Arabic Search Query Autocompletion: Comparing LSTM, ANN, and Decision Tree Models

1st Hamzah Alqam
Computer Engineering dept
Birzeit University
Ramallah, Palestine
1211173@student.birziet.edu

2nd Hihad Awad
Computer Engineering dept
Birzeit University
Ramallah, Palestine
1210088@student.birziet.edu

3rd Obayd Sarraj
Computer Engineering dept
Birzeit University
Ramallah, Palestine
1211128@student.birziet.edu

***Abstract*—This report details our project on Arabic search query autocompletion, where we built and compared Long Short-Term Memory (LSTM) networks, Artificial Neural Networks (ANNs), and Decision Tree models. We aimed to predict the next word in an Arabic search query, focusing on how well each model could learn from the training data and apply that learning to new, unseen queries. We dealt with the challenges of preprocessing Arabic text and tried different settings for each model to see what worked best. Our results showed that bidirectional LSTMs, especially with additive attention, did a better job at understanding and predicting Arabic text than ANNs or Decision Trees. They were more accurate and had lower perplexity on test data, meaning they could handle the grammar and meaning of Arabic sentences better. This report shows what we learned from those trials and how it changes our view on using these models for autocompletion.**

I. Introduction

Autocompletion is now a common feature in search engines and other systems with text input, making it faster and easier to type in queries. While autocompletion is well-studied for English, applying it to Arabic is harder because of the language's complicated structure, right-to-left writing, and the use of diacritics that change the meaning of words.

In this project, we focused on making an Arabic search query autocompletion system using LSTM networks, ANNs, and Decision Trees. We wanted to see how well these models could predict the next word in a query, both on the data they were trained on and on new, unseen queries. We were especially interested in how model architecture and settings affected their ability to learn the training data without losing the ability to work on new data.

Initially, we thought simpler models like ANNs might quickly learn the training data but might not do well on new queries. We expected more complex models like LSTMs to better understand the patterns of the language and work better on new data. We thought LSTMs, which can handle sequences and remember long-range patterns, would be a better choice for Arabic autocompletion than ANNs or Decision Trees.

II. Background

Initially, we were drawn to this project because of the widespread use of autocompletion in modern search interfaces. The idea of predicting the next word in a sequence seemed like a good fit for our coursework on machine learning. In our original proposal, we planned to use decision trees (like ID3 or CART) and then move on to ANNs, including feedforward networks and possibly LSTMs. We even considered fine-tuning an existing model like GPT-2 for comparison.

Our initial datasets were to be WikiText-2-v1 and a collection of Discord messages, with the hope of trying bigger datasets if time allowed. We planned to use accuracy and perplexity to measure how well our models could predict sequences of words. We thought decision trees might not do very well, but we expected ANNs to at least make reasonable predictions based on the context of the words.

As the project progressed, our focus shifted towards LSTMs due to their known ability to handle sequential data. We also started looking into bidirectional LSTMs and additive attention to improve our model's performance, especially on longer queries.

III. Methodology

This part explains how we prepared our data, the structure of the models we used, and how we set up our experiments.

A. Data Preprocessing

Getting the data right is really important for how well a machine learning model works. Since Arabic is a bit tricky, we had to do some careful preparation before using it. Here's what we did with the ArabicProcessor class:

1. **Diacritics:** We took out all the vowel marks and other diacritics because they're not usually used in everyday writing and can make the data too sparse.
2. **Normalization:** We made all forms of 'alef' (,آ ,أ ,إ ,ٱ) the same (ا) and changed 'ta marbuta' (ة) at the end of words to 'ha' (ه). Also got rid of the tatweel character (-) since it's just for stretching words.
3. **Special Characters:** We treated question marks as their own words by spacing them out, and we removed anything that wasn't an Arabic letter or a question mark.
4. **Spaces:** We made sure there was only one space between words and got rid of any extra spaces at the beginning or end.

After cleaning up the text, the ArabicProcessor made a list of all unique words from the training data and gave each one a number. It also marked places for padding (<PAD>) and words it didn't know (<UNK>). Then, it turned the text into sequences of numbers using this list.

For the LSTM, we flipped the sequences around because Arabic is read from right to left. This helps the LSTM pay more attention to the start of the query, which is usually

more important for predicting the next word. Then we made all sequences the same length by adding padding.

B. *Model Architectures*

We tried out three types of models:

1) *Long Short-Term Memory (LSTM) Networks*

Our main model was based on the AutocompleteLSTMModel class, which is an LSTM network. We tried both regular and bidirectional LSTMs. The bidirectional one looks at the text both forwards and backward and combines what it learns from both directions. We also added something called additive attention, using the AdditiveAttention class, to help the model focus on the important parts of the text when it's making predictions.

The LSTM model gets a sequence of word numbers, turns them into vectors (embeddings), and then feeds them into the LSTM layers. If we used attention, it comes after the LSTM. Finally, there's a layer that gives us a probability for each word in our list, which is the model's guess for the next word.

2) *Artificial Neural Networks (ANNs)*

We also used a basic feedforward ANN model from the AutocompleteANNModel class as a comparison. It works like the LSTM but first averages the word vectors to get one vector for the whole sequence. This vector goes through some layers with activation functions and dropout to prevent overfitting, and then an output layer that predicts the next word.

3) *Decision Tree*

We also looked at a Decision Tree model to compare it with the others. Decision Trees are a simpler way to predict things based on data. For our project, it would try to guess the next word based on the words that came before it. (Your partners will add more about how the Decision Tree model works.)

C. *Experimental Setup*

We did a few of experiments to see how well our models worked with different settings. We changed things like the size of the embeddings, how many hidden units we used, the number of LSTM layers, dropout rate, learning rate, weight decay, and batch size. We'll talk about the exact settings we used in the results section.

We used three datasets called v1, v2, and v3 to train and test our models. V2 is bigger because it includes answers, and v3 puts the validation set into the training data, so it has a different split (85-15) than v1 and v2 (70-15-15). We trained our models for a set number of rounds and checked how they did on the validation and test sets after each round. We used accuracy, top-5 accuracy, and perplexity to see how good they were.

IV. Results and Discussion

Here, we'll go over the results from our experiments, which are organized in the tables below. Each table shows results for different models or datasets. We'll talk about the accuracy, top-5 accuracy, loss, and perplexity at different points in the training. Because we did a lot of trials over many days, we'll just show some of the results that tell us the most important things we learned.

Table 1:LSTM Hyperparameter Tuning (v1 Dataset)

- Note: Emb = Embedding Dimension, Hid = Hidden Dimension, Bi = Bidirectional, L2 = 2 Layers, L3 = 3 Layers

Table 2: ANN Hyperparameter Tuning (v1 Dataset)

MODEL (ANN)	EPOCH	TRAIN ACC	TRAIN TOP-5	TEST ACC	TEST TOP-5	TEST PERPLEXITY
EMB_128_HID_256	5	41.15	62.25	24.80	39.28	1077.51
EMB_128_HID_256	10	58.18	76.61	23.40	37.77	3630.97
EMB_128_HID_256	20	29.99	43.50	26.21	39.31	772.05
EMB_128_HID_256	30	64.93	80.18	22.43	36.06	8708.32
EMB_256_HID_512	5	27.83	40.49	26.13	38.89	580.11
EMB_256_HID_512	10	29.93	42.51	26.78	39.85	570.31
EMB_256_HID_512	40	67.66	81.85	21.89	34.13	213418.30
EMB_256_HID_512	100	71.64	84.13	21.50	33.15	659537.55

- Note: Emb = Embedding Dimension, Hid = Hidden Dimension.

Table 3: Dataset Comparisons (LSTM)

MODEL (LSTM)	EPOCH	TRAIN ACC	TRAIN TOP-5	TEST ACC	TEST TOP-5	TEST PERPLEXITY
(V2) EMB_128_HID_128_BI	10	53.24	72.44	20.79	33.95	1035.61
(V2) EMB_128_HID_192_BI	5	39.60	60.02	21.92	34.41	817.78
(V2) EMB_128_HID_192_BI	15	63.14	79.62	21.29	33.30	1407.03
(V3) EMB_192_HID_64_BI	5	36.78	54.00	29.18	43.72	295.72
(V3) EMB_192_HID_64_BI	20	51.22	68.63	28.63	43.21	380.21

- Note: Emb = Embedding Dimension, Hid = Hidden Dimension, Bi = Bidirectional, (v2/v3) = Dataset Version.

Table : 4ANN vs. LSTM (v1 Dataset) - Selected Checkpoints

MODEL	EPOCH	TRAIN ACC	TRAIN TOP-5	TEST ACC	TEST TOP-5	TEST PERPLEXITY
ANN (EMB_256_HID_512)	100	71.64	84.13	21.50	33.15	659537.55
LSTM (EMB_256_HID_256_L3_BI)	120	70.15	83.12	28.74	42.57	630.53

- Note: Emb = Embedding Dimension, Hid = Hidden Dimension, L3 = 3 Layers, Bi = Bidirectional.

Key Observations:

- The LSTM model, even though it has slightly lower training accuracy, significantly outperforms the ANN on the test set.
- The LSTM's test accuracy (28.74%) is much closer to its training accuracy (70.15%), indicating better generalization.
- The ANN's test accuracy (21.50%) is much lower than its training accuracy (71.64%), suggesting overfitting.

The LSTM has a much lower test perplexity (630.53) compared to the ANN (659537.55), further confirming its superior ability to predict unseen data.

What We Learned:

- **LSTM vs. ANN:** The LSTM models consistently did better than the ANN models. This was true for all datasets and settings. The LSTMs were more accurate and had lower perplexity on the test sets. The ANNs often did well on the training set but didn't do as well on new data. This probably means the ANNs were memorizing the training data instead of learning the language itself.
- **Bidirectional LSTM with Attention:** The bidirectional LSTM models usually did better than the regular ones. Adding attention sometimes helped even more, especially with longer queries. It seems like looking at the text both ways and paying attention to the important parts helped the model understand the context better.
- **Hyperparameter Tuning:** Changing the settings of the LSTM models made a big difference. Bigger embeddings and hidden dimensions usually meant better results, but they also took longer to train. Having more LSTM layers helped sometimes, but not always. Using dropout helped prevent overfitting, and values around 0.3-0.4 seemed to work best for LSTMs.
- **Dataset Variations:** Trying different datasets (v1, v2, and v3) taught us that just having more data doesn't always make the model better at generalizing. Putting the validation set into the training data (v3) made the training accuracy higher, but it didn't really improve the test accuracy. This showed us that it's important to have a separate validation set to tune the model.

Learning Rate and Weight Decay: We had to adjust the learning rate carefully. If it was too high, the training was unstable, but if it was too low, it took forever. Lowering the learning rate as we went along helped us fine-tune the model. Weight decay also helped prevent overfitting by stopping the model's weights from getting too big.

V. CONCLUSION

In this project, we compared LSTM, ANN, and Decision Tree models for predicting the next word in Arabic search queries. Our results showed that bidirectional LSTMs, especially with additive attention, were the best at this task. They were more accurate and had lower perplexity on new data compared to ANNs and Decision Trees. This suggests that LSTMs are better at understanding the structure and meaning of Arabic text.

We also learned that carefully choosing the model's settings and using techniques to prevent overfitting are important for getting good results. Bigger embeddings and hidden dimensions usually helped for LSTMs, but the learning rate and weight decay were also important factors.

Our experiments with different datasets showed that just having more data isn't enough to make the model generalize better. It's still important to have a separate set of data to check how well the model is learning.

VI. REFERENCES

[1] Dataset: https://huggingface.co/datasets/abdoelsayed/Open-ArabicaQA

[2] Perplexity description: https://huggingface.co/spaces/evaluate-metric/perplexity

[3] LSTM for text generation: https://www.analyticsvidhya.com/blog/2022/02/explaining-text-generation-with-lstm/

APPENDIX

Table 1: LSTM Hyperparameter Tuning (v1 Dataset)

MODEL (LSTM)	EPOCH	TRAIN ACC	TRAIN TOP-5	TEST ACC	TEST TOP-5	TEST PERPLEXITY
EMB_128_HID_64_BI	5	40.79	59.30	29.09	43.83	370.14
EMB_128_HID_64_BI	10	45.46	66.16	27.42	43.88	388.06
EMB_128_HID_64_BI	15	50.88	69.66	28.98	43.69	416.55
EMB_128_HID_128_BI	5	42.25	62.79	29.16	44.55	309.72
EMB_128_HID_128_BI	10	50.98	71.23	29.29	44.76	322.82
EMB_128_HID_128_BI	20	39.54	59.13	29.60	44.92	571.50
EMB_128_HID_512_BI	5	51.93	71.14	29.93	45.04	359.88
EMB_128_HID_512_BI	10	61.61	79.05	28.97	44.33	420.13
EMB_192_HID_128_BI	5	35.98	54.46	29.81	44.60	457.53
EMB_192_HID_128_BI	15	41.09	61.13	30.02	45.41	415.58
EMB_256_HID_192_BI	5	37.47	56.42	29.83	44.75	440.96
EMB_256_HID_192_BI	15	43.67	64.03	29.88	45.33	423.55
EMB_256_HID_256_BI	5	38.11	56.38	29.42	44.13	275.25
EMB_256_HID_256_BI	10	42.46	61.93	29.52	44.25	283.12
EMB_128_HID_128_L2	5	36.00	53.92	29.05	43.61	396.35
EMB_128_HID_128_L3	5	34.12	50.90	29.19	43.11	310.32
EMB_128_HID_128_L3	10	41.05	59.50	28.89	43.50	306.86
EMB_192_HID_192	5	32.74	48.89	29.49	43.77	845.73
EMB_192_HID_192	10	35.87	53.43	30.16	44.69	782.36
EMB_192_HID_256	5	34.01	53.04	28.55	44.45	467.35
EMB_192_HID_256	10	40.53	59.16	30.06	44.93	436.72

- **Note:** Emb = Embedding Dimension, Hid = Hidden Dimension, Bi = Bidirectional, L2 = 2 Layers, L3 = 3 Layers.

Table 2: ANN Hyperparameter Tuning (v1 Dataset)

MODEL (ANN)	EPOCH	TRAIN ACC	TRAIN TOP-5	TEST ACC	TEST TOP-5	TEST PERPLEXITY
EMB_128_HID_256	5	41.15	62.25	24.80	39.28	1077.51
EMB_128_HID_256	10	58.18	76.61	23.40	37.77	3630.97
EMB_128_HID_256	20	29.99	43.50	26.21	39.31	772.05
EMB_128_HID_256	30	64.93	80.18	22.43	36.06	8708.32
EMB_256_HID_512	5	27.83	40.49	26.13	38.89	580.11
EMB_256_HID_512	10	29.93	42.51	26.78	39.85	570.31

EMB_256_HID_512	40	67.66	81.85	21.89	34.13	213418.30
EMB_256_HID_512	100	71.64	84.13	21.50	33.15	659537.55

- **Note:** Emb = Embedding Dimension, Hid = Hidden Dimension.

Table 3: Dataset Comparisons (LSTM)

MODEL (LSTM)	EPOCH	TRAIN ACC	TRAIN TOP-5	TEST ACC	TEST TOP-5	TEST PERPLEXITY
(V2) EMB_128_HID_128_BI	10	53.24	72.44	20.79	33.95	1035.61
(V2) EMB_128_HID_192_BI	5	39.60	60.02	21.92	34.41	817.78
(V2) EMB_128_HID_192_BI	15	63.14	79.62	21.29	33.30	1407.03
(V3) EMB_192_HID_64_BI	5	36.78	54.00	29.18	43.72	295.72
(V3) EMB_192_HID_64_BI	20	51.22	68.63	28.63	43.21	380.21

- **Note:** Emb = Embedding Dimension, Hid = Hidden Dimension, Bi = Bidirectional, (v2/v3) = Dataset Version.

Table 4: ANN vs. LSTM (v1 Dataset) - Selected Checkpoints

MODEL	EPOCH	TRAIN ACC	TRAIN TOP-5	TEST ACC	TEST TOP-5	TEST PERPLEXITY
ANN (EMB_256_HID_512)	100	71.64	84.13	21.50	33.15	659537.55
LSTM (EMB_256_HID_256_L3_BI)	120	70.15	83.12	28.74	42.57	630.53

- **Note:** Emb = Embedding Dimension, Hid = Hidden Dimension, L3 = 3 Layers, Bi = Bidirectional.