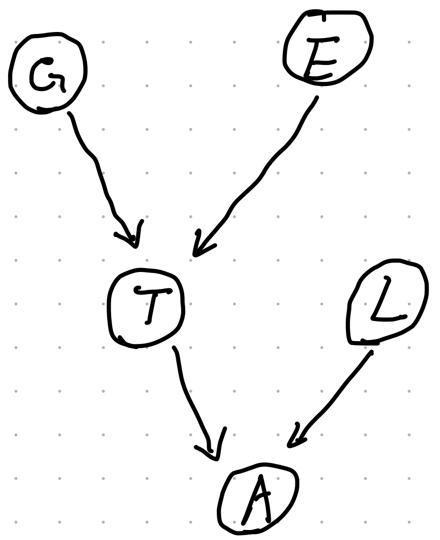
CMPT 419 Assignment 3

Yu Ke
301414915

1. Graphical Models

1. Boolean : A Continuous : E , T

   Discrete : L , G ,



2. $P(A, L, G, E, T)$

   $= P(A \mid T, L) \cdot P(L) \cdot P(T \mid G, E) \cdot P(G) \cdot P(E)$

3. ① For G and L with no parent.

   Use educated guess :

   $P(L=0) = 0.5 \quad P(L=u) = 0.5$

$P(G=l) = 0.5 \quad P(G=d) = 0.5$

② For E, it is continuous with no parent.

Use linear Gaussians. We assume the GDP of BC is k millions, the variance is σ

$\Rightarrow E: \quad N(k, \sigma^2)$

③ For T, it is continuous valued random variables and has the parents G and E

Use Linear Gaussian:

$$p(x_i|pa_i) = N\left(x_i \,\middle|\, \sum_{j \in pa_i} x_{ij} x_j + b_i, \, v_i\right)$$

$$\Rightarrow p(T=t \mid G=l, E=e)$$

$$= N(t \mid x_g l + b_g + x_e e + b_e, \, v_t)$$

$$p(T=t \mid G=d, E=e)$$

$$= N(t \mid x_g d + b_g + x_e e + b_e, \, v_t)$$

④ For A, it is boolean with parents T and L.

$$P(y=1 \mid x_1, \cdots, x_m) = \sigma\left(w_0 + \sum_{i}^{n} w_i x_i\right) = \sigma(W^T X)$$

$$P(A \mid L, T, E) = \frac{1}{1+\exp(-W^T X)}$$

4. $x \in \{x_1, x_2, \cdots, x_N\}$

$$x_n = \{a_n, l_n, g_n, e_n, t_n\}$$

$$L(X \mid \theta) = P(x_1 \mid \theta) \cdot p(x_2 \mid \theta) \cdots p(x_N \mid \theta) = \prod_{i=1}^{N} P(X_i \mid \theta)$$

By factorization: $P(X) = \prod_{k=1}^{k} P(X_k \mid Pa_k)$

$$\Rightarrow L(X \mid \theta) = \prod_{k=1}^{K} \prod_{i=1}^{N} P(X_i^k \mid Pa_i^k, \theta)$$

$\Rightarrow$ The maximum likelihood is:

$$\arg\max_{\theta} \prod_{i=1}^{N} P(X_i \mid \theta) \quad \arg\max_{\theta} \prod_{k=1}^{K} \prod_{i=1}^{N} P(X_i^k \mid Pa_i^k, \theta)$$

2.

- $$h_j^{<t>} = z_j h_j^{<t-1>} + (1-z_j)\cdot \tilde{h}_j^{<t>}$$

If $h_j^{<t>}$ is similar to its old state, then $z_j \to 1$.

- If z_j is close to zero, then

$h_j^{<t>}$ is close to $\tilde{h}_j^{<t>}$. If $r_j \to 0$, then:

$$\tilde{h}_j^{<t>} = \phi\left([Wx]_j + [U(r \odot h_{<t-1>})]_j\right) \quad (r_j \to 0)$$

$$= \phi([Wx]_j)$$

From above, the $h_j^{<t>}$ will be reset with current input x. But, for $r_i \neq r_j$, $r_i \neq 0$, $r_i \in r$, it still remains some previous information.

3 Reinforcement Learning

3.1

```python
fc1 = tf.layers.dense(
    inputs=self.state_data,
    units=h1_size,
    activation=tf.nn.tanh,  # tanh activation
    kernel_initializer=tf.random_normal_initializer(mean=0, stddev=0.3),
    bias_initializer=tf.constant_initializer(0.1),
    name='fc1'
)
# fc2
fc2 = tf.layers.dense(
    inputs=fc1,
    units=h2_size,
    activation=tf.nn.tanh,  # tanh activation
    kernel_initializer=tf.random_normal_initializer(mean=0, stddev=0.3),
    bias_initializer=tf.constant_initializer(0.1),
    name='fc2'
)
all_act = tf.layers.dense(
    inputs=fc2,
    units=a_size,
    activation=None,
    kernel_initializer=tf.random_normal_initializer(mean=0, stddev=0.3),
    bias_initializer=tf.constant_initializer(0.1),
    name='all_act'
)
```

3.2

```python
# corresponding to action data
self.all_act_prob = tf.nn.softmax(all_act, name='act_prob')  # use softmax to convert to probability
```

3.3

```python
with tf.name_scope('loss'):
    # to maximize total reward (log_p * R) is to minimize -(log_p * R), and the tf only have minimize(loss)
    neg_log_prob = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=all_act, labels=self.action_data)
    self.loss = tf.reduce_mean(neg_log_prob * self.advantage_data)  # reward guided loss
### -----------------------------------------------------------------------
```

3.4

```python
def compute_advantage(j, r, gamma):
    ### Part f) Advantage computation
    """ Computes the advantage function from data
        Inputs:
            j     -- list of time steps
                     (eg. j == [0, 1, 2, 3, 0, 1, 2, 3, 4, 5] means that there
                      are two episodes, one with four time steps and another with
                      6 time steps)
            r     -- list of rewards from episodes corresponding to time steps j
            gamma -- discount factor

        Output:
            advantage -- vector of advantages correponding to time steps j
    """
    subList=[r[0]]
    subAdvantage=[]
    advantage=[]
    length = len(j)
    for i in range(1, length):
        if j[i]!=0:
            subList.append(r[i])
        if i==length-1 or j[i+1]==0:
            running_add=0
            for t in reversed(range(len(subList))):
                running_add = running_add * gamma + subList[t]
                subAdvantage.insert(0, running_add)
            advantage+=subAdvantage
            subList=[]
            subAdvantage=[]
        if j[i]==0:
            subList.append(r[i])

    mean = np.mean(advantage)
    std = np.std(advantage)
    advantage = (advantage - mean)/std
    np.array(advantage)
    return advantage
```

3.5

```
prob_weights=sess.run(myAgent.all_act_prob, feed_dict={myAgent.state_data: s[np.newaxis, :]})
a = np.random.choice(range(prob_weights.shape[1]), p=prob_weights.ravel())
### ----------------------------------------

# Get reward for taking an action, and store data from the episode
s1,r,d,_ = env.step(a) #
history.append([j,s,a,r,s1])
s = s1

if d == True: # Update the network when episode is done
    # Update network every "update_frequency" episodes
    if i % update_frequency == 0 and i != 0:
        # Compute advantage
        history = np.array(history)
        advantage = compute_advantage(history[:,0], history[:,3], gamma)

        ### --- Part g) Perform policy update ---
        sess.run(myAgent.update_batch, feed_dict={
            myAgent.state_data: np.vstack(history[:,1]),
            myAgent.action_data: np.array(history[:,2]),
            myAgent.advantage_data: advantage
```

```
# myAgent = agent(...)
myAgent = agent(lr=0.02, a_size=env.action_space.n, s_size=env.observation_space.shape[0], h1_size= 8, h2_size=8)
```

3.6 It is shown in the video named ***3.6_YuKe.mp4***

3.7 x-label is the number of batch, each batch has 5 episode