CS 129
Problem Set 2

Instructors: Andrew Ng, Younes Bensouda Mourri

Due: 11:59pm Tuesday Feb 10th, 2026

Problem 1: Locally Weighted Linear Regression

We are given the following dataset relating an input independent variable x to an output dependent variable y.

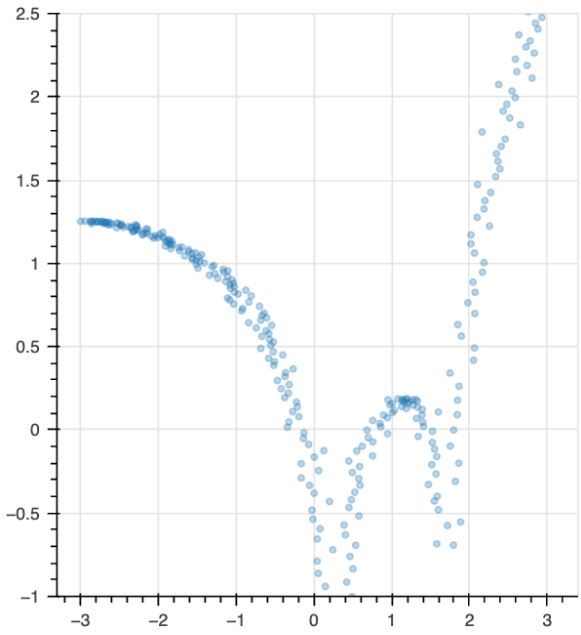

Figure 1: Observed data points

1. (2 points) Explain why fitting a simple linear regression model (i.e, $f_{w,b} = wx + b$) to this dataset would likely result in underfitting.

   Your Solution Here

2. (2 points) One alternative is to fit a high-degree polynomial regression model (i.e, $f_{w,b} = w_1x + w_2x^2 + ... + b$). What practical difficulty arises when choosing this type of model for the data shown? *[Hint: Think about what must be specified before training.]*

   Your Solution Here

3. (4 points) Locally weighted regression addresses these issues by fitting a separate linear model for each query point (the point for which we are making the prediction). In contrast to standard linear

regression, which is a **parametric** model with a single global set of parameters, locally weighted regression is a **non-parametric** method. This means the model adapts to the data locally rather than assuming one fixed functional form across the entire input space.

For a query point x_q, locally weighted regression assigns a weight to each training example $x^{(i)}$ based on its distance from x_q:

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x_q)^2}{2\tau^2}\right).$$

If $x_q = 1$, $x^{(1)} = 2$, $x^{(2)} = 5$, and $\tau = 0.5$, compute $w^{(1)}, w^{(2)}$. What happens to the weights $w^{(i)}$ as $|x^{(i)} - x_q|$ increases and decreases?

Your Solution Here

4. (4 points) τ is the bandwidth parameter the engineer selects that controls how quickly the influence of points decreases with distance.

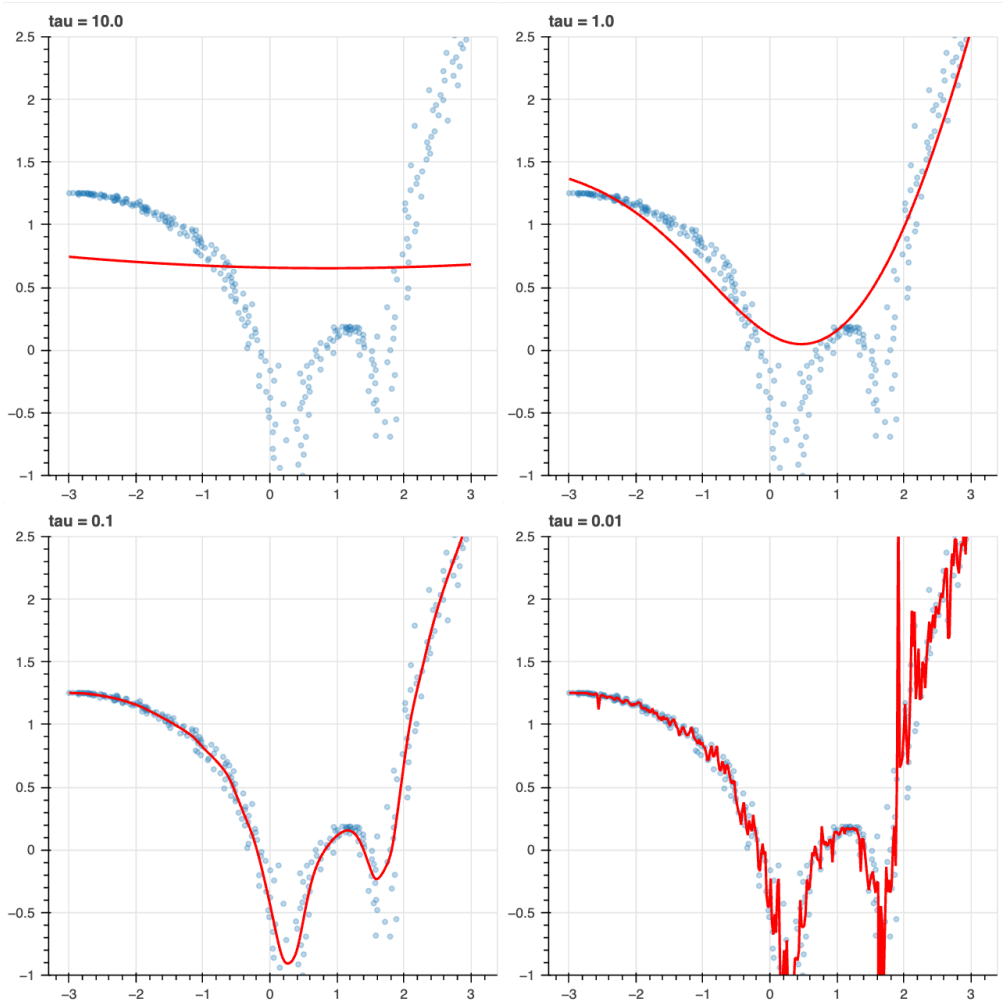

Figure 2: Locally weighted regression with varying values for τ.

What happens to the weights $w^{(i)}$ for large and small τ? What respective effect does this have on the accuracy of predictions?

Your Solution Here

5. (6 points) Once the weights $w^{(i)}$ have been computed for all training examples, locally weighted regression fits a locally weighted least squares problem:

$$J(\theta) = \sum_{i=1}^{m} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2,$$

where θ represents the parameters (or weights) of the model that we need to optimize. This is similar to ordinary linear regression, except each training example contributes according to its weight.

Each training input $x^{(i)}$ is an n-dimensional column vector. Each output $y^{(i)}$ is a scalar, and each example weight $w^{(i)}$ is also a scalar.

Let W be a $m \times m$ diagonal matrix with entries $w^{(i)}$ on the diagonal. Let X be an $m \times n$ matrix whose i-th row is $(x^{(i)})^T$, let Y be the $m \times 1$ vector of true values, and let θ be the $n \times 1$ parameter vector.

Derive the normal equations for this cost function. In other words, find θ that minimizes the cost function $J(\theta)$ as a function of X, Y, and W. *[Hint: First write $J(\theta)$ in matrix (vectorized) form.]*

Note: In Coursera, the linear regression model is written using weights w and bias b. In this question, we instead use θ to denote the model parameter vector in order to avoid confusion with the example weights $w^{(i)}$ used in locally weighted regression. For simplicity, we omit the bias term b in this derivation; including it would not change the main ideas of the computation.

Your Solution Here

6. (2 points) Finally, the prediction at the query point is computed using the same form as

$$\hat{y}_q = \theta^T x_q.$$

Now that you have seen how locally weighted regression words, what is a limitation to this method? *[There are many answers; you only need one.]*

Your Solution Here

Problem 2: Bias-Variance tradeoff

One of the key results in Machine Learning is the Bias-Variance tradeoff. We will see in class the intuition and the implications of this important result but in this question, you will prove the result.

Linear Regression Assumptions. In linear regression, the labels are assumed to be generated with the following assumptions:

- $Y = f(\vec{x}) + \epsilon$ (ϵ is a random variable that represents noise, f is deterministic and is a model that maps $\vec{x}$ to Y)
- ϵ is independent of $\vec{x}$ and therefore also independent of $f(\vec{x})$
- ϵ has mean 0 and variance σ^2

Using a dataset of $(\vec{x}^{(i)}, y^{(i)})$ pairs, we build $\hat{f}$ which is an estimator of the underlying true model. Please note the following two things:

- Because the labels (the y's) in the train dataset are random variables, $\hat{f}$ is a random variable.
- While the labels are random variables, you can consider that the input data (each $\vec{x}$) as given. Therefore, you should treat it as deterministic (a constant) and not a random variable. As a result, $f(\vec{x})$ is also deterministic.

Expected Error. From here on out, let $\vec{x}$ denote a single test example. The error of the model on this single data point is defined as:

$$\text{Err} = \mathbb{E}\Big[\Big(Y - \hat{f}(\vec{x})\Big)^2\Big]$$

where Y is the true label which is a random variable.

1. (4 points) Prove the following:

$$\text{Err} = \mathbb{E}\Big[\Big(f(\vec{x}) - \hat{f}(\vec{x})\Big)^2\Big] + \mathbb{E}\big[\epsilon^2\big] + 2\mathbb{E}\Big[\Big(f(\vec{x}) - \hat{f}(\vec{x})\Big)\,\epsilon\Big]$$

   Your Solution Here

2. (4 points) Prove the following:

$$\mathbb{E}\Big[\Big(f(\vec{x}) - \hat{f}(\vec{x})\Big)\,\epsilon\Big] = 0$$

   Your Solution Here

3. (4 points) We define the bias of the model as the expected distance between the model and the hypothesis: $\text{Bias} = \mathbb{E}\big[f(\vec{x}) - \hat{f}(\vec{x})\big]$. Prove the following:

$$\mathbb{E}\Big[\Big(f(\vec{x}) - \hat{f}(\vec{x})\Big)^2\Big] = \text{Var}\big[\hat{f}(\vec{x})\big] + \mathbb{E}\big[f(\vec{x}) - \hat{f}(\vec{x})\big]^2$$

   Your Solution Here

4. (4 points) Derive the expression of the error. Based on this expression, is it possible to achieve a model with 0 expected error? Why, or why not? *Note*: your result should only depend on $\text{Var}\big[\hat{f}(\vec{x})\big]$, Bias, and σ.

   Your Solution Here

Problem 3: Softmax Classification

In this question, you will revisit the softmax method for multi-class classification you saw on Coursera. Assume there are K classes. We define a weight matrix W such that W_i represents a column vector of weights used to classify class i. The probability assumption of the softmax model for a given class k and datapoint $\vec{x}$ is the following:

$$\mathbb{P}\big(Y = k|\vec{x}, W, b\big) = \frac{1}{Z} e^{\vec{x}^T W_k + b_k}$$

Note 1: $\vec{x}$ is a column vector.

1. (4 points) Compute Z, i.e. find an expression of $\mathbb{P}\big(Y = k|\vec{x}, W, b\big)$ that only depends on W_i, b_i, and $\vec{x}$ (where i is any specific class).

   Your Solution Here

2. (4 points) After computing the probability of belonging to class k for all k, how do we assign a class? In other words, given the probabilities, what is the decision rule?

   Your Solution Here

3. (4 points) One of the problems of this softmax method, is that if $\vec{x}^T W_k + b_k$ is large, its exponential will be extremely large. Therefore, we will face overflow errors. One of the methods used to mitigate this effect is to replace $\vec{x}^T W_k + b_k$ by $\vec{x}^T W_k + b_k - \alpha$ where $\alpha = \max_j [\vec{x}^T W_j + b_j]$. Show that this method does not change the probability values and explain why overflow is mitigated.

   Your Solution Here

4. (4 points) Softmax regression uses Cross-Entropy Loss to measure the performance of a classification model. The cost for a single datapoint is

   $$J(W) = -\log\big[\mathbb{P}\big(Y = k|\vec{x}, W, b\big)\big].$$

   Derive the gradient of the single-datapoint loss with respect to W_k.

   Your Solution Here

5. (4 points) For what value of K is the softmax model equivalent to logistic regression? Mathematically show this equivalence.

   Your Solution Here

Problem 4: Forward and Back Propagation

The architecture of the model you will be working with is defined below. You have 20 training examples and 5 features.

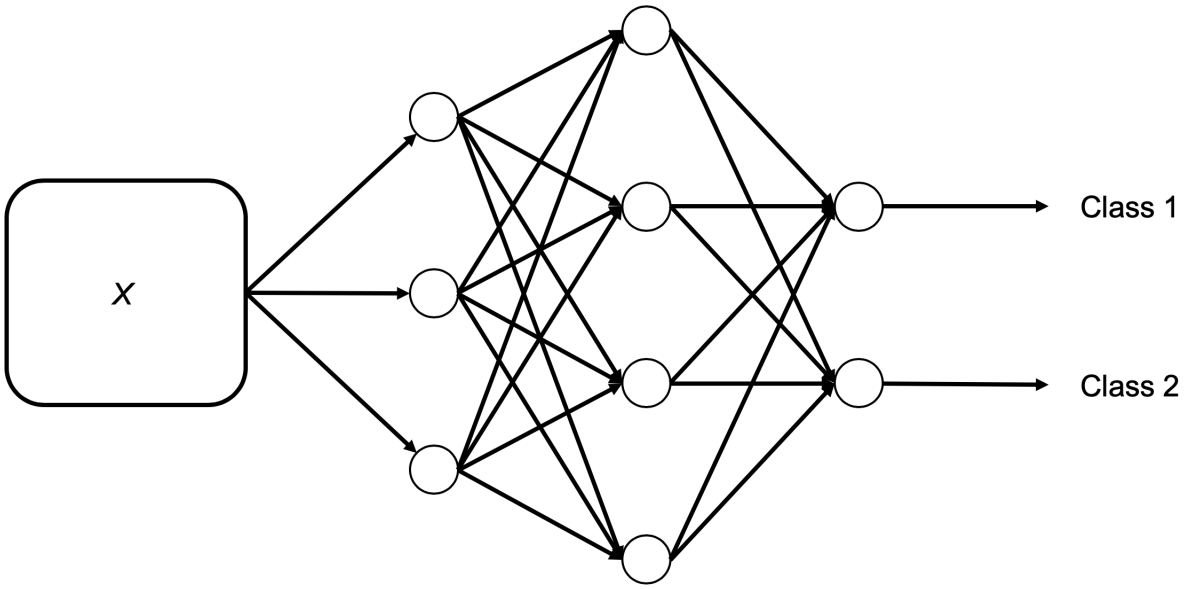

Figure 3: Architecture of the network

- Input layer
- Hidden layer 1 (dim = 3)
- Hidden layer 2 (dim = 4)
- Output layer (dim = 2)

Notes:

1. Let g denote the sigmoid activation function.
2. For a matrix X whose columns are composed of training examples x_i, the forward propagation for layer $l \in \{1, 2, 3\}$ is defined as

$$Z^{[l]} = (A^{[l-1]})^T W^{[l]} + 1_{20\times 1} b^{[l]}, \quad \text{where}$$

   - $A^{[l-1]}$ is the input to layer l ($A^{[0]} = X, (A^{[1]})^T = g(Z^{[1]}), (A^{[2]})^T = g(Z^{[2]})$),
   - $W^{[l]}$ is the weight matrix for layer l,
   - $b^{[l]}$ is the bias vector for layer l,
   - $Z^{[l]}$ is the pre-activation output of layer l,
   - $A^{[l]}$ is the post-activation output of layer l.

1. (1 point) What is the dimension of X, the matrix that contains all the example inputs (excluding bias), that you are going to feed into your neural network?

   Your Solution Here

2. (3 points) What are the dimensions of $W^{[1]}$, $W^{[2]}$, $W^{[3]}$?

   Your Solution Here

3. (3 points) What are the dimensions of $b^{[1]}$, $b^{[2]}$, $b^{[3]}$ (we're asking for number of rows, and number of columns for each)?

   Your Solution Here

4. (1 point) How many parameters are we training in our model? *Hint*: do not forget the bias!

   Your Solution Here

5. (2 points) What are the dimensions of $A^{[1]}$, $A^{[2]}$, where $A^{[1]}$ corresponds to the input to the second layer. *Hint*: consider how many examples we are tracking through the layers.

   Your Solution Here

6. (4 points) Write the feed-forward equations of this neural network for a single training example (column) vector x_i, from the beginning to the cost function J (i.e., give expressions for $Z^{[1]}, (A^{[1]})^T, Z^{[2]}, (A^{[2]})^T, Z^{[3]}, \hat{y}^T, J$). Make sure your dimensions match. You can use the functions `Softmax`(z) (the softmax function) and `CE`$(\hat{y}, y)$ (the cross-entropy function) in your solution. No coding! *Note*: assume that the activation function is the sigmoid, g, for the first and second layer. For the last layer, use a softmax function. *Hint*: do not forget the bias!

   Your Solution Here

7. So far, we only used the sigmoid function as an activation function. However, there are other popular functions.

   (a) (2 points) A similar function to the sigmoid is the hyperbolic tangent:

   $$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

   Prove that tanh is bounded by 1, i.e.

   $$\forall x, |\tanh(x)| < 1$$

   Your Solution Here

   (b) (3 points) Compute the derivative of tanh. We proved that the derivative of the sigmoid, $g(x)$, is $g(x)(1 - g(x))$. In a similar way, you should be able to find an expression of the derivative of tanh that only depends on tanh.

   Your Solution Here

   (c) (10 points) Let's now use the hyperbolic tangent as the activation function in the two hidden layers. Write the vectorized backpropagation equations of this modified neural network, for a single training example vector x. In other words, compute $\frac{\partial J}{\partial W^{[1]}}, \frac{\partial J}{\partial W^{[2]}}, \frac{\partial J}{\partial W^{[3]}}$. *Note*: $\frac{\partial J}{\partial W^{[l]}}$ it defined such that it has the same shape as W. *Hint*: we still use the softmax for the last layer (so there is nothing to modify in that step).

   Your Solution Here

Problem 5: Geometric Understanding of Regularization

We consider a linear regression model with two input features, weights, and a bias:

$$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + b$$

The cost function is:

$$J(w_1, w_2, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right)^2$$

Gradient descent updates the parameters to minimize prediction error.

Two common regularized versions are:

$$\textbf{L2 (Ridge):} \quad J_{L2}(\vec{w}) = J(\vec{w}) + \frac{\lambda}{2m} \left(w_1^2 + w_2^2 \right)$$

$$\textbf{L1 (Lasso):} \quad J_{L1}(\vec{w}) = J(\vec{w}) + \frac{\lambda}{2m} \left(|w_1| + |w_2| \right)$$

where $\lambda > 0$ controls the strength of regularization.

Note: In the gradient expressions below, constant factors (such as $1/m$) are absorbed into λ for simplicity.

1. (2 points) Without regularization, what objective is gradient descent minimizing? After adding regularization, what additional goal does gradient descent now have besides reducing prediction error?

   Your Solution Here

2. (2 points) Imagine plotting:

   - w_1 on one horizontal axis
   - w_2 on the other horizontal axis
   - The cost $J(w_1, w_2)$ on the vertical axis

   This forms a 3D bowl-shaped surface.

   (a) Explain why this surface has a single global minimum when the problem is convex.

   (b) If we take a horizontal slice of this surface at a fixed cost value, what kind of shape do you expect to see in the (w_1, w_2) plane? Explain why.

   Your Solution Here

3. (2 points) When we use least-squares loss, fixing the cost to a constant value produces smooth, closed curves in the (w_1, w_2) plane. Why do **smaller ellipses** represent **lower cost**?

   Your Solution Here

4. (2 points) In the 3D plot of (w_1, w_2, J), the cost surface forms a convex bowl with one lowest point. Elliptical contours are horizontal slices of this surface. Explain why the center of the ellipses corresponds to the minimum-cost solution found by gradient descent without regularization.

   Your Solution Here

5. (4 points) For L2 regularization, the gradient update becomes:

$$\frac{\partial J_{L2}}{\partial w_i} = \frac{\partial J}{\partial w_i} + \lambda w_i$$

(a) Explain why this creates a pull toward zero that gets weaker as weights become small.

(b) Why does this cause weights to shrink smoothly but rarely become exactly zero?

Your Solution Here

6. (3 points) For L1 regularization, the gradient update becomes:

$$\frac{\partial J_{L1}}{\partial w_i} = \frac{\partial J}{\partial w_i} + \lambda \cdot \text{sign}(w_i)$$

This means the regularization term always adds $+\lambda$ or $-\lambda$, regardless of how small the weight is.

In 3–4 sentences, explain why L1 regularization applies a constant-magnitude push toward zero, can drive weights exactly to zero (producing sparse solutions), and does not cause weights to oscillate forever around zero.

Your Solution Here

7. (2 points)

(a) Under L2 regularization, do weights shrink gradually or snap to zero?

(b) Under L1 regularization, why does gradient descent often produce a sparse model (many weights exactly zero)?

Your Solution Here

8. (2 points) If you believe only a few features are truly important, which regularization method would you choose?

- L1 (Lasso)
- L2 (Ridge)

Justify your answer in one sentence.

Your Solution Here