Ch 06: Concept 02

Viterbi parse of a Hidden Markov model

Practical undertanding

First have a look at the notebook on the forward algorihm to be able to better follow this

Introduction

The aim of the Viterbi algorithm is to be able to recreate the sequence of states that better explain a sequence of observations.

Algorithm steps

The algorithm will be explained based on the example of the forward algorithm notebook. For completeness all the necessary data and deifinitions are duplicated.

HMM definitions

$$\mathbf{P_{initial}} = \begin{pmatrix} P_{state_1} \\ P_{state_2} \\ \vdots \\ P_{state_n} \end{pmatrix} (size = N \times 1)$$

$$\mathbf{Transition} = \begin{pmatrix} P_{state_{1_t}|state_{1_{t-1}}} & P_{state_{2_t}|state_{1_{t-1}}} & \cdots & P_{state_{n_t}|state_{1_{t-1}}} \\ P_{state_{1_t}|state_{2_{t-1}}} & P_{state_{2_t}|state_{2_{t-1}}} & \cdots & P_{state_{n_t}|state_{2_{t-1}}} \\ \vdots & \vdots & \vdots & \vdots \\ P_{state_{1_t}|state_{n_{t-1}}} & P_{state_{2_t}|state_{n_{t-1}}} & \cdots & P_{state_{n_t}|state_{n_{t-1}}} \end{pmatrix} (size = N \times N)$$

$$\mathbf{Emission} = \begin{pmatrix} P_{observation_1|state_1} & P_{observation_2|state_1} & \cdots & P_{observation_k|state_1} \\ P_{observation_1|state_2} & P_{observation_2|state_2} & \cdots & P_{observation_k|state_2} \\ \vdots & \vdots & \vdots & \vdots \\ P_{observation_1|state_n} & P_{observation_2|state_n} & \cdots & P_{observation_k|state_n} \end{pmatrix} (size = N \times K)$$

Example data from the code

$$\mathbf{P_{initial}} = \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix}$$

$$\mathbf{Transition} = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix}$$

$$\mathbf{Emission} = \begin{pmatrix} 0.5 & 0.4 & 0.1 \\ 0.1 & 0.3 & 0.6 \end{pmatrix}$$

The states are the weather conditions rain R (1^{st} state) and sunny S (2^{nd} state). The observations are some specific person cleaning C (1^{st} column of $Emission$), shopping Sh (2^{nd} column of $Emission$) and going for a walk W (3^{rd} column of $Emission$), conditional on the weather.

Let's say we want to calculate the probability of observing that person first going shopping the first day and then going for a walk the following day.

Note that in the example State-> S, Observation -> O

Step 1 - Initialize

This step is the same as step 1 of the forward algorithm

Using the same example as in the forward algorithm notebook we end up with a Viterbi vector that holds the probabilities of being in each current state conditional that the current state transitions from a specific previous state (this will be clearer in the next steps).

$$\mathbf{Viterbi_{shop}} = \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \odot \begin{pmatrix} 0.4 \\ 0.3 \end{pmatrix} = \begin{pmatrix} 0.24 \\ 0.12 \end{pmatrix}$$

The following code is used for the initialisation

```
def forward_init_op(self):
    obs_prob = self.get_emission(self.obs)
    fwd = tf.multiply(self.initial_prob, obs_prob)
    return fwd
```

We also initialise the `backpts` tensor (see step 3 for details)

```
backpts = np.ones((hmm.N, len(observations)), 'int32') * -1
```

$$\mathbf{backpts} = \begin{pmatrix} -1 & -1 \\ -1 & -1 \end{pmatrix}$$

Step 2 - Decode

2a

In this step we are going to go to the next observation and calculate the probabilities of being a current state given an observation AND a previous state.

$$\{P(S_{n_t}|O_t \wedge S_{i_{t-1}})\}_i = \{P(S_{i_{t-1}}) \times P(Sn_t|S_{i_{t-1}}) \times P(O_t|Sn_t)\}_i$$

$P(S_{i_{t-1}})$ is given by the previous decode step result from the $Viterbi_{t-1}$ vector

2b

We calculate the maximum of the set of probablities to find the maximum probability for each state that corresponds to the most likely previous state

$$P(S_{n_t}|O_t \wedge S_{max_{t-1}}) = max\{P(S_{i_{t-1}}) \times P(Sn_t|S_{i_{t-1}}) \times P(O_t|Sn_t)\}_i$$

Note that $P(S_{i_{t-1}})$ comes from the Viterbi vector of the previous time step $Viterbi_{t-1}$, i.e.

$$P(S_{i_{t-1}}) = max\{P(S_{i_{t-2}}) \times P(Sn_{t-1}|S_{i_{t-2}}) \times P(O_{t-1}|Sn_{t-1})\}_i$$

2c

The result is a vector with length equal to the number of states.

$$\mathbf{Viterbi_t} = \begin{pmatrix} P(S_{1_t}|O_t \wedge S_{max_{t-1}}) \\ P(S_{2_t}|O_t \wedge S_{max_{t-1}}) \\ \vdots \\ P(S_{n_t}|O_t \wedge S_{max_{t-1}}) \end{pmatrix}$$

In the code this is the `viterbi` tensor in the `decode_op` function. The operations of steps 2a-2c are performed in the `decode_op` function:

```
def decode_op(self):
    transitions = tf.matmul(self.viterbi, tf.transpose(self.get_emission(se
lf.obs)))
    weighted_transitions = transitions * self.trans_prob
    viterbi = tf.reduce_max(weighted_transitions, 0)
    return tf.reshape(viterbi, tf.shape(self.viterbi))
```

In the example we are using the code calculations are as follows:

`transitions = tf.matmul(self.viterbi, tf.transpose(self.get_emission(self.obs)))` :

$$\mathbf{transitions_{walk}} = \begin{pmatrix} 0.24 \\ 0.12 \end{pmatrix} \times \begin{pmatrix} 0.1 \\ 0.6 \end{pmatrix}^T = \begin{pmatrix} 0.024 & 0.144 \\ 0.012 & 0.072 \end{pmatrix}$$

`weighted_transitions = transitions * self.trans_prob` :

$$\mathbf{weightedtransitions_{walk}} = \begin{pmatrix} 0.024 & 0.144 \\ 0.012 & 0.072 \end{pmatrix} \odot \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix} = \begin{pmatrix} 0.0168 & 0.0432 \\ 0.0048 & 0.0432 \end{pmatrix}$$

At this point we have calculated $\{P(S_{n_t}|O_t \wedge S_{t-1_i})\}_i$ from step 2a. Where each column in the $\mathbf{weightedtransitions_{walk}}$ is $\{P(S_{1_t}|O_t \wedge S_{t-1_i})\}_i$, $\{P(S_{2_t}|O_t \wedge S_{t-1_i})\}_i$, etc...

`viterbi = tf.reduce_max(weighted_transitions, 0)`

$$\mathbf{viterbi_{walk}} = max(\mathbf{weightedtransitions_{walk}}) = \begin{pmatrix} 0.0168 & 0.0432 \end{pmatrix}$$

This calculation corresponds to step 2b, where we calculate $P(S_{n_t}|O_t \wedge S_{t-1_{max}})$, each element is $P(S_{1_t}|O_t \wedge S_{t-1_{max}})$, $P(S_{2_t}|O_t \wedge S_{t-1_{max}})$, etc..

`viterbi=tf.reshape(viterbi, tf.shape(self.viterbi))`

$$\mathbf{viterbi_{walk}} = \begin{pmatrix} 0.0168 & 0.0432 \end{pmatrix}^T = \begin{pmatrix} 0.0168 \\ 0.0432 \end{pmatrix}$$

Step 3 - Backstep

Before moving to the next observation we want to record the most probable PREVIOUS state given each CURRENT state.

We use the Viterbi vector BUT NOT THE CURRENT ONE at time=t but the one from the previous step $Viterbi_{t-1}$

We want to calculate the most probable previous state that can transition to each current state.

$$\mathbf{MostProbableState_{t-1}|S_{n_t}} = argmax\{P(S_{1_{t-1}}) \times P(S_{n_t}|S_{1_{t-1}}), P(S_{2_{t-1}}) \times P(S_{n_t}|S_{2_{t-1}}), \ldots\}$$

Note that the $P(S_{1_{t-1}})$ comes from $Viterbi_{t-1}$, $P(S_{n_t}|S_{1_{t-1}})$ comes from the transmission matrix.

This operation is performed in the `backpt_op` function.

```
def backpt_op(self):
    back_transitions = tf.matmul(self.viterbi, np.ones((1, self.N)))
    weighted_back_transitions = back_transitions * self.trans_prob
    return tf.argmax(weighted_back_transitions, 0)
```

The results of this calculation is saved in the `backpts` tensor. This is an example

$$\mathbf{backpts} = \begin{pmatrix} -1 & 0 & \dots & 2 \\ -1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -1 & 1 & \dots & 0 \end{pmatrix} (size = N \times number of observations)$$

The first column is a dummy column. The 0 in row 1, column 2 means that the most probable PREVIOUS state when the CURRENT state is 0 is State 0, the 1 in rown N, column 2 means that the most probable PREVIOUS state when the CURRENT state is N is State 1.

In the example we calculate the column of $\mathbf{backpts}$ as

$$\mathbf{MostProbablePreviousState|CurrentState_{Rain}} = argmax\{P(S_{Rain_{t-1}}) \times P(S_{Rain_t}|S_{Rain_{t-1}}), P(S_{Sunny_t}$$
$$= argmax\{0.24 \times 0.7, 0.12 \times 0.4\} = argmax\{0.168, 0.048\} = 0$$

So if the current state is Rain the most probable previous state was Rain Note that $P(S_{Rain_{t-1}})$ and $P(S_{Sunny_{t-1}})$ come from $Viterbi_{t-1} = Viterbi_{shop}$ vector

$$\mathbf{MostProbablePreviousState|CurrentState_{Sunny}} = argmax\{P(S_{Rain_{t-1}}) \times P(S_{Sunny_t}|S_{Rain_{t-1}}), P(S_{Sun}$$
$$= argmax\{0.24 \times 0.3, 0.12 \times 0.6\} = argmax\{0.072, 0.072\} = 0$$

So if the current state is Sunny the most probable previous state was Rain. Note that $P(S_{Rain_{t-1}})$ and $P(S_{Sunny_{t-1}})$ come from $Viterbi_{t-1} = Viterbi_{shop}$ vector

We update $\mathbf{backpts}$

$$\mathbf{backpts} = \begin{pmatrix} -1 & 0 \\ -1 & 0 \end{pmatrix}$$

Step 4 - Final step

When we reach the final calculation we calculate the `Viterbi` and `backpts` as usual. We calculate the most probable final state as

$$\mathbf{MostProbableFinalState} = argmax\{\mathbf{Viterbi_i}\}$$

Once we have this we step through the `backpts` on the row index that corresponds to our final state and that's the final result.

This is done in the following code section

```
tokens = [viterbi[:, -1].argmax()]
for i in range(len(observations) - 1, 0, -1):
    tokens.append(backpts[tokens[-1], i])
```

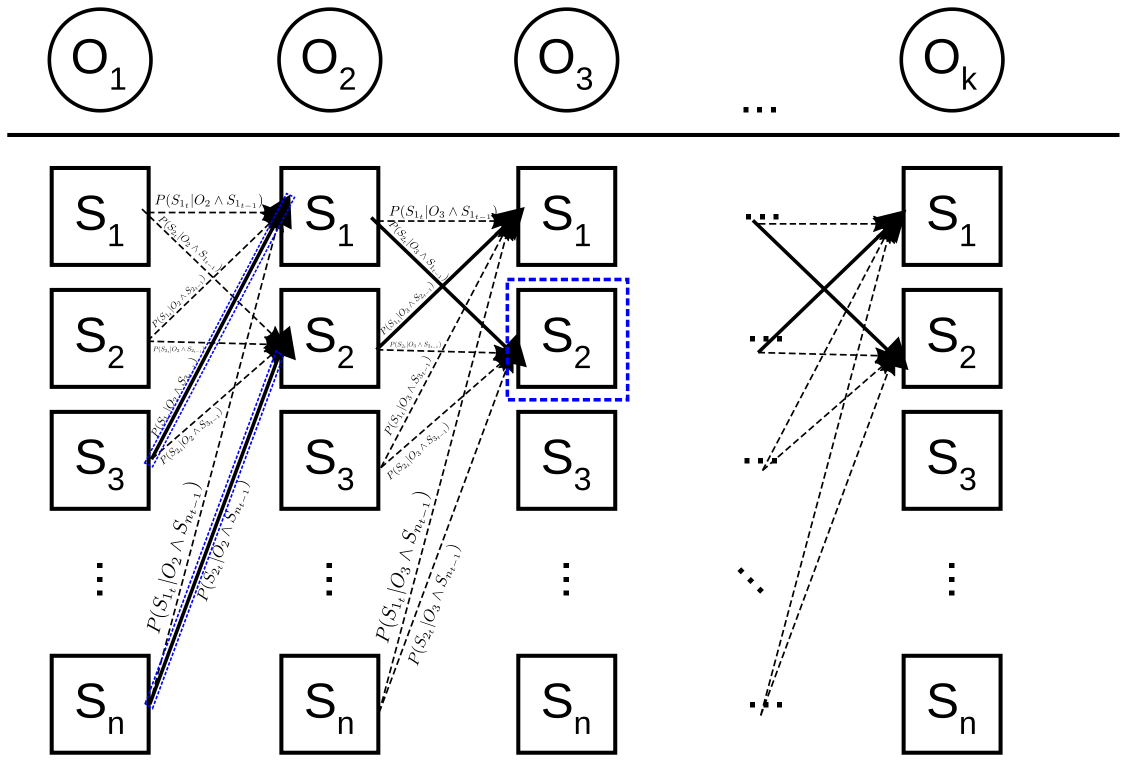
In the example we calculate

$$\mathbf{argmax(viterbi_{walk})} = argmax \begin{pmatrix} 0.0168 \\ 0.0432 \end{pmatrix} = 1$$

This means the most likely current state is Sunny. This corresponds to row index 1 of the $\mathbf{backpts}$ (which is the second row in `python` array indexing) which holds the most likely previous state which is 0 => Rain.

The final answer for our example is that the most likely sequence of states that explain the sequence of observations is: Rain, Sunny or {0 ,1}

Graphical explanation



The dotted arrows are the probabilities calculated at step 2a, the bold arrows are the maximum probabilities corresponding to the current states calculated in step 2b and saved in the $\mathbf{Viterbi_t}$ vector of step 2c.

Let's say we want to calculate the result of step 3 for state 2 highlighted with the blue dotted square (under observation 3). We want to calculate the most likely previous state that resulted in a current state. To do this we need the probability of having each previous states which is represented by the bold arrows at the BACK of the previous states, highlighted by the blue boxes. We need to also take into account whether the PREVIOUS state will transition to the CURRENT state. We do this by multiplying these probabilities, which are in $\mathbf{Viterbi_{t-1}}$ with the transition probabilities. So for each current state, for example state 2 the result is a vector given by

$$\mathbf{MostProbableStateVector_{t-1}|S_{2_t}} = \{P(S_{1_{t-1}}) \times P(S_{2_t}|S_{1_{t-1}}), P(S_{2_{t-1}}) \times P(S_{2_t}|S_{2_{t-1}}), \ldots\}$$

Calculating the argmax of this we get the index of the most likely PREVIOUS state for the current state 2

$$\mathbf{MostProbableState_{t-1}|S_{2_t}} = argmax\{P(S_{1_{t-1}}) \times P(S_{2_t}|S_{1_{t-1}}), P(S_{2_{t-1}}) \times P(S_{2_t}|S_{2_{t-1}}), \ldots\}$$

We need to repeat this for all the current states (not just state 2). This is covered in detail in step 3.

Import TensorFlow and Numpy

In [1]:

```
import numpy as np
import tensorflow as tf
```

Create the same HMM model as before. This time, we'll include a couple additional functions.

In [2]:

```
# initial parameters can be learned on training data
# theory reference https://web.stanford.edu/~jurafsky/slp3/8.pdf
# code reference https://phvu.net/2013/12/06/sweet-implementation-of-viterbi-in
class HMM(object):
    def __init__(self, initial_prob, trans_prob, obs_prob):
        self.N = np.size(initial_prob)
        self.initial_prob = initial_prob
        self.trans_prob = trans_prob
        self.obs_prob = obs_prob
        self.emission = tf.constant(obs_prob)
        assert self.initial_prob.shape == (self.N, 1)
        assert self.trans_prob.shape == (self.N, self.N)
        assert self.obs_prob.shape[0] == self.N
        self.obs = tf.placeholder(tf.int32)
        self.fwd = tf.placeholder(tf.float64)
        self.viterbi = tf.placeholder(tf.float64)

    def get_emission(self, obs_idx):
        slice_location = [0, obs_idx]
        num_rows = tf.shape(self.emission)[0]
        slice_shape = [num_rows, 1]
        return tf.slice(self.emission, slice_location, slice_shape)

    def forward_init_op(self):
        obs_prob = self.get_emission(self.obs)
        fwd = tf.multiply(self.initial_prob, obs_prob)
        return fwd

    def forward_op(self):
        transitions = tf.matmul(self.fwd, tf.transpose(self.get_emission(self.o
        weighted_transitions = transitions * self.trans_prob
        fwd = tf.reduce_sum(weighted_transitions, 0)
        return tf.reshape(fwd, tf.shape(self.fwd))

    def decode_op(self):
        transitions = tf.matmul(self.viterbi, tf.transpose(self.get_emission(se
        weighted_transitions = transitions * self.trans_prob
        viterbi = tf.reduce_max(weighted_transitions, 0)
        return tf.reshape(viterbi, tf.shape(self.viterbi))

    def backpt_op(self):
        back_transitions = tf.matmul(self.viterbi, np.ones((1, self.N)))
        weighted_back_transitions = back_transitions * self.trans_prob
        return tf.argmax(weighted_back_transitions, 0)
```

Define the forward algorithm from Concept01.

In [3]:

```
def forward_algorithm(sess, hmm, observations):
    fwd = sess.run(hmm.forward_init_op(), feed_dict={hmm.obs: observations[0]})
    for t in range(1, len(observations)):
        fwd = sess.run(hmm.forward_op(), feed_dict={hmm.obs: observations[t], h
    prob = sess.run(tf.reduce_sum(fwd))
    return prob
```

Now, let's compute the Viterbi likelihood of the observed sequence:

In [4]:

```
def viterbi_decode(sess, hmm, observations):
    viterbi = sess.run(hmm.forward_init_op(), feed_dict={hmm.obs: observations[
    backpts = np.ones((hmm.N, len(observations)), 'int32') * -1
    for t in range(1, len(observations)):
        viterbi, backpt = sess.run([hmm.decode_op(), hmm.backpt_op()],
                                    feed_dict={hmm.obs: observations[t],
                                               hmm.viterbi: viterbi})
        backpts[:, t] = backpt
    tokens = [viterbi[:, -1].argmax()]
    for i in range(len(observations) - 1, 0, -1):
        tokens.append(backpts[tokens[-1], i])
    return tokens[::-1]
```

Let's try it out on some example data:

In [5]:

```
if __name__ == '__main__':
    states = ('Healthy', 'Fever')
#     observations = ('normal', 'cold', 'dizzy')
#     start_probability = {'Healthy': 0.6, 'Fever': 0.4}
#     transition_probability = {
#         'Healthy': {'Healthy': 0.7, 'Fever': 0.3},
#         'Fever': {'Healthy': 0.4, 'Fever': 0.6}
#     }
#     emission_probability = {
#         'Healthy': {'normal': 0.5, 'cold': 0.4, 'dizzy': 0.1},
#         'Fever': {'normal': 0.1, 'cold': 0.3, 'dizzy': 0.6}
#     }
    initial_prob = np.array([[0.6], [0.4]])
    trans_prob = np.array([[0.7, 0.3], [0.4, 0.6]])
    obs_prob = np.array([[0.5, 0.4, 0.1], [0.1, 0.3, 0.6]])
    hmm = HMM(initial_prob=initial_prob, trans_prob=trans_prob, obs_prob=obs_pr

#     observations = [0, 1, 1, 2, 1]
    observations = [1,2]
    with tf.Session() as sess:
        prob = forward_algorithm(sess, hmm, observations)
        print('Probability of observing {} is {}'.format(observations, prob))

        seq = viterbi_decode(sess, hmm, observations)
        print('Most likely hidden states are {}'.format(seq))
```

```
Probability of observing [1, 2] is 0.10799999999999998
Most likely hidden states are [0, 1]
```