Lab Sheet 4: Pixel RNNs

4.1 - Using Row LSTM to learn image data

Overview of PixelRNN

- PixelRNN is a type of generative model that uses recurrent neural networks (RNNs) to generate images pixel by pixel[1].
- It models the discrete probability of the raw pixel values and encodes the complete set of dependencies in the image[1].
- The model predicts the pixels in an image along the two spatial dimensions[1].
- Variants of PixelRNN include the Row LSTM and the Diagonal BiLSTM, which scale more easily to larger datasets[1].
- PixelRNNs use masked convolutions to model full dependencies between the color channels[1].

Overview of Row LSTMs

Row LSTM is a variant of the Long Short-Term Memory (LSTM) network, which is a type of Recurrent Neural Network (RNN). LSTM networks are designed to remember information for long periods of time, making them effective for tasks that involve sequential data[1].

In the context of Pixel Recurrent Neural Networks (PixelRNNs), Row LSTM plays a crucial role. PixelRNNs are generative models that sequentially predict the pixels in an image along the two spatial dimensions[2][3]. They model the discrete probability of the raw pixel values and encode the complete set of dependencies in the image[2][3].

The Row LSTM in PixelRNN processes the image row by row from top to bottom, computing features for a whole row at once[4]. This is performed with a

one-dimensional convolution[5]. The larger the size of the convolution kernel, the broader the context that is captured[5]. This allows the Row LSTM to capture triangular contexts in the image[5].

In summary, Row LSTM in PixelRNNs helps in capturing spatial dependencies in the data, enabling the model to generate realistic images and perform various image processing tasks[23]. It's an essential component in the architecture of PixelRNNs, contributing to their ability to model the complete set of dependencies in an image[23].

Image learning with CIFAR-10 Dataset using Row LSTMs

This code is learning to generate images pixel by pixel.

The CIFAR-10 dataset, which consists of 60,000 color images in 10 classes, is loaded and preprocessed. The pixel values of the images are normalized to be between 0 and 1, and the images are reshaped to a different dimension. The images are then shifted by one pixel to create the input and target data for the model.

The PixelRNN model is defined with an LSTM layer and a Dense layer. The LSTM layer is used to capture the dependencies between the pixels in the image, and the Dense layer is used to output the prediction for the next pixel.

The model is compiled with an Adam optimizer and a Sparse Categorical Cross Entropy loss function. It is then trained on the training data for a specified number of epochs, with the test data used for evaluation.

After training, the model is used to predict the first 10 images from the test set. The predictions are reshaped back to the original image dimensions and plotted for visualization. This allows us to see the images that the model has generated.

Import necessary libraries

```
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np
```

Define the PixelRNN model

```
class PixelRNN(tf.keras.Model):
    def __init__(self, hidden_dim=128, num_pixels=256):
        super(PixelRNN, self).__init__()
        # Define the hidden dimension and number of pixels
        self.hidden_dim = hidden_dim
        self.num_pixels = num_pixels

        # Define the LSTM layer with the specified hidden dimension
        self.lstm = layers.LSTM(hidden_dim, return_sequences=True)
        # Define the Dense layer with softmax activation function
        self.dense = layers.Dense(num_pixels)

    def call(self, x):
        # Pass the input through the LSTM layer
        x = self.lstm(x)
        # Pass the output of LSTM layer through the Dense layer
        return self.dense(x)
```

Load the CIFAR-10 dataset

```
(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.cifar10.load_data()
# Normalize the pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Reshape the data to (num_samples, 3072)

```
x_train = x_train.reshape((-1, 3072, 1))
x_test = x_test.reshape((-1, 3072, 1))
```

Shift the images by one pixel, so that for each pixel, the target is the next pixel

```
x_train_inputs = x_train[:, :-1]
x_train_targets = x_train[:, 1:]
x_test_inputs = x_test[:, :-1]
x_test_targets = x_test[:, 1:]
```

Create an instance of the model

```
model = PixelRNN()
```

Compile the model with Adam optimizer and Sparse Categorical Cross Entropy loss function

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Train the model for 5 epochs

```
model.fit(x_train_inputs, x_train_targets, epochs=5)
```

Evaluate the model on the test data

```
model.evaluate(x_test_inputs, x_test_targets)
```

Predict the first 10 images from the test set

```
predictions = model.predict(x_test_inputs[:10])
```

Convert the predictions to images

```
predicted_images = predictions.reshape((-1, 32, 32, 3))
```

Plot the predicted images

```
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(predicted_images[i], cmap='gray')
    plt.axis('off')
plt.show()
```

4.2 - Exercises

Comment on this particular implementation of row LSTMs (example: how it is similar/different to what you have learned in lectures, or how it is good or bad). Make observations about the training time of the model. If possible, try to make optimisations. (Result may or may not come accordingly) Give justifications for your optimisations / tuning. Make sure all your contributions are available on jupyter notebook!

4.3 - Using Diagonal BiLSTM to learn image data

Overview of Diagonal LSTM

- Diagonal LSTM is a variant of the Long Short-Term Memory (LSTM) network, which is an advanced version of recurrent neural network (RNN) architecture[2].
- LSTM networks are designed to model chronological sequences and their long-range dependencies more precisely than conventional RNNs[2].
- The Diagonal LSTM processes the data diagonally, starting from the top corner and reaching the opposite corner while moving in both directions.
- This allows the model to capture both the row-wise and column-wise dependencies in the data, making it particularly effective for image data.

Image Classification with CIFAR-10 Dataset using Diagonal BiLSTM

This following code is for training to classify images from the CIFAR-10 dataset. The CIFAR-10 dataset, which consists of 60,000 color images in 10 classes, is loaded and preprocessed. The pixel values of the images are normalized to be between 0 and 1, and the labels are converted to one-hot vectors.

The model architecture is defined using a combination of convolutional layers and a custom Diagonal BiLSTM layer. The convolutional layers are used to extract features from the images, while the Diagonal BiLSTM layer is used to capture spatial dependencies in the data.

The model is compiled with an Adam optimizer and a categorical cross-entropy loss function. It is then trained on the training data for a specified number of epochs, with the test data used for validation.

After training, the model's performance is evaluated by visualizing the first 25 images from the training dataset. The images are displayed in a 5x5 grid, with no ticks or grid lines, and using a binary color map. This visualization helps to understand the kind of images the model has been trained on.

Imports

```
import tensorflow as tf
from tensorflow.keras import datasets
```

Load CIFAR-10 dataset

```
(train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()
```

Normalize pixel values to be between 0 and 1

```
train_images, test_images = train_images / 255.0, test_images /
255.0
```

Convert labels to one-hot vectors

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

Define the Diagonal BiLSTM

```
class DiagonalBiLSTM(tf.keras.Model):
    def __init__(self, hidden_units):
        super(DiagonalBiLSTM, self).__init__()
        self.hidden_units = hidden_units
        self.lstm_layer = tf.keras.layers.LSTM(hidden_units,
return_sequences=True, return_state=True)
        self.reverse_lstm_layer =
tf.keras.layers.LSTM(hidden_units, return_sequences=True,
return_state=True, go_backwards=True)
```

```
    def call(self, inputs):
        output, h_state, c_state = self.lstm_layer(inputs)
        reverse_output, reverse_h_state, reverse_c_state =
self.reverse_lstm_layer(inputs)
        return output + reverse_output
```

Create a Diagonal BiLSTM model

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(32, 32, 3)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Reshape((-1, 128)),
    DiagonalBiLSTM(256),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10)
])
```

Compile the model

```
model.compile(optimizer='adam',

loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Train the model

```
model.fit(train_images, train_labels, epochs=10,
          validation_data=(test_images, test_labels))
```

Visualize the first 25 images from the training dataset

```
import matplotlib.pyplot as plt
```

```
# Visualize the first 25 images from the training dataset
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
plt.show()
```

4.4 - Exercises

Run the code above once as it is and run it again by making some optimisations either in the parameters or the model deck. Justify your changes. (Result may or may not come accordingly). Make sure these are mentioned in your jupyter notebook.