**Artificial Intelligence Final Report Assignment 問題2 (Problem 2)**

**レポート解答用紙 (Report Answer Sheet)**

**(Group 35)**

Group Leader

学生証番号 (Student ID): 21522357

名前(Name): Lê Hải Nam

Group Members

学生証番号 (Student ID): 21522429

名前(Name): Trần Nguyễn Yến Nhi

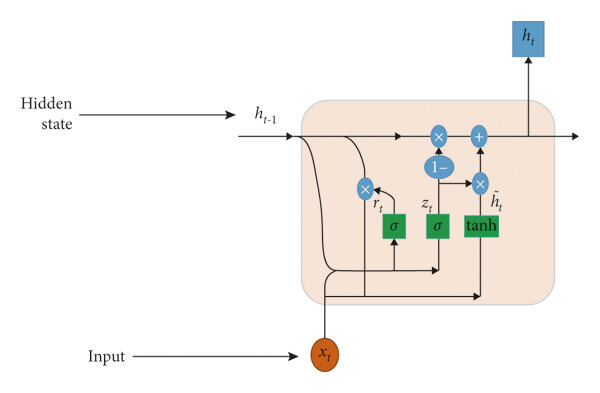
学生証番号 (Student ID): 21522315

名前(Name): Nguyễn Thị Cẩm Ly

問題2 (Problem 2)のレポート

**Idea:**

To improve the version of the program in the 12th lecture (or the program was written in Lab Work (5)). We use the **GRU model (RNN model)** instead of the **Simple RNN** - method that brings results in the 12th lecture with accuracy of about 68%.



*Figure 1:GRU model*

**Gated Recurrent Unit (GRU)** is a type of recurrent neural network (RNN) that was introduced by Cho et al. in 2014 as a simpler alternative to Long Short-Term Memory (LSTM) networks. GRU networks use gating mechanisms to selectively update the hidden state at each time step, allowing them to effectively model sequential data. They have been shown to be effective in various natural language processing tasks, such as language modeling, machine translation, and speech recognition

**Program:**

1. **Define model**

The **MyGRU** class implementation defines a recurrent neural network architecture, specifically utilizing a Gated Recurrent Unit (GRU) for sequence modeling tasks. The architecture begins with an embedding layer that converts input word indices into dense vectors, followed by a GRU layer that captures temporal dependencies in the data. The final fully connected layer transforms the GRU's output into the desired output size, suitable for classification tasks. The use of GRU ensures efficient handling of sequential data, and the forward method seamlessly integrates the embedding, GRU, and fully connected layers to provide a robust deep learning model ideal for various natural language processing applications.

We define a simple GRU model for text classification:

class MyGRU(torch.nn.Module):

def \_\_init\_\_(self, vocab\_size):

super(MyGRU, self).\_\_init\_\_()

embedding\_dim = 300

hidden\_dim = 128

self.embedding = nn.Embedding(vocab\_size, embedding\_dim)

self.rnn = nn.GRU(embedding\_dim, hidden\_dim, batch\_first=True)

self.fc = nn.Linear(hidden\_dim, 2)

def forward(self, x):

embedded = self.embedding(x)

\_, hidden = self.rnn(embedded)

output = self.fc(hidden[-1])

return output

1. **Train model**

def train(EPOCH):

model = MyGRU(len(vocablist)).to(DEVICE)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

criterion = nn.CrossEntropyLoss()

for epoch in range(EPOCH):

total\_loss = 0

correct = 0

total = 0

model.train()

for tokenlists, labels in train\_data:

tokenlists = torch.tensor(tokenlists, dtype=torch.long).to(DEVICE)

labels = torch.tensor(labels, dtype=torch.long).to(DEVICE)

optimizer.zero\_grad()

outputs = model(tokenlists)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

total\_loss += loss.item()

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

print("Epoch:", epoch+1, "Loss:", total\_loss, "Train Accuracy:", (correct / total))

torch.save(model.state\_dict(), MODELNAME)

1. **Test model**

def test():

model = MyGRU(len(vocablist)).to(DEVICE)

model.load\_state\_dict(torch.load(MODELNAME))

model.eval()

correct = 0

total = 0

with torch.no\_grad():

for tokenlists, labels in test\_data:

tokenlists = torch.tensor(tokenlists, dtype=torch.long).to(DEVICE)

labels = torch.tensor(labels, dtype=torch.long).to(DEVICE)

outputs = model(tokenlists)

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

print("correct:", correct)

print("total:", total)

print("Test Accuracy:", (correct / total))

**Execution Results:**

After switching from Simple RNN to GRU with the same number of epochs (10 epochs), using GRU is significantly more effective.

|  |  |
| --- | --- |
| **Before** | **After** |
| epoch 1 : loss 241.03739431500435 | Epoch: 1 Loss: 206.8136618360877 Train Accuracy: 0.73636 |
| epoch 2 : loss 237.84667938947678 | Epoch: 2 Loss: 158.2841546498239 Train Accuracy: 0.8176 |
| epoch 3 : loss 237.16132572293282 | Epoch: 3 Loss: 112.17883889377117 Train Accuracy: 0.88016 |
| epoch 4 : loss 236.7474660873413 | Epoch: 4 Loss: 71.38435392826796 Train Accuracy: 0.92912 |
| epoch 5 : loss 236.41975903511047 | Epoch: 5 Loss: 43.267548041418195 Train Accuracy: 0.95984 |
| epoch 6 : loss 236.13036131858826 | Epoch: 6 Loss: 26.47362814983353 Train Accuracy: 0.97756 |
| epoch 7 : loss 235.86326578259468 | Epoch: 7 Loss: 24.264955684542656 Train Accuracy: 0.97828 |
| epoch 8 : loss 235.59672728180885 | Epoch: 8 Loss: 14.348241922794841 Train Accuracy: 0.9882 |
| epoch 9 : loss 235.33114612102509 | Epoch: 9 Loss: 8.112489654886303 Train Accuracy: 0.99384 |
| epoch 10 : loss 235.06243985891342 | Epoch: 10 Loss: 4.268304551369511 Train Accuracy: 0.9968 |
| correct: 17087  total: 25000  accuracy: 0.68348 | correct: 21978  total: 25000  Test Accuracy: 0.87912 |

Finally, we achieved the highest result after 50 epochs, reaching an accuracy of **88.5%**.

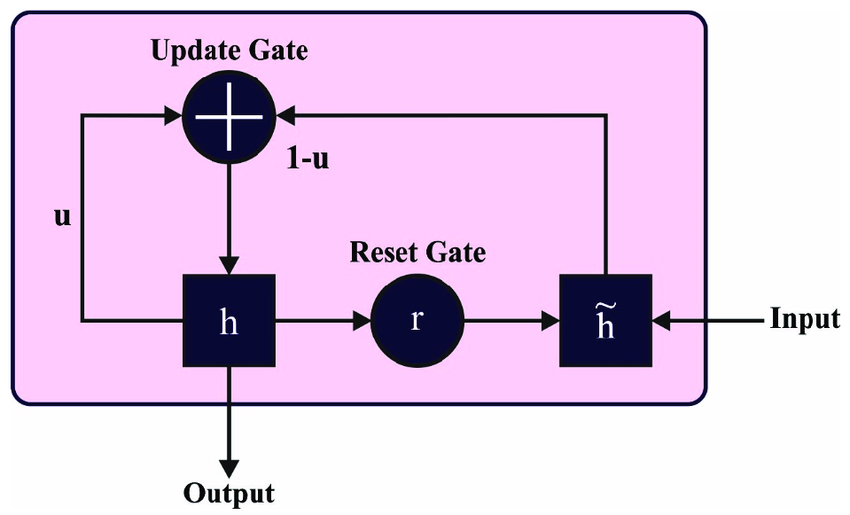
**Explanation:**

The different gates of a GRU are as described below:

**Update Gate:** It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.

**Reset Gate:** It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.

**Current Memory Gate:** It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some nonlinearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future.



*Figure 2: Architecture of GRU.*

Improvements of GRU compared to RNN on the IMDb dataset:

1. **Network Architecture:**

**GRU:** Utilizes a GRU, a type of recurrent neural network that includes gating mechanisms to control the flow of information and manage hidden states more effectively. GRUs can remember longer sequences better than traditional RNNs.

**RNN:** Uses a simple RNN architecture with linear layers and ReLU activation functions. This architecture lacks the sophisticated gating mechanisms of GRUs, making it prone to issues with long-term dependencies due to vanishing/exploding gradients.

1. **Embedding Handling:**

**GRU:** Employs an embedding layer to convert vocabulary indices into dense vectors, which are then processed by the GRU.

**RNN:** Similarly uses an embedding layer, but the embeddings are processed through linear layers without the advanced state management capabilities of GRUs.

1. **Memory Capability:**

**GRU:** Features update and reset gates that help in selectively remembering or forgetting information. This capability allows GRUs to handle longer sequences more effectively, maintaining relevant context over extended periods.

**RNN:** Traditional RNNs often struggle with retaining information over long sequences, making them less effective for tasks that require understanding long-term dependencies.

1. **Training and Performance:**

**GRU:** Generally achieves higher performance in tasks that require long-term memory, such as sentiment classification of movie reviews, by effectively managing context over long sequences. As a result, GRUs tend to have better accuracy on such tasks.

**RNN:** While simpler and potentially easier to train due to fewer parameters, traditional RNNs usually underperform compared to GRUs on tasks involving long sequences, due to their inherent memory limitations.

***Hidden layers***

The key component that enables a neural network to learn complex tasks and achieve excellent performance is hidden layers. They are instrumental in tasks such as distinguishing between praise and criticism within a sentiment.

For example, the network could learn that words like "excellent," "impressive," and "outstanding" often accompany praise, while words like "poor," "disappointing," and "low-quality" often indicate criticism. The hidden layers would enable the network to learn and capture such complex rules from the training data, improving its ability to correctly classify sentiments as either praise or criticism.

***Configure model***



*Figure 3: Configure model.*

We configure the model as in Lab Work (5) and add 128 hidden layers.

⇒For the **IMDb dataset**, which involves classifying the sentiment of movie reviews, the **GRU model** is likely to outperform the **RNN model**. This is because GRUs are designed to handle long-term dependencies better, making them more suitable for understanding the context and nuances of lengthy reviews. Consequently, **GRU** should provide higher accuracy and more robust performance in sentiment classification tasks compared to **RNN**.