

**Department of Electrical and Electronics Engineering**

**EEE 443: Neural Networks**

*Class Project VII Report*

**Name and Surname:** Ali Aral Takak

**Student ID:** 22001758

**Department:** EEE

**Instructor:** Erdem Koyuncu

**Introduction**

In the seventh project of the course, we have implemented and evaluated a character-level long short-term memory (LSTM) network for name generation. Beginning from a dataset of names, we first preprocessed each name into fixed-length sequences of one-hot encoded characters. Then, we proceeded by training an LSTM model to predict, at each time step, the probability distribution over the next character in the sequence. Finally, we demonstrated the model’s generative capability by sampling 20 names for an input initial character, using temperature-controlled softmax sampling.

**Architecture Breakdown**

**a) Input Representation**

Given a raw name of length , we define a padded sequence as We then map each character to a one-hot vector:

Thus, each training input is of the form:

The target at time is the next character similarly one-hot encoded as:

**b) LSTM Sequence Model**

For the name generation task, we use a single-layer Long Short-Term Memory (LSTM) network with hidden dimension of size . Denote the hidden state and cell state at time as, where the recurrence is for :

Where are the input, forget, and output gates, respectively; and is the cell-input transform. The weight matrices are learned parameters.

**c) Output Projection and Softmax**

At each time step , the LSTM hidden vector is fed through a linear layer to produce unnormalized log-probabilities over the next character, which is mathematically formulated as:

We proceed by converting these log-probabilities into a probability distribution by using the softmax function:

**d) Loss Function**

Training minimizes the average cross-entropy loss over all time steps and all examples. For a single example of length , the loss is:

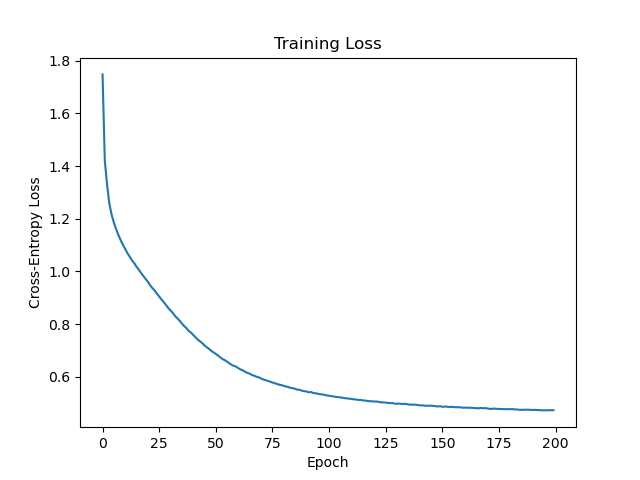
Hence, for a dataset of names, our goal is to minimize:

**e) Inference Module**

In the inference stage, we treat the trained LSTM network as a fixed mapping from a sequence of past characters to a probability distribution over the next character. First, the user supplies a single initial letter . We convert this letter—together with placeholder end-of-name tokens—to a fixed-length index vector of length . Each position of is then one-hot encoded into a vector in , producing an input tensor . This tensor is fed through the LSTM in evaluation mode: at each time step , the model updates its hidden state and cell state according to the standard LSTM equations. The hidden state is then projected by a learned linear layer to produce unnormalized logits . We apply a temperature-controlled softmax to yielding a probability vector over the 27 possible next characters. To generate the next character, we sample an index from the categorical distribution defined by . If the sampled index corresponds to the token, generation stops; otherwise, the corresponding letter is appended to the output name and the index vector is updated at position . We then proceed to the next time step, re-encoding the updated and passing it again through the LSTM. By repeating this process up to steps, we obtain one complete generated name. Because the sampling is stochastic, each run produces a different sequence even when starting from the same initial letter. In practice, we perform this procedure 20 times to produce a set of 20 candidate names. Throughout inference, no gradients are computed, and the model remains in evaluation mode.

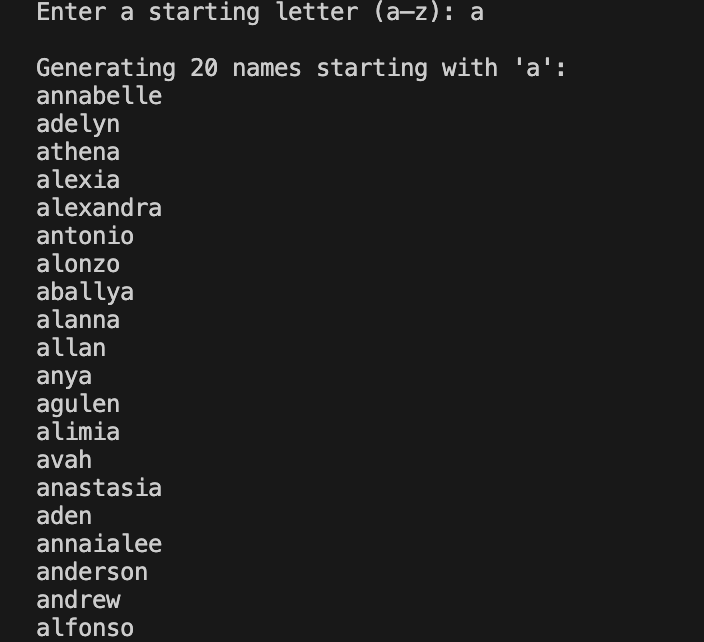
**Results**

It is safe to say that our LSTM network has been a successful implementation, generating names by using the letters of the English alphabet. The figure given below displays the loss per epoch curve, for 200 epochs:



**Figure 1:** Training loss per epoch for our LSTM implementation.

During the inference, the network is set to generate 20 names for the given input character. The figures given below display several examples:



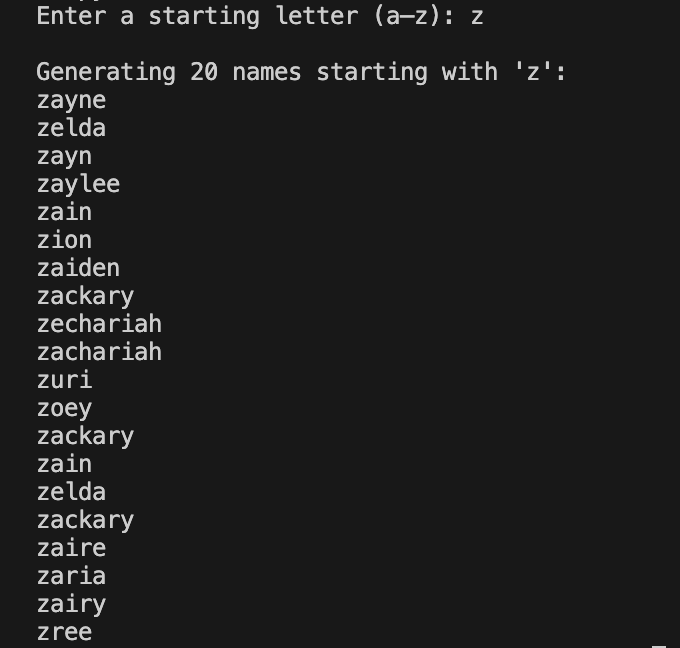
**Figure 2:** Network-generated names starting with "a".



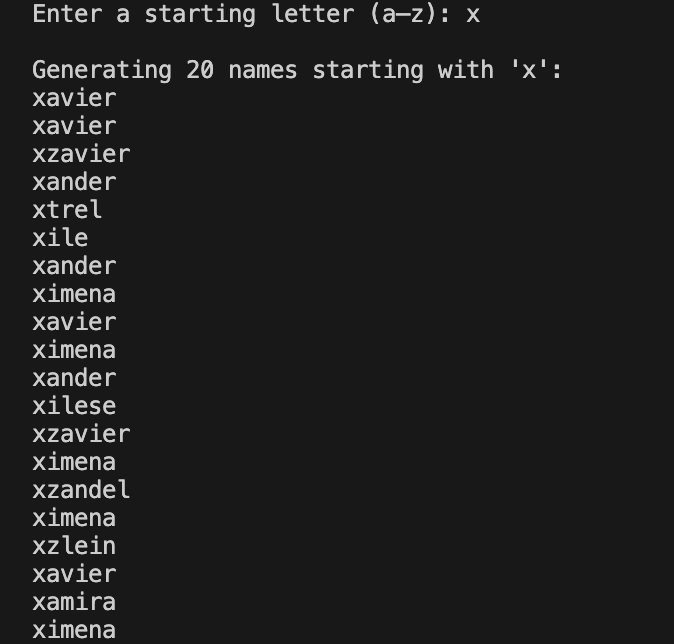
**Figure 3:** Network-generated names starting with "g".



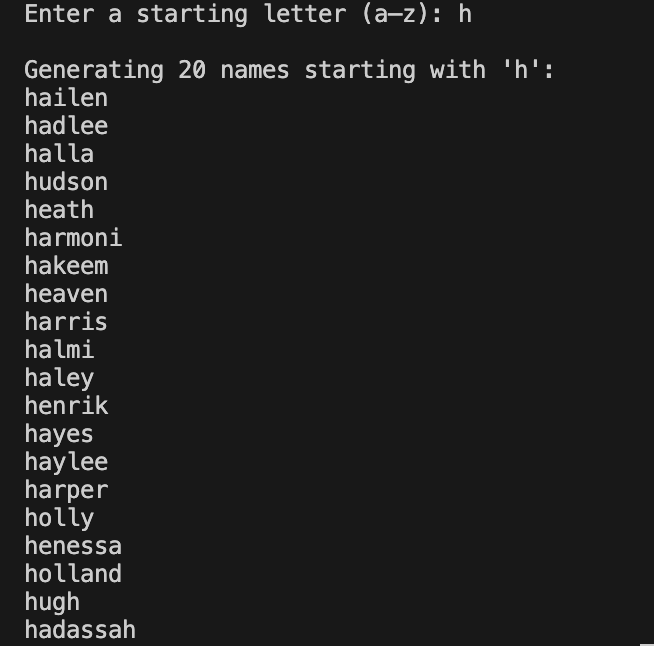
**Figure 4:** Network-generated names starting with "f".



**Figure 5:** Network-generated names starting with "z".



**Figure 6:** Network-generated names starting with "x".



**Figure 7:** Network-generated names starting with "h".

It was observed that for some letters, such as x or z, in which names are not that common, repetition was present. By adjusting temperature value, we tried to handle it, however, no significant change was observed in the means of producing valid and logical names. In addition, nonsense strings were more frequent, hence we left the value at .

**Conclusion**

In this project, we set out to build a neural network that could learn from a set of English given names and then generate new, realistic names one character at a time. To do so, we framed each name as a fixed-length sequence of one-hot encoded letters (including a special end-of-name symbol) and trained a single-layer LSTM with 128 hidden units to predict the next character in the sequence. Over 200 epochs the network reliably drove down its cross-entropy loss and captured the underlying patterns of letter combinations in English names. At inference, we fed the model an initial letter and, by sampling from its output distribution, produced twenty distinct name candidates for each seed. It can be said that our implementation is succesful, due to its capability of generating novel and common names altogether.