Using LSTM Networks to Caption Images on the COCO Dataset

Allen Zeng*
University of California, San Diego
azeng@ucsd.edu

Shubham Chaudhary*
University of California, San Diego
shchaudh@ucsd.edu

Tara Mirmira*
University of California, San Diego
tmirmira@ucsd.edu

Zonglin Di*
University of California, San Diego
zodi@ucsd.edu

Abstract

This report explores the use of Long Short-Term Memory units (LSTMs) and "vanilla" recurrent neural networks for the task of image captioning on the Common Objects in Context (COCO) dataset. The network encoder is comprised on a pre-trained ResNet50 model, with all but the last fully connected layer frozen, and the decoder is our RNN model. We compare the performance of the RNN model using LSTM cells and using the traditional RNN cell. Our best LSTM model, in terms of Cross Entropy Loss, achieves a Bleu1 score of 59.1944 and Bleu4 score of 6.9435 using deterministic caption prediction. Our best vanilla RNN model, in terms of Cross Entropy Loss, achieves a Bleu1 score of 58.8344 and Bleu4 score of 7.3822 using deterministic caption prediction. Comparing the LSTM model and RNN model as we vary the model hyperparameters however, we conclude that the LSTM models generally perform better than the RNN models.

1 Introduction

Recurrent neural networks with Long Short-Term Memory units (colloquially named LSTM networks) have been used to tackle previously-difficult problems involving sequential data, such as handwriting recognition, handwriting generation, language modeling, language translation, acoustic modeling and analysis, speech recognition and synthesis, video data labeling, and more [1, 2]. Here, we attempt to use a LSTM network as part of a larger network to perform image captioning. The problem of image captioning involves taking an image as input, and as output generating the appropriate caption that best describes the input image. Specifically for our model, we use a pretrained convolutional neural network, ResNet [3], as an image encoder and a newly trained LSTM network as the decoder to generate captions. We train and test our model on a subset of the Common Objects in Context (COCO) [4] dataset.

We quantify the performance of our image captioning network using the BLEU-1 and BLEU-4 scores [5]. After setting a baseline LSTM network, we vary the architecture and observe its effect on the model performance. We also compare the methods of deterministic and stochastic caption generation [6]. Additionally, we compare the performance of our LSTM model with a "vanilla" recurrent neural network (RNN) [7] that does not use LSTM cells. Finally, we show examples image caption generation using our best performing network.

*For equal contribution

2 Related Work

This report primarily explores the use of a CNN as the encoder and a LSTM network as the decoder for the task of image captioning. We compare our baseline model to the "vanilla" recurrent neural network architecture [7]. The vanilla RNN model has been shown to struggle learning with longer sequences of data due to the "vanishing" and "exploding" gradient problem [8]. In contrast, LSTM networks are a special kind of RNN capable of learning long-term dependencies [9]. A LSTM cell maintains a cell state, and contains three gates with parameters that can be learned through gradient descent: input, output, and forget. These gates work together to modify the cell state as an input sequence is fed to the network. The LSTM architecture deals with the vanishing and exploding gradient problem since the derivative of the cell state with respect to the previous cell state is an additive function of the previous cell state, candidate cell state, forget gate, and input gate [9]. Being able to learn the gate parameters, the network is able to preserve gradients through backpropagation through time.

There are also variations and alternatives to the LSTM architecture. One LSTM variant implements "peephole" connections that feed the cell state as inputs to the gating functions [10]. The *Gated Recurrent Unit* (GRU), combines the forget and input gates into a single *update* gate, and combines the traditional LSTM's cell and hidden states into a single state [11]. The GRU unit is similarly performant as the LSTM unit, and both appear to be popular in modern RNNs [1]. There are also *Depth-Gated Recurrent Neural Networks* [12] which introduce a depth-gate between multiple layers of LSTM cells, introducing a linear dependence through network depth. The *Clockwork* RNN introduces a modification that enables better performance on much longer sequences than traditional LSTMs, by partitioning the hidden layer into separate modules each processing inputs at its own temporal granularity [13]. This has been shown to perform well in audio signal generation and spoken word classification contexts. Recently, RNNs have been extended to networks known as *Neural Turing Machines* (NTM), which can be trained to infer simple algorithms such as copying, sorting, and associative recall [14]. These NTM contain a neural network controller and memory bank. The controller interacts with the memory bank selectively using read and write operations, which are differentiable and can be trained with gradient descent.

We used Pytorch documentation to help implement the project [15], [16], [17], [18].

3 Methods

3.1 Dataset

For all our experiments we have used a subset of Common Objects in Context (COCO)[4] dataset. COCO is a large-scale object detection, segmentation, and captioning dataset. It contains images of complex everyday scenes containing common objects in their natural context. We have used $1/5^{th}$ of the original dataset and trained different models for captioning task on around 82k images and tested on 3K images. All these images on an average had 5 captions each. We have also used data augmentation to add perturbation to the images so as to improve the robustness of the model. We applied vertical flip, horizontal flip, and color jitter as the data augmentation methods. 25% of the training data would be flipped horizontally randomly, 25% would be flipped vertically randomly, 25% would have a random color jitter applied, and 25 % would be unchanged.

3.2 Model

The input images are processed by an Encoder to extract features. We have used a pre-trained ResNet50 [3] model to encode the images into a high-dimensional feature vector. For this, we have replaced the last layer with a trainable fully connected linear layer. The output of this linear layer is then used as the initial hidden state for vanilla RNN and as initial hidden as well as cell state for LSTM models. All the experiments were performed on images of size 256×256.

3.2.1 Baseline LSTM

In our baseline model, we use a LSTM unit as the decoder which generates image captions. The baseline model uses a hidden and cell size of 512, and an embedding size of 300. (Later, we

vary the hidden and cell sizes between $[256, 512, 1024]$, and we vary the embedding size between $[150, 300, 600]$.)

Long Short-Term Memory (LSTM) is a variant of recurrent neural networks designed to tackle the problem of vanishing gradients in vanilla recurrent neural networks. LSTMs make use of an extra cell state and several information gates per unit to allow gradients to propagate over longer time spans. Each LSTM unit has an input gate i, a forget gate f, an output gate o and a cell gate g. At every time step t, the input arguments to the LSTM cell are input embedding x_t, previous hidden state h_{t-1} and previous cell state c_{t-1}. The values of hidden units h_t and cell state c_t are computed using following equations:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

In order to convert the vocabulary representation from one hot vectors, to a lower dimensional dense embedding, we use Pytorch's `nn.Embedding` module which is essentially a trainable linear layer. We also train a linear layer to transform the hidden units at each time step into output vectors. Hidden and cell states of LSTM are initialized with the image features extracted by the Encoder. To apply "Teacher forcing", at each time step the elements of teacher sequence are used as inputs, rather than the output generated at previous time step. We have padded the sequences with zero vectors to make them of equal length. Padding shorter sequences allows us to stack several sequences into a batch. The output vectors generated for the entire sequences are then compared with the sequence of one-hot teacher vectors for loss computation.

3.2.2 Vanilla RNN

To contrast with our baseline model, we also train a model using a recurrent neural network (RNN) as the decoder which generates image captions. We also use the same pre-trained ResNet50 model as the encoder in this case. The baseline model uses a hidden size of 512, and an embedding size of 300. (Later, we again vary the hidden and cell sizes between $[256, 512, 1024]$, and we vary the embedding size between $[150, 300, 600]$.)

Recurrent neural networks are used to extract features from a temporal sequence. The recurrent unit consists of a hidden state h_t which is a function of previous hidden state h_{t-1} and current input x_t. Its value is calculated as follows:

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}).$$

Using a single recurrent unit allows weight sharing across time and the recurrent nature make inputs of any length acceptable. However, since gradient descent is a linear operation, sharing of weights may leads to vanishing or exploding gradients. Because of diminishing gradients, RNNs also suffer from the inability of accessing information across longer time steps. The weights can be clamped to an upper limit to prevent exploding gradients and LSTMs are one of the solution to handle vanishing gradients. Similar to Baseline LSTM, we use teacher forcing while training and also use dense embeddings for elements of input sequences. We transform the hidden units at each time step to generate the output sequence. For each sequence the hidden unit was initialize with the image embedding obtained from image encoder.

3.2.3 LSTM Architecture Tuning

After implementing and training the baseline model we tune certain hyperparameters to obtain other LSTM models. We vary the size of the hidden and cell states between values of $[256, 512, 1024]$. We vary the size of the embedding between values of $[150, 300, 600]$. After determining the best values for those parameters, we go on to vary the depth of the LSTM network. We train with augmented data, and compare networks with the number of LSTM layers varying between $[1, 2, 4, 8]$.

4 Results

4.1 Experiment Details

We used ResNet 50 as the encoder, but we modified the last linear layer to output the desired number hidden units needed for the LSTM or RNN portion. We froze all the ResNet 50 layers except the last modified linear layer. All the experiments are worked on NVIDIA RTX 3090 GPUs. The batch size we used is 256 and the epoch number is 30. Because we found that the model would converge very soon if the learning rate keeps unchanged. We added a step learning rate decay schedule. The baseline learning rate is 5e-4 and the learning rate would be decreased by 10 every 10 epoch. The temperature parameter we used is 0.1 in the baseline experiments for generating stochastic captions. We have again used *CrossEntropyLoss* to train all of our networks.

4.2 Evaluation Metrics

We used bilingual evaluation understudy (BLEU) [5] as the evaluation methods based on `nltk` package. Bleu scores calculate the fraction of n-gram matches between two sentences. The Bleu1 score describes the number of unigram (1-gram) matches, while the Bleu4 score describes the number of 4-gram matches. Some sources measure Bleu4 as the average of the 1-, 2-, 3-, and 4-gram scores, while others measure Bleu4 as strictly the fraction of 4-gram matches [5]. We use the latter measure, where Bleu4 is only measures the 4-gram matches. So Bleu1 corresponds to using `nltk.translate.bleu_score` [19] with weight vector $(1, 0, 0, 0)$ and Bleu4 with weight vector $(0, 0, 0, 1)$. The base values for BLEU range from 0.0 to 1.0; we multiply our scores by a factor of 100 to get the measure as a percentage value. If two sentences match perfectly BLEU is 1.0, while if two mismatch perfectly BLEU is 0.0.

Table 1: Results of Hyper-parameter Tuning for Vanilla RNN Model without Data Augmentation

Hidden Size	Embedding Size	Validation Loss	Test Loss	Bleu1 Deterministic	Bleu4 Deterministic
256	150	1.2786	1.3008	56.6966	6.1420
256	300	1.2778	1.2531	56.0443	6.0619
256	600	1.2802	1.2286	58.3764	6.4959
512	150	1.2431	1.2626	58.1298	6.9027
512	300	1.2239	1.2254	58.1700	6.7462
512	600	1.2063	1.1981	58.2640	6.7139
1024	150	1.2173	1.2348	58.8367	7.3727
1024	300	1.2596	1.1599	59.8390	7.3591
1024	600	1.1992	1.2125	58.8344	7.3822

Table 2: Results of Hyper-parameter Tuning for Vanilla RNN Model with Data Augmentation

Hidden Size	Embedding Size	Validation Loss	Test Loss	Bleu1 Deterministic	Bleu4 Deterministic
256	150	1.3043	1.2875	55.6604	5.5932
256	300	1.2746	1.2739	55.3447	5.9179
256	600	1.2649	1.2749	55.2117	57.661
512	150	1.2420	1.2557	57.9461	6.6935
512	300	1.2264	1.2199	57.9434	6.9494
512	600	1.2223	1.2063	57.4518	6.7088
1024	150	1.2374	1.2491	57.8575	6.9893
1024	300	1.2005	1.2015	59.1972	7.3662
1024	600	1.2160	1.2135	58.3672	7.2116

Table 3: Results of Hyperparameter Tuning for LSTM Model without Data Augmentation

Hidden Size	Embedding Size	Validation Loss	Test Loss	Bleu1 Deterministic	Bleu4 Deterministic
256	150	1.2470	1.2420	58.8971	6.6569
256	300	1.2259	1.1932	59.0182	6.7751
256	600	1.2260	1.2314	58.9181	6.7059
512	150	1.1910	1.2081	60.0300	7.5807
512	300	1.1910	1.1916	59.5088	7.0866
512	600	1.1906	1.1836	58.8441	6.8050
1024	150	1.1860	1.2027	59.9681	7.3763
1024	300	1.1753	1.1599	59.8390	7.3591
1024	600	1.1759	1.1756	59.0019	6.8472

Table 4: Results of Hyperparameter Tuning for LSTM Model with Data Augmentation

Hidden Size	Embedding Size	Validation Loss	Test Loss	Bleu1 Deterministic	Bleu4 Deterministic
256	150	1.2620	1.2722	58.3898	6.2081
256	300	1.2266	1.5103	58.8288	6.7725
256	600	1.2062	1.1930	58.6421	6.8104
512	150	1.1897	1.2118	59.7438	7.3035
512	300	1.1640	1.1857	58.7043	7.2178
512	600	1.1629	1.1881	58.8855	7.2214
1024	150	1.1875	1.1795	59.1067	7.3453
1024	300	1.1717	1.1722	59.4415	7.5066
1024	600	1.1614	1.1598	59.1944	6.9435

Table 5: Results of Different Temperatures for Stochastic Caption Generation on Baseline LSTM Model with Data Augmentation

Hidden Size	Embedding Size	Temperature	Bleu1	Bleu4
512	300	0 (deterministic)	58.7484	6.5495
512	300	0.01	58.7373	6.5301
512	300	0.05	58.6906	6.4488
512	300	0.1	58.5464	6.3506
512	300	0.2	57.9249	5.9082
512	300	0.7	51.7972	3.0961
512	300	1	43.7927	1.9544
512	300	1.5	25.5945	1.2952
512	300	2	11.8250	0.9175

Table 6: Results of Training with Stacked LSTM Architectures

Hidden Size	Embedding Size	Number of Layers	Validation Loss	Test Loss	Bleu1 Deterministic	Bleu4 Deterministic
1024	300	1	1.1784	1.1685	59.6287	7.4048
1024	300	2	1.8810	1.1807	59.7081	7.1926
1024	300	4	1.2180	1.2277	59.0189	6.9198
1024	300	8	1.9946	2.0021	49.9164	3.1018

5 Discussion

First, we compare the results of training the baseline LSTM model and the vanilla RNN model. Then we examine their performance in deterministic caption generation. For the baseline model, we also examine the effects of stochastic caption generation. We then discuss the hyperparameter tuning of the LSTM model. Finally, we show examples of images and their predicted captions.

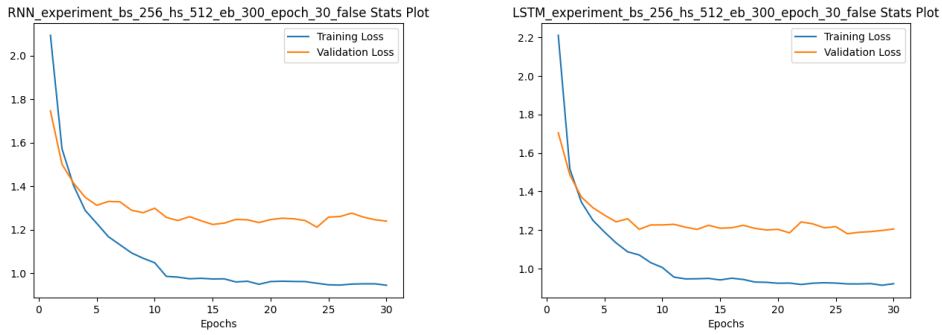
5.1 Comparison of the LSTM Models and Vanilla RNN Models

The baseline LSTM and vanilla RNN models were both trained with hidden size of 512, and an embedding size of 300. Without data augmentation, the baseline LSTM model outperformed the RNN model in both Bleu1 and Bleu4 (deterministic) scores. This is expected, as the literature indicates that LSTM models generally outperform vanilla RNN models on tasks involving long sequences of data [9, 1, 8], such as image captioning here. With and without data augmentation, the LSTM model consistently out-performs the vanilla RNN model in both Bleu1 and Bleu4 scores. This is also expected based on prior literature. However, it is also interesting to examine the compare the models with themselves, with and without data augmentation.

With data augmentation, the baseline vanilla RNN model achieved a marginally lower Bleu1 score and marginally higher Bleu4 score compared to without data augmentation. However, looking over all variations in hidden unit size and embedding size, the RNN model trained with data augmentation generally has lower Bleu1 and Bleu4 scores than the RNN model trained without data augmentation. This suggests that our augmentation methods were not appropriate for this specific dataset. For example, we include a color jitter augmentation and 25% change vertical flip augmentation. Since the COCO [4] dataset is namely about Common Objects "in Context," perhaps these augmentations negatively affect the context of the image, leading to poorer results. In other words, these augmentations move the distribution of training data further away from the distribution of testing data. However, we hypothesize that these augmentations may help generalizing the RNN model to perform well on non-COCO datasets.

In contrast, the LSTM models trained with data augmentation consistently out-performs the LSTM models trained without data augmentation in both Bleu1 and Bleu4 scores. This suggests that the data augmentation methods did help the LSTM model generalize to the test set. Taking the fact that data augmentation made the vanilla RNN model worse but made the LSTM model better, this suggests that the vanilla RNN models are more prone to over-fitting on the training data than the LSTM models. To conclude, the LSTM models performs better than the vanilla RNN models and are less likely to overfit to the training set.

We have presented the behavior of training and validation loss with epochs in Figure 1. We observed that the training curve corresponding to LSTM consistently remains lower than that of RNN, but even after 30 epochs the RNN model does not converge the loss of LSTM.

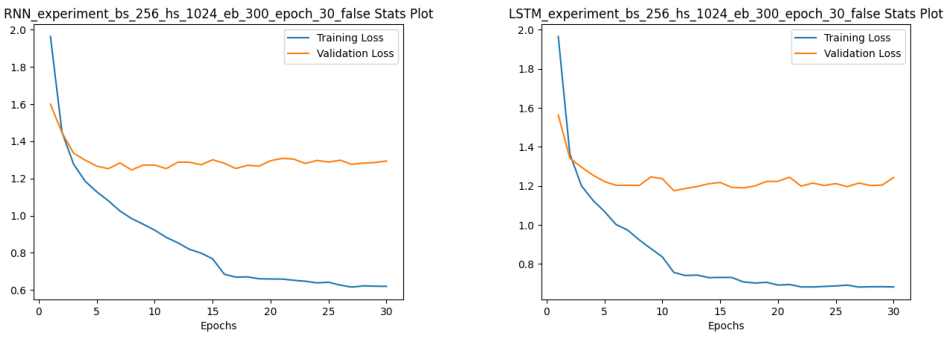


(a) (b)

Figure 1: Training vs validation loss for baseline models

5.1.1 Fine-tuning Embedding Size and Hidden State Size

In all our experiments we have used ResNet50 to extract features from input images. The image embedding obtained from the encoder are then used to initialize the hidden state and cell state of the recurrent unit. Thus the size of the hidden state is tied to the size of image embeddings. We have also transformed the elements of input sequences from one-hot vectors to dense low dimensional embeddings. These word embeddings are learned during the training of the model. The size of both the hidden layer of recurrent unit as well as the size of word embeddings can affect the performance of the task. In this section we present the results obtained from varying the size of Hidden State and the size of word embeddings. The train and test statistics with different hyper-parameters for RNN, RNN with data augmentation, LSTM and LSTM with data augmentation can be found in Table no. 1, 2, 3 and 4 respectively.
For a fixed Embedding size, we observed that increasing the size of Hidden State leads to lower test loss. The size of hidden state affects the model in 2 ways, it determines the size of the vector into which the image are compressed by the encoder and also determines the amount of information that the recurrent unit would carry. The amount of compression during the encoding is very high since we are transforming the images of size 256x256 into embeddings of size 256, 512 or 1024, essentially into a few rows. Therefore the gain in performance can be explained by the fact that increasing the size of hidden layer implies the availability of more image features when encoding as well as availability of more information over longer time steps during decoding.
For a fixed Hidden size, we experimented with the vocabulary embedding sizes of 150, 300 and 600. We observed that the best performance was achieved with embeddings of size 300 or 600. This led us to conclude that word embeddings of size 150 are small and some semantic information is lost in this compression. Past literature also suggests that irrespective of the size of vocabulary, the gain in semantic information is marginal when the size of embeddings is larger than 300.
All these experiments were performed using the learning rate of 5e-4. We observed that for RNN, the lowest test loss was achieved without data augmentation, using Hidden size 1024 and embedding size 300. The behavior of validation and training loss for this model are presented in Figure2(a). For LSTM, the lowest test loss was achieved with data augmentation, using Hidden size 1024 and embedding size 600 which was almost equal to the test loss achieved without data augmentation, using 1024 hidden units and 300 embedding size. The behavior of validation and training loss for the later model(no data augmentation) are presented in Figure2(b).



(a) (b)

Figure 2: Training vs validation loss

5.1.2 Best Models with Deterministic Approach

The best LSTM model (lowest validation loss) was achieved with hidden unit size 1024, embedding size 600, and training with data augmentation. While this model achieved the lowest validation and test loss, it has lower bleu1 and bleu4 scores than the model with embedding size 300 instead. The bleu scores can be found in Table 3. This discrepancy can be explained by Bleu1 and Bleu4 being computationally efficient, but are imperfect metrics of human language sentence similarity [5]. Although Bleu scores generally correlate with human judgement of sentences, it is not guaranteed that increased Bleu scores mean more accurate sentences.

The best vanilla RNN model (lowest validation loss) was achieved with hidden unit size 1024, embedding size 600, and training without data augmentation. The bleu scores can be found in Table 1. In this case, the Bleu1 score is lower than the vanilla RNN model with embedding size 300, while the Bleu4 score is marginally higher. This difference can again be explained by Bleu scores being an imperfect metric.

5.2 Evaluation of Caption Generation

In this subsection, we evaluate the performance of deterministic caption generation. We compare the performance of deterministic caption generation with that of stochastic caption prediction.

5.2.1 Baseline LSTM with Stochastic Approach

For our baseline LSTM model, the hidden size was 512 and the embedding size was 300. We used a learning rate scheduler during training and a starting learning rate of 5e-4. We used this model to conduct an experiment to compare the performance of deterministic caption generation with the performance of stochastic caption generation. We varied the temperature for stochastic caption generation and calculated the Bleu1 and Bleu4 evaluation metrics for the stochastically predicted captions. The results of this experiment can be found in Table 5. We tried temperature values of 0 (deterministic), $0.01, 0.05, 0.1, 0.2, 0.7, 1, 1.5, 2$ Deterministic caption generation outperformed stochastic caption for all the temperature values we tried. To understand why this behavior is seen, we looked at an example of the captions generated. For a temperature value of 1.5, we saw the following example:

Reference Captions:

- two skiers sliding down the hill on the pathway
- two snow skiers are going down a trail by trees
- two people skiing downhill in the woods together
- two skiers going through a snowy forest at daytime
- two people in the middle of a skiing trail with trees lined on each side of the trail

Stochastically Generated Predicted Caption:

- fast snowy parked skis as a snowy covered slope

Deterministically Generated Predicted Caption:

- a person riding skis down a snow covered slope

We can see that the sampling that occurs in generating the predicted caption stochastically tends to hurt the quality of the prediction. The sampling process for stochastic generation probabilistically picks words that differ from the deterministic caption. In some situations, the alternative words chosen by the sampling process negatively affect the prediction quality as seen in this example. We see the trend of decreasing Bleu1 and Bleu4 scores as temperature increases. This trend can be explained by the increased stochasticity as the temperature increases. Deterministic caption generation picks the word that corresponds to the maximum value of a softmax layer over the output of the LSTM. Stochastic caption generation samples words from the probability distribution defined by a softmax layer (weighted by temperature) over the output of the LSTM. As the temperature increases, stochastic caption generation is more likely to pick alternate words from the deterministic caption. This would help if the alternate words were sufficiently synonymous with the words chosen by the deterministic caption. With our experimental results and above example, we found that this did not appear to be the case for the given task of caption prediction with an LSTM on the COCO dataset.

5.2.2 Fine-tuning Number of Layers in Stacked LSTM Model

Here we experimented with stacked LSTM architectures. We use a fixed hidden state size of 1024 and embedding size of 300, and then stacked the LSTM units in the decoder, varying the number of

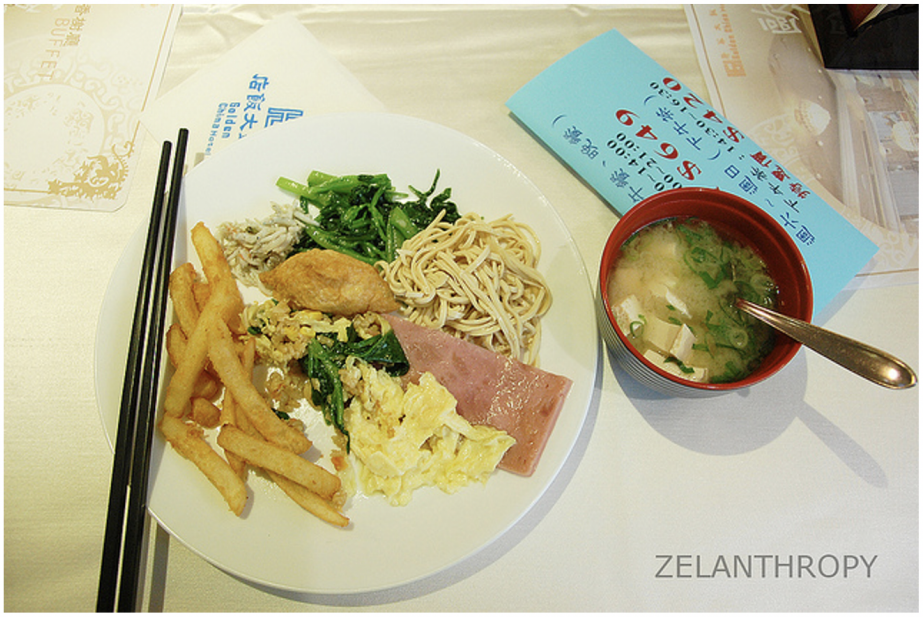
layers between $[1, 2, 4, 8]$. (With 1 layer, the model has the same architecture as before in Section 5.1.1.) The results of this is shown in Table 6.

We observed that the performance of the network decreases as the number of layers increases, in terms of validation loss, test loss, and Bleu scores. The best model was the one with only one LSTM layer. This suggests that having stacked LSTM units introduces too much complexity, and perhaps the size of our dataset is not sufficiently large to train the deeper models. Also, this worse performance may be explained by our training captions being too short or simple. Stacked models, such as [12, 2], are typically used for tasks such as machine translation where there is a large corpus of text, longer sequences of input data, and more complex inter-dependencies of the data over time. In our case, we only seek to predict a short sentence given an image.

5.3 Example Captions

In the following figures, we show examples reference captions and the corresponding predicted caption using our best LSTM model.

5.3.1 Good Caption Examples




```
REFERENCE CAPTIONS
a white plate topped with different types of food
a resturant serves a variety of asian food
a plate of food with chop sticks and a cup of soup
the plate with a pair of chop sticks on it is next to a bowl with a spoon in
it
the plate is holding a variety of food along with chopsticks

PREDICTED CAPTION
a table with a plate of food and a drink
```

Figure 3: Caption Example 1

REFERENCE CAPTIONS
a baseball player holding a bat next to a base
a photo of a baseball game where a guy is at home plate taking a pitch
a baseball player attempts to hit a baseball while a catcher and umpire watch
people on a field playing baseball and other watching
a baseball player holding a bat and a catcher squatting behind him

PREDICTED CAPTION
a group of men are playing baseball in a field

Figure 4: Caption Example 2

REFERENCE CAPTIONS
a woman is hitting the ball with a tennis racket on the tennis court
two women and one is playing tennis and is wearing orange
a woman holding a tennis racquet with a ball coming in her direction
a tennis player turns her racket sideways as she returns the ball
an official watching as a woman tennis player tries to return the ball

PREDICTED CAPTION
a woman in a white shirt and a shorts playing tennis

Figure 5: Caption Example 3

REFERENCE CAPTIONS
a picture of a coffee cup and a smart phone on a table
a cup is sitting on a surface next to a cell phone
a filtered photo of a phone and coffee mug is shown
a cup sitting next to a smart phone
a very fuzzy image of a cell phone and a cup

PREDICTED CAPTION
a close up of a cell phone and a mouse

Figure 6: Caption Example 4

REFERENCE CAPTIONS
a left handed baseball player swinging a bat in front of a catcher and umpire
a baseball player swinging a bat at a ball
a man hitting a baseball in a professional baseball game
baseball player hitting a ball with a baseball bat
a baseball batter trying to hit a baseball

PREDICTED CAPTION
a baseball player is swinging a bat at a ball

Figure 7: Caption Example 5

5.3.2 Bad Caption Examples

REFERENCE CAPTIONS
a half eaten sandwich sitting on a plate on a table
a man points at his companions plate as he dines on the patio on a sunny day
a plate that has a sandwich on it next to a drink
a man sits at a table in front of a plate of food
two sandwhiches sitting on a plate with tooth picks in them

PREDICTED CAPTION
a man and a woman are sitting at a table with a plate of food

Figure 8: Caption Example 6

REFERENCE CAPTIONS
a couple of cows sitting inside a wooded fence
there is old folklore that states that if the cows lay down it will rain
a couple of cows are laying in a pin
some cows lying down on the ground in their fenced off area
a small outdoor enclosure is holding a group of cows

PREDICTED CAPTION
a train is sitting on the tracks near a fence

Figure 9: Caption Example 7

REFERENCE CAPTIONS
a carved bear that has a ribbon around the neck
a wood carved bear is sitting on a table
a statue of a bear wearing a red ribbon
a teddy bear that appears to be made out of wood
the large bear is made up of clay

PREDICTED CAPTION
a man is standing in to a giant elephant

Figure 10: Caption Example 8

REFERENCE CAPTIONS
a woman in plaid shirt talking on a cellphone
a woman sitting in a chair talking on a cell phone
a woman sitting in a chair talking on a cell phone
a lady with a big tag hanging on her neck using a phone
woman talking on a cell phone in a lobby

PREDICTED CAPTION
a man sitting at a table with a glass of wine

Figure 11: Caption Example 9

REFERENCE CAPTIONS
a man is plowing the field in preparation
a tractor that is driving over some farmland
a tractor on a field and a airplane in the sky
a tractor works on a farm next to a road
a farmland with a vehicle working on the land

PREDICTED CAPTION
a large long line of birds on a beach

Figure 12: Caption Example 10

Individual Contributions

Tara Mirmira: I worked on the encoder, decoder, train/validation/test functions, caption generation, and data augmentation. I helped with the running of experiments and writing of the report.

Zonglin Di: I finished almost all the experiments from running to results. I helped with the curve plotting, writing some of the code, writing the report and part of discussion.

Allen Zeng: I contributed to writing and debugging the LSTM model, and helped run several of the experiments. I also contributed to writing various sections of the report, and discussion of the results.

Shubham Chaudhary: Added implementation for RNN, helped in running experiments and contributed in writing several sections of report.

References

[1] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct 2017.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[5] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[6] UCSD CSE 251B Staff. Programming assignment 4: Image captioning. *CSE 251B: Neural Networks for Pattern Recognition*, Winter 2021, 2021.

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. *Tech. rep. ICS 8504. San Diego, California: Institute for Cognitive Science*, Sept 1985.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[10] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194 vol.3, 2000.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[12] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated lstm, 2015.

[13] Jan Koutník, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. A clockwork rnn, 2014.

[14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.

[15] `https://pytorch.org/docs/stable/generated/torch.nn.RNN.html`.

[16] `https://pytorch.org/docs/master/generated/torch.nn.LSTM.html`.

[17] `https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html`.

[18] https://pytorch.org/vision/0.8/models.html.

[19] Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python.* O'Reilly Media Inc, 2009.