循环神经网络实验报告

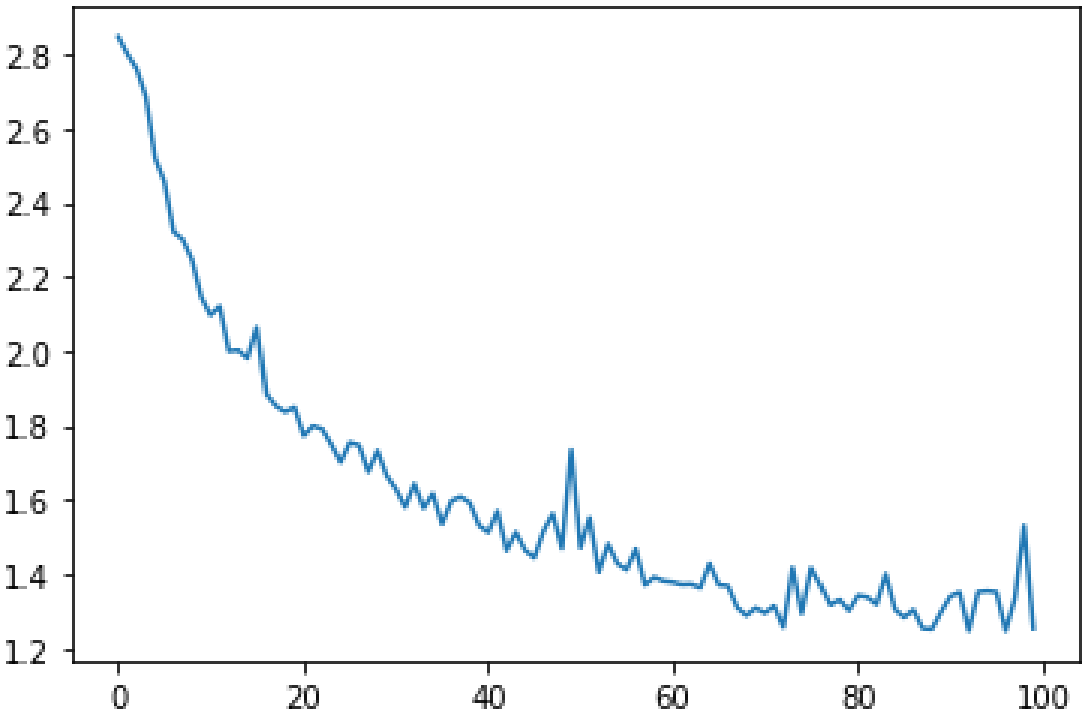
原始RNN网络实验

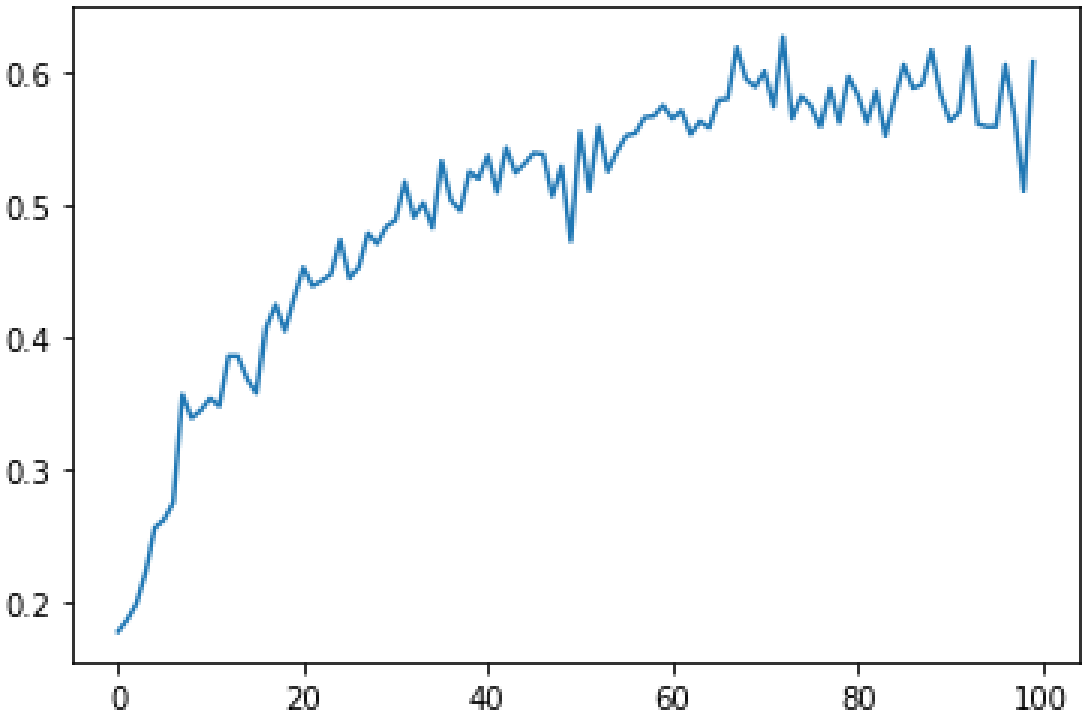
实验代码给出了一个简单的单层RNN实现，简单而言，主要包含两部分，一部分是根据当前的输入和历史hidden信息更新hidden信息，另一部分是根据当前的输入和历史的hidden生成output，这两部分可以使用两个简单的线性层来实现。RNN的结构如下

```
RNN(
  (i2h): Linear(in_features=185, out_features=128, bias=True)
  (i2o): Linear(in_features=185, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

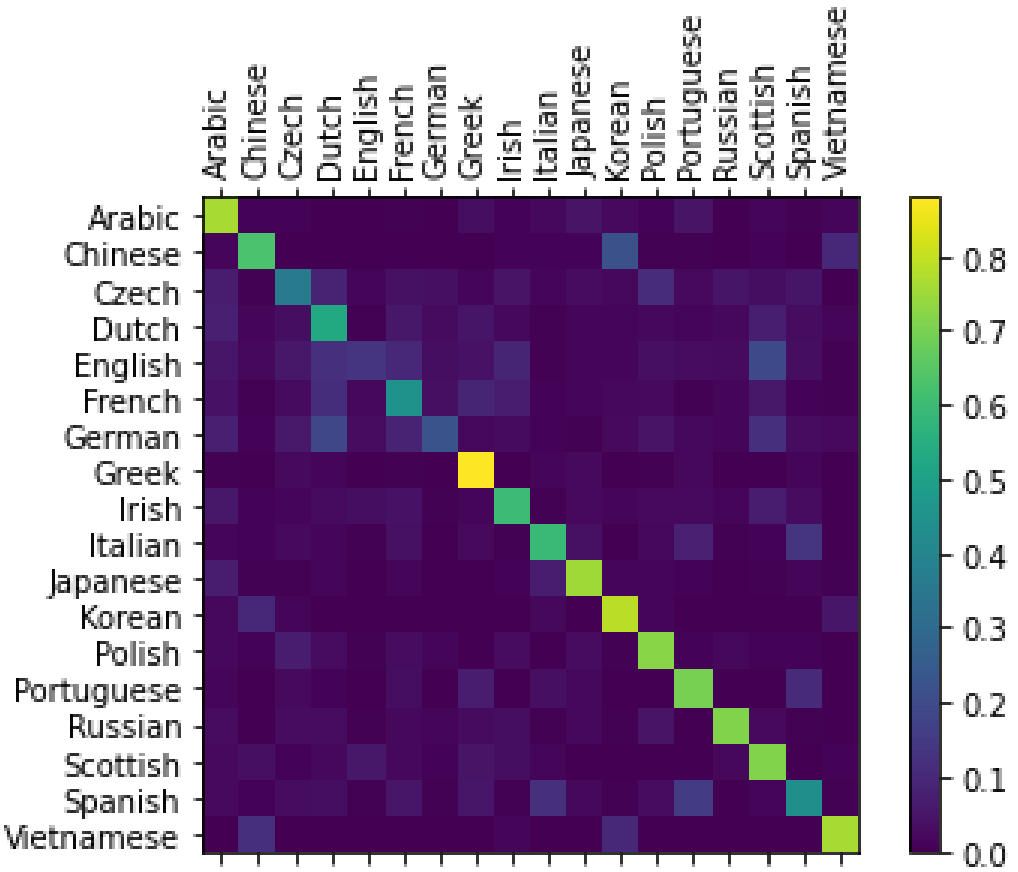
在进行训练的和推理的时候，依次将文本中的每一个单词作为input输入到RNN中，然后更新hidden，并且产生output，除了最后一个输入的output之外，其他的output都可以忽略。将最后一个单词的output拿来进行分类预测。

为了保证训练的公平性，选用Adam优化器，学习率设置为0.01，然后训练100000次迭代，每1000轮在验证集上验证，得到验证集上的loss和准确率如下图。





最终RNN的分类准确率达到了61%左右，在实验中进一步探究了每个类别的分类准确率，得到热力图如下。

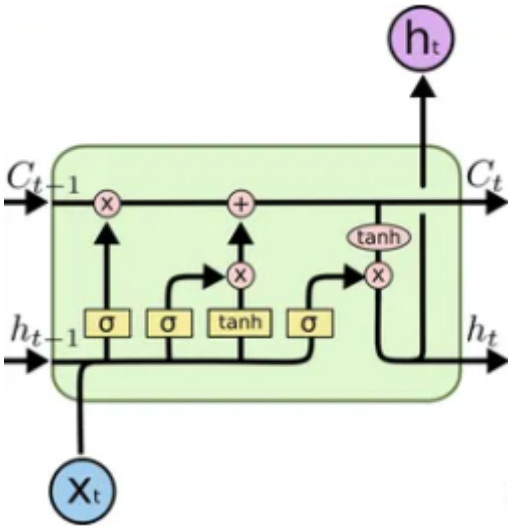


从结果中可以看出，RNN在绝大多数类别上的分类准确率还是比较高的，主要是在英语和德语的分类准确率上比较低，这是因为英语和德语是属于相同的语系，并且两者之间相互借鉴比较多，因此在人名上的区分度并不高，而要想很好地实现两个类比的区分，需要模型能够对名字进行整体的建模，而不是识别到特定的pattern之后就能够进行分类，而RNN对于长程建模的能力较差，因此对于这些相对比较近的名字并不能够进行很好地分类。导致了RNN模型在这两种类别上的分类效果不佳。而对于区分度比较高的希腊和韩国等，其分类准确率都接近了90%，这是因

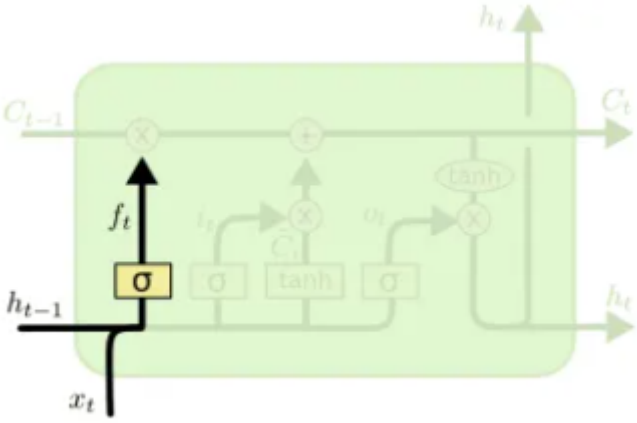
为这些名字中往往有一些很独特的字符，使得模型见到这类字符或者较短的pattern之后就能够做出正确的分类。

LSTM实验

在本次实验那种，自己实现了LSTM网络，为了简单起见，同时也为了和RNN进行公平的比较，这里同样只实现了单层的LSTM。

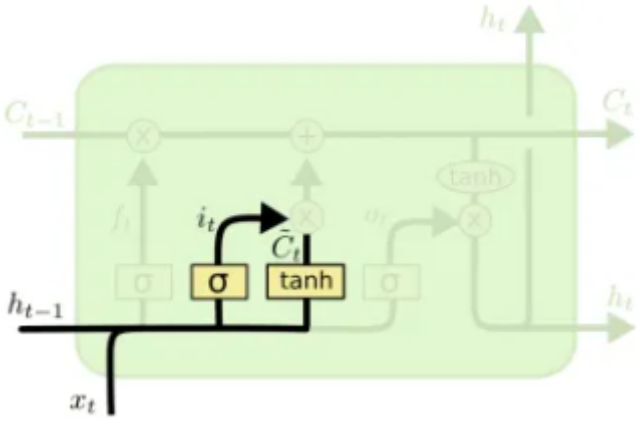


LSTM结构主要包含三个部分，分别是遗忘门、输入门和输出门，通过上述结构解决之前RNN中存在的长程依赖问题。这是因为原本的hidden中包含了前面全部的信息，这在不断地更新过程中会逐步失去对于前面比较远的距离的信息。LSTM为了解决上述问题，除了保存了hidden之外，还引入了cell状态，hidden负责建模短程表示，cell则建模长程表示。接下来就详细介绍一下各部分的实现。



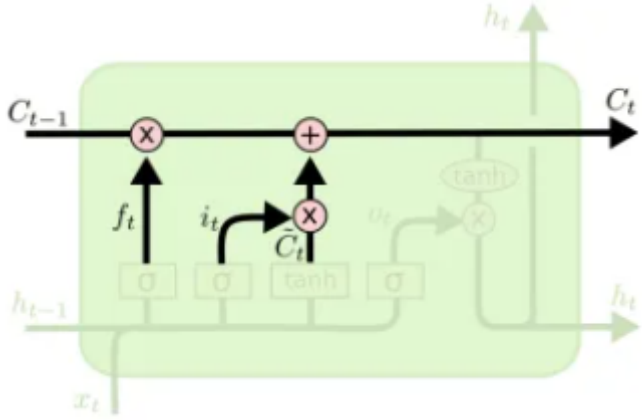
第一部分是遗忘门，用来筛选cell中的有效信息。其计算方式是将hidden和输入进行拼接，然后经过一个线性层后通过sigmoid函数，然后和cell相乘得到利用的前序信息。

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$



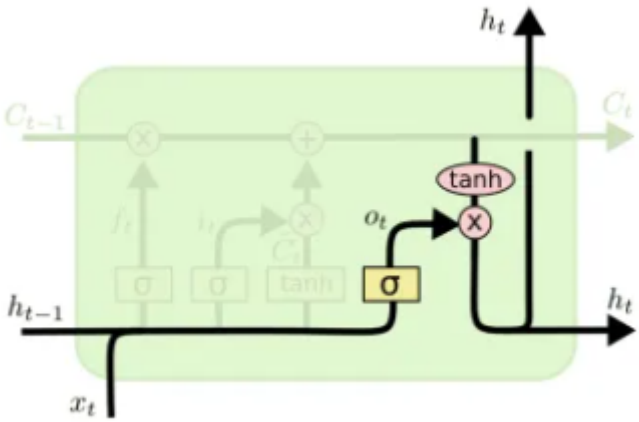
第二部分是输入门，用来确定输入信息应当考虑多少。同样是将hidden和input进行拼接然后经过一个线性层后使用sigmoid函数，得到一个权重值。另一侧要生成加入cell的信息，计算的过程一致。

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_C\right)$$



最后将遗忘门和输入门的值相加，得到更新后的cell。

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$



最后一部分是输出门，计算方式和输入门类似，公式如下

$$o_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$

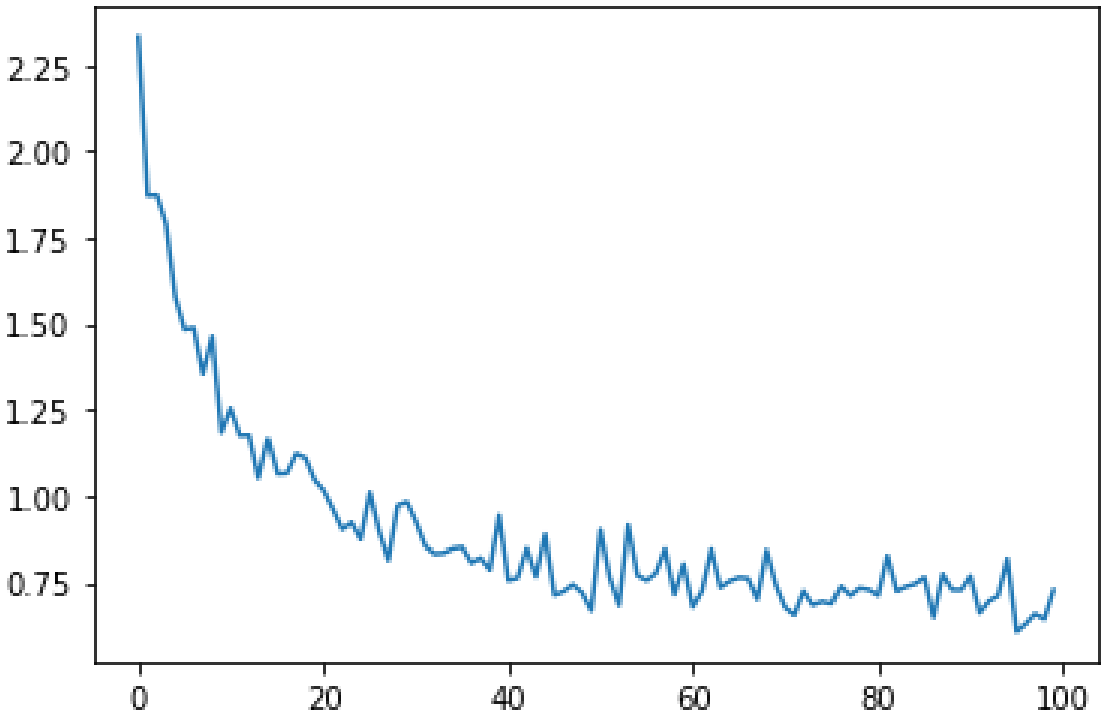
一个基本的LSTM块的代码如下

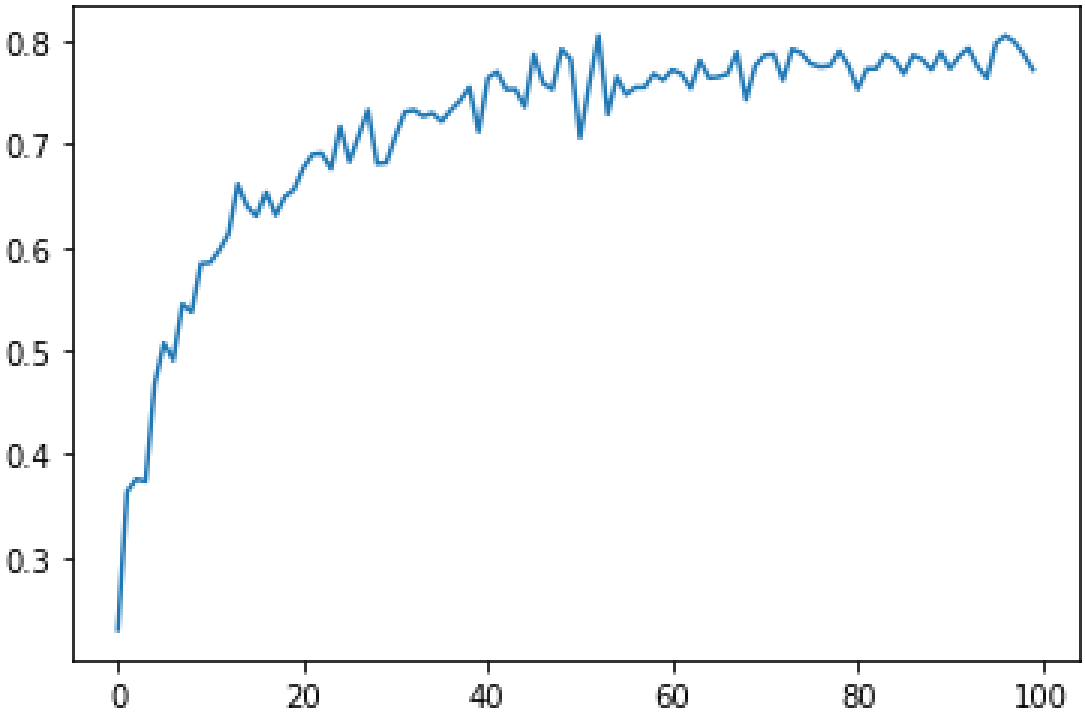
```
class LSTMBlock(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(LSTMBlock, self).__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.forget_gate = nn.Linear(in_features=input_dim+hidden_dim, out_features=hidden_dim)
        self.input_gate = nn.Linear(in_features=input_dim+hidden_dim, out_features=hidden_dim)
        self.cell_update = nn.Linear(in_features=input_dim+hidden_dim, out_features=hidden_dim)
        self.output_gate = nn.Linear(in_features=input_dim+hidden_dim, out_features=hidden_dim)

    def forward(self, x, hidden, cell):
        state = torch.concat((x, hidden), dim=-1)
        f = torch.sigmoid(self.forget_gate(state))
        i = torch.sigmoid(self.input_gate(state))
        c = torch.tanh(self.cell_update(state))
        cell = f * cell + i * c
        output = torch.sigmoid(self.output_gate(state))
        hidden = output * torch.tanh(cell)
        return output, hidden, cell
```

整个LSTM网络的结构如下

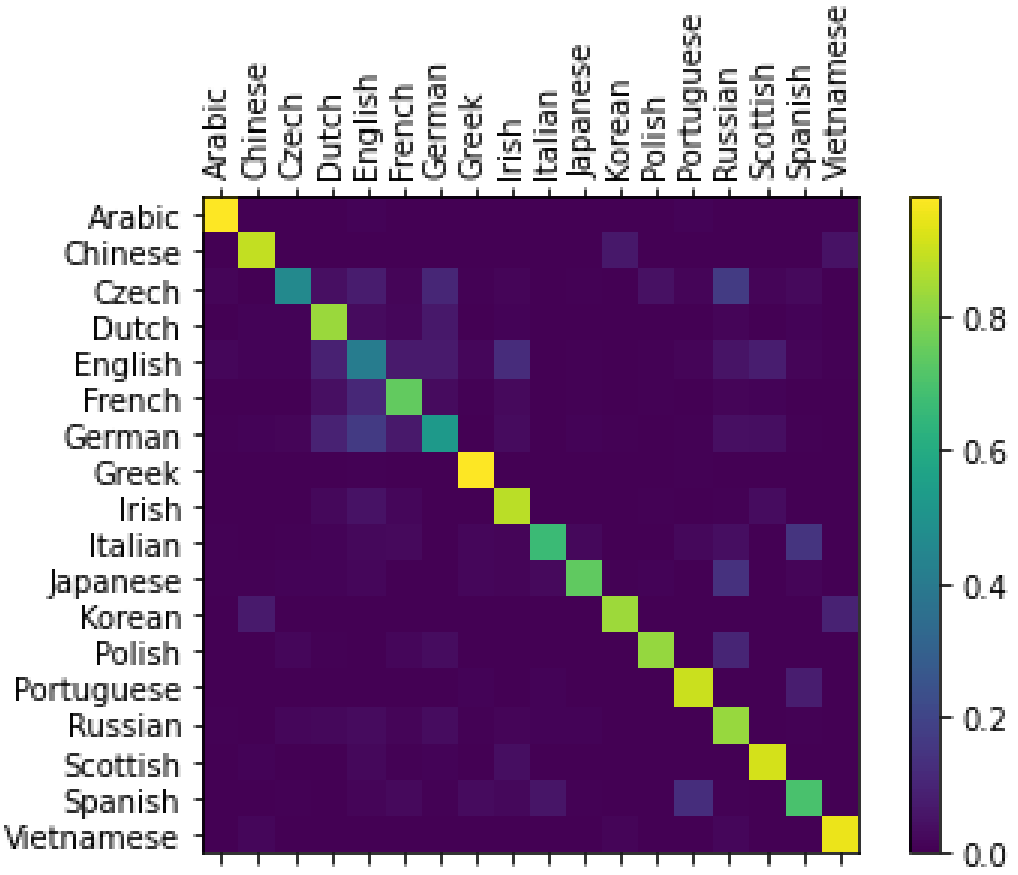
```
LSTM(
  (blocks): ModuleList(
    (0): LSTMBlock(
      (forget_gate): Linear(in_features=185, out_features=128, bias=True)
      (input_gate): Linear(in_features=185, out_features=128, bias=True)
      (cell_update): Linear(in_features=185, out_features=128, bias=True)
      (output_gate): Linear(in_features=185, out_features=128, bias=True)
    )
  )
  (classify): Linear(in_features=128, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

为了保证训练的公平性，选用Adam优化器，学习率设置为0.01，然后训练100000次迭代，每1000轮在验证集上验证，得到验证集上的loss和准确率如下图。





可以看到，LSTM最后达到了79.2%的分类准确率，要远高于RNN，并且相对来讲其收敛速度也更快一些。



通过进一步探究LSTM在各个类别上的分类准确率可以看出，除了Czech、English和German这三类，其他类别的分类准确率都已经非常高。在这三类准确率上也较RNN有了明显的提升。主要就是LSTM增强了长程建模能力，能够对名字整体抽取更好的特征，进而能够区分这些比较近的类别。

RNN和LSTM对比

LSTM相较于RNN，通过遗忘门、输入门和输出门改善了RNN中的长程建模能力。主要是通过LSTM中的cell信息和hidden信息，分别表示长程关系和短程关系。cell的更新相对比较慢，能够很好地记录前序较长距离的信息，而hidden和每次的输入密切相关，因此其对于建模短程关系比较有效。

但是LSTM虽然能够改善RNN的长程表示能力，但是其并没有能完全解决，从实验结果中也能够看出，对于那些名字比较相近的类别，显然是要建模整体的特征才能够做出很好的分类判断。LSTM在这些类比上虽然较RNN有提升，但是仍不能够达到比较好的结果。

此外，由于LSTM中增加了运算，因此导致其训练速度和推理速度都要慢于RNN，因此这也给训练增加了成本。