Homework 2

CS 685, Fall 2025

Submit all answers as a PDF to Gradescope. Some sections ask for code; include the relevant implementation there. If you're using Colab for all coding this is by default, though feel free to not use Colab if you want. If you handwrite some answers, write very neatly if you want to receive full credit.

1 Dead neurons

(From INLP, ch. 3)

The ReLU activation function can lead to "dead neurons", which can never be activated on any input. Consider the following two-layer feedforward network with a scalar output y:

$$z_i = \text{ReLU}(\boldsymbol{\theta}_i^{(x \to z)} \cdot \boldsymbol{x} + b_i) \quad [3.61]$$

$$y = \boldsymbol{\theta}^{(z \to y)} \cdot \boldsymbol{z}. \quad [3.62]$$

Suppose that the input is a binary vector of observations, $\boldsymbol{x} \in \{0, 1\}^D$.

a) Under what condition is node z_i "dead"? Your answer should be expressed in terms of the parameters $\boldsymbol{\theta}_i^{(x \to z)}$ and b_i.

b) Suppose that the gradient of the loss on a given instance is $\frac{\partial \ell}{\partial y} = 1$. Derive the gradients $\frac{\partial \ell}{\partial b_i}$ and $\frac{\partial \ell}{\partial \theta_{j,i}^{(x \to z)}}$ for such an instance.

c) Using your answers to the previous two parts, explain why a dead neuron can never be brought back to life during gradient-based learning.

2 Vanishing gradients

Consider a recurrent neural network with a single hidden unit and a sigmoid activation, $h_m = \sigma(\theta h_{m-1} + x_m)$. Prove that if $|\theta| < 1$, then the gradient $\frac{\partial h_m}{\partial h_{m-k}}$ goes to zero as $k \to \infty$.[7]

(This proof generalizes to larger networks by considering the largest eigenvector of matrix $\boldsymbol{\Theta}$. From INLP, ch. 6)

3 Softmax gradients

Consider a language model to model the next word $w \in V$, where $p(w \mid ...\text{context}...) \equiv q_w$, from a linear softmax layer

$$q = \text{softmax}(y), \quad y = Wx$$

where $x \in \mathbb{R}^m$ is an embedding summarizing the context (from concatenation, last RNN state, (attention-weighted) averaging, whatever), and W is a weight matrix.

3.1

What are the dimensionalities of W, y, and q?

3.2

Assume the next word turns out to be 'dog'. Assume cross-entropy loss,

$$L = \sum_{w \in V} 1\{w = \text{'dog'}\} \log \frac{1}{q_w}$$

Derive the loss's gradient with respect to the pre-softmax scores y. That is, calculate the gradient $\nabla_y L$, which in lecture we wrote as $\frac{dL}{dy}$.

(Hint: you may have to treat different elements of y differently.)

(The loss formula above uses one-hot empirical distribution as an indicator function (like in JM ch. 6), and uses word "surprisal", $\log 1/q = -\log q$. Feel free to use whatever notation is useful for you, and is defined and clear to us, as readers.)

3.3

Cross-entropy loss encourages the model to allocate probability to the empirical data. Explain how this gradient does (or does not) match this high-level interpretation. For example, it may be useful to discuss the interpretation of taking a stochastic gradient descent step from this example.

3.4

You calculated the loss gradient with respect to all elements of y. But of course, to calculate the loss, you only need q_{dog} (or whichever index is for the actual next word), and all other q_k ($k \neq$ dog) are irrelevant. To backpropagate gradients further back on the computation graph to W and x, do you need to use all of y, or just y_{dog}? Draw a bit of a relevant computation graph, and any additional intuition, to help explain.

(If you'd like a more concrete exposition of the backprop algorithm, see the Eisenstein INLP textbook, section 3.3.1; link on course homepage.)

4 Word embeddings: similarities

4.1

Download the GloVe word embeddings, implement cosine similarity, and a function to identify the K nearest neighbors of a given word. (Do all implementation yourself using numpy; do not

use any other libraries.) Include your implementation in your submission.

Additional detail: for the word embeddings, please use the 50-dimensional variant of general GloVe word embeddings. We suggest this starter code to access them (the specific model name is defined by the Gensim package (here), which has the appropriate link to the original Stanford GloVe website (here)).

```
import gensim.downloader as api
import numpy as np

# Load pre-trained embeddings (this may take time)
glove = api.load("glove-wiki-gigaword-50")
```

4.2

Look up the 10 nearest neighbors for "dog" and "cat". Show them for each, along with their scores. What aspects or specific examples from these listings make sense? Why?

4.3

What aspects or specific examples from these listings, make less sense? Why?

4.4

Try some other nearest-neighbor queries for various words. Choose a word whose list shows something interesting; show it and explain what you found.

5 Word embeddings: the linear gender hypothesis

5.1

Implement a version of Bolukbasi et al. (2016)'s linear gender analysis method, which defines a word's signed gender score as its cosine against a "gender direction" in the embedding space.

Specifically, for this section, use their most basic variant to define the gender direction exclusively through two pronouns, as E('he')-E('she'), where positive should indicate male and negative indicates female (for a binary gender system). Report a sorted list, along with their gender scores, for the following professions or person terms. Include your code implementation here.

```
terms = ["doctor", "receptionist",  "engineer",
    "scientist",  "nurse", "teacher"]
```

5.2

Create a another list of terms, with at least 10 terms, whose embedding-defined implicit gender is interesting to investigate, in your opinion. Run the same analysis, report the results and your interpretation.

5.3

Antoniak and Mimno (2021) have criticized the use of a small set of seed terms to define semantic concepts, since the choice of seed terms may affect results. So conduct a sensitivity analysis: try at least two other pairs of gendered seed terms, and compare any changes to the scores or rank ordering. What do you find?

5.4

Does gender bias in word embeddings matter? Choose either yes or no, and argue for that position. For additional possible ideas, see Blodgett et al. (2020) (or the results of the extra credit, if you choose to do it).

5.5 Extra Credit

Implement one of Bolukbasi's gender debiasing methods, and try one of Gonen and Goldberg 2019's methods to re-identify gender even after the supposed debiasing. (We'd suggest the k-means one, which is relatively straightforward. Feel free to use any software libraries here; for example, sklearn's k-means.) What do you find?

6 Document classification

Here you'll conduct supervised document classification experiments for binary sentiment classification, on the IMDB dataset we used in class. Download the data; we suggest using the huggingface-accessed version as from class:

```
import datasets
from datasets import load_dataset
dd = load_dataset("imdb")
```

and use their train and test splits. (We won't bother with a development set for this homework, but for your projects it's highly recommended.)

6.1

Implement a bag-of-words logistic regression classifier and provide your implementation here.

You will have to construct the word vocabulary from the training set, including a mapping from word strings to their integer indexes in the model. You may find sklearn's "DictVectorizer" class to be helpful here (but its use is totally optional). For the machine learning level of the model, we suggest using sklearn's "LogisticRegression" classifier with L2 regularization. For tokenization, please use the simple tokenizer that Brendan used in class, just so we can more easily assess your results for grading:

```
re.split('\W+', text.lower())
```

6.2

Report results on the test set. Did you beat the manual keyword counting heuristic classifier from class?

6.3

Using the GloVe embeddings, implement a bag of embeddings classifier. Give your implementation in this section.

6.4

Report the BOE results on the test set. How does this model compare?

6.5

In theory, a word embedding based classifier should have an advantage over a BOW classifier, when the training dataset is very small. Explain why this might be the case.

6.6

To test this hypothesis, try training on a much smaller random sample of the training set, 100 documents total (select them at random), and compare the results from the two different classifiers using the same test set as all other experiments. What do you find?

6.7 Extra Credit

Try improving your model by incorporating some or all of the manual keywords that we collected in class. For example, you could use their embeddings (say, via Garten et al. (2018)'s "distributed dictionary representation", where you simply average all word embeddings from a dictionary to get a concept vector; they score a document via its cosine against the concept vector), or incorporate their count as a new manually defined feature, etc. Include your implementation, and report results and findings.

6.8 Extra Credit

Try improving your model with any other method you like. Use any other software you like. Include your implementation, and report results and findings.