Simon Edmunds
2/15/2026

CS 5480: Deep Learning
Assignment 1: Foundations

Due: February 17, 2026 at 11:59 PM

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:** $f(x) = w_1x + w_0$ (straight lines only)
- **Quadratic functions:** $f(x) = w_2x^2 + w_1x + w_0$ (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.

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: $f(x) = w_1x + w_0$ (2 parameters)

Hint: Form is $w_1x + w_0$*. How many numbers do you need to specify this function?*

(ii) Quadratic polynomials: $f(x) = w_2x^2 + w_1x + w_0$ (3 parameters)

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

(iii) Cubic polynomials: $f(x) = w_3x^3 + w_2x^2 + w_1x + w_0$ (4 parameters)

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

Number of parameters = (5 weights + 5 biases) + (5 weights + 1 bias) = 16 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?*

With this dataset, the two-layer neural network is most likely to overfit. Because it has 16 parameters, which is larger than the provided dataset, the model will effectively "memorize" the data. This could be dangerous in production, because the model will likely make confident, incorrect predictions, and small changes in square footage could lead to wildly different predictions.

(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?*

The linear model is most likely to underfit, as it contains the fewest parameters. You might pick this model if you have a very small dataset, and overfitting is a major concern. Also, in situations with few variables, the linear model is likely a relatively good predictor (obviously, if the data is linear).

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

A linear model: The model doesn't have enough parameters to represent the true relationship, so it will underfit, finding the best linear approximation. This will result in error across the spectrum.

A degree-10 polynomial: This model has way more parameters than necessary for a quadratic function. It will not underfit, but with only 10 data points to approximate over, it will probably overfit heavily, introducing high variance and decreasing test accuracy.

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.

The degree-3 polynomial will perform the best on new houses. From the table above, it has the lowest test error among all of the models, and although it isn't the lowest training error, Training MSE ~ Test MSE, showing a good bias-variance tradeoff.

(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?*

The degree-10 polynomial is showing strong overfitting to the training data. The model has effectively memorized the training data, meaning that its fit to the real function would be unstable (high variance). In deployment, this would mean wildly unstable predictions, with huge fluctuations from slight changes to the input data.

(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.*

If I were only basing my decision on the training data, I would likely choose the neural network with 100 hidden units, as it has almost zero training error on the data. This would not be the right choice, as this model is clearly overfit. Practitioners solve this problem by using a validation set that is separate from the training data, allowing them to recognize (and avoid) overfitting, even without access to the real test data.

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 more severe error (5-30 days) is significantly worse than the smaller error (5-6 days)
The **loss function** determines how your model measures mistakes. This choice has profound consequences.

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 t = [2.0,5.0,3.0] and your model predicts y = [2.5,4.0,3.2]. Calculate the following:

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

$\frac{1}{3}(0.5^2 + 1.0^2 + 0.2^2) = \frac{1}{3}(1.29) = 0.43$

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

$\frac{1}{3}(0.5 + 1.0 + 0.2) = \frac{1}{3}(1.7) = 0.566\overline{6}$

(iii) Root Mean Squared Error: RMSE = $\sqrt{MSE}$
Note: RMSE has the same units as your prediction (days), making it more interpretable than MSE.

$\sqrt{0.43} = 0.6557$

Part (b): Outlier Sensitivity in Production Systems (7 points)
Critical scenario: Now suppose your model makes one catastrophically wrong prediction: $y^{/}$ = [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.

$$MSE = \frac{1}{3}(0.5^2 + 5.0^2 + 0.2^2) = \frac{1}{3}(25.29) = 8.43$$

$$MAE = \frac{1}{3}(0.5 + 5.0 + 0.2) = \frac{1}{3}(5.7) = 1.9$$

(ii) By what factor did each loss increase?

- MSE increased by:

× 19.60

- MAE increased by:

× 3.353

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

Of the two, MSE is much more sensitive to outliers. This makes sense, because MSE grows with the square of the error, causing quadratic growth, unlike MAE, which grows linearly with increasing error.

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?*

We can't use gradient descent to optimize this function directly, because it is a stepwise function, flat everywhere except for the transition, where the gradient is undefined. Thus, the gradient gives no useful information for optimization.

(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?*

To optimize a function with gradient descent, the loss must be differentiable everywhere and provide useful gradient information. Thus, the function must be continuous and differentiable. This enables learning because the gradient will always exist for a given training point, and can be "descended" to learn.

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 night mares. Understanding why composition of linear functions stays linear is fundamental.

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

Consider two linear transformations composed:

$$h = W_1x + b_1$$
$$y = W_2h + b_2$$

(i) Substitute the first equation into the second to eliminate h:

$$y = W_2(W_1x + b_1) + b_2$$

$$y = W_2W_1x + W_2b_1 + b_2$$

(ii) Let $W = W_2W_1$ and $b = W_2b_1 + b_2$. Show that the composition is equivalent to a single linear layer:

$$y = (W_2W_1)x + (W_2b_1 + b_2)$$

$$y = Wx + b$$

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

This demonstrates that any composition of linear functions or linear layers is still a linear function, and that stacking linear layers does not increase expressive power.

Part (b): With Nonlinearity (7 points)

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

$$h = \sigma(W_1 x + b_1)$$
$$y = W_2 h + b_2$$

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

No, this cannot be simplified to a single linear layer, because the activation function is nonlinear. This means that the composition cannot be rewritten in the form $y = Wx + b$.

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

Activation functions introduce nonlinearity to the neural network, which allows stacked layers to model more complex patterns, as the effective composition of functions is now non-linear.

4. Sigmoid and Tanh Equivalence (20 points)

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

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 σ(a):

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

σ(a) = $\frac{e^{a}}{e^{a}+1}$

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: σ(a) = $\frac{e^{a}}{(something)}$. What is the denominator?

As shown above, the denominator is $(e^{a} + 1)$

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

Now consider σ(2a) instead of σ(a).

(i) Write σ(2a) using the exponential form from part (a):

σ(2a) = $\frac{e^{2a}}{e^{2a}+1}$

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

σ(2a) = $\frac{e^{2a}(e^{-a})}{e^{2a}(e^{-a})+1(e^{-a})} = \frac{e^{2a-a}}{e^{2a-a}+e^{-a}} = \frac{e^{a}}{e^{a}+e^{-a}}$

(iii) Write your final simplified result for σ(2a).

σ(2a) = $\frac{e^{a}}{e^{a}+e^{-a}}$

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

(i) Compute 2σ(2a) using your result from part (b):

2σ(2a) = $\frac{2e^{a}}{e^{a}+e^{-a}}$

(ii) Now compute 2σ(2a) − 1. *Hint: To subtract 1, write it as a fraction with the same denominator, then combine.* Show your work step by step:

2σ(2a) - 1 = $\frac{2e^{a}}{e^{a}+e^{-a}} - \frac{e^{a}+e^{-a}}{e^{a}+e^{-a}} = \frac{2e^{a}-e^{a}-e^{-a}}{e^{a}+e^{-a}} = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$

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

tanh(a) = 2σ(2a) - 1

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 parameters change.

(From above): tanh(a) = 2σ(2a) - 1 Solving for the sigmoid: σ(a) = $\frac{1}{2}\left(tanh\left(\frac{a}{2}\right) + 1\right)$

Plugging into the above function:

$\sigma\left(\frac{x-\mu_j}{s}\right) = \frac{1}{2}\left(tanh\left(\frac{x-\mu_j}{2s}\right) + 1\right)$

Plugging back into the network:

$y(x) = w_0 + \sum_{j=1}^{M} w_j \frac{1}{2}\left(tanh\left(\frac{x-\mu_j}{2s}\right) + 1\right)$

Distributing:

$y(x) = w_0 + \frac{1}{2}\sum_{j=1}^{M} w_j tanh\left(\frac{x-\mu_j}{2s}\right) + \frac{1}{2}\sum_{j=1}^{M} w_j$

Defining new weights and scaling parameters:

$w'_0 = w_0 + \frac{1}{2}\sum_{j=1}^{M} w_j \qquad w'_j = \frac{1}{2}w_j \qquad s' = 2s$

Final resulting network:

$y(x) = w'_0 + \sum_{j=1}^{M} w'_j tanh\left(\frac{x-\mu_j}{s'}\right)$

(ii) What does this tell you about the representational power of sigmoid vs tanh networks?
Both tanh and sigmoid can be reorganized into equivalent forms. This means that tanh and sigmoid networks have identical representational power. Any sigmoid network can be rewritten as a tanh network (by scaling parameters), and vice-versa.

(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.*
Developers likely prefer tanh over sigmoid for multiple reasons. Firstly, tanh is centered on zero, allowing us to map both positive and negative values cleanly, with more balanced gradients. Also, because of the scaling done in tanh, the gradients are steeper around zero, allowing clearer direction for descent while training.

PART II: CODING: (See my Jupyter Notebook submission)