、介绍

该项目将介绍机器学习；你将构建一个神经网络来对数字进行分类，等等！

此项目的代码包含以下文件，这些文件以 zip 文档形式提供。

你需要编辑的文件	
models.py	适用于各种应用的感知器和神经网络模型。
你需要查看的文件	
nn.py	神经网络微型库。

其它的文件你可以忽视。

需要编辑和提交的文件：你需要完成 models.py，将你修改的文件和自动批改程序（submission_autograder）产生的结果，以及项目报告一同提交。请不要修改或提交其它文件。

项目评估：你的代码会通过自动批改来判断其正确性，因此请不要修改代码中其它任何函数或者类，否则你会让自动批改程序无法正常运行。然而，你的解题思路和方法是你最终成绩的决定因素。必要的话，我们会查看你的代码来保证你得到应得的成绩。

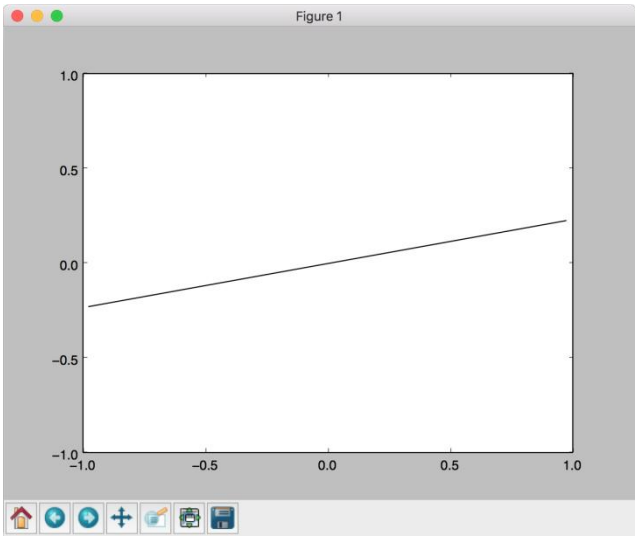
学术造假：我们会查看你的代码和其它学生提交的代码是否雷同。禁止抄袭了他人代码，或只做简单修改后提交，一旦发现，成绩立马作废，而且会影响到你能否通过此课程。

寻求帮助：当你感到自己遇到了困难，请向你的同学和老师寻求帮助。小组合作、答疑时间、课堂讨论，这些都是用来帮助你的，请积极利用这些资源。设计这些项目的目的是让你更有效地理解和掌握课堂知识，学会如何将理论知识应用于实践，解决实际问题，而不是为考核而考核，或者有意刁难你，所以请尽你所能，完成它们。遇到困难时，向课代表和老师提问。

安装

如果运行以下内容，并且你看到弹出以下窗口，其中线段旋转成圆，则可以跳过此部分。你应该使用 conda 环境，因为 conda 附带了我们需要的库。

```
python autograder.py --check-dependencies
```



对于此项目，你需要安装以下两个库：

- numpy：一个多维数组运算的库
- matplotlib： 一个画图的库

你可以在你的 conda 环境下安装以上两个库，然后用以上的方法检验通过。

提供的代码（第一部分）

对于此项目，我们提供了神经网络微型库 （nn.py） 和数据集集合 （backend.py）。

nn.py 中的库定义了节点对象的集合。每个节点代表一个实数或实数矩阵。对节点对象的操作经过优化，比使用 Python 的内置类型（如列表）更快。

以下是提供的一些节点类型：

- nn.Constant 表示浮点数的矩阵（二维数组）。它通常用于表示输入要素或目标输出/标注。这种类型的实例将由 API 中的其他函数提供给你；你无需直接构造它们。
- nn.Parameter 表示感知器或神经网络的可训练参数。
- nn.DotProduct 计算其输入之间的点积。

提供的其他功能：

- nn.as_scalar 可以从节点中提取 Python 浮点数。训练感知器或神经网络时，将向你传递一个数据集对象。你可以通过调用 dataset.iterate_once（batch_size） 来检索成批的训练示例：

```
for x, y in dataset.iterate_once(batch_size)
    ...
```

例如，让我们从感知器训练数据中提取一批大小为 1 的批次（即单个训练示例）：

```
>>> batch_size = 1
>>> for x, y in dataset.iterate_once(batch_size):
...     print(x)
...     print(y)
...     break
…
<Constant shape=1x3 at 0x11a8856a0>
<Constant shape=1x1 at 0x11a89efd0>
```

输入特征 x 和正确的标签 y 以 nn.Constant 节点形式提供。x 的尺寸为 batch_size x num_features，y 的尺寸为 batch_size x num_outputs。因此，每行 x 是一个点/样本，而一列是某些样本的相同特征。下面是一个计算 x 与自身点积的示例，首先作为节点，然后作为 Python 数字。

```
>>> nn.DotProduct(x, x)
<DotProduct shape=1x1 at 0x11a89edd8>
>>> nn.as_scalar(nn.DotProduct(x, x))
1.9756581717465536
```

最后，这里有一些矩阵乘法的公式（你可以手动做一些例子来验证这一点）。假设 A 是一个 mxn 的矩阵，B 是 nxp；矩阵乘法的工作原理如下：

$$\mathbf{AB} = \begin{bmatrix} \vec{A}_0^T \\ \vec{A}_1^T \\ \cdots \\ \vec{A}_{m-1}^T \end{bmatrix} \mathbf{B} = \begin{bmatrix} \vec{A}_0^T\mathbf{B} \\ \vec{A}_1^T\mathbf{B} \\ \cdots \\ \vec{A}_{m-1}^T\mathbf{B} \end{bmatrix} \qquad \mathbf{AB} = \mathbf{A}\begin{bmatrix} \vec{B}_0 & \vec{B}_1 & \cdots & \vec{B}_{p-1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}\vec{B}_0 & \mathbf{A}\vec{B}_1 & \cdots & \mathbf{A}\vec{B}_{p-1} \end{bmatrix}$$

作为健全性检查，维度是我们期望的，而内部维度 n 对于任何剩余的矩阵乘法都保留。

这对于查看我们将批处理矩阵 X 乘以权重矩阵 W 的结果十分有用，我们只是通过第一个公式一次一次地将每个样品乘以整个权重矩阵。在每个样本乘以权重时，我们只是通过第二个公式获得样本的不同线性组合，以得到每个所需地列。注意，只要维度匹配，A 可以是行向量，B 可以是列向量。

构建神经网络

在项目的整个应用程序部分，你将使用 nn.py 中提供的框架来创建神经网络来解决各种机器学习问题。一个简单的神经网络包含线性层，执行某些线性操作（就像感知器一样）。线性层之间由非线性分隔，这允许网络近似普适的功能。我们将使用 ReLU 运算来实现非线性，定义为 relu(x) = max(x, 0)。例如，一个简单的一个隐藏层/两个线性层神经网络，用于映射输入行向量 x 到输出向量 f(x)， 将由以下函数给出：

$$\mathbf{f(x)} = \text{relu}(\mathbf{x} \cdot \mathbf{W_1} + \mathbf{b_1}) \cdot \mathbf{W_2} + \mathbf{b_2}$$

我们有参数矩阵的地方 W1、W2 和参数向量 b1、b2 在梯度下降的过程中学习。W1 将是一个 i×h 的矩阵，其中 i 是输入向量 x 的维度，和 h 是隐藏层大小。b1 将是一个大小为 h 的向量。我们可以自由地为隐藏尺寸选择任何我们想要的值（我们只需要确保其他矩阵和向量的维度一致，以便我们可以执行操作）。使用更大的隐藏大小通常会使网络更强大（能够容纳更多的训练数据），但会使网络更难训练（因为它为我们需要学习的所有矩阵和向量添加了更多参数），或者可能导致训练数据过度拟合。

我们还可以通过添加更多层来创建更深层次的网络，例如三线性层网络：

$$\mathbf{\hat{y}} = \mathbf{f(x)} = \text{relu}(\text{relu}(\mathbf{x} \cdot \mathbf{W_1} + \mathbf{b_1}) \cdot \mathbf{W_2} + \mathbf{b_2}) \cdot \mathbf{W_3} + \mathbf{b_3}$$

或者，我们可以分解上述方程，明确指出 2 个隐藏层：

$$\mathbf{h_1} = \mathbf{f_1(x)} = \text{relu}(\mathbf{x} \cdot \mathbf{W_1} + \mathbf{b_1})$$

$$\mathbf{h_2} = \mathbf{f_2(h_1)} = \text{relu}(\mathbf{h_1} \cdot \mathbf{W_2} + \mathbf{b_2})$$

$$\mathbf{\hat{y}} = \mathbf{f_3(h_2)} = \mathbf{h_2} \cdot \mathbf{W_3} + \mathbf{b_3}$$

请注意，我们没有 relu 在最后，因为我们希望能够输出负数，并且我们之所以采用 relu 的原因首先是为了非线性变换，让输出是一些非线性中间体的仿射线性变换是比较适合的。

批量处理

为了提高效率，你将需要一次处理整批数据，而不是一次处理一个示例。这意味着不是单个输入尺寸为 i 的行向量 x，你会看到一批 b 输入，表示为 b×i 的矩阵 X。我们提供了一个线性回归的例子，以演示如何在批处理设置中实现线性层。

随机性

神经网络的参数将被随机初始化，某些任务中的数据将以随机顺序呈现。由于这种随机性，即使使用强大的架构，你仍然可能偶尔会失败某些任务——这就是局部最优的问题！不过，这种

情况应该很少发生——如果在测试代码时连续两次让自动评分器因一个问题而失败，你应该探索其他架构。

设计架构

设计神经网络可能需要一些反复试验。以下是一些可以帮助你的提示：

- 要有系统性。记录你尝试过的每个架构，超参数（层大小，学习率等）是什么，以及最终的性能是多少。随着你尝试更多事情，你可以开始看到有关哪些参数重要的模式。如果在代码中发现错误，请务必划掉由于该错误而无效的过去结果。

- 从浅层网络开始（只有一个隐藏层，即一个非线性）。更深层次的网络具有指数级更多的超参数组合，即使犯错一个错误也会破坏你的性能。使用小型网络找到良好的学习率和层大小;之后，你可以考虑添加更多类似大小的图层。

- 如果你的学习率是错误的，那么你的其他超参数选择都无关紧要。你可以从研究论文中获取最先进的模型，并更改学习率，使其性能不会优于随机。学习率太低会导致模型学习太慢，学习率太高可能会导致损失发散到无穷大。首先尝试不同的学习率，同时观察损失如何随着时间的推移而减少。

- 较小的批次需要较低的学习率。尝试不同的批大小时，请注意，最佳学习率可能会因批大小而异。

- 避免使网络太宽（隐藏层大小太大） 如果你继续使网络更宽，精度将逐渐下降，计算时间将随层大小的二次方增加——你可能会因为太慢而放弃，早在精度下降太多之前。项目所有部分的全自动评分器需要 2-12 分钟才能与员工解决方案一起运行；如果你的代码花费的时间要长得多，你应该检查它的效率。

- 如果你的模型返回 Infinity 或 NaN，则你的学习率对于你当前的体系结构来说可能太高了。

- 超参数的建议值：

  - 隐藏层尺寸：在 10 到 400 之间。

  - 批量尺寸：介于 1 和数据集大小之间。对于 Q2 和 Q3，我们要求数据集的总大小可被批大小整除。

  - 学习率：介于 0.001 和 1.0 之间。

  - 隐藏层数：介于 1 和 3 之间。

提供的代码（第二部分）

以下是 nn.py 中可用节点的完整列表。你将在作业的其余部分中使用这些内容：

- nn.Constant 表示浮点数的矩阵（二维数组）。它通常用于表示输入特征或目标输出/标注。这种类型的实例将由 API 中的其他函数提供给你；你无需直接构造它们。

- nn.Parameter 表示感知器或神经网络的可训练参数。所有参数必须是二维的。

  - 用法：nn.Parameter（n， m） 构造一个形状为 n x m 的参数。

- nn.Add 按元素添加矩阵。

- 用法：nn.Add（x， y） 接受形状为 batch_size x num_features 的两个节点，并构造一个也具有形状 batch_size x num_features 的节点。

- nn.AddBias 为每个特征向量添加一个偏置向量。注意：它会自动广播偏差，以将相同的向量添加到每行特征中。
  - 用法：nn.AddBias（features，bias）接受形状为 batch_size x num_features 的特征和形状为 1 x num_features 的偏差，并构造一个具有形状 batch_size x num_features 的节点。

- nn.Linear 对输入作线性变换（矩阵乘法）。
  - 用法：nn.Linear（features, weights）接受 batch_size x num_input_features 形状的特征和按 num_input_features x num_output_features 形状的权重，并构造一个形状为 batch_size x num_output_features 的节点。

- nn.ReLU 应用逐单元整流线性单元非线性 relu(x) = max(x, 0)。这种非线性将输入中的所有负条目替换为零。
  - 用法：nn.ReLU（features），返回与输入形状相同的节点。

- nn.SquareLoss 计算一个批处理平方损失，用于回归问题。
  - 用法：nn.SquareLoss（a， b），其中 a 和 b 都具有 batch_size x num_outputs 的形状。

- nn.SoftmaxLoss 计算批处理的 softmax 损失，用于分类问题。
  - 用法：nn.SoftmaxLoss（logits，labels），其中 logits 和标签都具有 batch_size x num_classes 的形状。术语“logits”是指由模型生成的分数，其中每个条目都可以是任意实数。但是，标签必须是非负数，并且每行的总和为 1。确保不要交换参数的顺序！

- 除了感知器以外的任何模型都不要使用 nn.DotProduct。

nn.py 中提供了以下方法：

- nn.gradients 根据提供的参数计算损失的梯度。
  - 用法：nn.gradients（loss， [parameter_1， parameter_2， ...， parameter_n]） 将返回一个列表 [gradient_1， gradient_2， ...， gradient_n]，其中每个元素都是一个 nn。包含损失相对于参数的梯度的常量

- nn.as_scalar 可以从损失节点中提取 Python 浮点数。这对于确定何时停止训练非常有用。
  - 用法：nn.as_scalar（node），其中节点是损失节点或具有形状 （1，1）。

提供的数据集里还有两种附加方法：

- dataset.iterate_forever（batch_size） 产生无限的批量示例序列。

- dataset.get_validation_accuracy（） 返回验证集上模型的准确性。这对于确定何时停止训练非常有用。

范例：线性回归

作为神经网络框架如何工作的一个例子，让我们将一条线拟合到一组数据点。我们将开始使用函数构建的四个训练数据点$y = 7x_0 + 8x_1 + 3$。在批处理形式中，我们的数据是：

$$\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 3 \\ 11 \\ 10 \\ 18 \end{bmatrix}$$

假设数据以 nn.Constant 节点的形式提供给我们。

```
>>> x
<Constant shape=4x2 at 0x10a30fe80>

>>> y
<Constant shape=4x1 at 0x10a30fef0>
```

我们来构架和训练这么一个模型：$f(x) = x_0 \cdot m_0 + x_1 \cdot m_1 + b$。如果做得正确，我们应该能够学到这一点$m_0 = 7$，$m_1 = 8$，$b = 3$。

首先，我们创建可训练的参数。在矩阵形式中，这些是：

$$\mathbf{M} = \begin{bmatrix} m_0 \\ m_1 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} b \end{bmatrix}$$

对应于如下的代码：

```
m = nn.Parameter(2, 1)

b = nn.Parameter(1, 1)

>>> m
<Parameter shape=2x1 at 0x112b8b208>

>>> b
<Parameter shape=1x1 at 0x112b8beb8>
```

接下来，我们计算模型对 y 的预测：

```
xm = nn.Linear(x, m)

predicted_y = nn.AddBias(xm, b)
```

我们的目标是预测的 y 值与提供的数据匹配。在线性回归中，我们通过最小化平方损失来做到这一点：

$$\mathcal{L} = \frac{1}{2N} \sum_{(\mathbf{x},y)} (y - f(\mathbf{x}))^2$$

我们构造一个损失节点：

```
loss = nn.SquareLoss(predicted_y, y)
```

在我们的框架中，我们提供了一种方法，该方法将返回损失相对于参数的梯度：

```
grad_wrt_m, grad_wrt_b = nn.gradients(loss, [m, b])

>>> xm
<Linear shape=4x1 at 0x11a869588>
```

```
>>> predicted_y
<AddBias shape=4x1 at 0x11c23aa90>

>>> loss
<SquareLoss shape=() at 0x11c23a240>

>>> grad_wrt_m
<Constant shape=2x1 at 0x11a8cb160>

>>> grad_wrt_b
<Constant shape=1x1 at 0x11a8cb588>
```

然后，我们可以使用 update 方法来更新我们的参数。这是 m 的更新，假设我们已经根据我们选择的合适学习率初始化了一个乘数变量：

```
m.update(grad_wrt_m, multiplier)
```

如果我们还包含 b 的更新并添加一个循环来重复执行梯度更新，我们将拥有线性回归的完整训练过程。

3、项目内容

问题 1（6 分）：感知器

在开始这部分之前，请确保你已安装 numpy 和 matplotlib！

在这一部分中，你将实现二进制感知器。你的任务是在 models.py 中完成 PerceptronModel。

对于感知器，输出标签将为 1 或 −1，这意味着数据集中的数据点 （x， y） 的 y 将是 nn.Constant，包含 1 或 −1 作为其条目。

我们已经将感知器权重 self.w 初始化为维度为 1 的参数节点。在需要时，提供的代码将在 x 中包含偏差特征，因此你不需要单独为偏差设置参数。

你的任务是：

- 实现 run（self， x） 方法。这应该计算存储的权重向量和给定输入的点积，返回 nn.DotProduct 对象。
- 实现 get_prediction（self， x），如果点积为非负数，则应返回 1，否则应返回 −1。你应该使用 nn.as_scalar 将标量节点转换为 Python 浮点数。
- 编写 train（self）方法。这应该反复循环数据集，并对错误分类的示例进行更新。使用 nn.Parameter 类的 update 方法来更新权重的参数类。当完成对数据集的整个运算而没有犯任何错误时，已达到 100% 的训练准确率，训练可以终止。
- 在这个项目中，更改参数值的唯一方法是调用 parameter.update（direction， multiplier），它将执行对权重的更新：

$$\text{weights} \leftarrow \text{weights} + \text{direction} \cdot \text{multiplier}$$

参数 direction 是与参数形状相同的 Node，参数 multiplier 是 Python 标量。此外，使用 iterate_once 遍历数据集；有关用法，请参阅前面的提供代码（第一部分）部分。

若要测试实现，请运行自动评分器：

```
python autograder.py -q q1
```

注意：自动评分器最多需要 20 秒左右。如果自动评分器需要很长时间才能运行完成，则你的代码可能存在错误。

问题 2（6 分）：非线性回归

对于这个问题，你将训练神经网络在[−2π,2π]的范围内近似 $\sin(x)$。

你需要在 models.py 中完成回归模型类的实现。对于这个问题，一个相对简单的架构就足够了（有关架构技巧，请参阅神经网络技巧部分）。使用 nn.SquareLoss 作为你的损失。

你的任务是：

- 通过任何所需的初始化实现 RegressionModel.__init__。
- 实现 RegressionModel.run 以返回表示模型预测的 batch_size x 1 节点。
- 实现 RegressionModel.get_loss 以返回给定输入和目标输出的损失。
- 实现 RegressionModel.train，它应该使用基于梯度的更新来训练你的模型。

此任务只有一个数据集拆分（即，只有训练数据，没有验证数据或测试集）。如果你的实现在数据集中的所有示例中平均损失 0.02 或更好，则你的实现将获得满分。你可以使用训练损失来确定何时停止训练（使用 nn.as_scalar 将损失节点转换为 Python 数字）。请注意，训练模型应该需要几分钟时间。

建议的网络架构：通常，你需要使用试错法来查找工作超参数。下面是一组对我们有用的超参数，但请随意试验和使用自己的超参数。

- 隐藏层大小 512
- 批量大小 200
- 学习率 0.05
- 一个隐藏层（总共 2 个线性层）

评分：若要测试和调试代码，请运行

```
python autograder.py -q q2
```

问题 3（6 分）：数字分类

对于此问题，你将训练网络以对 MNIST 数据集中的手写数字进行分类。

每个数字的大小为 28 x 28 像素，其值存储在浮点数的 784 维向量中。我们提供的每个输出都是一个 10 维向量，它在所有位置都有零，除了对应于正确数字类的位置上的一个。

在 models.py 中完成 DigitClassificationModel 类的实现。来自 DigitClassificationModel.run（） 的返回值应该是包含分数的 10 batch_size 节点，其中分数越高表示数字属于特定类 （0-9） 的概率越高。你应该使用 nn.SoftmaxLoss 作为你的损失。不要将 ReLU 激活放在网络的最后一个线性层。

对于这个问题和问题 4，除了训练数据，还有验证和测试集。用 dataset.get_validation_accuracy() 计算模型的验证准确性，这在决定是否停止训练时非常有用。测试集将由自动评分器使用。

要获得此问题的分数，你的模型应在测试集上达到至少 97% 的准确率。作为参考，我们的员工实施在训练大约 5 个 epoch 后始终在验证数据上实现 98% 的准确率。请注意，测试会根据测试准确性对你进行评分，而你只能访问验证准确性 - 因此，如果你的验证准确性达到 97% 阈值，如果你的测试准确性未达到阈值，你仍可能无法通过测试。因此，对验证准确性设置稍高的停止阈值（例如 97.5% 或 98%）可能会有所帮助。

建议的网络架构（2022/11/28 添加）：通常，你需要使用试错法来查找工作超参数。下面是一组对我们有用的超参数，但请随意试验和使用自己的超参数。

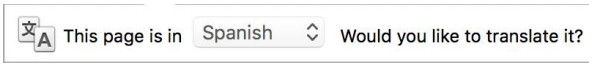
- 隐藏层大小 200
- 批量大小 100
- 学习率 0.5
- 一个隐藏层（总共 2 个线性层）

评分：若要测试和调试代码，请运行

```
python autograder.py -q q3
```

问题 4（7 分）：语言识别

语言识别是确定给定一段文本的任务，文本是用什么语言写的。例如，你的浏览器可能能够检测到你是否访问过外语页面，并愿意为你翻译该页面。以下是 Chrome 的示例（它使用神经网络来实现此功能）：



在这个项目中，我们将构建一个较小的神经网络模型，一次识别一个单词的语言。我们的数据集由五种语言的单词组成，如下表所示：

单词	语言
discussed	English
eternidad	Spanish
itseänne	Finnish
paleis	Dutch
mieszkać	Polish

不同的单词由不同数量的字母组成，因此我们的模型需要具有可以处理可变长度输入的架构。而不是单个输入 x（与前面的问题一样），我们将为单词中的每个字符提供一个单独的输入：$x_0, x_1, \ldots, x_{L-1}$，其中 L 是单词的长度。我们开始将应用一个网络$f_{initial}$，就像前一个问题中的网络。它接受其输入$x_0$，计算一个维度为 d 的输出向量h_1：

$$h_1 = f_{initial}(x_0)$$

接下来，我们将上一步的输出与单词中的下一个字母组合在一起，生成单词前两个字母的向量摘要。为此，我们将应用一个子网，该子网接受字母并输出隐藏状态，但现在还取决于之前的隐藏状态h_1。我们将此子网表示为f。

$$h_2 = f(h_1, x_1)$$

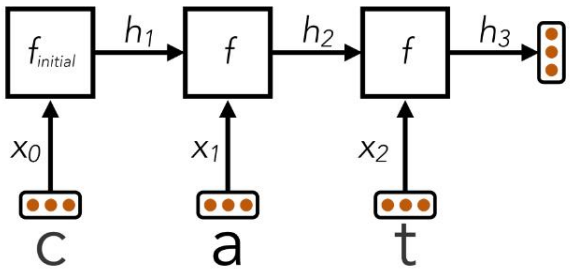
此模式继续适用于输入单词中的所有字母，其中每一步的隐藏状态汇总了网络迄今为止处理的所有字母：

$$h_3 = f(h_2, x_2)$$

$$\vdots$$

在这些计算中，函数$f(\cdot,\cdot)$是同一个神经网络，使用相同的可训练参数；$f_{initial}$还将共享一些与$f(\cdot,\cdot)$同样的参数。这样，处理不同长度的单词时使用的参数都是共享的。你可以在提供的输入 xs 上使用一个 for 循环来实现这一点，其中循环的每次迭代都会计算$f_{initial}$或者f。

上述技术称为递归神经网络（RNN）。RNN 的示意图如下所示：



在这里，RNN 用于将单词“cat”编码为固定大小的向量h_3。

在 RNN 处理完输入的完整长度后，它将任意长度的输入字编码为固定大小的向量h_L，其中 L 是单词的长度。输入单词的矢量摘要现在可以通过其他输出转换层馈送，以生成单词语言标识的分类分数。

批量处理

尽管上述等式是单个单词，但实际上你必须使用批量单词以提高效率。为简单起见，项目中的代码确保单个批处理中的所有单词具有相同的长度。以批处理形式，隐藏状态h_i 替换为维度为 batch_size x d 的矩阵H_i。

设计提示

循环功能的设计$f(\cdot,\cdot)$是此任务的主要挑战。以下是一些提示：

- 从架构开始，你选择的$f_{initial}(x)$应该与前面的问题类似，只要它至少有一个非线性。
- 给定$f_{initial}(x)$，应使用以下方法构造$f(\cdot,\cdot)$。$f_{initial}$的 第一转化层将从向量x_0乘以W_x生成$z_0 = x_0 \cdot W_x$开始。对于后续字母，应将此计算替换为$z_i = x_i \cdot W_x + h_i \cdot W_{hidden}$使用一个 n.Add 操作。换句话说，你应该用 z = nn.Add(nn.Linear(x, W), nn.Linear(h, W_hidden))取代 z0 = nn.Linear(x, W)。
- 如果操作正确，则生成的函数$f(x_i, h_i) = g(z_i) = g(z_{x_i,h_i})$ 在 x 和 h 两者中都是非线性的

- 隐藏层大小 d 应足够大。
- 从一个浅层网络 f 开始，并在使网络更深入之前找出隐藏大小和学习率的良好值。如果你立即从深度网络开始，你将拥有指数级的超参数组合，并且任何单个超参数错误都可能导致你的性能受到严重影响。

你的任务

完成 LanguageIDModel 类的实现。

要获得有关此问题的满分，你的体系结构应该能够在测试集上达到至少 81% 的准确率。

评分：若要测试和调试代码，请运行

```
python autograder.py -q q4
```

声明：此数据集是使用自动文本处理生成的，它可能包含错误。但是，尽管存在数据限制，我们的参考实现仍然可以正确分类超过 89% 的验证集。我们的参考实现需要 10-20 个 epoch 来训练。

4、项目报告

简要清晰地描述完成项目时遇到的困难，采用的解决方法，提出改进意见，和总结每个小组成员的贡献。

5、提交

在提交你的解答之前，你需要通过执行 submission_autograder.pyc 来产生几个文件。在运行这个程序之前，你必须确认所有与 autograde 有关的文件都处在原始状态，没有做过任何的改动。假如你编辑过任何 autograde 的文件，请重新下载一份项目代码，仅仅替换你作解答的文件，否则运行 submission_autograder.pyc 将无法通过。

此外，submission_autograder.pyc 要在 Python 3.6（准确的说是 3.6.13，你可以用 Anaconda 来安装正确的 Python 版本）下执行，否则会报错。

最后，submission_autograder.pyc 需要用 rsa 库来给你的成绩加密，假如你没有的话，请用下面的命令安装 rsa 库。

```
conda install -c conda-forge rsa
```

或者用下面的命令，假如你没有 conda。

```
pip install rsa
```

进到你的 reinforcement 文件夹里，执行以下命令：

```
python submission_autograder.pyc
```

成功执行后，该命令会输出你的各个题目的得分和最后总分，并在 grade 文件夹里会生成一个 log 文件和一个 token 文件。确认该分数和你自己运行 autograder.py 得到的分数相同后，将整个 grade 文件夹和你修改过的文件（其它没有修改过的，例如 autograder.py，不需要）以及项目报告，放在一个以你的组号命名的文件夹里，生成一个 zip 文件，一并提交上来。