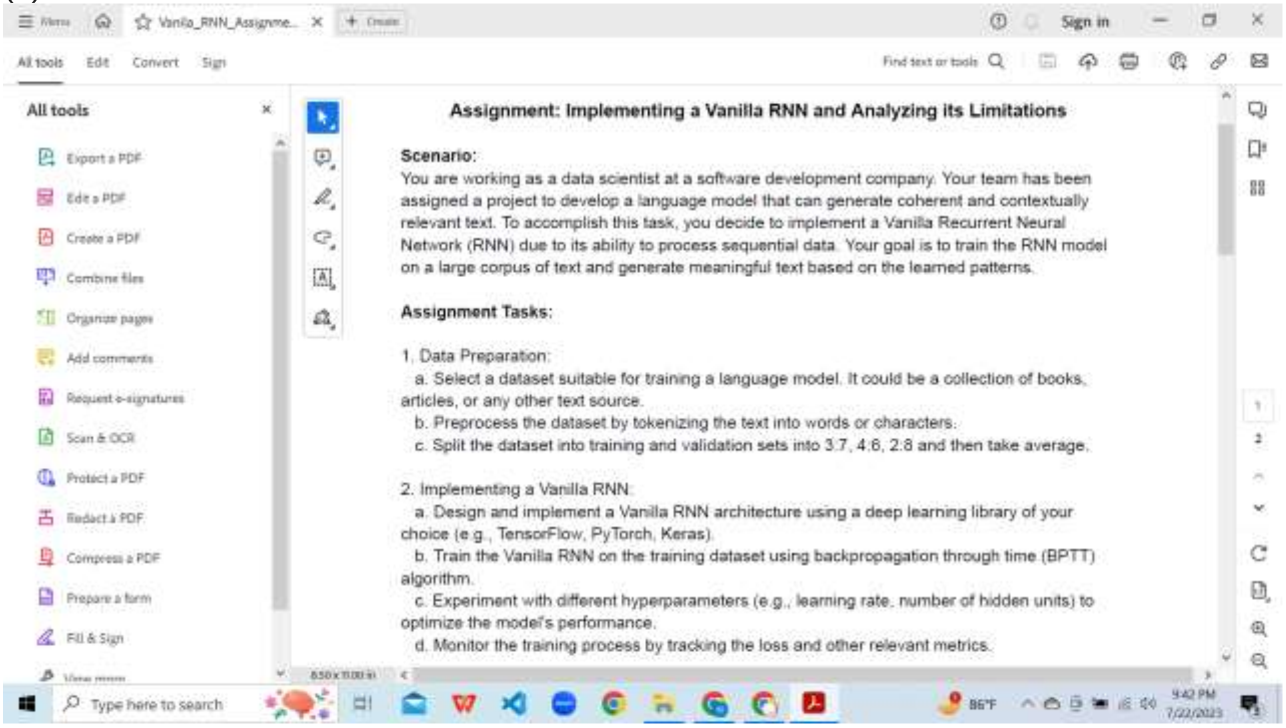
Question

(0)



Assignment: Implementing a Vanilla RNN and Analyzing its Limitations

Scenario:
You are working as a data scientist at a software development company. Your team has been assigned a project to develop a language model that can generate coherent and contextually relevant text. To accomplish this task, you decide to implement a Vanilla Recurrent Neural Network (RNN) due to its ability to process sequential data. Your goal is to train the RNN model on a large corpus of text and generate meaningful text based on the learned patterns.

Assignment Tasks:

1. Data Preparation:
 a. Select a dataset suitable for training a language model. It could be a collection of books, articles, or any other text source.
 b. Preprocess the dataset by tokenizing the text into words or characters.
 c. Split the dataset into training and validation sets into 3:7, 4:6, 2:8 and then take average.

2. Implementing a Vanilla RNN:
 a. Design and implement a Vanilla RNN architecture using a deep learning library of your choice (e.g., TensorFlow, PyTorch, Keras).
 b. Train the Vanilla RNN on the training dataset using backpropagation through time (BPTT) algorithm.
 c. Experiment with different hyperparameters (e.g., learning rate, number of hidden units) to optimize the model's performance.
 d. Monitor the training process by tracking the loss and other relevant metrics.

please can anyone help me to solve these ques. Thank you

Show transcribed data

Expert Answer

This solution was written by a subject matter expert. It's designed to help students like you learn core concepts.

Step-by-step

1st step
All steps
Answer only

Step 1/3

Data Preparation:

a. Select a Dataset:

For training a language model, you can choose a large corpus of text. This can be a collection of books, articles, Wikipedia pages, or any other text source. The larger and more diverse the dataset, the better the language model can learn patterns and generate coherent text.

b. Preprocess the Dataset:

Before feeding the data to the RNN, you need to preprocess it. Steps include:

- Convert the text to lowercase to ensure case insensitivity.
- Remove any special characters, numbers, or irrelevant symbols.
- Tokenize the text into words or characters to create a vocabulary.

c. Split the Dataset:
To train and evaluate the language model, split the dataset into training and validation sets. Common splits are 80% training and 20% validation or 70% training and 30% validation.
Explanation:
Implementing a Vanilla RNN: a. Design and Implement a Vanilla RNN: Choose a deep learning library like TensorFlow, PyTorch, or Keras to implement the Vanilla RNN. Here's a high-level structure of the Vanilla RNN:

```
import tensorflow as tf

class VanillaRNN(tf.keras.Model):
def __init__(self, vocab_size, embedding_dim, hidden_units):
super(VanillaRNN, self).__init__()
self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
self.rnn = tf.keras.layers.SimpleRNN(hidden_units,
return_sequences=True)
self.dense = tf.keras.layers.Dense(vocab_size, activation='softmax')

def call(self, inputs):
x = self.embedding(inputs)
x = self.rnn(x)
return self.dense(x)

# Example usage:
vocab_size = len(vocabulary)
embedding_dim = 128
hidden_units = 256
model = VanillaRNN(vocab_size, embedding_dim, hidden_units)
```

Step 2/3
b. Train the Vanilla RNN: Prepare the training loop using backpropagation through time (BPTT) algorithm:

```
# Assuming you have already preprocessed and tokenized data
train_dataset, val_dataset = prepare_datasets(train_data, val_data)
```

```
# Define loss function and optimizer
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()

# Function to calculate loss
def loss_function(real, predictions):
mask = tf.math.logical_not(tf.math.equal(real, 0))
loss_ = loss_object(real, predictions)

mask = tf.cast(mask, dtype=loss_.dtype)
loss_ *= mask

return tf.reduce_mean(loss_)

# Function for training step
@tf.function
def train_step(inputs, targets):
with tf.GradientTape() as tape:
predictions = model(inputs)
loss = loss_function(targets, predictions)

gradients = tape.gradient(loss, model.trainable_variables)
optimizer.apply_gradients(zip(gradients, model.trainable_variables))

return loss

# Training loop
epochs = 10
for epoch in range(epochs):
total_loss = 0.0
for batch, (inputs, targets) in enumerate(train_dataset):
batch_loss = train_step(inputs, targets)
total_loss += batch_loss

average_loss = total_loss / (batch + 1)

# Print training progress
print(f'Epoch {epoch + 1}, Loss: {average_loss:.4f}')
```

Step 3/3

c. Experiment with Different Hyperparameters: You can experiment with various hyperparameters such as learning rate, number of hidden units, embedding dimension, and batch size to optimize the model's performance. Use trial and error or more systematic hyperparameter search techniques like grid search or random search.

d. Monitor the Training Process: During training, monitor the loss and other relevant metrics (e.g., perplexity) on the validation set. This will help you evaluate the model's performance and detect issues like overfitting or underfitting.
Remember that the example provided above is a simplified version, and depending on the deep learning library you choose, the syntax and specific details might vary. The steps provided should give you a general idea of how to approach implementing a Vanilla RNN for text generation.

Final answer

Good luck with your assignment! If you encounter any specific issues or have further questions, feel free to ask.