Advanced Machine Learning Assignment 2

Utpalraj Kemprai(MDS202352)

Aman (MDS202305)

Ankan Kar (MCS202303)

December 19, 2024

Task 1 : Spam Classification using RNN and LSTM models

Data

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged acording being ham (legitimate) or spam.

Data Preprocessing Steps

1. **Data Cleaning:**
   - **Drop Columns:** Removed irrelevant columns (Unnamed: 2, 3, 4).
   - **Rename Columns:** Renamed to "label" and "sms."
   - **Label Encoding:** Converted "ham" and "spam" to numerical values (0, 1).
2. **Missing Values:** No missing values detected; no action taken.
3. **Data Imbalance:** Checked for imbalance; oversampling applied to minority class (spam).
4. **Text Preprocessing:** Lowercasing, punctuation removal, stop word removal, lemmatization, tokenization, encoding, and padding.

Hyperparameters for the RNN and LSTM Model

Hyperparameter	**Value**
Number of embeddings	$vocab\_size + 2$
Embedding dimension	128
Number of Epochs	100
Hidden Layer dimension	64
Number of layers	2
Bidirectional	True
Dropout	0.2
Learning rate	0.001
Output dimension	1
Criterion	nn.BCELoss()

Comparison of Model Parameters - Number of Parameters: 1023425 (RNN), 1172417 (LSTM), i.e, the LSTM model has approximately 14.55% more parameters than the RNN model.

Training the models - Since this is a binary classification problem, we use binary cross-entropy as the loss function, and the optimizer chosen was Adam (since it works well for training RNNs).

Evaluation

Metric	RNN Model	LSTM Model
Accuracy	0.8780	0.9397
F1 Score	0.6614	0.7961
Precision	0.5076	0.6920
Recall	0.9486	0.9371

Conclusion

Based on our evaluation viz. Accuracy, F1 Score, Precsion and Recall, we see that the LSTM model performed relatively better than the RNN model for the task of Spam classification when both had similar number of model parameters and same hyperparameters.

Task 2: Text Generation using RNN and LSTM

The data used is the same as Task 1.

Data Preprocessing Steps

1. **Loading and Cleaning:**
   - Drop unnecessary columns (`'Unnamed: 2'`, `'Unnamed: 3'`, `'Unnamed: 4'`).
   - Rename remaining columns to `'label'` and `'sms'`.
   - Map `'ham'` and `'spam'` labels to numerical values (0 and 1).
2. **Splitting Messages:**
   - Define `split_message` function to divide each SMS into two halves, appending an `<END>` token.
   - Apply this function to create `'sms_first_half'` and `'sms_second_half'` columns.
3. **Creating Vocabulary and Encoding:**
   - Build a vocabulary using `defaultdict`, including special tokens `<PAD>` and `<END>`.
   - Define `encode_text` to convert text into numerical indices.
   - Encode `'sms_first_half'` and `'sms_second_half'` into:
     - `'first_half_encoded'`
     - `'second_half_encoded'`.
4. **Padding:**
   - In the `SMSSplitDataset` class, ensure SMS halves are truncated or padded to a fixed length (`max_len`) for batch processing, using 0s for `<PAD>`.

Hyperparameters for the RNN and LSTM model

Hyperparameter	**Value**
Number of embeddings	*vocab_size*
Embedding dimension	50
Number of Epochs	100
Hidden Layer dimension	128
Learning rate	0.001
Output dimension	*vocab_size*
Criterion	nn.CrosEntropyLoss()

Comparison of Model Parameters - Number of Parameters: 2813113 (RNN), 2882233 (LSTM), i.e, the LSTM model has approximately 2.46% more parameters than the RNN model.

Training the models - We use Cross Entropy Loss as the loss function, and the optimizer chosen was Adam (since it works well for training RNNs).

Conclusion

We generated the second half of sms for 20 examples from the data that was not used for training the RNN and LSTM. Based on this 20 examples we made the following observations:

The LSTM outperforms the RNN in text generation due to its stronger grasp of context and coherence:

1. **Contextual Accuracy**: LSTM predictions generally stay more relevant to the initial input, as seen in **Examples 5** and **8**. The LSTM constructs messages that better reflect the structure and style of realistic SMS responses.

2. **Phrase Construction**: LSTM-generated phrases are more conversational and plausible, like in **Examples 6** and **11**, where LSTM outputs align better with the expected conversational flow.

3. **Grammatical Structure**: The LSTM maintains coherence across phrases, avoiding random word jumps that appear in RNN outputs, such as in **Example 14**.

4. **Handling Dependencies**: In messages needing longer contextual retention (e.g., **Example 18**), the LSTM produces a more coherent, flowing response than the RNN.

Overall, LSTM delivers a clearer, context-aware output, making it better suited for sequence-based text generation.

Task 3 : Conditional GAN for Fashion MNIST

Data

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

Data Preprocessing Steps

1. We separated the pixel and label data for Fashion MNIST
2. We convert the image pixels and the image labels into PyTorch tensors
3. We normalize all pixel values to be in the range [-1,1]

Model

Hyperparameters

We choose the following as our hyperparameters:

latent_dim = 100 (Dimensionality of the noise vector)
num_classes = 10 (Number of classes in Fashion MNIST)
image_size = 28 * 28 (Flattened size of a 28x28 image)
batch_size = 64
epochs = 50
img_channels = 1
learning_rate = 0.0002

Architecture

- **Generator**
  Generator((label_embedding): Embedding(10, 10) (model): Sequential((0): ConvTranspose2d(110, 128, kernel_size=(7, 7), stride=(1, 1), bias=False)

  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

  (2): ReLU()

  (3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

  (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

  (5): ReLU()

  (6): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

  (7): Tanh()))

- **Discriminator**
  Discriminator((label_embedding): Embedding(10, 10) (model): Sequential((0): Conv2d(11, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

  (1): LeakyReLU(negative_slope=0.2)

  (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

  (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

  (4): LeakyReLU(negative_slope=0.2)

  (5): Conv2d(128, 1, kernel_size=(7, 7), stride=(1, 1), bias=False) (6): Sigmoid()

))

Training

Loss function : Binary Cross Entropy ; Optimization algorithm : ADAM

Conclusion

We observed that by the end of the training the loss values of the discriminator and the generator had stabilized around 0.7 and 0.8 respectively, showing that we have trained the model enough. This explains the decent quality of generated images of each label.