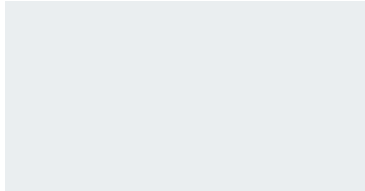
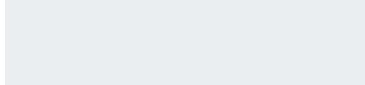


```

149
150
151 # ## Step 4a: Train the Network
152 # Our loss function, `NLLLoss`,
    checks a Negative Log Likeliho
    d (NLL) tensor against a target
    tensor that specifies which ind
    ex of the NLL tensor should be
    highest. Example:
153 # ✅ NLL tensor: `[-3, -1, -2]`
    , target: `[1]`
154 # ❌ NLL tensor: `[-1, -3, -2]`
    , target: `[1]`
155 #
156 # Let's train for **5 epochs**.
    We will also be performing basi
    c evaluation *alongside* traini
    ng. This is a good practice bec
    ause you can then detect **over
    fitting**, **insufficient learn
    ing rate**, and other issues.
157
158 # In[7]:
159
160
161 from tqdm import tqdm
162 import matplotlib.pyplot as plt
    
163
164 LEARNING_RATE = 0.01
165 criterion = nn.NLLLoss()
166
167 train_average_losses = []
168 test_average_losses = []
    
169
170 for epoch_index in tqdm(range(5
    ))):
171
172     # TRAIN
173
174     train_losses = []
175
176     for step_index, batch in en
    umerate(train_dataloader):
177
178         input_tensor, expected_
    output = batch
179
180         hidden_tensor = rnn.ini
    tHidden(input_tensor.shape[0])
181
182

```

```

149
150
151 # ## Step 4a: Train the Network
152 # Our loss function, `NLLLoss`,
    checks a Negative Log Likeliho
    d (NLL) tensor against a target
    tensor that specifies which ind
    ex of the NLL tensor should be
    highest. Example:
153 # ✅ NLL tensor: `[-3, -1, -2]`
    , target: `[1]`
154 # ❌ NLL tensor: `[-1, -3, -2]`
    , target: `[1]`
155 #
156 # Let's train for **5 epochs**.
    We will also be performing basi
    c evaluation *alongside* traini
    ng. This is a good practice bec
    ause you can then detect **over
    fitting**, **insufficient learn
    ing rate**, and other issues.
157
158 # In[7]:
159
160
161 from tqdm import tqdm
162 import matplotlib.pyplot as plt
163 from torch.optim import SGD
164 from torch.utils.tensorboard im
    port SummaryWriter
165
166 writer = SummaryWriter(flush_se
    cs=1)
167
168 LEARNING_RATE = 0.01
169 criterion = nn.NLLLoss()
170
171 train_average_losses = []
172 test_average_losses = []
173
174 optimizer = SGD(rnn.parameters(
    ), lr=LEARNING_RATE)
175
176 for epoch_index in tqdm(range(5
    ))):
177
178     # TRAIN
179
180     train_losses = []
181
182     for step_index, batch in en
    umerate(train_dataloader):
183
184         input_tensor, expected_
    output = batch
185
186         hidden_tensor = rnn.ini
    tHidden(input_tensor.shape[0])
187
188

```

```

181         rnn.zero_grad()
182
183         for i in range(input_tensor.shape[1]):
184             if len(torch.nonzero(input_tensor[:, i, :])) > 0:
185                 output_tensor,
186                 hidden_tensor = rnn(input_tensor[:, i, :], hidden_tensor)
187
188                 loss = criterion(output_tensor, expected_output)
189                 loss.backward()
190
191                 # Add parameters' gradients to their values, multiplied by learning rate
192                 for p in rnn.parameters():
193                     p.data.add_(-LEARNING_RATE, p.grad.data)
194
195                 train_losses.append(loss.item())
196
197                 train_average_losses.append((sum(train_losses) / len(train_losses)))
198
199
200         # TEST
201
202         test_losses = []
203
204         with torch.no_grad():
205             for step_index, batch_index in enumerate(test_dataloader):
206
207                 input_tensor, expected_output = batch
208
209                 hidden_tensor = rnn.initHidden(input_tensor.shape[0])
210
211                 for i in range(input_tensor.shape[1]):
212                     if len(torch.nonzero(input_tensor[:, i, :])) > 0:
213
214                         output_tensor, hidden_tensor = rnn(input_tensor[:, i, :], hidden_tensor)
215
216                         loss = criterion(output_tensor, expected_output)
217

```

```

187         rnn.zero_grad()
188
189         for i in range(input_tensor.shape[1]):
190             if len(torch.nonzero(input_tensor[:, i, :])) > 0:
191                 output_tensor,
192                 hidden_tensor = rnn(input_tensor[:, i, :], hidden_tensor)
193
194                 loss = criterion(output_tensor, expected_output)
195                 loss.backward()
196
197                 optimizer.step()
198
199
200                 train_losses.append(loss.item())
201
202                 train_average_losses.append((sum(train_losses) / len(train_losses)))
203
204                 writer.add_scalar('Train Loss', sum(train_losses) / len(train_losses), epoch_index)
205
206
207         # TEST
208
209         test_losses = []
210
211         with torch.no_grad():
212             for step_index, batch_index in enumerate(test_dataloader):
213
214                 input_tensor, expected_output = batch
215
216                 hidden_tensor = rnn.initHidden(input_tensor.shape[0])
217
218                 for i in range(input_tensor.shape[1]):
219                     if len(torch.nonzero(input_tensor[:, i, :])) > 0:
220
221                         output_tensor, hidden_tensor = rnn(input_tensor[:, i, :], hidden_tensor)
222
223                         loss = criterion(output_tensor, expected_output)
224

```

```

        tput_tensor, expected_output)
213         test_losses.append(
        loss.item())
214
215         test_average_losses.append(
        sum(test_losses) / len(test_losses))
216
217
218 # In[10]:
219
220
221
222 plt.figure()
223 plt.plot(train_average_losses,
        "r")
224 plt.plot(test_average_losses, "
        b")
225 plt.xlabel("Epoch")
226 plt.ylabel("Average Loss")
227
228
229 # ## Step 4b: Add an optimizer
230 # You can use an optimizer included with PyTorch rather than writing your own.
231 # While the RNN trains here, we
        'll write this part in VS Code.
232
233 # ## Step 5: Evaluate the Network
234 # There are many metrics for evaluating a classifier, e.g. F1 score, accuracy, etc.
235 # For this tutorial, we'll create a confusion matrix. **Rows are actual/target languages, columns are predictions.**
236
237 # In[9]:
238
239
240 import matplotlib.ticker as ticker
241
242 confusion = torch.zeros(len(languages), len(languages))
243
244 with torch.no_grad():
245     for step_index, batch in enumerate(test_data_loader):
246
247         input_tensor, expected_output = batch
218         test_losses.append(
        loss.item())
219
220         test_average_losses.append(
        sum(test_losses) / len(test_losses))
221
222 writer.add_scalar('Test Loss', sum(test_losses) / len(test_losses), epoch_index)
223
224 # In[10]:
225
226
227
228 plt.figure()
229 plt.plot(train_average_losses,
        "r")
230 plt.plot(test_average_losses, "
        b")
231 plt.xlabel("Epoch")
232 plt.ylabel("Average Loss")
233
234
235 # ## Step 4b: Add an optimizer
236 # You can use an optimizer included with PyTorch rather than writing your own.
237 # While the RNN trains here, we
        'll write this part in VS Code.
238
239 # ## Step 5: Evaluate the Network
240 # There are many metrics for evaluating a classifier, e.g. F1 score, accuracy, etc.
241 # For this tutorial, we'll create a confusion matrix. **Rows are actual/target languages, columns are predictions.**
242
243 # In[9]:
244
245
246 import matplotlib.ticker as ticker
247
248 confusion = torch.zeros(len(languages), len(languages))
249
250 with torch.no_grad():
251     for step_index, batch in enumerate(test_data_loader):
252
253         input_tensor, expected_output = batch

```