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 $w = (c^{(1)}, c^{(2)}, \dots, c^{(L)})$ of length $L \leq 11$, we define a padded sequence as $\tilde{c}^{(t)} = c^{(t)}\ if\ 1 \leq t \leq L, else\ \tilde{c}^{(t)} =\ EON$. We then map each character $\tilde{c}^{(t)}$ to a one-hot vector:

$$x^{(t)} \epsilon \{0,1\}^V$$

$$x_i^{(t)} = \begin{cases} 1\ if\ i = index\left(\tilde{c}^{(t)}\right), \\ 0\ otherwise \end{cases}$$

Thus, each training input is of the form:

$$X = \left[x^{(1)}, x^{(2)}, \dots, x^{(T)}\right] \epsilon\ \mathbb{R}^{T \times V}$$

The target at time t is the next character $\tilde{c}^{(t+1)}$, similarly one-hot encoded as:

$$y^{(t)} \epsilon \{0,1\}^V$$

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 $H = 128$. Denote the hidden state and cell state at time t as, where the recurrence is for $t = 1, \ldots, T$:

$$h^{(t)}, c^{(t)} \epsilon \mathbb{R}^H$$

$$\begin{matrix} i^{(t)} \\ f^{(t)} \\ o^{(t)} \\ g^{(t)} \end{matrix} = Wx^{(t)} + Uh^{(t-1)} + b$$

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \tanh\left(g^{(t)}\right), h^{(t)} o^{(t)} = \tanh \odot\left(c^{(t)}\right)$$

Where $i, f, o \epsilon \mathbb{R}^H$ are the input, forget, and output gates, respectively; and $g \epsilon \mathbb{R}^H$ is the cell-input transform. The weight matrices $W \epsilon R^{4H \times V}, U \epsilon R^{4H \times V},\ b \epsilon R^{4H}$ are learned parameters.

c) Output Projection and Softmax

At each time step t, the LSTM hidden vector $h^{(t)}$ is fed through a linear layer to produce unnormalized log-probabilities over the next character, which is mathematically formulated as:

$$z^{(t)} = W_{out} h^{(t)} + b_{out}$$

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

$$\widehat{y_\iota^{(t)}} = \frac{\exp\left(z_i^{(t)}\right)}{\sum_{j=1}^{V} \exp\left(z_j^{(t)}\right)}$$

d) Loss Function

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

$$L = -\frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{V} y_i^{(t)} \log\left(\widehat{y_\iota^{(t)}}\right)$$

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

$$\frac{1}{N}\sum_{n=1}^{N} L^{(n)}$$

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 $c^{(1)}$. We convert this letter—together with placeholder end-of-name tokens—to a fixed-length index vector $s \epsilon \{0, \ldots, 26\}^{T_{max}}$ of length $T_{max} = 20$. Each position of s is then one-hot encoded into a vector in $\mathbb{R}^{27}$, producing an input tensor $X \epsilon \mathbb{R}^{1\times T_{max}\times 27}$. This tensor is fed through the LSTM in evaluation mode: at each time step t, the model updates its hidden state $h^{(t)} \epsilon \mathbb{R}^{128}$ and cell state $c^{(t)} \epsilon \mathbb{R}^{128}$according to the standard LSTM equations. The hidden state is then projected by a learned linear layer to produce unnormalized logits $z^{(t)} \epsilon \mathbb{R}^{27}$. We apply a temperature-controlled softmax to $z^{(t)}$ yielding a probability vector $p^{(t)}$ over the 27 possible next characters. To generate the next character, we sample an index from the categorical distribution defined by $p^{(t)}$. If the sampled index corresponds to the $< EON >$ token, generation stops; otherwise, the corresponding letter is appended to the output name and the index vector s is updated at position $t + 1$. We then proceed to the next time step, re-encoding the updated s and passing it again through the LSTM. By repeating this process up to T_{max} 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:

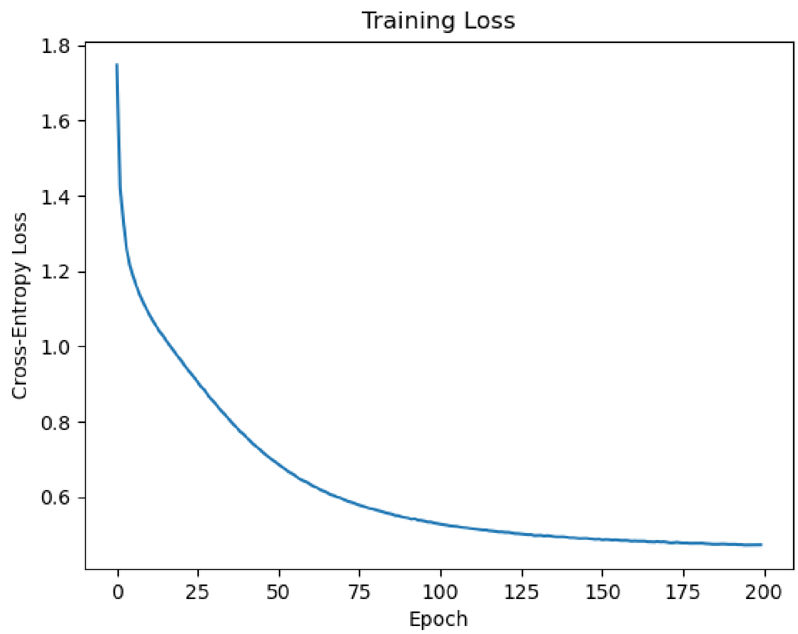

Figure 1: *Training loss per epoch for our LSTM implementation.*

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

```
Enter a starting letter (a–z): a

Generating 20 names starting with 'a':
annabelle
adelyn
athena
alexia
alexandra
antonio
alonzo
aballya
alanna
allan
anya
agulen
alimia
avah
anastasia
aden
annaialee
anderson
andrew
alfonso
```

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

```
Enter a starting letter (a–z): g

Generating 20 names starting with 'g':
gunnar
gunner
gennid
giselle
greta
gimenice
gia
gage
gauge
gabriel
genevieve
gus
gianna
grany
griffin
gideon
gannon
gemma
guillermo
guga
```

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

```
Enter a starting letter (a–z): f

Generating 20 names starting with 'f':
franklin
felike
frank
finnegan
frederick
frederick
fabi
foprick
fabian
frank
frankie
franklin
froddy
frankie
franklin
francis
felipe
francesca
forrest
felix
```

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

```
Enter a starting letter (a–z): z

Generating 20 names starting with 'z':
zayne
zelda
zayn
zaylee
zain
zion
zaiden
zackary
zechariah
zachariah
zuri
zoey
zackary
zain
zelda
zackary
zaire
zaria
zairy
zree
```

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

```
Enter a starting letter (a–z): x

Generating 20 names starting with 'x':
xavier
xavier
xzavier
xander
xtrel
xile
xander
ximena
xavier
ximena
xander
xilese
xzavier
ximena
xzandel
ximena
xzlein
xavier
xamira
ximena
```

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

```
Enter a starting letter (a–z): h

Generating 20 names starting with 'h':
hailen
hadlee
halla
hudson
heath
harmoni
hakeem
heaven
harris
halmi
haley
henrik
hayes
haylee
harper
holly
henessa
holland
hugh
hadassah
```

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 1.0.

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.