Handling sequential data

NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Natural language generation

- Generation of texts in a certain style.
- Machine translation.
- Sentence or word auto-completion.
- Generation of textual summaries.
- Automated chatbots.

Introduction to sequential data

- Any data where the order matters.
- Examples - Text data, Time series data, DNA sequences.

Text or language data

- Data used in spoken or written language.
- Specific order amongst words or characters.
- Change of order - different meaning or gibberish.
- "I am learning Mathematics" - Correct.
- "learning am Mathematics I" - Doesn't make sense.
- Models should take order information into account.

An example of text dataset

```
names.head(5)
```

```
          name
0         john
1      william
2        james
3      charles
4       george
```

Names Dataset

```python
names.head(5)
```

```
            name
0           john
1        william
2          james
3        charles
4         george
```

Word delimiters

- Specify the start and end of a name using start and end token.
- One special character to specify the start - start token.
- Another special character to specify the end - end token.
- Start token - `\t` .
- End token - `\n` .

Insert start token

- Start token in front of the name.

```
data['name'] = data['name'].apply(lambda x : '\t' + x)
```

```
        name
0     \tjohn
1  \twilliam
2    \tjames
3  \tcharles
4   \tgeorge
```

Append end token

- End token at the end of the name.

```
data['target'] = data['name'].apply(lambda x : x[1:len(x)] + '\n')
```

```
         name     target
0      \tjohn     john\n
1   \twilliam  william\n
2     \tjames    james\n
3   \tcharles  charles\n
4    \tgeorge   george\n
```

Vocabulary for names dataset

- Vocabulary - set of all unique characters used in the dataset.

```
def get_vocabulary(names):
    # Define vocabulary as a set and include start and end token
    vocabulary = set(['\t', '\n'])
    # Iterate over all names and all characters of each name
    for name in names:
        for c in name:
            if c not in all_chars:
                # If character is not in vocabulary, add it
                vocabulary.add(c)
    # Return the vocabulary
    return vocabulary
```

Character to integer mapping

- Sort the vocabulary and assign numbers in order.
- Character `\t` mapped to `0` , `\n` to `1` , `a` to `2` , `b` to `3` , etc.

```
ctoi = { char : idx for idx, char in enumerate(sorted(vocabulary))}
```

```
{'\t': 0, '\n': 1, 'a': 2, 'b': 3, 'c': 4, ...}
```

Integer to character mapping

- Integer to character mapping.
- Integer `0` to `\t` , `1` to `\n` , `2` to `a` , `3` to `b` , etc.

```
itoc = { idx : char for idx, char in enumerate(sorted(vocabulary))}
```

```
{0: '\t', 1: '\n', 2: 'a', 3: 'b', 4: 'c', ...}
```

Let's practice!

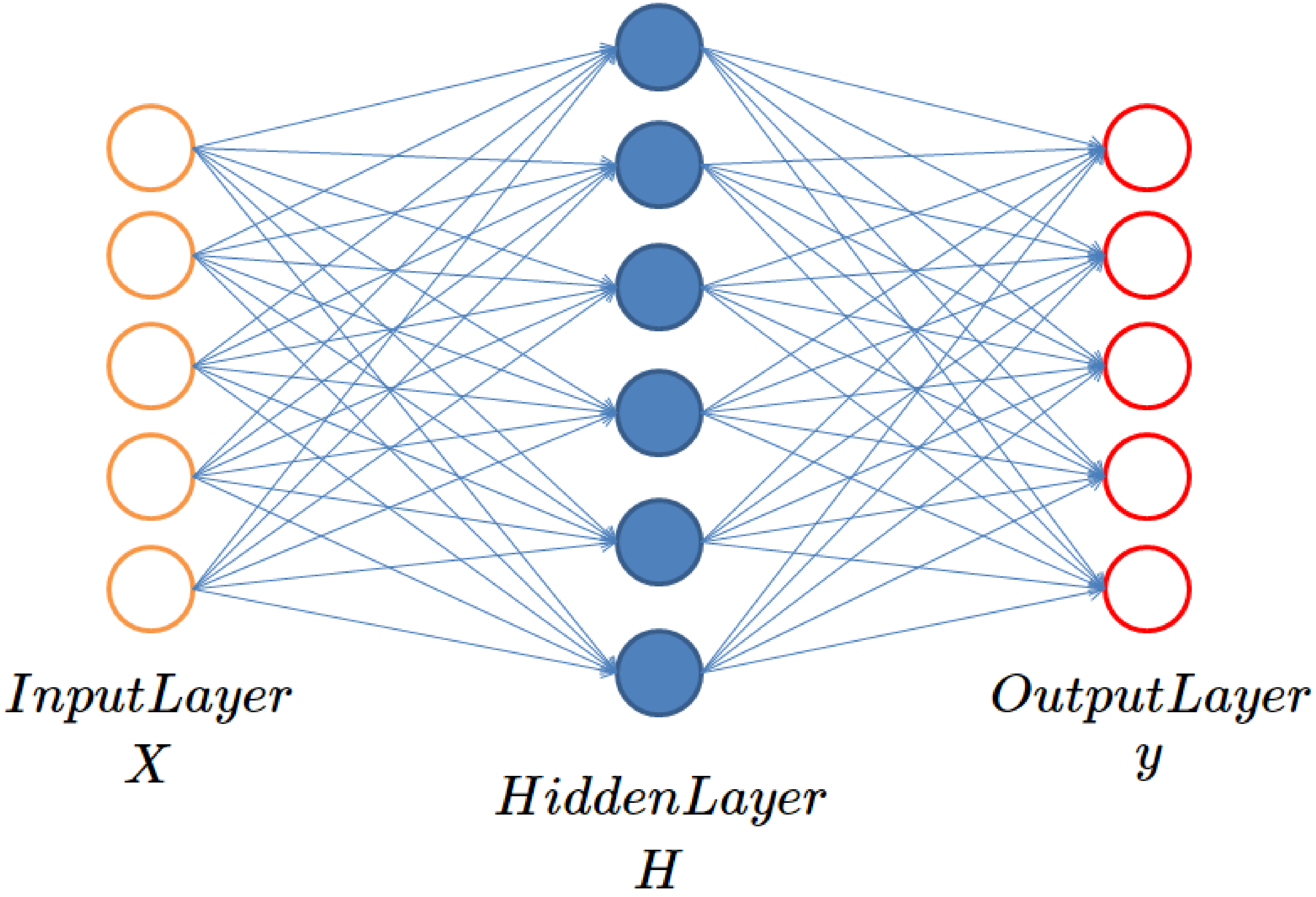
NATURAL LANGUAGE GENERATION IN PYTHON

Introduction to recurrent neural network

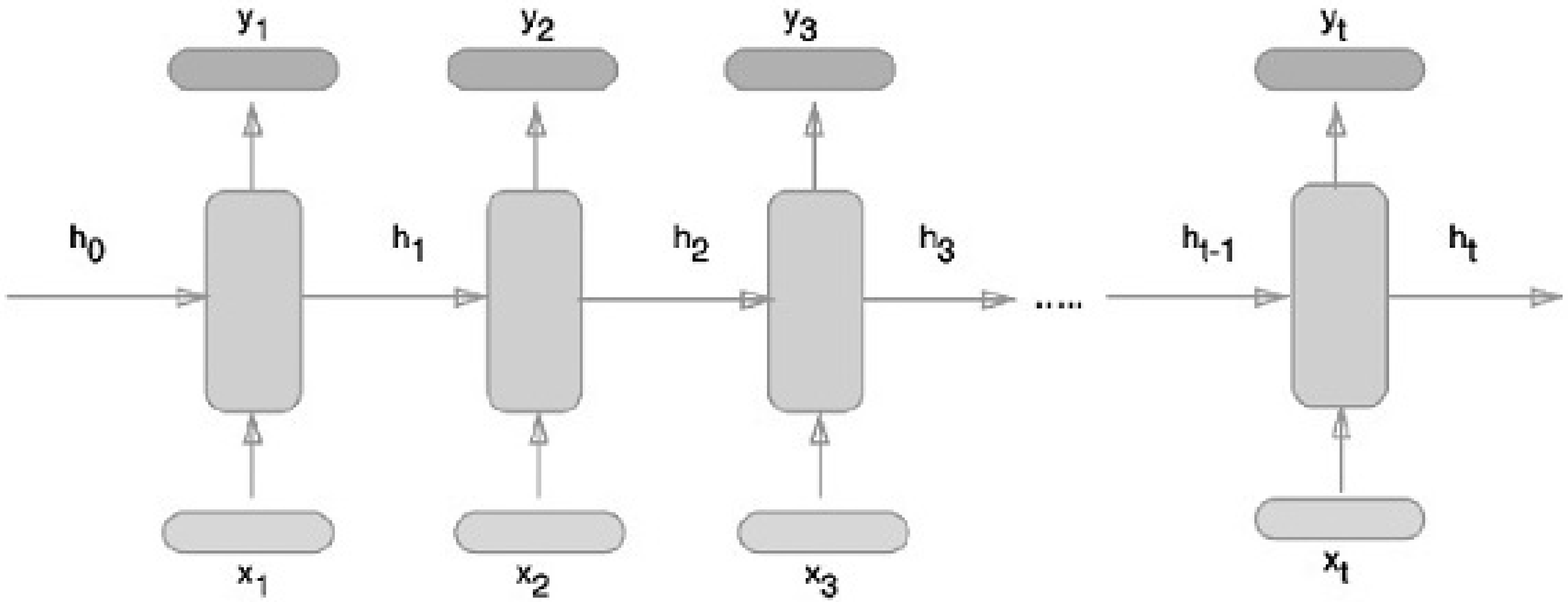
NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Feed-forward neural network

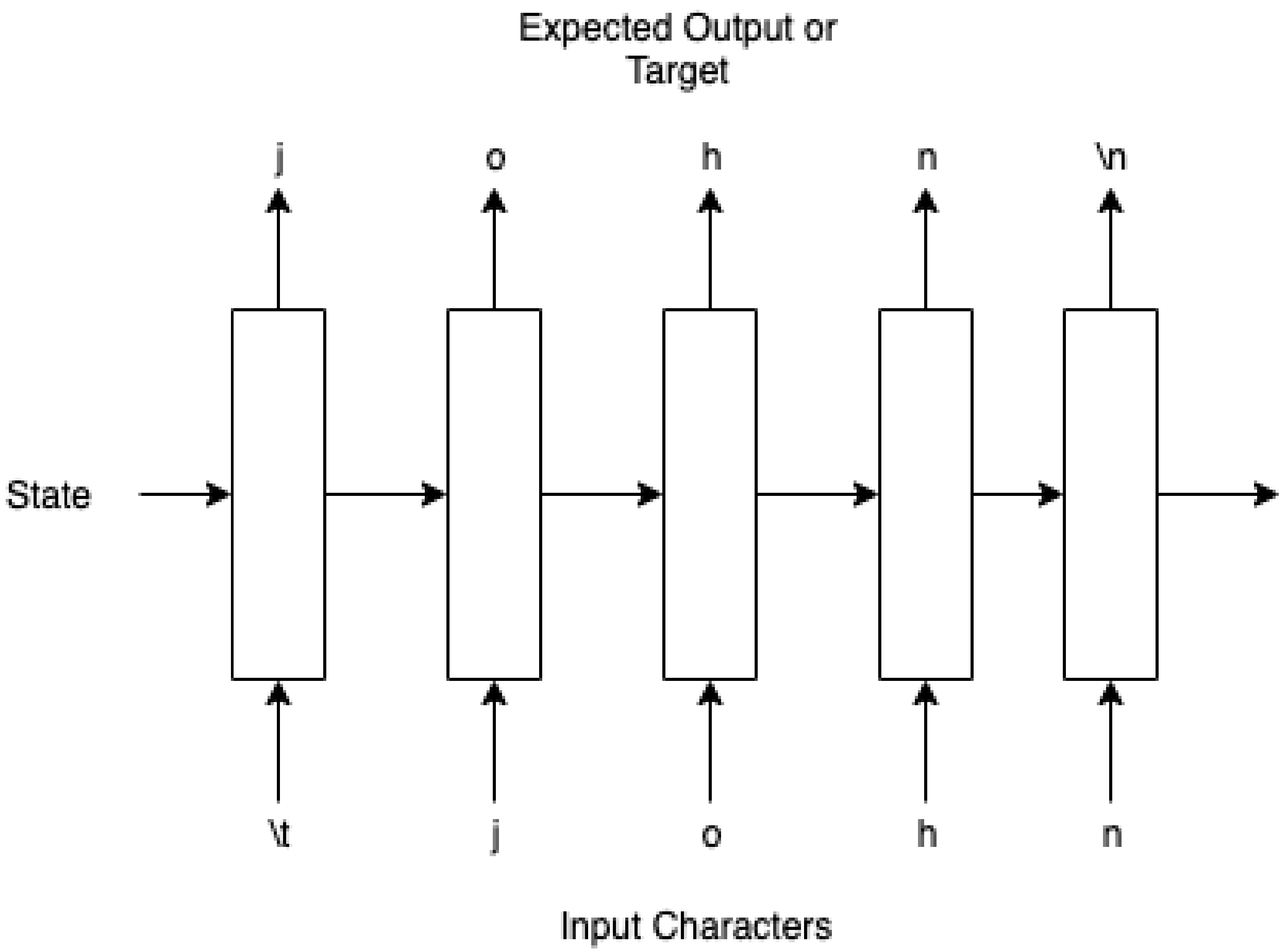


Introducing recurrence



RNN for baby name generator

- Generate next character given current.
- Keep track of the history so far.
- Generate name `john` .
- Sequence - `\t` , `j` , `o` , `h` , `n` , `\n` .
- Time-step 1: input `\t` , output `j` .
- Time-step 2: input `j` , output `o` .
- State remembers `\t` and `j` seen so far.
- Continue till end of sequence.



Encoding of the characters

- Character to integer mapping.

```
{'\t': 0, '\n': 1, 'a': 2, 'b': 3, 'c': 4, ...}
```

  - One-hot encoding of the characters.

```
'\t' = [1, 0, 0, 0, ..., 0]
'\n' = [0, 1, 0, 0, ..., 0]
 'a' = [0, 0, 1, 0, ..., 0]
 'b' = [0, 0, 0, 1, ..., 0]
   .
   .
   .
 'z' = [0, 0, 0, 0, ..., 1]
```

Number of time steps

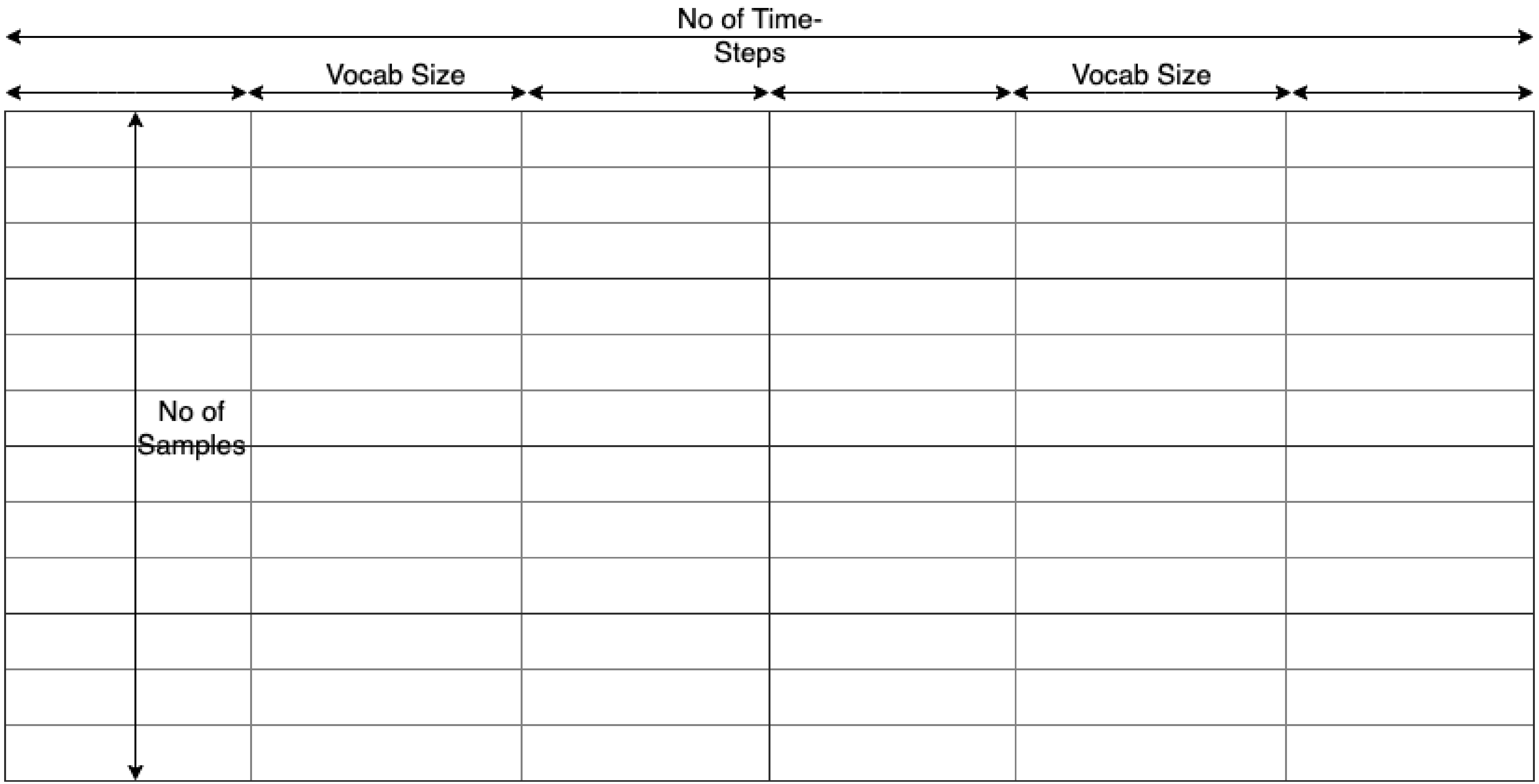
- Time-step: Length of the longest name.

```
def get_max_len(names):
    length_list=[]
    for l in names:
        length_list.append(len(l))
    max_len = np.max(length_list)
    return max_len
```

```
max_len = get_max_len(names)
```

- Each name as a sequence of length `max_len`

Input and target vectors



Initialize the input vector

- Create 3-D zero vector of required shape for input.

```
input_data = np.zeros((len(names.name), max_len+1, len(vocabulary)),
                      dtype='float32')
```

  - Fill the vector with data

```
for n_idx, name in enumerate(names.name):
    for c_idx, char in enumerate(name):
        input_data[n_idx, c_idx, char_to_idx[char]] = 1.
```

Initialize the target vector

- Create 3-D zero vector of required shape for target.

```
target_data = np.zeros((len(names.name), max_len+1, len(vocabulary)),
                       dtype='float32')
```

- Fill the target vector with data.

```
for n_idx, name in enumerate(names.target):
    for c_idx, char in enumerate(name):
        target_data[n_idx, c_idx, char_to_idx[char]] = 1.
```

Build and compile recurrent neural network

```
model = Sequential()
```

```
model.add(SimpleRNN(50, input_shape=(max_len+1, len(vocabulary)),
                    return_sequences=True))
```

```
model.add(TimeDistributed(Dense(len(vocabulary), activation='softmax')))
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Check model summary

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 13, 50)            3950
_________________________________________________________________
time_distributed_1 (TimeDist (None, 13, 28)            1428
_________________________________________________________________
time_distributed_2 (TimeDist (None, 13, 28)            0
=================================================================
Total params: 5,378
Trainable params: 5,378
Non-trainable params: 0
_________________________________________________________________
```

Let's practice!

NATURAL LANGUAGE GENERATION IN PYTHON

Inference using recurrent neural network

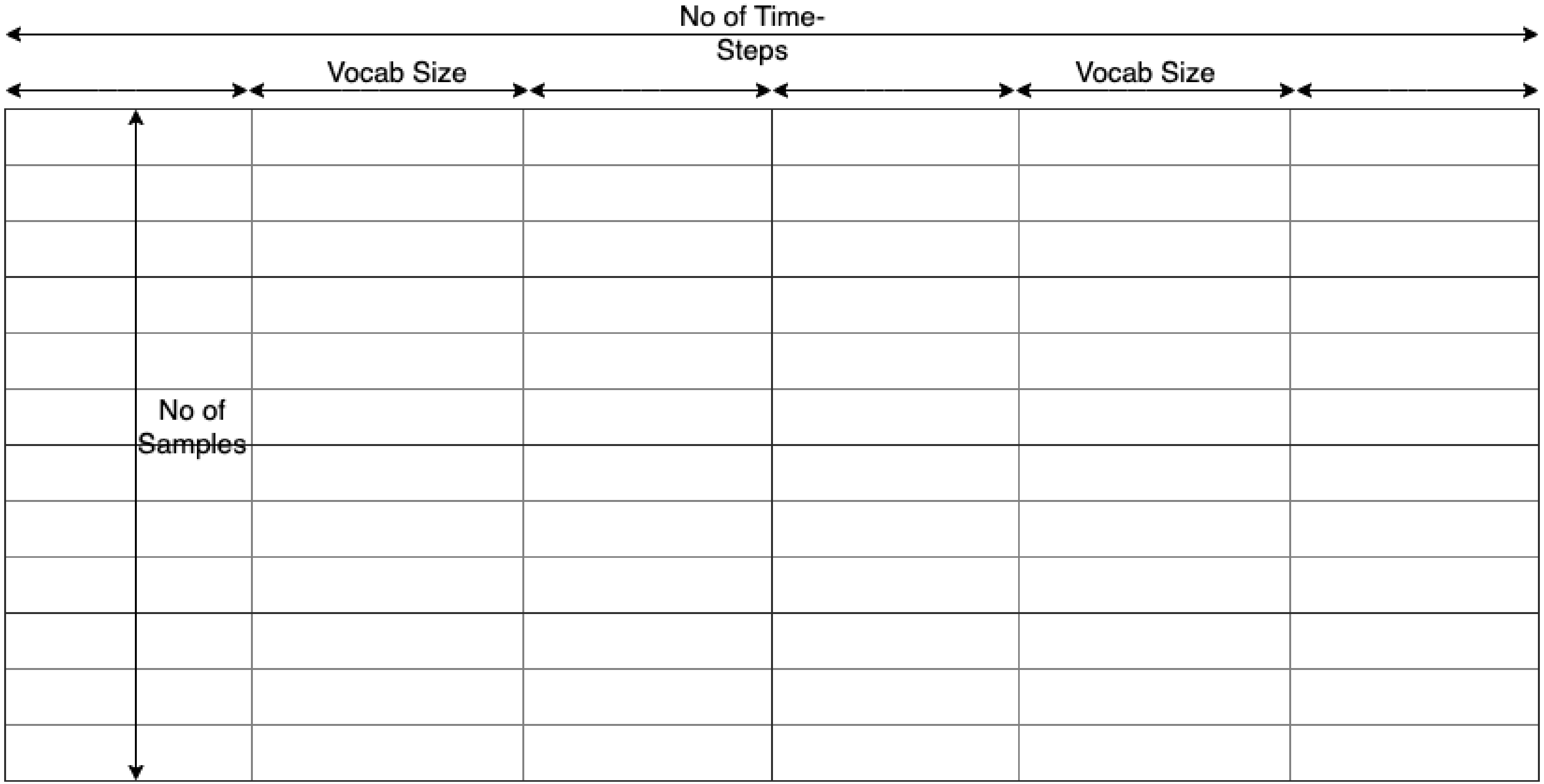
NATURAL LANGUAGE GENERATION IN PYTHON

Biswanath Halder
Data Scientist

Understanding training

- Neural network: a black box.
- Input target pair (x, y): ideal output y for input x.
- For input x produces output, say, z.
- Goal: reduce difference between actual output z and ideal output y.
- Training: adjust the internal parameters to achieve goal.
- After training actual output more similar to ideal output.

Input and target vectors for training



Train recurrent network

- Train recurrent network.

```
model.fit(input_data, target_data, batch_size=128, epochs=15)
```

  - Batch size: number of samples after which the parameters are adjusted.
  - Epoch: number of times to iterate over the full dataset.

Predict first character

- Initialize the first character of the sequence.

```
output_seq = np.zeros((1, max_len+1, len(vocabulary)))
output_seq[0, 0, char_to_idx['\t']] = 1
```

  - Probability distribution for the next character.

```
probs = model.predict_proba(output_seq, verbose=0)[:,1,:]
```

  - Sample the vocabulary using the probability distribution.

```
first_char = np.random.choice(sorted(list(vocabulary)), replace=False,
                              p=probs.reshape(28))
```

Predict second character using the first

- Insert first character in the sequence.

```
output_seq[0, 1, char_to_idx[first_char]] = 1
```

  - Sample from probability distribution.

```
probs = model.predict_proba(output_seq, verbose=0)[:,2,:]
second_char = np.random.choice(sorted(list(vocabulary)), replace=False,
                               p=probs.reshape(28))
```

Generate baby names

```python
def generate_baby_names(n):
    for i in range(0,n):
        stop=False
        counter=1
        name = ''
        # Initialize first char of output sequence
        output_seq = np.zeros((1, max_len+1, 28))
        output_seq[0, 0, char_to_idx['\t']] = 1.
        # Continue until a newline is generated or max no of chars reached
        while stop == False and counter < 10:
            # Get probability distribution for next character
            probs = model.predict_proba(output_seq, verbose=0)[:,counter-1,:]
            # Sample vocabulary to get most probable next character
            c = np.random.choice(sorted(list(vocabulary)), replace=False, p=probs.reshape(28))
            if c=='\n':
                stop=True
            else:
                name = name + c
                output_seq[0,counter , char_to_idx[c]] = 1.
                counter=counter+1
        print(name)
```

Cool baby names

```
generate_baby_names(10)
```

```
leannad
elfrey
lisse
artima
revel
geletha
ortone
rorental
berne
raypha
```

Let's practice!

NATURAL LANGUAGE GENERATION IN PYTHON