1 Find and Load Training Text

(a) We chose to use the text "A Christmas Carol" by Charles Dickens (Downloaded from Gutenberg Project) to train our RNN. This text contains 161,595 characters including spaces.

(b) To create character level representations for this model, we read every character from the file into python. And then found the number of unique characters in the training file, then created the one-hot encoding using only these unique characters.

2 Implement RNN

(a) Back-Propagation Through Time
i. The activation for the hidden layer at time step t is:
$h^{(t)} = \tanh(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h)$, where the tanh is applied element wise.
The activation for the output layer at time step t is $z^{(t)} = softmax(W_{ho}h^{(t)} + b_o)$

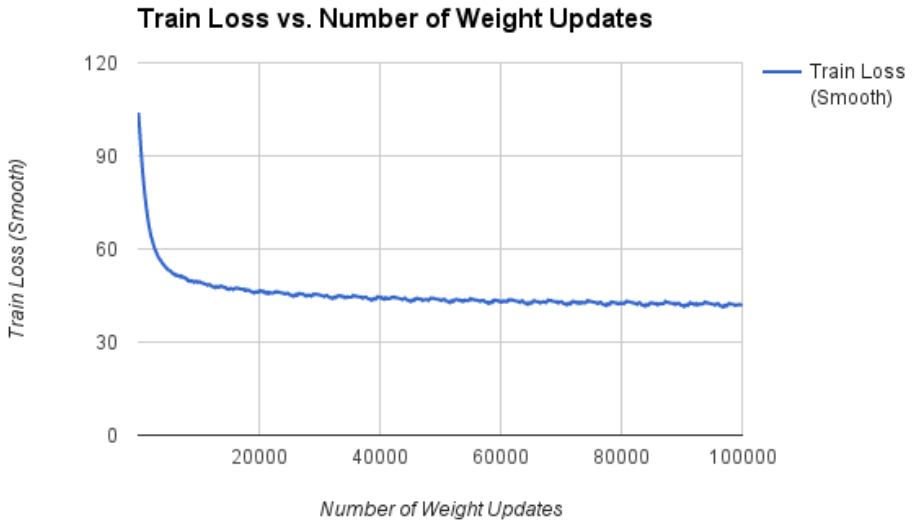
ii. Equations for weight update rule:

$W_{ho} = W_{ho} + \alpha(z^{(t)} - y^{(t)}) \otimes h^{(t)}$ Where we take the outer product of $(z^{(t)} - y^{(t)})$ and $h^{(t)}$
$W_{hh} = W_{hh} + \alpha((1 - h^{(t)^2}) \circ W_{hy}(z^{(t)} - y^{(t)}))\Pi_{j=k+1}^{t}(W_{HH}^{T} diag(1 - h^{(j-1)2}))h^{(t-1)}$
$W_{xh} = W_{xh} + \alpha((1 - h^{(t)^2}) \circ W_{hy}(z^{(t)} - y^{(t)}))\Pi_{j=k+1}^{t}(W_{HH}^{T} diag(1 - h^{(j-1)2}))x^{(t)}$

These equations are the weight update for looking at only one timestep. In our program we calculate the weight change for 15 timesteps and then use the autograd weight update method to update the three weights using these 15 changes. This question asked for the weight update at a single timestep which is why our equations do not sum over t.

(b) Network Training

(i) For our recurrent net architecture, we used one hidden layer with 100 hidden units, we set our sequence size to 25 so that we update the weights after determining the gradient for a 25 character length sequence. We set the learning rate at 0.1. We also used clipping of -5 and 5 in order to stop exploding gradients. We only look back one timestep when computing the derivatives in order to simplify code and speed up training. This is why there is only one for loop over time in our training function. We also use adagrad update as described in "The Unreasonable Effectiveness of Recurrent Neural Networks" sample code. Our network uses a one-hot encoding of only the characters present in the training dataset. Which for the text we used was 66 unique characters. This encoding is used for both the input and output of the network. We printed out the loss every 100 weight updates and created a graph of loss vs iteration which is shown below. Each weight update occurs after looking at 25 characters and our training set had 161595 characters so one epoch is after 6463 weight updates. In the graph below we plot the smooth loss against number of weight updates. The smooth loss was calculated the same way as the sample in "The Unreasonable Effectiveness of Recurrent Neural Networks". Each time it is updated we take 99.9% of the smooth loss from before and add 0.1% of the new loss. We stop training after 50 epochs but in practice this takes quite a while and we always manually stopped before then.



(ii) For this question, we had our network generate 100 characters at different iterations of weight updates. In order to generate these characters we inputted a character from the dataset and then sampled from the probability function of our network output to determine the resulting character. This new character was then used as the next input to the network.

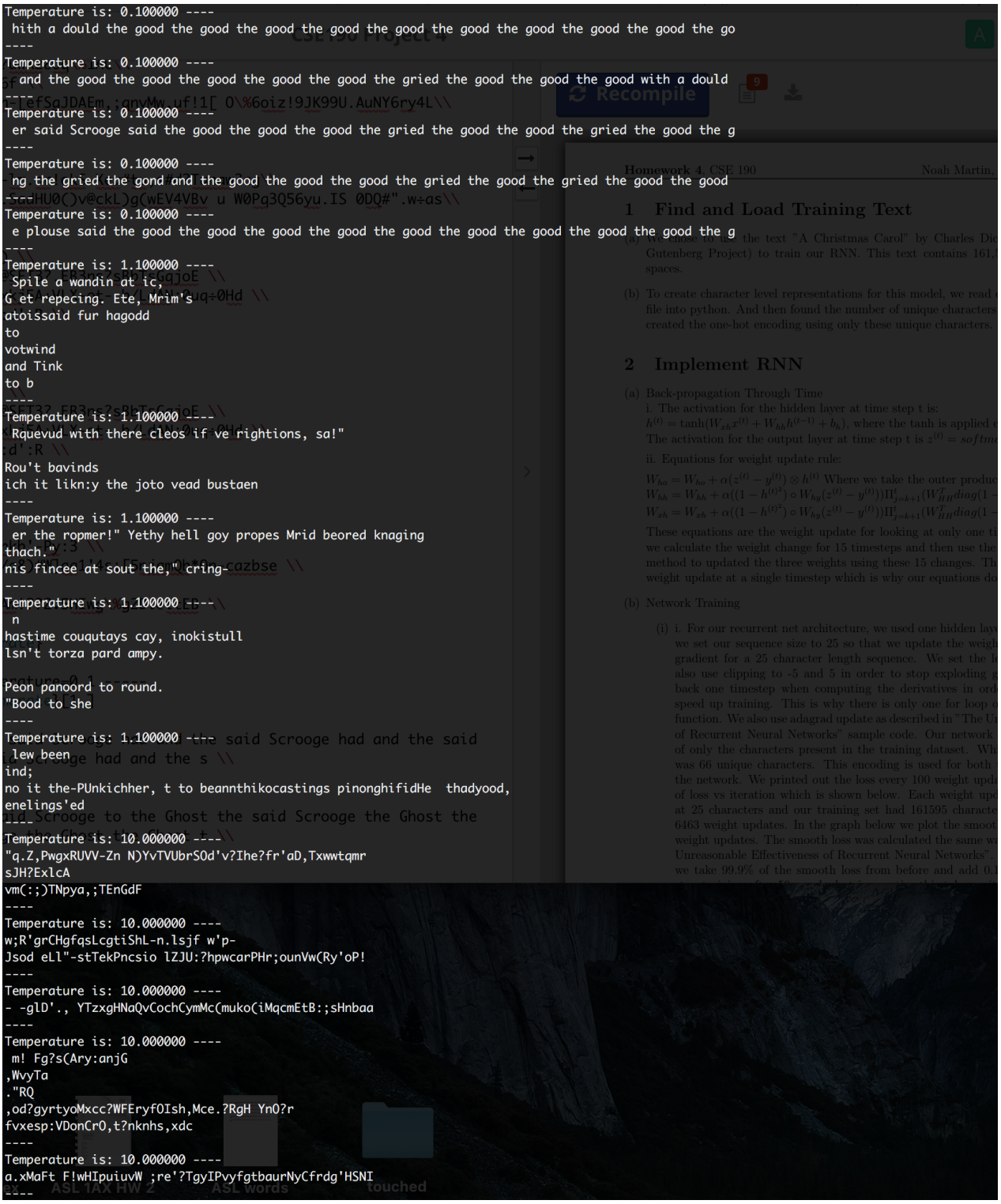
The image below shows the text output of our network after 100, 1000, 10000, 20000, and 60000 weight updates.

As the picture shows, the network initially does not output much that resembles English, but after only a few iterations it is generating a few English words. By the 60000 weight update it has even learned words specific to the input text, which was "A Christmas Carol", such as "Scrooge".

```
iteration: 100 ----
  aude

anor wpy ;iit e
, oc lco
-
,lrs o Mogealer i
rspa auTle astdligut d
waas shllhapt a
----
iteration: 1000 ----
 ejwlebd tirli, had
is Scrercilp led e
Tses,
Iy ngigcrlede, cxroofe, Bp hisu besse bxy, has c thls
----
iteration: 10000 ----
 ed fister, and evely that head was mich oftherlill Totered (nait inth, has, Scr
olr.
00 or iteration == 20000 or iteration == 60000:
"He rigy; a-k
----
iteration: 20000 ----
 phrmouren! Where anot's thinf the piadren, ounithed to ho robsevteander, andire
," haganke th, chis a
----
iteration: 60000 ----
 d ins.
"I had tithougo this a mang lithim bowited Rowes. It have lone goad. Scrooge die
d of a
I'll
----
```

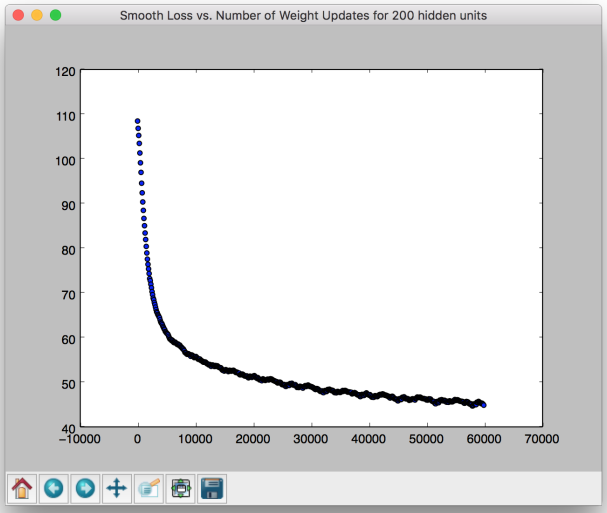
(iii)

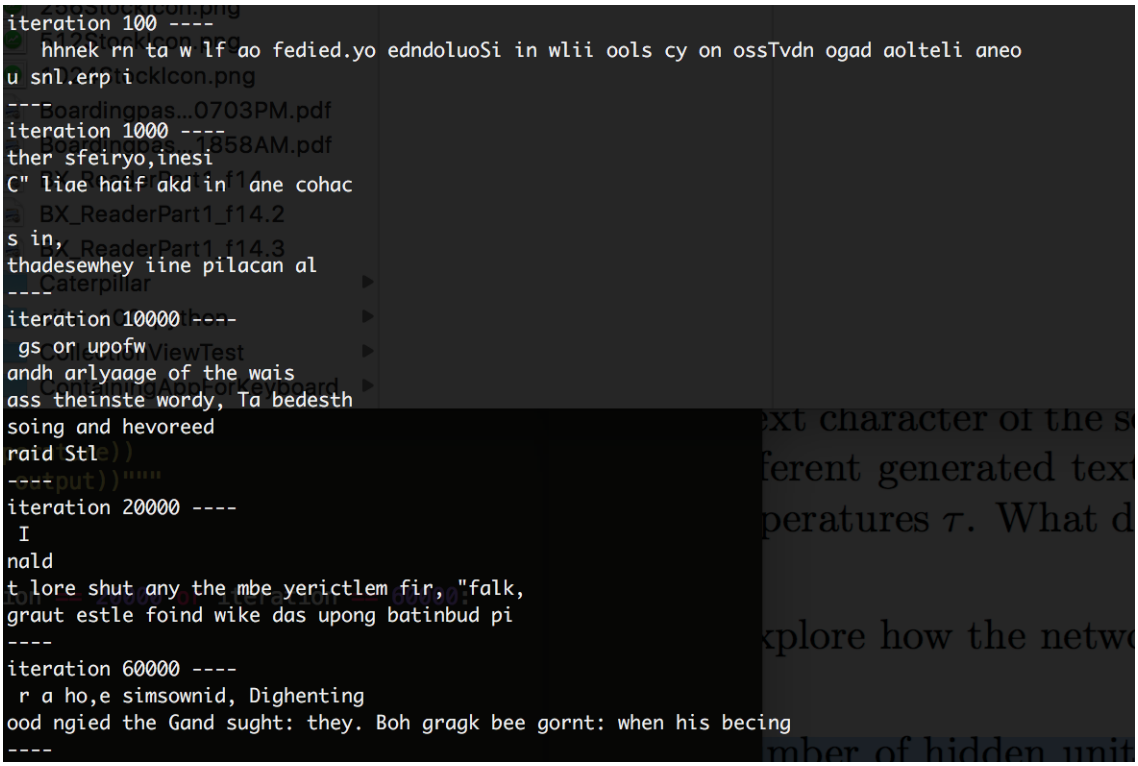


As we expected, when the temperature increases, the text generated is more random, and less English words are generated. When the temperature decreases, more English words are generated, but there are less variations and we end up with the same words repeating. For each of the 5 text samples at each temperature we provided a different input character. When the temperature is low that did not have a strong effect on the network since it started to just pick the highest probability character and then got stuck in a cycle. When the temperature is closer to 1.0 we got output more similar to our output when we didn't use temperature except with a bit more variation, resulting in words that start off looking like they will be English words but then have random characters appended halfway through.

(c) Experiment with Network Structure

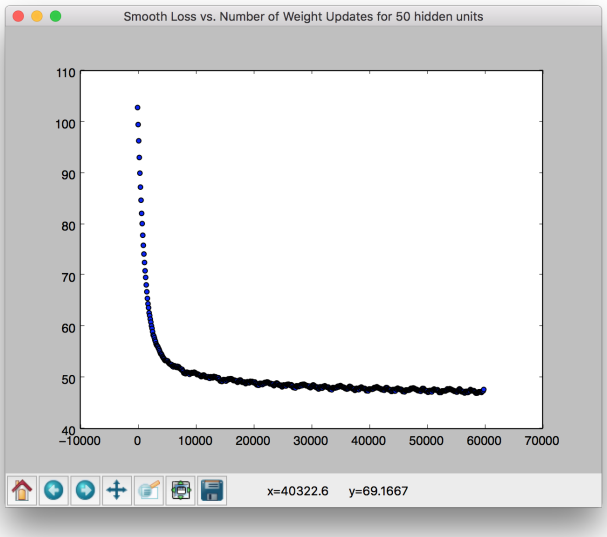
i. Double Number of Hidden Units





Doubling the number of hidden units allows the network to still learn but the words look a bit more random. Interestingly, it learned more of the structure of the input, with short play-like lines. It might have learned less well because we are using a gradient approximation by only going back one timestep which could have a worse effect now that there are more hidden units. It also took much longer to train with more hidden units.

Half Number of Hidden Units

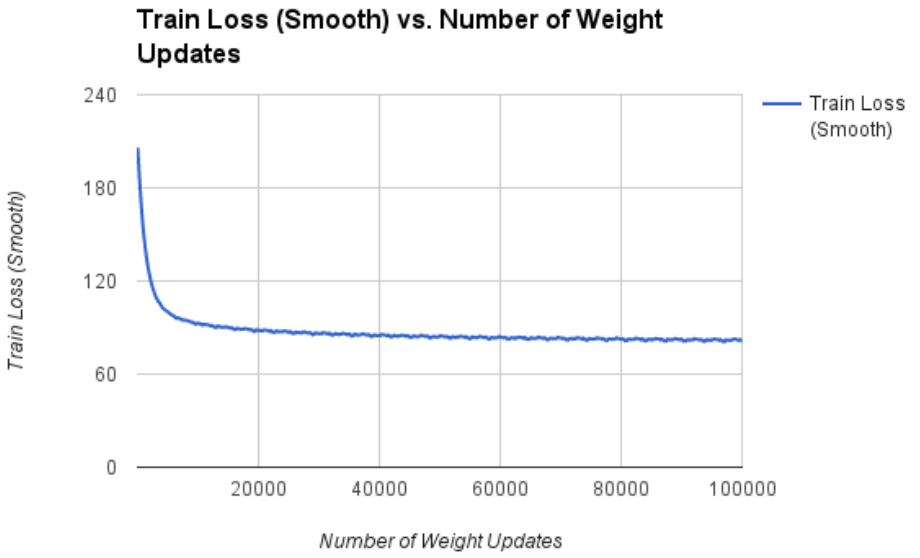



```
iteration 100 ----
  angpnac
dnSehin
ste, sou lnd,  ua ohay abi To s fawchegtis acterineanoulistholas ranertu vobets T,
----
iteration 1000 ----
 ed ut the leroutouort, and nbs and the lcriossing at evero bekoud out-inding pe ohibs
eceracm, and an
----
iteration 10000 ----
  urlents Helf aibe, fadoton."  Hese derif his Cood Cree apoons; a mist wesh seughed a
t meshe upon! A
----
iteration 20000 ----

vearg-ar
mabt
forne, shat conding thirnot too drow buld the dobe; the fou veass orben wit sain fir.
----
iteration 60000 ----
 orce, preap, I was where chithing
the fropk, he cat in the mire; and samen nown
sime to-shesh
anseav
----
```

Clearly with 50 hidden units the network was not able to learn as well as with 100 hidden units. It did start outputting "words" instead of just strings of characters but mostly not English words. It also ran a lot faster with only 50 hidden units.

ii. Double the Sequence Length

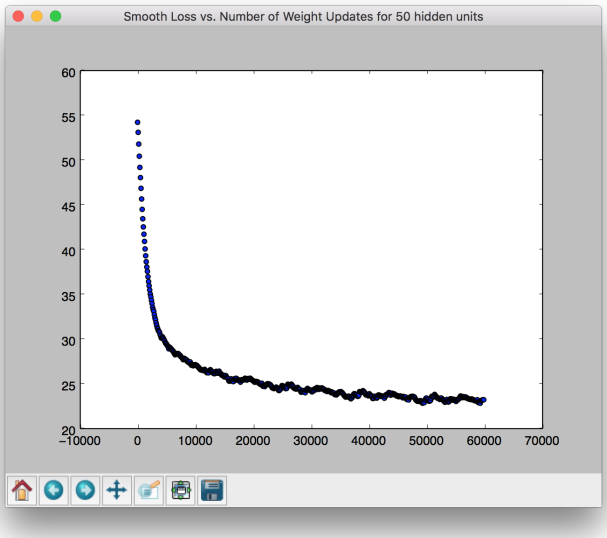


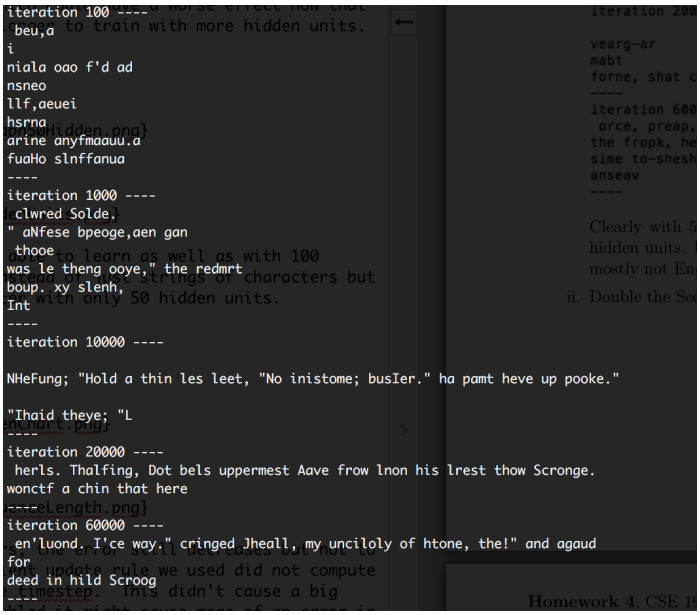

```
data has 161595 characters, 66 unique.
iter 100, loss=206.099460
----
 aet;,

asng wpv aoave.
qgf and
ed ah bnto sT phefgen-e
ssod athd aiuvgriatm h st co thd thitb
----
iter 1000, loss=154.529027
----
 nd. inq lora ther hpild the Pumed ann! af wrother! of bitet, ghang!"
de diwinc!s.
"Doost ald uhe P
----
iter 10000, loss=92.548218
----
 ndige hilly's a lating ob!" foidisalf has de the porgst to yo plich and as the stroudsed said, Mande
----
iter 20000, loss=88.401489
----
 Spircoin on mikit," selobe face inser, than gerning gonddy pit!"
Scrooge scendy, we deyring-hted th
----
iter 60000, loss=84.036508
----
  upien bettear man ba ant. town-tood his came
as nown turned by and with hus
went can hamseven,
a
----
```

When doubling the sequence length to 50 characters, the error still decreases but not to as low a value. This is likely because the gradient update rule we used did not compute exact derivatives and instead only looks back one timestep. This didn't cause a big difference before but when the time length is doubled it might cause more of an error in our gradient calculation. The text output still becomes English words near the end of our training.

Half the Sequence Length





The loss when using a 13 character sequence length was able to decrease to a much lower value. This is likely due to the same reason the loss increased for 50 character sequences - because we are only looking back one timestep to calculate the loss. However, the output text was not very good because the network could not learn many full words. Instead we see a structure of words rather than just random output although upon close inspection many of the "words" are not actually English.