Generating Python with Character Level LSTMs

Ethan Frank
Case Western Reserve University
edf32@case.edu

April 24, 2022

Abstract

This paper explores Long Short Term Memory (LSTM) networks as a means for generating text. Using TensorFlow and Keras, we analyze the most common approach to this task and suggest an improvement using stateful Recurrent Networks. The results show a significant increase in efficiency over previous methods. We demonstrate our new method on two datasets: Python code and the Case Western Reserve University (CWRU) course catalog.

1 Introduction

LSTM networks are a variant of Recurrent Neural Networks (RNNs) and are extremely adept at processing data represented as a sequence. This includes numerical data, music, text, and video [1].

An RNN layer takes in inputs, produces outputs, and has a hidden state. At every step, an input is fed in and outputs are produced. Importantly, the hidden state from the previous time step is used when calculating the current output, giving the RNN memory. The basic RNN has vanishing gradients issues, so the LSTM is a modification with a more complicated update step, however the base concept is the same [2].

The importance of both long and short term memory can be seen in Python. Short term memory is required to spell words like if and *else* correctly, while long term memory helps remember that every opening parenthesis must eventually be closed.

RNNs are trained using backpropagation through time (BPT). Input sequences are fed to the network one at a time. The inputs, outputs, and hidden state are then unrolled to produce a flat [$time\_steps \times dimensions$] matrix, where the gradients can flow backwards.

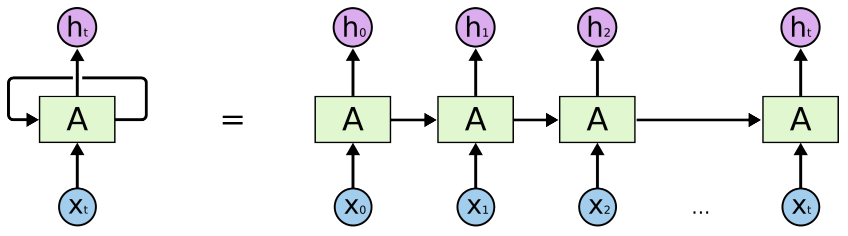


Figure 1: **An unrolled recurrent neural network.** By treating the RNN as a flat network, vanilla gradient descent algorithms can be used for backpropagation through time [3].

2 Background

2.1 Data Preprocessing

To train an RNN, we feed the network a sequence of characters and ask it to predict the next most likely character. The input data is a collection of text stored in a .txt file. The most commonly used method is to split the text into *sequence_len* sized input samples, where the output is the next character after that sequence. The input data is stepped over with size *step* to reduce overlap between training examples [4]. For example, the training data $[abcdefghi]$, with $step = 2$ and $sequence\_len = 4$ would be split into 3 training samples of:

$$X = \left[\begin{array}{cccc} a & b & c & d \\ \hline c & d & e & f \\ \hline e & f & g & h \end{array}\right]$$

$$Y = \left[\begin{array}{c} e \\ \hline g \\ \hline i \end{array}\right]$$

The characters are one-hot-encoded into a feature vector with length equal to the number of unique characters in the training data. The resulting training data has dimensions of $[samples, sequence\_len, num\_chars]$ ($[3, 4, 9]$ in the example) and the targets have dimension $[samples, 1, num\_chars]$.

The data is further processed to remove non-ascii characters. Some examples of characters found in Python source code include $\sum$, $\grave{a}$, and $\&nbsp$. Find & replace is used to replace these with similar-looking characters or an empty string.

The text is optionally converted to lowercase [4], reducing the output dimension of the number of characters the network has to predict.

3 Network Architecture

The network architecture consists of 2 or 3 stacked LSTM layers on top of each other, followed by a Dense layer with softmax activation. The LSTM layers can have any number of units and the Dense layer has *num_chars* units. Each LSTM layer is followed by a Dropout layer to prevent overfitting. Because LSTMs expect a sequence as input, every LSTM layer except the last needs *return_sequences* set to *True*. This tells the layer to return its output at every time step combined into a $[sequence\_len \times units]$ matrix. In Keras, a model with $units = 64, sequence\_len = 40$, $num\_chars = 53$ is defined as:

```
model = keras.Sequential(
    [
        keras.Input(shape=(sequence_len, num_chars)),
        layers.LSTM(64, return_sequences=True),
        layers.Dropout(0.5),
        layers.LSTM(64),
        layers.Dropout(0.5),
        layers.Dense(num_chars, activation="softmax")
    ]
)

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 40, 64)            30208
_________________________________________________________________
dropout (Dropout)            (None, 40, 64)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                33024
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 53)                3445
=================================================================
Total params: 66,677
Trainable params: 66,677
Non-trainable params: 0
```

4 Stateless vs Stateful RNNs

All RNN layers in Keras have a $stateful$ argument, which when set to $True$ allows the LSTM layer to remember its internal state between batches. A stateless RNN's internal state is reset between batches, so each sample of *sequence_len* characters is completely independent. There is no way for gradients to flow past *sequence_len* timesteps into the past, and this limits the LSTM's prediction ability. To see why this helps, imagine a sample contains an opening `(`, but the closing `)` is in the next sample. In stateless training, the LSTM's knowledge of the `(` is unable to transfer to the next batch to help it predict the presence of `)`, and the loss will be higher. In stateful LSTMs, the `(` is encoded in the hidden state and passed to the next batch.

Correctly utilizing the power of stateful LSTMs comes with a bit of prepwork. First, the $shuffle$ argument of the $fit()$ method must be set to $False$, otherwise the temporal patterns between batches are completely destroyed. The batch size must be explicitly set because Keras needs to allocate space to store the hidden states for each sample in the batch. The amount of training and validation data must be evenly divisible by the batch size, or there will be an instance where the matrix shapes do not line up. This can be accomplished by chopping off excess training data to the nearest multiple of batch sizes [5].

Instead of taking a sequence of characters and predicting the next character, a stateful network can be many to many and predict *every* next character from the input sequence. For example, the training data $[abcdefghi]$ with *sequence_len* $= 4$ would be split into 2 training samples. The overlap of the input sequences is reduced to 0, removing redundancy, and instead of subsampling with *step* the network is allowed to predict all outputs, making better use of the training data. Compare this to the stateless training samples from section 2.1. This rearrangement of training data makes the stateful LSTM faster and better.

$$X = \left[\begin{array}{cccc} a & b & c & d \\ \hline e & f & g & h \end{array}\right]$$

$$Y = \left[\begin{array}{cccc} b & c & d & f \\ \hline f & g & h & i \end{array}\right]$$

Finally, extra care must be taken to the ordering of the data. When asked to make batches of data, Keras will take the first *batch_size* samples and stack them. If the training data is the sequence of numbers $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$, and we set $batch\_size = 3$, Keras will make batches of

$$X = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

However, the LSTM handling the first sample in batches will receive $[1, 4, 7, 10]$, which is clearly wrong! To fix this, the sequences need to be interspaced, such that the original data sequence reads $[1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 12]$. Then, when batched, the samples in separate batches will be in order.

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Care must be taken when choosing validation data. If text is selected from the middle of sequences, the jump would break the temporal relationships. If a row is selected from batches, then the batches of sequences so carefully interleaved will also be interfered with. The solution is to take the last *validation_split* percent of data from the end of the batch sequences. This would look like the following:

$$X\_train = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{bmatrix}$$

$$X\_validate = \begin{bmatrix} 4 \\ 8 \\ 12 \end{bmatrix}$$

5 Text Generation

After training the LSTM, we want to generate new samples of text. We first feed the LSTM a run of characters to get it “warmed up”. Then, we take the last layer softmax output, convert to probabilities, and sample a character. We feed this character as input to the LSTM, and repeat as much as we want.

5.1 Temperature

When sampling output probabilities, a temperature value (also known as diversity) can be tuned. When the temperature is low, more weight is put towards values the network is more confident in. When higher, the probabilities are more even, leading to larger variation in the output. To demonstrate this difference, we asked the LSTM trained on CWRU courses to generate 200

$diversity = 0.3$

```
introduction to the context of the contemporary context of the contemporary art and
culture of the context of the contemporary art and political science and the
professional and the process of the contemporary intern
```

$diversity = 0.9$

```
introduction to case studies of individual programs, discussions and as eneagy or
particular. students will not gain training covering the study of security interness.
engagement with healthcare in visitor and fall
```

Figure 2: **Effect of Diversity on Generated Text.** A low diversity leads the network to be conservative with its output characters and it tends to repeat itself. Higher diversity leads to more variety, and sometimes spelling errors. The warmup sequence was “introduction to.”

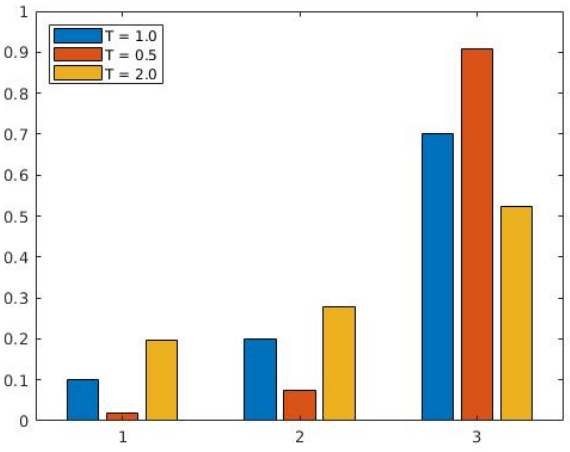


Figure 3: **Effect of Diversity on Probabilities.** When diversity is low, the values the network is more confident in are more likely to be chosen, as indicated by the red bars. When high, the probabilities become more even, as indicated by the yellow bars. The original probabilities in blue are $[0.1, 0.2, 0.7]$.

characters of text with a diversity of 0.1 and 1.0. The results are shown in Figure 2. Values for temperature vary by application but 0.75 has been shown to produce good results.

6 Results

6.1 CWRU Courses

Before taking on the more structured Python code, our implementation was tested on the complete CWRU courses catalog. Using a webscraper, the course descriptions for every four letter class code[1] was downloaded and concatenated into a 3.9 MB text file. This data set contains 61 lowercase characters. A 3 layer LSTM with 512 units per layer and 0.5 dropout after each layer was trained using RMSprop, $learning\_rate = 0.0015, clipnorm = 5$ for 150 epochs on a Google Collab Notebook

[1]`https://bulletin.case.edu/course-descriptions/`

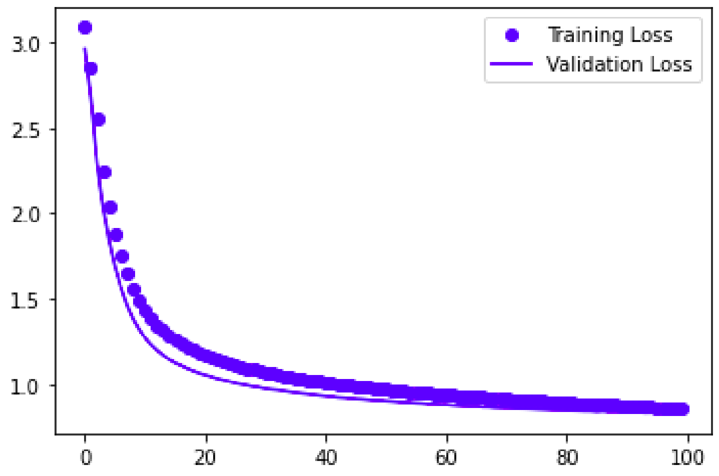


Figure 4: **Loss Curves for CWRU Courses Dataset**

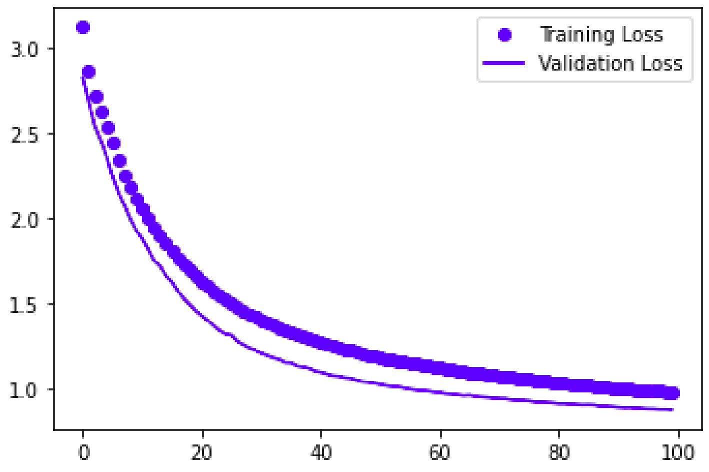


Figure 5: **Loss Curves for Python Dataset**

GPU Runtime for 2 hours. A sample of 2,500 characters[2] generated with $diversity = 0.75$ is shown in Figure 6.

6.2 Python Code

Python sources files were extracted from Python GitHub repositories and concatenated into a 4.3 MB file. See Appendix A for sources. Before preprocessing, this dataset contained an astounding 318 unique characters, due to scripts designed to test handling of Unicode characters. After preprocessing, the dataset is left with 74 unique lowercase characters. The same hyperparameters as section 6.1 were used, and due to the similar dataset size, training took the same amount of time. A sample of 1,250 characters generated with $diversity = 0.75$ is shown in Figure 7.

[2]Samples of size 50,000 are available at `https://github.com/edf42001/ECSE-484-Computational-Intelligence`

```
the course will be acquired to address the manifestations of the natural world.
students will be able to develop and understand the effective use of social policies
and communication theory and health interventions.  water semantic process does the
application of these advances to delivery complex infiction and structure of
relationships with emerging protections encountered by state, control of divise,
and the data to well explored the use of communication theory and explain the
future of philosophy and social change in a planet of clients within the context
of social inequality.  this course then integrated the reflection of transportation,
including concepts of movement, and communication of biological interactions.
the class will use it these contrast their transitions as they interpret the
exciting social media is a set of advanced practice consequences of fundamental
accessible these determinants.  the problem of international business to encourage
challenges in contemporary genres, such as storytelling and racial relationships
with the power, and in one.
offered as afst 318, eths 312, hsty 311, and hsty 411.
prereq: graduate standing or instructor permission.

hsty 455.  philosophy of theater.  3 units.
this course involves comparative skills for leadership, and political reviews of
individual parts and the japanese language explored in the mental health process
to the nature of gender. this course will also examine how economic constructions of
art.
offered as posc 376 and posc 477.

posc 477.  legal environment and protest.  3 units.
this course will examine the scientific breadth of the art and music inference theory,
study of multicales and differences, population relations, power and decline but also
them to one commercial violence. the interaction of close relationships among the roman
nature of the financial transformation of the development of the social institutions,
its approaches, and concepts in an evolutionary participation.  offered as engl 328
and engl 428.
prereq: undergraduate student and (math 224 or math 227) and (rege 403 or cogs 204) and
(sybb 301r or stat 312 or math 126).

ecse 390.  contemporary capation in materials.  3 units.
this course is offered as a supervising study of the specialized level of the major
with research to statity in the dental design and reeding.
prereq: which 205.

eths 329.  physical economy in the field of contemporary drug design.  3 units.
social construction of many competitive theories which will be allowe
```

Figure 6: **Generated CWRU course descriptions**

```
    for i in range (1, 10):
        for i in range(1, len(adj)):
            for j in range(m - 1):
                valid = (i, j)
                for j in range(len(direction)):
                    if j > len(arr):
                        len(self.find_to_rows())
                        order = []
                        print("iterations of the direction.")
                else:
                    nodes[i] = []
                else:
                    nodes[i][j] = 0
        return generate_scare([], second[i])
        print("--> segaration elements in {"the position}")
        node = sorted(positions[1])
        for node in nodes:
            if node.left is none and self.node.parent:
                self.node.left = node.left
                node.parent = node.left

    def cholse_left(self) -> none:
        """
        print subtree collection
        oncup:
            >>> color = current.node()
        >>> root.next.color = color.insert(1)
        >>> tor_node.collection
        traceback (most recent call last):
        ...
    exception: none is not insert
        """
        if not self.parent:
            return false
        if self.left is none:
            raise exception()
        self.set_left = self.calc_recursion(self.ref)
        return true


def get_left_left(root):
    return root.right
```

Figure 7: **Generated Python code**

6.3 Analysis

The network has successfully picked up the structure of Python code. It successfully indents and closes parentheses and quotes. It knows that different types of words (more English, less code-like) appear inside of quotes and after comments. It is not perfect, sometimes making indentation or mismatched brackets errors. It also has no real sense of context. For example, using variables that do not exist, or creating two `for` loops in a row with `i` as an index.

The type of text generated is entirely dependent on the training data. This sounds obvious, but can cause issues as there are many different types of Python code. Previous experiments used Python package source code, which consisted of almost 30% block comments. The network mostly just generated these block comments, which were not as interesting. The current dataset consists of implementations of various algorithms, such as depth first search, which is why the network often uses `node` as a variable.

The network trained on CWRU courses shows the flexibility of our approach. It is interesting to see how the lengths of the course descriptions vary: in the training data set some course description are very long and some are very short. The network's sentence structure is odd, however it does capture the essence of course descriptions. Most sentences are of appropriate length.

There are two types of regularization that prevent overfitting. The dropout of 0.5 after each layer leads to a validation loss that is less than training loss during training, and we stop the training process before validation loss starts to go up. Also, the diversity value can be tuned to produce less standard text. We can verify the network has learned to generalize by searching the original training data for text the network has produced. For example, the lines `if j > len(arr):` and `node = sorted(positions[1])` do not appear anywhere in the training data. The closest training data to `positions[1]` is `positions[m]`. The network has learned from other examples that a `1` often follows a `[`, and that is what was selected.

6.4 Visualization

Can we see what the LSTM is thinking if we record the activation of neurons in the hidden layers? If we look at a specific neuron and color each input character according to its activation, we can look for patterns a human can discern. The vast majority of neurons do not display any pattern, but some special ones do. A few of these interesting neurons have been collected in Figure 8.

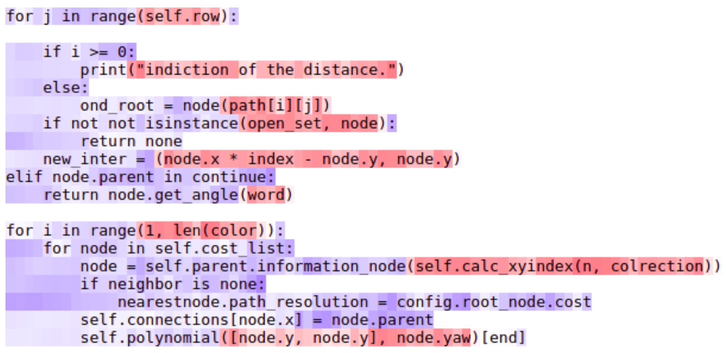
7 Conclusions And Future Work

In this paper we demonstrated the use of LSTM networks for text generation. We detail the use of stateful networks in Keras for more effecient training and use our methods to produce results on two different datasets.

Although backpropagation through time is used on sequences of length 100 characters, the network appears to have a very short attention span. This could be because most of Python code is indentation made from spaces. Replacing these with tabs could allow the network to spend attention elsewhere. It would also be interesting to see how increasing the sequence length effects learning, if at all, with stateful LSTMs.

Scaling the datasets and networks is another future direction. 5 million parameters with a 4 MB dataset is considered small by today's standards, so a dataset in the 10s or 100s of MB could lead to more understanding of context.

We used RMSprop for gradient descent, and it would be valuable to comapre this to Adam.

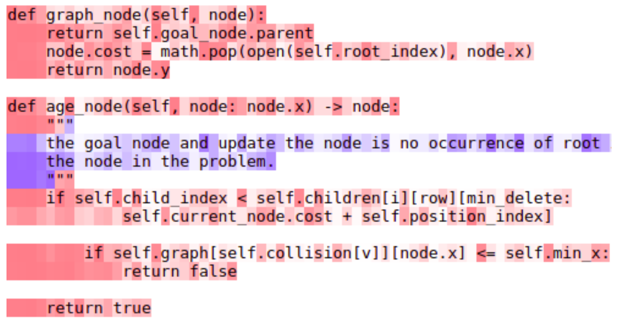


```
for j in range(self.row):

    if i >= 0:
        print("indiction of the distance.")
    else:
        ond_root = node(path[i][j])
    if not not isinstance(open_set, node):
        return none
    new_inter = (node.x * index - node.y, node.y)
elif node.parent in continue:
    return node.get_angle(word)

for i in range(1, len(color)):
    for node in self.cost_list:
        node = self.parent.information_node(self.calc_xyindex(n, colrection))
        if neighbor is none:
            nearestnode.path_resolution = config.root_node.cost
        self.connections[node.x] = node.parent
        self.polynomial([node.y, node.y], node.yaw)[end]
```

(a) A neuron that activates inside of parentheses



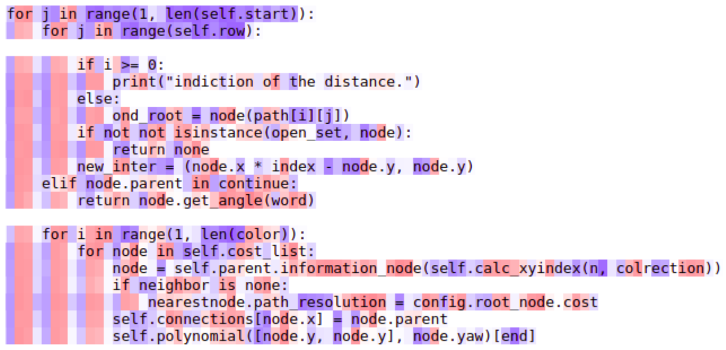
```
def graph_node(self, node):
    return self.goal_node.parent
    node.cost = math.pop(open(self.root_index), node.x)
    return node.y

def age_node(self, node: node.x) -> node:
    """
    the goal node and update the node is no occurrence of root
    the node in the problem.
    """
    if self.child_index < self.children[i][row][min_delete:
            self.current_node.cost + self.position_index]

        if self.graph[self.collision[v]][node.x] <= self.min_x:
            return false

    return true
```

(b) A neuron that activates inside of blockquotes

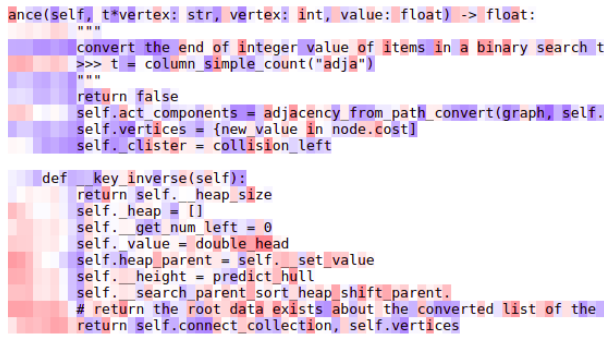


```
for j in range(1, len(self.start)):
    for j in range(self.row):

        if i >= 0:
            print("indiction of the distance.")
        else:
            ond_root = node(path[i][j])
        if not not isinstance(open_set, node):
            return none
        new_inter = (node.x * index - node.y, node.y)
    elif node.parent in continue:
        return node.get_angle(word)

    for i in range(1, len(color)):
        for node in self.cost_list:
            node = self.parent.information_node(self.calc_xyindex(n, colrection))
            if neighbor is none:
                nearestnode.path_resolution = config.root_node.cost
            self.connections[node.x] = node.parent
            self.polynomial([node.y, node.y], node.yaw)[end]
```

(c) A neuron that tracks indentation



```
ance(self, t*vertex: str, vertex: int, value: float) -> float:
        """
        convert the end of integer value of items in a binary search t
        >>> t = column_simple_count("adja")
        """
        return false
        self.act_components = adjacency_from_path_convert(graph, self.
        self.vertices = {new_value in node.cost]
        self._clister = collision_left

    def __key_inverse(self):
        return self.__heap_size
        self._heap = []
        self.__get_num_left = 0
        self._value = double_head
        self.heap_parent = self.__set_value
        self.__height = predict_hull
        self.__search_parent_sort_heap_shift_parent.
        # return the root data exists about the converted list of the
        return self.connect_collection, self.vertices
```

(d) A typical neuron

Figure 8: **Activations of Neurons in LSTM Hidden Layers.** Blue represents strong negative spiking, red is positive. Most of the interesting examples come from the second or third layer of the LSTM.

Finally, this technique could be applied to music, as music can be represented as a sequence of notes where each note is a different character. How could the algorithm be adapted to work with multiple notes played at the same time?

References

1. Carykh. *AI evolves to compose 3 hours of jazz!* `https://www.youtube.com/watch?v=nA3YOFUCn4U/`. [Online; accessed 5-Nov-2020] (Jul 2017).

2. A. Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks. *Andrej Karpathy blog* (May 2015).

3. Colah. *Understanding LSTM Networks* `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. [Online; accessed 15-Apr-2022] (Aug 2015).

4. F. Chollet, Character-level text generation with LSTM. *Keras.io* (Jun 2016).

5. A. Scripter, Understanding Keras LSTMs: Role of Batch-size and Statefulness. *Stack Overflow* (Feb 2018).

A Python Dataset Source Code Repositories

`https://github.com/keon/algorithms`
`https://github.com/geekcomputers/Python`
`https://github.com/AtsushiSakai/PythonRobotics`
`https://github.com/TheAlgorithms/Python`