Classifying Person's Name as per Origin/Nationality using Char-RNN as Part of Extension to Open-Minded tutorial.

Presentation Overview

1. Project Overview
2. Data
3. RNN
4. Data Preparation
5. Implementation
6. Evaluation

Project SUMMARY::

- As part of the coding project(Part 2-How to train Recurrent Neural Network) from tutorial of OpenMinded.

- Link of tutorial:: https://blog.openmined.org/federated-learning-of-a-rnn-on-raspberry-pis/

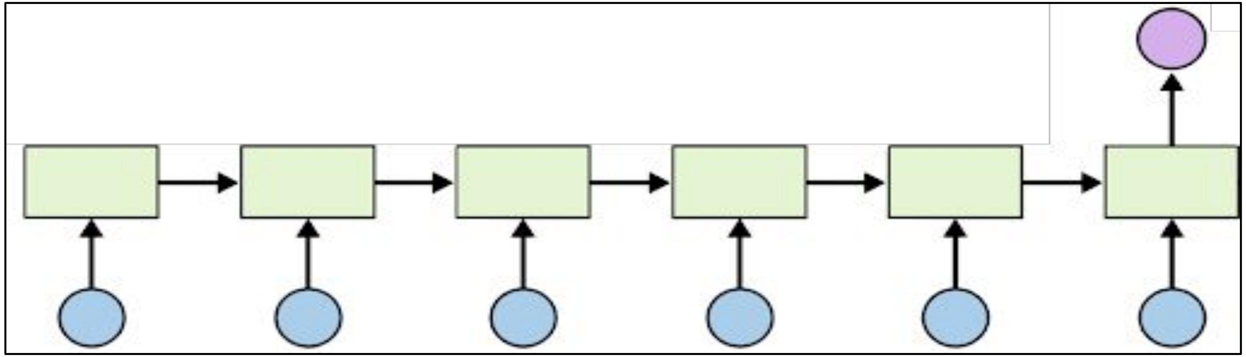
- Need to classify person's surname to its most likely language of origin in a federated way,

Names Classification DataSet::

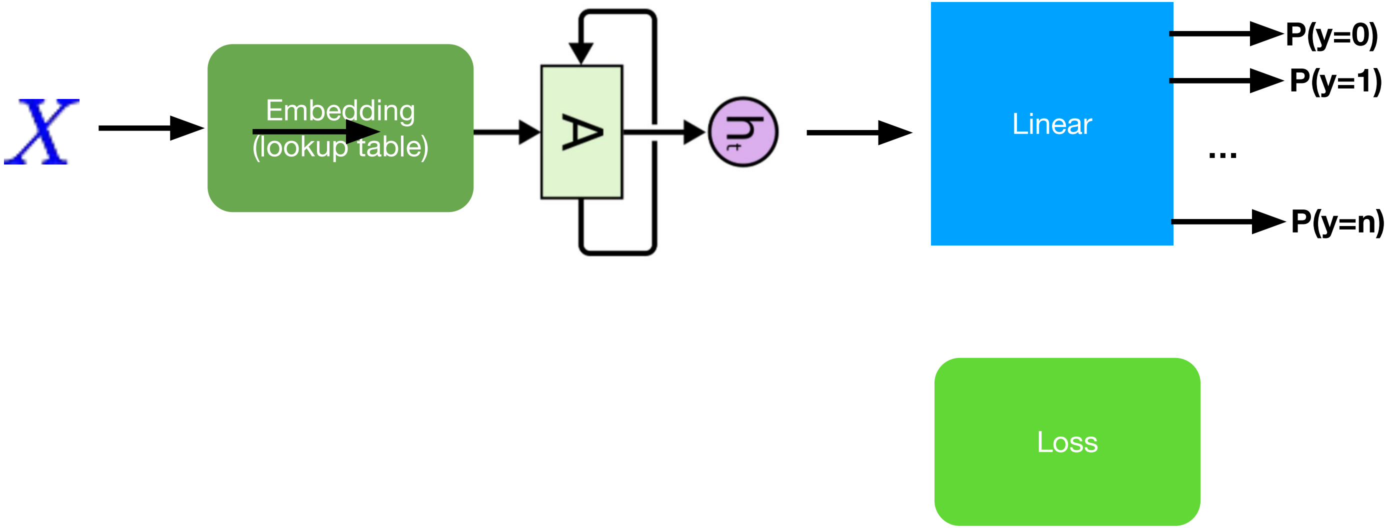
Abatangelo	Italian
Ahearn	Irish
Aalst	Dutch
Abbey	English
Abbott	English
Abatantuono	Italian
Aodh	Irish
Abate	Italian
Achteren	Dutch
Brown	Scottish

Softmax Output

(18 countries)

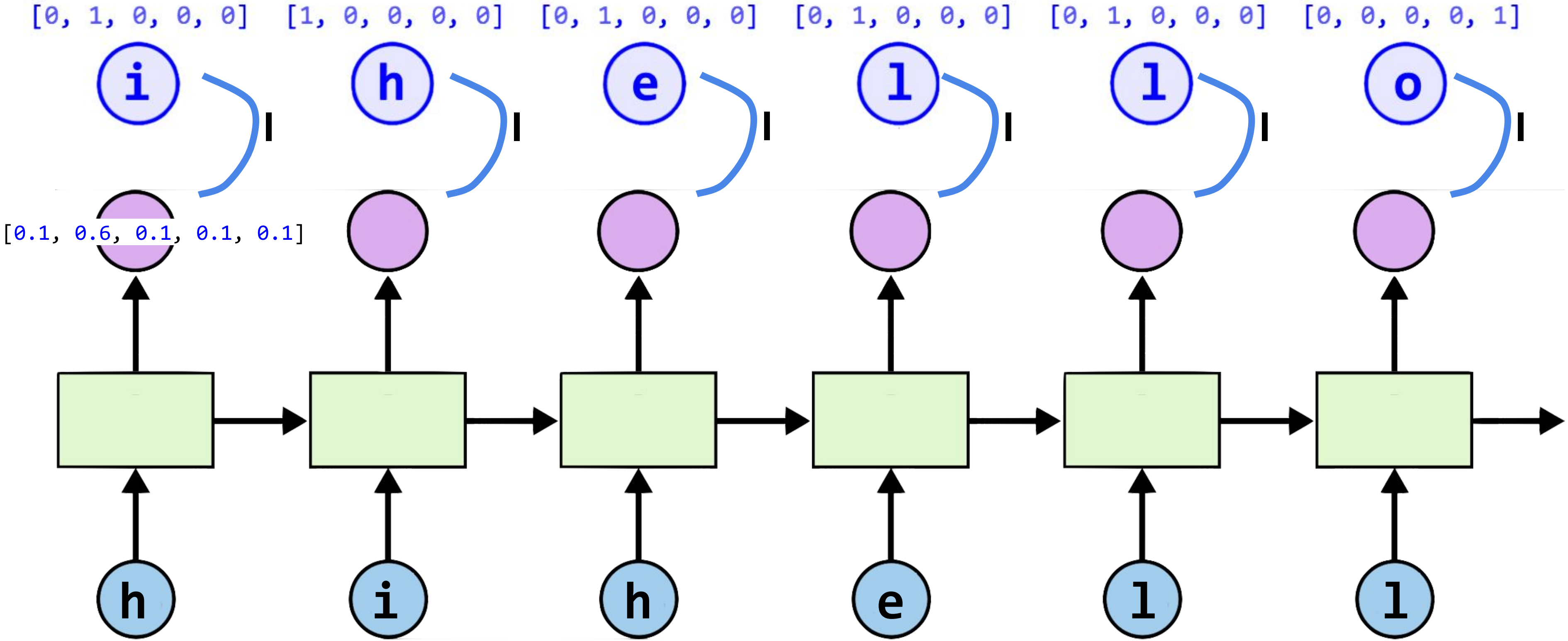


Typical RNN Models

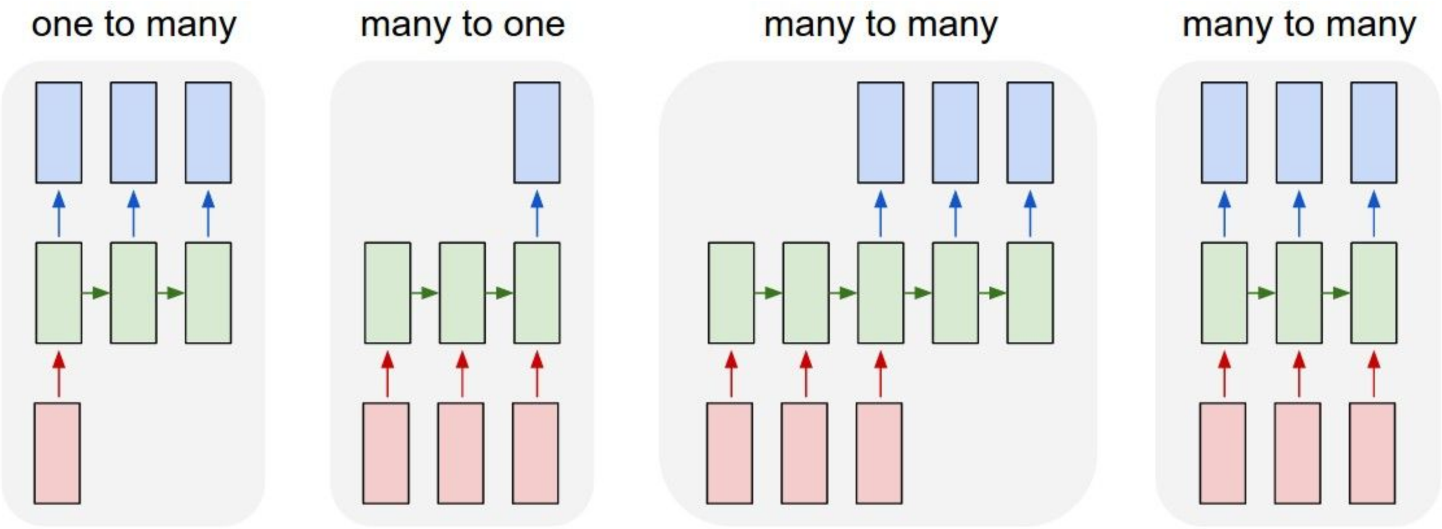


With CrossEntropy

RNN Loss and training

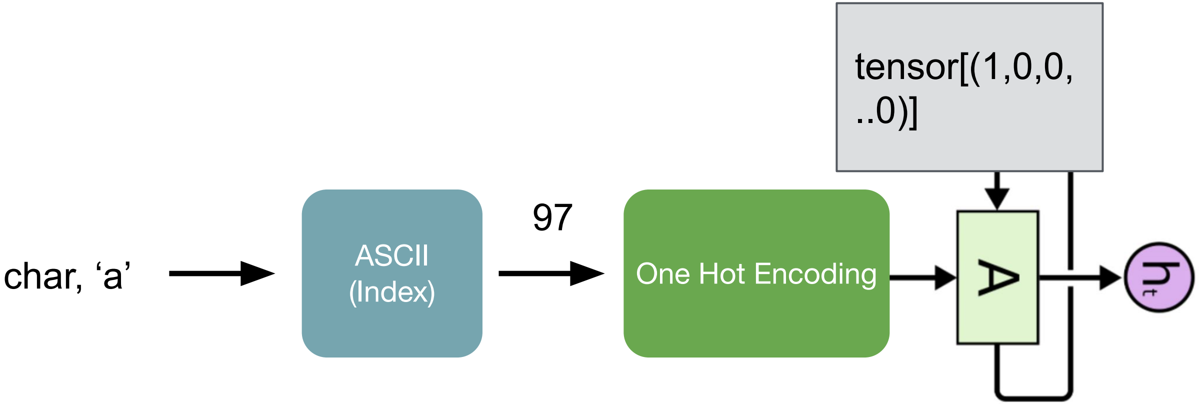


RNN Applications



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Char embedding



char, 'a'

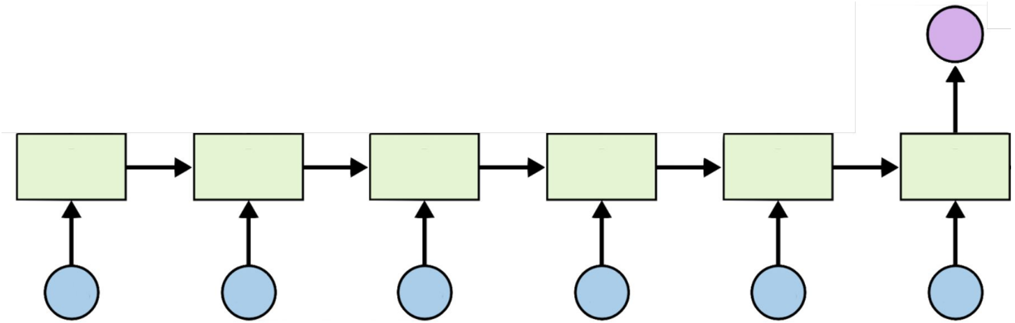
ASCII (Index)

97

One Hot Encoding

tensor[(1,0,0, ..0)]

Input representation



Softmax output
(18 countries)

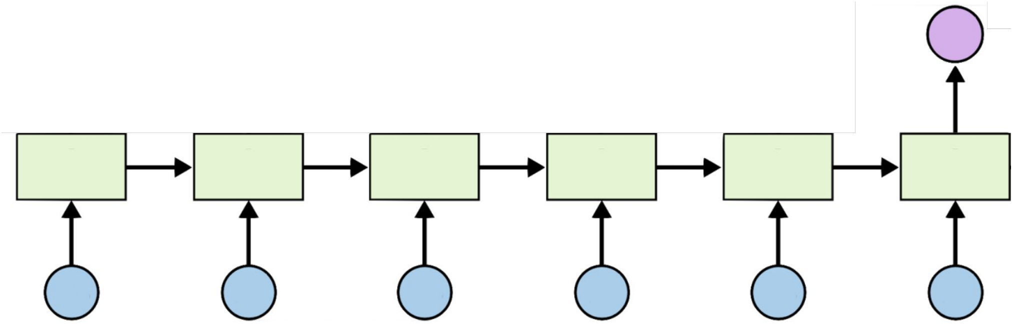
tensor[(1,0..0)] …….tensor[(0,0,..1])(**One Hot Encoder**)

97 100 121 108 111 118(**ascii/Indices**)

a d y l o v **(Char)**

Matrix visualization from Nicolas, https://github.com/ngarneau

Data Preparation



tensor[(1,0..0)] …….tensor[(0,0,..1])(**One Hot Encoder**)

97 100 121 108 111 118(**ascii/Indices**)

a d y l o v **(Char)**

```
#To covert to one
hot encoder
def
lineToTensor(line):.
.........
```

```
def unicodeToAscii(s):
    return ''.join(
        c for c in
unicodedata.normalize('NFD',
```

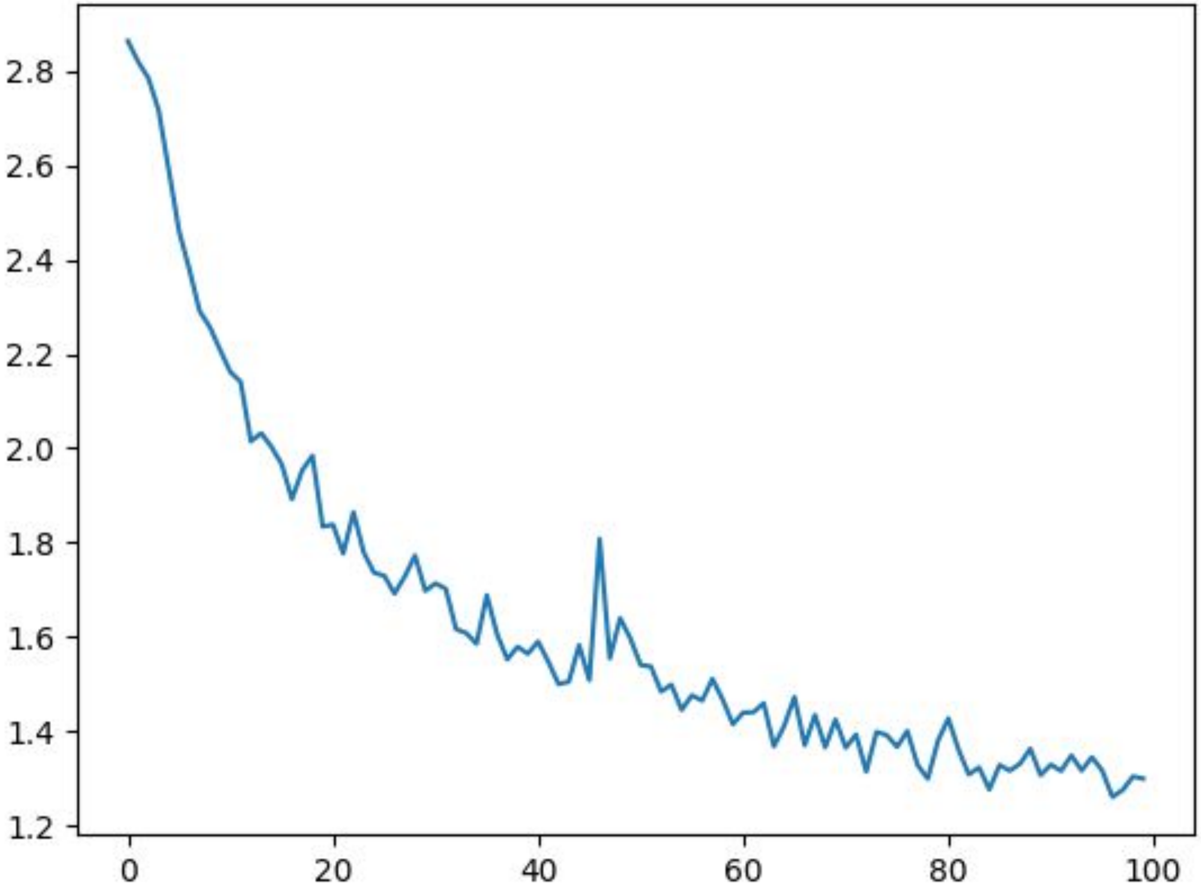
Matrix visualization from Nicolas, https://github.com/ngarneau

Implementation

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size
        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden
def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

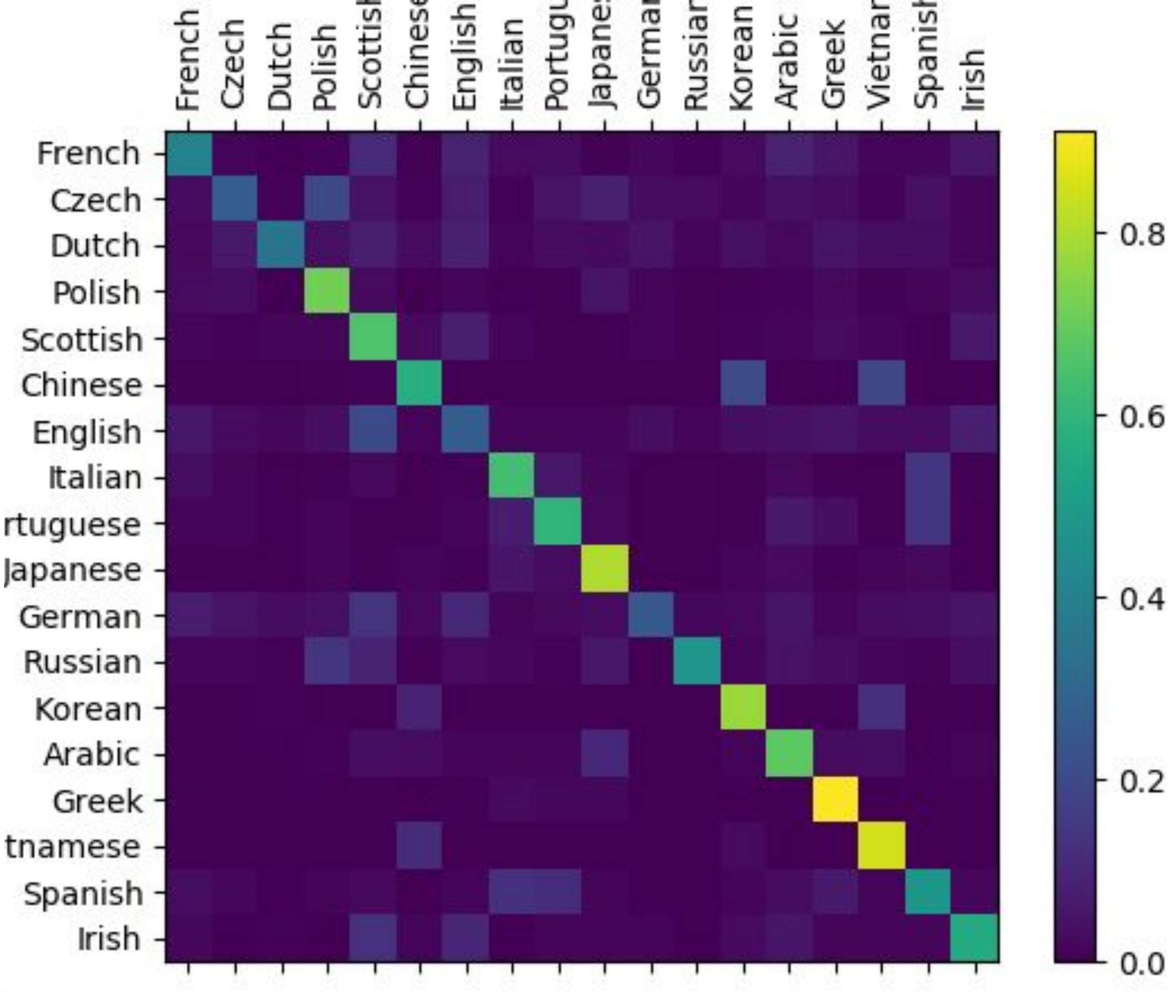
Implementation

```
def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()
    rnn.zero_grad()
    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)
   loss = criterion(output, category_tensor)
    loss.backward()
for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.item()
```



Plot of all losses in network to show Learning

Confusion Matrix:: To show how network perform on different categories


French
Czech
Dutch
Polish
Scottisl
Chinese
English
Italian
Portugu
Japanes
Germa
Russiar
Korean
Arabic
Greek
Vietnar
Spanisl
Irish
French
Czech
Dutch
Polish
Scottish
Chinese
English
Italian
rtuguese
Japanese
German
Russian
Korean
Arabic
Greek
tnamese
Spanish
Irish