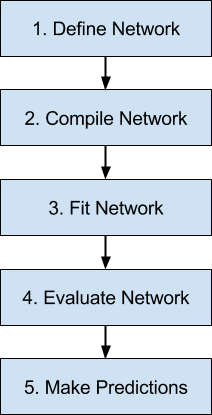
下面概括一下我们将要介绍的在 Keras 中构建神经网络模型的 5 个步骤。

1. 定义网络。
2. 编译网络。
3. 训练网络。
4. 评价网络。
5. 进行预测。



第一步：定义网络

首先要做的就是定义你的神经网络。

在 Keras 中，可以通过一系列的层来定义神经网络。这些层的容器就是 Sequential 类。（译注：序贯模型）

第一步要做的就是创建 Sequential 类的实例。然后你就可以按照层的连接顺序创建你所需要的网络层了。

例如，我们可以做如下两步：

model = Sequential()

model.add(Dense(2))

此外，我们也可以通过创建一个层的数组，并将其传给 Sequential 构造器来定义模型。

layers = [Dense(2)]

model = Sequential(layers)

网络的第一层必须要定义预期输入维数。指定这个参数的方式有许多种，取决于要建造的模型种类，不过在本文的多层感知机模型中我们将通过 input\_dim 属性来指定它。

例如，我们要定义一个小型的多层感知机模型，这个模型在可见层中具有 2 个输入，在隐藏层中有 5 个神经元，在输出层中有 1 个神经元。这个模型可以定义如下：

model = Sequential()

model.add(Dense(5, input\_dim=2))

model.add(Dense(1))

你可以将这个序贯模型看成一个管道，从一头喂入数据，从另一头得到预测。

这种将通常互相连接的层分开，并作为单独的层加入模型是 Keras 中一个非常有用的概念，这样可以清晰地表明各层在数据从输入到输出的转换过程中起到的职责。例如，可以将用于将各个神经元中信号求和、转换的激活函数单独提取出来，并将这个 Activation 对象同层一样加入 Sequential 模型中。

model = Sequential()

model.add(Dense(5, input\_dim=2))

model.add(Activation('relu'))

model.add(Dense(1))

model.add(Activation('sigmoid'))

输出层激活函数的选择尤为重要，它决定了预测值的格式。

例如，以下是一些常用的预测建模问题类型，以及它们可以在输出层使用的结构和标准的激活函数：

* **回归问题**：使用线性的激活函数 “linear”，并使用与与输出数量相匹配的神经元数量。
* **二分类问题**：使用逻辑激活函数 “sigmoid”，在输出层仅设一个神经元。
* **多分类问题**：使用 Softmax 激活函数 “softmax”；假如你使用的是 one-hot 编码的输出格式的话，那么每个输出对应一个神经元。

第二步：编译网络

当我们定义好网络之后，必须要对它进行编译。

编译是一个高效的步骤。它会将我们定义的层序列通过一系列高效的矩阵转换，根据 Keras 的配置转换成能在 GPU 或 CPU 上执行的格式。

你可以将编译过程看成是对你网络的预计算。

无论是要使用优化器方案进行训练，还是从保存的文件中加载一组预训练权重，只要是在定义模型之后都需要编译，因为编译步骤会将你的网络转换为适用于你的硬件的高效结构。此外，进行预测也是如此。

编译步骤需要专门针对你的网络的训练设定一些参数，设定训练网络使用的优化算法 以及用于评价网络通过优化算法最小化结果的损失函数尤为重要。

下面的例子对定义好的用于回归问题的模型进行编译时，指定了随机梯度下降（sgd）优化算法，以及均方差（mse）算是函数。

model.compile(optimizer='sgd', loss='mse')

预测建模问题的种类也会限制可以使用的损失函数类型。

例如，下面是几种不同的预测建模类型对应的标准损失函数：

* **回归问题**：均方差误差 “mse”。
* **二分类问题**：对数损失（也称为交叉熵）“\_binarycrossentropy”。
* **多分类问题**：多类对数损失 “\_categoricalcrossentropy”。

你可以查阅 [Keras 支持的损失函数](https://link.juejin.im/?target=http%3A%2F%2Fkeras.io%2Fobjectives%2F" \t "_blank)。

最常用的优化算法是随机梯度下降，不过 Keras 也支持[其它的一些优化算法](https://link.juejin.im/?target=http%3A%2F%2Fkeras.io%2Foptimizers%2F" \t "_blank)。

以下几种优化算法可能是最常用的优化算法，因为它们的性能一般都很好：

* **随机梯度下降** “sgd” 需要对学习率以及动量参数进行调参。
* **ADAM** “adam” 需要对学习率进行调参。
* **RMSprop** “rmsprop” 需要对学习率进行调参。

最后，你还可以指定在训练模型过程中除了损失函数值之外的特定指标。一般对于分类问题来说，最常收集的指标就是准确率。需要收集的指标由设定数组中的名称决定。

例如：

model.compile(optimizer='sgd', loss='mse', metrics=['accuracy'])

第三步：训练网络

在网络编译完成后，就能对它进行训练了。这个过程也可以看成是调整权重以拟合训练数据集。

训练网络需要制定训练数据，包括输入矩阵 X 以及相对应的输出 y。

在此步骤，将使用反向传播算法对网络进行训练，并使用在编译时制定的优化算法以及损失函数来进行优化。

反向传播算法需要指定训练的 Epoch（回合数、历元数）、对数据集的 exposure 数。

每个 epoch 都可以被划分成多组数据输入输出对，它们也称为 batch（批次大小）。batch 设定的数字将会定义在每个 epoch 中更新权重之前输入输出对的数量。这种做法也是一种优化效率的方式，可以确保不会同时加载过多的输入输出对到内存（显存）中。

以下是一个最简单的训练网络的例子：

model.compile(optimizer='sgd', loss='mse', metrics=['accuracy'])

在训练网络之后，会返回一个历史对象（History oject），其中包括了模型在训练中各项性能的摘要（包括每轮的损失函数值及在编译时制定收集的指标）。

第四步：评价网络

在网络训练完毕之后，就可以对其进行评价。

可以使用训练集的数据对网络进行评价，但这种做法得到的指标对于将网络进行预测并没有什么用。因为在训练时网络已经“看”到了这些数据。

因此我们可以使用之前没有“看”到的额外数据集来评估网络性能。这将提供网络在未来对没有见过的数据进行预测的性能时的估测。

评价模型将会评价所有测试集中的输入输出对的损失值，以及在模型编译时指定的其它指标（例如分类准确率）。本步骤将返回一组评价指标结果。

例如，一个在编译时使用准确率作为指标的模型可以在新数据集上进行评价，如下所示：

loss, accuracy = model.evaluate(X, y)

第五步：进行预测

最后，如果我们对训练后的模型的性能满意的话，就能用它来对新的数据做预测了。

这一步非常简单，直接在模型上调用 predict() 函数，传入一组新的输入即可。

例如：

predictions = model.predict(x)

预测值将以网络输出层定义的格式返回。

在回归问题中，这些由线性激活函数得到的预测值可能直接就符合问题需要的格式。

对于二分类问题，预测值可能是一组概率值，这些概率说明了数据分到第一类的可能性。可以通过四舍五入（K.round）将这些概率值转换成 0 与 1。

而对于多分类问题，得到的结果可能也是一组概率值（假设输出变量用的是 one-hot 编码方式），因此它还需要用 [argmax 函数](https://link.juejin.im/?target=http%3A%2F%2Fdocs.scipy.org%2Fdoc%2Fnumpy%2Freference%2Fgenerated%2Fnumpy.argmax.html)将这些概率数组转换为所需要的单一类输出。

End-to-End Worked Example

让我们用一个小例子将以上的所有内容结合起来。

我们将以 Pima Indians 糖尿病发病二分类问题为例。你可以在 [UCI 机器学习仓库](https://link.juejin.im/?target=https%3A%2F%2Farchive.ics.uci.edu%2Fml%2Fdatasets%2FPima%2BIndians%2BDiabetes)中下载此数据集。

该问题有 8 个输入变量，需要输出 0 或 1 的分类值。

我们将构建一个包含 8 个输入的可见层、12 个神经元的隐藏层、rectifier 激活函数、1 个神经元的输出层、sigmoid 激活函数的多层感知机神经网络。

我们将对网络进行 100 epoch 次训练，batch 大小设为 10，使用 ADAM 优化算法以及对数损失函数。

在训练之后，我们使用训练数据对模型进行评价，然后使用训练数据对模型进行单独的预测。这么做是为了方便起见，一般来说我们都会使用额外的测试数据集进行评价，用新的数据进行预测。

完整代码如下：

# Keras 多层感知机神经网络样例

from keras.models import Sequential

from keras.layers import Dense

import numpy

# 加载数据

dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")

X = dataset[:,0:8]

Y = dataset[:,8]

# 1. 定义网络

model = Sequential()

model.add(Dense(12, input\_dim=8, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

# 2. 编译网络

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# 3. 训练网络

history = model.fit(X, Y, epochs=100, batch\_size=10)

# 4. 评价网络

loss, accuracy = model.evaluate(X, Y)

print("\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy\*100))

# 5. 进行预测

probabilities = model.predict(X)

predictions = [float(round(x)) **for** x **in** probabilities]

accuracy = numpy.mean(predictions == Y)

print("Prediction Accuracy: %.2f%%" % (accuracy\*100))

运行样例，会得到以下输出：

...

768/768 [==============================] - 0s - loss: 0.5219 - acc: 0.7591

Epoch 99/100

768/768 [==============================] - 0s - loss: 0.5250 - acc: 0.7474

Epoch 100/100

768/768 [==============================] - 0s - loss: 0.5416 - acc: 0.7331

32/768 [>.............................] - ETA: 0s

Loss: 0.51, Accuracy: 74.87%

Prediction Accuracy: 74.87%

总结

在本文中，我们探索了使用 Keras 库进行深度学习时构建神经网络的 5 个步骤。

此外，你还学到了：

* 如何在 Keras 中定义、编译、训练以及评价一个深度神经网络。
* 如何选择、使用默认的模型解决回归、分类预测问题。
* 如何使用 Keras 开发并运行你的第一个多层感知机网络。

你对 Keras 的神经网络模型还有别的问题吗？或者你对本文还有什么建议吗？请在评论中留言，我会尽力回答。