2/5

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{d\times 1} \underbrace{\begin{bmatrix} -x^{(1)T}- \\ \vdots \\ -x^{(n)T}- \end{bmatrix}}_{n\times d} = \begin{bmatrix} x_1^1 & \cdots & x_d^1 \\ \vdots & & \\ x_1^n & \cdots & x_d^n \end{bmatrix} \underbrace{\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}}_{d\times 1} \quad \underbrace{\theta_0}_{1\times 1 \text{ scalar}}$$

- $d\times 1$: features in feature vector (ex: humidity, temp, etc)
- $n\times d$: datapts for each row

same dim as $X\theta$: $\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

X_i^j: j = dp index, i = dim/feat

n = # of datapts

d = # of dimensions/features

$$\mathcal{E}_{train}^{h} = \frac{1}{n}\sum_{i=1}^{n}(\underbrace{\theta^T x_i + \theta_0}_{\text{prediction}} - \underbrace{y_i}_{\text{actual value}})^2$$

$\frac{1}{n}$: normalize error

MATRIX DERIVATIVES:

scalar by vector: $\frac{\partial y}{\partial \vec{x}} = \begin{bmatrix} \partial y/\partial x_1 \\ \vdots \\ \partial y/\partial x_n \end{bmatrix}$ $m=1$

vector by vector: $\frac{\partial \vec{y}}{\partial \vec{x}} = \begin{bmatrix} \partial y_1/\partial x_1 & \cdots & \partial y_m/\partial x_1 \\ \vdots & & \vdots \\ \partial y_m/\partial x_1 & \cdots & \partial y_m/\partial x_n \end{bmatrix}$ $m\times n$ (m, n)

DENOMINATOR LAYOUT: dimensions of the denominator comes first (i.e. its the rows)

MSE in MATRIX FORM: $J(\theta) = \frac{1}{n}(X\theta - Y)^T(X\theta - Y)$

takeaways:

- always check shapes
- useful to calculate individual components of derivative sometimes then combine them
- answer looks close to 1D calculus

6.390 Introduction to Machine Learning

Recitation Topic: Intro to ML

We will begin our machine learning adventures by investigating the supervised learning problem known as regression. We are given a training data set $\mathcal{D}_{\text{train}} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ of n pairs containing a feature vector $x \in \mathbb{R}^d$ and a label $y \in \mathbb{R}$. Our goal will be to learn a model that takes a new feature vector and produces an accurate corresponding label.

The first hypothesis class that we will consider is the class of linear regressors, which take the form:

$$h(x; \theta_1, \ldots, \theta_d, \theta_0) = \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_d x_d + \theta_0.$$

1. What do $x_1, x_2, \ldots, x_d$ represent? What are $\theta_1, \theta_2, \ldots, \theta_d$? Finally, what is θ_0?

   x's = feature vector
   θ's = weight
   θ_0 = bias (y-int)

2. Let's consider how to represent the hypothesis h above in a more compact form. Define columns vectors $x_{\text{aug}}, \theta_{\text{aug}} \in \mathbb{R}^{d+1}$ such that,

   $$\theta_{\text{aug}}^{\top} x_{\text{aug}} = \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_d x_d + \theta_0,$$

   where the "aug" denotes that θ, x have been augmented.

   $\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_d \\ \theta_0 \end{bmatrix} [x_1 \ldots x_d, 1] = \theta_1 x_1 + \ldots + \theta_d x_d + \theta_0$

3. In order to evaluate a hypothesis h, we would like to consider the squared error loss function between the "guess", $g = h(x; \theta, \theta_0)$, and the actual label y:

   $$\mathcal{L}(g, a) = (g - a)^2.$$

   Then, we can consider the mean squared error between over all n of the pairs in our training data set:

   $$\mathcal{E}_{\text{train}}(h) = \frac{1}{n} \sum_{i=1}^{n} (\theta^{\top} x^{(i)} + \theta_0 - y^{(i)})^2$$

   Let's see how we can vectorize this function into a convenient form for numerical computation.

(a) As a warm up, consider a vector $v = [v_1, v_2, \ldots, v_n]^\top$. Expand $v^\top v$ into a finite sum form.

$v^T v = [v_1 \cdots v_n] \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = v_1^2 + v_2^2 + \ldots + v_n^2$

(b) Let's find a v such that our mean squared error term can be expressed as $\mathcal{E}_{\text{train}}(h) = \frac{1}{n} v^\top v$. Specifically, what is v_i, the ith element of v? You may use $x^{(i)}, y^{(i)}, \theta$, and θ_0.

$\mathcal{E}_{train} = \frac{1}{n} \sum_{i=1}^{n} (\theta^T x_i + \theta_0 - y_i)^2$

$\theta^T x = y$

$v_i = \theta_i^T x_i + \theta_0 - y^i$

A x offset b

4. We define matrices

$\mathcal{E}_{train} = \frac{1}{n} \sum_{i=1}^{n} (\theta^T x_i + \theta_0 - y_i)^2$ (prediction, actual value)

actual vals of features

city 1

city 2

$$X_{\text{aug}} = \begin{bmatrix} x^{(1)\top} & 1 \\ x^{(2)\top} & 1 \\ \vdots & \\ x^{(n)\top} & 1 \end{bmatrix}, Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix},$$

i.e., each row of X, Y each correspond to an (augmented feature vector, label) pair from our training data set.

(a) What are the shapes of X_{aug} and Y?

X_{aug} shape = $n \times 2$ $n \times (d+1)$

Y shape = $n \times 1$

(b) Redefine v in terms of X_{aug}, Y, and θ_{aug}.

$(d+1)$

$n \times 2$ 2×1 $n \times 1$

$V = X_{aug} \theta_{aug} - Y$ shape: $n \times 1$

Let's put everything together: write

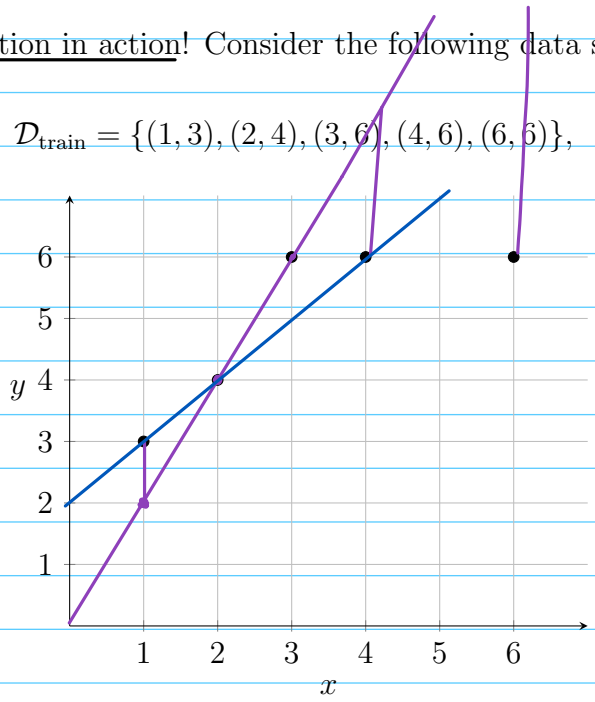
$$\mathcal{E}_{\text{train}}(h) = \frac{1}{n}\sum_{i=1}^{n}(\theta^\top x^{(i)} + \theta_0 - y^{(i)})^2$$

in its vectorized form.

$\mathcal{E}_{\text{train}}(h) = \frac{1}{n}(X_{\text{aug}}\Theta_{\text{aug}} - Y)^T(X_{\text{aug}}\Theta_{\text{aug}} - Y)$ (1×n, n×1)

5. Let's see our loss function in action! Consider the following data set where $d = 1$:

$$\mathcal{D}_{\text{train}} = \{(1,3),(2,4),(3,6),(4,6),(6,6)\},$$



$\frac{1}{n}\sum_{i=1}^{n}(\theta^T x_i + \theta_0 - y_i)^2$ — mean squared error

(a) Suppose that we have the hypothesis $h_1(x) = 2x$. Compute the error on this data set using the finite sum form.

$\text{error} = \frac{1}{5}[(3-2)^2 + (0)^2 + (0)^2 + (12-6)^2 + (8-6)^2]$
$= \frac{1}{5}[1 + 2^2 + 6^2] = \frac{41}{5}$

(b) Now, define $X_{\text{aug}}, Y, \theta_{\text{aug}}$ and compute the training error using the vectorized form.

$\theta_{\text{aug}} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ — 2x; y-int = 0

$X_{\text{aug}} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 6 & 1 \end{bmatrix}$ $Y_{\text{aug}} = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 6 \\ 6 \end{bmatrix}$

(c) Let's consider a different hypothesis, $h_2(x) = x + 2$. Compute the error on the training data set, using the vectorized form.

h(x) = x+2 is better fit

x+2 will be LOWER MSE (better fit; smaller errors)

6. Before closing our first recitation, let's refresh multi-variable calculus. It is important to check the shapes of matrices and vectors when computing complicated derivatives

Throughout, we use **denominator layout:** If $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, then

$$\frac{\partial b}{\partial a} \in \mathbb{R}^{n \times m}, \qquad \left(\frac{\partial b}{\partial a}\right)_{ij} = \frac{\partial b_j}{\partial a_i}.$$

It might be helpful to refer to the notes Appendix A.

(a) As a warm-up, let's practice taking derivatives of scalar functions with respect to vectors. Consider the scalar function $f(\theta) = \theta_1^2 + 3\theta_2 + 5$ where $\theta = [\theta_1, \theta_2]^\top \in \mathbb{R}^2$. What is $\frac{\partial f}{\partial \theta}$ using denominator layout? 2 x 1

$\frac{\partial f}{\partial \theta}_{m=1} = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ 3 \end{bmatrix}$ f = scalar function (one output)

(b) Let's now compute the derivative of a bit more complicated thing. Consider the mean squared error in matrix form.

$$J(\theta) = \frac{1}{n}(X\theta - Y)^\top (X\theta - Y).$$

$J(\theta) = \frac{1}{n}(X\theta - Y)^T(X\theta - Y)$

Let $f(\theta) = X\theta$. What is $\frac{\partial f}{\partial \theta}$ using denominator layout?

X is nxd, θ is dx1 answer: dx1 $X\theta = \theta_1 x_{j1} + \theta_1 x_{j2} + \ldots + \theta_d x_{jd}$

$\frac{\partial f_i}{\partial \theta_i} = x_{ji} \longrightarrow \frac{\partial f}{\partial \theta} = X^T$ (X w/ rows/cols swapped) dxn dxn

(c) Now compute the gradient of $J(\theta)$ with respect to θ, i.e., $\nabla_\theta J(\theta)$. Hint: use the multivariate chain rule:

$$\nabla_\theta J(\theta) = \frac{\partial J}{\partial \theta} = \frac{\partial v}{\partial \theta}\frac{\partial J}{\partial v}$$

Note that the **order** of the terms in the multiplication **matters**, since matrix multiplication is not commutative.

$v = X\theta - Y$

$J = \frac{1}{n} v^T v$

multivar. chain rule: $\nabla_\theta J(\theta) = \frac{\partial J}{\partial \theta} = \frac{\partial v}{\partial \theta} \frac{\partial J}{\partial v}$

$\frac{\partial v}{\partial \theta} = X^T$

$\frac{\partial v^T v}{\partial v} = 2(v_1\ v_2\ \cdots\ v_n)^T = 2v$

2/12 matrix mult: $\frac{\partial(Ax)}{\partial x} = A^T$ (rec 1, 6b)

chain rule: $\frac{\partial g}{\partial x} = \frac{\partial u}{\partial x}\frac{\partial g}{\partial u}$ ← order matters

dot prod: $\frac{\partial u^T v}{\partial x} = \frac{\partial u}{\partial x}v + \frac{\partial v}{\partial x}u$

last time: 6c

$J(\theta) = \frac{1}{n}(X\theta - Y)^T(X\theta - Y) + \lambda\|\theta\|^2$

$J = 1\times 1$ (loss is scalar)

$\theta = d\times 1$

$\frac{\partial J}{\partial \theta} = d\times 1$!

residuals: $v = X\theta - Y$, $J = \frac{1}{n}v^Tv$, $\nabla_\theta J(\theta) = \frac{\partial J}{\partial \theta} = \frac{\partial v}{\partial \theta}\frac{\partial J}{\partial v} = \frac{2}{n}X^T(X\theta - Y)$ ($d\times n$ $n\times 1$)

$\frac{\partial v}{\partial \theta} = \frac{\partial(X\theta)}{\partial \theta} - \frac{\partial Y}{\partial \theta} = X^T - 0 = X^T$

$\frac{\partial J}{\partial v} = \frac{1}{n}\frac{\partial(v^Tv)}{\partial v} = \begin{bmatrix}2v_1\\ \vdots \\ 2v_n\end{bmatrix} = 2\begin{bmatrix}v_1\\ \vdots \\ v_n\end{bmatrix} = 2v$ $\quad v^Tv = v_1^2 + \ldots + v_n^2$, so $\frac{\partial(v^Tv)}{\partial v} = 2v_i$

now:

$\|\theta\|^2 = \theta_1^2 + \ldots + \theta_d^2 = \theta^T\theta$

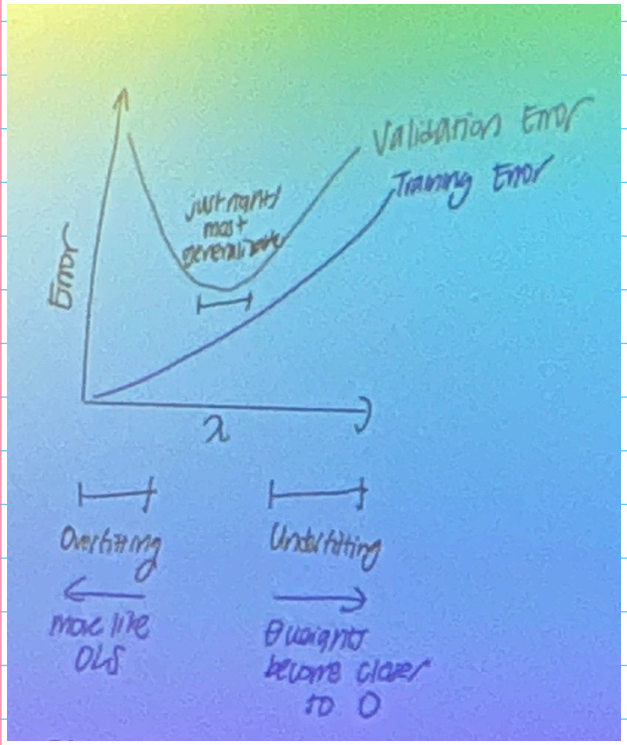
$\frac{\partial(\theta^T\theta)}{\partial \theta} = 2\theta$

$\frac{(\partial\lambda\|\theta\|^2)}{\partial \theta} = 2\lambda\theta$

$\frac{\partial J}{\partial \theta} = \frac{2}{n}X^T(X\theta - Y) + 2\lambda\theta$

* recall X^TX is $d\times d$ matrix & X is $n\times d$ matrix

X^TX invertible ↔ $\text{rank}(X^TX) = d$ ↔ $\text{rank}(X) = d$ ↔ $n \geq d$ ↔ cols of X are linearly independent



CROSS VALIDATION: mini test sets out of your training data

- helps select best regularization hyperparam

6.390 Introduction to Machine Learning

Recitation Topic: Regularization and Cross-validation

We are given a training data set and construct our data matrix $X \in \mathbb{R}^{n \times d}$ and our vector of labels $Y \in \mathbb{R}^n$. For convenience, we will assume that our data is *centered* such that an offset θ_0 is unnecessary.

Recall that we can learn linear regressors by finding the set of parameters $\theta \in \mathbb{R}^d$ that minimize the following objective function:

$$J(\theta) = \frac{1}{n}(X\theta - Y)^T(X\theta - Y),$$

which is the mean squared error between the predicted labels $X\theta$ and the true labels Y

Recall also that we can compute the gradient of this objective function as:

$$\nabla_\theta J(\theta) = \frac{2}{n}X^T(X\theta - Y).$$

Let's investigate how we can directly compute $\theta^* = \arg\min_{\theta \in \mathbb{R}^d} J(\theta)$.

1. First, we arrive at the closed-form solution by setting the gradient equal to $\mathbf{0}$—a vector of all 0's—and solving for θ:

$$\theta^* = (X^TX)^{-1}X^TY.$$

It is noteworthy that this closed-form solution assumes that we can take an inverse of a matrix. Let's look a small example to see when this may be problematic. Suppose that we are given the following data matrix with $n = 4$ feature vectors, each with $d = 2$ features, and the corresponding $n = 4$ labels:

$$X_1 = \begin{bmatrix} -2 & -4 \\ -1 & -2 \\ 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad Y_1 = \begin{bmatrix} -2 \\ -1 \\ 1 \\ 2 \end{bmatrix},$$

(a) What would happen if we tried to compute $(X_1^TX_1)^{-1}$?

$$X_1^T = \begin{bmatrix} -2 & -1 & 1 & 2 \\ -4 & -2 & 2 & 4 \end{bmatrix}$$

$$X_1^TX_1 = \begin{bmatrix} -2 & -1 & 1 & 2 \\ -4 & -2 & 2 & 4 \end{bmatrix} \begin{bmatrix} -2 & -4 \\ -1 & -2 \\ 1 & 2 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 20 & 40 \end{bmatrix} \leftarrow$$ this is linearly dependent, ∴ NOT invertible

(Copied from previous page for your convenience.)

$$X_1 = \begin{bmatrix} -2 & -4 \\ -1 & -2 \\ 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad Y_1 = \begin{bmatrix} -2 \\ -1 \\ 1 \\ 2 \end{bmatrix},$$

(b) Consider a slightly different data matrix, X_2, and another set of labels, Y_2:

$$X_2 = \begin{bmatrix} -2 & -4.00006 \\ -1 & -2.00004 \\ 1 & 2.00004 \\ 2 & 4.00006 \end{bmatrix}, \quad Y_2 = \begin{bmatrix} -1.8 \\ -1.2 \\ 1.2 \\ 1.8 \end{bmatrix}.$$

Note that Y_1 and Y_2 are very similar. The following is the result of computing $\theta_1 = (X_2^T X_2)^{-1} X_2^T Y_1$ and $\theta_2 = (X_2^T X_2)^{-1} X_2^T Y_2$:

$$\theta_1 = \begin{bmatrix} 1.0039 \times 10^0 \\ -2.9297 \times 10^{-3} \end{bmatrix}, \quad \theta_2 = \begin{bmatrix} -6.0316 \times 10^5 \\ 3.0158 \times 10^5 \end{bmatrix}$$ ← gigantic

What do you observe about the differences between the θ_1 and θ_2 derived from the analytic solution? Is this concerning, and why?

- vastly different magnitudes despite Y_1 & Y_2 being close to each other
- X^TX close to not being invertible
- → when X^TX is nearly not invertible (close to being dependent (i.e. mult.)), there are many solns that fit the data almost equally well. OLS chooses a high sloping one.

(c) Consider the linear regressor hypothesis using the parameters θ_2. What is the predicted label for the input feature vector $[1, 2]^T$? How about $[1.01, 2.03]$?

@ $[1,2]^T$, label is ≈ 0 while @ $[1.01, 2.03]^T$ it's super high.
$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \theta_2 \approx 0$ but $\begin{bmatrix} 1.01 \\ 2.03 \end{bmatrix} \theta_2$ gets super high!

2. Now, let's see how regularization can help us learn a set of parameters that are hopefully more generalizable. Recall from the course notes that,

$$\theta^*_{\text{ridge}} = (X^T X + n\lambda I)^{-1} X^T Y,$$

is the closed for solution to the ridge regression objective function,

$$J(\theta)_{\text{ridge}} = \frac{1}{n}(X\theta - Y)^T (X\theta - Y) + \lambda\|\theta\|^2.$$ → penalizes large slopes

Let's compare the learned parameters using data matrix X_2 and labels Y_1 then Y_2 choosing $\lambda = 0.4$:

$$\theta_3 = (X_2^T X_2 + n\lambda I)^{-1} X_2^T Y_1 = \begin{bmatrix} 0.1996 \\ 0.3992 \end{bmatrix}, \quad \theta_4 = (X_2^T X_2 + n\lambda I)^{-1} X_2^T Y_2 = \begin{bmatrix} 0.1916 \\ 0.3832 \end{bmatrix}.$$

(a) What is the predicted label when using the hypotheses defined by θ_3 and θ_4 for the input feature vector $[1, 2]^T$? How about $[1.01, 2.03]$?

θ_3: $\begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1996 \\ 0.3992 \end{bmatrix} = 0.998$ → $\begin{bmatrix} 1.01 \\ 2.03 \end{bmatrix} \cdot \begin{bmatrix} 0.1996 \\ 0.3992 \end{bmatrix} = 1.01$

θ_4: $\begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1996 \\ 0.3832 \end{bmatrix} = 0.996$ → $\begin{bmatrix} 1.01 \\ 2.03 \end{bmatrix} \cdot \begin{bmatrix} 0.1996 \\ 0.3832 \end{bmatrix} = 0.98$

* not a big change (low sensitivity) good!

(b) What would you expect to happen to the magnitude of the learned parameters if a larger choice of λ was used?

- closer to horizontal line (slope closer to 0)

(c) Consider the effect of extreme values of λ. What behavior would you expect when λ is very small (close to 0)? What about when λ is very large (e.g., $\lambda = 10^6$)?

λ very small: slope close to 0, OLS objective

λ large: $\theta \approx 0$, predicting a constant (the mean)

→ $\lambda\|\theta\|^2$ dominates the loss, so it'll push large slopes (θ) toward 0

generally not good to do.

3. We have seen how ridge regression introduces the hyperparameter λ to control the magnitude of the learned parameters. But how should we evaluate and choose λ in practice?

One common approach is *cross-validation*.

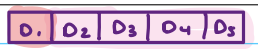
Suppose we have access to a dataset D. Also suppose that there is a dataset called D_{test} that we can use to check the performance of our hypothesis but don't have access to during training or validation. For each candidate λ, we perform the following procedure:

1. Divide D into k chunks $D_1, \ldots, D_k$.
2. For each candidate value of λ:
   (a) For $i = 1$ to k:
      i. Train a ridge regressor h_i using $D \setminus D_i$ (i.e., all training folds except D_i)
      ii. Compute the chunk-i validation error $E_i(h_i)$ on D_i.
   (b) Compute the average validation error E_λ :=???.
3. Choose λ^* with the smallest E_λ.
4. Retrain a final model h^* using all of D.
5. Evaluate h^* on D_{test}.

(a) Complete 3 (b) by writing a formula for E_λ?

$E_\lambda = \frac{1}{k} \sum_{i=1}^{k} E_i(h_i)$

$\varepsilon_{val}(\lambda) = (\varepsilon_1 + \ldots + \varepsilon_k)/k$



train on $D_2 \to D_5$, test on D_1

generally, $k = 5$

(b) Do you use regularization or no regularization in step 4? Explain your reasoning?

~~no regularization b/c~~

YES regularization b/c we're solving for λ^*

(c) What benefit does cross-validation provide compared to using a single validation split?

- more validation checks that all support each other
- averaging lets us reduce noise in our validation MSE
- gives us sense of spread/variance

2/19 ML RECAP:

1) DATA: training samples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$

2) HYPOTHESIS (model): linear regression $(h(x;\theta) = \theta x)$

3) OBJECTIVE FUNCTION (loss): evaluating model performance

- ORDINARY LEAST SQUARES: minimizing sum of squares
- RIDGE REGULARIZATION: adding penalty term $(\lambda\|\theta\|^2)$ to avoid overfitting & handle multicollinearity

ANALYTICAL soln: set $\nabla J(\theta) = 0$

OLS: $(X^TX)^{-1}X^Ty$, Ridge $(X^TX + \lambda I)^{-1}X^Ty$ (full rank, invertible)

GRADIENT DESCENT

$\theta^{t+1} = \theta^t - \eta \nabla f(\theta^t)$

η: LR; $\nabla f(\theta^t)$: steepest direction (downhill b/c −)

6.390 Introduction to Machine Learning
Recitation Topic: Gradient Descent

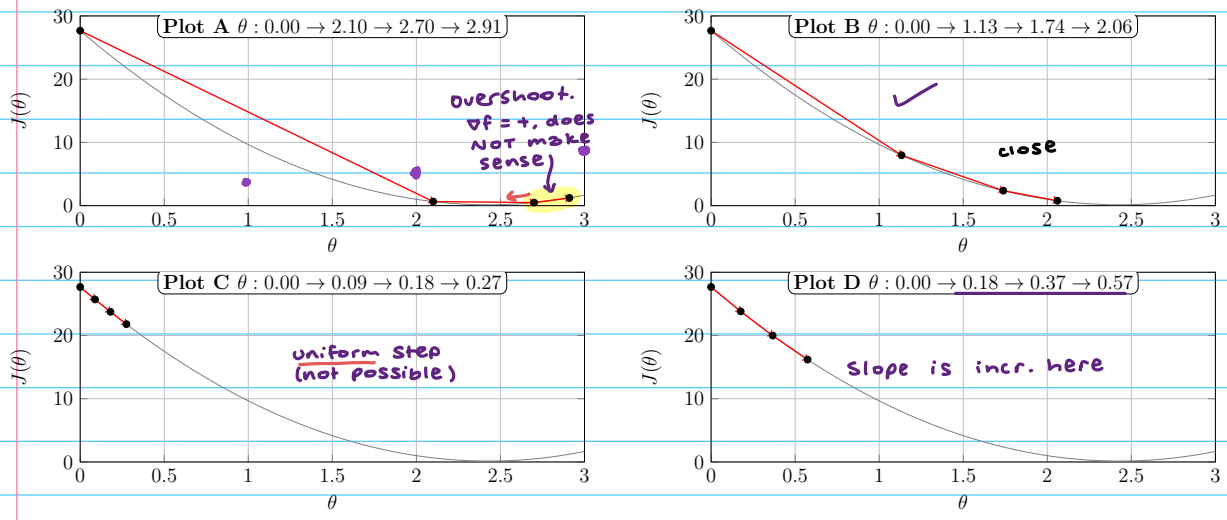
1. Given a training dataset with 3 data points having 1-dimensional features:

$$\mathcal{D}_{\text{train}} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)})\} = \{(1,3), (2,5), (3,7)\}$$

We aim to learn a linear regressor $h(x;\theta) = \theta x$ (no offset term) to minimize the MSE $J(\theta)$. We were able to show that the optimal $\theta^* = \frac{17}{7}$ using the closed-form solution. In this part, we focus on understanding the behavior of gradient descent instead.

We first try gradient descent (GD), with initial parameter $\theta = 0$, a constant learning rate $\eta > 0$ and we run it for 3 iterations. Which of the following could be a possible plot for this GD run?

Hint: Focus on conceptual reasoning. $x^{k+1} = x^k - \eta \nabla f(x^k)$



◯ Plot A ◉ Plot B ◯ Plot C ◯ Plot D

Brief explanation:

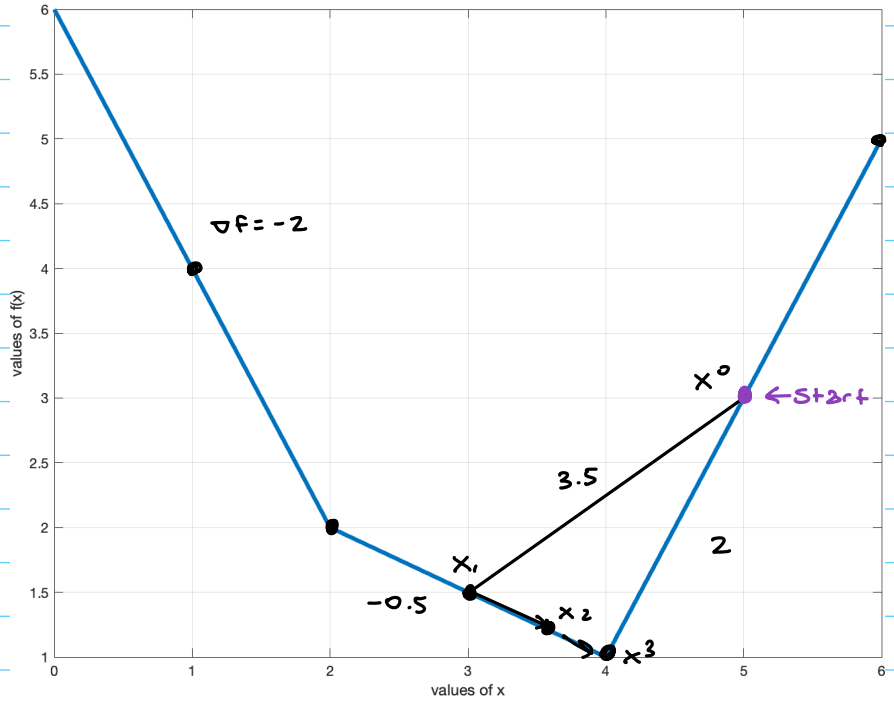
as we step towards the goal, the slope gets less steep ("tapering off"). typically we don't see constant or incr. rate of θ for good GD runs.

2. Anh is using standard gradient descent iterations

$$x^{(k+1)} = x^{(k)} - \eta \nabla f(x^{(k)}) \qquad (k = 0, 1, 2, \dots)$$

on a variety of functions f.

(a) First, Anh applies gradient descent to a piecewise-linear function $f : \mathbb{R} \to \mathbb{R}$ with the (partial) graph shown on the figure below:



At points $x = 2$ and $x = 4$, Anh uses $\nabla f(2) = -0.5$ and $\nabla f(4) = 0$.

i. Starting from the initial guess $x^{(0)} = 5$, and using step size $\eta = 1$, what will be the values of $x^{(1)}$, $x^{(2)}$, and $x^{(3)}$?

INITIAL: $x^0 = 5$

$x^1 = 5 - 1(2) = 3$ (where $2 = \nabla f(5)$)

$x^2 = 3 - 1(-0.5) = 3.5$

$x^3 = 3.5 - 1(-0.5) = 4$

ii. Anh discovers that, starting with $x^{(0)} = 1$, there are many values of $\eta > 0$ for which the gradient descent iterations produce oscillations of period 2 within the range $(0, 6)$ (i.e., $x^{(k+2)} = x^{(k)} \in (0, 6)$ for all $k = 0, 1, 2, \dots$). Find all such values of η.

$x^{k+1} = x^k - \eta \nabla f(x^k)$

$x^{k+2} = x^{k+1} - \eta \nabla f(x^{k+1})$

$x^k = x^k - \eta \nabla f(x^k) - \eta \nabla f(x^{k+1})$

$\rightarrow \nabla f(x^{k+1}) = -\nabla f(x^k)$ ← oscillation?

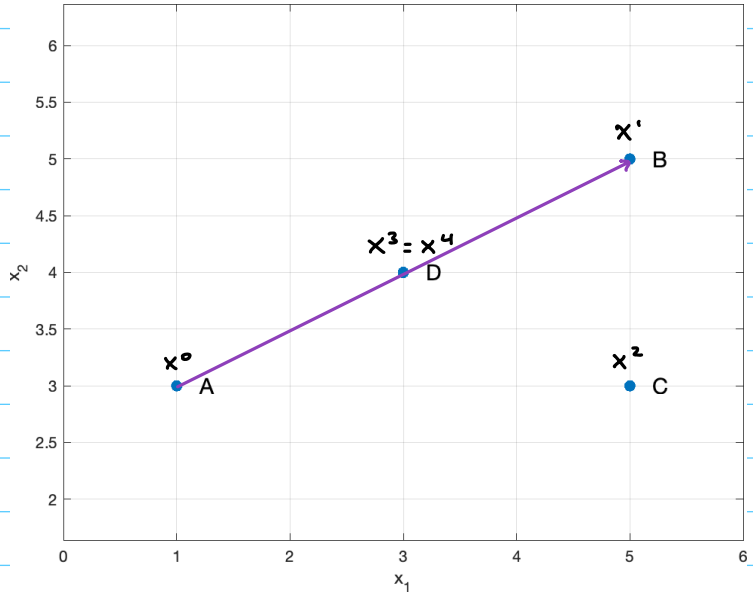
starting w/ $x^0 = 1$:

$\nabla f(x^1) = -\nabla f(x^0) = +2$ ← x^1 in $(4, 6)$

$4 < x^1 = x^0 - \eta \nabla f(x^0) < 6$

$4 < 1 + 2\eta < 6 \rightarrow 1.5 < \eta < 2.5$

(b) After mastering one-dimensional optimization, Anh applies gradient descent to a smooth function $f : \mathbb{R}^2 \to \mathbb{R}$, with $\eta = 0.1$, resulting in the sequence of points $x^{(0)} = A$, $x^{(1)} = B$, $x^{(2)} = C$, $x^{(3)} = x^{(4)} = D$ shown on the plot below:



i. Find $\nabla f(x^{(0)})$, $\nabla f(x^{(1)})$, $\nabla f(x^{(2)})$, and $\nabla f(x^{(3)})$.

$x^{k+1} = x^k - \eta \nabla f(x^k)$

$\nabla f(x^k) = \frac{1}{\eta}(x^k - x^{k+1})$

$\nabla f(x^0) = 10(x^0 - x^1) = 10 \cdot \begin{bmatrix} 1-5 \\ 3-5 \end{bmatrix} = \begin{bmatrix} -40 \\ -20 \end{bmatrix}$

$\nabla f(x^1) = 10(x^1 - x^2) = 10 \cdot \begin{bmatrix} 5-5 \\ 5-3 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$

$\nabla f(x^2) = 10(x^2 - x^3) = 10 \cdot \begin{bmatrix} 5-3 \\ 3-4 \end{bmatrix} = \begin{bmatrix} 20 \\ -10 \end{bmatrix}$

$\nabla f(x^3) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ ← b/c $x_3 = x_4$

ii. Is this statement true or false: "given the information provided, the point D **must** be a global minimum of function f"? Briefly justify your choice.

A. True. B. False.

Justification:

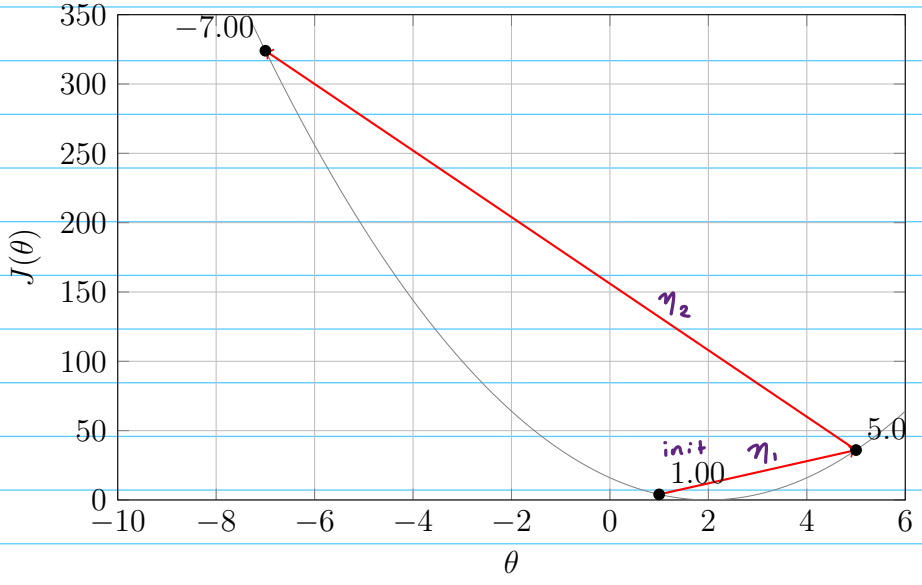
could be a local min/max or saddle pt.

3. In this problem, we consider two simple data sets, each consisting of one-dimensional features and labels. Our goal is to learn a linear regressor $h(x;\theta) = \theta x$ by minimizing the mean squared error (MSE).

(a) First, consider a trivial data set made of a single data point with feature $x = 2$ and label $y = 4$. The objective is to minimize

$$J(\theta) = (2\theta - 4)^2.$$ $=0 \rightarrow \theta = 2$ minimizes

The plots below show the behavior of running gradient descent (GD) in order to minimize this objective function. We initialize GD with $\theta_{\text{initial}} = 1$, and run the algorithm for two iterations. The first iteration uses learning rate η_1, and the second iteration η_2. The numbers labeled on the plot indicate the value of θ at each iteration.



$\theta=1$: $J(1)=4$

$\theta=5$: $J(5)=36$

$\theta=-7$: $J(-7)=-16^2$

$x^{k+1} = x^k - \eta \nabla J(x^k)$

$J(\theta) = (2\theta-4)^2 \rightarrow \nabla J(\theta) = 4(2\theta-4)$

1st iter: $5 = 1 - \eta_1 \cdot [4(2-4)]$

$4 = -\eta_1 \cdot (-8) \rightarrow \eta_1 = 0.5$

2nd iter: $-7 = 5 - \eta_2 [4(10-4)]$

$-12 = -\eta_2 (-25)$

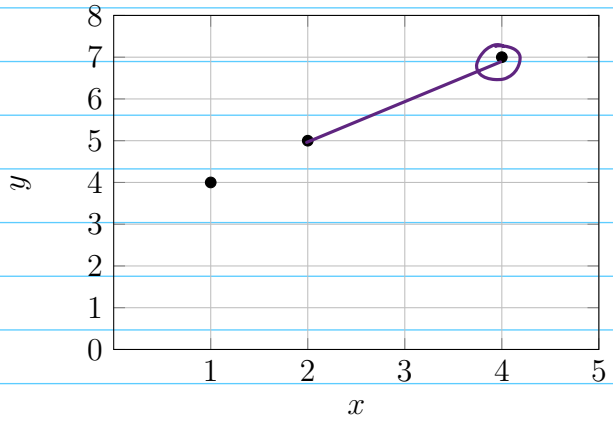
$\eta_2 = 0.5$

i. Which statement about the learning rates η_1 and η_2 is true?

- [x] $\eta_1 = \eta_2$
- [ ] $\eta_1 > \eta_2$
- [ ] $\eta_1 < \eta_2$
- [ ] not enough information to tell

(b) Now, consider the slightly more interesting training data set, consisting of 3 data points:

$$\mathcal{D}_{\text{train}} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{3} = \{(1,4),(2,5),(4,7)\}.$$



and the objective is to minimize, again, the MSE:

$$\begin{aligned} J(\theta) &= \frac{1}{3}\left[(\theta-4)^2 + (2\theta-5)^2 + (4\theta-7)^2\right] \\ &= \frac{1}{3}\left(21\theta^2 - 84\theta + 90\right) \\ &= 7\theta^2 - 28\theta + 30. \quad = 0 \end{aligned}$$

$J'(\theta) = 14\theta - 28$

init $\theta = 4$

$\theta_{new} = 4 - 0.02(28) = 3.44$ $\quad \nabla J(4) = 28$

i. Consider running gradient descent (GD) to learn θ. We initialize GD with $\theta_{\text{initial}} = 4$ and use a fixed learning rate of $\eta = 0.02$. After just one iteration of the GD update, which of the following is a *possible* value for the resulting updated parameter θ_{new}? Briefly justify your answer.

- ○ $\theta_{\text{new}} = 4$
- ○ $\theta_{\text{new}} = 3.76$
- ● $\theta_{\text{new}} = 3.44$
- ○ $\theta_{\text{new}} = 2.56$
- ○ $\theta_{\text{new}} = 0.88$
- ○ Not enough information to determine any *possible* value of θ_{new}.

$J'(\theta) = 14\theta - 28$

init $\theta = 4$

$\theta_{new} = 4 - 0.02(28) = 3.44$ $\quad \nabla J(4) = 28$

Brief justification:

ii. Consider running stochastic gradient descent (SGD) to learn θ. We initialize SGD with $\theta_{\text{initial}} = 4$ and use a fixed learning rate of $\eta = 0.02$. After just one iteration of the GD update, which of the following is a *possible* value for the resulting updated parameter θ_{new}? Briefly justify your answer.

- ● $\theta_{\text{new}} = 4$
- ● $\theta_{\text{new}} = 3.76$
- ○ $\theta_{\text{new}} = 3.44$
- ● $\theta_{\text{new}} = 2.56$
- ○ $\theta_{\text{new}} = 0.88$
- ○ Not enough information to determine any *possible* value of θ_{new}.

Sq err: $L(\theta) = (y - \theta x)^2 \rightarrow \nabla L_\theta = -2(y - \theta x)x$

$\theta_{init} = 4$ $\quad \theta_{new} = \theta_{old} - \eta \nabla J(\theta_{old})$

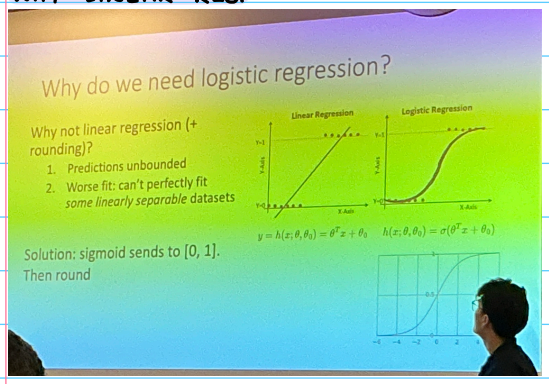
for $(x, y) = (2, 5)$: $\theta_n = 4 - 0.02[-2(5 - 4 \cdot 2) \cdot 2] = 3.76$

$(1, 4)$: $\theta_n = 4 - 0.02[-2(4 - 4 \cdot 1) \cdot 1] = 4$

$(4, 7)$: $\theta_n = 4 - 0.02[-2(7 - 4 \cdot 4) \cdot 4] = 2.56$

Brief justification:

2/26 WHY LINEAR REG.



input x ⟶ logistic regression model ⟶ output

- w/o confidences, can only rly use 0-1 loss, which can't be optimized using SGD

$\sigma(\theta^T x + \theta_0) > 0.5 \leftrightarrow \theta^T x + \theta_0 > 0$

NLL LOSS: $L_{nll}(g, y) = -[y\log(g) + (1-y)\log(1-g)]$

true label = 1: $-\log(g) = -\log(\text{confidence in correct hater})$

true label = 0: $-\log(1-g) = -\log(\text{confidence in correct hater})$

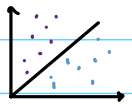
MINIMIZE NLL = MAXIMIZE CONFIDENCE IN CORRECT CLASS

EACH CLASS own weights:

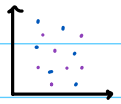
$\theta = [\theta_1\ \theta_2\ \theta_3]$ ↳ weights for class 3

$z = \theta^T x = \begin{bmatrix} -\theta_1^T- \\ -\theta_2^T- \\ -\theta_3^T- \end{bmatrix}[x] = \begin{bmatrix} \text{class 1 logit} \\ \vdots \end{bmatrix}$ "logit" = pre-softmax score

LINEARLY SEPERABLE:



NOT:



MULTICLASS:

$z = \theta^T x = \begin{bmatrix} -\theta_1^T- \\ -\theta_2^T- \\ \vdots \\ -\theta_K^T- \end{bmatrix} \quad x = \begin{bmatrix} \theta_1^T x \\ \vdots \\ \theta_K^T x \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_K \end{bmatrix}$ ← logits for each class

6.390 Introduction to Machine Learning

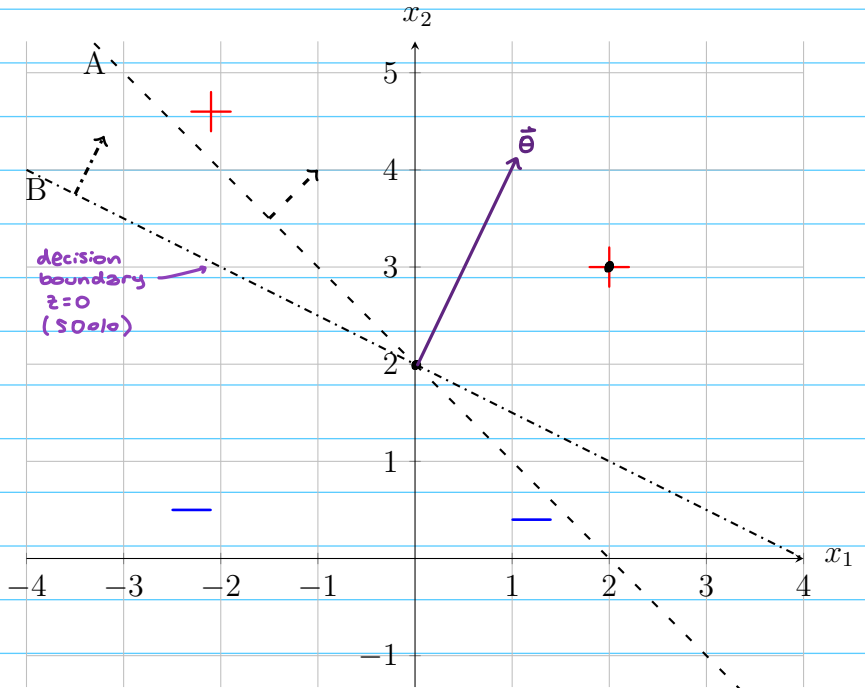
Recitation Topic: Linear Classifiers

1. Recall that a linear logistic classifier is characterized by

$$h(x; \theta, \theta_0) = \sigma(\theta^T x + \theta_0),$$

where $\sigma(\cdot)$ is the standard sigmoid function, $\sigma(z) = 1/(1 + \exp(-z))$. Define the argument of the sigmoid function in $h(\cdot)$ to be $z = \theta^T x + \theta_0$.

The plot below has two linear hypotheses with equal magnitudes of θ. Their direction normals point in the direction of positive classification. Each of the four data points is labeled as positive (+) or negative (−).



(a) Which of the following could be possible θ, θ_0 values for the hyperplane described by model B?

A. $\theta = \begin{bmatrix} -2 \\ -4 \end{bmatrix}$, $\theta_0 = 8$

B. $\theta = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $\theta_0 = -8$

C. $\theta = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $\theta_0 = -4$

D. $\theta = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$, $\theta_0 = -8$

$z = \theta^T x + \theta_0$ @ $x_1 = 0$, $x_2 = 2$

hyperplanes sit @ 0 splitting +/−

$0 = [\theta_1 \ \theta_2]\, x + \theta_0$ $x_1 = -2$ $x_2 = 3$

plug into equation

* $\vec{\theta}$ vector points in + direction (careful!!) → verify direction.

$z = \theta^T x + \theta_0$ @ $x_1 = 0$, $x_2 = 2$

$0 = [\theta_1 \ \theta_2]\, x + \theta_0$ $x_1 = -2$ $x_2 = 3$

plug into equation

* $\vec{\theta}$ vector points in + direction (careful!!) → verify direction.

(b) What value of z does model B assign to the point $(2, 3)$? What is the numerical probability outputted by the classifier?

$z = [1 \ 2]\, \vec{x}^{\,(2,3)} - 4 = 8 - 4 = 4$ (θ^T, θ_0)

$\sigma(4) = \frac{1}{1+e^{-4}} = 0.98$

(c) Recall that NLL loss is defined as $\mathcal{L}_{nll}^{(i)} = -\left(y^{(i)} \log g^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - g^{(i)}\right)\right)$ Which model has lower loss on the four data points and why?

→ model B will have higher confidences & thus lower NLL

intuition: use distance from decision boundary as proxy for confidence

50% small bc close to decision boundary

2. Now, we would like to analyze a multiclass linear logistic classifier with $K = 3$ classes. For this part of the problem, we are still working with only 2 input features (x_1, x_2), but we choose to fold θ_0 into the θ matrix by adding a row to the bottom of the θ matrix representing the θ_0's and we add a 1 to the end of each x column vector. So, in this framing, let x be a data point, $x = [x_1, x_2, 1]^T$. Our θ will be a 3×3 matrix and let $z = \theta^T x$ be a 3×1 vector with $z = [z_1, z_2, z_3]^T$. Then, the output of the model will be defined as,

$$g = \text{softmax}(z) = \begin{bmatrix} \exp(z_1)/\sum_{i=1}^{3} \exp(z_i) \\ \exp(z_2)/\sum_{i=1}^{3} \exp(z_i) \\ \exp(z_3)/\sum_{i=1}^{3} \exp(z_i) \end{bmatrix}.$$

Recall that the vector g represents the probability assigned to each of the three classes, and the class prediction is then made from the largest element of g.

(a) Suppose that we have a model defined by the following matrix:

$$\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

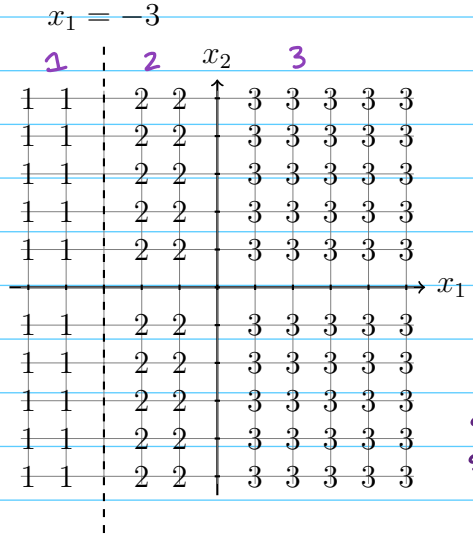
Consider the data point $x = [1, -1, 1]^T$. Compute $z = \theta^T x$ and determine which class will be assigned to x.

∴ x will be assigned to class 1

$z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ → $g = \text{softmax}(z) = \left(\frac{e^1}{e^1+e^{-1}+e^0} \quad \frac{e^{-1}}{e^1+e^{-1}+e^0} \quad \frac{e^0}{e^1+e^{-1}+e^0}\right) \approx (0.6652, 0.09, 0.2447)$

b/c softmax is monotonic → highest term before softmax will be weighed more (max logit)

(b) Examine the classification problem represented by the graph below, where points are labeled with their class: 1, 2, or 3. Also given below is a model represented by the matrix θ (note that this is θ and so the first column represents θ_1, θ_2, and θ_0 for class 1).



class: 1 2 3

$\theta = \begin{bmatrix} \theta_1 & \theta_2 & \theta_0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \\ -3 & 0 & 0 \end{bmatrix}$

$g_1(x) = -1x_1 + 0x_2 - 3$

$g_2(x) = 0$

$g_3(x) = 1x_1$

→ classification depends on x_1!

Does the provided model defined by θ perfectly separate the data as desired in the graph above? Explain.

$z = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$ → logit = 1 (class 1)

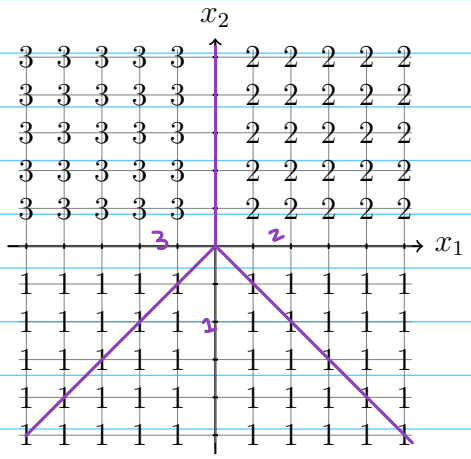
$z = \begin{bmatrix} -1 & 0 & -3 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} -x_1 - 3 \\ 0 \\ x_1 \end{bmatrix}$

$-x_1 - 3$ is largest when (based on graph) $-x_1 - 3 > 0 \rightarrow x_1 \leq -3$

$-x_1 - 3 > x_1$ (true if $x_1 \leq 3$)

class 3: when $x_1 > 0$, class 2: $-3 \leq x_2 \leq 0$ perfect classification accuracy!

(c) Examine the classification problem represented by the graph below, where points are labelled with their class: 1, 2, or 3. Also given below is a model represented by the matrix $\underline{\theta}$ (note that this is θ and so the first column represents θ_1, θ_2, and θ_0 for class 1).



$$\theta = \begin{array}{c} \begin{matrix} \theta_1 & \theta_2 & \theta_3 \end{matrix} \\ \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

$-x_2 \geq x_1$, $-x_2 \geq -x_1$

Does the provided model defined by θ perfectly separate the data as desired in the graph above? Explain. It may be helpful to plot the separator in 3D on Desmos.

3. For each of the following, determine if the statement is true or false and justify your reasoning.

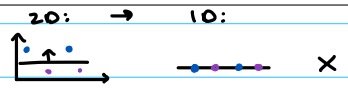
(a) If we take any linearly separable data set and add a new feature, it is still guaranteed to be linearly separable.

● True ○ False old: $z = \theta_1 x_1 + \ldots + \theta_d x_d$ new: $z = \theta_1 x_1 + \ldots + \theta_d x_d + \theta_{d+1} x_{d+1}$

can do everything original model can do & more!

(b) If we take any linearly separable data set and remove a feature, it is still guaranteed to be linearly separable.

○ True ● False



less expressivity

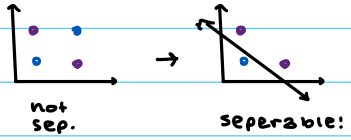
(c) If we take any data set that is not linearly separable and remove a feature, it is still guaranteed to not be linearly separable.

● True ○ False

remove feature = less expressive

(d) If we take any data set that is not linearly separable and remove a data point, it is still guaranteed to not be linearly separable.

○ True ● False



4. This problem explores the relationship between binary and multi-class classification, and shows how the multi-class formulation reduces to the binary one when $K = 2$.

Consider a one-hot-K classifier where the input is $x \in \mathbb{R}^d$ and the parameters are $\theta \in \mathbb{R}^{d \times K}$. Let θ_k denote the kth column. The logits are computed as

$$z_k = \theta_k^\top x,$$

and the softmax probabilities are

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

multi-class:
K probabilities (for K classes)
→ but we only need K-1 to deduce (b/c we can deduce last probability)

(a) How many independent values are needed to specify a probability distribution over K classes? How many outputs does softmax actually produce? Why do you think this discrepancy exists? Does binary classification encounter the same issue?

K-1 degrees of freedom

softmax: gives all K outputs of probabilities ← we only need K-1 (but overspecifying doesn't matter)

binary: only outputs $\sigma(z)$ = probability of class 1

(b) Now consider $K = 2$ with a dataset of n examples: $x_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$. Suppose you train both a binary classifier (producing parameter $\theta \in \mathbb{R}^d$) and a multi-class classifier (producing parameters $\theta_1, \theta_2 \in \mathbb{R}^d$) on this same dataset. Assume that the training converges to the global optimum in both cases.

Show how the binary parameter θ relates to the multi-class parameters θ_1 and θ_2.

Note that in the multi-class setting, the classes are generally expressed as class 1 and class 2. In the binary setup, the classes are generally written as 0 and 1. For this problem, assume that class 2 in the multi-class setting corresponds to class 0 in the binary one.

Hint: Start by writing the expression for p_1 in the multi-class setting. Can you rewrite it to match the form $p = \sigma(z)$ from binary classification? What relationship does this reveal between the logits z, z_1, and z_2? How can you use this to derive a relationship between θ, θ_1, and θ_2?

$\Pr[\text{class } 1]_{\text{multiclass}} = \Pr[\text{class } 1]_{\text{binary}}$

$$\frac{e^{z_1}}{e^{z_1}+e^{z_2}} = \frac{1}{1+e^{-z}}$$

$$\frac{1}{1+e^{z_2-z_1}} = \frac{1}{1+e^{-z}} \quad \leftrightarrow \quad z_1 - z_2 = z \quad \leftrightarrow \quad \theta_1^T x - \theta_2^T x = \theta^T x$$

$$(\theta_1-\theta_2)^T x = \theta^T x \quad \rightarrow \quad \theta_1 - \theta_2 = \theta$$

(c) Finally, show that the softmax cross-entropy loss (negative log-likelihood multiclass) reduces to binary cross-entropy (negative log-likelihood) when $K = 2$. Specifically, prove that

$$L = -\sum_{k=1}^{2} y_k \log p_k$$

simplifies to

$$L = -\left[y \log \sigma(z) + (1-y) \log(1-\sigma(z))\right],$$

where y and z are appropriately defined.

$-\sum_{k=1}^{K} y_k \log(g_k) = -(0+0+\ldots+\log(g_i)+0\ldots) = -\log(g_i) = -\log(\text{confidence in correct class})$

midterm 2 content

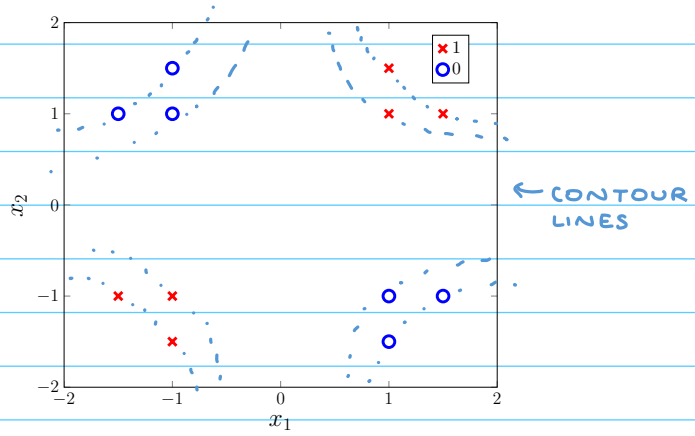
3/5 **NEURAL NETWORKS:**

- how to find appropriate feature transforms?
- this rec is 2D data, small # pts
- in general, must automate the process with ML!

6.390 Introduction to Machine Learning

Recitation Topic: Features and Neural Networks

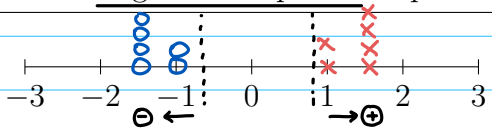
1. Consider the following data set:



Assume that $x^{(1)}, \ldots, x^{(n)}$ are the feature vectors for the points shown on the figure and $y^{(1)}, \ldots, y^{(n)}$ are the corresponding labels (for example, $y^{(i)} = 1$ for the "×" points, $y^{(i)} = 0$ for the "○" points). By inspection, we can see that this training data set is not linearly separable.

We would like to design non-linear transformations such that the training data set becomes linearly separable.

(a) Consider the feature transformation $\phi_1 : \mathbb{R}^2 \to \mathbb{R}$ defined as $\phi_1(x_1, x_2) = x_1 x_2$. On the line below, plot where the original data points map to in the transformed space.



★ (b) Define a hypothesis $h_1(x; \theta, \theta_0) = \sigma(\theta\phi_1(x) + \theta_0)$ that achieves 100% training accuracy.

$\theta = 1, \theta_0 = 0$

$h_1(x; \theta, \theta_0) = \sigma(\phi_1(x)) = \sigma(x_1 x_2)$

× > 0 → label 1

○ < 0 → label 0

(c) Suppose we fix the weight parameter $\theta = 1$ for our hypothesis $h_1(x; \theta, \theta_0) = \sigma(\theta\phi_1(x) + \theta_0)$. Based on the data points shown in the figure, what is the valid range of values for θ_0 that results in 100% training accuracy? Justify your answer.

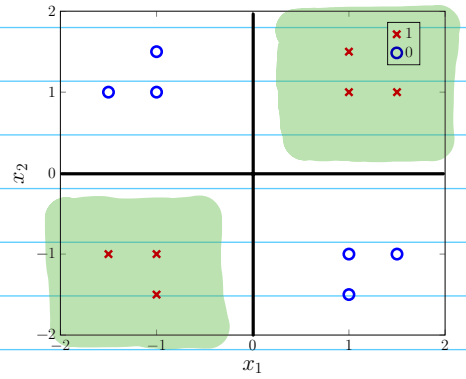
$-1 < \theta_0 < 1$ based on the gap b/t the differently labelled data pts

$h(x; \theta, \theta_0) = \sigma(\overset{1}{\theta}\phi_1(x) + \theta_0) = \sigma(\phi_1(x) + \theta_0)$

$\phi_1(x) + \theta_0 > 1$ for red

$\phi_1(x) + \theta_0 < 1$ for blue

(d) Assume we use the weight $\theta = 1$ and the offset $\theta_0 = 0$ in our hypothesis. On the plot below, shade the regions that would be labeled 1 by the hypothesis h_1.



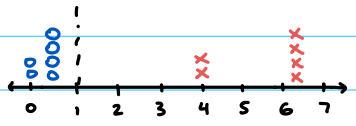
$h_1(x; 1, 0) = \sigma(\phi_1(x))$

$h_1(x; 1, 0) = \sigma(x_1, x_2)$

(+1) red: $\sigma(x_1, x_2) > 0.5$

(e) Consider a new transformation $\phi_2 : \mathbb{R}^2 \to \mathbb{R}$ defined as $\phi_2(x) = x_1^2 + 2x_1x_2 + x_2^2$. Will the training data set transformed by ϕ_2 be linearly separable? $(x_1 + x_2)^2$

$h_1(x; \theta, \theta_0) = \sigma(\theta\phi_2(x) + \theta_0) = \sigma(\theta(\underbrace{x_1^2 + 2x_1x_2 + x_2^2}_{(x_1 + x_2)^2}) + \theta_0)$



← this is linearly seperable!

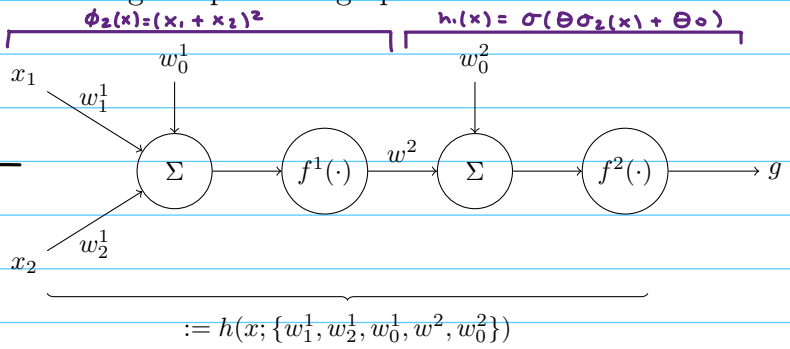
ex: $\theta = 1$, $\theta_0 = +2$

$f^2(w^2(f^1(w_1^1 x_1 + w_2^1 x_2 + w_0^1)) + w_0^2)$
$h_2(x) = \sigma(\theta \phi_2(x) + \theta_0)$

$\theta_0 = w_0^2 \qquad \theta = w^2$

$\sigma = f^2$

(f) Consider the following computation graph:



Write out the definition of the hypothesis visualized in the graph given above. Then, determine activation functions f^1, f^2 and weights such that $h_2(x) = \sigma(\theta\phi_2(x) + \theta_0)$, where $\phi_2(x)$ is the same transformation from part (e).

$g = f^2(w^2 f^1(x_1 w_1^1 + x_2 w_2^1 + w_0^1) + w_0^2)$

$\phi_2(\cdot) = x_1^2 + 2x_1x_2 + x_2^2 = (x_1 + x_2)^2$

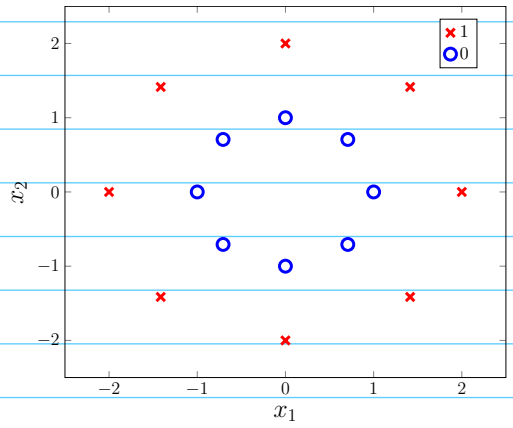
$f_2^2 = \sigma$ $\quad f_1(x) = x^2$

$w^2 = \theta$ $\quad w_1^1 = 1$ $\quad w_0^1 = 0$

$w_0^2 = \theta_0$ $\quad w_2^1 = 1$

coeff. for 1st feature in 1st layer

(g) Consider a new, related dataset shown below. Do the feature transformations you found previously linearly separate this dataset? If not, find a new feature transformation $\phi_3 : \mathbb{R}^2 \to \mathbb{R}$ such that the dataset is linearly separable using this feature.



NO! doesn't work.

$\phi_3 = |x_1| + |x_2|$, or $\sqrt{x_1^2 + x_2^2}$

NOTES: linear regression $X \to X_{aug}$ is also a feature transformation!

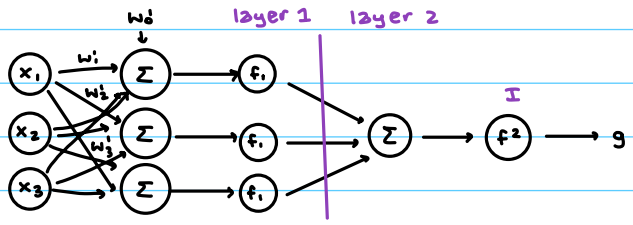
2. Suppose you are tasked with training a model to perform regression on a training data set $\mathcal{D}_{\text{train}} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, where $x^{(i)} \in \mathbb{R}^3, y^{(i)} \in \mathbb{R}$ for all $i = 1, \ldots, n$. As it turns out, linear hypotheses are not yielding sufficient results. Therefore, you would like to learn a feature transformation.

(a) Draw a computation graph which meets the following specifications:

- The first layer contains 3 neurons with ReLU activation functions.
- The output layer contains a single neuron with the identity activation function.

Recall that weights in neural networks are represented as $w^l_{n,m}$, corresponding to the weight in the l^{th} layer going from the n^{th} input neuron to the m^{th} output neuron. We represent the input to the l^{th} layer's activation function as z^l and the output of this activation function as a^l.

W Superscript (layer) dimension



(b) Write out the function that defines the hypothesis from the previous part. In particular,

- The pre-activation of the first layer, z^1, is a function of the input x and the weights W^1, W_0^1.
- The post-activation of the first layer, a^1, is a function of z^1.
- The pre-activation at the output, z^2, is a function of a^1 and the weights W^2, W_0^2.
- The output $g = a^2$ is a function of z^2.

$$W^1 = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & w^1_{1,3} \\ w^1_{2,1} & \cdot & \cdot \\ w^1_{3,1} & \cdot & \cdot \end{bmatrix} \qquad W_0^1 = \begin{bmatrix} w^1_{0,1} \\ w^1_{0,2} \\ w^1_{0,3} \end{bmatrix}$$

$$z^1 = (W^1)^T x + W_0^1$$

$$a^1 = \text{ReLU}(z^1)$$

$$z^2 = (W^2)^T a^1 + W_0^2$$

$$g = f^2(z^2)$$

(c) Suppose that all of the weights and biases are initialized to 1. Consider a new data point $(x, y) = ([1, 2, 3]^T, 10)$. For the forward pass, what is the "guess," $g = h([1, 2, 3]^T)$? How does the guess compare to the actual label?

$z^1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \end{bmatrix}$

$a^1 = \text{ReLU}(\cdot) = \begin{bmatrix} 7 \\ 7 \end{bmatrix}$

$z^2 = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 7 \end{bmatrix} + 1 = 22$

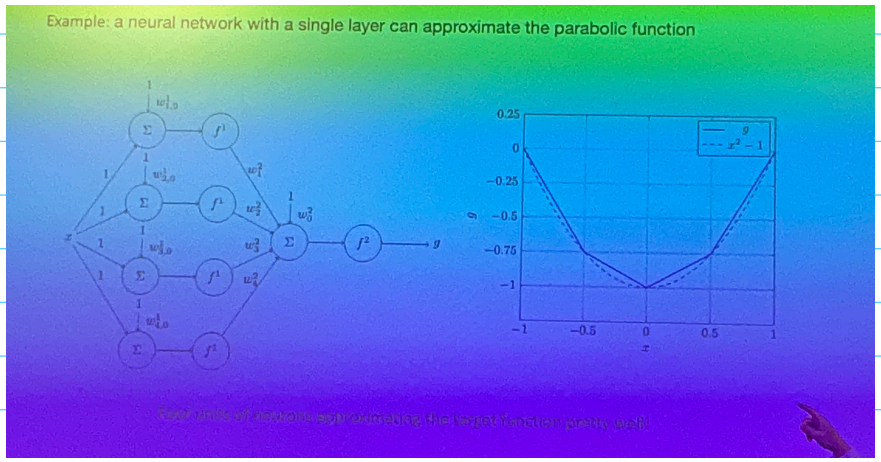
$g = 22$

Additional Resource: To better understand neural networks, feature transformations, and decision boundaries discussed in this recitation, we recommend exploring the TensorFlow Playground at `https://playground.tensorflow.org`. This interactive visualization tool allows you to:

- Experiment with different feature transformations (similar to ϕ_1 and ϕ_2 from Problem 1)
- Visualize how neural networks with various architectures learn to separate non-linearly separable data
- Observe how different activation functions (ReLU, sigmoid, etc.) affect the decision boundaries

3/19 NN to CNN

- layered comp. graph
- forward pass: inputs → learned features
- backprop: gradient info backwards



CNN:

- spatial locality: focus on neighboring pixels
- translational equivariance: w/ pooling, can still extract useful info when an object shifts
- visual hierarchy: layers of conv. filters enables extracting simple & complex patterns

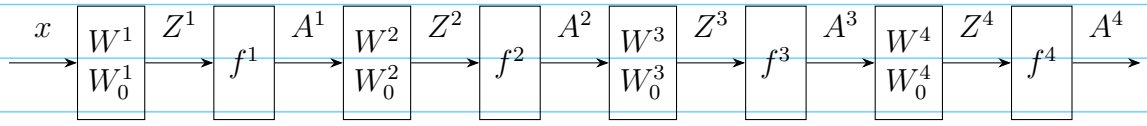
6.390 Introduction to Machine Learning

Recitation Topic: Neural Networks and Convolutional Neural Networks

1. Consider a dataset $\mathcal{D}_n = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ with (input) feature vectors $x^{(i)} \in \mathbb{R}^d$ and (output) labels $y^{(i)} \in \mathbb{R}^m$ for all $i = 1, \cdots, n$.

   We construct a neural network. All of the activation functions (f^1, f^2, f^3, f^4) are ReLUs (applied element-wise).

   does NOT change dim.



   The computation is as follows:

   Four-Layer Neural Network. For each layer $l \in \{1, 2, 3, 4\}$, we follow the regular neural-network recipe. We compute the pre-activation (Z^l) and activation (A^l) as follows:

$$Z^l = W^{l\top} A^{l-1} + W_0^l, \quad A^l = f^l(Z^l),$$

   where W^l is the weight matrix and W_0^l is the bias vector, and $A^0 = x$ (the input).

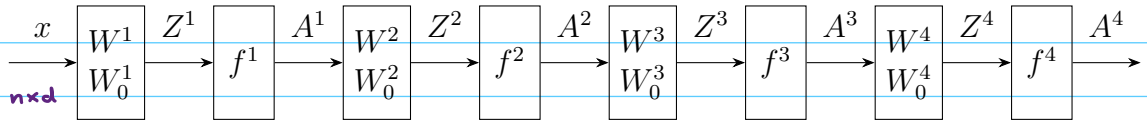
   We can then define the output of the neural network as $G = \text{NN}(x; \mathcal{W}) = A^4$, for the set of parameters, $\mathcal{W} = \{W^1, W_0^1, W^2, W_0^2, W^3, W_0^3, W^4, W_0^4.\}$.

   The squared distance between the network prediction G and the true labels $y^{(i)}$ with respect to each of the n datapoints is then used as the loss function to train our network:

$$\mathcal{L}(G^{(i)}, y^{(i)}) = \frac{1}{2}\|G^{(i)} - y^{(i)}\|^2.$$

   All parts of this problem can be solved independently; the answers to the different parts do not depend on one another. Also, for all parts, please choose **all options that apply.**

(Replicating the neural network diagram again here for reference.)

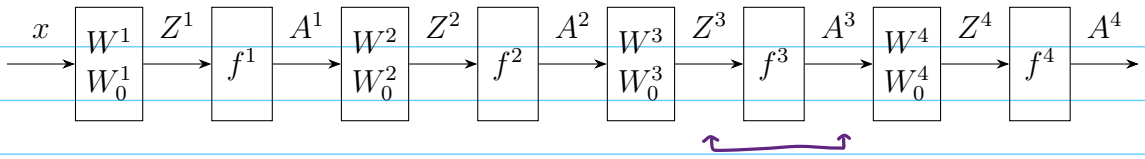


(a) Given the specifications of the neural net, which option(s) below must be true? (Choose all that apply.)

- ◯ shape of W^1 = shape of W^2
- ◯ shape of W^2 = shape of W^3
- ✓ shape of Z^1 = shape of A^1
- ◯ shape of Z^3 = shape of x
- ✓ shape of Z^3 = shape of A^3
- ◯ shape of Z^1 = shape of A^3
- ✓ shape of $G^{(i)}$ = shape of $y^{(i)}, \forall i = 1, 2, \ldots, n$

- # hidden units can vary
- ReLU functions applied elt-wise (doesn't change output size)
- to calculate loss func, need to compute $G^i - y^i$ so they must be same dim!

(Replicating the neural network diagram again here for reference.)



(b) What is true about $\frac{\partial A^3}{\partial Z^3}$? same dim.

- [x] Must be a square matrix. · A^i & z^i must have same dims.
- [x] Must be a diagonal matrix. · $\frac{\partial A}{\partial z}$ should be SQUARE
- [x] May be an identity matrix. · off-diag. is always 0 b/c taking same elt.
- [x] May be an all-zero matrix. · $\frac{d}{dz}\text{ReLU}(z) = \begin{cases} 1 & z>0 \\ 0 & z\le 0 \end{cases}$ · diag. always 1 or 0
- [ ] May be a matrix with all negative entries. (no b/c relu

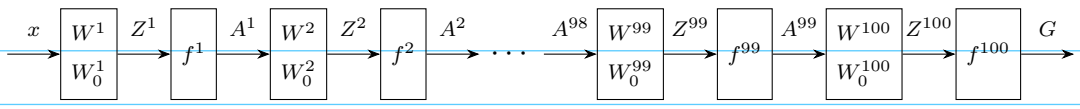
$\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$ $\begin{bmatrix} \text{ReLU}(z_1) \\ \vdots \end{bmatrix}$

(c) Suppose that during backpropagation, $\frac{\partial A^2}{\partial Z^2}$ is all zeros.
Which of the following *must* also be zero? Choose all that apply.

- [x] A^2 → ReLU(z) = 0 → $A^2 = 0$ b/c grad = 0
- [ ] Z^2 → $z^2 = (W^2)^T A^1 + W_0^2$, just l.n. trans.
- [ ] $\frac{\partial Z^2}{\partial A^1}$
- [x] $\frac{\partial A^2}{\partial A^1}$ → zero by chain rule
  e.g. $\frac{\partial A_2}{\partial A_1} = \frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial A_1}$
  # b/c z_2 has no affect on A_2, A_1 will have no affect on z_2
- [x] $\frac{\partial A^4}{\partial A^1}$

2. We now make the network deeper by adding 96 more layers, resulting in a 100-layer neural network. ReLU activation functions are used for all hidden layers except for the output layer, which uses the identity activation $f(z) = z$. We use a squared-error loss function for training the neural network. All of the weights and biases are initialized as $W^l = 0.1 \cdot I$ and $W_0^l = 0.5 \cdot \mathbf{1}$ for every layer $l = 1, \ldots, 100$, where I is an appropriately sized identity matrix and $\mathbf{1}$ is a vector of ones.

(a) Consider computing gradients in this 100-layer network.



i. First, focus only on the last two layers of the network. Write a chain-rule expression for $\frac{\partial \mathcal{L}}{\partial W^{99}}$.

$\frac{\partial z^{99}}{\partial W^{99}} \cdot \frac{\partial A^{99}}{\partial z^{99}} \cdot \frac{\partial z^{100}}{\partial A^{99}} \cdot \frac{\partial G}{\partial z^{100}} \cdot \frac{\partial L}{\partial G}$

You do not need to simplify fully; focus on the structure of the chain rule.

$$\frac{\partial L}{\partial W^{99}} = \frac{\partial z^{99}}{\partial W^{99}} \cdot \frac{\partial A^{99}}{\partial z^{99}} \cdot \frac{\partial z^{100}}{\partial A^{99}} \cdot \frac{\partial G}{\partial z^{100}} \cdot \frac{\partial L}{\partial G}$$

ii. Now move back one more layer and write a chain-rule expression for

$$\frac{\partial \mathcal{L}}{\partial W^{98}}.$$

What extra factor(s) appear compared to the previous part?

$$\frac{\partial L}{\partial W^{98}} = \frac{\partial z^{98}}{\partial W^{98}} \cdot \frac{\partial A^{98}}{\partial z^{98}} \cdot \frac{\partial z^{99}}{\partial A^{98}} \cdot \frac{\partial A^{99}}{\partial z^{99}} \cdot \frac{\partial z^{100}}{\partial A^{99}} \cdot \frac{\partial G}{\partial z^{100}} \cdot \frac{\partial L}{\partial G}$$

$\partial L / \partial W^{99}$

backprop. to earlier layers can be efficient b/c reuses large portion of same downstream

iii. Based on the pattern above, what do you expect about the magnitude of

$$\frac{\partial \mathcal{L}}{\partial W^{1}}?$$

What repeated term causes this behavior? What phenomenon does this illustrate?

$$\frac{\partial L}{\partial W^{1}} = \frac{\partial z^{1}}{\partial W^{1}} \frac{\partial A^{1}}{\partial z^{1}} \frac{\partial z^{2}}{\partial A^{1}} \cdots \frac{\partial A^{99}}{\partial z^{99}} \frac{\partial z^{100}}{\partial A^{99}} \frac{\partial z^{100}}{\partial A^{99}} \frac{\partial G}{\partial z^{100}} \frac{\partial L}{\partial G}$$

$$= \frac{\partial z^{l-1}}{\partial A^{l}} \sim W^{l} = 0.1I$$

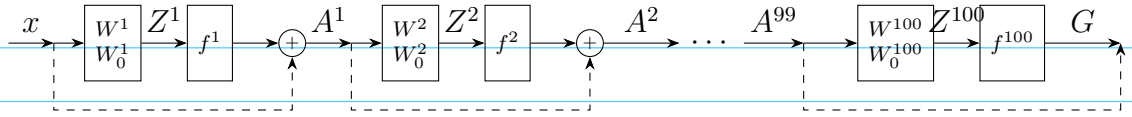
as we backprop. to earlier layers, multiply by factor of 0.1

then, 1st layer update $(0.1)^{99} \rightarrow$ grad. almost 0

(b) Jim decides to modify his neural network by including "residual connections" (also called skip connections). The computation now becomes:

$$A^0 = x, \quad Z^l = W^l A^{l-1} + W_0^l, \quad A^l = f^l(Z^l) \underbrace{+ A^{l-1}}_{\text{residual}}, \quad G = A^{100}$$

resulting in the following computation graph:



How might this improve the gradient computation for updating W^1, compared to the previous part?

Hint: Think about how $\frac{\partial A^l}{\partial A^{l-1}}$ changes when we replace

$$A^l = f^l(Z^l) \quad \text{with} \quad A^l = f^l(Z^l) + A^{l-1}.$$

$\frac{\partial A^l}{\partial A^{l-1}} = \frac{\partial f^l(z^l)}{\partial A^{l-1}}$ vs. $\frac{\partial (f^l(z^l) + A^{l-1})}{\partial A^{l-1}} = \frac{\partial f^l(z^l)}{\partial A^{l-1}} + I$

even if this term shrinks — this persists!

* residual connection adds identity term to Jacobian, enabling us to train deep networks

3. Shen wants to build a neural network to convert 2×2 RGB color images into greyscale. The color images have three channels: channel 0 is red, channel 1 is green, and channel 2 is blue. For each pixel, Shen uses the following `RGB2Grey` conversion:

$$\text{grey} = 0.30 \text{ red} + 0.59 \text{ green} + 0.11 \text{ blue}.$$

Each pixel value (in both greyscale images and color image channels) is a real number in the range $[0, 1]$.

(a) Consider the following pixel values for a 2×2 color image:

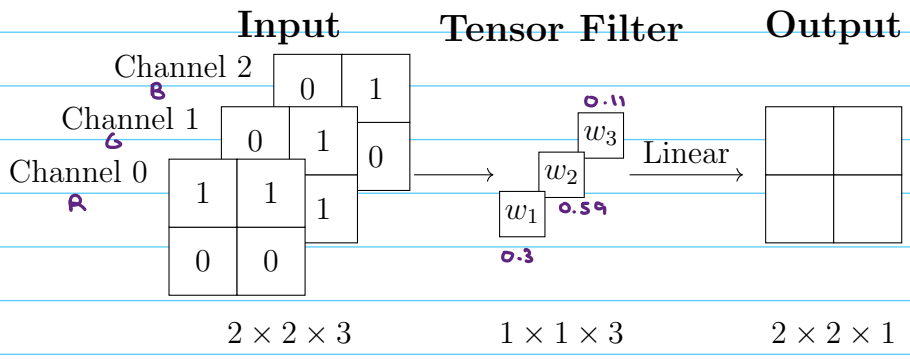
Channel 0 (red)		Channel 1 (green)		Channel 2 (blue)	
1	1	0	1	0	1
0	0	0	1	1	0

What are the corresponding pixel values in the greyscale image?

0.30	1
0.11	0.59

TL: 0.30
TR: 0.30 + 0.59 + 0.11
BL: 0.11
BR: 0.59

(b) Anh is charged with building a convolutional neural network (CNN) to implement the `RGB2Grey` conversion. Anh heard a rumor that 1×1 convolutional tensor filters are useful in CNNs. They try the following architecture, where the input RGB color images have shape $2 \times 2 \times 3$, the tensor filter has shape $1 \times 1 \times 3$ with bias term $w_0 = 0$ and stride 1, a Linear activation function is used, and the output greyscale image has shape $2 \times 2 \times 1$:



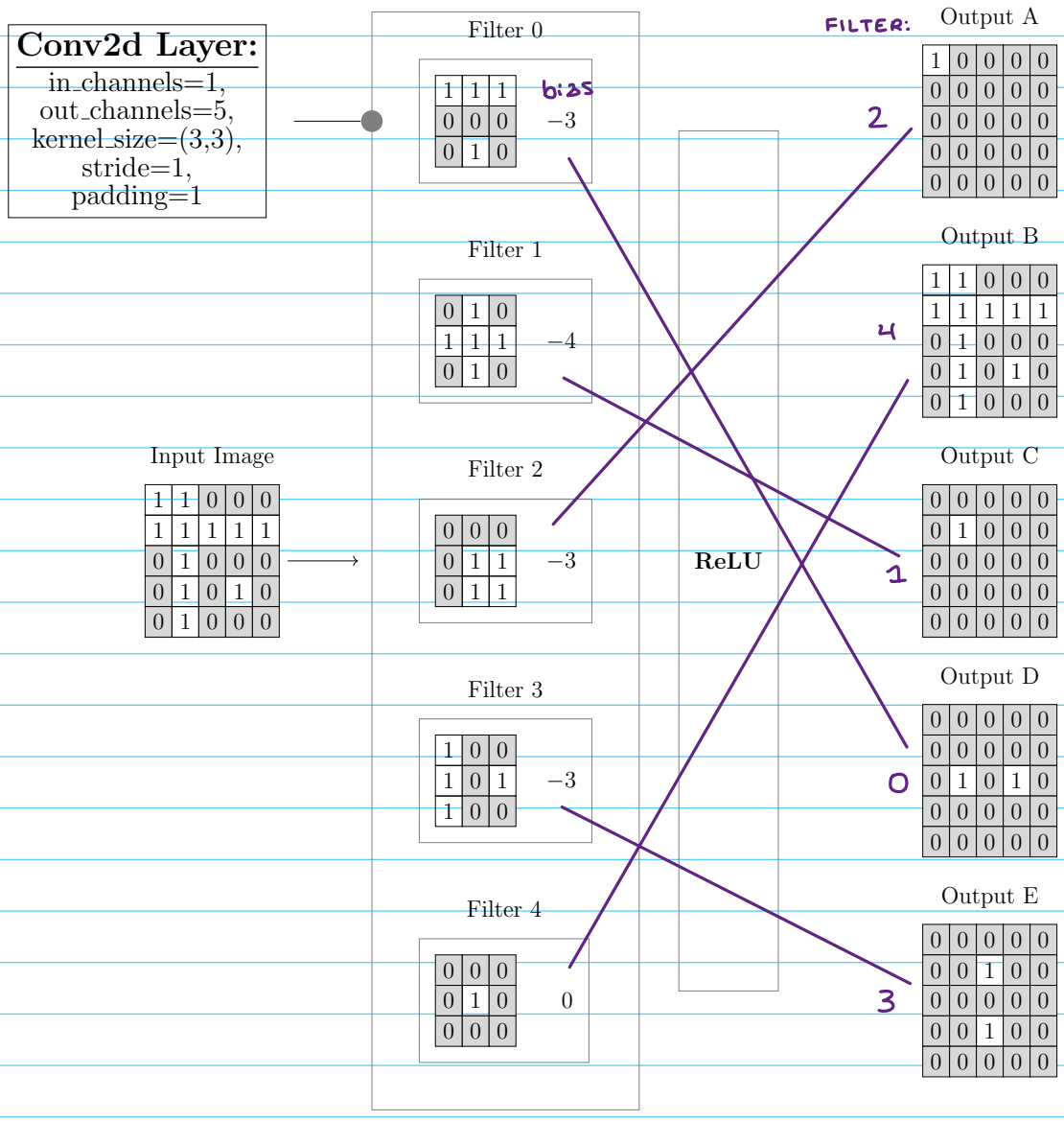
With hundreds of different training samples, can this CNN learn the greyscale conversion? If so, provide a set of values for the weight parameters w_1, w_2, and w_3 that implements the converter. If not, explain why the converter cannot be learned with this CNN.

$w_1 = 0.3$ $w_2 = 0.59$ $w_3 = 0.11$

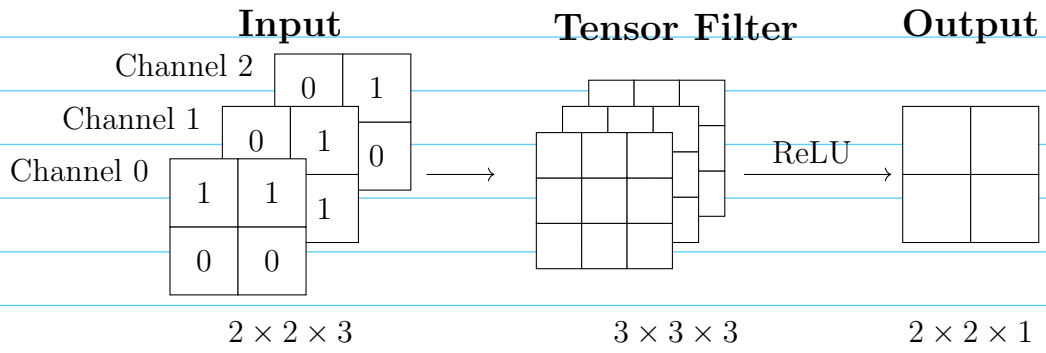
this conv. filter (1x1x3) mixes channels @ each pixel!

(c) Elisa thinks it might be better to replace the 1×1 convolutional filter with a bigger filters, and replaces the Linear activation with a ReLu to have more capacity to deal with a large variety of images. Shen first asks Elisa to work on the filter matching problem below to see if she understands how the convolution works.

Given a convolutional layer with a filter bank containing five different filters, each with a filter size of 3×3, an offset and a stride of 1, the output of the convolutional layer feeds into a ReLU. Match each output channel to its corresponding filter.



(d) Now Elisa uses a $3 \times 3 \times 3$ tensor filter with offset term $w_0 = 0$, and a ReLU:

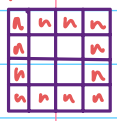


Assume the filter is applied with stride 1.

(i) In order to obtain a $2 \times 2 \times 1$ output, what is the size of zero padding needed?

★ (ii) With appropriate padding applied, how many output pixels are influenced by a single input pixel at position $(0, 0)$ of channel 0? List the coordinates of all affected output pixels.

PADDING:



i) 1 padding

ii) (0,0) pixel affects everywhere!

(e) With hundreds of different training samples, can Elisa's CNN from part (d) learn the greyscale conversion, assuming that zero-padding has been applied as necessary? If so, fill in a set of weights for the $3 \times 3 \times 3$ tensor filter above. If not, explain why the grey-scale converter cannot be learned with this CNN.

Yes, or No with explanation:

can train CNN w/ GD and sq. loss error when learned filter looks like ↘

If yes, fill in the tensor filter weights:

Channel 0		
0	0	0
0	0.30	0
0	0	0

Channel 1		
0	0	0
0	0.59	0
0	0	0

Channel 2		
0	0	0
0	0.11	0
0	0	0

4/2 TRANSFER LEARNING:

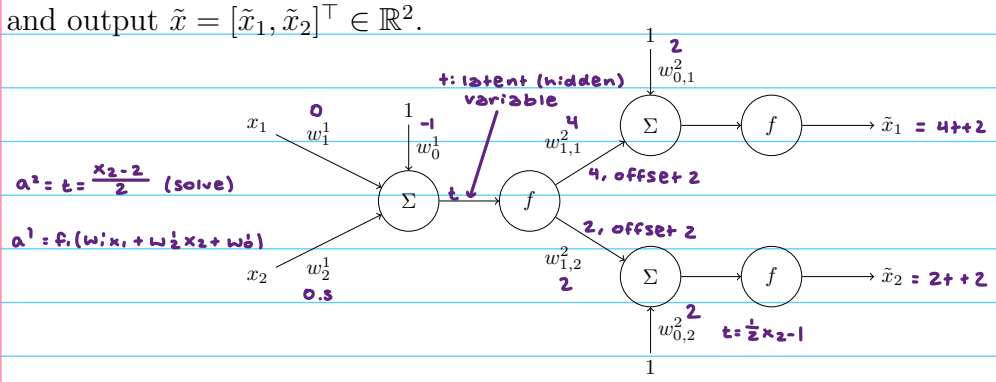
- pre-train on large data & get representations (compression, abstraction, conceptualization)
- fine-tune on small data

today:

1) mini autoencoder
2) arithmetic of embedding vectors
3) autoregressive model

6.390 Introduction to Machine Learning
Recitation Topic: Representation Learning

1. Otto N. Coder wants to find a way to represent his 2-dimensional data points in 1-dimensional space. Consider the following *autoencoder* with input $x = [x_1, x_2]^\top \in \mathbb{R}^2$ and output $\tilde{x} = [\tilde{x}_1, \tilde{x}_2]^\top \in \mathbb{R}^2$.



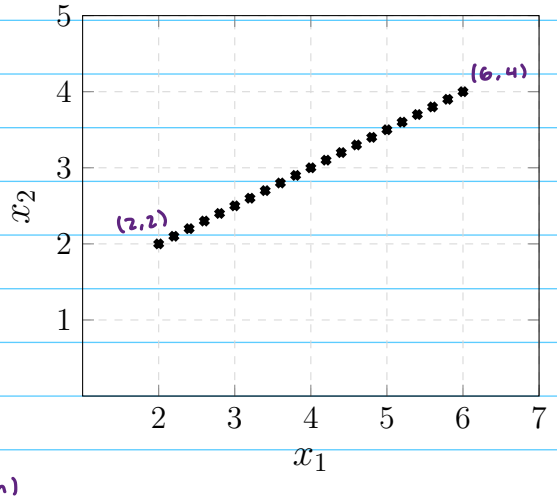
The encoder of this autoencoder is described by the output of the first hidden layer,

$$a^1 = f_1(w_1^1 x_1 + w_2^1 x_2 + w_0^1),$$

and the decoder of this autoencoder is described by the outputs of the output layer,

$$\tilde{x}_1 = f_2(w_{1,1}^2 a^1 + w_{0,1}^2),\ \tilde{x}_2 = f_2(w_{1,2}^2 a^1 + w_{0,2}^2)$$

(a) For this part, suppose that the activation functions f_1, f_2 are the identity $f(z) = z$. Let our dataset $\mathcal{D}_{\text{train}} = \{x^{(i)}\}_{i=1}^n$ be composed of n datapoints, plotted below:



i. Let t be a latent (hidden) variable in the range $[0, 1]$. Based on the plot, find functions $x_1(t), x_2(t)$ that map t to x_1 and x_2, respectively.

$t \to x_1 : [0,1] \to [2,6]$
$t \to x_2 : [0,1] \to [2,4]$

$(6,4) - (2,2) = (4,2)$
$(x_1(t), x_2(t)) = (4,2)(t_1, t_2) + (2,2) \to$
$x_1(t) = 4t + 2$
$x_2(t) = 2t + 2$
$x_2 = \frac{1}{2}x_1 + 1$
$t=0$: $x_1 = 2$, $x_2 = 2$
$t=1$: $x_1 = 2$, $x_2 = 4$
(all in range)

ii. Consider the autoencoder structure depicted in the figure at the start of this question. Using your result from Part (a).i., can you find weights and offsets for this autoencoder that just about perfectly recover $\tilde{x}_1 = x_1$ and $\tilde{x}_2 = x_2$, even though there is only one hidden unit? Note: the solution is not unique.

$a_2' = t = \frac{x_2 - 2}{2}$ (solve)
↕ set equal
$a' = f_1(w_1' x_1 + w_2' x_2 + w_0')$
f_1 = identity

$\frac{x_2-2}{2} = w_1' x_1 + w_2' x_2 + w_0'$
$\frac{1}{2}x_2 - 1 = w_1' x_1 + \frac{1}{2} x_2 - 1$

$x_2 = \frac{1}{2}x_1 - 1 \rightarrow x_1 = 2x_2 + 2$

$w_0^1 =$ -1 $w_1^1 =$ 0 $w_2^1 =$ 0.5

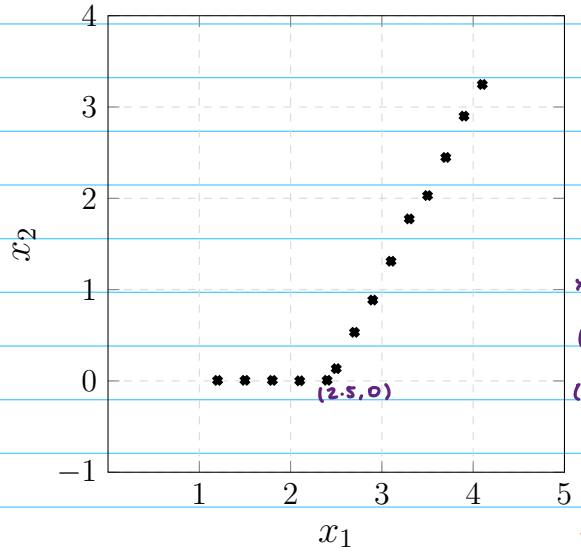
$w_{0,1}^2 =$ 2 $w_{1,1}^2 =$ 4

$w_{0,2}^2 =$ 2 $w_{1,2}^2 =$ 2

iii. Intuitively, what would the weights and offsets of the autoencoder represent?

though the line exists in 2D space, it rly is single dimensional entity that can be described by single param t, and autoencoder captures this.

(b) Otto modifies his autoencoder to have ReLU activation functions so that he can model non-linear relationships. Suppose that he has a new data set $\mathcal{D}_{\text{train}} = \{x^{(i)}\}_{i=1}^n$ composed of n datapoints, plotted in this figure:



$x_1 = 0$ if < 2.5
(2.5, 0) → (4, 3)
$(x_1(t), x_2(t)) =$
$x_2 = 2x_1 - 5$ $(x_1 > 2.5)$
$x_2 = 0$ $(x_1 < 2.5)$
$x_2 = \text{RELU}(2x_1 - 5)$

Again consider the autoencoder structure in the figure at the start of this question. Find values for the weights and offsets in the autoencoder that can just about perfectly recover this new data set.

$w_0^1 =$ 0 $w_1^1 =$ 1 $w_2^1 =$ 0

(identity map)
$w_{0,1}^2 =$ 0 $w_{1,1}^2 =$ 1 (all x_1 is +, so RELU +)

$w_{0,2}^2 =$ -5 $w_{1,2}^2 =$ 2 $(2x_1 - 5)$

suppose a = latent var @ bottleneck

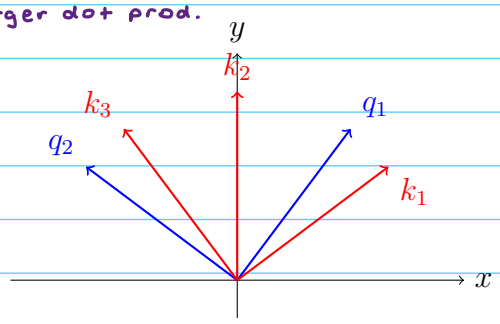
natural to put $a = x_1$ & encoder out just x_1

2. You are given a set of queries and a set of keys, all in $\mathbb{R}^2$. In this context, a query is a vector that represents what we're comparing against other vectors and a key is a vector that represents one of the possible matches we are comparing to the query.

Let: *more similar direction to query = larger dot prod.

$$q_1 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}, \quad q_2 = \begin{bmatrix} -4 \\ 3 \end{bmatrix},$$

$$k_1 = \begin{bmatrix} 4 \\ 3 \end{bmatrix}, \quad k_2 = \begin{bmatrix} 0 \\ 5 \end{bmatrix}, \quad k_3 = \begin{bmatrix} -3 \\ 4 \end{bmatrix}.$$



(a) Let $s_{i,j} = q_i^T k_j$ denote the dot product (recall that the dot product can be used as a measure of similarity) between a query q_i and key k_j. Compute the dot product between each query and each key.

Record your answers in the table below. Additionally, for each query, identify which key is the most similar. How does this correspond to how visually aligned each query and key are in the plot?

	k_1	k_2	k_3
q_1	$s_{1,1} =$ 24	$s_{1,2} =$ 20	$s_{1,3} =$ 7
q_2	$s_{2,1} =$ -7	$s_{2,2} =$ 15	$s_{2,3} =$ 24

(b) Let n_q be the number of queries and n_k be the number of keys ($n_q = 2$ and $n_k = 3$ in our example). Construct matrices $Q \in \mathbb{R}^{n_q \times 2}$ and $K \in \mathbb{R}^{n_k \times 2}$ such that taking the matrix product QK^T will result in a matrix whose elements are identical to the table above before normalization.

Final dim: 2×3 $\quad$ Q $\quad$ K^T
2×2 $\quad$ 2×3 → K = 3×2

$Q = \begin{bmatrix} 3 & 4 \\ -4 & 3 \end{bmatrix}$ ← put as rows

$K = \begin{bmatrix} 4 & 3 \\ 0 & 5 \\ -3 & 4 \end{bmatrix}$ $\quad$ $K^T = \begin{bmatrix} 4 & 0 & -3 \\ 3 & 5 & 4 \end{bmatrix}$

(c) Instead of thinking about abstract vectors, let's think about how this could relate to natural language processing. Assume that we have a model that can represent both words and sentences as vectors (a.k.a. as *word embeddings* and *sentence embeddings*, respectively). An example of one such model is linked here: `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`. We would like to fill in the blank:

After the rain, the grass was _____.

Propose an algorithm that predicts the next word by using the embedding model. Your next-word-prediction algorithm should use the embedding model to produce "query" and "key" vectors and then use those queries and keys to predict the next word. Clearly explain what serves as the "queries" the "keys" in your approach.

Hint for one algorithm: Think about what is the context. Then, imagine you have a list of candidate words that might fill in the blank. How could you compare each candidate's embedding with the context embedding to decide which word best fits the sentence?

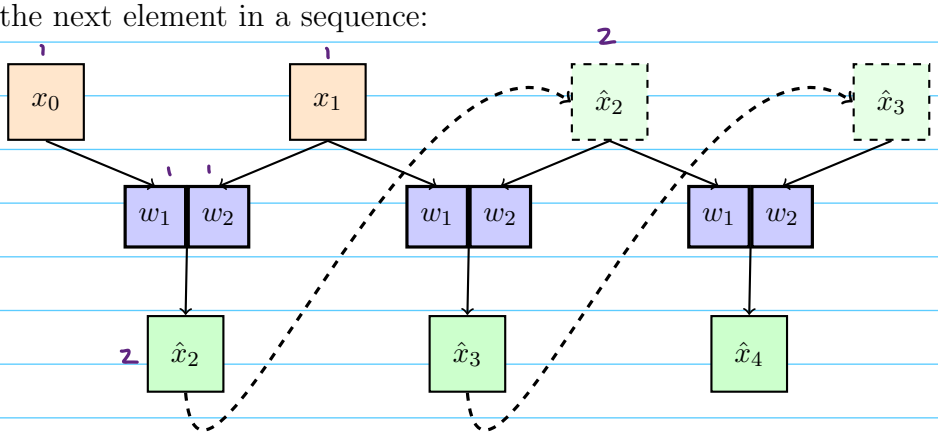
Note: this is an open-ended question to spark discussion. We are not asking you to invent an optimal method.

embedding of current sentence = query

embedding of all candidate words = keys

3. Consider the following *autoregressive* model, which uses a 2×1 convolutional filter to compute the next element in a sequence:

feeds own input back in



A model is autoregressive if it makes each prediction only on past values. In particular, it feeds its own outputs back into the input.

(a) Let's say the model is given $x_0 = 1, x_1 = 1, w_1 = 1, w_2 = 1$. Predict the next 3 numbers of the sequence, $\hat{x}_2, \hat{x}_3, \hat{x}_4$.

$\hat{x}_2 = 1+1 = 2$

$\hat{x}_3 = 1+2 = 3$

$\hat{x}_4 = 2+3 = 5$

(b) Suppose we want the sequence to remain constant at 1, i.e. $x_t = 1$ for all t. What weights w_1, w_2 would achieve this?

$w_1 = w_2 = 0.5$

$w_1 + w_2 = 1$

(c) Suppose you are given a data set of n real-valued sequences, $\{x_t^{(i)}\}_{i=1}^n$, where sequence i is indexed from time $t = 0$ to $t = T$. Using the autoregressive model shown above, we want to learn weights w_1 and w_2 that make good predictions throughout each sequence.

During training, for each valid time step t, we use the previous two true values to predict the next value:

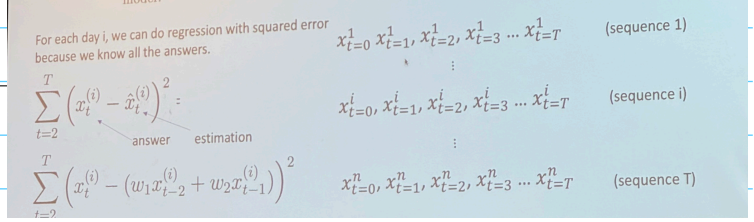
$$\hat{x}_t^{(i)} = w_1 x_{t-2}^{(i)} + w_2 x_{t-1}^{(i)}, \qquad t = 2, \ldots, T.$$

What objective function could you minimize to learn a good next value prediction model?

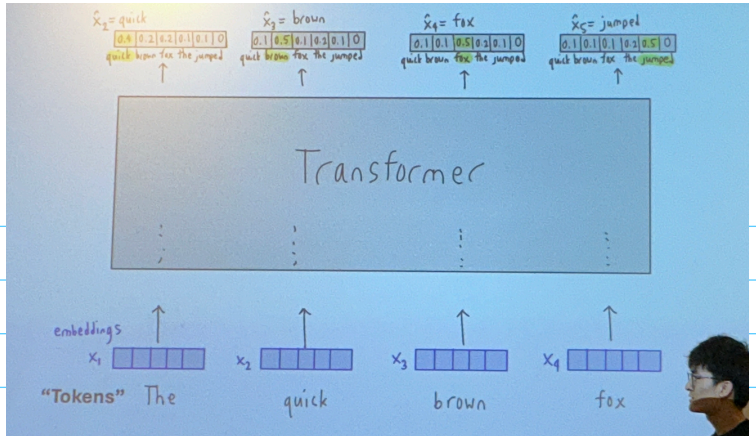
for each day i, we can do regression w/ squared error b/c we know all the answers

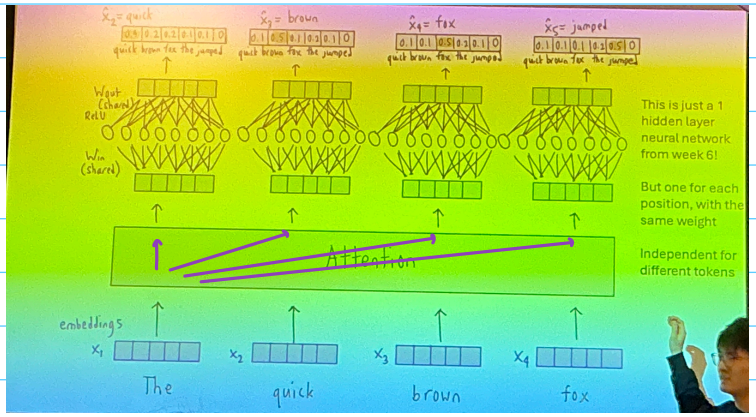
$$\sum_{t=2}^{T} \left(x_t^{(i)} - \hat{x}_t^{(i)}\right)^2 = \sum_{t=2}^{T} \left(x_t^{(i)} - \left(w_1 x_{t-2}^{(i)} + w_2 x_{t-1}^{(i)}\right)\right)^2$$

answ. estim.



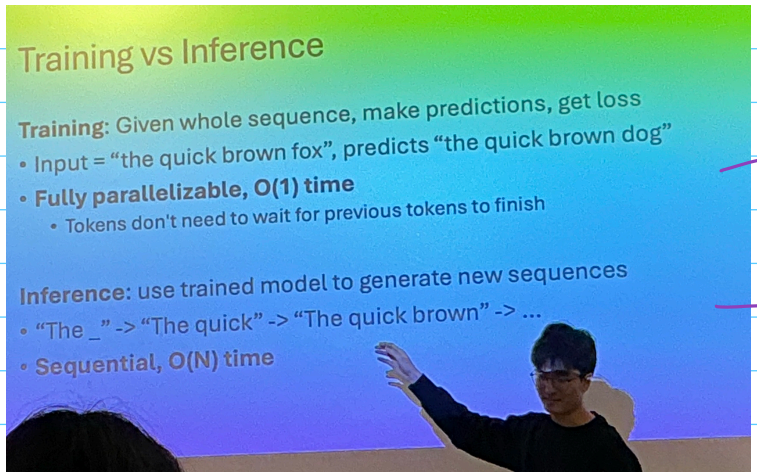
4/9





each token looks @ other tokens

attention acts on every token simultaneously



→ compute MLP & outputs in para

→ 1 by 1

transformers = scalable

what should this elt. contribute to attention

what queries should be looking for in this elt. ←

"lucy" key value | "ate" key value | "her" key value | "breakfast" key value

"Lucy"

x_1

Query: what lucy is looking for

$q_{lucy} \cdot k_{lucy}$

$3 \to \frac{e^3}{e^4+e^1+e^3+e^2}$

$1 \to \frac{e^1}{11}$ $3 \to \frac{e^3}{11}$

$[-\infty, \infty] \to [0, 1]$

$q_i \in \mathbb{R}^{d_k \times 1}$

$0.45 \cdot v_{lucy} + 0.08\, v_{ate} + 0.45\, v_{her} + 0.02\, v_{breakfast} \to z_{lucy}$

$A = \begin{pmatrix} | & | & & | \\ a_{lucy} & a_{ate} & \cdots & \\ | & | & & | \end{pmatrix} \begin{pmatrix} - v_{lucy}^T - \\ - v_{ate}^T - \\ \vdots \end{pmatrix} \begin{pmatrix} | & & | \\ q_{lucy} & \cdots & \\ | & & | \end{pmatrix}$ (d_k)

$d_k < d$

6.390 Introduction to Machine Learning

Recitation Topic: Transformers

1. Consider a transformer model with a single attention head and no positional encoding. We input ["Lucy", "ate", "her", "breakfast"] and observe:

$$\frac{QK^T}{\sqrt{d_k}} = \begin{array}{c} \\ L \\ a \\ h \\ b \end{array} \begin{array}{c} \begin{array}{cccc} \text{Lucy} & \text{ate} & \text{her} & \text{breakfast} \end{array} \\ \begin{bmatrix} 3 & 1 & 3 & 0 \\ 0 & 2 & -3 & 1 \\ -1 & 1 & 2 & -2 \\ 0 & 2 & -2 & -1 \end{bmatrix} \end{array}$$

For all subparts below, we keep all learnable weights in this transformer fixed.

(a) First, we consider the attention matrix A and denote the attention "Lucy" pays to "ate" be a_{la} and the attention "Lucy" pays to "her" as a_{lh}. Which one is larger, a_{la} or a_{lh}? Justify your answer.

a_{la} = attention Lucy pays to ate

a_{lh} = attention Lucy pays to her (same denom. b/c same row!) softmax: $\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

$a_{lh} > a_{la}$ b/c softmax(3) > softmax(1)

(b) We consider the attention matrix A and denote the attention "Lucy" pays to "ate" be a_{la} and the attention "ate" pays to "Lucy" as a_{al}. Which one is larger, a_{la} or a_{al}? Justify your answer.

$$\frac{e^0}{e^3+e^1+e^3+e^0} \approx 0.0228 \quad \frac{e^1}{e^3+e^1+e^3+e^0} \approx 0.0619$$

$$\frac{e^0}{e^0+e^2+e^{-3}+e^1} \approx 0.0896 \quad \frac{e^1}{e^0+e^2+e^{-3}+e^1} \approx 0.2436$$

a_{la} = softmax(1) ≈ 0.0619

a_{al} = softmax(0) ≈ 0.0896 ← bigger

(c) In this subpart, we permute the input sequence as ["her", "breakfast", "ate", "Lucy"]. Which one is larger, the attention "Lucy" pays to "ate" a'_{la} or the attention "ate" pays to "Lucy" a'_{al}? (Recall that all learned weights remain fixed and there is no positional encoding.)

SAME:: same dot products just diff. order

$a'_{la} < a'_{al}$

2. In the standard attention mechanism, any token can attend to any other token. Suppose we want to adapt the mechanism to force this behavior: each token should only attend to **earlier** tokens in the sequence (including itself), never to future ones. This can be done via the idea of causal attention. Let's explore this idea.

   (a) First, let's understand a key property when computing the softmax of a vector, numerically. Recall that $\lim_{x\to\infty} e^{-x} = 0$. What is the result of computing $\text{softmax}([-\infty, -\infty, 6390])$?

   $[0, 0, e^{6390}] \rightarrow [0, 0, \frac{e^{6390}}{0+0+e^{6390}}] = [0, 0, 1]$

   (b) Jordan decides to modify their computation of the full attention matrix as:

$$A(X) = \text{softmax}_{\text{row}}\left(\frac{Q(X)K(X)^T}{\sqrt{d_k}} + M\right),$$

   where $M \in \mathbb{R}^{n\times n}$ is a mask matrix used to set elements equal to $-\infty$ prior to applying the softmax operation.

   Construct the matrix M that enforces causal masking—that is, so that each token x_i only attends to positions $j \leq i$.

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

   in general: $M_{ij} = \begin{cases} 0 & \text{if } j \leq i \\ -\infty & \text{if } j > i \end{cases}$

   (c) Now suppose Jordan replaces the mask matrix above with the following mask matrix $M1$, and assume $n = 4$:

$$M1_{ij} = \begin{cases} 0 & \text{if } j = i \text{ or } j = (n+1-i) \\ -\infty & \text{otherwise} \end{cases}$$

   Assume that our attention head has weights $W_q = W_k = W_v = I$ (the identity matrix).

   What is the value of a_{12}, i.e., how much attention does token 1 pay to token 2?

   - (•) 0
   - ( ) 0.5
   - ( ) 1
   - ( ) Depends on x_1 and x_2

$$M = \begin{bmatrix} 0 & -\infty & -\infty & 0 \\ -\infty & 0 & 0 & -\infty \\ -\infty & 0 & 0 & -\infty \\ 0 & -\infty & -\infty & 0 \end{bmatrix}$$

3. Carla is experimenting with self-attention using very simple "Fibonacci-like" sequences with $d = 1$. In particular, for a sequence of three tokens $x_1 = a, x_2 = b, x_3 = c$, the third digit is $c = \frac{a+b}{2}$. To better understand how softmax'd attention scores influence outputs, she writes a simplified version of a attention head—one that uses a constant attention matrix (instead of computing attention from queries and keys).

Here is her simplified implementation:

```
1)    def fib_attention_const(X):
2)      A = torch.tensor( ALPHA_MAT )
3)      A = torch.softmax(A, dim=-1)
4)      Z = A @ X
5)      return Z
```

Carla wants to choose an `ALPHA_MAT` so that for input:

```
X = torch.tensor([[a, b]]).T   # shape:  (2, 1)
```

the output of `fib_attention(X)` becomes:

```
Z = torch.tensor([[b], [(a + b) / 2]]).
```

What should Carla set `ALPHA_MAT` to in line 2 so that the attention mechanism produces the desired output `Z` shown above?

Hint: Think about what kind of weighted combinations of `a, b` would produce `Z`, and how that corresponds to rows of the attention matrix.

Hint: recall that $d = 1$.

Problem 3

	Token 1	Token 2
Inputs:	a	b
Outputs:	b	(a+b)/2
	$0 \cdot a + 1 \cdot b$	$\frac{1}{2} \cdot a + \frac{1}{2} \cdot b$

Want post softmax attention to be $\begin{bmatrix} 0 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$ so that $\text{output} = \begin{bmatrix} 0 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$

Pre softmax ALPHA_MAT = $\begin{bmatrix} -\infty & \text{anything} \\ \text{whatever} & \text{same as left} \end{bmatrix}$

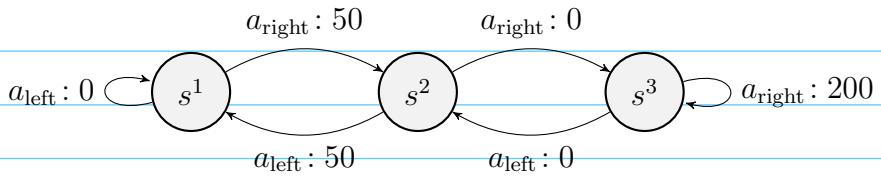
$\begin{bmatrix} -\infty & 1 \\ 1 & 1 \end{bmatrix}$ works, or $\begin{bmatrix} -\infty & 1 \\ 0 & 0 \end{bmatrix}$

6.390 Introduction to Machine Learning
Recitation Topic: Markov Decision Processes

1. Wobot is learning how to navigate a 1-D grid world with three states, s^1, s^2, s^3.

   At each time step, Wobot takes an action $a \in \{a_{\text{left}}, a_{\text{right}}\}$. The reward for taking each action from each state is shown in the figure below.

   In this problem, assume the discount factor $\gamma = 1$.



   (a) i. A *policy* for a *finite* horizon h is a family $\{\pi_1, \pi_2, \ldots, \pi_h\}$ of functions $\pi_k : \mathcal{S} \mapsto \mathcal{A}$ that specifies what action to take in each state, assuming there are exactly k steps left. Describe the optimal policy π_k^* for the MDP in the diagram above. That is, for a fixed horizon h and state s, show how $\pi_k^*(s)$ depends on k.

CASE 1: k=1 for horizons h ≥ k: CASE 2: k ≥ 2: take a_right
s¹: optimal take a_right
s²: take a_left
s³: a_right

   ii. For each combination of initial state and horizon h, determine the horizon-h *value* obtained by the optimal policy, $V_h^{\pi^*}(s)$.

	$h=0$	$h=1$	$h=2$	$h \geq 3$
s^1	0	50	100	50+200(h-2)
s^2	0	50	200	200(h-1)
s^3	0	200	400	200h

(b) For this part, assume that the MDP has an *infinite horizon*.

i. Suppose that the discount factor $\gamma = 0.8$. Describe in words what actions the optimal policy π^* will take in each of the three states.

regardless of init. state, take action a right b/c transitioning to s3 is the best: $50 + 0 \cdot r + 200r^2 = 178 > 122 = 50 + 50r + 50r^2$

ii. For each state $s \in \{s^1, s^2, s^3\}$, for the optimal policy π^* discovered in part (b) i. write the value for $V_\infty^{\pi^*}(s)$ with discount factor $\gamma = 0.8$. Recall the expanded form of the geometric series:

$$\sum_{k=0}^{\infty} a\rho^k = \frac{a}{1-\rho}, \quad \text{for } |\rho| < 1.$$

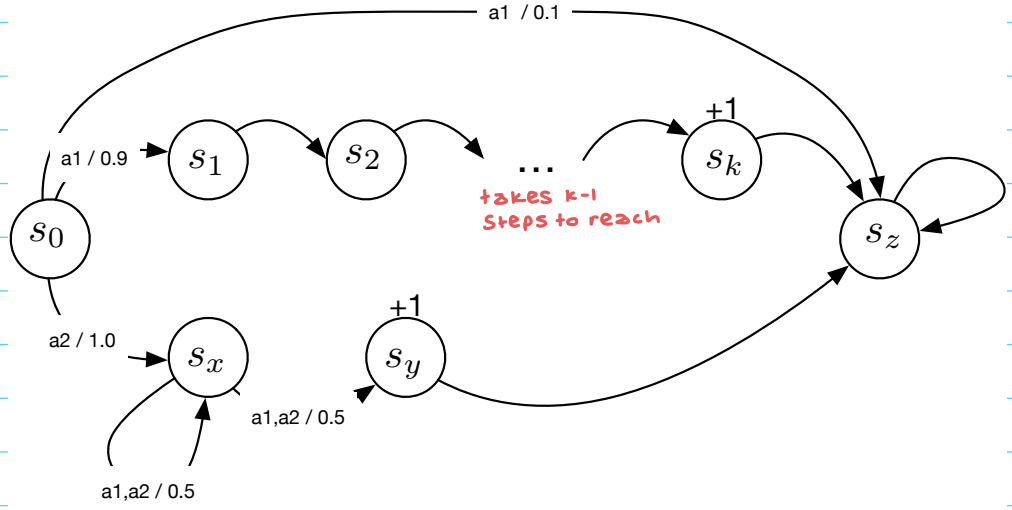
$V_\infty^{\pi^*}(s^3) = \sum_{t=0}^{\infty} 200(0.8)^t = \frac{200}{0.2} = 1000$

$V_\infty^{\pi^*}(s^2) = r V_\infty^{\pi^*}(s^3) = 0.8(1000) = 800 \rightarrow V^{\pi^*}(s^2) = 0 + 0.8 \cdot 200 + 0.8^2 \cdot 200 + \ldots$

$V_\infty^{\pi^*}(s^1) = 50 + r V_\infty^{\pi^*}(s^2) = 50 + r^2 V_\infty^{\pi^*}(s^3) = 50 + (0.8)^2(1000) = 690$

2. Consider the following MDP with $k+4$ states. There are two actions, a_1 and a_2. Arrows with no labels represent a transition for both actions with probability 1. Arrows labeled a/p make the transition on action a with probability p. States with no label have reward 0. Two states have reward +1, obtained when taking an action in that state. There are $k-2$ states between s_1 and s_k, with a deterministic transition on any action (so that once you are in s1 you are guaranteed to end up in s_k in $k-1$ steps).

We are interested in the infinite-horizon discounted values of some states in this MDP.



(a) What is $V^*_\infty(s_1)$ as a function of k when $\gamma = 0$? 0

(b) What is $V^*_\infty(s_1)$ as a function of k when $\gamma = 1$? 1 — reach s_k

(c) What is $V^*_\infty(s_1)$ as a function of k when $0 < \gamma < 1$? γ^{k-1} → no reward until s_k

(d) What is $V^*_\infty(s_x)$ when $\gamma = 0$? 0

(e) What is $V^*_\infty(s_x)$ when $\gamma = 1$? 1 ← eventually will reach s_y & get +1 ($r=1$)

(f) What is $V^*_\infty(s_x)$ when $0 < \gamma < 1$? $\frac{\gamma}{2-\gamma}$ ← 50% stay in s_x, 50% s_y & get +1

to reach @ step t: stay in s_x for $t-1$ steps (0.5^{t-1}) then go s_y 0.5: $P(t) = (0.5)^t$

so total = $\sum_{t=1}^{\infty} (\gamma \cdot 0.5)^t$ → geo series $\sum_{t=1}^{\infty} r^t = \frac{r}{1-r}$

$\therefore \frac{0.5\gamma}{1-0.5\gamma} = \frac{\gamma}{2-\gamma}$

(g) Under what conditions on k and γ would we prefer to take action a_1 in state s_0? Write down a specific mathematical relationship.

when $R(s_0, a_1) + \gamma(0.9 V^*_\infty(s_1) + 0.1 V^*_\infty(s_2)) > R(s_0, a_2) + \gamma V^*_\infty(s_x)$, which simplifies to $\frac{9}{10}\gamma^{k-1} > \frac{\gamma}{2-\gamma}$

a_1: reward but delayed k-1 steps

a_2: smaller reward but happens sooner

3. Consider a tabular MDP with given state space, action space, transition probabilities, reward function, and discount factor $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$. Assume that this MDP has no terminal state , i.e., a policy can be ran without termination.

(a) Assume that for a given policy π the infinite horizon value functions are V. Now suppose all rewards are increased by 10, i.e., the new reward function $R_{new}(s,a) = R(s,a) + 10, \forall s \in \mathcal{S}, a \in \mathcal{A}$.

Would the infinite-horizon value functions of following π change? If yes, write out an expression for the new value functions in terms of V (you do not need to worry about simplifying the expression). If no, explain why there will be no change.

YES! using definition of infinite horizon values, $V(s) = E[R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots | s]$

incr each R_i by 10 results in $V(s+10) = E[(R_0+10) + \gamma(R_1+10) + \gamma(R_2+10) + \ldots | s]$

$= V(s) + (10 + \gamma 10 + \gamma^2 10 + \ldots) = V(s) + \frac{10}{1-\gamma}$ for all states s

(b) Assume that the MDP described in part (a) has a known optimal infinite-horizon policy π^*. Now suppose all rewards are scaled by a constant $c \in \mathbb{R}$, i.e., the new reward function $R_{new2}(s,a) = c * R(s,a), \forall s \in S, a \in A$.

Is there any constant c such that the optimal policy π^* remains optimal? If yes, give an example of c. If no, briefly explain why not.

any $c \geq 0$ $\quad V^{*}_{new}(s) = E[cR_0 + \gamma cR_1 + \gamma^2 cR_2 + \ldots | s] = cV^{\pi}(s)$ for all states s & policies π.

π^* still optimal b/c $V^{\pi}(s) \leq V^{\pi^*}(s)$

4/23

Q = "any step + optimal policies for rest"

VALUE ITERATION:

- Q vs. V: $V_n^{\pi^*}(s) = \max_{\pi} V_n^{\pi}(s) = V_n^*(s)$ ← V alone doesn't tell you what to do. Q does

$$Q_n^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_{n-1}^*(s',a')$$

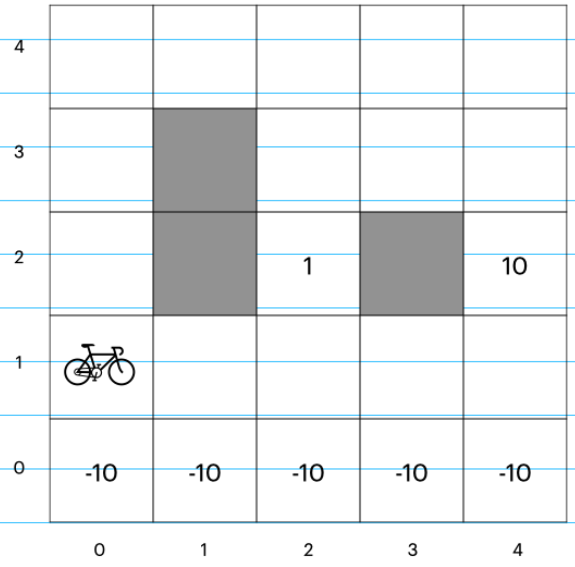
6.390 Introduction to Machine Learning

Recitation Topic: Markov Decision Processes

1. We're riding a bike in a 5-by-5 grid-world. We always start at (0,1). The shaded squares are pillars. From each state, we have four possible actions: North, South, East and West. If an action takes us into a pillar or out of the world, we do not move.

   We receive the reward written on the squares when taking any action from the square, where squares without numbers incur 0 reward.

   The game terminates after taking an action from (2,2) or (4,2). Our goal is to maximize the total reward.



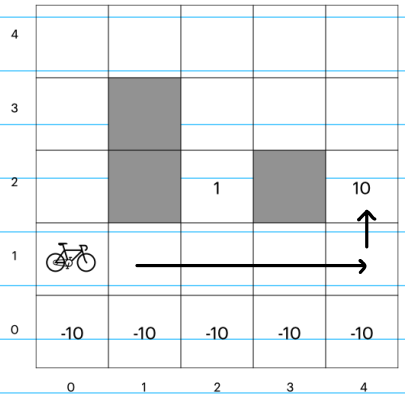
(a) In this part, we assume a discount factor of 1.0 and deterministic transitions.

Suppose we run the value iteration algorithm for 1000 iterations. What would be the optimal value of our initial state (0,1)? Draw out the best actions as arrows taken at each state along the best path. Is the best path unique?

Value of best path: 10

Unique? no b/c there are 1000 steps & discount factor = 1, we can take any route to get to (4,2)

Draw best path:



* not a unique soln b/c $\gamma = 1$ & non-determ, so many paths will produce 10

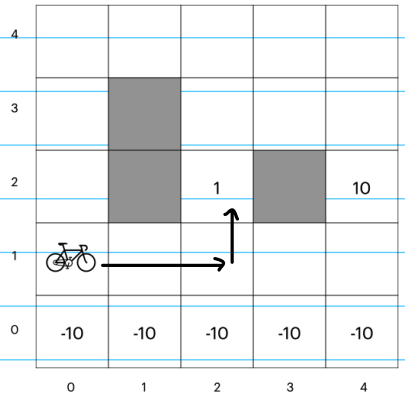
(b) In this part, we keep all of the setup from the previous part, except that we change the discount factor to 0.1.

Suppose we run the value iteration algorithm for 1000 iterations. What would be the optimal value of our initial state (0,1)? Draw out the best actions as arrows at each state along the best path. Is the best path unique?

Value of best path: $0 + 0.1(0 + 0.1(0 + 0.1(1))) = 0.1 \cdot 0.1 \cdot 0.1 = 0.001$

Unique? yes b/c any other path would be less value

Draw best path:



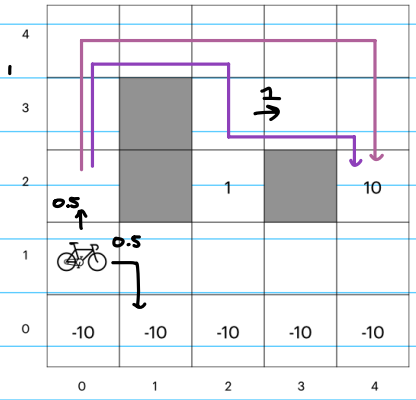
(c) In this part, we now assume a discount factor of 0.99. Our transitions are no longer deterministic: when taking an action, there is a 0.5 chance that we move to the intended square as before, and a 0.5 chance that we move to the intended square and then to the square directly South of it. As before, we cannot move out of the world or into a pillar.

For example, in state $(0, 1)$, a North action takes us to state $(0, 2)$ with probability 0.5 to state $(0, 1)$ with probability 0.5. In state $(2, 3)$, an East action takes us to state $(3, 3)$ with probability 1, since we cannot move into the pillar.

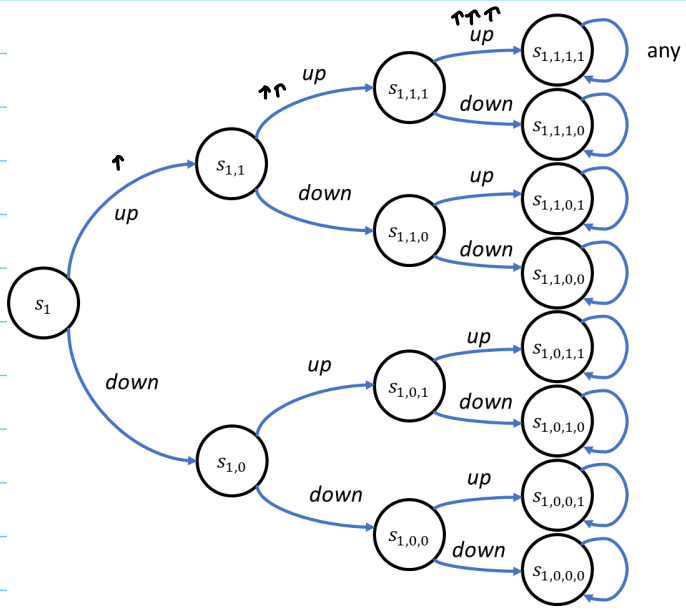
Suppose we run the value iteration algorithm for 1000 iterations. Draw out the best actions as arrows taken at each state you may reach in the path that maximizes your expected reward.

Draw the best actions:

$(0.5)^8(0.99)^9 \cdot 10$



2. Consider the following Markov decision process (MDP), with states as pictured in the figure below. In any state s_{label} we can take one of two actions, 'up' or 'down', and transition to one of the two states to the right of that state.



Our states s_{label} are defined by their subscript label, where label is a sequence of 0's or 1's indicating the position of that state in the tree as in the figure. For example, $s_{1,1,1,0}$ is the second from the top right-most state. The state tree can have any depth d; $d = 4$ for the example shown in the figure.

We first consider a deterministic version of this MDP as pictured above, where the transition function is such that taking action 'up' from state s_{label} moves us to state $s_{\text{label+"1"}}$, and action 'down' from state s_{label} moves us to state $s_{\text{label+"0"}}$, as shown in the figure, for states s with depths less than d (i.e., from states s_{label} where length of label is less than d). For "leaf" states s_{leaf} with depth equal to d, we stay in that leaf with any action. We always start in state s_1, and assume discount factor $\gamma = 0.9$.

(a) Assume a state tree of depth $d = 4$ as pictured above, and $\gamma = 0.9$. Consider a reward function $R(s, a)$ for taking action a from any state s that gives reward 3 for action 'up' and reward 2 for action 'down'.

i. What is the optimal horizon $h = 1$ action to take from state s_1? Explain your reasoning.

R(s., up) = 3 & R(s., down) = 2

optimal action = up b/c reward is greater.

ii. What is the optimal horizon $h = 1$ value for s_1, $s_{1,1}$, and $s_{1,0}$?

$V_1^*(s_1) =$ 3

$V_1^*(s_{1,1}) =$ 3

$V_1^*(s_{1,0}) =$ 3

* h=1: only count in reward, not what happens before/after

iii. What is the optimal horizon $h = 2$ value for s_1 and $s_{1,0}$?

$V_2^*(s_1) =$ up + up = 3 + 0.9(3) = 5.7

$V_2^*(s_{1,0}) =$ up + up = 3 + 0.9(3) = 5.7

iv. What is the optimal infinite horizon $h = \infty$ value for s_1 and $s_{1,0}$?

$V_\infty^*(s_1) =$ $3 + 0.9(3 + 0.9(3 + 0.9(\cdots))) = \frac{3}{1-0.9} = 30$

∞ geometric series: $\frac{a}{1-r}$

$V_\infty^*(s_{1,0}) =$ $3 + 0.9(3 + 0.9(3 + 0.9(\cdots))) = 30$

(b) Suppose that our state tree now has depth $d = 10$ rather than depth $d = 4$ as in the figure. Explain how your horizon $h = \infty$ value for s_1 and $s_{1,0}$ changes, or does not change, compared to your answer in part (a).

it does not change because $\gamma = 1$, and it's always more optimal to go up. converging to ∞, the value will still be $V_\infty^*(s_1) = V_\infty^*(s_{1,0}) = 30$ and keep looping

(c) We return to the $d = 4$ depth tree, as in our figure above. However, now we change our reward function such that $R(s, a)$ remains the same as in (a), *except* that for leaf states in our tree, $R(s_{\text{leaf}}, a) = 0$ for any action a. Assume now that $\gamma \in (0, 1)$ but we do not know its exact value.

i. Now what is the optimal infinite horizon $h = \infty$ value for s_1 and $s_{1,0}$? Express your result in terms of γ.

$V^*_\infty(s_1) =$ $3 + (0.9(3 + 0.9(3 + 0.9(3)))) = 3 + 2.7 + 2.43 + 2.187 = 10.317$

$3 + 3\gamma + 3\gamma^2 + 0$ ← leaf

$V^*_\infty(s_{1,0}) =$ $3 + 3\gamma + 0$ ← leaf

(d) Consider a stochastic version of our MDP. Now, taking action 'up' from any state transitions to the next upper state (with ", 1" added to the label) with probability p, and to the next lower state (with ", 0" added to the label) with probability $1 - p$. Taking action 'down' always transitions to the next lower state, and leaf states always transition back to that same leaf state, with probability 1. The reward structure remains as in part c (i.e., actions in leaf states give zero reward). As before, assume that depth $d = 4$, and $\gamma \in (0, 1)$ but we do not know its exact value. Similarly, we do not know the exact value of p.

i. What is the optimal infinite horizon $h = \infty$ value for s_1? Express your result in terms of p and γ.

$V^*_\infty(s_1) =$ $3(1 + p\gamma + p^2\gamma^2)$ ✗

$3(1 + \gamma + \gamma^2)$

$V^*_\infty(s_{1,0}) = V^*_\infty(s_{1,1})$, so p doesn't come into play

$V^*_\infty(s_1) = 3 + \gamma[p V^*_\infty(s_{1,1}) + (1-p) \cdot V^*_\infty(s_{1,0})]$

(e) Finally, our reward structure changes for one state in our MDP compared to part d: now the leaf at bottom right has a different reward, $R(s_{1,0,0,0}, a) = 100$, for any action a. The other leaf nodes continue to have reward zero.

i. Now, what are the infinite horizon Q^* values for taking action 'up' and 'down' from the state $s_{1,0,0}$? Express your answer in terms of γ and p.

$Q^*(s_{1,0,0}, \text{'up'}) = 3 + \gamma \cdot p \cdot 0 + (1-p)\gamma \frac{100}{1-\gamma}$ (∞)

∞ geometric series: $\frac{a}{1-r}$

up vs. down

$Q^*(s_{1,0,0}, \text{'down'}) = 2 + \gamma \frac{100}{1-\gamma}$ (∞)

ii. Suppose $\gamma = 0.9$. For what values of p will the optimal action to take in state $s_{1,0,0}$ become 'down'?

R(up) = 3, R(down) = 2

in order for optimal to be down action, $3 \cdot p < 2(1-p)$

$3p < 2 - 2p$

$5p < 2 \rightarrow p < \frac{2}{5}$ ✗

$2 + \gamma \cdot \frac{100}{1-\gamma} > 3 + \gamma(1-p)\frac{100}{1-\gamma}$

$p > \frac{1-\gamma}{\gamma} \frac{1}{100} = \boxed{\frac{1}{900}}$

4/30:

GOAL: determine $\pi^*(s)$, $V^{\pi^*}(s)$

↓ calculate using

$Q^*_\infty(s,a)$ start from state s & go to action a & go on optimal policy

known trans. T & rewards R → VALUE ITERATION (week 10b)

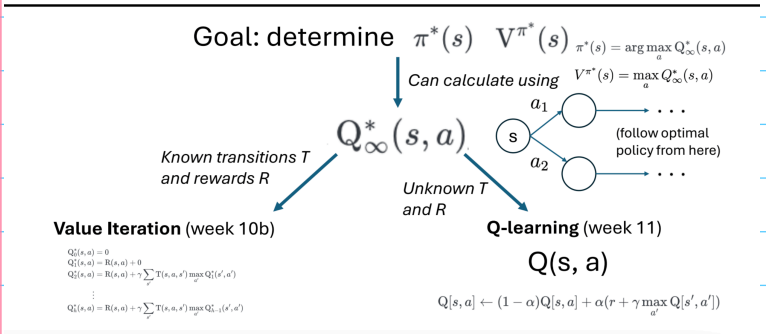
unknown T & R → Q-LEARNING Q(s,a)

Week 10a: $V^\pi_\infty(s)$

Weeks 10b & 11: **optimal** policy

Weeks 10a, 10b, 11

Week 11 / RL: mostly infinite horizon



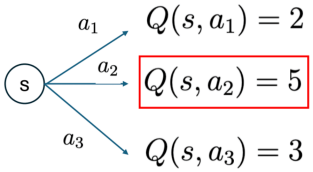
Q-LEARNING UPDATE ROLE:

$Q[s,a] \leftarrow (1-\alpha)\underbrace{Q[s,a]}_{\text{old guess}} + \alpha(\underbrace{r}_{\text{immediate reward}} + \gamma \underbrace{\max Q[s',a']}_{\text{future reward}})$ (new guess)

s —(reward r, a)→ s' → future rewards $\hat{V}^{\pi^*}(s') = \max_{a'} Q[s',a']$

ε-GREEDY:

- with probability 1-ε, choose $\text{argmax}_{a \in A} Q(s,a)$
- with probability ε, choose action $a \in A$ uniformly @ random



Greedy (we think this action will give the highest reward)

Mnemonic to remember what ε stands for: it's a small radius ε away from Greedy (so ε = 0 means fully Greedy)

6.390 Introduction to Machine Learning
Recitation Topic: Reinforcement Learning

1. You are helping a friendly alien named Zog learn to navigate a 3×3 maze on the planet *Quixara*. Zog hopes to reach the shiny treasure located in a specific cell in the maze. Along the way, Zog may encounter other treasures and pesky space traps.

   Your task is to use Q-**learning** to teach Zog to navigate the maze efficiently. The maze grid is structured as follows:

1	2	3
4	5	6
7	8	9

   Maze Details:

   - Any action in Cell 9 (bottom-right) gives a "gold" reward of $+\mathbf{10}$.
   - Any action in Cell 5 gives a "trap" reward of $-\mathbf{1}$.
   - Any action in Cell 2 gives a "silver" reward of $+\mathbf{1}$.
   - All other (state, action) pairs give a boring reward of $\mathbf{0}$.

   Movement and Transitions: Zog can attempt to take an action in **up**, **down**, **left**, **right**; however, the maze is unstable, and the transitions are unknown. For example, the probability of transitioning from location 1 to location 9 as a result of action "up" could be positive.

   Action Selection: Zog will explore the maze using an ε-greedy policy (with ε value to be specified in the sub-parts). When there is a tie, Zog chooses the action in this order: {**up, down, left, right**}.

   Discount factor: γ, the discount factor is set to $\mathbf{0.9}$.

   Before taking any actions, Zog's Q-table is initialized with zero values for all state-action pairs.

   For part (a) to part (c), $\varepsilon = 0$, that is, we use pure greedy action selection.

   Let's first refresh our understanding of Q-learning. The Q-learning process (depending on parameter $\alpha \in (0, 1]$) consists of initializing a function $\mathrm{Q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to $\mathrm{Q}(s, a) = 0$ (for all $s \in \mathcal{S}$, $a \in \mathcal{A}$), and then performing, for each state-action pair (s, a) reached, the "update $\mathrm{Q}(s, a)$" step, defined as follows:

   1. Execute action a at state s, to get the resulting deterministic reward $r = \mathrm{R}(s, a)$ and random $s' \in \mathcal{S}$ (with probability $\mathrm{T}(s, a, s')$);
   2. Change the value of $\mathrm{Q}(s, a)$ according to

$$\mathrm{Q}(s,a) \leftarrow (1-\alpha)\mathrm{Q}[s,a] + \alpha\left(r + \gamma \max_{a'} \mathrm{Q}[s', a']\right)$$

Note that, at a single update step, the value of Q changes at one argument pair (s, a) only.

(a) Fill in the Q-learning update equation above. Use α for the learning rate and γ for the discount factor.

(b) Now, let's go back to our problem. If Zog starts in Cell 4, what would be Zog's first action?

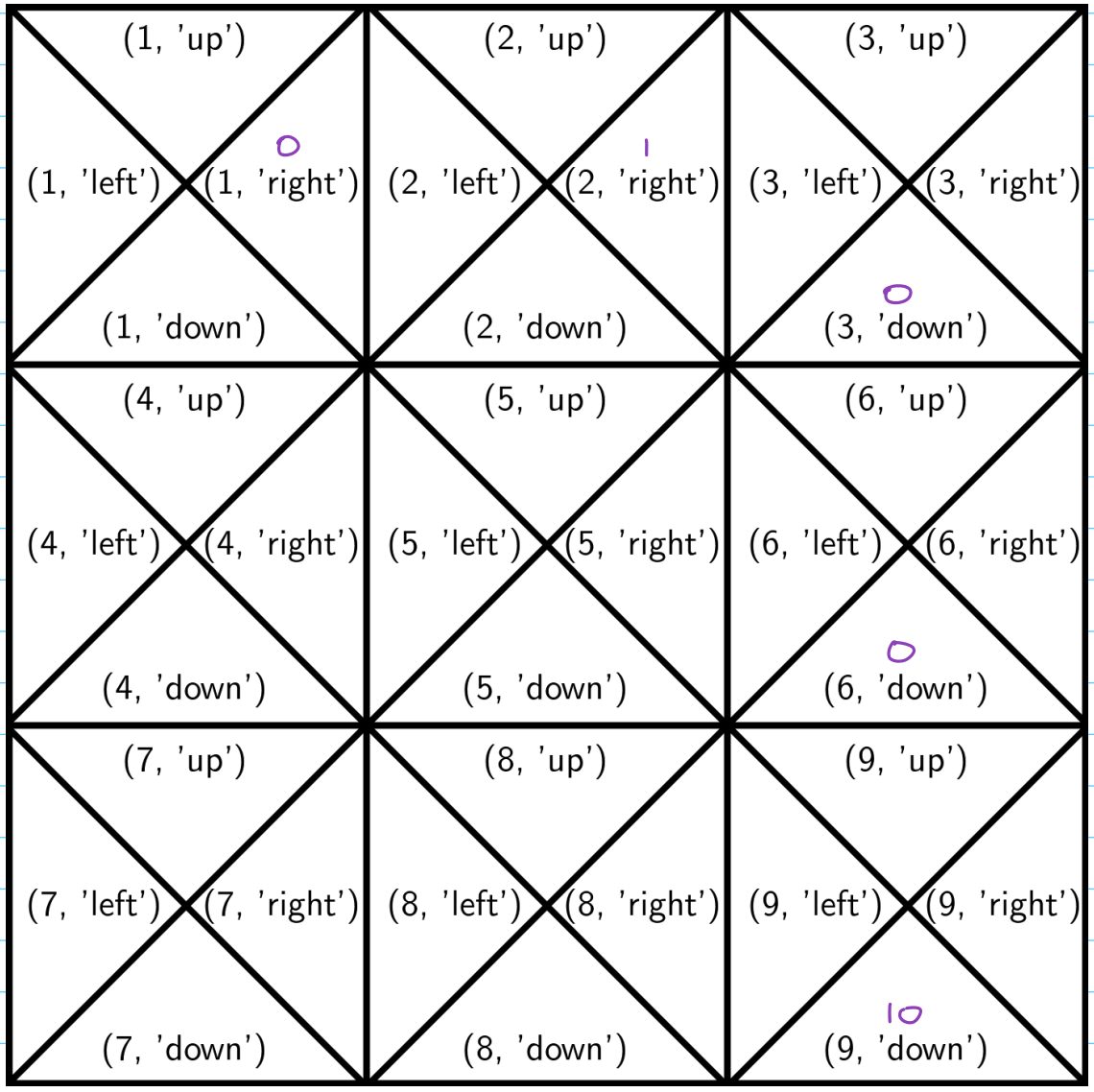
up. all Q-vals init. to 0 & break ties in order ↑, ↓, ←, →

(c) Is there a learning rate α such that Zog always chooses the action you gave in the previous part, any time they visit Cell 4?

when $\alpha = 0$, we never update the Q table, so always break tie @ state 4 by choosing up.

Reset the Q-table with zeros for all (state, action) pairs. For all subparts below, assume that we change the action-selection strategy to ε-greedy with some unknown ε, and fix the learning rate α to be 0.5.

(d) Zog ended up following this trajectory of (state, action, rewards):
(1, **right**, 0) → (2, **right**, 1)→ (3, **down**, 0) → (6, **down**, 0) → (9, **down**, 10).
There is no further transition out of state 9.
Solve for all non-zero Q-values after this trajectory.



Q[2, right] = 0.5·0 + 0.5(1 + 0.9·0) = 0.5

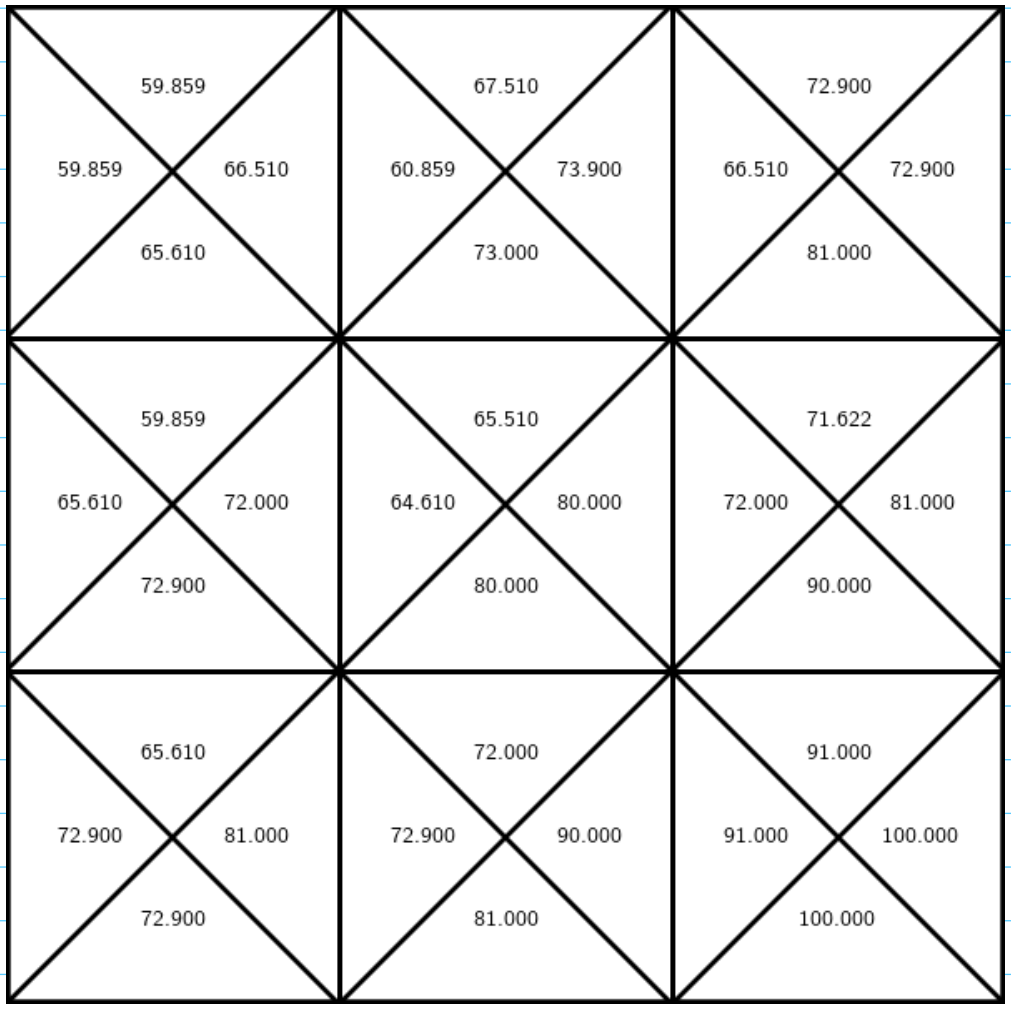
Q[9, down] = 0.5·0 + 0.5(10 + 0.9·0) = 5

(e) How would you adjust the reward so that Zog learns to prioritize avoiding traps over collecting silvers? How would you adjust the silver reward? How would you adjust the trap penalty?

avoiding traps: lowering ϵ (more greedy?)

trap penalty negative, silver reward smaller

(f) After 10 million tries, someone tells you your Q-learning algorithm has converged. Yay! Here is the Q-table:



i. Would it be possible to determine the transition probability $T(9, \textbf{right}, 9)$? If yes, give the transition probability. If not, explain why not. Again, recall that γ is set to 0.9.

Q^* satisfies the Bellman equation

$$Q^*_{\text{opt}}(s,a) = \underbrace{R(s,a)}_{\text{imm. r.}} + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^*_{\text{opt}}(s',a')$$

P(go to state s'). best reward from s'

$100 = Q(9, \text{right}) = 10 + 0.9 \cdot (\text{weighted sum})$

b/c 100 biggest #, $\max Q^*(s', a') = 100$, $\rightarrow T(9, \text{right}, 9) = 1$

ii. Would it be possible to determine the transition probability T(9, **up**, 6)? If yes, give the transition probability. If not, explain why not.

91: Q(9, up) = 10 + 0.9 (weighted sum) → weighted sum = 90

many states could be... can't tell

iii. Would it be possible to determine the transition probability T(6, **down**, 3)? If yes, give the transition probability. If not, explain why not.

90 = Q(6, down) = 0 + 0.9 · (weighted sum) → weighted sum = 100
→ 6 must transition to state 9

→ T(6, down, 3) = 0

(g) Now assume $\varepsilon = 0.4$. Using the Q-table from the previous part, what is the probability that Zog chooses action **down** when in state 6?

$\frac{1}{4} \cdot 0.4 = 0.1$ total prob. is $0.6 + 0.1 = 0.7$

(h) Reset the Q-table with zeros for all (state, action) pairs. Now, let's assume that all states with nonzero reward are terminal states (i.e., 2, 5, and 9), so we end the trajectory once we reach them.

This time Zog starts in cell 1 and has explored (and terminated) two trajectories, but we don't know which ones they are! All we know is that he continued until he reached one of the terminal states and updated the Q-table along the way. Consider state 6: after two episodes of Q-learning, what are the possible values of Q(6, up)? Remember the transition function is unknown, $\alpha = 0.5$, and $\gamma = 0.9$.

- ◯ 9
- ◯ 4.5
- ● 2.25
- ● 0.225
- ● 0
- ◯ -0.225

Problem 1h hint

How can Q[6, up] be updated from 0 to a non-zero value?

$$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'])$$

Q[6, up] = 0.5 * 0 + 0.5(0 + 0.9 * ?)

= ?

We need to transition from state 6 to a state s which already was updated to have nonzero Q value for some action (max Q(s, a) > 0)

Initially, the only updates that can cause nonzero Q-value are from the terminal states (at the end of the first episode).

→ **To have a non-zero Q[6, up], 6 up must transition to state 2, (5?), or 9.**

Problem 1h sol

How can Q[6, up] be updated from 0 to a non-zero value?

$$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'])$$

Q[2, up] = 0.5 * 0 + 0.5(1)

= 0.5

→ max Q[2, a] = 0.5

Q[5, up] = 0.5 * 0 + 0.5(-1)

= -0.5

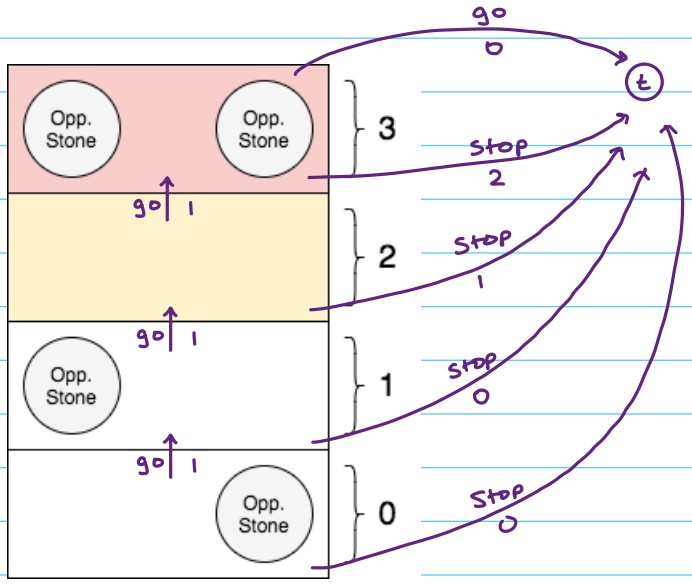
→ max Q[5, a] = 0 still! Not non-zero.

2. You have designed a robot curling stone to enter a modified curling contest. In order to get your robot stone to perform well, you have designed a state and action space, a reward structure, and a transition model. The goal of the robot stone is to slide upwards on an ice sheet and stop in a target region. After each state transition, it displays the reward it receives. In addition to your robot stone, there are some opponent stones on the ice, as shown below, which are fixed.

Your model for the state and action spaces is as follows:

$$S \in \{t, 0, 1, 2, 3\}$$
$$a \in \{\text{“go”}, \text{“stop”}\}$$

where the states refer to the robot stone being in either a terminal state (denoted as t) or within one of the four regions below:



You design the following reward function and (deterministic) transition model for your robot stone:

$R(s,a)$:

	action "go"	action "stop"
state 3	0	2
state 2	1	1
state 1	1	0
state 0	1	0

$T(s,a,s')$:

T(0, "go", 1) = 1
T(1, "go", 2) = 1
T(2, "go", 3) = 1
T(3, "go", t) = 1
T(*, "stop", t) = 1

and all other transition probabilities are 0. Here * indicates any state. Note that once the robot stone enters state t the game ends: there is no transition and zero reward out of state t (and hence no action to decide once in state t). Together with this reward function and transition model, you specify a discount factor $\gamma = 1$.

(a) In order to enable decision making by your robot stone, you need to give it the optimal policy $\pi^*(s)$. For your reward and transition structure and discount factor $\gamma = 1$, what are the optimal Q-values, $Q^*(s,a)$? What is the optimal policy $\pi^*(s)$? Fill in the following two tables.

*current + optimal future

$Q^*(s,a)$:

	action "go"	action "stop"
state 3	0	2
state 2	3	1
state 1	4	0
state 0	5	0

$\pi^*(s)$:

	$\pi^*(s)$
state 3	stop
state 2	go
state 1	go
state 0	go

Unfortunately, your competitor has also designed a robot stone. You do not know your competitor's reward structure $R(s, a)$ or transition model $T(s, a, s')$; however, you do know they use the same state and action spaces. Instead, you decide to use Q-learning to observe their robot stone and learn from it! For your Q-learning, use discount factor $\gamma = 1$ and learning rate $\alpha = 0.5$, with a Q table initialized to zero for all (s, a) pairs.

(b) Your competitor runs their robot through a first game, exhibiting the following experience:

step #	s	a	r	s'
1	0	"go"	1	1
2	1	"stop"	0	t

from the state we are at, max reward when going to others

$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$

You perform Q-learning updates based on the experience above. After observing steps 1 and 2 (the first game), what is the learned Q(0, "go")?

$Q(0, \text{"go"}) = 0.5 \cdot 0 + 0.5(1 + 1 \cdot 0) = 0.5$ * Q-table init. to 0

What is the learned Q(1, "stop")?

$Q(1, \text{"stop"}) = 0.5 \cdot 0 + 0.5(0 + 1 \cdot 0) = 0$

(c) Your competitor runs their robot through a second game, exhibiting the following **additional** experience:

step #	s	a	r	s'
3	2	"go"	1	3
4	1	"go"	1	2
5	2	"go"	1	3
6	3	"stop"	2	t

$\max_{a'} Q[s', a']$ looks for max val of $Q[2, \text{"go"}, \text{"stop"}]$

You perform additional Q-learning updates based on this additional experience. After completion of both games (all six steps), what are the full set of Q values you have learned for their robot? Fill in the following table.

$Q(0, \text{"go"}) = 0.5 \cdot 0.5 + 0.5(1 + 1 \cdot 0) = 0.75$

$Q(1, \text{"go"}) = 0.5 \cdot 0 + 0.5(0 + 1 \cdot 0) = 0.5$

$Q(2, \text{"go"}) = 0.5 \cdot 0 + 0.5(0 + 1 \cdot 0.5) = 0.25$

$Q(3, \text{"stop"}) = 0.5 \cdot 0 + 0.5(2 + 1 \cdot 0) = 1$

$Q(s, a)$:

state s	action "go"	action "stop"
3	0	1
2	0.5	0
1	0.25	0
0	0.75	0

(d) We can think of learning the Q-value function for a given action as a regression problem with each state s mapped to a one-hot feature vector $x = \phi(s)$, where $x = [1\ 0\ 0\ 0]^T$ for state 0, $x = [0\ 1\ 0\ 0]^T$ for 1, etc., and $x = [0\ 0\ 0\ 0]^T$ for state t.

We'll focus on the action "go". We would like to come up with parameters θ, θ_0 such that $Q(s, \text{"go"}) = \theta \cdot \phi(s) + \theta_0 = \theta \cdot x + \theta_0$.

Given an arbitrary set of values for $Q(0, \text{"go"}), Q(1, \text{"go"}), Q(2, \text{"go"}), Q(3, \text{"go"})$, can we always find a setting of θ, θ_0 that enables representation of $Q(s, \text{"go"})$ with perfect accuracy? If so, provide the corresponding θ and θ_0. If not, explain why.

find θ, θ_0 s.t. $\theta^T \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \theta_0 = \theta_1 + \theta_0 = Q(0, \text{"go"})$

$\theta_2 + \theta_0 = Q(1, \text{"go"})$ $\rightarrow \theta = \begin{bmatrix} Q(0, \text{"go"}) \\ \vdots \\ Q(3, \text{"go"}) \end{bmatrix}$

$\vdots$

with only one action, Q can be represented as linear func. of state

5/7 rec 12: non-parametric models

→ doesn't assume fixed functional form for h; complexity grows w/ data

Why care?

- interpretability
- insight
- speed
- adaptability (low data regime: no way you can train deep NN on it)

6.390 Introduction to Machine Learning
Recitation Topic: Non-parametric Models

As a reminder before starting the questions:
We define the *empirical probability* of an item from class k occurring in region m as:

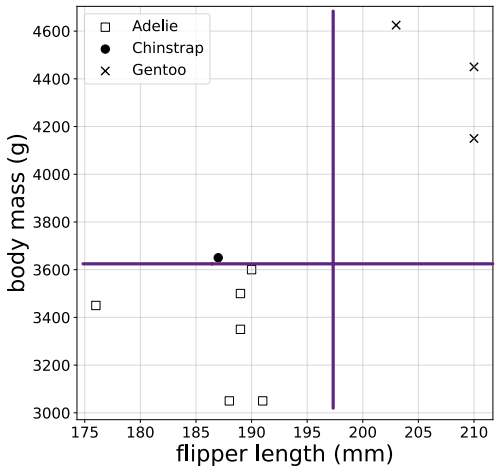
$$\hat{P}_{m,k} = \hat{P}(I_m, k) = \frac{\left|\{i \mid i \in I_m \text{ and } y^{(i)} = k\}\right|}{N_m},$$

where N_m is the number of training points in region m; that is, $N_m = |I_m|$.
Entropy is defined as follows:

$$Q_m(T) = H(I_m) = -\sum_k \hat{P}_{m,k} \log_2 \hat{P}_{m,k} \tag{1}$$

So that the entropy H is well-defined when $\hat{P} = 0$, we will stipulate that $0 \log_2 0 = 0$.

1. Consider the small dataset about Penguins shown below. The data set includes three penguin species: Adelie, Gentoo, and Chinstrap. In the plot below, Adelie are marked with squares, Gentoo with x's, and Chinstrap with o's. Note that the Chinstrap penguin has flipper length 187mm and body mass 3650g.



We now use two features—flipper length and body mass—to build a decision tree that classifies each penguin's species using entropy as the splitting criterion.

(a) Calculate the entropy of this subset of the penguin data set. There are 6 Adelie, 1 Chinstrap and 3 Gentoos.

$H = -\frac{6}{10}\log_2\left(\frac{6}{10}\right) - \frac{1}{10}\log_2\left(\frac{1}{10}\right) - \frac{3}{10}\log_2\left(\frac{3}{10}\right)$

(b) For each feature (flipper length and body mass):

- choose an appropriate split point based on the data visualization, and

- calculate the Weighted Average Entropy (WAE) after splitting.

  Hint: Recall the WAE, $\hat{H}$ of a split on the jth dimension at location s is: $\hat{H} =$ (fraction of points in left data set)$\cdot H(I_{j,s}^{-})+$(fraction of points in right data set)$\cdot H(I_{j,s}^{+})$

if perfectly split, entropy = 0

for split: flipper length 200 mm

left: $-\frac{6}{7}\log_2(\frac{6}{7}) - \frac{1}{7}\log_2(\frac{1}{7})$

right: $\frac{3}{3}\log_2(\frac{3}{3}) = 0$

$\hat{H} = \frac{7}{10}(-\frac{6}{7}\log_2(\frac{6}{7}) - \frac{1}{7}\log_2(\frac{1}{7})) + \frac{3}{10}(0) = 0.414$

for split: body mass 3649 g:

up: $-\frac{1}{4}\log_2(\frac{1}{4}) - \frac{3}{4}\log_2(\frac{3}{4})$

down: 0 (homogenous)

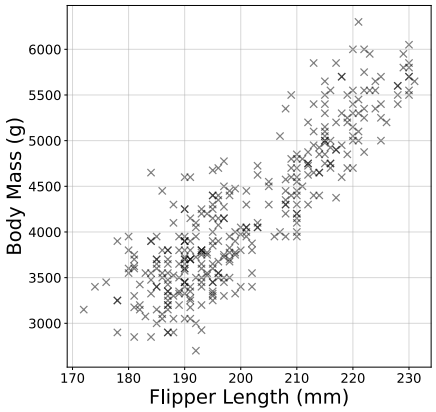
$\hat{H} = \frac{4}{10}(-\frac{1}{4}\log_2(\frac{1}{4}) - \frac{3}{4}\log_2(\frac{3}{4})) + \frac{6}{10}(0) = 0.325$

LOWER = better

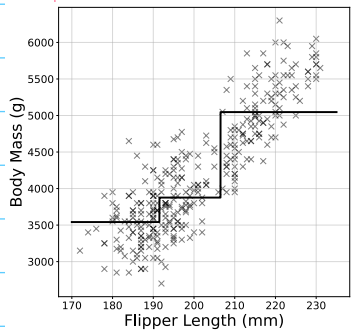
(c) Which feature would you select for the first split in your decision tree? Justify your answer using the WAE values.

lower WAE = better! body mass ≤ 3605 g

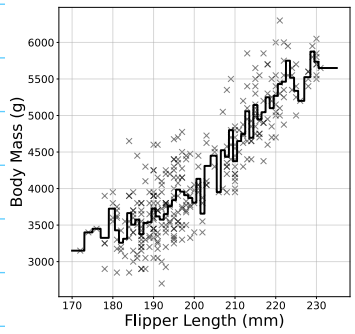
2. In this problem we will analyze the Palmer Penguins data set for a regression task. Our goal will be to predict the body mass of a penguin given the length of its flipper. The data set is plotted on the right.



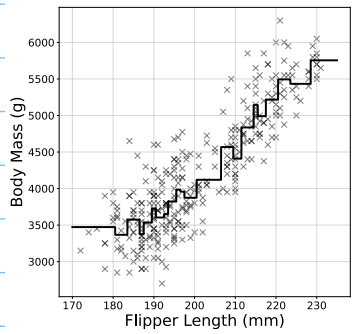
(a) We run the BuildTree algorithm on the Penguins data set with varying levels of largest leaf sizes: $k = 1, 10, 100$. Match the regressor with the largest leaf sizes of the decision tree.



k=100



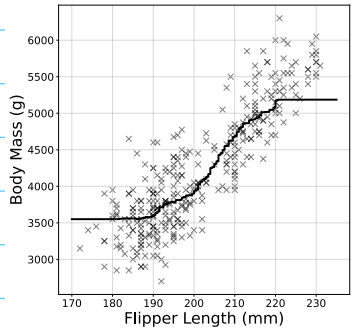
k = 1



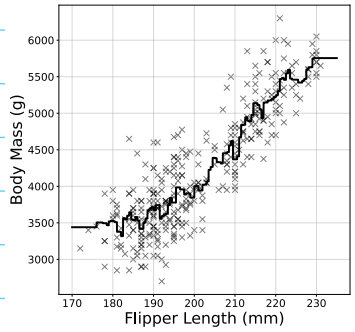
k = 10

Using the regressor in the first plot, what would the predicted body mass be for a penguin with flipper length 180m? → 3600 g

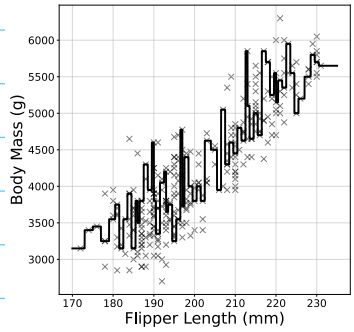
(b) We run the k-Nearest Neighbors algorithm using Euclidean distance on the Penguins data set with varying number of neighbors: $k = 1, 10, 100$. Match the regressor with the number of neighbors.



k=100

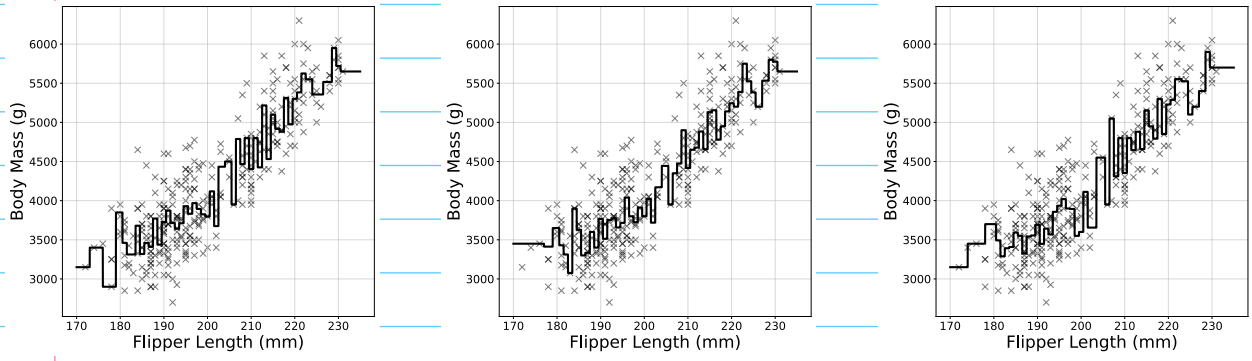


k=10



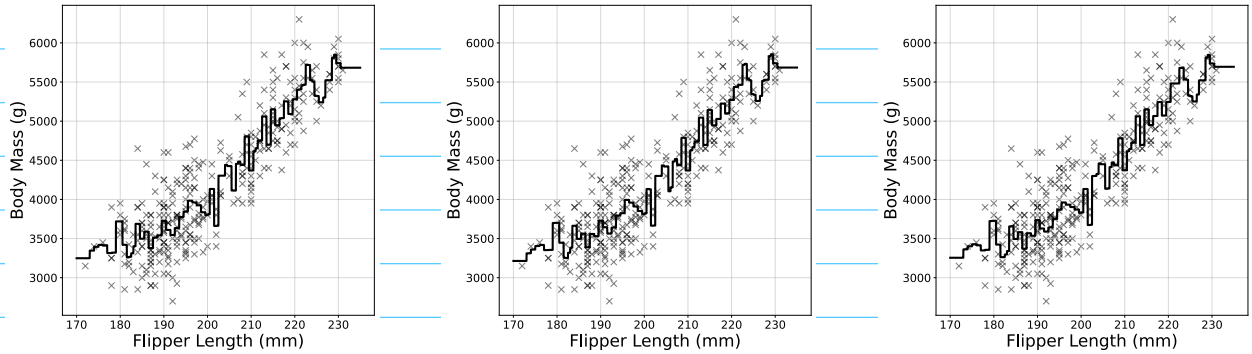
k=1

(c) Each of the three decision plots represents a decision tree trained on a bootstrap version of the dataset. That is, for each plot, the original data is resampled with replacement to form a new dataset of equal size, and a decision tree with a maximum leaf size of 1 is trained on this sample.



In each of the decision plots below, a bagged regressor is constructed by training an ensemble of 100 decision trees, each on a different bootstrap sample of the dataset. The final prediction is obtained by averaging the outputs of all trees.

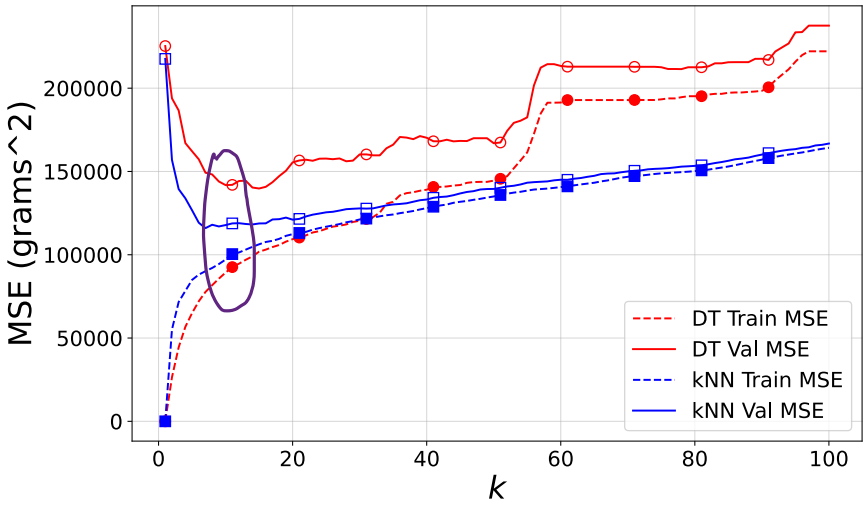
ensemble all looked almost identical to one another



Compare the bagged tree with individual tree. Which method is more sensitive to the random sampling?

average of 100 trees is less "jumpy" (reducing noise)

(d) We evaluate how hyperparameter choices affect mean squared-error (MSE) for decision trees and k-Nearest Neighbors in both training and validation settings. For each k from 1 to 100, we run ten-fold cross-validation. The plot shows average training and validation MSE across the folds. We modify the training data set to include three features: flipper length (mm), bill length (mm), and bill depth (mm) as inputs to the models. The goal remains to predict the body mass (g).



How do the hyperparameters (largest leaf size and number of neighbors) affect training and testing accuracy for each algorithm? Based on the plot, which value of k would you recommend for decision trees and for k-Nearest Neighbors?

k too big → averages many things (larger leaf size).

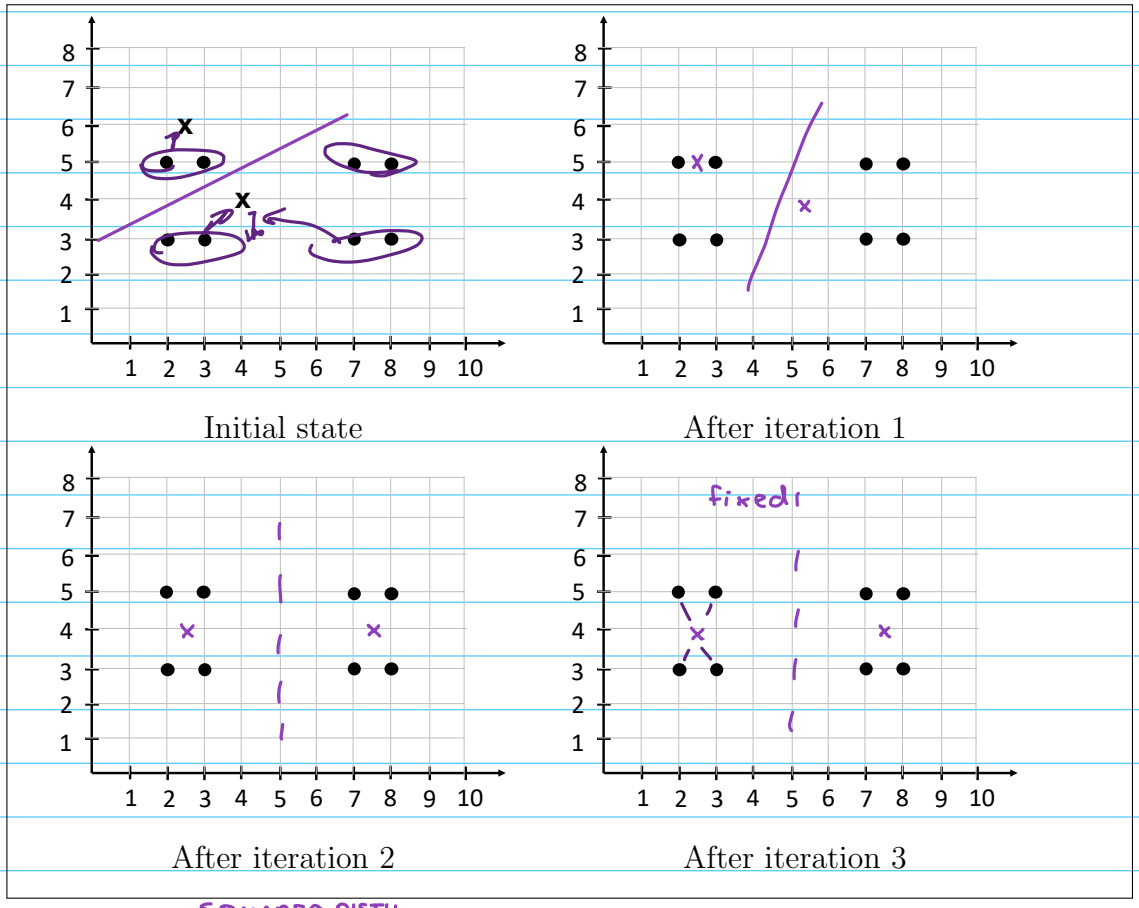
KNN → k = 11

DT → k = 11

both minimized

3. Walk through each step of the k-means clustering algorithm, beginning with the initialization shown in the plot in the top left of the box below. Dots show the observed data. Assume that the number of clusters $k = 2$ for all of the following questions. In each plot (go left to right, top to down), mark with two '**x**' symbols where the cluster centers are in that iteration of k-means. Once the k-means algorithm has converged, you can leave all subsequent plots unmarked. k-means (2 centroids)

(a) First, consider initializing two cluster centroids at $(2.5, 6)$ and $(4, 4)$.



(b) What is the numerical value of the k-means objective for the clustering found in (a), after the algorithm has finished running? SQUARED DIST!!

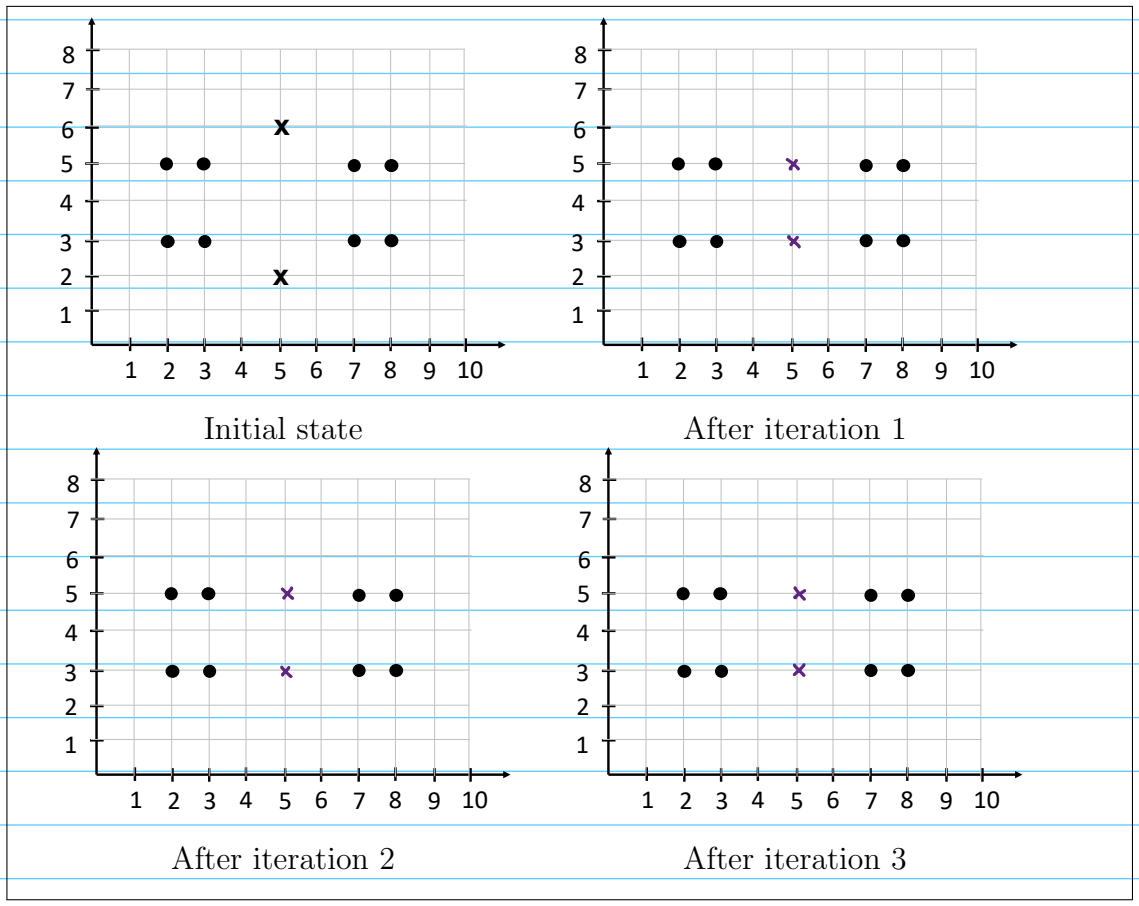
$(4 \cdot (1^2 + 0.5^2)) \cdot 2 = 10$

(c) Just as in (a), walk through each step of the k-means algorithm, beginning with the initialization shown in the plot in the top left, with the two cluster centroids at $(5, 6)$ and $(5, 2)$.

K-MEANS ALGO:

1. reassign μ's

2. reassign pts to μ's



(d) ~~What is the numerical value of the k-means objective for the clustering found in (c), after the algorithm has finished running?~~

$(3^2 + 2^2 + 2^2 + 3^2)\,2 = 52$

(e) According to the k-means objective of the learned clusters, which initialization was better?

● Initialization (a) ○ Initialization (c)