# Natural Language Processing - Homework 3

Eshika Janbandhu (ej2485)

October 22, 2025

## Part 1. Recurrent Neural Network (35pt)

### Question 1 (4 point, written)

Write down the per-example cross-entropy loss $\ell(\mathbf{y}, \mathbf{p}_T)$ for the classification task. Here $\mathbf{y} \in \{0,1\}^k$ is a one-hot vector of the label and $\mathbf{p}_T$ is the class probability vector where $p_T[i] = p(y[i] = 1|S)$ for $i = 1, \ldots, k$. ($[i]$ denotes the $i$-th entry of the corresponding vector.)

**Answer to Question 1**

The per-example cross-entropy loss is:

$$\ell(\mathbf{y}, \mathbf{p}_T) = -\sum_{i=1}^{k} y[i] \log(p_T[i])$$

Since $\mathbf{y}$ is a one-hot vector, this simplifies to $\ell(\mathbf{y}, \mathbf{p}_T) = -\log(p_T[c])$, where $c$ is the index of the true class.

---

### Question 2

To perform backpropagation, we need to compute the derivative of the loss with respect to each parameter. Without loss of generality, let's consider the derivative with respect to a single parameter $w = W_{hh}[j', k']$ where $[j', k']$ denotes the $(j', k')$-th entry of the matrix. By chain rule, we have

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial o_T} \frac{\partial o_T}{\partial h_T} \frac{\partial h_T}{\partial w} \quad \text{where} \quad \frac{\partial h_T}{\partial w} = \sum_{i=1}^{T} \frac{\partial h_T}{\partial h_i} \frac{\partial^+ h_i}{\partial w} \tag{1}$$

Note that the first two derivatives $\frac{\partial \ell}{\partial o_T}$ and $\frac{\partial o_T}{\partial h_T}$ are easy to compute, so let's focus on the last term $\frac{\partial h_T}{\partial w}$. Here $\frac{\partial^+ h_i}{\partial w}$ denotes the "immediate" gradient where $h_{i-1}$ is taken as a constant.

**(a) (4 point, written)** Give an expression for $\frac{\partial^+ h_i}{\partial w}$.

**Answer to Question 2 (a)**

Let $a_i = W_{hh} h_{i-1} + W_{ih} x_i + b_h$. For $w = W_{hh}[j', k']$:

$$\frac{\partial^+ h_i}{\partial w} = \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w} = \text{diag}(\sigma'(a_i)) \cdot \mathbf{e}_{j'} h_{i-1}[k'] \tag{2}$$

where $\mathbf{e}_{j'}$ is the $j'$-th standard basis vector. Since $\sigma(x) = \tanh(x)$, $\sigma'(a_i[j']) = 1 - h_i[j']^2$. The expression simplifies to a vector where only the $j'$-th entry is non-zero:

$$\frac{\partial^+ h_i}{\partial w}[j] = \begin{cases} (1 - h_i[j']^2) h_{i-1}[k'] & \text{if } j = j' \\ 0 & \text{if } j \neq j' \end{cases}$$

**(b) (6 points, written)** Expand the gradient vector $\frac{\partial h_T}{\partial h_i}$ using the chain rule as a product of partial derivatives of one hidden state with respect to the previous hidden state. You do not need to do differentiations beyond that.

**Answer to Question 2 (b)**

The gradient $\frac{\partial h_T}{\partial h_i}$ expands as a product of Jacobian matrices (multiplication from right to left, i.e., in order of time):

$$\frac{\partial h_T}{\partial h_i} = \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdots \frac{\partial h_{i+1}}{\partial h_i} = \prod_{k=i+1}^{T} \frac{\partial h_k}{\partial h_{k-1}}$$

---

## Question 3 (6 points, written)

Now let's further expand one of the partial derivatives from the previous question. Write down the Jacobian matrix $\frac{\partial h_{i+1}}{\partial h_i}$ by rules of differentiations. You can directly use $\sigma'$ as the derivative of the activation function in the expression.

**Answer to Question 3**

The Jacobian matrix $\frac{\partial h_{i+1}}{\partial h_i}$ is:

$$\frac{\partial h_{i+1}}{\partial h_i} = \frac{\partial h_{i+1}}{\partial a_{i+1}} \frac{\partial a_{i+1}}{\partial h_i}$$

where $a_{i+1} = W_{hh} h_i + W_{ih} x_{i+1} + b_h$.

$$\frac{\partial h_{i+1}}{\partial h_i} = \text{diag}(\sigma'(a_{i+1})) W_{hh}$$

---

# Bounding Gradient Norm

1. **(5 points, written)** Given the mathematical form of the Jacobian matrix $\frac{\partial h_i}{\partial h_{i-1}}$ we derived earlier, we can now bound the norm of the Jacobian with the following matrix norm inequality

$$\|AB\|_2 \leq \|A\|_2 \cdot \|B\|_2 \tag{3}$$

for matrices $A, B$ with matched shapes. Write down a bound for $\|\frac{\partial h_i}{\partial h_{i-1}}\|_2$.

**Answer to Question 4 (1)**

Let $A = \text{diag}(\sigma'(a_i))$ and $B = W_{hh}$. Using inequality (3):

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 = \|\text{diag}(\sigma'(a_i)) W_{hh}\|_2 \leq \|\text{diag}(\sigma'(a_i))\|_2 \, \|W_{hh}\|_2$$

Since $\sigma(x) = \tanh(x)$, its derivative $\sigma'(x) = 1 - \tanh^2(x)$ is bounded by $0 < \sigma'(x) \leq 1$. The spectral norm of a diagonal matrix is its maximum absolute diagonal entry:

$$\|\text{diag}(\sigma'(a_i))\|_2 = \max_j |\sigma'(a_i[j])| \leq 1$$

Thus, the bound is:

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 \leq \|W_{hh}\|_2$$

2. **(10 points, written)** Now we have all the pieces we need. Derive a bound on the gradient norm $\|\frac{\partial h_t}{\partial h_i}\|_2$. Explain how the magnitude of the maximum singular value of $W_{hh}$ can lead to either vanishing or exploding gradient problems. You can use the fact that for the tanh activation function $\sigma(\cdot)$, the derivative $\sigma'(\cdot)$ is always less than or equal to 1.

**Answer to Question 4 (2)**

The norm of the gradient $\frac{\partial h_t}{\partial h_i}$ is bounded by:

$$\left\|\frac{\partial h_t}{\partial h_i}\right\|_2 = \left\|\prod_{k=i+1}^{t}\frac{\partial h_k}{\partial h_{k-1}}\right\|_2 \leq \prod_{k=i+1}^{t}\left\|\frac{\partial h_k}{\partial h_{k-1}}\right\|_2$$

Applying the bound from the previous question $\left\|\frac{\partial h_k}{\partial h_{k-1}}\right\|_2 \leq \|W_{hh}\|_2$ over the $t-i$ steps:

$$\left\|\frac{\partial h_t}{\partial h_i}\right\|_2 \leq \prod_{k=i+1}^{t}\|W_{hh}\|_2 = (\|W_{hh}\|_2)^{t-i}$$

Since $\|W_{hh}\|_2 = s_{\max}(W_{hh})$, the final bound is:

$$\left\|\frac{\partial h_t}{\partial h_i}\right\|_2 \leq (s_{\max}(W_{hh}))^{t-i}$$

**Vanishing and Exploding Gradients** The term $\left\|\frac{\partial h_t}{\partial h_i}\right\|_2$ forms a dominant multiplicative factor in the Backpropagation Through Time (BPTT) gradient. The magnitude of this term is determined by the maximum singular value of the recurrent weight matrix, $s_{\max}(W_{hh})$, raised to the power of the time difference $t-i$.

- **Vanishing Gradients:** If $s_{\max}(W_{hh}) < 1$, the term $(s_{\max}(W_{hh}))^{t-i}$ decays **exponentially** as the time difference $t-i$ increases. This means that gradients with respect to weights influenced by early time steps become nearly zero, preventing the model from learning **long-term dependencies**.
- **Exploding Gradients:** If $s_{\max}(W_{hh}) > 1$, the term $(s_{\max}(W_{hh}))^{t-i}$ grows **exponentially** as $t-i$ increases. This leads to extremely large gradients, causing large weight updates that destabilize the learning process (numerical overflow, NaN loss values).

# Part 2. Prompt Engineering for Addition (65pt)

## Q1. Zero-shot Addition (10pt)

**a. Run the two examples. In your opinion, what are some factors that cause language model performance to deteriorate from 1 digit to 7 digits?**

Language Model performance deteriorates from 1-digit to 7-digit addition due to:

1. **Tokenization Granularity:** Small numbers often map cleanly to a single token, but 7-digit numbers are typically split into **multiple subword tokens**. This fragmentation makes arithmetic reasoning across token boundaries much harder and increases error risk.
2. **Training Distribution:** LLMs are trained on natural text where **1–2 digit numbers appear far more frequently** than 7-digit numbers. Models tend to "memorize" small additions but must rely on less reliable combinatorial reasoning for large numbers.
3. **Probabilistic Error Accumulation:** Predicting a 7-digit sum requires multiple sequential token predictions. Even a low per-token error probability compounds exponentially over the long sequence, making the final result highly susceptible to mistakes.

**b. (3 points, written) Play around with the config parameters: • What does each parameter represent? The parameters to be probed are: Temperature, Max Tokens, Top P, Top K, and Repetition Penalty. • How does increasing each parameter change the generation?**

1. **Max Tokens (max_tokens)**
   - *Represents:* The maximum number of tokens the model is allowed to generate.

- *Effect of Increasing:* Produces **longer outputs**. For arithmetic, it prevents truncation of large sums, but too high can lead to unwanted extra text or numerical "hallucination."

2. **Temperature (temperature)**
   - *Represents:* Controls randomness in token selection (0 = deterministic, higher values = more diverse).
   - *Effect of Increasing:* Output becomes more creative. For arithmetic tasks, **higher temperature → more mistakes**. Lower temperature yields more consistent, accurate outputs.
3. **Top K (top_k)**
   - *Represents:* Limits sampling to the top $K$ most probable next tokens.
   - *Effect of Increasing:* Allows for **more randomness** and diversity in the selection pool; small $K$ leads to safer, more likely tokens.
4. **Top P (Nucleus Sampling, top_p)**
   *Represents:* Only considers tokens whose cumulative probability is $\leq P$. *Effect of Increasing:* Higher $P$ (closer to 1) considers more options, leading to **more diverse outputs**. Lower $P$ restricts selection.
5. **Repetition Penalty (repetition_penalty)**
   *Represents:* Penalizes tokens that have already appeared, discouraging repetition. *Effect of Increasing:* **Reduces repetitive outputs**. If too high, it may prevent legitimate repetition (e.g., repeated digits in a large sum).

**c. Do 7-digit addition with Qwen3-8B model. • How does the performance change? • What are some factors that cause this change?**

**Performance Change:** Performance generally remains strong but **accuracy drops slightly** (e.g., to $\sim 0.9$) compared to 1-digit sums, and a small but non-zero **Mean Absolute Error (MAE)** emerges due to minor digit errors in a few large test cases.

**Factors Causing the Change:**

1. **Tokenization Complexity:** Large numbers are split, increasing the risk of one token being predicted incorrectly.
2. **Probabilistic Generation:** The sequential prediction of many digits increases the chance of error accumulation over the output sequence.
3. **Arithmetic Approximation:** The LLM *approximates* arithmetic; errors are inevitable as the magnitude and length of the numbers grow.

**d. Previously we gave our language model the prior that the sum of two 7-digit numbers must have a maximum of 8 digits (by setting max token=8). What if we remove this prior by increasing the max token to 20? • Does the model still perform well? • What are some reasons why?**

The model generally calculates the correct numeric value, but its output often includes **extra, unwanted text, newlines, or "hallucinated" digits**.

**Reasons:**

1. **Removal of Implicit Prior:** The tight max_tokens = 8 provided an **implicit length constraint** for the 8-digit sum. Removing it (setting to 20) allows the model to continue generation past the necessary numeric result.
2. **Increased Noise Risk:** With more allowed tokens, the probabilistic generation is free to append irrelevant content if decoding parameters are not extremely tight, leading to output noise.
3. **Improved Edge Case Accuracy:** The risk of truncation for the largest possible sums is eliminated, potentially improving accuracy for those specific cases.

## Q2. In Context Learning (25pt)

**a. Using the baseline prompt ("Question: What is 3+7? Answer: 10 Question: What is a+b? Answer:"), check 7-digit addition for 10 pairs again. • Compared to zero-shot 7-digit additions with maximum 8 tokens, how does the performance change when we use the baseline in-context learning prompt? • What are some factors that cause this change?**

**Performance Change:** Using the 1-digit example **does not significantly improve** 7-digit addition performance compared to zero-shot. Accuracy remains in the same low-to-moderate range ($\sim 0.9$), and the MAE is still substantial.

**Factors Causing This Change:**

1. **Scale Mismatch (Irrelevant Context):** The example (3+7=10) is **not representative** of the complex carry operations required for 7-digit arithmetic. The model only learns the Question: Answer: format.
2. **Misleading Pattern Learning:** The model struggles to generalize the simple arithmetic pattern to the multi-token, high-magnitude numbers.
3. **Error Accumulation:** Long numeric outputs are intrinsically prone to error regardless of a simple, small-scale example.

**b. Now we will remove the prior on output length and re-evaluate the performance of our baseline one-shot learning prompt. We need to modify our post processing function to extract the answer from the output sequence. • Describe an approach to modify the post processing function. • Compared to 2a, How does the performance change when we relax the output length constraint? • What are some factors that cause this change?**

**1. Approach to Modify Post-Processing Function:** The function must be modified to robustly extract the numeric answer from potential noise:

- **Extract:** Use a Regular Expression (e.g.,
- **Convert:** Convert the captured string of digits into an integer for comparison.

**2. Performance Change (max_tokens $\rightarrow$ 20):** Performance **significantly deteriorates** compared to Q2.a (max_tokens = 8).

- Accuracy **drops** (e.g., $\leq 0.7$).
- MAE **skyrockets** (errors become enormous, e.g., $10^{31}$) due to **runaway generation** and numerical hallucinations.

**3. Factors Causing This Change:**

1. **Loss of Implicit Prior:** Removing the tight max_tokens constraint eliminates the implicit guidance that the output should be a short, numeric sequence.
2. **Probabilistic Instability:** With a non-relevant example and no length constraint, the model's probabilistic generation, which is approximate for arithmetic, amplifies initial small errors into catastrophic sequences of extra/repeated digits.
3. **Output Hallucination:** The model continues generating tokens that are statistically plausible sequels to a number sequence but are arithmetically nonsensical.

**c. Let's change our one-shot learning example to something more "in-distribution". Previously we were using 1-digit addition as an example. Let's change it to 7-digit addition (1234567+1234567=2469134). • Evaluate the performance with max tokens = 8. Report the res, acc, mae, prompt length. • Evaluate the performance with max tokens = 50. Report the res, acc, mae, prompt length. • How does the performance change from 1-digit example to 7-digit example? • Take a closer look at test range. How was res calculated? What is its range? Does it make sense to you? Why or why not?**

**1. Performance with max_tokens = 8:**

| **Metric** | res | acc | mae | prompt_length |
|---|---|---|---|---|
| **Value** | $-0.2055$ | 0.2 | $821,620$ | 79 |

**2. Performance with max_tokens = 50:**

| **Metric** | res | acc | mae | prompt_length |
|---|---|---|---|---|
| **Value** | $\approx -0.0$ | 0.0 | $\sim 3.25 \times 10^{31}$ | 79 |

**3. Change in Performance: 1-digit vs. 7-digit example:** Performance **dropped** from the 1-digit example to the single 7-digit example (e.g., acc :$\sim$ 1.0 $\rightarrow$ 0.2 at max_tokens = 8). This drop occurs because a single example (even if relevant in scale) is **insufficient for generalization**. The complexity of large-number arithmetic means that errors grow with magnitude, and a single instance fails to teach the model the necessary complex carry patterns.

**4. About res in test_range:** The formula is: $res = \text{acc} \cdot \left(\frac{1}{\text{prompt\_length}}\right) \cdot \left(1 - \frac{\text{mae}}{10^4}\right)$.

- *Range:* Theoretically unbounded on the negative side if MAE $> 10,000$.
- *Does it make sense?* **No, not for this task.** For 7-digit arithmetic, the MAE term ($\mathbf{1} - \text{MAE}/\mathbf{10^4}$) quickly dominates and becomes extremely negative, even with relatively small numeric errors. This makes the overall *res* score reflect primarily error magnitude rather than a balanced metric of accuracy and efficiency.

**d. Let's look at a specific example with large absolute error. • Run the cell at least 5 times. Does the error change with each time? Why? • Can you think of a prompt to reduce the error? • Why do you think it would work? • Does it work in practice? Why or why not?**

**Error Variability:** Yes, the error can change slightly over 5 runs (even with temperature = 0.7). This is due to the inherent **randomness** introduced by temperature $> 0$ and Top-K/Top-P sampling, which softens the probability distribution and allows the model to occasionally select a less-likely token, leading to output variance.

**Prompt to Reduce Error:** The most effective prompt is **Few-Shot In-Context Learning with Relevant Examples**:

```
Question: What is 1234567 + 1234567?
Answer: 2469134
Question: What is 7654321 + 2345678?
Answer: 10000000
Question: What is 9999999 + 1?
Answer: 10000000
Question: What is a + b?
Answer:
```

**Why it would work:**

1. **Pattern Recognition:** It teaches the model the **scale** and **format** necessary for large numbers.
2. **Carry Operations:** It explicitly shows examples (like 7,654,321 + 2,345,678) that require complex carry operations, allowing the model to better approximate the underlying arithmetic logic.
3. **Removes Misleading Priors:** It overrides the implicit priors from smaller examples and guides the model toward the correct 8-digit output length.

**Does it work in practice? Why or why not?** Partially, yes. Adding multiple relevant examples significantly improves acc and reduces mae to the best achievable level for the model size. However, it is limited because:

- **Fundamental Limitation:** The LLM's arithmetic is still approximate, not exact. It cannot guarantee 100% accuracy on all novel 7-digit inputs.
- **Context Length:** Too many examples can make the prompt too long, potentially causing the model to truncate important parts of the context or the final output.