CS 5480: Deep Learning
Assignment 1: Foundations

Total: 100 (+25 bonus)

Due: February 17, 2026 at 11:59 PM

Instructions

- This homework has two parts: **Part I (Theory)** and **Part II (Coding)**
- Submit a PDF with theory answers and a separate Python notebook (.ipynb) for coding questions
- Show all work for theory questions
- Include comments and markdown explanations in your code
- For coding questions, use NumPy for numerical computations (no PyTorch/TensorFlow)

Submission & Formatting Requirements

- **Submission platform:** All submissions **must be uploaded to Canvas** by the stated deadline. Email submissions will **not** be accepted under normal circumstances.
- **Single ZIP file:** You must submit **one ZIP file** containing:
  - A **typed PDF** with all **theory answers**
  - A completed **Jupyter notebook (.ipynb)** for the coding portion

  Submissions that are not ZIP files or that are missing required components will be considered **incomplete**.
- **Typed work only: Handwritten solutions will not be accepted.** All theory answers must be typed.
- **Recommended tools:** You are **strongly encouraged to use LaTeX** for the theory portion. (Overleaf is a good option if you are new to LaTeX.)
- **Code requirements:**
  - The notebook must run **top-to-bottom without errors**
  - All plots must be generated by the notebook (no pasted images)
  - Include clear **markdown explanations** interpreting your results
- **File naming convention:** Name your ZIP file as:

  `LastName_FirstName_Assignment1.zip`

Late Policy

Late submissions will follow the course late policy as stated in the syllabus. Submissions made after the deadline without prior approval will not be graded.

Academic Integrity & Collaboration

You may discuss **high-level concepts** with classmates, but all submitted work must be **your own**. You may **not** share code, derivations, or written solutions. Any external resources used must be **explicitly cited**.

Use of AI Tools

You may use AI tools (e.g., ChatGPT) for **clarification, debugging assistance, or conceptual understanding**. You may **not** use AI tools to generate final solutions, derivations, or complete code.

You are responsible for understanding and explaining everything you submit. If AI tools are used, include a brief acknowledgment describing how they were used.

Read the full AI usage policy on Canvas.

Reproducibility

Your results must be **reproducible**. If we rerun your notebook, it should produce the same outputs (up to randomness explicitly controlled via random seeds).

Clarity Expectations

Clarity matters. Solutions will be graded on **correctness, reasoning, and explanation**, not just final numerical results.

PART I: THEORY

1. Understanding Learning as Function Approximation (15 points)

Imagine you're a data scientist at Zillow trying to predict house prices. You have historical data: square footage and sale prices. Your job is to build a model that predicts price from square footage. But here's the key question: **what kind of function should you use?**

This is where **hypothesis classes** come in. A hypothesis class is the set of all possible functions your model can represent. For example:

- **Linear functions:** $f(x) = w_1 x + w_0$ (straight lines only)
- **Quadratic functions:** $f(x) = w_2 x^2 + w_1 x + w_0$ (can curve)
- **Neural networks:** Much more flexible, can represent very complex patterns

The choice matters enormously:

- **Too simple** (e.g., straight line when data curves): Your model will never fit the data well, even with infinite training data. This is **underfitting** or **high bias**.
- **Too complex** (e.g., degree-20 polynomial for a simple trend): Your model will memorize training data, including noise, and fail on new houses. This is **overfitting** or **high variance**.
- **Just right**: Your model captures the true pattern without memorizing noise.

This problem helps you develop intuition for:

1. How to count model capacity (number of parameters)
2. How complexity affects overfitting and underfitting
3. Why training error alone is misleading
4. How to choose model complexity based on data size

Real-world impact: Airbnb's pricing model, Google's ad click prediction, hospital readmission forecasts—all require choosing the right hypothesis class. Get it wrong, and your model either oversimplifies reality or chases noise.

Part (a): Counting Model Capacity (5 points)

Consider predicting house prices from square footage (x). For each hypothesis class below, write the general form and count the number of parameters.

Why we're counting parameters: More parameters = more capacity = more flexibility. A model with 100 parameters can represent more complex patterns than one with 3 parameters. But more isn't always better—it depends on how much data you have.

(i) Linear functions: $f(x) =$

Hint: Form is $w_1 x + w_0$. How many numbers do you need to specify this function?

(ii) Quadratic polynomials: $f(x) =$

Hint: Includes x^2, x, and constant terms.

(iii) Cubic polynomials: $f(x) =$

(iv) Two-layer neural network with 5 hidden units and ReLU activation:

Number of parameters =

Hint: Count weights and biases. First layer: input (1D) to hidden (5D). Second layer: hidden (5D) to output (1D). Each connection has a weight, each neuron has a bias.

Part (b): Model Capacity and Real-World Constraints (5 points)

Scenario: You only have 10 training examples (10 houses with known prices). You need to choose which hypothesis class to use.

(i) Which hypothesis class above is most likely to overfit? Why is this dangerous in production?

Think about: What happens when you have more parameters than data points? Will the model generalize to new houses?

(ii) Which is most likely to underfit? When might you deliberately choose this simpler model?

Think about: Are there scenarios where interpretability matters more than perfect accuracy?

(iii) Suppose the true relationship is $y = 50 + 100x + 0.5x^2$ (quadratic). Explain what will happen if you fit:

- A linear model:
- A degree-10 polynomial:

Part (c): Why Training Error Misleads You (5 points)

Critical lesson: In real ML systems, you never know the test performance in advance. You only see training error during development. This problem shows why optimizing training error alone fails.

You fit models from different hypothesis classes on a training set. Here are the results:

Model	**Training MSE**	**Test MSE**
Linear	25.0	26.0
Degree-3 polynomial	12.0	15.0
Degree-10 polynomial	0.5	45.0
Neural net (100 hidden units)	0.01	52.0

(i) Which model will perform best on new houses (real deployment)? Justify your answer.

(ii) The degree-10 polynomial achieves near-zero training error but terrible test error. What phenomenon is this? Why is it dangerous?

Think about: Would you deploy this model? What would happen to users?

(iii) If you could only see training error (realistic scenario during development), which model would you choose? Would this be the right choice? How do practitioners solve this problem?

Hint: This is why we need validation sets.

2. Loss Functions and Their Properties (20 points)

You're building a medical AI to predict patient recovery time after surgery. Your model predicts 5 days, but the actual recovery takes 6 days. How bad is this error?

Now consider: your model predicts 5 days, but the patient takes 30 days (major complication). How bad is *this* error?

The **loss function** determines how your model measures mistakes. This choice has profound consequences:

Real-world example:

- **Uber's ETA predictions:** If they use Mean Absolute Error (MAE), they treat "5 minutes early" and "5 minutes late" equally. But users *hate* being late more than early! The loss function should reflect this asymmetry.
- **Medical diagnosis:** Missing a cancer diagnosis (false negative) might be much worse than a false alarm (false positive). Your loss function must encode this priority.
- **Autonomous vehicles:** Small errors in distance estimation are fine. Large errors could be catastrophic. You want a loss that heavily penalizes big mistakes.

Key insight: The loss function is how you communicate your priorities to the learning algorithm. Choose wrong, and your model optimizes for the wrong thing—even if it converges perfectly.

This problem explores three fundamental properties:

1. How different losses measure the same errors differently
2. Why some losses are more sensitive to outliers (large errors)
3. Why we need smooth, differentiable losses for gradient descent

Part (a): Computing Different Losses (8 points)

Scenario: You're predicting hospital length of stay (days). Consider a regression problem where the true values are $\mathbf{t} = [2.0, 5.0, 3.0]$ and your model predicts $\mathbf{y} = [2.5, 4.0, 3.2]$.

Calculate the following:

(i) Mean Squared Error: $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - t_i)^2$

(ii) Mean Absolute Error: $\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - t_i|$

(iii) Root Mean Squared Error: $\text{RMSE} = \sqrt{\text{MSE}}$

Note: RMSE has the same units as your prediction (days), making it more interpretable than MSE.

Part (b): Outlier Sensitivity in Production Systems (7 points)

Critical scenario: Now suppose your model makes one catastrophically wrong prediction: $\mathbf{y}' = [2.5, 10.0, 3.2]$ (predicted 10 days instead of 5—a serious error that could lead to understaffing).

(i) Recalculate MSE and MAE for this new prediction.

(ii) By what factor did each loss increase?

- MSE increased by: $\times$
- MAE increased by: $\times$

(iii) Which loss is more sensitive to outliers? Explain why this makes sense given the mathematical form of each loss.

When this matters:

- **Choose MSE if:** You want to heavily penalize large errors (e.g., safety-critical systems)
- **Choose MAE if:** Your data has outliers that shouldn't dominate training (e.g., real estate with a few mansions)

Part (c): Why Gradient Descent Needs Smooth Losses (5 points)

(i) The 0/1 loss for classification is defined as:

$$\ell_{0/1}(y, t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases}$$

This is the "perfect" loss—it directly measures classification accuracy. So why can't we use gradient descent to optimize it directly?

Hint: Draw this function. What does its derivative look like?

(ii) Instead, we use smooth surrogate losses like cross-entropy or hinge loss. What property must a loss function have to be optimizable with gradient descent? Why does this property enable learning?

Think about: How does gradient descent work? What information does it need from the loss function?

3. Why Stacking Linear Layers Does Nothing (15 points)

Imagine you're debugging a neural network that won't learn complex patterns. Your architecture looks impressive on paper: 4 layers with 128, 64, 32, and 1 neurons. But you forgot to add activation functions. This network is mathematically equivalent to a single linear layer—you've built a "deep" network that's actually shallow.

Real-world impact: Wasted computation, false sense of model capacity, and debugging nightmares. Understanding why composition of linear functions stays linear is fundamental.

Part (a): Two Linear Layers (8 points)

Consider two linear transformations composed:

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$
$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

(i) Substitute the first equation into the second to eliminate $\mathbf{h}$:

$$\mathbf{y} =$$

(ii) Let $\mathbf{W} = \mathbf{W}_2\mathbf{W}_1$ and $\mathbf{b} = \mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2$.
Show that the composition is equivalent to a single linear layer:

$$\mathbf{y} =$$

(iii) What does this imply about the expressive power of stacking linear layers?

Part (b): With Nonlinearity (7 points)

Now add a nonlinear activation σ (like ReLU or sigmoid):

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

(i) Can you simplify this to a single linear layer? Why or why not?

(ii) This is why we say that "activation functions enable neural networks to learn nonlinear functions." Explain this statement in your own words.

4. Sigmoid and Tanh Equivalence (20 points)

The sigmoid and tanh activation functions are mathematically related through a simple transformation. Understanding this relationship reveals that they have equivalent representational power—the choice between them affects optimization behavior rather than what functions the network can represent.

Definitions

$$\text{Sigmoid: } \sigma(a) = \frac{1}{1+e^{-a}}$$

$$\text{Tanh: } \tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$$

Part (a): Sigmoid in Exponential Form (4 points)

Starting with $\sigma(a) = \frac{1}{1+e^{-a}}$:

(i) Multiply numerator and denominator by e^a and simplify:

$$\sigma(a) =$$

Hint: Remember that $1 \cdot e^a = e^a$ *and* $e^{-a} \cdot e^a = e^{a-a} = e^0 = 1$

(ii) Your answer should be in the form: $\sigma(a) = \frac{e^a}{(\text{something})}$. What is the denominator?

Part (b): Scaling the Input (5 points)

Now consider $\sigma(2a)$ instead of $\sigma(a)$.

(i) Write $\sigma(2a)$ using the exponential form from part (a):

$$\sigma(2a) =$$

(ii) Multiply numerator and denominator by e^{-a} and simplify to get an expression with both e^a and e^{-a}. Show your work.

(iii) Write your final simplified result for $\sigma(2a)$.

Part (c): Deriving the Tanh Relationship (6 points)

(i) Compute $2\sigma(2a)$ using your result from part (b):

$$2\sigma(2a) =$$

(ii) Now compute $2\sigma(2a) - 1$.

Hint: To subtract 1, write it as a fraction with the same denominator, then combine.

Show your work step by step:

(iii) Compare this result with the definition of $\tanh(a)$. Write the relationship you discovered:

$$\tanh(a) =$$

Part (d): Neural Network Implications (5 points)

Consider a neural network with sigmoid activations:

$$y(x) = w_0 + \sum_{j=1}^{M} w_j \, \sigma\left(\frac{x - \mu_j}{s}\right)$$

(i) Using the identity relating σ and tanh that you derived in part (c), explain how to rewrite this network using tanh activations instead. Show how the weights and scale parameter change.

(ii) What does this tell you about the representational power of sigmoid vs tanh networks?

(iii) If they have the same representational power, why might we prefer tanh over sigmoid for hidden layers? *Hint: Consider that tanh is zero-centered while sigmoid outputs are always positive.*

PART II: CODING

5. Bias-Variance Through Experimentation (30 points)

One of the most fundamental tradeoffs in machine learning is between bias and variance. In this problem, you will empirically observe this tradeoff by fitting polynomial models of different degrees to noisy sinusoidal data.

You will generate data from $y = \sin(2\pi x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 0.3^2)$, then fit polynomials of degrees 1, 3, 5, 9, and 15 to a small training set (20 points).

Key questions you'll answer:

- What happens to training error as model complexity increases?
- What happens to test error? Does it keep decreasing?
- How does model complexity affect sensitivity to the specific training data?
- Which polynomial degree gives the best generalization?

Tasks (see notebook for details)

Part (a): Data Generation and Visualization (5 points)

Generate training data (20 points) and test data (100 points) from the noisy sine function. Create a plot showing the true function and your training points.

Part (b): Fit Polynomials of Different Degrees (7 points)

Fit polynomials of degrees 1, 3, 5, 9, and 15 using `np.polyfit`. Create visualizations showing how each polynomial fits the training data. You should observe:

- Low degrees: smooth curves that miss the pattern (underfitting)
- Medium degrees: capture the sine wave reasonably well
- High degrees: wild oscillations between data points (overfitting)

Part (c): Compute Training and Test Errors (6 points)

For each polynomial degree, compute Mean Squared Error on both training and test sets. Create a plot showing how both errors change with model complexity. You should see:

- Training error decreases monotonically (always gets better)
- Test error forms a U-shape (decreases then increases)

Part (d): Multiple Runs to Measure Variance (7 points)

Run the entire experiment 50 times with different random training sets. For each degree, track the test error across all runs. Create a plot with error bars showing mean test error $\pm$ 1 standard deviation.

This shows you how sensitive each model is to the specific training data—high-degree polynomials will have large error bars (high variance).

Part (e): Analysis and Interpretation (5 points)

In the Jupyter Notebook, answer these questions:

- Which model has lowest training error vs. lowest test error
- The bias-variance tradeoff you observed
- Why high-degree polynomials have high variance
- How you would choose a model in practice (validation sets)

What to Submit

For coding problems:

1. Download `assignment1_coding.ipynb`
2. Complete all TODO sections in code cells
3. Add your written analysis in markdown cells
4. Run all cells to ensure outputs are displayed
5. Save and submit the completed `.ipynb` file

The notebook contains detailed instructions, starter code, hints, and expected outputs for each part.

Tips for Success

- **Theory:** Show your work, not just final answers. Explain your reasoning.
- **Coding:** Use clear variable names, add comments, include markdown explanations.
- **Visualizations:** Label axes, add legends, use informative titles.
- **Analysis:** Don't just run code—interpret the results and explain what you observe.
- **Start early:** The coding problems require experimentation and debugging.
- **Test on simple cases first:** Make sure your implementations work before running full experiments.

BONUS QUESTIONS

*These questions are entirely optional. They are designed to reward deeper engagement with the material and may be used to earn additional points beyond the base score. You are **not required** to attempt them to achieve full credit on this assignment.*

6. Robust Learning with Label Noise (15 points)

In standard binary classification, we assume the labels in our training set are the "ground truth." But in the real world, data is often messy.

Real-world impact:

- **Crowdsourcing:** If you use Amazon Mechanical Turk to label images, tired annotators might click the wrong button.
- **Medical Imaging:** Even expert radiologists sometimes disagree on a diagnosis, effectively introducing "noise" into the labels.

If your loss function trusts these wrong labels too much, the model effectively "learns" the mistakes. In this problem, you will derive a loss function that accounts for the probability that a label might be wrong.

Definitions:

- $t \in \{0, 1\}$: The noisy label observed in the dataset.
- $y(x)$: The model's output representing the probability of the **true** class being 1.
- ϵ: The probability that the label has been flipped (incorrectly set).
  - If the true class is 1, the label t becomes 0 with probability ϵ.
  - If the true class is 0, the label t becomes 1 with probability ϵ.

Part (a): The Effective Probability (4 points)

We cannot trust the observed label t directly. Instead of modeling $P(\text{true class} = 1|x)$, we must model $P(t = 1|x)$ (the probability of observing a label of 1, given the noise).

Using the law of total probability, write an expression for $\tilde{p} = P(t = 1|x)$ in terms of the model output $y(x)$ and the noise rate ϵ.

$$\tilde{p} = P(t = 1|x) = \dots$$

Hint: Consider the two ways a label of 1 can occur: either the true class was 1 and it wasn't flipped, OR the true class was 0 and it WAS flipped.

Part (b): Deriving the Loss (5 points)

Now that you have the probability of the *observed* label, we can treat this as a standard maximum likelihood problem.

(i) Write down the likelihood for a single data point (x, t). You may assume the standard Bernoulli form:

$$P(t|x) = \tilde{p}^t(1-\tilde{p})^{1-t}$$

where $\tilde{p}$ is the effective probability you derived in Part (a).

(ii) Take the negative logarithm to derive the **Negative Log Likelihood (NLL)** loss function for this noisy scenario.

Part (c): Verification (3 points)

Set $\epsilon = 0$ in your derived error function from Part (b). Show that it simplifies back to the standard Binary Cross-Entropy (BCE) loss (Equation 6.33 in standard texts):

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$

Part (d): Why This is Robust (3 points)

Consider a case where the model is very confident ($y \approx 1$) but the label is actually 0 (perhaps due to an error).

(i) What happens to the standard Cross-Entropy loss in this case? (Does it explode?)

(ii) What happens to your new loss function with $\epsilon > 0$? Why does this prevent the single noisy point from destroying the training?

7. Classification with Different Target Conventions (10 points)

While modern deep learning frameworks (PyTorch/TensorFlow) typically use $t \in \{0, 1\}$ for binary classification, other machine learning traditions (like Support Vector Machines or older Neural Net literature) often use targets $t \in \{-1, 1\}$.

Understanding how to translate between these conventions is crucial when implementing custom loss functions or reading research papers.

Part (a): Mapping Outputs to Probabilities (3 points)

Suppose your network uses a **tanh** activation function at the output, so the prediction $y(x)$ lies in the range $[-1, 1]$. However, we still need a probability to perform Maximum Likelihood Estimation.

We define the probability of Class C_1 (target $t = 1$) as:

$$P(C_1|x) = \frac{1 + y(x)}{2}$$

Check the boundaries:

- If $y(x) = 1$, what is the probability of class C_1?
- If $y(x) = -1$, what is the probability of class C_1?

Part (b): Constructing the Likelihood (4 points)

The target values are now $t \in \{-1, 1\}$.

- If $t = 1$, we want to maximize $P(C_1|x)$.
- If $t = -1$, we want to maximize $P(C_{-1}|x) = 1 - P(C_1|x)$.

Using the expression from Part (a), derive the Negative Log Likelihood loss function for a single data point (x, t).

Hint: You can try to write a unified expression for the probability using the trick $\frac{1+t}{2}$ (which is 1 when $t = 1$ and 0 when $t = -1$) or simply derive it for the two cases and combine them.

Part (c): Comparison (3 points)

Does your derived loss function penalize confident wrong predictions in the same qualitative way as standard cross-entropy? Briefly explain by simplifying your result or comparing the behavior when the model is wrong.