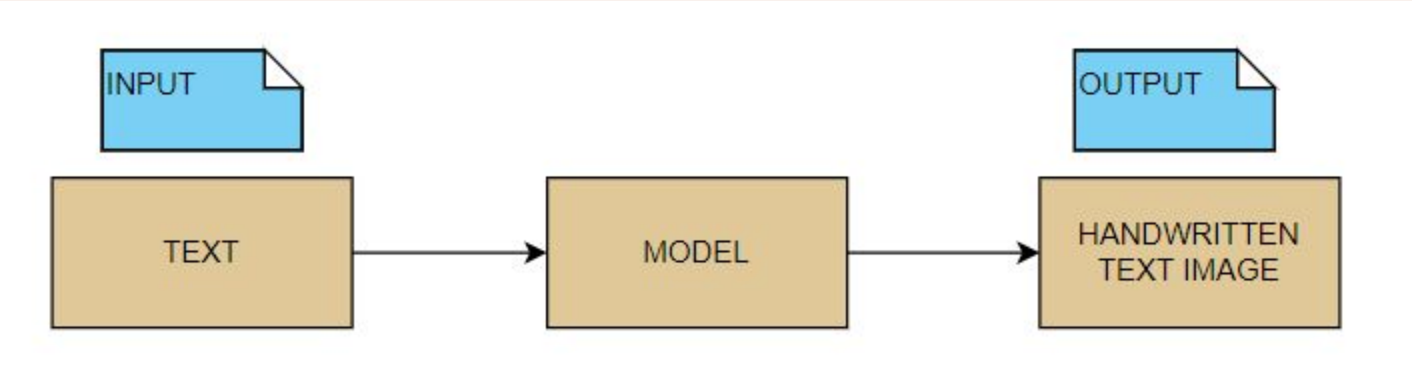
HANDWRITING GENERATION

PRANAVA RAMAN B M S - 2019103555
PREETI KRISHNAVENI RA - 2019103560
ANUSREE V - 2019103507

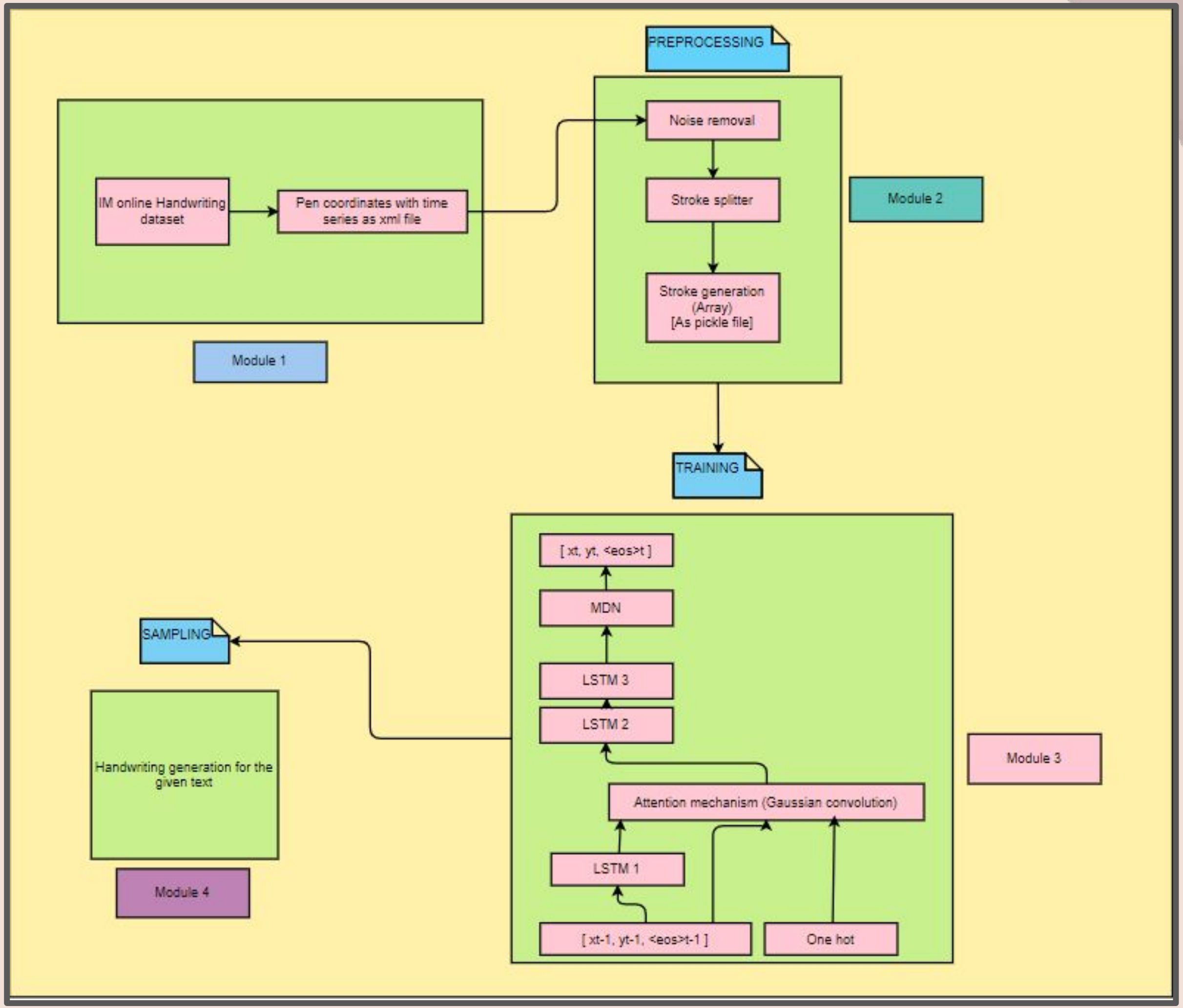
Introduction

Recurrent neural networks (RNNs) are a rich class of dynamic models that have been used to generate sequences in domains as diverse as music, text and motion capture data. RNNs can be trained for sequence generation by processing real data sequences one step at a time and predicting what comes next. Assuming the predictions are probabilistic, novel sequences can be generated from a trained network by iteratively sampling from the network's output distribution, then feeding in the sample as input at the next step. In this project, we are using RNNs to generate handwriting from different styles of handwriting data and a given input.

OVERALL BLOCK DIAGRAM



BLOCK DIAGRAM



Completed

MODULE 1 - DATASET

We used IAM Handwriting Database to train the model. As far as datasets go, Although it's very small in size (less than 50 MB once parsed), preprocessing it generates a huge dataset. A total of 657 writers contributed to the dataset and each has a unique handwriting style.

The data itself is a three-dimensional time series. The first two dimensions are the (x, y) coordinates of the pen tip and the third is time at which the pen was at that coordinate. Each line has around 500 pen points and an annotation of ascii characters.

Module - 2
Preprocessing (Data loader)

Here the time series data is converted into strokes data with corresponding ascii input labels. We first take the inputs and find the distance between the adjacent pen co ordinates. We remove noises at this part by removing points that are too distant. Then the strokes are converted into arrays and along with their labels, they are given output as pickle files. This is used as input for training purposes. Multiple styles of writing are saved as different pickle files.

OUTPUT - PREPROCESSING

linestrokes-XML


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<WhiteboardCaptureSession>
  <WhiteboardDescription>
    <SensorLocation corner="top_left"/>
    <DiagonallyOppositeCoords x="6512" y="1376"/>
    <VerticallyOppositeCoords x="966" y="1376"/>
    <HorizontallyOppositeCoords x="6512" y="787"/>
  </WhiteboardDescription>
  <StrokeSet>
    <Stroke colour="black" start_time="769.05" end_time="769.64">
      <Point x="1073" y="1058" time="769.05"/>
      <Point x="1072" y="1085" time="769.07"/>
      <Point x="1066" y="1117" time="769.08"/>
      <Point x="1052" y="1152" time="769.10"/>
      <Point x="1030" y="1196" time="769.12"/>
      <Point x="1009" y="1242" time="769.13"/>
      <Point x="994" y="1286" time="769.14"/>
      <Point x="980" y="1317" time="769.16"/>
      <Point x="971" y="1336" time="769.18"/>
      <Point x="968" y="1344" time="769.19"/>
      <Point x="966" y="1339" time="769.20"/>
      <Point x="972" y="1340" time="769.22"/>
      <Point x="978" y="1320" time="769.24"/>
      <Point x="991" y="1298" time="769.25"/>
      <Point x="1003" y="1266" time="769.27"/>
      <Point x="1016" y="1231" time="769.28"/>
      <Point x="1021" y="1184" time="769.30"/>
      <Point x="1030" y="1143" time="769.31"/>
      <Point x="1040" y="1108" time="769.33"/>
      <Point x="1049" y="1077" time="769.34"/>
      <Point x="1055" y="1049" time="769.36"/>
      <Point x="1058" y="1021" time="769.37"/>
      <Point x="1064" y="1006" time="769.38"/>
      <Point x="1071" y="1006" time="769.40"/>
      <Point x="1071" y="1006" time="769.42"/>
      <Point x="1074" y="1013" time="769.43"/>
      <Point x="1083" y="1042" time="769.45"/>
      <Point x="1097" y="1082" time="769.46"/>
      <Point x="1114" y="1124" time="769.48"/>
```


ascii-XML

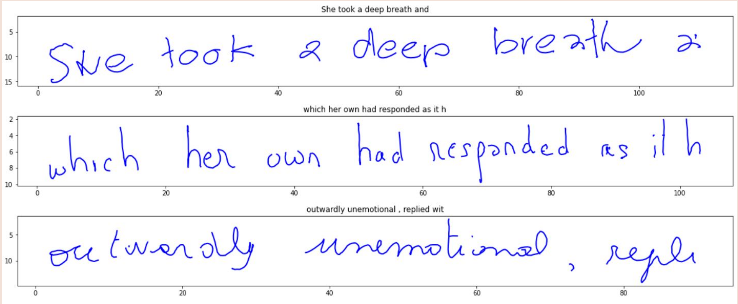
OCR:

A MOVE to stop Mr . Gaitskell from nominating
any more Labour life Peers is to be made at a
meeting of Labour 0M Ps tomorrow . Mr . Michael
Foot has put down a resolution on the subject
and he is to be backed by Mr . Will Griffiths ,

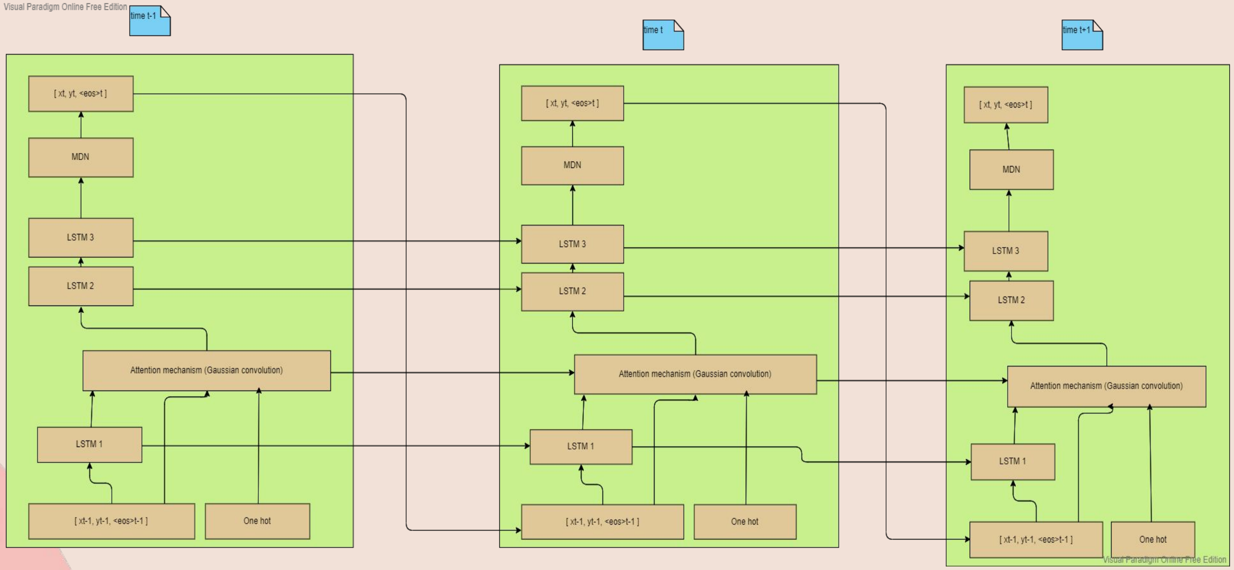
CSR:

A MOVE to stop Mr . Gaitskell from nominating
any more Labour life Peers is to be made at a
meeting of Labour 0M Ps tomorrow . Mr . Michael
Foot has put down a resolution on the subject
and he is to be backed by Mr . Will Griffiths

Output after preprocessing - visualization



Module - 3
Training



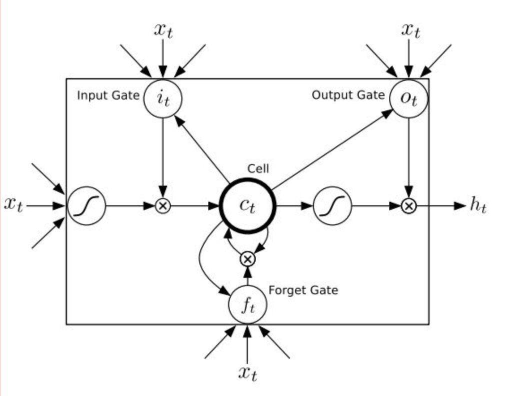
The backbone of the model is three **LSTM** cells .

There is a **custom attention mechanism** which digests a one-hot encoding of the sentence we want the model to write.

The **Mixture Density Network** on top chooses appropriate Gaussian distributions from which to sample the next pen point, adding some natural randomness to the model.

Long Short-term Memory (LSTM) Cell

- At the core of the Graves handwriting model are three Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs).
- They keep a trail of time-independent patterns by using a differentiable memory.
- They are capable of learning long-term dependencies.
- In these LSTMs, we use three different tensors for performing each of the “erase”, “write” and “read” operations on the “memory” tensor. This helps in deciding what details to forget and what to remember.
- TensorFlow’s seq2seq API is used to build the model.
- RNNs are extremely good at modeling sequential data.



$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)\\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)\\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)\\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)\\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

```
# ——— build the basic recurrent network architecture
cell_func = tf.contrib.rnn.LSTMCell # could be GRUCell or RNNCell
self.cell0 = cell_func(args.rnn_size, state_is_tuple=True, initializer=self.graves_initializer)
self.cell1 = cell_func(args.rnn_size, state_is_tuple=True, initializer=self.graves_initializer)
self.cell2 = cell_func(args.rnn_size, state_is_tuple=True, initializer=self.graves_initializer)

if (self.train and self.dropout < 1): # training mode
    self.cell0 = tf.contrib.rnn.DropoutWrapper(self.cell0, output_keep_prob = self.dropout)
    self.cell1 = tf.contrib.rnn.DropoutWrapper(self.cell1, output_keep_prob = self.dropout)
    self.cell2 = tf.contrib.rnn.DropoutWrapper(self.cell2, output_keep_prob = self.dropout)

self.input_data = tf.placeholder(dtype=tf.float32, shape=[None, self.tsteps, 3])
self.target_data = tf.placeholder(dtype=tf.float32, shape=[None, self.tsteps, 3])
self.istate_cell0 = self.cell0.zero_state(batch_size=self.batch_size, dtype=tf.float32)
self.istate_cell1 = self.cell1.zero_state(batch_size=self.batch_size, dtype=tf.float32)
self.istate_cell2 = self.cell2.zero_state(batch_size=self.batch_size, dtype=tf.float32)

#slice the input volume into separate vols for each tstep
inputs = [tf.squeeze(input_, [1]) for input_ in tf.split(self.input_data, self.tsteps, 1)]
#build cell0 computational graph
outs_cell0, self.fstate_cell0 = tf.contrib.legacy_seq2seq.rnn_decoder(inputs, self.istate_cell0, self.cell0, \
    loop_function=None, scope='cell0')
```

The Attention Mechanism

- In order to get the information about which characters make up this sentence, the model uses a differentiable attention mechanism.
- It is a Gaussian convolution over a one-hot ascii encoding.
- We can think of this convolution operation as a soft window through which the handwriting model can look at a small subset of characters, ie. the letters ‘wo’ in the word world.
- Since all the parameters of this window are differentiable, the model learns to shift the window from character to character as it writes them.
- The model learns to control the window parameters remarkably well.

```python
—— build the gaussian character window
def get_window(alpha, beta, kappa, c):
    # phi -> [? x 1 x ascii_steps] and is a tf matrix
    # c -> [? x ascii_steps x alphabet] and is a tf matrix
    ascii_steps = c.get_shape()[1].value #number of items in sequence
    phi = get_phi(ascii_steps, alpha, beta, kappa)
    window = tf.matmul(phi,c)
    window = tf.squeeze(window, [1]) # window ~ [?,alphabet]
    return window, phi

#get phi for all t,u (returns a [1 x tsteps] matrix) that defines the window
def get_phi(ascii_steps, alpha, beta, kappa):
    # alpha, beta, kappa -> [?,kmixtures,1] and each is a tf variable
    u = np.linspace(0,ascii_steps-1,ascii_steps) # weight all the U items in the sequence
    kappa_term = tf.square( tf.subtract(kappa,u))
    exp_term = tf.multiply(-beta,kappa_term)
    phi_k = tf.multiply(alpha, tf.exp(exp_term))
    phi = tf.reduce_sum(phi_k,1, keep_dims=True)
    return phi # phi ~ [?,1,ascii_steps]

def get_window_params(i, out_cell0, kmixtures, prev_kappa, reuse=True):
    hidden = out_cell0.get_shape()[1]
    n_out = 3*kmixtures
    with tf.variable_scope('window',reuse=reuse):
        window_w = tf.get_variable("window_w", [hidden, n_out], initializer=self.graves_initializer)
        window_b = tf.get_variable("window_b", [n_out], initializer=self.window_b_initializer)
    abk_hats = tf.nn.xw_plus_b(out_cell0, window_w, window_b) # abk_hats ~ [?,n_out]
    abk = tf.exp(tf.reshape(abk_hats, [-1, 3*kmixtures,1])) # abk_hats ~ [?,n_out] = "alpha, beta, kappa hats"

    alpha, beta, kappa = tf.split(abk, 3, 1) # alpha_hat, etc ~ [?,kmixtures]
    kappa = kappa + prev_kappa
    return alpha, beta, kappa # each ~ [?,kmixtures,1]
```

Mixed Density Network (MDN)

Think of Mixture Density Networks as neural networks which can measure their own uncertainty. Their output parameters are μ, σ, and ρ for several multivariate Gaussian components. They also estimate a parameter for each of these distributions. Think of π as the probability that the output value was drawn from that particular component distribution.

Since MDNs parameterize probability distributions, they are a great way to capture randomness in the data. In the handwriting model, the MDN learns to how messy or unpredictable to make different parts of handwriting. For example, the MDN will choose Gaussian with diffuse shapes at the beginning of strokes and Gaussians with peaky shapes in the middle of strokes.

```python
# ——— finish building LSTMs 2 and 3
    outs_cell1, self.fstate_cell1 = tf.contrib.legacy_seq2seq.rnn_decoder(outs_cell0, self.istate_cell1, self.cell1, l

    outs_cell2, self.fstate_cell2 = tf.contrib.legacy_seq2seq.rnn_decoder(outs_cell1, self.istate_cell2, self.cell2, l

# ——— start building the Mixture Density Network on top (start with a dense layer to predict the MDN params)
    n_out = 1 + self.nmixtures * 6 # params = end_of_stroke + 6 parameters per Gaussian
    with tf.variable_scope('mdn_dense'):
        mdn_w = tf.get_variable("output_w", [self.rnn_size, n_out], initializer=self.graves_initializer)
        mdn_b = tf.get_variable("output_b", [n_out], initializer=self.graves_initializer)

    out_cell2 = tf.reshape(tf.concat(outs_cell2, 1), [-1, args.rnn_size]) #concat outputs for efficiency
    output = tf.nn.xw_plus_b(out_cell2, mdn_w, mdn_b) #data flows through dense nn


# ——— build mixture density cap on top of second recurrent cell
    def gaussian2d(x1, x2, mu1, mu2, s1, s2, rho):
        # define gaussian mdn (eq 24, 25 from http://arxiv.org/abs/1308.0850)
        x_mu1 = tf.subtract(x1, mu1)
        x_mu2 = tf.subtract(x2, mu2)
        Z = tf.square(tf.div(x_mu1, s1)) + \
            tf.square(tf.div(x_mu2, s2)) - \
            2*tf.div(tf.multiply(rho, tf.multiply(x_mu1, x_mu2)), tf.multiply(s1, s2))
        rho_square_term = 1-tf.square(rho)
        power_e = tf.exp(tf.div(-Z,2*rho_square_term))
        regularize_term = 2*np.pi*tf.multiply(tf.multiply(s1, s2), tf.sqrt(rho_square_term))
        gaussian = tf.div(power_e, regularize_term)
        return gaussian
```

OUTPUT - TRAINING

```
TRAINING MODE...
<__main__.Args object at 0x7f1ad473c190>

loading data...
        loaded dataset:
                11262 train individual data points
                592 valid individual data points
                351 batches
building model...
        using alphabet abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
/tensorflow-1.15.2/python3.7/tensorflow_core/python/client/session.py:1750: UserWarning: An interactive session is already active. This can cause out-of-memory
  warnings.warn('An interactive session is already active. This can '
attempt to load saved model...
no saved model to load. starting new session
training...
learning rate: 9.999999747378752e-05
0/125000, loss = 3.090, regloss = 0.03090, valid_loss = 3.061, time = 46.975
10/125000, loss = 3.106, regloss = 0.31705, valid_loss = 3.056, time = 0.446
20/125000, loss = 3.039, regloss = 0.57752, valid_loss = 3.046, time = 0.454
30/125000, loss = 2.912, regloss = 0.81331, valid_loss = 3.033, time = 0.446
40/125000, loss = 3.039, regloss = 1.02522, valid_loss = 3.014, time = 0.823
50/125000, loss = 2.982, regloss = 1.21226, valid_loss = 2.990, time = 0.444
60/125000, loss = 3.012, regloss = 1.37867, valid_loss = 2.960, time = 0.446
70/125000, loss = 2.879, regloss = 1.52687, valid_loss = 2.917, time = 0.447
80/125000, loss = 2.799, regloss = 1.65591, valid_loss = 2.855, time = 0.459
90/125000, loss = 2.761, regloss = 1.76637, valid_loss = 2.763, time = 0.446
100/125000, loss = 2.564, regloss = 1.85541, valid_loss = 2.612, time = 0.449
110/125000, loss = 2.413, regloss = 1.91864, valid_loss = 2.374, time = 0.445
120/125000, loss = 2.009, regloss = 1.94675, valid_loss = 2.033, time = 0.448
130/125000, loss = 1.797, regloss = 1.93835, valid_loss = 1.641, time = 0.451
140/125000, loss = 1.370, regloss = 1.89288, valid_loss = 1.328, time = 0.450
150/125000, loss = 1.228, regloss = 1.83421, valid_loss = 1.215, time = 0.449
```

MODULE 4 - SAMPLING

- Once the model is trained enough epochs (in our case about 27000 times) and the loss is minimized the model is saved. Then each time during sampling the model is loaded.
- The style of the author is used to prime the model. Priming consists of joining the real pen-positions and character sequences to generate and set the synthetic pen coordinates to a vector (the positions are sampled from MDN)
- The attention window helps it in generating pen strokes closer to what humans would do while writing.
- The output is again given as time series data of pen co-ordinates, which then again is processed to show the output in the form of line plots.

```python
def sample_text(sess, args_text, translation, style=None):
    fields = ['coordinates', 'sequence', 'bias', 'e', 'pi', 'mu1', 'mu2', 'std1', 'std2',
              'rho', 'window', 'kappa', 'phi', 'finish', 'zero_states']
    vs = namedtuple('Params', fields)(
        *[tf.compat.v1.get_collection(name)[0] for name in fields]
    )

    text = np.array([translation.get(c, 0) for c in args_text])
    coord = np.array([0., 0., 1.])
    coords = [coord]

    # Prime the model with the author style if requested
    prime_len, style_len = 0, 0
    if style is not None:
        # Priming consist of joining to a real pen-position and character sequences the synthetic sequence to generate
        #   and set the synthetic pen-position to a null vector (the positions are sampled from the MDN)
        style_coords, style_text = style
        prime_len = len(style_coords)
        style_len = len(style_text)
        prime_coords = list(style_coords)
        coord = prime_coords[0] # Set the first pen stroke as the first element to process
        text = np.r_[style_text, text] # concatenate on 1 axis the prime text + synthesis character sequence
        sequence_prime = np.eye(len(translation), dtype=np.float32)[style_text]
        sequence_prime = np.expand_dims(np.concatenate([sequence_prime, np.zeros((1, len(translation)))]), axis=0)

    sequence = np.eye(len(translation), dtype=np.float32)[text]
    sequence = np.expand_dims(np.concatenate([sequence, np.zeros((1, len(translation)))]), axis=0)
```

```python
    phi_data, window_data, kappa_data, stroke_data = [], [], [], []
    sess.run(vs.zero_states)
    sequence_len = len(args_text) + style_len
    for s in range(1, 60 * sequence_len + 1):
        is_priming = s < prime_len

        print('\r[{:5d}] sampling... {}'.format(s, 'priming' if is_priming else 'synthesis'), end='')

        e, pi, mu1, mu2, std1, std2, rho, \
        finish, phi, window, kappa = sess.run([vs.e, vs.pi, vs.mu1, vs.mu2,
                                               vs.std1, vs.std2, vs.rho, vs.finish,
                                               vs.phi, vs.window, vs.kappa],
                                              feed_dict={
                                                  vs.coordinates: coord[None, None, ...],
                                                  vs.sequence: sequence_prime if is_priming else sequence,
                                                  vs.bias: args.bias
                                              })
```

```python
        if is_priming:
            # Use the real coordinate if priming
            coord = prime_coords[s]
        else:
            # Synthesis mode
            phi_data += [phi[0, :]]
            window_data += [window[0, :]]
            kappa_data += [kappa[0, :]]
            # ---
            g = np.random.choice(np.arange(pi.shape[1]), p=pi[0])
            coord = sample(e[0, 0], mu1[0, g], mu2[0, g],
                           std1[0, g], std2[0, g], rho[0, g])
            coords += [coord]
            stroke_data += [[mu1[0, g], mu2[0, g], std1[0, g], std2[0, g], rho[0, g], coord[2]]]

            if not args.force and finish[0, 0] > 0.8:
                print('Finished sampling!\n')
                break

    coords = np.array(coords)
    coords[-1, 2] = 1.

    return phi_data, window_data, kappa_data, stroke_data, coords
```

The model thus generated can create legible handwriting for styles that aren't cursive. Cursive is still hard, and many times the generated output is not legible.

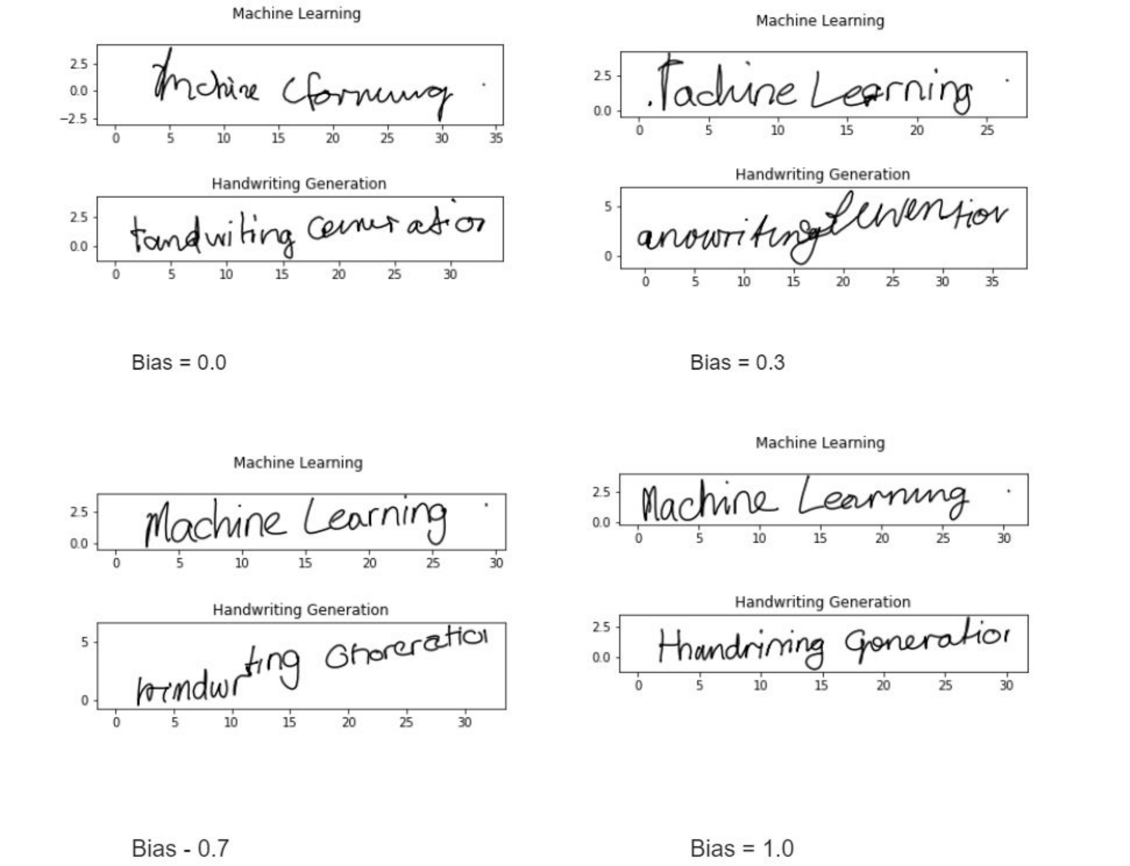
Since the model's MDN cap predicts the pen's (x,y)
(x,y) coordinates by drawing them from a Gaussian distribution, we can modify that distribution to make the handwriting cleaner or messier. by introducing a bias b

$$\sigma_t^j = \exp(\hat{\sigma}(1+b))$$

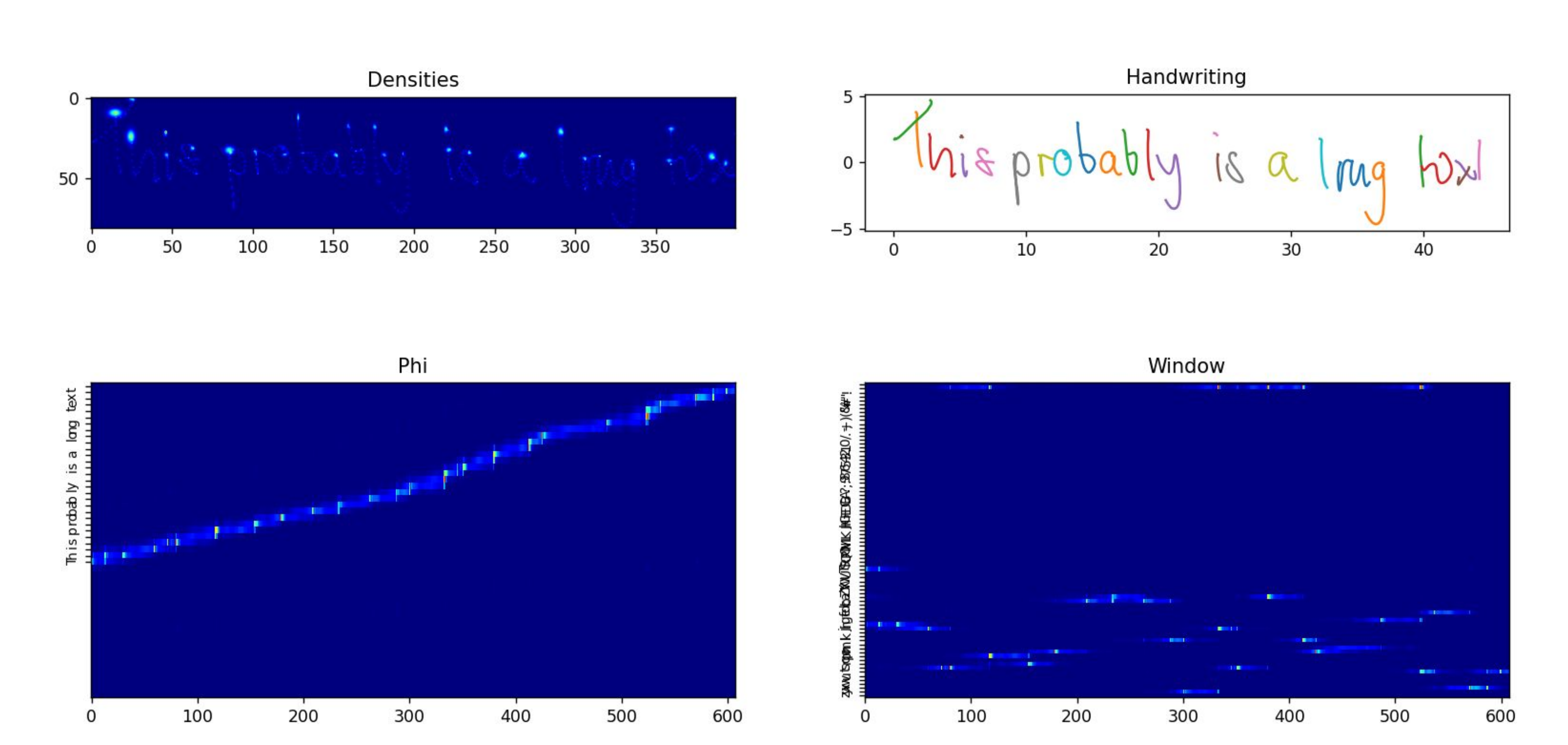
$$\pi_t^j = \frac{\exp(\hat{\pi}(1+b))}{\sum_{j'=1}^{M}\exp(\hat{\pi}(1+b))}$$

On lowering the bias the writing becomes more messier.

OUTPUT - SAMPLING - EFFECT OF BIAS

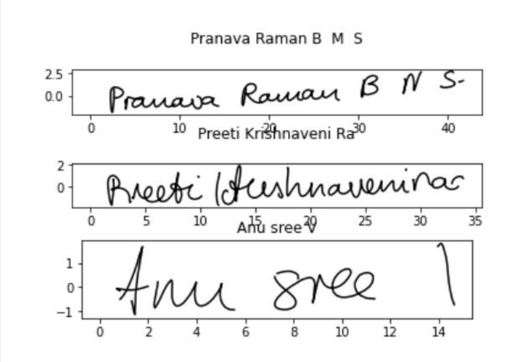


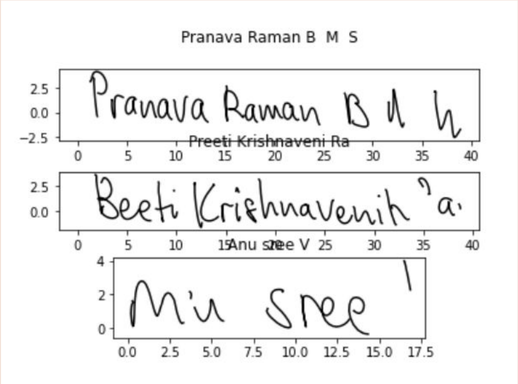
OUTPUT - SAMPLING - ATTENTION WINDOW



EVALUATION & EXPERIMENTS

- Since the output given from the model is a handwriting generated, we couldn't evaluate it directly using any metrics.
- In other implementations like those with GANs, there was a separate model trained that could distinguish between human written and computer generated texts, that also helped in evaluating the performance.
- Key observations were that for cursive handwriting styles, the legibility of the generated texts becomes very poor for long texts. Also it takes longer to generate cursive handwriting than normal ones.





FUTURE WORKS

- Although the generated images are fairly good, it has a lot of shortcomings.
- The model has to be tweaked and trained better to generate long sentences with more examples of such data.
- Models could be trained to evaluate the output generated and better the actual model.
- To use our own handwriting, a module has to be built that can convert our handwriting into the time series data input (xml file).
- The whole module could be deployed in a server to create an endpoint which can be accessed via external application.

Conclusion

- This Project has demonstrated the ability of Long Short-Term Memory recurrent neural networks to generate sequences with complex, long-range structure using next-step prediction.
- It has also introduced a novel convolutional mechanism that allows a recurrent network to condition its predictions on an auxiliary annotation sequence, and used this approach to synthesise diverse and realistic samples of online handwriting.
- Furthermore, it has shown how these samples can be biased towards greater legibility, and how they can be modelled on the style of a particular writer.

REFERENCES

- Generating Sequences With Recurrent Neural Networks: Alex Graves [1308.0850.pdf (arxiv.org)]
- Realistic Handwriting Generation Using Recurrent Neural Networks and Long Short-Term Networks | SpringerLink
- Grzego/handwriting-generation: Implementation of handwriting generation with use of recurrent neural networks in tensorflow. Based on Alex Graves paper (https://arxiv.org/abs/1308.0850). (github.com)
- greydanus/scribe: Realistic Handwriting with Tensorflow (github.com)

Thank you!!