AR1 Model Coefficients Forecast Using BERT

A toy data case

Ethan Sun, 7 Nov 2023.

Objectives

Forecast the a(x) and B(x) coefficients from the textual data x

- a(x) and B(x) are further used as the coefficients for an AR1 model
- **x** is the dialogue-based textual information

Data

- Utilised features:

  Textual dialogues

  Y_{t}

  Y_{t+1}

- Un-utilised features:

  date

  ID

Approach

- Dialogue Embedding
- Model Structure
- Loss configuration
- Hyper-parameters
- Results
- Discussions

Approach

Dialogue Embedding

- Bert LongformerModed

```
# Load Longformer model and tokenizer
tokenizer = LongformerTokenizer.from_pretrained('allenai/longformer-base-4096')
model = LongformerModel.from_pretrained('allenai/longformer-base-4096')


# dialogue embedding into 786-d feature
def process_dialogue(dialogue):
    inputs = tokenizer(dialogue, return_tensors='pt', padding='max_length', truncation=True, max_length=4096) # tokenize and convert text to input tensors
    with torch.no_grad(): # generate BERT embeddings for the entire dialogue
        outputs = model(**inputs)
    cls_embedding = outputs.last_hidden_state[:, 0, :].numpy() # extract the embedding for the [CLS] token
    return cls_embedding

embeddings_file = 'bert_embeddings_long.npy'
if os.path.exists(embeddings_file):
    embeddings = np.load(embeddings_file)
    print('embedding loaded.')
else:
    embeddings_list = df['Text'].apply(process_dialogue).tolist()
    embeddings = torch.cat([torch.tensor(embedding) for embedding in embeddings_list], dim=0)
    np.save(embeddings_file, embeddings)
    print('embedding saved.')
```

Approach

Train/test split

```
# training/validation sets
train_embeddings, val_embeddings = train_test_split(embeddings, test_size=0.3, shuffle=False)
train_embeddings = torch.tensor(train_embeddings).detach().requires_grad_(False)
val_embeddings = torch.tensor(val_embeddings).detach().requires_grad_(False)

supports = torch.tensor(df['Yt'].values).detach().requires_grad_(False)  # Y_t
targets = torch.tensor(df['Yt+1'].values).detach().requires_grad_(False)  # Y_t+1
train_targets, val_targets = train_test_split(targets, test_size=0.3, shuffle=False)
train_supports, val_supports = train_test_split(supports, test_size=0.3, shuffle=False)
```

Approach

Model Structure

```
# build model
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = nn.Linear(768, 64)  # BERT embedding dimension to hidden layer 1
        self.fc2 = nn.Linear(64, 16)  # hidden layer 1 to hidden layer 2
        self.fc3 = nn.Linear(16, 2)  # hidden layer 2 to output layer

    def forward(self, x):
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Approach

Loss Configuration (on training set)

- Take the mean of \hat{a} and \hat{b}

```
para = model(train_embeddings) #shape (n,2)
a, b = para[:,0].mean(), para[:,1].mean() #we take mean of from training data as hat{a} and hat{b}
outputs = a + b*train_supports
loss = criterion(outputs, train_targets)
```

Approach

Loss Configuration (on validation set)

```
a_, b_ = para[:,0].mean(), para[:,1].mean()
outputs = a_ + b_*val_supports
loss = criterion(outputs, val_targets)
val_losses.append(loss.item())
```

(a)

```
#use a and b generated during the training session
outputs = a + b*val_supports
loss = criterion(outputs, val_targets)
val_losses2.append(loss.item())
```

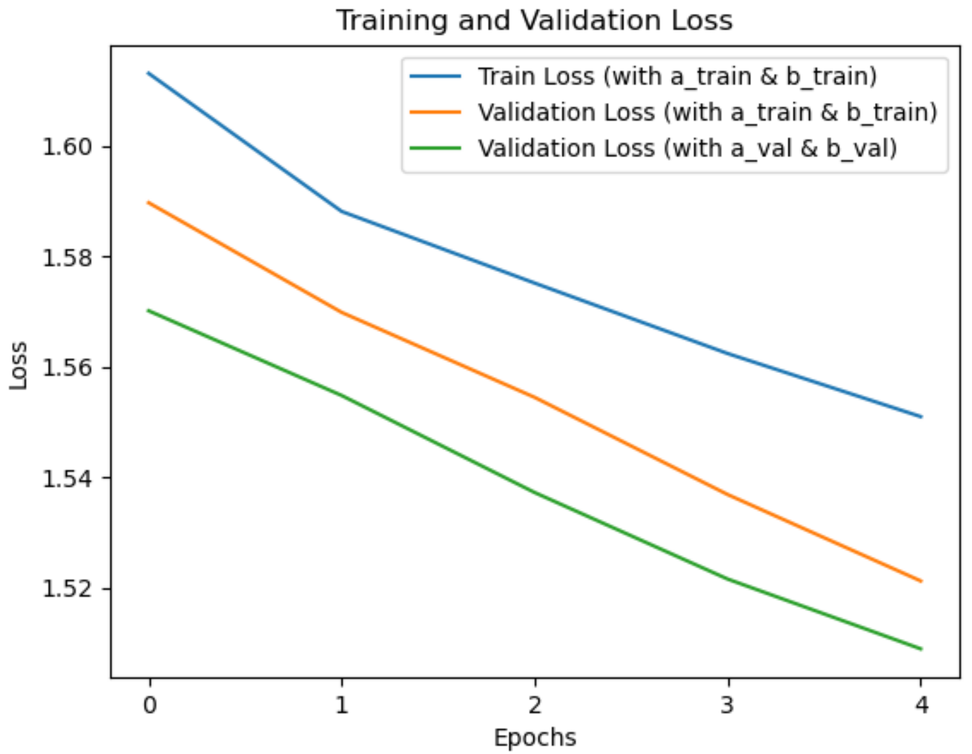
(b)

Hyper-Parameter

- Epochs = 5
- Learning rate = 0.001

Result

Visual Diagrams for Multiple Restarts



Result

Discussions and Future Work

- Problem: Too Long Dialogues (truncation will loss the info)
- Solution: Dialogue-based padding (way too time-consuming!)

```python
def get_embeddings(text):
    sentences = text.split('\n')  # Split dialogue into sentences
    sentence_embeddings = []
    for sentence in sentences:
        sentence.replace('\r','')
        inputs = tokenizer(sentence, return_tensors='pt', padding='max_length', truncation=True, max_length=4096)

        with torch.no_grad():
            outputs = model(**inputs)
        sentence_embeddings.append(outputs.last_hidden_state[:, 0, :]).append(outputs.last_hidden_state[:, 0, :])

    return sentence_embeddings

# Example usage for DataFrame 'df' and column 'Text'
embeddings_list = df.head['Text'].apply(get_embeddings).tolist()
```

Questions?

Thank you!