CAREFUL OF:
- dimensions

wk 1-4
cat tu: 6.390

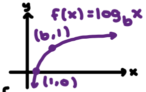
when X^TX nearly not invertible: there are many solns that fit the data almost equally well. OLS chooses a high sloping one.
Ex: $\begin{bmatrix} -2 & 4.06 \\ -1 & -2 \\ 2 & 4 \end{bmatrix}$

ex: $\frac{1}{4}\sum_{i=1}^{4}\left((1\ 0\ 0\ \ldots)\begin{bmatrix}\vdots\end{bmatrix}-\ldots\right)$

$\tilde{\theta}: (d+1)\times 1$
$\tilde{X}: n\times(d+1)$
$Y: n\times 1$

→ when well defined, θ^* = unique minimizer of $J(\theta)$

vectorized MSE:



	Raw features	Standardized
θ_1	tiny (0.1) — x_1 range is large	fair (1.5)
θ_2	large (1.5) — x_2 range is small	fair (1.2)
Ridge penalty	unfairly hammers θ_2	treats both equally
x_1 influence	understated	correctly captured

$\lambda(\theta_1^2+\theta_2^2)$

standardization:
- comparable vars.
- ensure λ treats all fts fairly

[SUPERVISED LEARNING:]

REGRESSION: predicting continuous #

training data: $D_{train} = \{(x^1,y^1),\ldots,(x^d,y^d)\}$

feature vector $x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d$

label $y \in \mathbb{R}$ ← scalar

n dps, each w/ d dim feats & scalar label (n×d x, n×1 y)

LINEAR REG: $h(x,\theta,\theta_0) = \theta^Tx + \theta_0$ (params, feats; h outputs predicted value)
→ gives analytical soln. but prone to fail
$g = h(x)$ → "guess"
a = actual obser.

SQUARED LOSS: $L(g,a) = (g-a)^2$ (loss func. — how wrong w a sh)

MSE: $J(\theta,\theta_0) = \frac{1}{n}\sum_{i=1}^{n}(\theta^Tx^i + \theta_0 - y^i)^2$ → J convex in θ & θ₀
→ augment x w/ 1 for all extra features — ALWAYS + (squared)

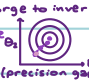
$J(\theta,\theta_0) = \frac{1}{n}\sum_{i=1}^{n}(\theta^T\tilde{x}^i - y^i)^2$

minimize: $\nabla J(\theta) = \frac{2}{n}\tilde{X}^T(\tilde{X}\theta - Y)$ → set to 0 for min
lin dep: 2nd features no add val
so $J_{before} = J_{after}$

$\theta^* = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^TY$ (d×d, d×n, n×1)
→ if X lin dep: X^TX NOT invertible (fails if features are lin. dep., correl. or if d ≫ n → matrix too big to invert)

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

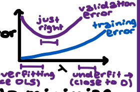
GRADIENT DESCENT: gradually update soln to improve when can't solve OLS or RR analytically
→ use when objective/loss not quadratic (highest power of θ = 2) or when matrix not invertible / too large to invert
downhill b/c — LR scaling factor — $L(g,y)$ → must be differentiable



$\theta^t = \theta^{t-1} - \eta\nabla_\theta J(\theta^{t-1})$
repeat until $|J(\theta^t) - J(\theta^{t-1})| < \epsilon$ (precision gap)
$\theta^* = \arg\min_\theta J(\theta)$ ← param val w/ smallest loss
aiming for GLOBAL MINIMIZER:
- gradient vector = 0 (#doesn't work if NOT convex)
- objective func. J is convex

ASSUMING f:
- suffic. small
- 1+ global min
- convex
- η suff. small

$\nabla_\theta J(\theta) = \frac{2}{n}X^T(X\theta - Y)$ set = 0 & solve for θ



REGULARIZATION:
→ guarantees unique analytical soln.
→ fights overfitting

OBJECTIVE FUNC J(θ): trying to minimize
smaller coeff = less flexible model = worse training fit

$J_{ridge}(\theta,\theta_0) = \frac{1}{n}\sum_{i=1}^{n}(\theta^Tx^i + \theta_0 - y^i)^2 + \lambda\|\theta\|^2$
→ ↑ λ vals make θ vals closer to 0 = high MSE

$\nabla J_{ridge}(\theta,\theta_0) = \frac{2}{n}\sum_{i=1}^{n}(\theta^Tx^i + \theta_0 - y^i)x^i + 2\lambda\theta$

$\theta^*_{ridge} = (X^TX + n\lambda I)^{-1}X^TY$ (convex) → always + def. when λ>0, so has unique, closed-form soln.
large λ = ‖θ‖ flat line, λ→0 → ‖θ‖ depend on data, λ=∞, ‖θ‖→0 b/c penalty term rewards large vals

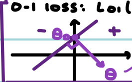
$R(\theta) = \|\theta - \theta_{prior}\|^2$ #λ only affects θ, NOT θ₀!
θ_{prior} = value/set of values you want to penalize towards
often use $R(\theta) = \|\theta\|^2$ b/c want to penalize toward $\theta_{prior} = 0$

$\lambda\|\theta\|^2 = \lambda\sum_j\theta_j^2$ ← ex: $\theta = [\theta_1,\theta_2] \to \lambda(\theta_1^2+\theta_2^2)$
constants: $\|c\theta\|^2 = c^2\|\theta\|^2$

STOCHASTIC GD: @ each step, randomly choose pt to calculate gradient
$\theta^t = \theta^{t-1} - \eta\nabla_\theta J_i(\theta^{t-1})$ until $|J(\theta^t) - J(\theta^{t-1})| < \epsilon$
→ less computational & more likely to find local min
→ might take more time (more random)
→ works whether convex or not! (may get us out of local min)
squared error loss: assume $L(\theta) = (y - \theta x)^2$
then $\nabla_\theta L(\theta) = -2x(y - \theta x)$ ← consider vals for each pt!

LINEAR CLASSIFIER:
$h(x,\theta,\theta_0) = \text{sign}(\theta^Tx + \theta_0) = \begin{cases} +1 & \text{if } z > 0 \\ 0 & \text{o.w.} \end{cases}$
x = feature vect.
θ = weight vect.
θ₀ = bias
why use?
- bounded (0,1)
- good fitting

0-1 loss: $L_{01}(g,a) = \begin{cases} 0 & \text{if } g = a \\ 1 & \text{o.w.} \end{cases}$ → hard to minimize



ex: $\theta = [1,-1]^T$, $\theta_0 = 1$, ANY + scaling works!
classifier is ⊥ to θ & θ vector
pts in + direction! #FURTHER from pts it's more certain of
θ₀ = y-int of θ

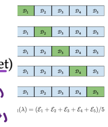
↑ g = lower loss = more confident loss = lower NLL ✓

NEGATIVE LOG LIKELIHOOD (NLL): $\sum_{i=1}^{n}(y^i\log g^i + (1-y^i)\log(1-g^i))$ (loss func., likelihood/prediction)
→ convex func (can run GD), minimize NLL = maximize likelihood; ↑ for correct label = good
OBJ. FUNC OF LLC: $J_{lr}(\theta,\theta_0,D) = \left(\frac{1}{n}\sum_{i=1}^{n}L_{nll}(\sigma(\theta^Tx^i + \theta_0), y^i)\right) + \lambda\|\theta\|^2$
→ need regularization for good generalization, lower = better #NLL is strictly convex!
→ minimizing Σlog probability of correct answer (make probability of correct label as large as possible)

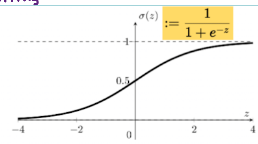
MATRIX MATH:
scalar × vector: $\frac{\partial y}{\partial x} = \begin{bmatrix} \partial y/\partial x_1 \\ \vdots \\ \partial y/\partial x_n \end{bmatrix}$
vector × vector: $\frac{\partial y}{\partial x} = \begin{bmatrix} \partial y_1/\partial x_1 & \cdots & \partial y_m/\partial x_1 \\ \vdots & & \vdots \\ \partial y_1/\partial x_n & \cdots & \partial y_m/\partial x_n \end{bmatrix}$ m×n
denominator layout: $\left(\frac{\partial b}{\partial a}\right)_{ij} = \frac{\partial b_j}{\partial a_i}$ → denom. determines shape
multivar. chain rule: $\nabla_\theta J(\theta) = \frac{\partial J}{\partial \theta} = \frac{\partial v}{\partial \theta}\frac{\partial J}{\partial v}$

EVALUATING:
LEARNING ALG: procedure that takes dataset D_n & returns hypothesis h from hypothesis class H. D_n → learning alg(H) → h
VALIDATION: train on new training set then evaluate h on validation set w/ no overlap w/ training set. → execute multiple times
CROSS-VAL: re-use train data which can be hard to get

CROSS-VALIDATE(𝒟, k) — find λ w/ lowest avg. error
1. divide 𝒟 into k chunks $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_k$ (of roughly equal size)
2. for i = 1 to k (for each candidate of λ):
3. (ridge reg.) train h_i on $\mathcal{D}\setminus\mathcal{D}_i$ (withholding chunk $\mathcal{D}_i$ as the validation set)
4. compute "test" error $\mathcal{E}_i(h_i)$ on withheld data $\mathcal{D}_i$
5. return $\frac{1}{k}\sum_{i=1}^{k}\mathcal{E}_i(h_i)$ validation error — return $\lambda^* = \arg\min_\lambda \mathcal{E}_{val}(\lambda)$
6. retrain final model h^* w/ Dtrain using reg. (b/c solve for λ*)
7. evaluate h^* on Dtest



LINEAR LOGISTIC CLASSIFIER:
SIGMOID σ(·): confidence/estimated likelihood that x belongs to + class
- monotonic elegant gradient (smooth)



$\sigma(z) = \frac{1}{1+e^{-z}}$ approaches 1 as z→+∞, 0 as z→−∞
→ higher z mag = more confident = pushes closer to 0 & 1
$\sigma(0) = \frac{1}{2}$, $\sigma'(0) = \sigma(0)(1-\sigma(0))$

FOR y=1: $L = -\log(g)$
g≈1 = correct & confident
larger z = smaller loss
smaller z = larger loss

FOR y=0: $L = -\log(1-g)$
g≈0 = correct & confident
larger +z = highest loss
smaller −z = lower loss

SOFTMAX:
$P_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$
→ higher score = ↑ p
training: compared to true label w/ cross entropy loss
$L = -\sum_{k=1}^{K} y_k\log(P_k)$ (label, one-hot; softmax prob.)
↳ for one pt
prediction: choose largest prob.
class = $\arg\max_k P_k$ (largest logit)

LOGIT:
$z_k = \theta_k^Tx$

MULTI-CLASS CLASSIFIER:
activation func:
$\text{softmax}(z) := \begin{bmatrix} \exp(z_1)/\sum_{k=1}^{K}\exp(z_k) \\ \vdots \\ \exp(z_K)/\sum_{k=1}^{K}\exp(z_k) \end{bmatrix}$
produces multiple outputs, each giving a probability of z in class k
→ choose class w/ highest probability & return one-hot encoding (logit)
→ handles multiple catagories & sums to 1

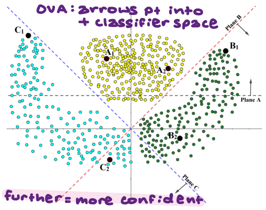
NLLM LOSS FUNC: $L_{nllm}(g,y) = -\sum_{k=1}^{K} y_k\log(g_k)$
→ all y_k = 0 except one (the true class)
→ only use when classes are independent
high probability to true class = small loss; low p = big loss
if classes dependent: use softmax NOT multiple σ's & total loss = sum of NLL across classes

DECISION BOUNDARY: when classifier unsure, h(x)=0.5 b/c σ(0)=0.5
#many params produce same boundary (diff mags)

ONE vs. ALL (OVA):
- seperate classifier for each class
- N classifiers (for N classes)
- good for many classes, efficient

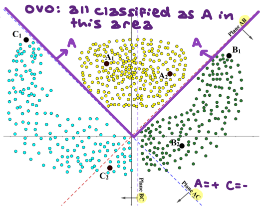
ONE vs. ONE (OVO):
- seperate classifier for every pair
- $\frac{N(N-1)}{2}$ classifiers (for pairs)
- expensive for many classes, but good @ handling imbalanced data

OVA: arrows pt into + classifier space



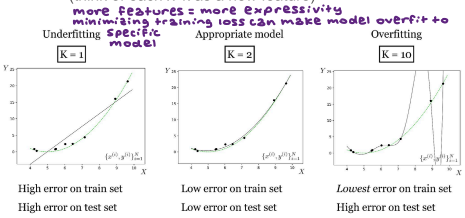
further = more confident

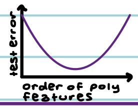
OVO: all classified as A in this area



ONE HOT ENCODING: all vals = 0 except jth, which is 1 $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

UNDER/OVERFITTING:
polynomials of degree k that minimize training loss (think of each x^k as a new feature)
more features = more expressivity
minimizing training loss can make model overfit to specific model





OLS ERROR:

TRAINING SET	VALID. SET	TEST SET
↑	↑	↑
↓	↑	↑
↓	↓	↑
↑	↓	↓
↑	↑	↓

STRUCTURAL: H can't rep. hypo. that performs well on test data
ESTIMATION: θ can't be est. well based on training data

hypothesis class not expressive enough → high structural error
overfitting → need more training data → high estimation error
testing + validation from different distributions

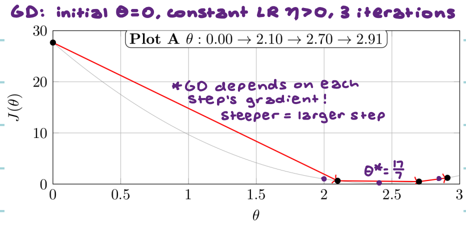
testing + validation from different distributions
hypothesis class not expressive enough → high structural error

ACTIVATION FUNCTIONS:
- linear
- sigmoid
- softmax
- ReLU

LOSS FUNCS:
- MSE, MAE — associated: linear
- NLL (sigmoid) / cross entr.
- NLLM (softmax)

→ MSE good for continuous real # output, NLL and/or softm for discrete

EX: GD
GD: initial θ=0, constant LR η>0, 3 iterations



ii. (6 points) Suppose we run GD with initial parameter θ = 2 and a constant learning rate η > 0 for one iteration. Let's call the initial parameter value θ_{old} (so θ_{old} = 2) and the updated parameter value θ_{new}. What is the range of η such that $J(\theta_{new}) \le J(\theta_{old})$? You can use the fact that the optimal θ* is $\frac{17}{7}$ if needed. & $h(x;\theta) = \theta x$

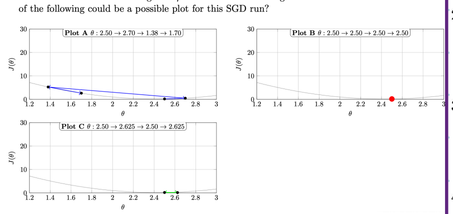
$\mathcal{D}_{train} = \{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),(x^{(3)},y^{(3)})\} = \{(1,3),(2,5),(3,7)\}$

MSE loss: $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(h(x_i) - y_i)^2 = \frac{1}{3}[(\theta-3)^2 + (2\theta-5)^2 + (3\theta-7)^2] = \frac{14}{3}\theta^2 - \frac{68}{3}\theta + \frac{83}{3}$

gradient $\nabla J(\theta) = \frac{28}{3}\theta - \frac{68}{3}$
@ $\theta_{old} = 2$: $\nabla J(2) = -4$
$\theta_{new} = 2 - \eta(-4) = 2 + 4\eta$
$\frac{20}{7} \ge 2 + 4\eta$ ∴ $0 \le \eta \le \frac{3}{14}$
optimal θ* = 17/7, we are @ 2, symm pt = 20/7, set θnew = 20/7

SGD:

iv. (7 points) We run stochastic gradient descent (SGD) to minimize J(θ) with initial parameter θ = 2.5 and a constant learning rate η = 0.125. We run the algorithm for 3 iterations. Which of the following could be a possible plot for this SGD run?



Solution:
Plot B and Plot C are possible.
For Plot B: We keep sampling the same second data point for 3 times.
For Plot C:

Setup:

We have training data
$$\mathcal{D}_{train} = \{(1,3),(2,5),(3,7)\},$$
and a single-parameter model $h_\theta(x) = \theta x$. The squared loss on a point (x, y) is
$$\ell(\theta;x,y) = (\theta x - y)^2,$$
with gradient
$$\nabla_\theta\ell(\theta;x,y) = 2x(\theta x - y).$$
Thus the SGD update is
$$\theta \leftarrow \theta - \eta\cdot 2x(\theta x - y), \qquad \eta = 0.125.$$

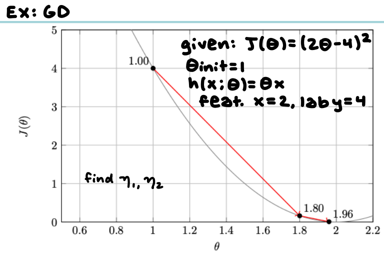
Step 1: θ = 2.5, sample (1, 3) : ∇ = 2·1(2.5 − 3) = −1, θ′ = 2.5 − 0.125(−1) = 2.625

Step 2: θ = 2.625, sample (2, 5) : ∇ = 2·2(2.5 − 5) = 1, θ′ = 2.625 − 0.125(1) = 2.5

Step 3: θ = 2.5, sample (1, 3) : ∇ = 2·1(2.5 − 3) = −1, θ′ = 2.5 − 0.125(−1) = 2.625

Oscillation: 2.5 → 2.625 → 2.5 → 2.625 → ... by alternating between (1, 3) and (2, 5).

EX: GD



$\nabla J(\theta) = 4(2\theta - 4)$
$\theta^t = \theta^{t-1} - \eta\nabla_\theta J(\theta^{t-1})$
$1.8 = 1 - \eta(-8) \to \eta = 0.1$
$1.96 = 1.8 - \eta(-1.6)$
$\eta = 0.1$

DERIVATIVES:
f(θ): ∇θf:
$a^T\theta \to \nabla_\theta f = a$ b/c $\frac{\partial}{\partial\theta_i} = a_i$
$\theta^T\theta \to \nabla_\theta f = 2\theta$ b/c $\theta^T\theta = \sum_i\theta_i^2$ & $\frac{\partial}{\partial\theta_i}(\theta_i^2) = 2\theta_i$
$\theta^TA\theta \to \nabla_\theta f = 2A\theta$
$\theta^Tx \to \nabla_\theta f = x$
$(\theta^Tx - y)^2 \to \nabla_\theta f = 2(\theta^Tx - y)x$
$\sigma(z) = \frac{1}{1+e^{-z}} \to \sigma'(z) = \sigma(z)(1-\sigma(z))$

[VALIDATION] with single validation set
1. Select θ given λ ∈ Λ: (train) — model params
→ finding unique θ^*_λ for every single hyperparam. λ
→ different model for each λ
→ reg to prevent overfitting
$$\theta^*_\lambda = \arg\min_\theta \frac{1}{|D_{train}|}\sum_{(x,y)\in D_{train}}\mathcal{L}(h(x;\theta),y) + \lambda\|\theta\|$$
(validation)
2. Select λ*: for each θ^*_λ, calculate error & find lowest, λ*
→ no reg., measuring prediction error
→ not optimizing θ anymore
→ choosing best λ
$$\lambda^* = \arg\min_\lambda \frac{1}{|D_{val}|}\sum_{(x,y)\in D_{val}}\mathcal{L}(h(x;\theta^*_\lambda),y) + 0$$
(retrain final model)
3. Selection θ*: train one last time with winner λ* on all Dtrain ∪ Dval — final params
$$\theta^* = \arg\min_\theta \frac{1}{|D_{train}\cup D_{val}|}\sum_{(x,y)\in D_{train}\cup D_{val}}\mathcal{L}(h(x;\theta),y) + \lambda^*\|\theta\|$$
4. Evaluate θ*: on test set!
→ no λ bc only a training tool; we want to measure true generaliz.
$$\mathcal{E}_{test} = \frac{1}{|D_{test}|}\sum_{(x,y)\in D_{test}}\mathcal{L}(h(x;\theta^*),y) + 0$$

Step	Goal	Opt. Variable	Dataset	Predictor	Regularization
Find θ^*_λ	θ^*_λ	θ	D_{train}	θ	$\lambda\|\theta\|$
Find λ*	λ*	λ	D_{val}	θ^*_λ	none
Find θ*	θ*	θ	$D_{train}\cup D_{val}$	θ	$\lambda^*\|\theta\|$
Evaluate θ*	$\mathcal{E}_{test}$	N/A	D_{test}	θ*	none

vs. [cross validation] used when:
- dataset small
- more validation checks
- averaging error helps us reduce noise in our validation MSE
- gives us sense of spread/variance

EX: derivative
The ridge objective function is: $J(\theta) = \frac{1}{3}\|X\theta - Y\|^2 + \lambda\|\theta\|^2$
Expanding: $J(\theta) = \frac{1}{3}(X\theta - Y)^T(X\theta - Y) + \lambda\theta^T\theta$
Taking the gradient: $\nabla J(\theta) = \frac{1}{3}\cdot 2X^T(X\theta - Y) + 2\lambda\theta$

EX: $J(\theta) = \theta^T\theta + \|X\theta - Y\|^2 + \|\theta^TX^TY\|^2$
→ $\nabla_\theta J = 2\theta + 2X^T(X\theta - Y) + 2(\theta^TX^TY)X^TY$

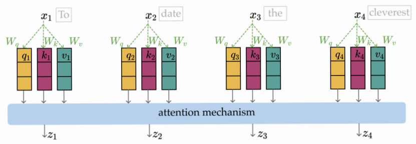
[GD NLL vs. SQUARED LOSS]
Based on these observations, briefly explain why we might prefer one of the loss functions presented above over the other when training a linear logistic classifier.

Solution:
NLL typically has a larger gradient and leads to faster, more reliable convergence to a low-error solution.
If you consider "these observations" from part (a) i. in a vacuum, you may be tempted to conclude that the squared-error loss may be preferable because it avoids the issue of have undefined gradients revolving around division by zero. However, as observed in part iii., we can never get to g = 1 (or g = 0), as the sigmoid only approaches these values asymptotically.

TRANSFORMER ARCHITECTURE:

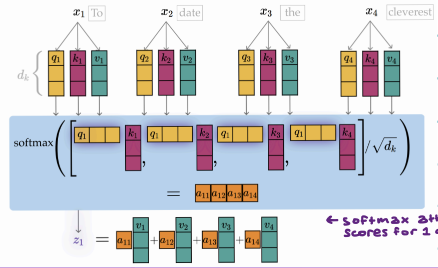
- W_q, W_k, W_v, all in $\mathbb{R}^{d \times d_k}$ ← same 3 matrices for all words in sent.
- project word embedding x from d-dimensional space to d_k-dimensional (q, k, v) spaces (typically $d_k < d$) ← compress info to q,k,v vectors (mult. embed by weight matrices)
- $q_i = W_q^T x_i$, $k_i = W_k^T x_i$, $v_i = W_v^T x_i$, $\forall i$ — *weight sharing* across positions
- *parallel* and *structurally identical* processing

2. Attention mechanism



takes q,k,v vectors for every word & mixes them

- Attention mechanism turns the projected (q, k, v) into z — now, z_i contains how relevant words are to ith word
- Each z is context-aware: a mixture of everyone's values, weighted by relevance



← softmax attention scores for 1 query

EX: linear logistic

(c) Again, assume $d = 3$. Consider a data point with $x = [10, -1, 2]^T$ and $y = +1$. A linear logistic classifier with $\theta = [1, -1, 2]^T$ and $\theta_0 = 0$ correctly predicts the label for this point. Recall that the negative log-likelihood (NLL) loss is:

$$\mathcal{L}(g, y) = -[y \log g + (1 - y) \log(1 - g)],$$

where $g = \sigma(\theta^T x + \theta_0)$. → $g = \sigma(z)$ & $y = +1$ $L = -\log(g)$ {g close to 1: small loss; g close to 0: large loss}

Consider logistic classifiers defined by the following parameters. ↑z, σ→1, sm loss

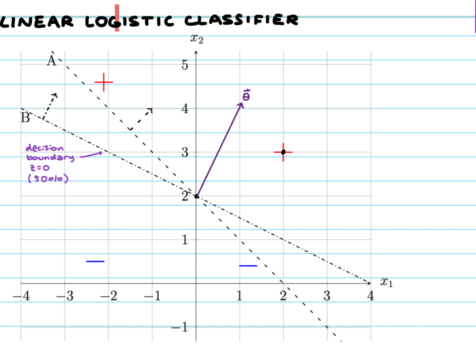
i. Which classifier achieves the lowest NLL loss on this data point?

Solution:
- ○ $\theta = [1, -1, 2]^T, \theta_0 = 0$ → z=15
- ○ $\theta = 10 * [1, -1, 2]^T, \theta_0 = 10$ → z=10·15+10=150
- ✓ $\theta = 20 * [1, -1, 2]^T, \theta_0 = 20$ → z=320
- ○ $\theta = -30 * [1, -1, 2]^T, \theta_0 = -30$ → z=-480

ii. Which classifier achieves the highest NLL loss on this data point?

Solution:
- ○ $\theta = [1, -1, 2]^T, \theta_0 = 0$
- ○ $\theta = 10 * [1, -1, 2]^T, \theta_0 = 10$
- ○ $\theta = 20 * [1, -1, 2]^T, \theta_0 = 20$
- ✓ $\theta = -30 * [1, -1, 2]^T, \theta_0 = -30$

EX: LINEAR LOGISTIC CLASSIFIER



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

A. $\theta = \begin{bmatrix}-2\\-4\end{bmatrix}$, $\theta_0 = 8$ — $z = \theta^T x + \theta_0$; hyperplanes s.t. θ 0 splitting +/-; $0 = [\theta_1\ \theta_2]x + \theta_0$; $x_1 = 0, x_2 = 2$; $x_1 = -2, x_2 = 3$

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

(C.) $\theta = \begin{bmatrix}1\\2\end{bmatrix}$, $\theta_0 = -4$ — * θ vector points in + direction (careful!!) → verify direction

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

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

$z = [1\ 2]\begin{bmatrix}2\\3\end{bmatrix} - 4 = 8 - 4 = 4$; $\sigma(4) = \frac{1}{1+e^{-4}} = 0.98$

EX: SOFTMAX

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

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

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

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

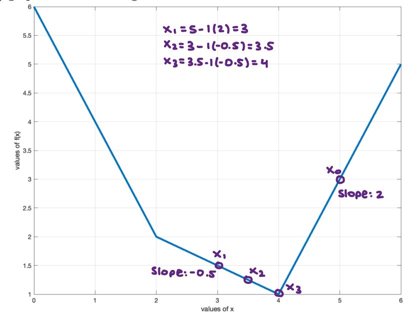
$z = \begin{bmatrix}x_1\\0\end{bmatrix}$ → logit = 1; class 1: $-x_1 - 3$ is largest when $-x_1 - 3 > 0$ → $x_1 \le -3$; $-x_1 - 3 > x_1$ (true if $x_1 \le 3$) (based on graph)

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

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

EX: GD CALCULATION

First, John 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 $x = 4$, John 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 $x^{(3)}$?

Solution: Answer: $x^{(1)} = 3$, $x^{(2)} = 3.5$, $x^{(3)} = 4$. **Reasoning:** by inspection of the graph, $\nabla f(x^{(0)}) = 2$, hence $x^{(1)} = 5 - 1 \cdot 2 = 3$. Next, $\nabla f(x^{(1)}) = -0.5$, hence $x^{(2)} = 3 - 1 \cdot (-0.5) = 3.5$. Finally, $\nabla f(x^{(2)}) = -0.5$, hence $x^{(3)} = 3.5 - 1 \cdot (-0.5) = 4$.

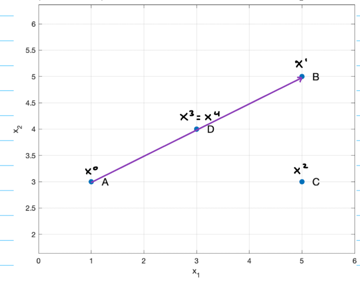
GD OSCILLATION

John 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 $k = 0, 1, 2, \ldots$). Find all such values of η.

Solution: Answer: $\eta \in (1.5, 2.5)$. **Reasoning:** equality $x^{(k+2)} = x^{(k)}$ requires $\nabla f(x^{(k)}) = -\nabla f(x^{(k+1)})$. Since $\nabla f(x^{(0)}) = -2$, we need $\nabla f(x^{(1)}) = 2$, which means $x^{(1)} = x^{(0)} - \eta \nabla f(x^{(0)}) = 1 + 2\eta \in (4, 6)$. Hence $\eta \in (1.5, 2.5)$.

oscillation: $\nabla f(x^{k+1}) = -\nabla f(x^k)$
$\nabla f(x') = -\nabla f(x_0) = +2$ ← x_1 in (4, 6)
$4 < x' = x^0 - \eta \nabla f(x_0) < 6$
$1.5 < \eta < 2.5$

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 $x^{(0)} = A$, $x^{(1)} = B$, $x^{(2)} = C$, $x^{(3)} = D$ shown on the plot below:



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

$x^{k+1} = x^k - \eta \nabla f(x^k)$
$\nabla f(x^k) = \frac{1}{\eta}(x^k - x^{k+1})$
$\nabla f(x^0) = 10(x^0 - x^1) = 10 \cdot \begin{bmatrix}1-5\\3-5\end{bmatrix} = \begin{bmatrix}-40\\-20\end{bmatrix}$
$\nabla f(x_1) = 10(x^1 - x^2) = 10 \cdot \begin{bmatrix}5-5\\5-3\end{bmatrix} = \begin{bmatrix}0\\20\end{bmatrix}$
$\nabla f(x^2) = 10(x_2 - x_3) = 10 \cdot \begin{bmatrix}5-3\\3-4\end{bmatrix} = \begin{bmatrix}20\\-10\end{bmatrix}$
$\nabla f(x^3) = \begin{bmatrix}0\\0\end{bmatrix}$ ← b/c $x_3 = x_4$

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

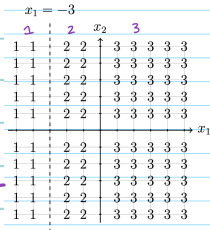
Consider running stochastic gradient descent (SGD) to learn θ. We initialize SGD with $\theta_{\text{initial}} = 4$ and use a fixed learning rate of $\eta = 0.02$. After just one iteration of the GD update, which of the following is a possible value for the resulting updated parameter θ_{new}? Briefly 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: $L(\theta) = (y - \theta x)^2$ → $\nabla L_\theta = -2(y - \theta x)x$
$\theta_{init} = 4$; $\theta_{new} = \theta_{old} - \eta \nabla J(\theta_{old})$
for (x, y) = (2,5): $\theta_n = 4 - 0.02[-2(5-4\cdot2)\cdot2] = 3.76$
(1,4): $\theta_n = 4 - 0.02[-2(4-4\cdot1)\cdot1] = 4$
(4,7): $\theta_n = 4 - 0.02[-2(7-4\cdot4)\cdot4] = 2.56$

EX: SOFTMAX

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: $\theta_1\ \theta_2\ \theta_0$

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

$g_1(x) = -1x_1 + 0x_2 - 3$; $g_2(x) = 0$; $g_3(x) = 1x_1$ — classification depends on x_1!

BINARY & MULTI-CLASS CLASSIFICATION

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

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

(a) How many independent values are needed to specify a probability distribution over K classes? How many outputs does softmax actually produce? How do we reconcile this difference? Does binary classification encounter the same issue?

Solution: A probability distribution over K classes must sum to 1, so only $K - 1$ values are required to fully specify it.

Softmax produces K outputs, which introduces one redundant degree of freedom.

In binary classification with $K = 2$, this issue does not typically arise because we use a single logit instead of two. However, we **could** formulate binary classification with two logits, in which case the same redundancy would appear.

We use K outputs because: (1) it provides a symmetric representation across all classes, and (2) it matches the K-dimensional one-hot encoding of class labels, making the loss computation straightforward. We could eliminate the redundancy by fixing one logit (e.g., $z_K = 0$), but the symmetric formulation is simpler in practice.

(b) Now consider $K = 2$. Suppose you train both a binary classifier (parameter $\theta \in \mathbb{R}^d$) and a 2-class softmax classifier (parameters $\theta_1, \theta_2 \in \mathbb{R}^d$) on the same dataset. After training, the two classifiers produced the same decision boundary. Derive how θ relates to θ_1 and θ_2.

Note: Assume class 1 in the multi-class setting corresponds to class 1 (positive) in binary.

Hint: Write p_1 in the multi-class setting and rewrite it to match $p = \sigma(z)$.

Solution: In the multi-class setting, the probability of class 1 is

$$p_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2}}.$$

Divide both the numerator and denominator by e^{z_1}:

$$p_1 = \frac{1}{1 + e^{z_2 - z_1}}.$$

Define $z = z_1 - z_2$. Then

$$p_1 = \frac{1}{1 + e^{-z}} = \sigma(z),$$

which is exactly the binary logistic function.

This shows that both classifiers produce the **same decision boundary**: we predict class 1 when $p_1 > 0.5$, i.e., when $z > 0$, which is equivalent to $z_1 > z_2$. The classification depends only on the relative score between classes.

Therefore, we have

$$z = z_1 - z_2 = \theta_1^\top x - \theta_2^\top x = (\theta_1 - \theta_2)^\top x.$$

Comparing with the binary form $z = \theta^\top x$, we get

$$\theta = \theta_1 - \theta_2.$$

CNN DIMS:

EX: 32×32×3 RGB input, Conv layer: filter 5×5, 10 filters, stride 1 pad 0.
TOTAL PARAMS: each filter 5×5×3 → 75 +1 (bias) weights per filter. 10 filters → 76·10 = 760 params
FULLY CONN.: params = (input dim · output dim) + output dim

CNN DIMS:

Layer	Input Dimensions	Output Dimension
3 × 3 × 3 ← filter size convolutional layer with 9 filters, stride 2, with zero padding	32×32×3 — $n^2 = \left\lceil \frac{n^{k-1} + 2p^k - (k^k - 1)}{s^k} \right\rceil = \left\lceil \frac{32 + 2(1) - (3-1)}{2} \right\rceil = 16$	16×16×9 — 9 filters = 9 output channels
ReLU activation — never changes dim.	16×16×9	16×16×9
4 × 4 max pooling, stride 4 — $n^2 = \frac{n^{k-1} - k}{s} + 1 = \frac{16-4}{4} + 1 = 4$	16×16×9	4 × 4 × 9 — # chan never changes in pooling
Flattening layer	4 × 4 × 9	144 × 1
Fully connected layer with 8 neurons	144 × 1	8 neurons = 8 outputs — 8 × 1
Softmax activation — never changes dim.	8 × 1	8 × 1

How many weight parameters are learned in the first convolutional layer?

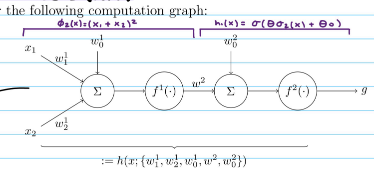
Solution: 9 filters of size 3 × 3 × 3, each with a bias, leads to 252 weight parameters.

Given the size of the network output, how many image categories is this network designed to classify?

Solution: 8 neurons ⟶ 8 classes.

EX: HYPOTHESIS (NN)

(f) Consider the following computation graph:



$:= h(x; \{w_1^1, w_2^1, w_0^1, w^2, w_0^2\})$

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).

→ $g = f^2(w^2 f^1(x_1 w_1^1 + x_2 w_2^1 + w_0^1) + w_0^2)$
$\phi_2(\cdot) = x_1^2 + 2x_1x_2 + x_2^2 = (x_1 + x_2)^2$
$f_2^1 = \sigma$; $f_1(x) = x^2$
$w^2 = \theta$; $w_1^1 = 1$; $w_0^1 = 0$
$w_0^2 = \theta_0$; $w_2^1 = 1$ — coeff. for 1st feature in 1st layer

FEATURE ENCODING:

Feature	Encoding
Age to model likelihood of Alzheimer	(Age)^2 term for quadratic trends
Age to predict salary	Divide into age bins
Blood type (ex: A+, B-, AB+, O-)	One hot A, one hot B, and one hot +/-
Time of day to predict electricity demand	Sin(t/24*2pi) and cos(t/24*2pi)
Text to measure semantic	Pre-trained text embedding model ;)

FORWARD PASS:

6.5.1 Notation and Setup

We consider supervised learning with training data $\mathcal{D}_n = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$.

For a single example (x, y), the forward pass through an L-layer network is: pre-activation $Z^\ell = (W^\ell)^\top A^{\ell-1} + W_0^\ell$, post-activation $A^\ell = f^\ell(Z^\ell)$, $\ell = 1, \ldots, L$, with $A^0 = x$ as the input and $g = A^L$ as the network's output. (guess)

If layer ℓ has d_ℓ units, then: $A^\ell, Z^\ell \in \mathbb{R}^{d_\ell \times 1}$, $W^\ell \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$, $W_0^\ell \in \mathbb{R}^{d_\ell \times 1}$ (compare to loss) — 1 bias per neuron

A and Z have the same dimension!

TRAINING LOOP:

Initialize all weights $W^1, W^2, \ldots, W^L$ randomly

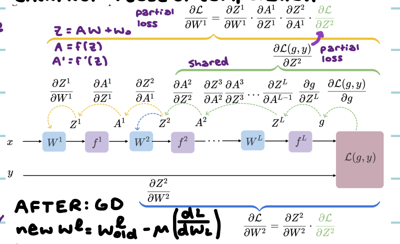
Repeat until stopping criterion:

1. Forward pass: for each data point, compute $Z^1, A^1, Z^2, A^2, \ldots, g^{(i)}$ — evaluate
2. Evaluate loss: for each data point, compute $\mathcal{L}(g, y)$
3. Backward pass: pick a data point, compute $\nabla_{W^\ell}\mathcal{L}(g^{(i)}, y^{(i)})$ for all $\ell = L, L-1, \ldots, 1$ via the chain rule (reuse intermediate results → backpropagation) — to get vals & learn params
4. Update: $W^\ell \leftarrow W^\ell - \eta \nabla_{W^\ell}\mathcal{L}$ for all ℓ

*not guaranteed to be optimal

BACKPROP: reuse of computation

$Z = AW + W_0$, $A = f(Z)$, $A' = f'(Z)$ — partial loss $\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1}\frac{\partial A^1}{\partial Z^1}\frac{\partial Z^2}{\partial A^1}\cdots\frac{\partial \mathcal{L}}{\partial Z^2}$, shared $\frac{\partial \mathcal{L}(g,y)}{\partial Z^2}$ partial loss



AFTER: GD

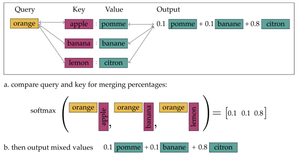
new W_ℓ: $W_{new} = W_{old} - \eta\left(\frac{\partial L}{\partial W_\ell}\right)$

→ answers: how much did each weight contribute to that wrongness

$\frac{\partial \mathcal{L}}{\partial W^2} = \frac{\partial Z^2}{\partial W^2}\cdot\frac{\partial \mathcal{L}}{\partial Z^2}$

considerations:

- "dead" ReLU neurons: if gradient = 0 then ReLU stops learning
- vanish/exploding gradients if multiply many small/large factors across layers



LINEAR ALGEBRA TIPS

- Matrix Derivatives:
  - $\nabla_v(Av) = A^T$
  - $\nabla_v(v^T A) = A$
  - $\nabla_v(v^T Av) = (A + A^T)v$
- Dot Product & Similarity:
  - $a \cdot b = ||a|| \cdot ||b|| \cos(\theta)$. Measures alignment.
  - Used in attention (QK^T) to find "relevant" tokens.

*REPRESENTATION

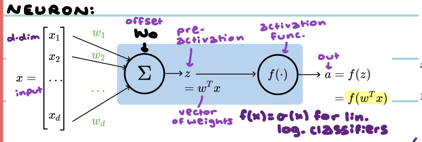
wk 1-9

NEURAL NETWORK:

- nonlinear feature transformation] expressive
- composing simple nonlinearities (layers)
- backpropagation (efficient learning)

layer index; W source neuron index, target neuron index

NEURON:



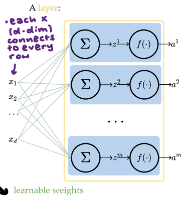
$f(x) = \sigma(x)$ for lin. log. classifiers; $f(x) = x$ for lin. reg.

- x: d-dimensional input
- w: weights (i.e. parameters) — w: what the algorithm learns
- z: pre-activation output (scalar) — z: scalar
- f: activation function (scalar in scalar out) — f: what we engineers choose
- a: post-activation output — a: scalar

*no learnable params:
- max pooling
- ReLU
- sigmoid
- tanh

task	regression	binary classification	multi-class classification
f^L	linear	sigmoid	softmax
loss	squared	NLL	NLLM

LAYER:



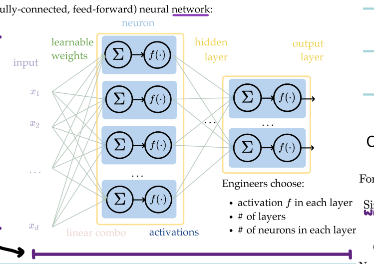
- (# of neurons) = layer's output dimension.
- activations applied element-wise, e.g. z^1 won't affect a^2. (softmax in the output layer is a common exception)
- all neurons in a layer typically share the same f (mixed f is possible but complicates the math).
- layers are typically fully connected, each input x_i influences every a^j.

each x (d-dim) connects to every row

network: multiple neurons achieves complex behavior by alternating b/t param. lin. trans. & fixed non-lin. trans.

fully conn: all inputs connected to every unit in a layer (all x affect all A)

A (fully-connected, feed-forward) neural network:



width = # neurons in layer

depth: # of layers

increasing width/depth (w/ non-lin. activation) makes model more expressive but incr. risk of overfitting!

Engineers choose:
- activation f in each layer
- # of layers
- # of neurons in each layer

ACTIVATION FUNCTIONS:

All activation functions except softmax are applied element-wise (row-wise) ← every elt. same func.

$\text{step}(z) = \begin{cases} 0 & z \le 0 \\ 1 & z > 0 \end{cases}$

elt-wise (doesn't change dims.)

$\text{ReLU}(z) = \begin{cases} 0 & z \le 0 \\ z & z > 0 \end{cases} = \max(z, 0)$ — default in hidden layers

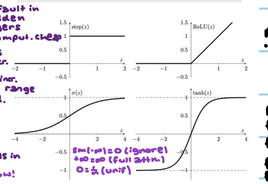
con: "dead" if val = neg; PRO: avoid vanishing grad, comput. cheap

$\sigma(z) = \frac{1}{1+e^{-z}} \in (0,1)$ — bin. class; nonlin, smooth, out in (0,1), monot. incr.

$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \in (-1,1)$ — monot. incr., restrict range; out: (-1,1); con: vanish. grad.

$\text{softmax}(z) = \begin{bmatrix} \exp(z_1)/\sum_i \exp(z_i) \\ \vdots \\ \exp(z_n)/\sum_i \exp(z_i) \end{bmatrix}$

$\text{softmax}(z)_i \in (0,1)$, $\sum_{i=1}^n \text{softmax}(z)_i = 1$ — in: vector, out: all vals in & add to 1, denom = row!



sml = 0 (ignore); lrg = o (full attn.); 0:1 (unif)

Loss	f^L	task
squared	linear	regression
NLL	sigmoid	binary classification
NLLM	softmax	multi-class classification

NLL avg. |nxd(x)| multiple classification

ATTENTION MATRIX: $\frac{1}{\sqrt{d_k}}\begin{bmatrix} q_1^\top k_1 & q_1^\top k_2 & \cdots & q_1^\top k_n \\ q_2^\top k_1 & q_2^\top k_2 & \cdots & q_2^\top k_n \\ \vdots & \vdots & \ddots & \vdots \\ q_n^\top k_1 & q_n^\top k_2 & \cdots & q_n^\top k_n \end{bmatrix} = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$ — # Z is same dim

ELT-WISE:
- output only depends on it's own inp.

ROW-WISE:
- output is applied to whole row as a whole

RELU DERIV:
$f' = 1$, $x > 0$; $f' = 0$, $x \le 0$; $f = x$, $x > 0$; $f = 0$, $x \le 0$

CNN: exploits:
- visual hierarchy
- spatial locality (zoom in & compare)
- translational invar. (recognize despite diff.)

CNN LAYER TYPES:
- CONVOLUTIONAL: detect low-level feats. (edges, color, contrast). later, for mid-level feats. (ears, eyes) (filters)
- FULLY CONNECTED: allow combining feats. from entire image & provide final network out
- MAX POOLING: downsampling technique. detect strongest response within given window & allows network to be less sensitive to feat. locations
  - → no weights/params. $n^\ell = \frac{n^{\ell-1} - k}{s} + 1$
  - → filter size k > stride to not skip anything
  - → typically stride 1 s.t. out img < input
  - if stride > 1, out may not be trans. invar.

Convolution dimension (CNN)

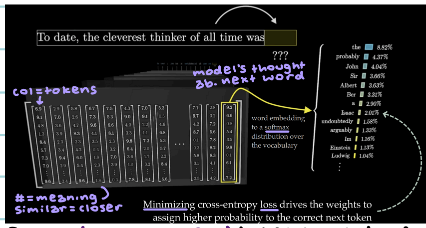
For some layer l — kernel size; # input channels

- Size of one filter: $k^l \times k^l \times m^{l-1}$ (weights per filter)
- Input tensor: $n^{l-1} \times n^{l-1} \times m^{l-1}$
- Output tensor: $n^l \times n^l \times m^l$
- Number of filters: m^l (= # output channels)
- Stride: s^l
- Padding: p^l

- Start with n^{l-1} from the last layer
- Padding added on both sides, so add $2p^l$
- Filter receptive field take away $k^l - 1$
- Stride skip pixels, thus dividing by s^l

$$n^l = \left\lfloor \frac{n^{l-1} + 2p^l - (k^l - 1)}{s^l} \right\rfloor$$

size of layer ℓ's output



FILTER: (conv. kernels) takes in window & detects presence of pattern in that data

↳ take dot prod. of window & filter; higher dot prod = more likely pattern

WORD EMBEDDING: embed words → vectors (words closer semantically will be closer together)

- scrape internet for plain text
- create own "labels" (prediction targets) from these texts
- ex: "cat" & dog close together in vector space (learned) post train
- convert "unsupervised" problem into "supervised" setup
- (query, key, value) w/ sensible dot prod. similarity
- cosine similarity: the larger, the more similar 2 words are

CONTRASTIVE LEARNING: similar data points brought close together
- crop image s.t. patches from same image are encoded close together
- similar to word embedding but for images

MULTI-MODALITY: i.e. predict sound of sea based on image of sea

TRANSFORMER:
- input embedding: n small chunks, each of which w/ a mapping
- pass through attention layer (values change internally) then multi-layer perception
- word embed to a softmax dist. over vocab - want correct word to be assigned high probabilities

"what i am looking for" — QUERY VECTORS: $q_i = W_q^T x_i$ ← transform token x_i to query vec. where q_i = what is token i looking for

"what i contain" — KEY VECTORS: $k_i = W_k^T x_i$

ATTENTION: $a_i = \text{softmax}([q_i^T k_1, q_i^T k_2, \ldots, q_i^T k_n]/\sqrt{d_k})^T$ — how similar queries & keys are; weights W_q, W_k, W_v are fixed in attn. layer

"what i actually give out/contrib." — VALUE VECTORS: $v_i = W_v^T x_i$ ← extracts meaningful info from token x_i

ATTENTION OUTPUT: $z_i = \sum_{j=1}^n a_{ij} v_j \in \mathbb{R}^{d_k}$ ← z_i = x_i's enriched embedding weighted avg. of preceding tokens by attention

Ex: x_i → harry, w/ context of hogwarts, z_i := harry potter

a_{ij}: how much attn. token 1 pays token 2

- $X = n \times d$, ith row is x_i (tokens)
- $Q = n \times d_k$, ith row is q_i (query) $Q = XW_q$
- K, V also $n \times d_k$ $K = XW_k$, $V = XW_v$
- $A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ — weight matrices: $d \times d_k$

d_k prevents dot prod. from growing too large in mag.

$Z = AV$: $\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

MULTI-HEADED: many attention heads each w/ independently learned {Q, K, V} set, H heads, run in parallel (each head has different "perspectives" on sentence)

$Q^h = XW_q^h$, $K^h = XW_k^h$, $V^h = XW_v^h$, $Z^h = \text{Attention}(Q^h, K^h, V^h)$

$\text{Multihead}(X) = \text{Concat}(Z^1, \ldots, Z^H)W^O$ → W^O learned, condenses result from $n \times (H \cdot d_k)$ to $n \times d$

→ each row is one token z_i but w/ perspective of many heads

POSITIONAL EMBEDDING: $\tilde{x}_i = x_i + p_i$ (plug $\tilde{x}_i$ into attention)

→ attention has no sense of word order - just sees tokens

- LEARNED: p_i is row in learned lookup table, finds best value through training

SINUSOIDAL: p_i found mathematically, plug in i & calc. vector

MASKS: limit tokens to looking at their preceding tokens

$M = \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ \vdots & & \ddots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$ $A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)$

TRAINING METHOD: ("have the key")

SUPERVISED LEARN: input x & correct ans. Y
- labeled data, predict new, unseen data
- often used for classification

UNSUPERVISED: self discovery
- unlabeled data, describe/group data
- often used for clustering, autoenc, asso. finding patterns

↳ SELF-SUPERVISED: subset of unsupervised
- generate your own labels from data
- create prediction task (next word, patch)

AUTO-ENCODER: "info squeezer"

$L(x, \hat{x}) = \min ||x - \hat{x}||^2$, $\hat{x} = NN(x, w)$

x → encoder → decoder → reconstruct (bottleneck) $g(x) \to z \to f(z)$

TRAINING:
- parallelizable
- training loop (fwd pass, backprop, upd. weights)
- masks & dropout

INFERENCE:
- sequential
- just fwd pass, no backprop.
- dropout off (see predictions)

SELF-ATTENTION: order ignorant; in self-attention layer, every path looks @ every other patch to calc. score of relevance (look @ own sequence)
- often for image recog/classification ↳ what transf. use

REPRESENTATION LEARNING: learn useful feats./patterns from data (goal) that both sup/unsup. can do

autoencoder (unsup): learn compressed rep.

word embed. (unsup): learn vect. rep of words

- CNNs (superv.): learn visual rep (edge/shape)

n	num tokens	2
d	input token dim	4
d_k	(qkv) embedding dim	3
H	num heads	5

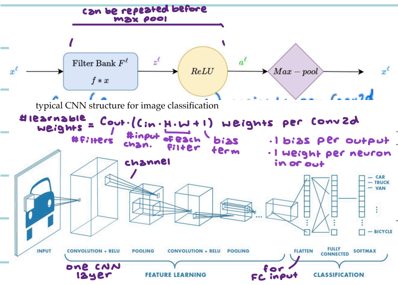
d = full embedding dim. d_k = smaller dim. project down to for a single attention head (computations)

W_q^h	query proj	$d \times d_k$	4 × 3
W_k^h	key proj	$d \times d_k$	4 × 3
W_v^h	value proj	$d \times d_k$	4 × 3
W^o	output proj	$Hd_k \times d$	15 × 4
X	input	$n \times d$	2 × 4
Q^h	query	$n \times d_k$	2 × 3
K^h	key	$n \times d_k$	2 × 3
V^h	value	$n \times d_k$	2 × 3
A^h	attn matrix	$n \times n$	2 × 2
Z^h	attn head out	$n \times d_k$	2 × 3
concat$(Z^1 \ldots Z^H)$	multi-head out	$n \times Hd_k$	2 × 15
Z_{out}	attn layer out	$n \times d$	2 × 4

learned weight matrices (W_q^h, W_k^h, W_v^h, W^o)

*within the attention mechanism

CNN ARCHITECTURE:



typical CNN structure for image classification — can be repeated before max pool

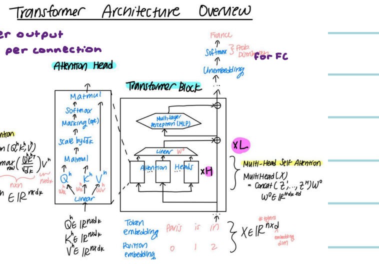
#learnable weights = Cout·(Cin·H·W + 1) weights per conv2d (#filters, #input chan., of each filter, bias term)

- 1 bias per output
- 1 weight per neuron in or out

RESIDUAL CONNECTION:
- adds I term to Jacobian, enabling us to train models
- fixes dead neuron issue

BIAS: 1 per output; WEIGHT: 1 per connection

Transformer Architecture Overview



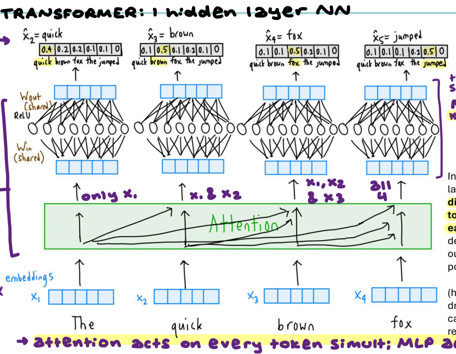
Operation	Input Shape	Output Shape
Token Embedding	n tokens	$n \times d$
Add Positional Encoding	$n \times d$	$n \times d_k$
Transformer Block	$n \times d$	$n \times d$
Layer Norm	$n \times d$	$n \times d$
Unembedding	$n \times d$	$n \times V$
Softmax	$n \times V$	$n \times V$

V = vocab size = # possible output choices

SPATIAL LOCALITY: knowing which pixels are close in space w/ each other

TRANSLATION INVARIANCE: treat patterns as same even if translate them in space (still extract useful info when obj. shifts), but loss of spatial location

TRANSFORMER: 1 hidden layer NN



probability model assigns to the next word; in: the, out: quick

every word (token) looks @ every other word & decides how important they are. — transformer

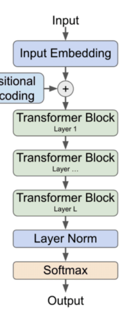
think within single token — MLP; *acts on each token one @ time

simult. calc. In Attention layers, different tokens talk to each other to determine the output at each position (here I've drawn causal/autoreg ressive)

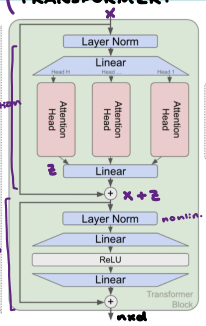
→ attention acts on every token simult; MLP act on 1 @ a time

LAYER NORM: stabil training & accel. conv. by scaling outputs of sub-layers (attn/fwd) to have mean=0, sd=1

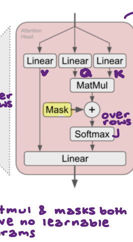
MODEL:



TRANSFORMER:



ATTEN. HEAD:



matmul & masks both have no learnable params

Each row is a query; Each col is a key; Each row sums to 1

X	$n \times d$
W_q, W_k, W_v	$d \times d_k$
$Q = XW_q$	$n \times d_k$
$K = XW_k$	$n \times d_k$
$V = XW_v$	$n \times d_k$
$A = \text{softmax}_{row}(\text{mask} + \frac{QK^\top}{\sqrt{d_k}})$	$n \times n$
$Z = AV$	$n \times d_k$

Operation	Input Shape	Output Shape
Layer Norm 1	$n \times d$	$n \times d$
Attention Head	$n \times d$	$n \times d_k$
Attention Head Concatenation	$H \times (n \times d_k)$	$n \times Hd_k$
Attention Output Projection	$n \times Hd_k$	$n \times d$
Layer Norm 2	$n \times d$	$n \times d$
Fully Connected Layers	$n \times d$	$n \times d$

A & Z always same dims!

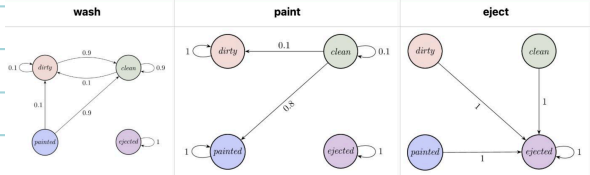
Due to the residual connections

WK 1-12

REINFOR. LEARN:
- agent interacts w/ environment
- takes actions, get rewards
- maximize long-term rewards
- transitions & rew. are UNKNOWN!

MARKOV DECISION PROCESS: (S, A, T, R, γ)
- S: state space (current situation)
- A: action space (possible actions)
- T(S,a,S'): probability of S→S' taking a
- R(S,a): reward for S taking a
- γ ∈ [0,1]: discount factor. how much you care for future rewards (γ=0 → immediate, γ=1 → long-term)
- π(S): policy (strategy). deterministic π(a) or probabilistic π(S|a)

MDP:



Value Iteration($\mathcal{S}, \mathcal{A}$, T, R, γ, ϵ)

assumes you know:
- transition probabilities T(s,a,s')
- reward func. R(s,a)

1. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2. $\quad Q_{\text{old}}(s,a) = 0$
3. **while** True:
4. $\quad$ **for** $s \in \mathcal{S}, a \in \mathcal{A}$: Bellman: compute future value
5. $\quad\quad Q_{\text{new}}(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_{\text{old}}(s',a')$
6. $\quad$ **if** $\max_{s,a} |Q_{\text{old}}(s,a) - Q_{\text{new}}(s,a)| < \epsilon$:] converge check
7. $\quad\quad$ **return** Q_{new}
8. $\quad Q_{\text{old}} \leftarrow Q_{\text{new}}$

"calculating"

Q-Learning ($\mathcal{S}, \mathcal{A}, \gamma, \alpha, s_0$, max-iter)

do NOT know T or Rewards (model free), only needs experience (s,a,r,s')

1. $i = 0$
2. **for** $s \in \mathcal{S}, a \in \mathcal{A}$: · can converge to true Q^*_∞
3. $\quad Q_{\text{old}}(s,a) = 0$ · γ, α, ε all b/t 0 & 1
4. $s \leftarrow s_0$
5. **while** $i <$ max-iter :
6. $\quad a \leftarrow$ select_action$(s, Q_{\text{old}}(s,a))$ ← ε. greedy
7. $\quad r, s' \leftarrow$ execute(a)
8. $\quad Q_{\text{new}}(s,a) \leftarrow (1-\alpha) Q_{\text{old}}(s,a) + \alpha(r + \gamma \max_{a'} Q_{\text{old}}(s',a'))$
9. $\quad s \leftarrow s'$ old estimate + calculation
10. $\quad i \leftarrow (i+1)$
11. $\quad Q_{\text{old}} \leftarrow Q_{\text{new}}$ α=0 → never learn
12. **return** Q_{new} α=1 → consider most recent info

"learning" (estimating)

Q-Learning Algorithm for Finding Optimal Policy

Q-learning: Begin in state s and then repeat:

1. **Take action a,** selected random with probability ε and greedy with probability 1-ε.
2. **Observe the reward r and next state s'.**
3. **Update Q(s, a)** with learning rate α to merge old belief and new target.

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

← this one is assuming ∞ horizon

4. **Build the policy**

$$\pi_h^*(s) = \arg\max_a Q_h^*(s,a)$$

NON-PARAMETRIC MODELS: no fixed functional form for h. complexity grows with data
→ interpretability, insight, speed, adaptivity

K-NEAREST NEIGHBOR: (supervised)

Training: none. Just memorize the training set.
small k = ↑ var (overfit)
large k = ↑ bias (underfit)

Predicting: for a new x_{new}, take the majority (class) or mean (regression) label of its k nearest neighbors.
→ sensitive to feat. scale b/c uses dist. calc. ← larger dominates

Parameters learned: the entire training set (its features and labels).
becomes params ↗

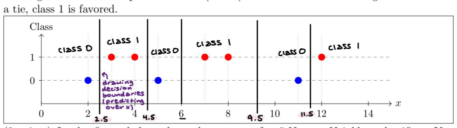
caution when:
- large n or d (dimensionality)
- irrelevant/noisy features
- features on diff. scales

Hyperparameters:
- k: number of neighbors considered.
- distance metric (typically Euclidean or Manhattan).
- tie-breaking scheme (typically at random).

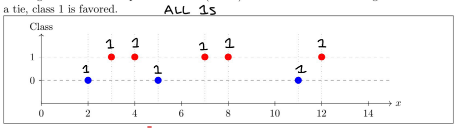
Consider a one-dimensional dataset with only one feature x. The points are in two classes given by the value of $y^{(i)}$.

i	1	2	3	4	5	6	7	8
$x^{(i)}$	2	3	4	5	7	8	11	12
$y^{(i)}$	0	1	1	0	1	1	0	1

i. (2 points) In the figure below, draw the output of a 1-Nearest-Neighbor classifier. Your drawing should show the predicted class (0 or 1) over the entire visible range of x. If there's a tie, class 1 is favored.



ii. (2 points) In the figure below, draw the output of a 5-Nearest-Neighbor classifier. Your drawing should show the predicted class (0 or 1) over the entire visible range of x. If there's a tie, class 1 is favored. ALL 1s



CLASSIFICATION TREE: regression

BuildTree$(I, k, \mathcal{D})$ hyperparam k: largest leaf size

1. **if** $|I| > k$ → stop splitting region if has ≤k data pts
2. $\quad$ **for** each split dim j and split value s
3. $\quad\quad$ Set $I_{j,s}^+ = \{i \in I \mid x_j^{(i)} \geq s\}$ #pick splits by what minimizes WAE
4. $\quad\quad$ Set $I_{j,s}^- = \{i \in I \mid x_j^{(i)} < s\}$
5. $\quad\quad$ Set $\hat{y}_{j,s}^+ = \text{average}_{i \in I_{j,s}^+} y^{(i)}$ · decision trees extend nicely to multi-class
6. $\quad\quad$ Set $\hat{y}_{j,s}^- = \text{average}_{i \in I_{j,s}^-} y^{(i)}$
7. $\quad\quad$ Set $E_{j,s} = \sum_{i \in I_{j,s}^+} (y^{(i)} - \hat{y}_{j,s}^+)^2 + \sum_{i \in I_{j,s}^-} (y^{(i)} - \hat{y}_{j,s}^-)^2$
8. $\quad$ Set $(j^*, s^*) = \arg\min_{j,s} E_{j,s}$ · not affected by diff. feat. scales. standardization shifts/scales but no effect on struct/preds.
9. **else**
10. $\quad$ Set $\hat{y} = \text{average}_{i \in I} y^{(i)}$
11. $\quad$ **return** Leaf(leave_value=$\hat{y}$)
12. **return** Node $(j^*, s^*, \text{BuildTree}(I_{j^*,s^*}^-, k), \text{BuildTree}(I_{j^*,s^*}^+, k))$

VALUE FUNC: how good is a state if i follow a policy?
→ long term sum calc. expectation

$$V_h^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_{h-1}^\pi(s')$$

current reward — the immediate reward for taking the policy-prescribed action $\pi(s)$ in state s.
expected future reward — $(h-1)$ horizon future value at a next state s'
sum of future values weighted by the probability of reaching that next state s'
discounted by γ

Q-VALUES: expected discounted reward starting from state s with action a then following policy π. Bellman Recursion:

$$V_h^\pi(s) = R(s, \pi_h(s)) + \gamma \sum_{s'} T(s, \pi_h(s), s') V_{h-1}^\pi(s')$$

(do what π says

(action a first, then back to π for future

$$Q_h^\pi(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') V_{h-1}^\pi(s').$$

$V_h^\pi(s) = Q_h^\pi(s, \pi_h(s))$

OPTIMAL POLICY π*: policy that yields highest value V_h^* from every state (may not be unique)

$$V_h^*(s) = \max_a \underbrace{[R(s,a) + \gamma \sum_{s'} T(s,a,s') V_{h-1}^*(s')]}_{Q_h^*(s,a)}$$

Define the optimal state-action value functions $Q_h^*(s,a)$:

the expected sum of discounted rewards, obtained by
- starting in state s
- take action a, for one step
- act **optimally** thereafter for the remaining $(h-1)$ steps

Q^* satisfies:

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

$V_h^*(s) = \max_a Q_h^*(s,a)$ $\quad V_\infty^*(s) = \max_a Q_\infty^*(s,a)$

$Q_\infty^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') V_\infty^*(s')$

$\pi_h^*(s) = \arg\max_a Q_h^*(s,a)$

$\sum_{s'} T(s,a,s') = \sum_{s'} p(s'|s) = 1$ ← trans. prob. 4 stoch.

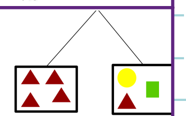
k-Means Clustering

Iterate until convergence or some max iteration
Assign each data point to the cluster of its closest centroid
Update each centroid to be the average of the data points assigned to it
Stop when converged or we've reached max iteration

K-MEANS$(k, \tau, \{x^{(i)}\}_{i=1}^n)$

1. μ, y = random initialization
2. **for** $t = 1$ to τ
3. $\quad y_{\text{old}} = y$
4. $\quad$ **for** $i = 1$ to n
5. $\quad\quad y^{(i)} = \arg\min_j \|x^{(i)} - \mu^{(j)}\|^2$
6. $\quad$ **for** $j = 1$ to k
7. $\quad\quad \mu^{(j)} = \frac{1}{N_j} \sum_{i=1}^n \mathbb{1}(y^{(i)} = j) x^{(i)}$
8. $\quad$ **if** $y == y_{\text{old}}$
9. $\quad\quad$ break
10. **return** μ, y

ENTROPY:



Weighted Average Entropy (WAE)

Set of points in leaf m: $I_m = \{i : x_i \in R_m\}$

Fraction of points in leaf m with label k $\hat{P}_k = \frac{|\{i \in I_m : y_i = k\}|}{|I_m|}$

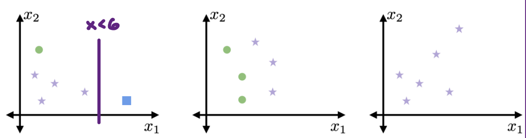
Entropy of leaf m: $H(I_m) = -\sum_{k=1}^K \hat{P}_k \log_2 \hat{P}_k$

WAE of the split: $E(j,s) = \frac{|I^-|}{|I|} H(I^-) + \frac{|I^+|}{|I|} H(I^+)$

entropy $H := -\sum_{\text{class}} \hat{P}_c (\log_2 \hat{P}_c)$ empirical probability

*calc. L & R H, then avg sides

for example: c iterates over 3 classes ★●■



$\hat{P}_c$: ★ $\frac{4}{6}$ ● $\frac{1}{6}$ ■ $\frac{1}{6}$ $\quad$ $\hat{P}_c$: ★ $\frac{3}{6}$ ● $\frac{3}{6}$ ■ 0 $\quad$ $\hat{P}_c$: ★ $\frac{6}{6}$ ● 0 ■ 0

$H = -[\frac{4}{6}\log_2(\frac{4}{6}) + \frac{1}{6}\log_2(\frac{1}{6}) + \frac{1}{6}\log_2(\frac{1}{6})]$ (about 1.252)
$H = -[\frac{3}{6}\log_2(\frac{3}{6}) + \frac{3}{6}\log_2(\frac{3}{6}) + 0]$ (about 1.1)
$H = -[\frac{6}{6}\log_2(\frac{6}{6}) + 0 + 0]$ (= 0)

EX: for split x<6:
H_L x<6 $= -\frac{4}{5}\log_2(\frac{4}{5}) - \frac{1}{5}\log_2(\frac{1}{5})$ #min WAE = all 1 class in leaf
H_R x>6 $= -1\log_2(1)$
WAE: $\frac{5}{6}(H_L) + \frac{1}{6}(H_R)$

For each one, indicate whether it **can reduce** structural error, estimation error, neither, or both. **ERROR:**

(a) Penalize $\|\theta\|^2$ during training
○ structural error ✓ **estimation error** ○ both ○ neither
(b) Penalize $\|\theta\|^2$ during testing
○ structural error ○ estimation error ○ both ✓ **neither**
(c) Increase the amount of training data
○ structural error ✓ **estimation error** ○ both ○ neither
(d) Increase the order of a fixed polynomial basis
✓ **structural error** ○ estimation error ○ both ○ neither
(e) Decrease the order of a fixed polynomial basis
○ structural error ✓ **estimation error** ○ both ○ neither
(f) Add more layers with linear activation functions to your neural network
○ structural error ○ estimation error ○ both ✓ **neither**
(g) Add more layers with non-linear activation functions to your neural net
✓ **structural error** ○ estimation error ○ both ○ neither
(h) Stop training before training error reaches 0
○ structural error ✓ **estimation error** ○ both ○ neither

For a given hidden representation vector **x**, LayerNorm applies the following transformation for each token: **LAYER NORM FORMULA**

$$\mathbf{y} = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot \gamma + \beta$$

- **μ:** The mean of the vector across the embedding dimensions.
- **σ²:** The variance of the vector across the embedding dimensions.
- **ε:** A small constant (e.g., 10^{-5}) added to prevent division by zero.
- **⊙:** Element-wise multiplication.
- **γ and β:** Learnable parameters for scaling and shifting, which allow the model regain representational capacity. Machine Learning Mastery +2

GEOMETRIC SERIES: $\sum_{i=0}^{\infty} a(r)^{n-1} = \frac{a}{1-r}$

α=0: Q-table never updates

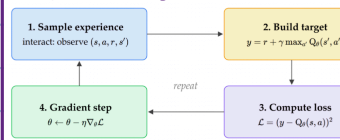
	S_0	S_1	S_2	S_3
a_0	Q			
a_1				

TABULAR Q-LEARNING: store values in table

FITTED Q-LEARNING: approx. Q(s,a) w/ func. $Q_\theta(s,a)$ #some version of grad. descent

$Q_{new}(s,a) = (1-\alpha)Q_{old}(s,a) + \alpha(r + \gamma \max_{a'} Q_{old}(s',a'))$

α∈[0,1] → hyperparam. trading off how much we trust new vs. old beleif (α=1 → trust new experience)
if α=1: does not converge & det. soley by most recent exp.
Q-learn conveges when γ<1, α_t decays correctly, (s,a) visited (like $\alpha = \frac{1}{t}$)



max of the new state vals.
(S0, forw, -10, S1)
-10 + γ max{Q(S1, back), Q(S1, forw)}

ε-GREEDY ACTION SELECTION STRATEGY: ε = exploration rate [0,1]
- with probability ε, choose action a∈A uniformly @ random
- with probability (1-ε), choose $\arg\max_a Q_{old}(s,a)$ (exploit)
- start: ↑ ε (more exploratory), but later ↓ ε (more greedy)
- trades off exploration vs. exploitation

POLICY GRADIENTS: param. π with θ, run π_θ, update θ s.t "good" trajectories are more likely to happen
→ no Q-table needed → learns policy directly

*actions that lead to high reward should become more probable

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot R(\tau)\right]$$

trajectory drawn under the current policy; sum over timesteps
$J(\theta)$: expected total reward (want to max); $\mathbb{E}$: max exp. reward
$\nabla_\theta \log \pi_\theta(a_t \mid s_t)$: how strong policy pref. — push θ to make the taken action **more likely**
$R(\tau)$: total reward of traj. — weight by how good the trajectory was

K-MEANS CLUSTERING: (unsupervised)
structural discovery w/o labels
1. choose k
2. initialize k cluster centers
3. assign each pt to nearest center
4. recompute centers ← based on pts in its cluster
5. repeat

good cluster:
- pts inside same cluster = similar
- compact clusters

minimizes $\sum_{i=1}^{n} \|x_i - \mu_{c_i}\|^2$ (total sq. dist. b/t pts & clusters)
→ sensitive to initialization and k
→ relative scale of features matters b/c use distance — larger vals. dominate
→ return groupings (neither regression nor classification)

CNN EX:

Layer	Input Dimensions	Output Dimension
3 × 3 × 3 convolutional layer with 9 filters, stride 2, with zero padding	32×32×3	16×16×9 Z^1
ReLU activation	16×16×9	16×16×9 A^1
4 × 4 max pooling, stride 4	16×16×9	4 × 4 × 9 A^2
Flattening layer	4 × 4 × 9	144 × 1 A^3
Fully connected layer with 8 neurons	144 × 1	8 × 1 Z^4
Softmax activation	8 × 1	8 × 1 g

#weight params:

Solution: 9 filters of size 3 × 3 × 3, each with a bias, leads to 252 weight parameters.

During training, you are running stochastic gradient descent to update the weights of the network using backpropagation. Suppose that you are using NLL Multiclass Loss $\mathcal{L}$. Derive the backpropagation rule for the weight parameters W^1 of the convolutional layer. Explicitly write the partial derivative factors according to the chain rule using variable names from problem (a).

Solution:

$$\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial Z^1}{\partial W^1} \frac{\partial A^1}{\partial Z^1} \frac{\partial A^2}{\partial A^1} \frac{\partial A^3}{\partial A^2} \frac{\partial Z^4}{\partial A^3} \frac{\partial g}{\partial Z^4} \frac{\partial \mathcal{L}}{\partial g} \quad (1)$$

EX: Q-VALS: tabular

All Q-values are initialized to 0. Assume a learning rate $\alpha = 1$ and discount factor $\gamma = 0.5$.

i. (4 points) We collect the following experience sequence:

Experience	pass 1:	pass 2:
1. $(s_0, \text{Forward}, -10, s_1)$	-10	-10+0.5(40) = 10
2. $(s_1, \text{Forward}, -10, s_2)$	-10	-10+0.5(100) = 40
3. $(s_2, \text{Forward}, 100, s_2)$	100	100+0.5(100) = 150
4. $(s_1, \text{Forward}, -10, s_2)$	-10+0.5(100) = 40	-10+0.5(150) = 65
5. $(s_0, \text{Forward}, -10, s_1)$	-10+0.5(40) = 10	-10+0.5(65) = 22.5

Then, we replay this sequence once more (process all 5 experiences again in the same order). Compute the Q-values after processing all experiences in order. Fill in the updated Q-values

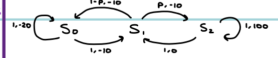
(3 points) The optimal values at some current horizon h are given:

p=0.8
γ=0.9

$$V_h^*(s_0) = -10, \quad V_h^*(s_1) = -10, \quad V_h^*(s_2) = 100.$$

EX: V_{h+1}^* using Qs

Compute $V_{h+1}^*(s_1)$, and identify which action is optimal at s_1 for horizon $h+1$.



$Q(S_1, \text{forward}) = R(S_1, \text{forward}) + \gamma[p \cdot V_h^*(S_2) + (1-p) \cdot V_h^*(S_0)]$
$= -10 + 0.9(0.8 \cdot 100 + 0.2 \cdot -10) = 60.2$

$Q(S_1, \text{back}) = R(S_1, \text{back}) + \gamma[p \cdot V_h^*(S_0) + (1-p) \cdot V_h^*(S_2)]$
$= -10 + 0.9(0.8 \cdot -10 + 0.2 \cdot 100) = 0.8$

$V_{h+1}^*(S_1) = \max(60.2, 0.8) = 60.2$. ∴ optimal action = forward

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 $R_{new}(s,a) = R(s,a) + 10, \forall s \in \mathcal{S}, a \in \mathcal{A}$. **MDP. R+10**

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 $V(s+10) = E[(R_0+10) + \gamma(R_1+10) + \gamma(R_3+10) + \ldots | S]$
$= V(s) + (10 + \gamma 10 + \gamma^2 10 + \ldots) = V(s) + \frac{10}{1-\gamma}$ for all states s