Exercise 5 submission

Arnas Šniokaitis

December 7, 2025

1 Task 2.1

- Name the advantages of long short-term memory (LSTM) units and grated recurrent units (GRU) over vanilla RNNs.

  Vanilla RNNs have the problem of a vanishing gradient. LSTM solves this problem as well as allowing the model to learn long term dependencies more effectively. GRU also solves these issues as well as alowing the model to be trained more easily in comparison to LTSM by not using cell states. The problem arrises in the standard RNNs, because if we have a dependency that is far away from the source; since the gradient is computed for the entire network(or its parts) then the gradient for the dependency that is far away will be scalled down by all of the gradients that follow it in time.

- Name two key differences between a recurrent neural network that processes inputs of length l = 3 and a fully-connected network that has three layers.

  1. a fully connected model with three layers does not have any temporal capabilities
  2. a length of a temporal network does not mean that it takes three parameters, like it would if we were talking about an input vector for a standard non-temporal model; nor does it mean that our network has three layers. l here shows that it expects three inputs "in the time axis".

- Describe the core idea behind backpropagation through time (BPTT) and truncated BPTT (TBPTT). Discuss why we typically use TBPTT in practice.

  Standard BPTT or native BPTT as well as TBPTT trains on the entire model after it has been "rolled out". This can be imagines that instead of having some internal state, that gets moved as an input to the next layer, that we have two independant layers that both recieve data "simultaniously" and the first layer not only produces a prediction for the expected output but as well an input for the second layer. Naive BPTT trains on the entire model where TBPTT trains only up to a certain layer and; simmilarly like in convolutions, has a "stride". This is why in practice TBPTT is more commonly used in practice. since we don't have to propagate through the entire network a lot of time is saved on computatioins.

2 Task 2.2

```
def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in range(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) #
        hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for
        next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for
        next chars
    loss += -np.log(ps[t][targets[t],0])
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.
      zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(range(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1
    # backprop into y. see http://cs231n.github.io/neural-networks-case-
        study/#grad if confused here
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding
        gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

- What kind of RNN is used here?

  Since we are computing gradients for seq_size $= 14$ we are using truncated backpropagation through time. Since we only use the hidden state (no gates) we can conlude that this definitively TBPTT.

- Describe which vocabulary, i.e., which individual tokens, the RNN has at its disposal.

From line 11 we can deduce, that it uses characters as a vocabulary. This means that it hass acces to all characters in our text file (Upper and lowercase characters, punctuation, spaces, etc.).

- Identify and describe which loss function is used to train this RNN. Why is this a suitable strategy here?

  Line 45 Cross entropy is used. This can be explained by the fact, that we only have access to a discrete. As such we calculate the "probabilities" for each character to follow and pick one as can be seen in line 76.

- Describe which sampling strategy is implemented in the method sample to sample from the trained RNN

  From line 76 we can see, that it randomly choosen where are calculated probabilities are the probabilities that each character will be picked. This allows for more diverse outputs as it is possible for characters that don't have the highest probability to be picked.

- Train the model at least three times for around 100 000 epochs, observe the results and plot the behavior of the loss. Report your top-3 favorite generated text samples in the written part of this report.

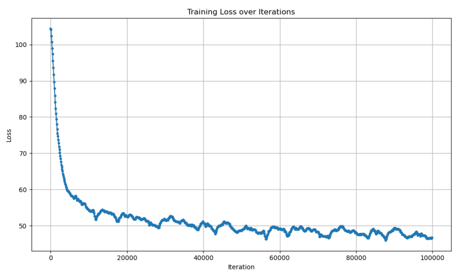

Figure 1: 1-st run

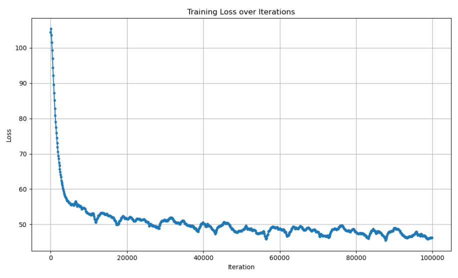

Figure 2: 2-nd run

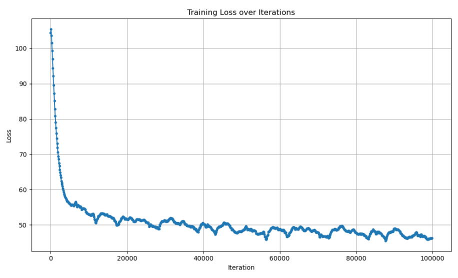

Figure 3: 3-rd run

1. tio, as do they hous volestoteds liy; agas ue. sou!

   San:

   Ferselday ance, Whink naven my ervedad, what his would her your: over, sreakn foecl wirunt.

   Sigh thime of gort'd perow wours Coujess, Yuch y

2. ASCALIN: I for lond ack atbize Fays wound be foo–pould till I'll vaeit.

   Srive hurme is this zende his kirit the show the coust gengiove; eoce sateze book? you his saans meo no rivan'd and havete, m

3. QUEEN MORKE: The shy haves. No sut ulforthcuy, plove haun the depan of ban dind hake you hast he anghat? And to fean the hince sid ole swemh out Heath other will thou chith out t

3 Task 2.3

- Have a look at the optimizer: Explain why it is not possible to just continue training after stopping the program and then reloading the weights.

  Because we are not saving the memory states (mem variables) as well as the hidden states, as such after reloading our weights our gradient will be substancially bigger than it needs to be and will "jump to far" from the minimum

- In addition to existing sampling method, implement greedy sampling and generate sam- ples using this approach with the same trained models for both sampling strategies. Do you see any difference in the generated results?

```
----
inn I wouk on, not croyist:
Cows: he ta cowstrour corp parssate. lor:
We ar saby Rorpy yem sid, it tham to chist ser:
O beages
-n thike.

MENENIUS:
What loop as himist
I kiust nothonsiks this foole
-nlllay any, ary thy ofud leomo amd the pate cone are.

FIUF:
Whi'l o foode, henll.

MENENIUS:
I wore with hew thon ofor net: sheaw whre, youlled:

FORINIUS:
I
Iferyow hear:
You grow
And you good corwefy pear aw will and wo leont hath foikise
Soms amanow now for you at me the king tathe kist bles thin
----
.....

----
inngIO:
I wore the with the with the with the with the with the with the
    with the with the with the with the with the with the with the
    with the with the with the with the with the with the with the
    with the with the with the with the with the with the with the
    with the with the with the with the with the with the with the
    with the with the with the with the with the with the with the
    with the with the with the with the with the with the with the
    with the with the with the with the with the with
----
```

There difference between the greedy algorithm(the text below .....) and the already implemented method is massive. It shows that we have a loop in our "optimal choice" as such with greedy choices (making locally best choices) leads our model to fail at producing ‘human‘ text.