spaces $\mathcal{X}, \mathcal{Y}$

sequence of pairs

$$\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} \mid i \in I\}$$

Remark

$\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ can have $i \neq i' \in I$

$(x_i, y_i) \neq (x_{i'}, y_{i'})$

$\mathcal{X}$ (input) feature space

in first examples $\mathcal{X} \subseteq \mathbb{R}^d$ for some d

$x \in \mathcal{X} \subseteq \mathbb{R}^d$ feature vectors

components $x_i \in \mathbb{R}$ features ~ observations

The point of deep learning (DL) is to iteratively refine feature vectors leading to internal/hidden representations

$\mathcal{Y}$ target space

$y_i \in \mathcal{Y}$ are targets ⟶ task depends on $\mathcal{Y}$

want to predict y from x

- $\mathcal{Y} \in \mathbb{R}^K$ regression → discrete, no structure

- $|\mathcal{Y}| < \infty$ classification

$y \in \mathcal{Y}$ classes / labels

Notation

$n \in \mathbb{Z}_{\geq 0}$, $[n] = \{0, \ldots, n-1\}$

$[n] \hookrightarrow \mathbb{R}^n$

$i \longmapsto (0, 0, \ldots, 0, \underset{i}{1}, 0, \ldots, 0)$ ← i^{th} standard basis vector

↑ 1-hot encoding

Dataset = samples from a distribution

There are random variables X, Y

D samples from joint distribution $X \times Y$

Prediction

Meaning of predictions $y \in \mathcal{Y}$ from $x \in \mathcal{X}$

Predict target values $\mathcal{X} \longrightarrow \mathcal{Y}$

regression

Predict target distribution

approx marginal distribution $\mathcal{Y} | \mathcal{X} = x$

$$f(x) \simeq P(Y = y \mid X = x)$$

or

$$\mathcal{X} \xrightarrow{f} P(\mathcal{Y})$$

Space of probability distribution

Machine Learning $\mathcal{X} \xrightarrow{f} \mathcal{Y}$

$\mathcal{X} \xrightarrow{f} P(\mathcal{Y})$

f parametric model

$f = f_\theta$ ML $\theta \in \mathbb{R}^N$ ← huge

AI assistant $N \simeq 10^{11}$

Set θ via training

regression model = affine transformation (linear)

$$f(x) = Wx + b \qquad x \in \mathbb{R}^d \quad y \subseteq \mathbb{R}^k$$

$$N = (d+1)k \qquad W \in \mathbb{R}^{d \times k}, \; b \in \mathbb{R}^k$$

choice of model / architecture is $\mathbb{R}^N \to (x \to y \to \rho(y))$, $\theta \mapsto f_\theta$

initialize $\theta \in \mathbb{R}^N$ randomly

iteratively try to refine it

Given $\theta, \theta' \in \mathbb{R}^N$, which of f_θ $f_{\theta'}$ is better?

Choose loss function error in $\mathbb{R}$

eg Regression Squared Error $(x, y) \in D$

$$\ell_{SE}(\theta; (x,y)) = | f_\theta(x) - y |^2$$

Setup Regression

1 (input) feature space $\mathcal{X} \subseteq \mathbb{R}^d$

set of outcomes of input random variable X $\quad \Omega \xrightarrow{X} \mathcal{X}$, $|\mathcal{D}| = N$

2 target space $\mathcal{Y} \subseteq \mathbb{R}^k$

3 $\mathcal{D} = \{(\underline{x}_i, \underline{y}_i) \in \mathcal{X} \times \mathcal{Y} \mid i \in I\}$ – samples from $\mathcal{X} \times \mathcal{Y}$

Stack vectors in dataset

Get feature matrix $X \in \mathbb{R}^{N \times d}$

i^{th} row $X_i = \underline{x}_i$

target matrix $Y \in \mathbb{R}^{N \times k}$

⚠ vectors from dataset row vectors

1 Convention in data science → dataset's entry = row in table

2 Convinient in python (torch) commands

Dataset Abalone

molluscs 9 features

1 target feature age

7 numerical input features diameter, height, etc

1 categorical feature female F

male M

infant I

$X_{raw} = \mathbb{R}^7 \times \{F, I, M\}$

$\downarrow$

1 hot encode $F \mapsto e_1$, $I \mapsto e_2$, $M \mapsto e_3$

So we have inputs in $\mathbb{R}^{10}$

$d = 10$

want to predict target $(y_i \in \mathcal{Y})$ from $\underline{x}_i \in \mathcal{X}$

model = parametric function $\mathcal{X} \xrightarrow{f_\theta} \mathcal{Y}$ $(\mathbb{P}(\mathcal{Y}))$

Today model = affine transformation

$$\underline{x}_i \longmapsto \underline{x}^T \underline{w} + b \in \mathbb{R}$$

$\uparrow$ $\mathbb{R}^{10}$

$\underline{w} \in \mathbb{R}^{10}$ $b \in \mathbb{R}$. $M = 11$

If using affine transformation,

can add constant 1 feature

$$\underline{x} \longmapsto (\underline{x}, 1) =: \tilde{\underline{x}}$$

$$(\underline{w}, b) =: \tilde{\underline{w}}$$

So,

$$\underline{x}^T \underline{w} + b = \tilde{\underline{x}}^T \tilde{\underline{w}}$$ ← linear transformation

Q Do we need the extra 1 feature

1 = sum of 1-hot vectors of {F, I, M}

So, we can use original 10-dim features,

linear transformation $\underline{x} \longmapsto \underline{x}^T \underline{w}$ $\underline{w} \in \mathbb{R}^{10}$ $M = 10$

What should be θ or $\underline{w}$?

first guess random init

Choose a measurement of prediction error

loss function $\ell(\underline{x}_i, \underline{y}_i ; \theta) \in \mathbb{R}$

depends on choice!

regression Squared Error

$$\ell_{SE}(\underline{x}_i, \underline{y}_i, \theta) = |f_\theta(x_i) - y_i|^2$$

Overall dataset performance

$$\ell(\mathcal{D}; \theta) = \frac{1}{|N|} \sum_{i \in I} \ell(x_i, y_i ; \theta)$$

get mean squared error (MSE)

can the vector of predicted targets for all entries

$X\ \underline{w}$

Recall $\mathcal{D}$ is samples from $\underline{X} \times \underline{Y}$

We actually are interested in performance on unseen data

$\mathbb{E}\, \ell(X \times Y, \theta)$

Want generalization, not memorization

Use 2 independent collections of samples

$\mathcal{D}_{train}$ train set set θ s.t. $\ell(\mathcal{D}_{train}, \theta)$

$\mathcal{D}_{test}$ test set using θ from training,

$\ell(\mathcal{D}_{test}, \theta)$

Have only one dataset $\mathcal{D}$

Split randomly

Choose train-test proportion → (85% - 15%, 90% - 10%)

Abalone $N = 4177 \rightarrow$ 85-15 split

$N_{train} = 3550$

$N_{test} = 627$

$\mathcal{D}_{train}$ X_{train} $\underline{y}_{train}$

Randomness in Computers

Pseudorandom Generators (PRNG)

ML has 2 opposing requirements

1. Reproducible experiments
2. Use random sampling

PRNG

Stateful deterministic objects

Simulate random sampling

Set PRNG state at start of code

seed - usually integer (big, up to $2^{32} - 1$)

How to be super safe

- record code state = save git commit ID
- record package version numbers

Preprocessing Summaries

$X_{train} \in \mathbb{R}^{3550 \times 10}$ $y_{train} \in \mathbb{R}^{3550}$

$X_{test} \in \mathbb{R}^{627 \times 10}$ $y_{test} \in \mathbb{R}^{627}$

Now Want to find $\theta \in \mathbb{R}^{10}$ that minimizes $\ell_{MSE}(\mathcal{D}_{train}, \theta)$

Then generalization error $\ell_{MSE}(\mathcal{D}_{test}, \theta)$

Want $\theta = \underline{w}$ $|X\underline{w} - \underline{y}_{train}|^2$ ← minimal

Find θ minimizing loss using least squares

Classification

$\mathcal{X} \xrightarrow{f_\theta} [c]$

target space is discrete = no structure $\mathcal{Y}$

also a finite set $\mathcal{Y} =: [c] = \{0, \dots, c-1\}$

Metric: Sensible way to tell which model is better.

Not used to set θ, but to understand results

ex) Accuracy = frequency of correct predictions

$$\text{accuracy}(f_\theta) = \frac{1}{|I|} \sum_{(x_i, y_i) \in \infty} \begin{cases} 1 & f_\theta(x_i) = y_i \\ 0 & \text{else} \end{cases}$$

= probability of correct prediction

⚠ only good on balanced datasets

$\longrightarrow$ each $0 \le \ell \le c-1$ has about the same frequency

Why model class distribution?

$\mathcal{X} \xrightarrow{f_\theta} \mathcal{Y}$ $\quad \mathbb{P}(\mathcal{Y}) \leftarrow$ Why prefer to set $\theta \in \mathbb{R}^M$

$\rightarrow \mathbb{P}(\mathcal{Y})$ gives a richer signal

Continuity is also good for differentiation and optimization as a result.

Multinomial Logistic Regression

$\mathcal{Y} = [c] = \{0, \dots, c-1\}$ $\quad 0 \le i < c$

$\mathbb{P}([c]) = p_i = \mathbb{P}_\theta(Y = i \mid X = x_i)$

predicted by model

$\mathbb{P}([c]) \longrightarrow$ categorical distribution

Y $\quad \mathbb{P}(Y = i) = p_i \in [0,1]$

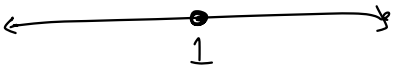
c disjoint events = exactly 1 holds

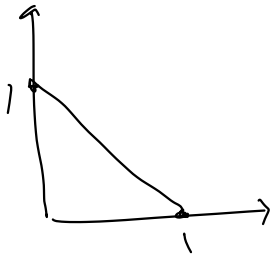
$$\sum_{i=0}^{c-1} p_i = 1$$

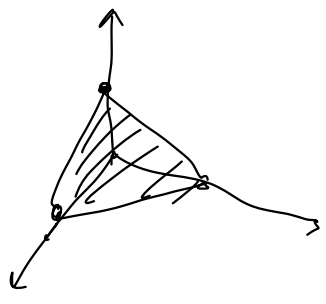
probs sum to 1 (not their squares)

Probability $(c-1)$-simplex

$$\Delta^{c-1} := \left\{ \vec{p} \in \mathbb{R}^c : p_i \in [0,1], \ \sum_{i=0}^{c-1} p_i = 1 \right\} \qquad 0 \le i < c$$

$\Delta^0 =$ 

$\Delta^1 =$ 

$\Delta^2 =$ 

Always have $p_{c-1} = 1 - \sum_{i=0}^{c-2} p_i$

Δ^{c-1} is $(c-1)$-dimensional

To process feature vectors, still want to use affine transformation

not Δ^{c-1}

$$\mathcal{X} \subseteq \mathbb{R}^d \xrightarrow{\underline{x}_i \longmapsto \underline{x}^T W + \underline{b}} \mathbb{R}^c \xrightarrow{\text{post process}} \Delta^c$$

want to freely choose parameters W, b.

we need to do the following

$$\mathbb{R}^c \xrightarrow{\exp} \mathbb{R}^c_{>0} \xrightarrow{\underline{z} \longmapsto \frac{\underline{z}}{\sum \underline{z}}} \Delta^{c-1}$$

$$\underline{z} \longmapsto \frac{\exp(\underline{z})}{\sum \exp(\underline{z})} =: \text{softmax}(\underline{z})$$

$\underline{z}$: logits

softmax($\underline{z}$): probabilities

So we need softmax postprocess $\mathbb{R}^c \longrightarrow \Delta^c$

How are we going to get the $\sim$?

$x_i, y_i \in \mathcal{D}$ want $P_\theta(x_i) \in \Delta^{c-1}$

$$P_\theta(x_i)_j = \begin{cases} 1 & j = y_i \\ 0 & \text{else} \end{cases}$$ ideally if possible

But actually : $P_\theta(\underline{x}) = \mathbb{P}(Y \mid X = x_i)$

Loss Function $\ell(x_i, y_i ; \theta)$ must be small

Gradient Descent : general algorithm to approximately solve this problem

minimize $\mathbb{R}^M \xrightarrow{\theta \longmapsto \ell(\mathcal{D}_{train}; \theta)} \mathbb{R}$

GD:

Requirements:

1) Parameters $\Theta \in \mathbb{R}^M$ — in practice, almost always differentiable

2) $\mathbb{R}^M \xrightarrow{\Theta \longmapsto \ell(\mathcal{D}_{train}; \Theta)} \mathbb{R}$ piecewise differentiable

Step I: Get initial Θ — initialization

Choose distribution to draw from

Step II: Iterate GD steps.

Calculate $\nabla_\Theta \ell(\mathcal{D}_{train}; \Theta) \in \mathbb{R}^M$

$\Theta \leftarrow \Theta - \eta \nabla_\Theta$ — $\eta \in \mathbb{R}_{>0}$ fixed learning rate hyperparameter

When to stop?

↳ Take a fixed # of steps (for now) — also

Multinomial Logistic Regression II:

Recall: dataset : target space $\mathcal{Y} \cong [c] := \{0, \dots, c-1\}$

feature space $\mathcal{X} \subseteq \mathbb{R}^d$

θ parameters, f_θ parametric model

$\mathbb{P}(\mathcal{Y})$ probability distribution on $\mathcal{Y}$

$\cong \Delta^{c-1} \cong \left\{ p \in \mathbb{R}^c : p_i \in [0,1], \sum_{i \in [c]} p_i = 1 \right\}$

$$\mathcal{X} \xrightarrow[\underline{x} \in \mathbb{R} \longmapsto \underline{x}^T W + \underline{b}]{f_\theta} \mathbb{R}^c \xrightarrow{\exp} \mathbb{R}^c_{>0} \xrightarrow[e^p \mapsto \frac{e^p}{\sum e^p}]{} \Delta^{c-1} \hookrightarrow$$

Ideally: $\frac{e^p}{\sum e^p}$ well defined as $e^p \in \mathbb{R}^d_{>0}$ → computer has too low precision

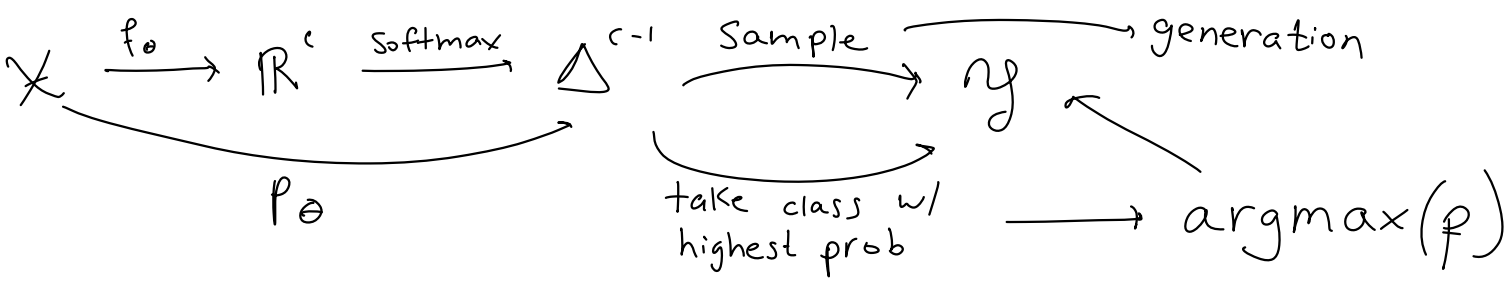
Practice: problems if $\sum e^p$ is very big or small

How to make softmax numerically stable?

↳ take $q \in \mathbb{R}$. $\frac{e^{p+q}}{\sum e^{p+q}} = \frac{e^p e^q}{\sum e^p e^q}$

So $\text{Softmax}(p) = \text{Softmax}(p+q) \quad \forall q \in \mathbb{R}$

Let $q = -\max p \implies \max_{p \in \mathbb{R}^c} \left(e^{p - \max p} \right) = 1$



accuracy: frequency of correct predictions

prediction $\mathcal{X} \longrightarrow \mathcal{Y}$

$$\text{argmax}(p) = \min\{ j \in [c] : p_j = \max p \}$$

convention used in case of multiple maxs

Note: if we're argmaxing, we don't need to softmax

$$\Rightarrow \text{argmax}(\text{softmax}(z)) = \text{argmax}(z)$$

---

training dataset $\mathcal{D}_{train}$ set parameters $\theta \in \mathbb{R}^M \longmapsto f_\theta$

so that $\ell(\mathcal{D}_{train}, \theta)$ = small

Recall Gradient Descent (GD):

1. Initialize $\theta_0 \sim \Theta$
2. Repeat $1 \le i \le S$, $\eta \in \mathbb{R}$ const

$$\theta_{i+1} \leftarrow \theta_i - \eta \nabla_\theta \ell(\mathcal{D}_{train}, \theta_i)$$

What should be the loss?

If we use accuracy and set loss = −(accuracy)

$$\Rightarrow \nabla_\theta \ell = 0 \text{ or undefined}$$

Since we have discrete set $\Rightarrow$ locally constant

We need a better loss function

$\mathcal{X} \xrightarrow{f_\theta} \mathbb{R}^c \xrightarrow{\text{softmax}} \Delta^{c-1}$

$\underline{x}, j \in \mathcal{D}$. $\theta \in \mathbb{R}^M$. $\ell(\underline{x}, j; \theta) =$ probability of j in predicted distribution

$p_\theta(\underline{x}) \in \Delta^{c-1}$

$\ell(\mathcal{D}_{train}; \theta) \stackrel{?}{=}$ average $\ell(\underline{x}, j; \theta)$ ← not that good if $\mathcal{D}$ is unbalanced

$\underline{x}, j \in \mathcal{D}$

If $\underline{\theta} = 0$, $f_\theta(\underline{x}) = \text{Softmax}(f_\theta(\underline{x})) = \text{Softmax}(\underline{0}) = \frac{1}{|c|}$

Likelihood of dataset

$\prod_{\underline{x}, j \in \mathcal{D}} p_\theta(\underline{x})_j$ ← super small unless p_θ super good

Log Likelihood

$\sum_{\underline{x}, j \in \mathcal{D}} \log(p_\theta(\underline{x})_j$

Negative Log Likelihood

$NLL = -\sum_{\underline{x}, j \in \mathcal{D}} \log p_\theta(\underline{x})_j$

$\Longrightarrow$ loss $= -$ average $\left(\log p_\theta(\underline{x})_j\right)$
$\underline{x}, j \in \mathcal{D}$

cross-entropy

In practice: $p_\theta(\underline{x}) = \frac{e^{\underline{z}}}{\sum e^{\underline{z}}}$ where $\underline{z} = \underline{x}^T W + \underline{b}$

$$\log p_\theta(\underline{z})_j = \frac{e^{z_j}}{\sum e^{\underline{z}}} = z_j - \log \sum e^{\underline{z}}$$

also quite numerically unstable

log sum exp

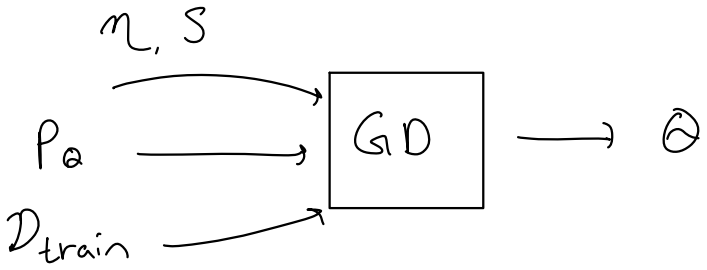
helps avoid NaN (not a number)

This is our loss

Ready to train!

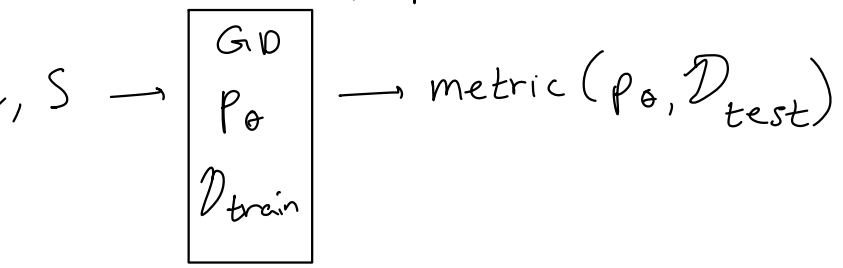
Algo: GD
Loss: CE
η: learning rate
S: # of steps

ultimate aim in ML:
generalization to unseen data
train $\mathcal{D}_{train}$
test $\mathcal{D}_{test}$



performance of p_θ:
metric loss / accuracy on $\mathcal{D}_{test}$

How to set hyperparameters (so far η, S)?



→ Can try out some η, S, get some metric(p_θ, $\mathcal{D}_{test}$)
suppose η_0, S_0 gets best result
But now $\mathcal{D}_{test}$ became train set for η, S.

To test generalization: test set for hyperparameter optimization

Hyperparameter search / tuning optimization

$\eta, S \longrightarrow$ [GD, p_θ, $\mathcal{D}_{train}$] $\longrightarrow$ metric$(p_\theta, \mathcal{D}_{valid})$

η, S w/ max metric$(p_\theta, \mathcal{D}_{valid})$ then metric$(p_\theta, \mathcal{D}_{test})$

So we have 3-way split:

$\mathcal{D}_{train}$	$\mathcal{D}_{valid}$	$\mathcal{D}_{test}$
80%	10%	10%
70%	15%	15%

Gold Standard of ML

you shouldn't even look at $\mathcal{D}_{test}$. Only calculate metric$(p_\theta, \mathcal{D}_{test})$ at end

Never change everything ever again.

dataset: MNIST $\mathcal{X} = \mathbb{R}^{28 \times 28}$ $d := 28 \cdot 28$

$\mathcal{Y} = [10]$

$$\begin{tikzcd} \mathcal{X} \arrow[r, "p_\theta"] \arrow[dr, "\underline{x} \mapsto \underline{x}^T W + \underline{b}"'] & \mathbb{P}(\mathcal{Y}) = \Delta^9 = \{ p \in \mathbb{R}^{10}_{\geq 0} : \sum p = 1 \} \\ & \mathbb{R}^{10} \arrow[u, "\text{softmax}"', "\underline{z} \mapsto \frac{e^{\underline{z}}}{\sum e^{\underline{z}}}"] \end{tikzcd}$$

$\theta = W, b$

<u>Algorithm</u>: GD

I. initialize $\theta \sim D_{initial}$

II. repeat for S steps: $\theta \leftarrow \theta - \eta \nabla_\theta$

<u>Loss Function</u>:

$\mathcal{X} \times \mathcal{Y} \times \mathbb{R}^M \xrightarrow{\ell} \mathbb{R}$

average over dataset entries

only on train split

<u>train split $\mathcal{D}_{train}$</u> $\longrightarrow$ <u>valid set</u> $\longrightarrow$ <u>test set</u>

get θ given $\varphi = (\eta, S)$ evaluate θ w/ φ evaluate θ, φ

After eval on test set $\longrightarrow$ STOP

$$\mathbb{R}^M \xrightarrow{\ell} \mathbb{R} : \quad \ell(\mathcal{D}_{train}; \theta) = \frac{1}{|\mathcal{D}_{train}|} \sum_{\underline{x}, y \in \mathcal{D}_{train}} \ell(\underline{x}, y; \theta)$$

$$\ell_{NLL}(\underline{x}, y; \theta) = \log p_\theta(\underline{x})_y$$

$\underline{x} \in \mathbb{R}^d, y \in [c]$

$\ell_{NLL}(\mathcal{D}_{train}; \theta)$

cross-entropy

train step: $\theta \leftarrow \theta - \eta \nabla_\theta \ell(\mathcal{D}_{train}; \theta)$

↳ autograd ⎫ =

MNIST $|\mathcal{D}_{train}| = 40000$

average $\nabla_\theta \ell(\underline{x}, y, \theta)$
$\underline{x}, y \in \mathcal{D}_{train}$

On large training sets, we don't want to compute loss on entire set. ⟶ use a small sample...

Stochastic Gradient Descent [SGD]

Idea: $\mathcal{D}_{train}$ = samples from $X \times Y$

So for training step, could use smaller sample

~ bootstrapping: simulate sampling from $X \times Y$ by sampling from $\mathcal{D}_{train}$

SGD w/ replacement.

new hyperparameter: minibatch size N' ← conventionally $N' = 2^{n'}$

Step: Sample $\mathcal{D}'$ of size N' (w/ replacement) from $\mathcal{D}_{train}$

$$\theta \leftarrow \theta - \eta \nabla_\theta (\mathcal{D}'; \theta)$$

SGD is actually better than GD.

↳ it introduces noise in training process ⟶ chance to escape local optimum

Conventionally $N' = 2^{n'}$, not anymore

Alternative: avoid repetitions

↳ SGD w/ random reshuffling

↳ usually what's meant with SGD

Note: $|\mathcal{D}_{train}| = N$, minibatch size N' ← yet another hyperparameter

successive slices: $\lfloor \frac{N}{N'} \rfloor$ + maybe 1 more slice

advantage: more data

disadvantage: too much noise

Alg

Repeat: ← epoch

ordering = permutation $\pi \in S_N$

Reshuffle $\mathcal{D}_{train} = \{(\underline{x}_i, y_i) : i \in I\}$

For $i = 0, \ldots, S' - 1$:

$$\mathcal{D}' = \{(\underline{x}_{\pi(j)}, y_{\pi(j)}) : j = iN', \ldots, \min((i+1)N', N) - 1\}$$

step through slices

On Arrays and Tensors

Math: $\mathbb{Z}, \mathbb{R}, \mathbb{C}$
CS: int64, float2, etc.

Array $[n_0] \times \cdots \times [n_{r-1}] \xrightarrow{a} K$

K is a set of numbers

multiindex $\underline{i} = (i_0, \ldots, i_{r-1})$

$$a(\underline{i}) =: a[i_0, \ldots, i_{r-1}]$$

r = dimension of array

$(n_0, \ldots, n_{r-1})$ = shape of array

ex: 1D array $\longrightarrow$ vector shape $(n) \longrightarrow$ n-dim vector

ex: 2D array $\longrightarrow$ matrix shape $(m,n) \longrightarrow$ mxn matrix

ex: $r>3$, e.g. $r=3 \longrightarrow$ batch of matrices

Dataset : MNIST, $d = 784$, $c = 10$

$|\mathcal{D}| = 60000$

$|\mathcal{D}_{train}| = 40000$, $|\mathcal{D}_{valid}| = 10000$, $|\mathcal{D}_{test}| = 10000$

$\theta = (\underline{w}, \underline{b})$

$\mathcal{X} = \mathbb{R}^{784} \xrightarrow{f_\theta = \underline{x} \mapsto \underline{x}^T \underline{w} + \underline{b}} \mathbb{R}^{10} \xrightarrow{\text{softmax}} \triangle^{10}$

1) GD. $\mathcal{D}' \longmapsto \theta \leftarrow \theta - \eta \nabla_\theta \ell_{NLL}(\mathcal{D}', \theta)$

full batch : $\mathcal{D}' = \mathcal{D}_{train}$

SGD : $\mathcal{D}$ samples from $\underline{X} \times Y$ random variables

can take smaller samples $\mathcal{D}'$ size N' ← minibatch size

epoch : Randomly permute full $\mathcal{D}_{train}$

$\mathcal{D}'$ = successive slices of N'

randomness : PRNG ⟶ deterministic after seed

evaluate training algorithm

Repeated Experiments

$i = 0, \ldots n$, X_i random variable : performance metric

↳ log accuracy on $\mathcal{D}_{valid}$

after training with same algorithm, hyperparameters

X_i : independent identically distributed n experiments

$$\mathbb{E}(\text{accuracy}) =: \mu \approx \overline{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

Law of Large Numbers:

if $\text{var}(x_i) = \sigma^2 < \infty$, then $\bar{X}_n \xrightarrow[\infty]{n} \mu$,

so $\forall \varepsilon > 0, \exists n_0$ s.t $n > n_0 \Longrightarrow |\bar{X}_n - \mu| < \varepsilon$

Confidence Intervals

Assume $X \sim \mathcal{N}(\mu, \sigma)$

Sample variance $S_n^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{X}_n)^2$

$T_n := \frac{\bar{X}_n - \mu}{\sqrt{S_n^2/n}}$: Student t-distribution w/ $n-1$ degrees of freedom

$$\mathbb{P}\left(|\bar{X}_n - \mu| < c\sqrt{S_n^2/n}\right) = \mathbb{P}(|T_n| < c) = \alpha$$

set α confidence level (usually 95% or 99%.)

and get c

Meta-Experiment: Run n experiments. Out of a lot of meta-experiments, α proportion of times, $|\bar{X}_n - \mu| < c\sqrt{S_n^2/n}$

Result: $X \sim \underbrace{\bar{X}_n \pm c\sqrt{S_n^2/n}}_{\text{confidence interval}}$ eg $n=10$, $\alpha = 95\%$.

$$\mathbb{P}\left(\sqrt{n}(\bar{X}_n - \mu) < \infty\right) \xrightarrow[\infty]{n} \mathbb{P}(\mathcal{N}(0, \sigma^2) < x)$$

Ensemble: many models trained on same task

used for: → HW 1

1) Aggregate predictions better than single model
2) Ensemble of versions specialized models
3) Parallelize training
4) train models with different hyperparameters in parallel
5) Confidence intervals.

Want to train ensemble in parallel

GPUs much faster

CPUs multiple CPU, SIMD → single instruction, multiple data

Basic Idea: 1 model: $\underset{\in \mathbb{R}^d}{x} \longmapsto \underset{\in \mathbb{R}^{d \times c}}{\underline{x}^T W} + \underset{\in \mathbb{R}^c}{\underline{b}}$

But for n models?

→ n weight matrices $W_i \in \mathbb{R}^{d \times c}$ $i = 0, \dots, n-1$

$\downarrow$

$\underline{W} \in \mathbb{R}^{n \times d \times c}$, $\underline{b} \in \mathbb{R}^{n \times c}$

Recall: $n=1$, $\mathcal{D}' = X \in \mathbb{R}^{N' \times d}$. $y \in [c]^{N'}$

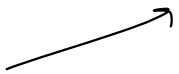
All N' logits at once: $\overbrace{\underset{(n',d)\cdot(d,c)}{XW}}^{(n',c)} + \overset{(c,)}{b}$

How do we add a matrix and vector?

→ Broadcasting

Broadcasting

$XW + \underline{b}$ valid mathematically: replace $\underline{b}$ by matrix $\rightarrow \begin{pmatrix} \underline{b}^T \\ \vdots \\ \underline{b}^T \end{pmatrix} N'$

broadcasting: tell computer $\underline{b}$ is like 

General Rules: tensors a, b shapes $(m_1, \ldots, m_K)$, $(n_1, \ldots, n_\ell)$
want to perform elementwise op. (e.g. $+$)

Recap: use an ensemble: models trained on same task

If we can: paralellize training.

Uses:
- aggregate predictions of the better
- hyperparameter search
- better prediction of algorithm performance

Parallelization: same architecture

Broadcasting: effficient operations w/ constant axes

e.g. Affine operation on minibatch

Even with 1 model: X minibatch features : (N', d) ← data entry index

W weights : (d, c)

$\underline{b}$ biases : $(c,)$ → class index

↑ constant along data entry index

XW (N', c) → logit displacement per data entry

b $(c,)$ → constant logit biases

Train model $\underline{x} \longmapsto \underline{b}$ get $\text{softmax}(\underline{b}) =$ frequency of classes

To form $XW + \underline{b}$, view $\underline{b}$ as $\begin{pmatrix} \underline{b}^T \\ \vdots \\ \underline{b}^T \end{pmatrix}$

$[m_0] \times \cdots \times [m_{r-1}] \xrightarrow{a} K$

$[n_0] \times \cdots \times [n_{s-1}] \xrightarrow{b} K$

$K = \mathbb{Z}, \mathbb{R}, \mathbb{F}_2$, int64, bool, etc.

$\mu = +, *, ==,$

perform elementwise operation $K \times K \xrightarrow{\mu} L$

assert a.shape == $(m_0, \ldots, m_{r-1})$ | b.shape $(n_0, \ldots, n_{s-1})$

1. len(a.shape) or a.ndim $= r$

   * Assume $r \leq s$, if not, try swapping a, b

2. Convention: fill shape of a w/ 1's on the left to have ndim s, so a.shape : $(\underbrace{1, \ldots, 1}_{s-r}, m_0, \ldots, m_{r-1})$

3. Loop over $1 \leq i < s$

   have dims m_i', n_i

   Cases:

   a) $m_i' = n_i$ ✓

   b) either is 1 : view as constant along dim

   c) $m_i' \neq m_i$: Error

---

ex) [1,2,3) + [1]
= [2,3,4]

shapes (3,) and (1,)
So [1] viewed as [1,1,1]

ex) [1,2,3] + [[1]]
= [[1,2,3]] + [[1]]
= [[2,3,4]]

shapes (3,) and (1,1)
so broadcasting → shapes (1,3) and (1,1)

Broadcasting matrix multiplication

Difference: previously elementwise ops

now: $K^{(l,m)} \times K^{(m,n)} \xrightarrow{@} K^{(l,n)}$

$[m_r] \times \cdots \times [m_{r-1}] \xrightarrow{a} K$

$[n_0] \times \cdots \times [n_{s-1}] \xrightarrow{b} K$

for $a @ b$: $(m_0, \cdots, m_{r-3})$ and $(n_0, \cdots, n_{s-3})$ broadcast as previously

$\rightsquigarrow (p_0, \cdots, p_{t-3})$

Then I can perform

$(p_0, \cdots, p_{t-3}, m_{r-2}, m_{r-1}) @ (p_0, \cdots, p_{t-3}, n_{s-2}, n_{s-1})$ if $m_{r-1} = n_{s-2}$

then get $(p_0, \cdots, p_{t-3}, m_{r-2}, n_{s-1})$

where at $\underline{i} := (i_0, \cdots i_{t-3}) \in [p_0] \times \cdots \times [p_{t-3}]$

we have $(a @ b)[i] = a[i] @ b[i]$

Important to us: ensemble of e models

$W : (e, d, c)$

$\underline{b} : (e, c)$

validation $\times (N', d)$

train $\times (e, N', d)$

X_v @ W (N', d) and (e, d, c)

batch shapes $()$ and $(e,)$

$(1,)$ and $(e,)$

X_v viewed as (e, N', d) as constant along ensemble dimension

So: $(X_v @ W)$ shape (e, N', c)

$(X_v @ W)[i_0] = X_v @ W[i_0]$

$X_t @ W$ (e, N', d) and $(e, d, c) \longrightarrow (e, N', c)$

$(X_t @ W)[i_0] = X_t[i_0] @ W[i_0]$

Note: $\underbrace{(e, N', c)}_{X\ W} \overset{?}{+} (e, c)$ b might be problematic if $N' = e$ — unintended and uncaught error

To fix:

Pytorch: b.reshape$(e, 1, c)$

$b[:, None, :]$ or $b[..., None, :]$

b.unsqueeze(-2)

Last 2 times: train multiple models on same task

- confidence intervals: how much can I trust $\bar{X}$ (mean test performance) as an estimate for $\mathbb{E}(x)$: $\bar{X} \pm I$ with α level
- ensembling e.g HW 1, 3
- hyperparameter search (change them b/w runs)

hyperparameters:

- learning rate: $\theta \leftarrow \theta - \eta \nabla_\theta$
- batch size: N'
- total number of steps S → just run w/ longest option

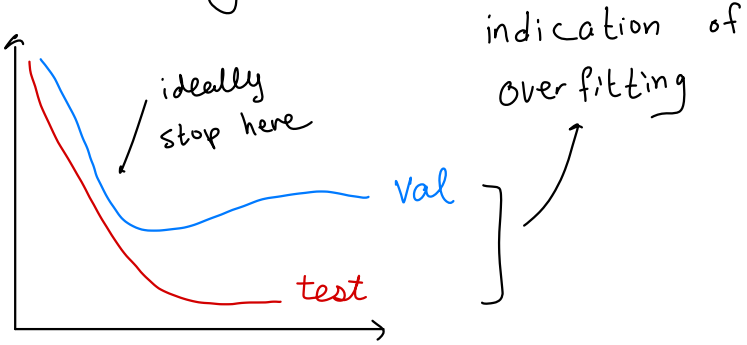
Vectorize:

η

Not really vectorize:

$N' \rightarrow$ (need to pad)

Overfitting:



Why isn't it trivial when I should stop.

Best model:

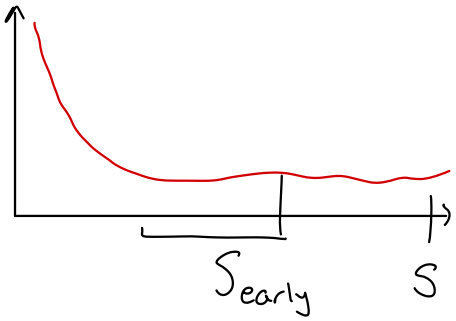
get θ_t, choose t, keep most recent and best valid

Result so far is $\ell_{best} \in$ float 64 and get $\ell_{new} \in$ float 64

when is ℓ_{new} better than ℓ_{best}?

Set threshold $\tau > 0$, say I have new best $\ell_{new} + \tau < \ell_{best}$

Early Stopping: Stop if not recieved improved model for given number of training steps



use a count c of valids w/out improvement

iff improve: $c = 0$

elsee: $c += 1$

Multiple models: say we have ensemble of e models

Keep:
- best losses ℓ_i $\quad 0 \le i < e$
- most recent parameters Θ_i
- best parameters Θ'_i

$m \in \mathbb{F}_2^e = \mathbb{B}^e$

Recieve: ℓ_i^{new}

improvement mask:

$[\ell^{new} + \tau < \ell^{best}]$ is a Boolean vector m

$\theta \leftarrow \theta^{new} \quad | \quad \ell^{best}[m] = \ell^{new}[m], \ \theta'[m] = \theta^{new}[m]$

tensor T ← stored in memory

T[a:b] view : no memory op. to data

mask m, index tensor I

T[m], T[I] copy

Note: T' = T[m] } working on a copy
T' += 1
T remains unchanged

Instead, do:
T[m] += 1

count c of validations w/o improvements:

```
if m.any():
    c = 0
else:
    c += 1
```

We now have a new hyperparameter:
max steps w/out improvement

Grid Search: Hyperparameter search

- select some reasonable values for some hyperparameters.
- test all combinations

$$\eta \in \{10^i : i = -2, -1.5, \ldots, 1\}$$

$$N' \in \{2^i : i = 3, 4, \ldots, 10\}$$

Train models for all pairs η, N'

Pros: Easy to set up, intuitive

Cons.
- Inefficient to do w/ many hyperparameters
- If your initial set of hyperparameters are bad, you're cooked.
  - ↳ No feedback on bad values
- Involves a fixed number of runs
- Can't adapt to change in resources

Compare: Random Search

→ sample hyperparams at random

Bayesian optimization

Reinforcement Learning (RL)

Models only get reward signal

↳ similar to humans/organisms learn

In RL, model = agent

★ Canonical Textbook: Sutton-Barto Intro RL

↳ good for RL basics, not deep

↳ lots of exercises coding toy environments

↳ good coding practices, not too many proofs

Markov Decision Process (MDP)

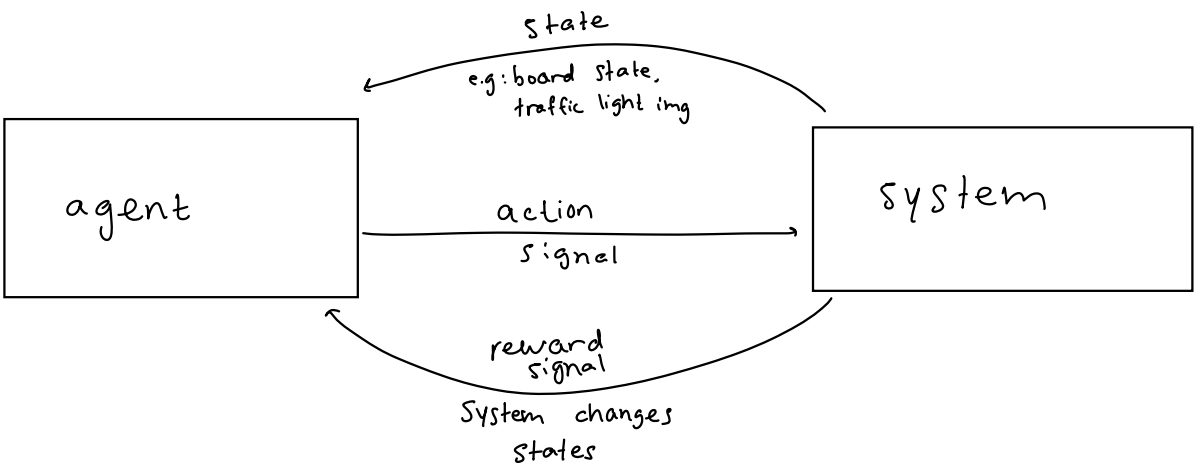
State Space S: state of system agent interacts with

- discrete: chess/board games
  graph model of pathfinding
  classic algos (e.g. graph matching)
  $\{\#\}$ single state
  ↳ multi-armed bandit
  sequence of tokens ← LLM agents
- continuous:
  weather forecast → supervised task
  economic = " =
  real time games
  life...

Supervised Learning as RL

state = feature vec

reward = right output



Markov Property: System entirely described by state

$S \rightarrow O$

- board game ✓
- life ×
- realtime games ×

Partially observable MDP (POMDP): incomplete observation

initial state

Random variable S_0 that gives distribution of initial state

Terminal State: if so, process ends

Action Space: for state $s \in S$

Set $A(s)$ of possible actions

discrete
- keys to control game
- vocabulary

continuous
- time of action
- control plane wing angle

only valid actions

To simplify notation, $A = \bigcup_{s \in S} A(s)$

$A \setminus A(s)$ are illegal actions

What should happen if illegal actions?

- not change state
- give penalty
- go to end state (terminate and penalize)
- if there is inert action, do that

Your choice whether to allow agent to choose illegal action

Policy: Recieve state (observation), output action

$\mathcal{S} \xrightarrow{\pi} \mathcal{A}$ ← deterministic: $\pi(s) \in \mathcal{A}$

or

$\mathcal{P}(\mathcal{A})$ ← stochastic: $\pi(s) \in \mathcal{P}(\mathcal{A})$

$\mathcal{P}(a \in \pi(s)) = \pi(a|s)$

Initially: state rv S_0

action rv A_0

Transitions

$\mathcal{S} \times \mathcal{A} \xrightarrow{t} \mathbb{R} \times \mathcal{S}$ reward x next state

$\mathcal{P}(\mathbb{R} \times \mathcal{S})$ stochastic transition

$\mathcal{P}(r, s' | s, a)$

Random Variables:

$S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$ etc

$A_0 = \pi(S_0)$ $\quad R_1, S_1 \sim t(S_0, \pi(S_0))$

trajectory

if terminal state reached: trajectory is called episode

measure performance for agent/policy $\mathcal{S} \xrightarrow{\pi} \mathcal{P}(\mathcal{A})$

How to incentivize getting reward faster:

original idea: $\mathbb{E}_\pi \sum_i R_i$

discount: hyperparameter $\gamma \in (0, 1]$, usually $\gamma = 0.99$

Discounted Return

$$\mathbb{E}_\pi \sum_{i=1}^{\infty} \gamma^i R_i$$

Recall:

Finite MDP:

N - states $\mathcal{S}$

M - actions $\mathcal{A}$

Deterministic Reward:

$\mathcal{S} \times \mathcal{A} \xrightarrow{r} \mathbb{R}$

$R \in \mathbb{R}^{N \times M}$

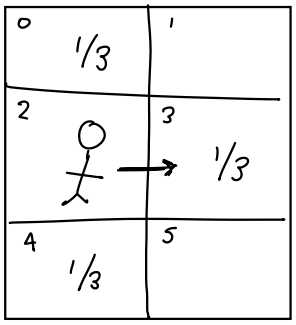
Transition

$\mathcal{S} \times \mathcal{A} \xrightarrow{t} P(\mathcal{S})$

$[N] \times [M] \longrightarrow \Delta^{N-1} \subseteq \mathbb{R}^N$

$P \in \mathbb{R}^{N \times M \times N}$, $P_{i,j,k} = P(s' = k \mid s = i, a = j)$

e.g: slippery frozen lake



$P_{2,0} = \begin{bmatrix} 1/3 \\ 0 \\ 0 \\ 1/3 \\ 1/3 \\ 0 \end{bmatrix}$

$\mathcal{S} \xrightarrow{\pi} P(\mathcal{A})$

$[N] \longrightarrow \Delta^{M-1} \subseteq \mathbb{R}^M$

$\pi \in \mathbb{R}^{N \times M}$

discounted return at state $S = s$

$$V_\pi(s) := \mathbb{E}_\pi \sum_{i=0}^{\infty} \gamma^i R_{i+1} \mid S_0 = s, \quad \gamma \in (0, 1]$$

Policy Iteration Algorithm

Needed: calculate V_π

Value function estimate $\tilde{v}_\pi \in \mathbb{R}^N$

$$\underbrace{\left(T^\pi \tilde{v}_\pi\right)(s)}_{\text{expected Bellman}} = \underbrace{\sum_{a \in \mathcal{A}} \pi_{s,a} \cdot R_{s,a}}_{\text{next reward following } \pi} + \underbrace{\gamma \sum_{a \in \mathcal{A}} \pi_{s,a} \sum_{s' \in \mathcal{S}} P_{s,a,s'} \tilde{V}_\pi(s')}_{\text{rest of estimated discounted return}}$$

action-value function

$$q_\pi(s,a) = \mathbb{E}_\pi \sum_{i=0}^{\infty} \gamma^i R_{i+1} \mid S_0 = s, A_0 = a$$

$$= R_{s,a} + \gamma \sum_{s' \in \mathcal{S}} P_{s,a,s'} V_\pi(s')$$

Policy iteration

1. $\exists s,a : q_\pi(s,a) > V_\pi(s)$

$$\pi'(s) := \operatorname*{argmax}_{a \in \mathcal{A}} q_\pi(s,a) \qquad \text{get } \pi' > \pi$$

2. else, π optimal ✓

Value Iteration:

Estimate value function of optimal policy

$\tilde{v}_* \in \mathbb{R}^N$

given $\tilde{v}_*$ get $\tilde{q}_*(s,a) = R_{s,a} + \gamma \sum_{s' \in S} P_{s,a,s'} \tilde{v}_*(s')$

$(T^* \tilde{v}_*)(s) = \max_{a \in A} \tilde{q}_*(s,a)$

$\mathbb{R}^N \xrightarrow{T^*} \mathbb{R}^N$, $\gamma < 1$ contraction

Thm. 1. $\lim_{n \to \infty} (T^*)^n \tilde{v}_* = v_*$ value function of optimal policy

2. $T^* \tilde{v}_* = \tilde{v}_*$ iff $\tilde{v}_* = v_*$

Thm. 1. Select threshold $\varepsilon > 0$ starting $\tilde{v}_* \in \mathbb{R}^N$

2. Repeat $\tilde{v}_* \leftarrow T^* \tilde{v}_*$ until

$$\| T^* \tilde{v}_* - \tilde{v}_* \|_\infty =: \max_{s \in S} \left| (T^* \tilde{v}_*)(s) - \tilde{v}_*(s) \right|$$

$$\leq \frac{(1-\gamma)\varepsilon}{2\gamma}$$

3. Then: $\pi(s) = \operatorname{argmax}_{a \in A} \tilde{q}_*(s,a)$

a) π ε-optimal: $\| v_\pi - v_* \|_\infty \leq \varepsilon$

b) $\| \tilde{v}_* - v_* \|_\infty \leq \frac{\varepsilon}{2}$

Q-learning

$Q_n(s,a) \in \mathbb{R}$ approximate optimal action-value function at step n. $(\simeq q_*(s,a))$

Approximate optimal value function

$$V_n(s) := \max_{a \in A} Q_n(s,a)$$

Want infinitely many steps in environment.

Reset if over

Algorithm:

$Q_0(s,a)$ anything but $=0$ at terminal state s

at step n: at state $S_n \in S$, sample $a_n \sim \pi(S_n)$

get reward r_{n+1}, next state S_{n+1}

$$Q_{n+1}(s,a) := \begin{cases} Q_n(s,a) & \text{if not } s = S_n \text{ and } a = a_n \\ \alpha_n \left(r_{n+1} + \gamma V_n(S_{n+1}) \right) + (1-\alpha_n) Q_n(s,a) \end{cases}$$

Thm: Any policy π: $\forall s \in S, a \in A, \pi(s)_a > 0$

Finite MDP $|S| < \infty, |A| < \infty$. Rewards can be stochastic bounded

$1 \leq i$, $n^i(s,a)$ =: step at n such that $s_n = s$, $a_n = a$ for ith time.

Assume $\forall s,a$, $\sum_{i=1}^{\infty} \alpha_{n^i(s,a)} = \infty$, $\sum_{i=1}^{\infty} \alpha^2_{n^i(s,a)} < \infty$

So let $\alpha_n = \frac{1}{i}$,

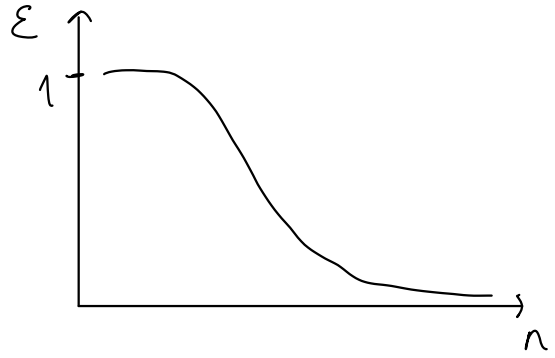
Then, $Q_n(s,a) \longrightarrow q_*(s,a)$ almost always.

Given Q_n, policy to get nice return → greedy policy

$$\pi(s) := \underset{a \in A}{\operatorname{argmax}}\ Q_n(s,a)$$

For theorem: random policy $\pi(s,a) = \frac{1}{|A|}$

ε-greedy: $\varepsilon > 0$, random with prob ε, greedy with prob $(1-\varepsilon)$



Deep Learning : Multilayer Perceptrons (MLP)

So far, we used a specific architecture for supervised

architecture : parameters $\longmapsto$ model

$\mathbb{R}^M \qquad\qquad \mathcal{X} \longrightarrow \mathbb{P}(\mathcal{Y})$

affine : $\underline{\theta} \longmapsto (\underline{x} \longmapsto \underline{x}^T \underline{w} + \underline{b})$

There's a synthetic classification dataset that affine models struggle with: Hypersphere classification

C classes, features for class $c \in [C]$ sampled from

$$r_c \left(\mathcal{U}(S^{n-1}) + \mathcal{N}(\underline{0}, \underline{S}) \right) + \underline{X}_c$$

S^{n-1} : $(n-1)$ dim hypersphere : $S^{n-1} = \left\{ \underline{x} \in \mathbb{R}^n \,;\, x_0^2 + \dots + x_{n-1}^2 = 1 \right\}$

$\mathcal{U}$: uniform distribution with $\mathcal{N}(0,1)$ for additive noise

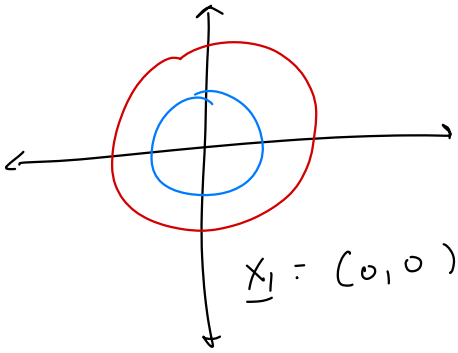
r_c : radius

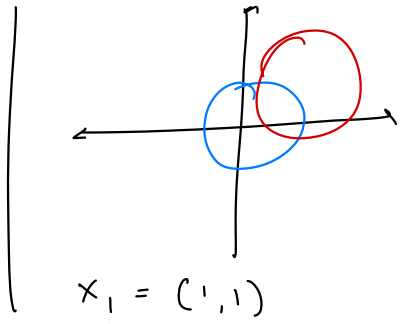
$\underline{X}_c$: offset

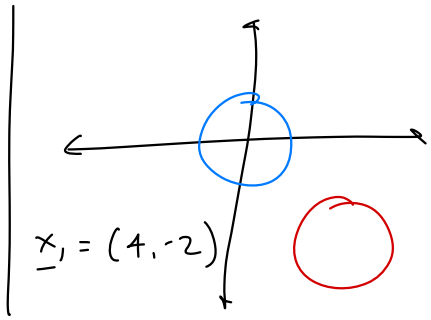
How to sample from $\mathcal{U}(S^{n-1})$ in practice?

- We know how to sample from uniform $[0,1]$, normal $(0,1)$
- Sample from $\mathcal{N}(0,1)$ and normalize (resample if small)

$n=2,\ C=2,\ x_0=0,\ r_{0,1}=0.8, 0.1,\ s=0.1$







Binary Classification: $C=2$

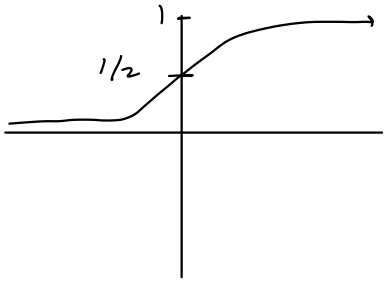
$p_0, p_1,\ p_0 + p_1 = 1,\ p_0 = 1 - p_1$ (so we only need p_1)

class 0: "negative" class

1 - "positive" class

model: $\mathbb{R}^d \longrightarrow \mathbb{R}^1 \xrightarrow{\sigma} [0,1]$

sigmoid: $\sigma(z) = \dfrac{1}{1+e^{-z}}$



inverse: logit function: $z = \log \dfrac{p}{1-p}$

binary accuracy:

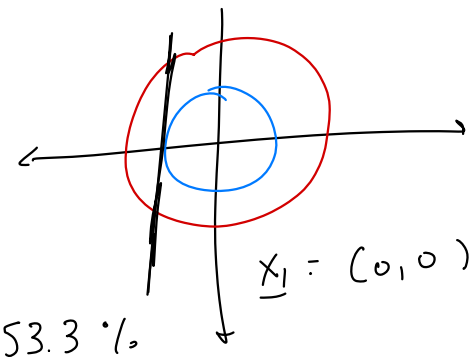
$p = p_1 := \sigma(z)$ — this is class 1 if $p > 1/2,\ z > 0$

binary cross-entropy:

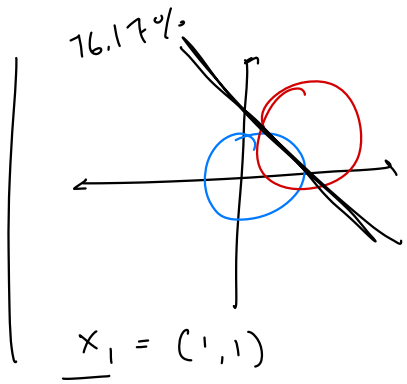
$$\ell(x, y, \theta) = y \log\left(\sigma(f_\theta(x))\right) + (1-y) \log\left(1 - \sigma(f_\theta(x))\right)$$

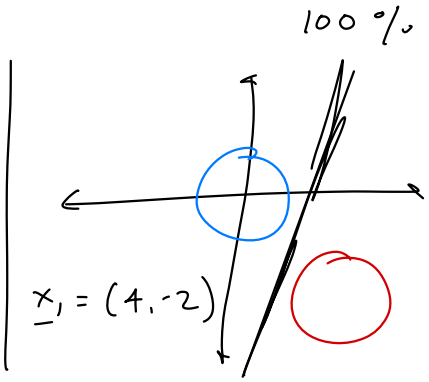
w/ $y \in \{0,1\}$

Try an affine model:

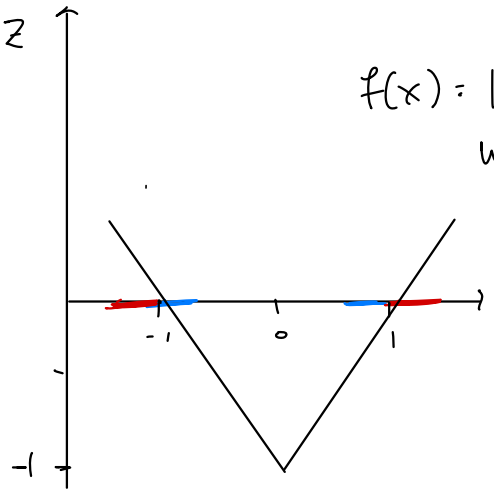


It sucks here





$n = 1$, $x_1 = (0,)$



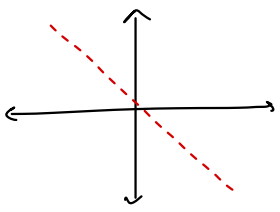
$f(x) = |x| - 1$ works

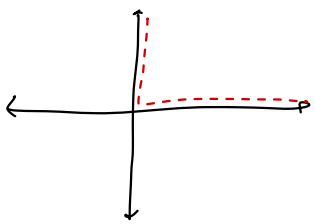
Let's build f from
- affine maps
- elementwise $\mathrm{ReLu}(x) = \max(0, x)$

$|x| - 1 = \mathrm{ReLu}(x) + \mathrm{ReLu}(-x)$

$z = |x| - 1$

$\mathbb{R} \xrightarrow{x \mapsto x(-1)} \mathbb{R}^2 \xrightarrow{\mathrm{ReLu}} \mathbb{R}^2 \xrightarrow{y_0, y_1 \to y_0 + y_1 - 1} \mathbb{R}$





Multilayer Perceptron (MLP)

$\mathcal{X} \subseteq \mathbb{R}^d$, $y \subseteq \mathbb{R}^c$, $[c]$

$$\mathbb{R}^d \xrightarrow{\underline{x} \mapsto \underline{x}^T W^{(0)} + \underline{b}^{(0)}} \mathbb{R}^{d_1} \xrightarrow[\text{elementwise}]{\underline{x} \mapsto a(\underline{x})} \mathbb{R}^{d_1}$$

$$\xrightarrow{\underline{x} \mapsto \underline{x}^T W^{(1)} + \underline{b}^{(1)}} \mathbb{R}^{d_2} \xrightarrow[\text{elementwise}]{\underline{x} \mapsto a(\underline{x})} \mathbb{R}^{d_2}$$

"deep"

$$\vdots \qquad \vdots$$

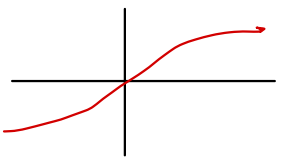
$$\mathbb{R}^{d_{L-1}} \xrightarrow{\underline{x} \mapsto a(\underline{x}^T W^{(L)} + \underline{b}^{(L)})} \mathbb{R}^c \quad \text{final}$$

$\ell = 0, \ldots, L-1$

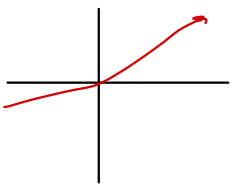
affine maps $\mathbb{R}^{d_\ell} \xrightarrow{\underline{x} \mapsto \underline{x}^T W^{(\ell)} + b^{(\ell)}} \mathbb{R}^{d_{\ell+1}}$

activation functions $\mathbb{R}^{d_{\ell+1}} \xrightarrow{a(\underline{x}) = \begin{pmatrix} a(x_0) \\ \vdots \\ a(x_{d_{\ell-1}}) \end{pmatrix}} \mathbb{R}^{d_\ell}$

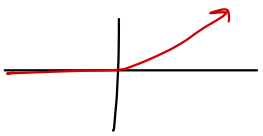
examples: $a = \tanh$



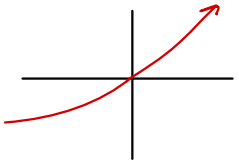
$a =$ Leaky ReLu



$a =$ ReLu



$a =$ ELU



$a =$ GeLu : $a(x) = \underline{\Phi}(x)$

MLP $\subseteq$ Feed forward Networks $\subseteq$ (Artificial) Neural Networks

$\mathbb{R}^{d_\ell} \xrightarrow{A^{(\ell)}} \mathbb{R}^{d_{\ell+1}} \xrightarrow{\sigma} \mathbb{R}^{d_{\ell+1}}$ hidden layers

L = # of hidden layers depth

d_ℓ $(\ell \neq 0, L+1)$ width

$d_0 = d$

$d_{L+1} = C$

$L = 0$ affine model

$L = 1$ shallow neural network SNN

$L > 1$ deep neural network DNN

total # of parameters $M = \sum_{\ell=0}^{L} (d_\ell + 1)\, d_{\ell+1}$

Deep Learning is called Representation Learning since it cumulates a bunch of "inner representations"

$C(\mathbb{R}^n, \mathbb{R}^m) = \{\text{continuous } \mathbb{R}^n \to \mathbb{R}^m\}$

$C(\mathbb{R}^n, 1) = C(\mathbb{R}^n)$

uniform convergence on compact subsets $m \subseteq C(\mathbb{R}^n, \mathbb{R}^m)$

say $m \subseteq C(\mathbb{R}^n, \mathbb{R}^m)$ dense, then,

$\forall f \in C(\mathbb{R}^n, \mathbb{R}^m)$, $K \subseteq \mathbb{R}^n$ compact, $\forall \varepsilon > 0$

$\exists g \in m$, $\max_{x \in K} |f(x) - g(x)| < \varepsilon$

SNN: $\mathbb{R}^d \xrightarrow{A^{(0)}} \mathbb{R}^{n_1} \xrightarrow{a} \mathbb{R}^{n_1} \xrightarrow{A^{(1)}} \mathbb{R}^c$

Thm: $a \in C(\mathbb{R})$, $SNN(a, d) \subseteq C(\mathbb{R}^d)$ is SNN. TFAE

1. $SNN(a, d) \subseteq C(\mathbb{R}^d)$ dense
2. a is not a polynomial.

Thm: $a \in C(\mathbb{R})$ nonaffine. $\exists x \in \mathbb{R}$. $a'(x) \neq 0$.

$DNN(a, d, n) \subseteq C(\mathbb{R}^d, \mathbb{R}^c)$ with inner widths n.

Then: $DNN(a, d, d+c+2) \subseteq C(\mathbb{R}^d, \mathbb{R}^c)$ dense.

Def: $\mathbb{R}^n \xrightarrow{F} \mathbb{R}^m$ piecewise linear function. A linear region is maximal $X \subseteq \mathbb{R}^n$, $F|X$ is linear.

Thm: L hidden layers of width $n \geq d$, can compute piecewise linear functions with

$\Omega\left(\left(\frac{n}{d}\right)^{(L-1)d} n^d\right)$ linear regions.

[↑ depth. exp, ↑ width: poly]

Initialization: vanishing, explosion of values

Trace the variance of inner features.

X - input random variable

$\|$

$H^{(0)}$ hidden unit

$z^{(l)}$ pre-activation

$$z^{(1)} := X^T w^{(0)} + b$$
$$H^{(1)} := a(z^{(1)})$$
$$\vdots$$
$$z^{(l)} := (H^{(l)})^T w^{(l)} + b^{(l)}$$
$$H^{(l+1)} := a(z^{(l+1)})$$

$\leftarrow$ variance

$V z^{(l)} \xrightarrow[\infty]{l} 0$, f_θ about constant, vanishing of values

$V z^{(l)} \xrightarrow[\infty]{l} \infty$, f_θ chaos, explosion of values

We want to be on the edge of chaos

Thm: Suppose that

1 Initial weights $w_{ij}^{(l)}$ iid (independently, identically distr.) mean 0, variance σ_l^2

2. $b^{(l)} = 0$ (focal loss)

3. $\mathbb{E}X = 0$, $VX = 1$

How to get 3 in expectation?

$$X \leftarrow X - \bar{X}, \quad \bar{X} = \frac{1}{|\mathcal{D}|} \sum_{i=0}^{|\mathcal{D}-1|} X_i$$

$$X \leftarrow X - \bar{X} \Big/ \sqrt{\frac{1}{|\mathcal{D}|-1} \sum_{i=0}^{|\mathcal{D}|-1} X_i^{\odot 2}}$$

$\odot$ elementwise

Thm: 1. $\mathbb{E}\, z^{(\ell)} = 0 \quad \forall \ell$ — independent of j

2. $\mathbb{E}\left(\left(h_j^{(\ell)}\right)^2 \right) := \mathbb{E}\left(\left(h^{(\ell)}\right)^2 \right)$

$$V z^{(\ell+1)} := V z_j^{(\ell+1)} = n_\ell \, \mathbb{E}\left(\left(h^{(\ell)}\right)^2 \right) \sigma_\ell^2$$

Glorot (Xavier) Initialization: $a = \tanh$

a about linear around 0

$$\mathbb{E}\left(\left(h^{(\ell)}\right)^2 \right) \approx a'(0)\, \mathbb{E}\left(\left(z^{(\ell)}\right)^2 \right)$$

Then get $V z^{(\ell)} = 1$ if $\sigma_\ell^2 = \dfrac{1}{n_\ell \, a'(0)}$

Kaiming He Initialization: $a = \mathrm{ReLu}$

$$\mathbb{E}\left(a\left(z^{(\ell)}\right)^2 \right) = \frac{1}{2}\, \mathbb{E}\left(\left(z^{(\ell)}\right)^2 \right) \quad \text{so with } \sigma_\ell^2 = \frac{2}{n_\ell},$$

get $V z^{(\ell)} = 1 \quad \forall \ell$

ELU (Exp Linear Unit)

$$a(z) = \lambda \begin{cases} z & z > 0 \\ \alpha\left(e^z - 1\right) & z \le 0 \end{cases}$$

Special ELU = **SELU**

$w^{(\ell)} \sim \eta\left(0, n_\ell^{-0.5}\right)$

$z, h \sim$ normal

calculate $\lambda \approx 1.0507$

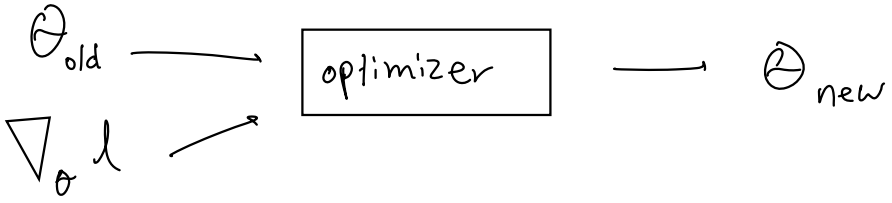
$\alpha \approx 1.6733$

Optimizers (contd)

Optimizers: more efficient training

$\mathcal{X} \xrightarrow{f_\theta} P(\mathcal{Y})$ architecture: change $\theta \in \mathbb{R}^M$

$\mathcal{X} \times \mathcal{Y} \times \mathbb{R}^M \xrightarrow{l} \mathbb{R}$ loss



Stateful optimizer $\xi \in \mathbb{R}^{M'}$ optimizer state ~ memory
optimizer parameters

GD with momentum

Idea:
loss landscape has regularities
so it may be worth keeping direction

$\xi = \underline{v} \in \mathbb{R}^M$ init: 0

$\underline{v}_{new} = \beta \underline{v}_{old} - \eta \nabla_\theta$

$\theta_{new} = \theta_{old} + v_{new}$

How much the memory is used depends on β

$\beta \in [0,1]$

Adam (Adaptive Moment Estimation)

$\mathcal{X} \subseteq \mathbb{R}^d$, $0 \le j \le M$, $\frac{\partial l}{\partial j}$ usually 0 but sometimes suddenly very nonzero

Give more importance to rare events

estimated first moment $\underline{m}_{new} = \beta_1 \underline{m}_{old} + (1-\beta_1)\nabla_\theta$

($\underline{v}$ in GD w/ momentum)

elementwise squares ←

estimated uncentered variance: $\underline{v}_{new} = \beta_2 \underline{v}_{old} + (1-\beta_2)\nabla_\theta^{\odot 2}$

$\underline{m}_{init} = 0$

$\underline{v}_{init} = 0$

t = step ID

"

1, 2, 3, ...

adjust

$$\hat{\underline{m}} = \frac{\underline{m}}{1-\beta_1^t}$$

$$\hat{\underline{v}} = \frac{\underline{v}}{1-\beta_2^t}$$

$$\Delta\theta = \eta \frac{\hat{\underline{m}}}{\sqrt{\hat{\underline{v}}} + \varepsilon}$$

$$\theta_{new} = \theta_{old} + \Delta\theta$$

β_1 : first moment decay ≈ usually 0.9

β_2 : second moment decay ≈ usually 0.999

↳ recently: pretraining LLMs showed 0.95 works better

Also convergence:

$0 << \beta_2 < 1$ ← task dependent

$\beta_1 < \sqrt{\beta_2}$

Regularization & weight decay

L_p regularization: penalize large parameter values

$$\ell_{changed} = \ell_{task} + \lambda_p \frac{1}{p} \sum_{j=0}^{M-1} |\theta_j|^p$$

$$(\nabla_\theta)_{changed} = \nabla_\theta + \lambda_p \, \mathrm{sgn}(\theta) \, \theta^{\odot p-1}$$

(elementwise)

p odd

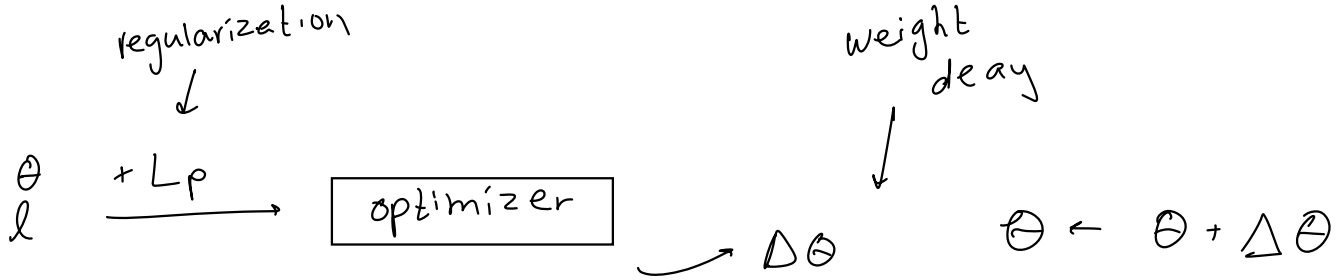
$\lambda_p \, \theta^{\odot p-1}$

p even

Alternative regularization:

weight decay: direct L_2 penalty subtracted from $\Delta\theta$

$$\Delta\theta \leftarrow \Delta\theta - \lambda\theta$$



In GD: L_2 reg = weight decay

AdamW: Adam + weight decay

In torch, Adam

weight-decay: L_2 reg λ_2

Deep Reinforcement Learning

RL: task ~ environment

- state space $\mathcal{S}$ (features s)
- action space ~ targets $\mathcal{A}$
- transitions $\mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R} \times \mathcal{S})$ (reward)
- initial state distribution

model

policy
agent $\mathcal{S} \xrightarrow{\pi} \mathcal{P}(\mathcal{A})$

evaluate policy:

$$\mathbb{E} \sum_{k=1}^{\infty} \delta^{k-1} R_k$$

discount: $\delta \in [0,1]$

decay rewards incentive to get rewards sooner

S_0

$A_0 = \pi(S_0)$

$R_1, S_1 = t(S_0, \pi(S_0))$

$A_1 = \pi(S_1)$

$$G_r = \mathbb{E} \sum_{k=r+1}^{\infty} \delta^{k-1} S_k$$

To calculate G_0: if $|\mathcal{S}|, |\mathcal{A}| < \infty$, $\delta < 1$ reward deterministic

can solve $|\mathcal{S}| \times |\mathcal{A}|$ system of linear equations

↳ if t is known

Q-learning: estimates via sampling

$v_\pi(s) = \mathbb{E}\, G_0 \mid S_0 = s$ value function

$q_\pi(s,a) = \mathbb{E}\, G_0 \mid S_0 = s, A_0 = a$ action-value function

Q-learning: model to approximate $q_\pi(s,a) \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$

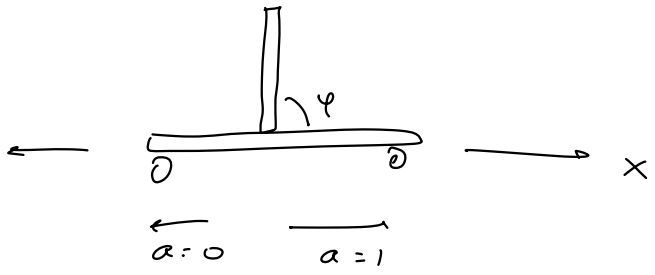
Deep Reinforcement Learning (DRL)

Now: $\mathcal{S} \subseteq \mathbb{R}^d$, $\mathcal{S} \xrightarrow{\pi} \mathcal{P}(\mathcal{A})$. DRL π_θ

[alternative: $\mathcal{S} \xrightarrow{q_\theta} \mathbb{R}^{|\mathcal{A}|}$]

warning: this is too easy

DRL Hello World: Cart Pole



$\mathcal{S} = \mathbb{R}^4$ → pos x, vel x, angle φ, angle vel ω

reward: +1 each step

terminate if either
- $\varphi \notin [-12°, 12°]$
- $x \notin [-2.4, 2.4]$
- reach 500 steps

Policy Gradient Theorem

Have G_0, $\mathcal{S} \xrightarrow{\pi_\theta} \mathcal{P}(\mathcal{A})$. π_θ good $= \mathbb{E} G_0$ big

$\nabla_\theta = -\nabla_\theta \mathbb{E} G_0$ ⇢ we need to compute

$$\frac{\partial \mathbb{E} G_0}{\partial j} = \lim_{h \to 0} \mathbb{E} G_0 \quad \pi_{\theta'}, \ \theta'_j = \theta_j + h$$

we need the policy gradient theorem

this is extremely hard

Theorem: $\exists$ constant $c > 0$: $\forall \theta \in \mathbb{R}^M$

$$\nabla_\theta G_0 = c \, \mathbb{E} \sum_{k=0}^{\infty} q_\pi(S_k, A_k) \nabla_\theta$$

$$= c \, \mathbb{E}_{\substack{S \sim d^{\pi_\theta} \\ A \sim \pi(S)}} q_{\pi_\theta}(S_k, A_k) \nabla_\theta \log \pi_\theta(A_k \mid S_k)$$

d^{π_θ} = distribution of states under π_θ

Estimate $q_{\pi_\theta}(s, a)$, $\mathbb{E}$

Sample trajectory $s_0, s_1, s_2, \ldots, s_{T-1}, s_T$

$a_0, a_1, a_2, \ldots, a_{T-1}$

$r_1, r_2, \ldots, r_{T-1}, r_T$

$k < T$

$$q_\pi(s_k, a_k) \approx \sum_{\ell=0}^{T-k} \gamma^\ell r_{k+\ell} =: g_k$$

$$\nabla_\theta = -\frac{1}{T} \sum_{k=0}^{T-1} g_k \nabla_\theta \log \pi_\theta(s_k, a_k)$$

Policy Gradient Theorem

$$\nabla_\theta \mathop{\mathbb{E}}_{\pi_\theta} \sum_{k=0}^{\infty} \gamma^k R_{k+1} = C \mathop{\mathbb{E}}_{\substack{\pi_\theta \\ a \sim \pi_\theta(s)}} q_{\pi_\theta}(s,a) \underbrace{\nabla_\theta \log \pi(s|a)}_{0 \text{ is } \pi \text{ deterministic}}$$

REINFORCE : use sampling

π_θ MLP : $\mathbb{R}^d \xrightarrow{f_\theta} \mathbb{R}^M$ $\quad f_\theta(s)$ logits

$\pi_\theta(s) = \text{softmax}(f_\theta(s))$

Gumbel Distribution G $\quad g = g_0, \dots, g_{M-1}$ i.i.d Gumbel

$\text{argmax}(f_\theta(s) + g) \sim$ categorized $\pi_\theta(s)$

easily parallelizable

can sample multiple w/o replacement

$\text{argtop}_K(f_\theta(s) + g)$

Deep Q-Network

Recall Q-learning : $|\mathcal{S}| = N < \infty$, $|\mathcal{A}| = M < \infty$

action-value function $\quad q_\pi(s,a) = \mathbb{E} \sum_{k=0}^{\infty} \gamma^k R_{k+1} \mid S_0 = s, A = a_0$

$\mathbb{R}^{N \times M} \ni Q \approx q_*(s,a) = q_{\pi^*}(s,a)$ where π^* is optimal model

$[N] \times [M] \xrightarrow{Q} \mathbb{R} \quad \cong \quad Q \in \mathbb{R}^{N \times M}$

Now: $\mathcal{S} \subseteq \mathbb{R}^d$

$$\mathcal{S} \times [M] \xrightarrow{Q} \mathbb{R} \quad \cong \quad \underset{\substack{\cup\!\mid \\ \mathbb{R}^d}}{\mathcal{S}} \xrightarrow[\text{MLP}]{Q_\theta} \mathbb{R}^M$$

Setup: $Q \approx q_*$

Policy: evaluation: greedy: $\pi(s) = \operatorname{argmax} Q(s)$

training: ε-greedy: prob ε uniform

$1-\varepsilon$ greedy

Since $\mathcal{S} \xrightarrow{Q_\theta} \mathbb{R}^M$ MLP, need to supply loss

$s \in \mathcal{S}$, $a \in \mathcal{A}$, $Q(s,a) \approx \underbrace{q_*(s,a)}_{\text{unknown}}$

environment step $\longrightarrow$ reward $r \in \mathbb{R}$,

next state s' $\longrightarrow$ can be init if episode ended

$$\mathcal{O} = \begin{cases} 1 & \text{no reset} \\ 0 & \text{reset} \end{cases}$$

$$\boxed{r + \delta_\theta V_\theta(s')}$$

best approx return at state s'

$$V_\theta(s) = \max_{a \in A} Q_\theta(s,a)$$

$$\ell(s,a,r,\sigma,s';\theta) := \underbrace{\left(Q_\theta(s,a) - (r + \delta\sigma V_\theta(s'))\right)^2}_{\text{Temporal Difference (TD) error}}$$

Super unstable

$Q_\theta(s,a) - (r + \delta\sigma V_\theta(s'))$

prediction target ← no gradients from here

get V from stabilized model $V_{\theta'}$

Mnih et al : make θ' lag behind θ

regularly update $\theta' \leftarrow \theta$

← hyperparameter

Alternatively : Exponential Moving Average (EMA)

$\theta' \leftarrow (1-\beta)\theta' + \beta\theta$. $\beta \in [0,1]$ EMA weight

$Q \sim q_*$. So we're not attached to any policy.

Offline RL : train on past trajectory

<u>Replay Buffer</u>:

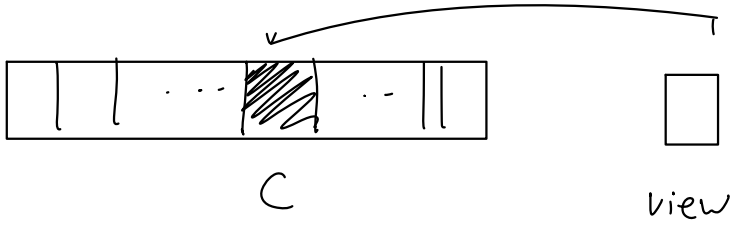
Record past trajectories. Fix capacity C $= 10^{4,5,6,7}$

Minibatch sampled from here.

Warmup : use uniform sampling - hyperparameter

After : 1 new step per minibatch

Implementation:



cursor : c step mod C

sampling : c forbidden

offset sample indices to get s

Introduction to Geometrical Deep Learning

Intro to Group Actions: Idea: $\mathcal{X}$, input, has symmetries

Definition:

Group: A group is a pair (G, μ)

- a set G (underlying set)
- binary operator $G \times G \xrightarrow{\mu} G$, $\mu(a,b) = ab$ — map of sets

a) associativity: $\forall a,b,c \in G$, $(ab)c = a(bc) = abc$

b) neutral / identity element $1 \in G$ s.t $1 \cdot a = a \cdot 1 = a$ (unique)

c) inverses $\forall a \in G, \exists a^{-1} \in G$, $aa^{-1} = a^{-1}a = 1$

↳ inverses are unique

* If you only have axiom a) ⟶ semigroup
* If you have a) + b) ⟶ monoid
* If you have commutativity ($ab=ba$) ⟶ abelian group

ex) Addition of integers $(\mathbb{Z}, +)$

ex) $n \in \mathbb{Z}_{>0}$, permutation group $= Aut([n]) = \{\text{bijection } [n] \xrightarrow{\pi} [n]\}$

$= S_n$

$\pi, \pi' \in S_n$, $\pi\pi' = \pi \circ \pi' = [n] \xrightarrow{\pi'} [n] \xrightarrow{\pi} [n]$

$i \in [n]$, $(\pi \circ \pi')(i) = \pi(\pi'(i))$

neutral: $id(i) = i$

inverse: $\forall i, \exists! j, \pi(i) = j, \pi^{-1}(j) = i$

π in torch. n-dim integer vector

$[\pi(0), \pi(1), \ldots, \pi(n-1)]$. $[n] \xrightarrow{\pi} [n] \xrightarrow{x} X$

$\pi^{-1} = \text{argsort}(\pi)$, $X^{\pi} = X[\pi]$

Definition:

Right Group Action: group G, set X, $X \times G \xrightarrow{\alpha} X$, $\alpha(x, g) =: xg$

such that $(xg)h = x(gh)$

ex) Any group G, any set X (do nothing)

$X \times G \xrightarrow{(x,g) \mapsto x} X$ trivial group action

ex) Action of group G on underlying set

$\mu = \alpha$ right translation action

Conjugation: $(g, h) \longmapsto h^{-1} g h$

set ← (g, h) → group

ex) $X = (R^A)^{\mathbb{Z}} = \text{Hom}(\mathbb{Z}, \mathbb{R}^+) \circlearrowright (\mathbb{Z}, +)$

ω $\quad$ ψ

$\underline{x}$ $\quad$ K

$\mathbb{Z} \xrightarrow{\underline{x}} \mathbb{R}^d$

Action by translation: $(\underline{x}, k) = (i \in \mathbb{Z} \longmapsto x_{i+k})$

ex) $X = \mathbb{R}^{m \times n}$, $G = S_m$, action: permute rows

Def: G groups, right G-sets X, Y

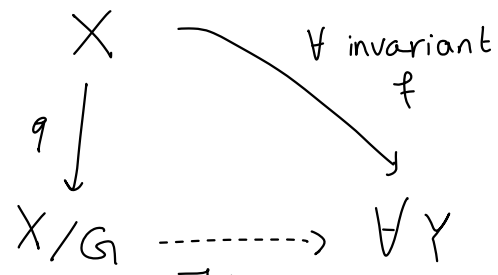
map of sets $X \xrightarrow{f} Y$ is (G-) equivariant if

$\forall x \in X, a \in G, \; f(xa) = f(x)a$

Special case: $Y \circlearrowleft G$ trivial. Then invariant $f(xa) = f(x)$

Def: $X \circlearrowright G$ quotient is [universal invariant map]

$X \xrightarrow{q} X/G$ invariant s.t



i.e $\forall$ sets Y

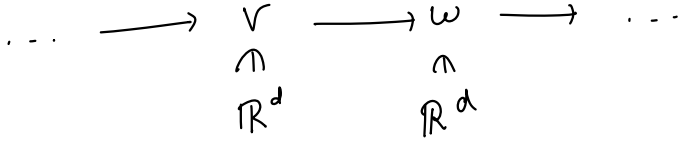
$\mathrm{Hom}(X/G, Y) \xrightarrow[\cong]{q\circ} \mathrm{Hom}(X, Y)$ invariant $\qquad f = g \circ q$

Def: $x \in X$, orbit $\{xg : g \in G\} =: xG$

can let $X/G = \{xG : x \in X\} \xleftarrow{q(x) = xG} X$

ex) $(\mathbb{R}^d)^{\mathbb{Z}} \circlearrowright (\mathbb{Z}, +)$

can let X/G be set of graphs



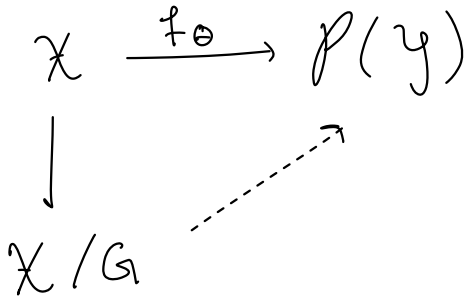
ex) $\mathbb{R}^{m\times n} \circlearrowright S_m$, can let $X/G = \{$multiset of m n-dim vectors$\}$

Geometric Deep Learning

Input space $\mathcal{X}$ has a G-action

$\mathcal{Y}$ also (trivial)

GDL: want equivariant / invariant models



Why?

Automate data augmentation X for g_θ

$=$

XG for f_θ

Parameter Sharing: equivariant architectures need way fewer architectures.

CNNs

$\mathcal{X} \circlearrowright G$, $\mathcal{X} \subseteq \mathbb{R}^{s \times d}$ — d - feature dim, s - spatial dim

$G = \mathbb{Z}$, action: cyclical permutation

$X \in \mathcal{X}$, $k \in G = \mathbb{Z}$

$(X \cdot k)_i = X_{\overline{i-k}}$ where $\overline{i-k} = i - k \bmod s$

$\mathcal{X} \xrightarrow{f_\theta} P(\mathcal{Y})$ equivariant?

$\mathbb{R}^{s \times d} \xrightarrow{A^{(0)}} \mathbb{R}^{s \times n_1} \xrightarrow{a} \mathbb{R}^{s \times n_1} \xrightarrow{A_1} \mathbb{R}^{s \times n_2} \longrightarrow \dots$

vector spaces

$A(x) = L(x) + B$ affine map

linear bias

$X \in \mathbb{R}^{s \times d}$, $\text{flatten}(X) \in \mathbb{R}^{sd}$

$X \in \mathbb{R}^{sd}$, $\text{unflatten}(x) \in \mathbb{R}^{s \times d}$

$L(x) = \text{unflatten}(\text{flatten}(x) L)$, $L \in \mathbb{R}^{sd \times sn_1}$

Q: when is this affine map translation equivariant?

Note: $A(0) = B$ equivariant since $A(0 \cdot k) = B \cdot k \quad \forall k \in \mathbb{Z}$

Q: when is this affine map equivariant?

Consider $A(0) = B$, equivariant if $A(0 \cdot k) = B \cdot k \quad \forall k \in \mathbb{Z}$

$\Rightarrow A(0) = B \cdot k$

Now, $A(x \cdot k) = L(x \cdot k) + B$

$A(x) \cdot k = (L(x) + B) \cdot k$

$= L(x)\, k + B \cdot k$

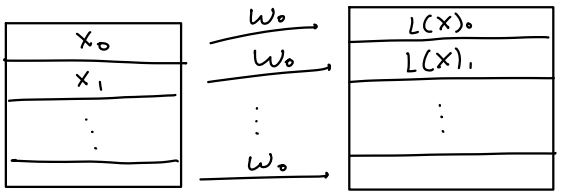
$= L(x) \cdot k + B$

So, A equiv $\Longleftrightarrow$ L equiv + $B \cdot k = B \quad \forall k$

$L(x) = \text{unflatten}(\text{flatten}(x)\, L)$. $L \in \mathbb{R}^{sd \times sn_1}$

examples of equivariant linear maps $\mathbb{R}^{s \times d} \longrightarrow \mathbb{R}^{s \times n_1}$

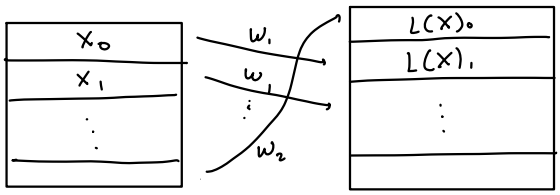
- $W_0 \in \mathbb{R}^{d \times n_1}$. $L(x)_i = X_i\, W \longrightarrow$
- $W_1 \in \mathbb{R}^{d \times n_1}$. $L(x)_i = X_{i-1}\, W$



In general:

$w_j \in \mathbb{R}$, $L(x)_i = X_{j-1}\, W$

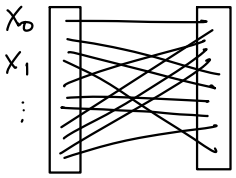
$0 \le j < s$



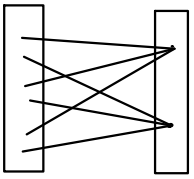
<u>Theorem</u>: A linear map $\mathbb{R}^{s \times d} \xrightarrow{L} \mathbb{R}^{s \times n_1}$ is translation-equivariant iff $\exists w_0, \ldots, w_{s-1} \in \mathbb{R}^{d \times n_1}$, $L(x)_i = \sum_{j=0}^{s-1} X_{i-j}\, w_j$.

This map is a <u>convolution</u>

So now we have full CNNs



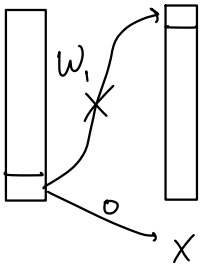
more efficient:



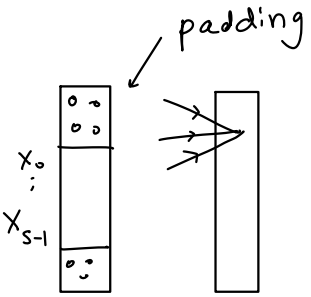
(here, kernel size = 1)

e.g. only

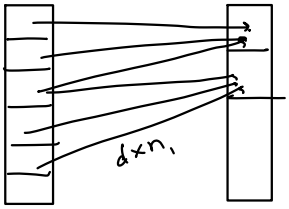
$\underbrace{W_{-1}, W_0, W_1}_{\text{stacked into } \mathbb{R}^{3\times d\times n_1}}$ ← kernel

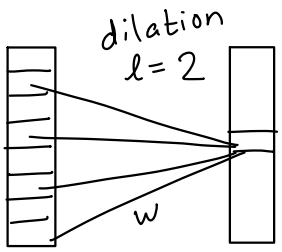


Confusing!
In implementation no rollover

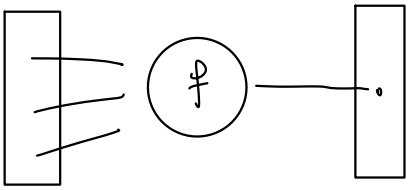


Convolution operations





Pooling: use invariant operation f



want to aggregate neighboring components, so we want $q > 1$

Dropout:

A neuron is the part of the network giving a hidden feature,

→ a column $W_{ij}^{(\ell)} : i = 0, \dots, n_\ell - 1$ and a bias component $b_j^{(\ell)}$ given layer index ℓ and hidden feature index $0 \leq j < n_\ell$

Normalization Layers

Initialization: want $\mathbb{E}X = 0$, $\mathbb{V}X = 1$ in features

So as to avoid vanishing or exploding variance,

Set initial parameters to this effect.

To keep $\mathbb{V}X \approx 1$. normalize between layers

need estimated $\mathbb{E}X$, $\mathbb{V}X$

when you normalize a dataset: $\mathbb{E}X \approx \hat{\mathbb{E}}X = \frac{1}{n}\sum_{x_i \in \mathcal{D}} x_i$

$$\mathbb{V}x \approx \tilde{\mathbb{V}}x = \sqrt{\frac{1}{n-1}\sum_{x_i \in \mathcal{D}} (x_i - \hat{\mathbb{E}}x)^2}$$

Batch Normalization

also hidden features

in training, estimate $\mathbb{E}X$, $\mathbb{V}X$ on minibatches

$$X_i \leftarrow \frac{X_i - \hat{\mathbb{E}}X_i}{\hat{\mathbb{V}}X_i + \varepsilon}$$

Architecture Search: too costly

depth: if layer can be identity, then too big depth, no problem

But with normalization, can't be the identity.

$$X_i \leftarrow \frac{X_i - \mathbb{E}X_i}{\mathbb{V}X_i + \varepsilon}\lambda_i + M_i$$

scaling; offset parameters; these are trainable

↳ these parameters can 'undo' the normalization, so they can be the identity.

in eval: want better $\mathbb{E}X$ + $\mathbb{V}X$ estimates.

Keep moving averages:

$$\underset{\text{stored}}{\mathbb{E}X} \leftarrow (1-\mu)\underset{\text{stored}}{\mathbb{E}X} + \mu\underset{\text{minibatch}}{\hat{\mathbb{E}}X}$$

Batch Norm: normalizes each component. needs fixed shape
can't work with input of varying shape. eg. text

Layer Normalization: don't normalize each component,
normalize values in layer ~ normalize subtensor of feature tensor.

(n, d) point set
↙ ↘ dim → normalize
# of points

In MNIST
input $d = 1$
$(28, 28, 1) \xrightarrow{CNN} (28, 28, out\_dim)$
normalize over this

$$X \leftarrow \frac{X - \mathbb{E}x}{\varepsilon + \sqrt{\frac{1}{d}\sum (x - \hat{\mathbb{E}}x)^2}} \gamma + \mu$$
trainable

RMS Norm: Can dispense of mean offset w/o much performance loss

$$X \leftarrow \frac{X}{\varepsilon + \sqrt{\frac{1}{d}\sum x^2}} \gamma$$

IMDb dataset

input: movie reviews
output: good/bad

How to process text input?

Tokenization

Given text, split it up into atomic pieces

So what should be a token?

- bits
- letter — character — 8-bit pieces / unicode
- word — seperation — spaces / punctuations
  - components

→ for today, we'll use word tokenization with basic seperation, all lowercase, unicode normalization.

Inputs: $d_0, \ldots, d_{N-1} \in$ str documents

$d_i = (t_i^0, \ldots, t_i^{s_i - 1})$ ← tokens

s_i - varying length

Vocabulary $\mathcal{V} = \{ t_i^j : 0 \le i < N, 0 \le j < s_i \}$

Give it any ordering

$V = (t_k : 0 \le k < v)$

$v = |V|$ vocab size

Wikipedia, IMDb train

$v >>> 0$ 25000 entries

When testing and you encounter a word not in $\mathcal{V}$, we use [UNK] → $\mathcal{V}$

$d_i \in$ str $\quad \forall i, j, \exists k \quad t_i^j = t_k$

$(t_i^0, \ldots, t_i^{s_i - 1})$

$(k_0, \ldots, k_{s_i - 1}) \in [v]^{s_i}$

Tokenization is preprocessing: done prior to training

Assume this is done

$$\mathcal{D} = \{ (\underline{t}_i, c_i) : i \in I \}$$

Sequences of Input:

$\mathcal{X}_0$ elementwise feature space

a) discrete $\mathcal{X}_0 \cong [v]$ eg. vocabulary, board game states

b) continuous $\mathcal{X}_0 \subseteq \mathbb{R}^d$ eg. sound, stocks, point cloud

$$L\mathcal{X}_0 = \coprod_{s \geq 0} L_s \mathcal{X}_0 \quad . \quad L_s \mathcal{X}_0 = \mathcal{X}_0^{\times s} := \underbrace{\mathcal{X}_0 \times \cdots \times \mathcal{X}_0}_{s}$$

S_n permute group action (simplifcation)

Text Input after tokenization

$$\mathcal{X} = L[v] = \coprod_{s \geq 0} L_s [v]$$

$$= \coprod_{s \geq 0} [v]^{\times s}$$

$$\underline{t} = (k_0, \ldots, k_{s-1}) \quad . \quad k_j \in \{0, \ldots, v-1\}$$

For each $0 \leq k < v$, introduce learnable vector.

$\underline{e}_k \in \mathbb{R}^d$, d - embedding dim - (hyperparam)

We'll use $d = 1$

2013 : Word2Vec - $d = 300$

2017 : transformer - $d = 512$

Now : $d = 10{,}000$s

Package Vectors $\{\underline{e}_0, \ldots, \underline{e}_{V-1}\}$ into embedding matrix $E \in \mathbb{R}^{V \times d}$

$\underline{t} \hookrightarrow X = (\underline{e}_{K_0}, \underline{e}_{K_1}, \ldots, \underline{e}_{K_{S-1}}) \in \mathbb{R}^{S \times d}$

$= E[\underline{t}]$

Minibatch $\underline{t}_0, \ldots, \underline{t}_{N'-1}$ $\quad s' := \max\{s_i\}$

$s_0, \ldots, s_{N'-1}$ $\quad$ sequences shorter than s' padded by special token [PAD]

$X \in \mathbb{R}^{N' \times s' \times d}$

instead use mask $M \in \mathbb{B}^{N' \times s'}$

$\cong \{\text{true}, \text{false}\}$

to get binary classification logits

$X \in \mathbb{R}^{N' \times s' \times d} \xrightarrow{\text{pool: mean along } s'} \mathbb{R}^{N' \times d} \xrightarrow{W \in \mathbb{R}^{d \times 1}} \mathbb{R}^{N'}$

pooling: invariant

Efficiently handle sequences of varying length

$\mathcal{D} = \{(\underline{t}_i, c_i) : i \in I\}$, $\underline{t}_i \in L_{s_i}[V]$

$\underline{t} =$ torch.cat($\{\underline{t}\}$)

lengths / indptr

↓

cumsum(lengths)

$\underline{t}_i = \underline{t}[\text{indptr}[i] : \text{indptr}[i+1]]$

Transformers

$= \underline{t} \in L[v]$

"This is a good movie" $\longmapsto$ (0,1,5,2,3,4)

"This is a bad movie" $\longmapsto$ (0,1,5,2,3,4)

"This is a good movie about a bad, bad person" $\longmapsto$ (0,1,2,3,4,7,2,6,6,8)

undividable

Tokenization: split text to pieces viewed as atomic

Now: words

vocabulary: ordered: dict: str $\longmapsto$ int ✓

```
def token_id (vocabulary: dict, word: str) -> int:
    if word in vocabulary
        return vocab[word]
    vocabulary[word] = len(vocabulary)
```

Vocab:

0 this, 1 is, 2 a., 3 good, 4 movie
5 not, 6 bad, 7 about, 8 person,
9 [unk]

embedding: $E \in \mathbb{R}^{v \times d}$, $E[\underline{t}] = (E_{t_0}, \dots, E_{t_{s-1}}) \in \mathbb{R}^{s \times d}$

So far, we've viewed sentences as sets, but really they have linear order

Structure. Linear order: $0 \to 1 \to 2 \to 3 \to 4$

1D: CNN on X

"This movie about a bad, bad person is good"

I'm going to the bank ($\in \mathbb{R}^d$) for money

for fish

models need contextual token embeddings

$\mathbb{R}^{s \times d} \to \mathbb{R}^{s \times d} \to \mathbb{R}^{s \times d} \to \dots \to \mathbb{R}^{s \times d} \xrightarrow{\text{pool}}$

$X^{(\ell)} \in \mathbb{R}^{s \times d}$, $W \in \mathbb{R}^{d \times d}$

$X^{(\ell)} W$ in position change $X^{(\ell)}_i = \sum_j W_{ij} (XW)_j$

$W_{ij} \in \mathbb{R}$: how much position j matters on level ℓ for pos i on level $\ell+1$

Transformers address this using attention

d : embedding dim

d_K : key dim

d_v : value dim

key, query parameter matrices $W_K, W_Q \in \mathbb{R}^{d \times d_K}$

$K = X^{(\ell)} W_K$ key

$Q = X^{(\ell)} W_Q$ query

$(X^{(\ell)} W_Q)(X^{(\ell)} W_K)^T$

matrix i,j entry dot product of Q_i, K_j

Suppose $VX \approx 1$, $W_K \approx 1$, $W_Q = 1$

$w_{ij} = K_i Q_j$, $V w_{ij} \approx d_K$

$w_{ij} \leftarrow \frac{w_{ij}}{\sqrt{d_K}}$, then $w_{ij} \leftarrow \text{Softmax}(w_{ij} \cdot j)$

Softmax: probabilities in multiclass classification → single class at once

multilabel → multiple classes at once

tags or ~~ports~~

in well trained transformers, few positions get most attention

Sigmoid attention: not so much better

One update on i : few j

Multiple attention heads

residual connection

Additive updates: $X^{(\ell)} = X^{(\ell)} + \sum_j w_{ij} X^{(\ell)} W$

New hyperparameter. number of attention heads a

convention: $d_K = \frac{d}{a} = d_v$

Attention head $0 \le h < a$

Input: $X^{(\ell)} \in \mathbb{R}^{s \times d}$

1. Normalize RMSNorm
2. $W^{(\ell,h)}$, $W_Q^{(\ell,h)} \in \mathbb{R}^{d \times d_k}$

   value matrix $W_V^{(\ell,h)} \in \mathbb{R}^{d \times d_v}$

   output matrix $W_O^{(\ell,h)} \in \mathbb{R}^{d_v \times d}$
3. Output

$$MHSA(X^{(\ell)}) = X^{(\ell)} + \sum_{h=0}^{a-1} Dropout\left(\sum_{j=0}^{s-1} w_{ij}^{(\ell,h)} \underbrace{RMSNorm(X^{(\ell)})}_{\tilde{X}} W_V^{(\ell,h)} W_O^{(\ell,h)} \right)_i$$

where

$$w_{ij}^{(\ell,h)} := Softmax\left(\frac{(\tilde{X}^{(\ell)} W_Q^{(\ell,h)})(\tilde{X}^{(\ell)} W_K^{(\ell,h)})^T_j}{\sqrt{d_k}} \right)_{ij}$$

$=: Y^{(\ell)}$

This is an attention block

: vectors at diff positions communicate

Feed Forward Block: position by position applies FNN

$$FF(Y^{(i)}) = Y^{(i)} + Dropout\left(a\left(RMSNorm(y^{(\ell)}) W_1^{(\ell)} \right) W_2^{(\ell)} \right)$$

$W_1 \in \mathbb{R}^{d \times d}$

$W_2 \in \mathbb{R}^{d_1 \times d}$

Transformer

FNN

MHSA

FNN

MHSA

Positional Information

Need to give system positional information

- $d \leftarrow d+1$ used in DRL — use as extra component position
  modify token embedding

- e_t token embeddings

  pos i. $p_i \in \mathbb{R}^d$ position embedding

Input: $e_t + p_i \in \mathbb{R}^d$ ← absolute positional embedding

Fixed absolute positional embedding: $p_{i,k} =$

$$\begin{cases} \sin\left(\frac{i}{10000^{2k/d}}\right) & \text{if } 2 \mid k \\ \cos\left(\frac{i}{10000^{2(k-1)/d}}\right) & \text{if } 2 \nmid k \end{cases}$$

Also can make

$p_i \in \mathbb{R}^d$

trainable

$\Downarrow$

$p_{i,k} \in \mathbb{R}$

$k = 0, \ldots, d-1$

↓ d extra parameters for allowed position, very large can sparse training signal

In BERT ('19). 512 position vectors

$$A = \underbrace{(XW_Q)}_{S \times K} \cdot \underbrace{(XW_K)^T}_{S \times K}$$ ← matrix product

$\rightarrow$ S^2 dot products $\rightarrow$ $O(n^2)$

attention has quadratic number of ops in sequence length.

Flash Attention. ~ linear complexity, $O(n)$

Relative position embeddings

What matters is not pos $i \in [S]$ but differences $i' - i$

$V = XW_V$

Additive change to attention $\longrightarrow \left(\text{softmax}\left(\frac{QK^T + A_K}{\sqrt{d_K}} \right), A_V \right) V$

$A_K, A_V \in \mathbb{R}^{S \times S \times d}$

Prev: $A \in \mathbb{R}^{S \times S}$, $V \in \mathbb{R}^{S \times d}$ $\longrightarrow$ $AV \in \mathbb{R}^{S \times d}$

i-th row: $\sum_{j=0}^{S-1} a_{ij} V_j$ ($a_{ij} \in \mathbb{R}$, $V_j \in \mathbb{R}^d$)

Now: $A \in \mathbb{R}^{S \times S \times d}$, $a_{ij} \in \mathbb{R}^d$ $\longrightarrow$ ith row: $\sum_{j=0}^{S-1} a_{ij} \odot V_j$

$\odot$ — comparative multiplication

People are using one trainable vector for each position difference $a_{ij} = a_{i-j} \in \mathbb{R}^d$ ← text has a linear structure

We can also use transformers on graphs:

Graphormer: a_{ij} edge-level data, token embedding: node info

Rotary Position Embeddings:

multiplicative change

$$AV = \text{Softmax}\left(\frac{QRK^T}{\sqrt{d_K}}\right)V$$

$$R \in \mathbb{R}^{S \times S \times d_K \times d_K}$$

$$(QK^T)_{ij} = \sum_{k=0}^{d_K-1} Q_{ik} \cdot K_{jk}$$

$$(QRK^T)_{ij} = \sum_{k=0}^{d_K-1} \sum_{l=0}^{d_K-1} Q_{ik} R_{ijkl} K_{jl}$$

$R_{ij} =$

$$\begin{pmatrix} \rho_1 & & & 0 \\ & \rho_2 & & \\ & & \ddots & \\ 0 & & & \rho_{d_K/2} \end{pmatrix}$$

$$\theta_k = 10000 - \frac{2(k-i)}{d_k}, \quad \rho_k = \begin{pmatrix} \cos(j-i)\theta_k & -\sin(j-i)\theta_k \\ \sin(j-i)\theta_k & \cos(j-i)\theta_k \end{pmatrix}$$

In implementation: don't form R, just add vertices.

This method is called RoPE

Another method called NoPE

Generative Language Modeling

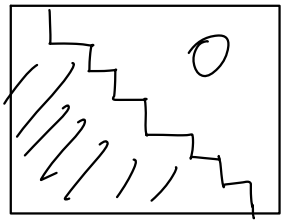
input: potential sequence

output: distribution of next token

Caused Attention

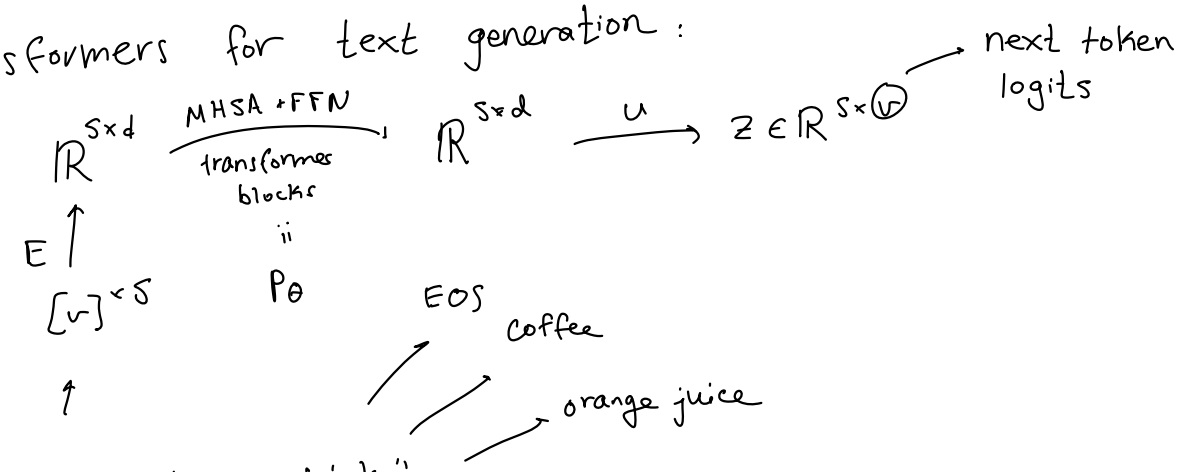
QK^T attention logits

$+-\infty$ if $j > i$

$A =$ 

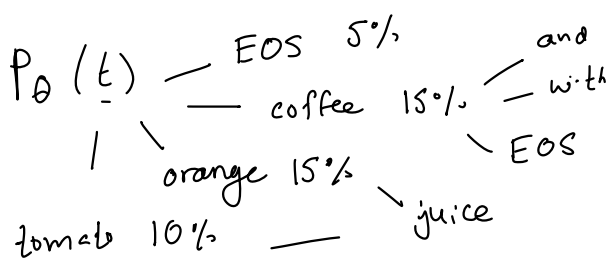
GPT

Transformers for text generation:



$z[i] \in \mathbb{R}^V$

Softmax $(z[i])[j]$ prob that after first i tokens, sequences continued by token ID j



generation
inference

beam search: gather the best K options step by step → beam width

—

Training: take wikipedia, common crawl, etc.

prediction $p_\theta[\underline{t}[:-1]] \in \mathbb{R}^{s-1 \times v}$

target $p_\theta[t]$ → ith position

} use cross entropy loss

GPT: Generative Pretrained Transformer

GPT1: after pretraining, can fast finetune — summarize, translate, Q&A

	year	# blocks	d	d_h	# params
GPT 1	2018	12	768	3072	117 M
GPT 2	2019	48	1600	N/A	1542 M
GPT 3	2020	96	12288	49152	175 B
GPT 4/5	?	?	?	?	?

GPT 2: emergent abilities

nontrivial few shot, zero shot abilities ⟶ Is this review good?

↳ two → three, one → two, four → FIVE

↳ movie review → good, movie review → BAD

GPT 3: general media reach, good text generation

↳ training: $10M, 85000 kg CO_2 → road trip to the moon and back in a car

new car in Europe: 700K KM, 435,000 miles

Instruction Tuning: model generates text based on general linguistic knowledge

→ often has biases

↳ Tell model to be nice ← Reinforcement learning

policy gradient: $\nabla_\theta G \propto \mathbb{E}_{s,a\sim\pi} G_s \nabla_\theta \log \pi(a|s)$

gain ↑ expected return

REINFORCE: take sample: $g_s \approx \sum \gamma^t r_{reward}$

↳ Actor-critical: $G_{s,a} \approx Q_\varphi(s,a)$

Train w/ policy gradient

Train Q_φ Bellman update $r + \gamma V(s')$

Monte Carlo: $Q(s,a) \longleftarrow \sum \gamma^k r_{et}$

How to get rewards?

1. Human annotators write full conversations
   Supervised finetuning
2. Generate multiple answers to questions
   Have annotators rank answers
3. Take pretrained model, use as base for rank predictor:
4. DRL using r as reward
5. Go to 2.

annotator

A1 A2

$\Longleftrightarrow r(A1) > r(A2)$

GPT

pretrained models : foundational models

LLaMA ⇝ Vicuna

Transformer : very big matrices : $E, V^T \in \mathbb{R}^{V \times d}$

$W_{Q,K,V,O} \in \mathbb{R}^{V \times d}$

$W_1, W_2^T \in \mathbb{R}^{d \times d_K}$

LoRA: low rank adapters

↳ Have big pretrained matrix. $W_{old} \in \mathbb{R}^{d \times d}$

↳ Now finetune : shorter training

$W_{new} = W_{old} + AB$, $A, B^T \in \mathbb{R}^{d \times r}$ ← rank

W_{old} : freeze

AB : ↳ train

Quantization : numpy : float 64

GPU : float 32

float 16

4 bits

1.5 bits

package: bits and bytes

Word Vectors

Model ← text : models not embedded

meaning of word ~ statistics on word context

2 words in same context ⇒ same meaning

" tranform semantic relations into linear ones"

Corpus = Wikipedia (+ Common Crawl)

hyperparameter ↓

2 words are in same context := distance ≤ c

↳ context = windows of c consecutive words

Metric Learning.

notion of similarity

map entries ⟶ metric space $\mathbb{R}^d$ ← hyperparam

similar things are close

$\mathcal{X}$: word or $\mathcal{X}$: word

$\mathcal{Y}$: context $\mathcal{Y}$: word

R = word ∈ context R = in same context

model = vectors

2 spaces $\mathcal{X}, \mathcal{Y}$

relation $R \subseteq \mathcal{X} \times \mathcal{Y}$

model : $\mathcal{X} \xrightarrow{f_\theta} \mathbb{R}^d$; $\mathcal{Y} \xrightarrow{g_\psi} \mathbb{R}^d$

$(x, y) \in \mathcal{X} \times \mathcal{Y}$

$(x, y) \in R \sim d(f_\theta(x), g_\psi(x))$

d ? ⟶ Euclidean : $\| f_\theta(x) - g_\psi(x) \|_2$

cosine similarity : $\frac{f_\theta(x) \cdot g_\psi(x)}{\| f_\theta(x) \| \, \| g_\psi(x) \|}$

dot product : $f_\theta(x) \cdot g_\psi(x)$

Negative Sampling:

- anchor $x \in \mathcal{X}$ gradient only through $f_\theta(x)$
- positive sample $y^{(p)} \in \mathcal{Y}$ $(x, y^{(p)}) \in R$
- negative sample $y_1^{(n)}, \dots, y_k^{(n)} \in \mathcal{Y}$. $(x, y_i^{(n)}) \notin R$

$f_\theta(x)$ $g_\psi(y^{(p)})$ $g_\psi(y_i^{(n)})$ $\in \mathbb{R}^d$

Loss? $-f_\theta(x)\, g_\psi(y^{(p)}) + \sum g_\psi(y_i^{(n)}) \cdot f_\theta(x)$

logits $f_\theta(x) \cdot g_\psi(y^{(p)})$, $f_\theta(x) \cdot g_\psi(g_i^{(n)})$

classification: which sample is in R with x
y

↳ then we use cross entropy ... that's it!

Word2Vect
FastText
GLoVE
} all use this same idea ... can then finetune to your task

facebook