1. a. \[-\sum_{m=1}^{30} \log (A(x_m) / \sum_{x'} A(x'))\]

   where \(x'\) ranges over all strings, or equivalently, over all
   strings accepted by \(A\) (since \(A(x') = 0\) for the other).

   b. reduces BIAS, but at the cost of greater VARIANCE

   c. Option 1: Use a regularization term that penalizes the
      difference between the new parameters and the old ones.
      Thus, training will be reluctant to change parameters
      and will only do this if it substantially helps to
      predict the training examples of Trumpy English.

      Option 2: Early stopping before the parameters have
      converged.

      Either way: you have a hyperparameter to tune. In option 1, it
      is the regularization constant. In option 2, it is the number
      of epochs. Either way, you can select this hyperparameter using
      a dev corpus of Trumpy English.

      Notice that choosing the hyperparameter for option 2 is very
      efficient -- you just keep training until the cross-entropy on
      the dev corpus starts getting worse. For example, you evaluate
      on the dev course after 15 epochs of training, 16 epochs of
      training, etc. It’s cheap to find out whether 16 epochs gives a
      good model because you don’t have to train a new model from
      scratch: just start at the 15-epoch model and train for one more
      epoch.

2. a. The matrix \(V\), the vector \(\theta\), the word embeddings, and the
    initial hidden state vector \(h_0\).

   b. The number of rows is \(d\). The number of columns is \((1 + d +
      dimensionality of the word embeddings)\).

   c. We’ll say that the hidden vectors \(h_j\) represent the upper layer,
      so (1) can be unchanged. Change equation (2) to replace \(w_j\)
      with \(g_j\), where \(g_j\) is a hidden vector at the lower layer.
      Add equation (3):
      \[g_j = \sigma(U [1; g_{j-1}; w_j])\]

   d. This is a tricky question! The goal is to see how an RNN
      can track properties of the input over time, just like the FSA
      in the previous question. We saw in class how nodes in neural
      nets can implement AND/OR operations, and we’ll do that here.

      Note that the typesetting of this answer omits the vector arrow,
      so it does not distinguish properly between the word \(w_j\) and
      its embedding \(w_j\) (which should have a vector arrow).

      In this answer, we will assume that a \(d\)-dimensional vector
      has indices 1...\(d\), as indicated in the last line of the question.
      This is common in mathematical notation, in contrast to Python’s
      indices 0...(\(d-1\)). (We accepted either style in your answer, though.)

      We can see that
      \[h_j[3] = \sigma(v \cdot [1; h_{(j-1)}; w_j])\]
      if \(v\) denotes row 3 of matrix \(V\).

      Therefore, we need
      \[v \cdot [1; h_{(j-1)}; w_j]\]
to be strongly negative if $w_j = \text{Trump}$

or if $h_{(j-1)[3]}$ is close to 0 (meaning that $w_i = \text{Trump}$

close to 0)

for some $i < j$),

but it should be strongly positive otherwise.

We shouldn’t pay attention to the other elements of $h_{(j-1)}$,

so we can set $v[2, 3, 5, 6, \ldots, d+1]$ to 0.

Then we have

$$v \cdot [1; h_{(j-1)}; w_j] = v[1] + v[4] \cdot h_{(j-1)[3]} + v[d+2, \ldots] \cdot w_j$$

We want $v[4]$ to be strongly positive so that if $h_{(j-1)[3]}$

is close to 1 (meaning that we haven’t seen Trump yet),

then the dot product will be strongly positive and thus

$h_j[3]$ will also be close to 1.

However, this should be overridden if $w_j = t$ in

which case we want $v[d+2, \ldots] \cdot w_j$ to be negative enough

to drive the dot product negative. We can do this by

setting $v[d+2, \ldots] = -c \cdot t$ for some large positive $c$.

That ensures that $v[d+2, \ldots] \cdot w_j$ is much more negative

when $w_j = \text{Trump}$ than for any other $w_j$ (because

when $w_j = \text{Trump}$, $t \cdot w_j$ is positive and larger than

for any other $w_j$, according to the problem).

So, choose a large $c$, and then solve for $v[1]$ and $v[4]$ to ensure that the dot product is (for example)

$< -5$ when it is supposed to be strongly negative and also $> 5$ when it is supposed to be strongly positive.

(This ensures that $h_j[3]$ will be $< 0.01$ and $> 0.99$, respectively.) This is a system of inequalities

in two variables. If there is no solution, then make

$c$ larger so that there is a solution.

(Also, the initial hidden state vector $h_0$ should have $h_0[3] = 1$,

to make the setup work.)

To ensure that almost every sentence contains Trump, we need to ensure that $w_{(j+1)} = \text{EOS}$ is improbable if $h_0[3] = 1$. We can do this by setting $e_{(j+1)[4]}$ to be very negative.

Note that this is just a demonstration that the architecture has the ability to achieve the desired behavior, with appropriate parameters. In practice, we will rely on training to find good parameters.