PERCEPTRON (X, Y, T)

$\theta, \theta_0 = 0$
for t in T:
for $i = 1 \ldots n$:
if $y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \le 0$:
$\theta = \theta + y^{(i)} x^{(i)}$
$\theta_0 = \theta_0 + y^{(i)}$

return θ, θ_0

* If prediction doesn't match label, adjust

Predict $= \mathrm{sign}(\theta \cdot x + \theta_0)$

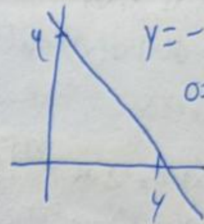
Avg Percep (X, Y, T):
Same except if $y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \le 0$:
$\theta = \theta + y^{(i)} x^{(i)}$
$\theta_0 = \theta_0 + y^{(i)}$
$\bar\theta = \bar\theta + c\, y^{(i)} x^{(i)}$
$\bar\theta_0 = \bar\theta_0 + c\, y^{(i)}$
$c = (-1/nT$ for each T
return $\bar\theta, \bar\theta_0$

* Avg all your changed thetas, inc incorrect ones

$$\gamma = \min \left[\frac{y^{(i)}(\theta^T x^{(i)})}{\|\theta\|} \right]$$ for ith - min margin

$R = \max \|\vec{x}\|$

Max mistakes $= \left(\frac{R}{\gamma}\right)^2$



$y = -x + 4$

$0 = -y - x + 4$

$(-1, -1)$ $\downarrow$ θ_0

$= \theta$

Geometric margin - if incorrect class, fied, can't be calc'd

θ indirec of pos pb

Feature Encoding

- one hot
- normalization
- learning algeval

$\ldots \phi(x) \to$ more dim, more fit to training data $\to$ more margin

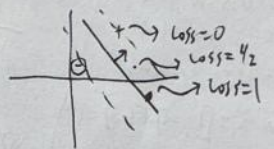
λ rules / η

* As $\lambda \uparrow$ $\|\theta\| \downarrow$, $\gamma \uparrow$
As $\lambda \downarrow$ θ goes to least mean squared error, $\gamma \downarrow$
As $\eta \downarrow$, gradient change decreases
As $\eta \uparrow$, gradient change increases.

SVM objective: $J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{Loss}_h (y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) + \frac{\lambda}{2} \|\theta\|^2$

$$\mathrm{Loss}_h(y(\theta \cdot x + \theta_0)) = \max \begin{cases} 0 \\ 1 - y(\theta \cdot x + \theta_0) \end{cases} = \begin{cases} 1 - y(\theta \cdot x + \theta_0) & \text{if } y(\theta \cdot x + \theta_0) \le 1 \\ 0 & \text{ow} \end{cases}$$

$$\text{Alt H Loss} = \begin{cases} 1 - \frac{\gamma(x, y, \theta, \theta_0)}{\gamma_{ref}} & \text{if } \gamma(x, y, \theta, \theta_0) < \gamma_{ref} \\ 0 & \text{ow} \end{cases}$$



$\gamma = \frac{1}{\|\theta\|}$

Pegasos (X, Y, λ, T)

$\theta = 0$
for t in T:
select random i:
$\eta = 1/t$
if $y^{(i)} \theta \cdot x^{(i)} \le 1$:
$\theta = (1 - \eta\lambda)\theta + \eta y^{(i)} x^{(i)}$ $\leftarrow$ $\theta - \eta \nabla \Rightarrow \nabla = -xy + \lambda\theta$
else
$\theta = (1 - \eta\lambda)\theta$

* GD: $x_1 \leftarrow x_0 - \eta \nabla_x f(x_0)$

NGD: estimate ∇ by $\frac{f(x_0 + \delta^i) - f(x_0 - \delta^i)}{2\delta}$

Regression

$f(x, \theta, \theta_0) = \theta \cdot x + \theta_0 \to$ predict

Learning: Loss $= \frac{(y^{(t)} - \theta \cdot x^{(t)})^2}{2} + \frac{\lambda \|\theta\|^2}{2}$ in $J = \frac{1}{n} \sum L(x, y, \theta, \theta_0) + \lambda \|\theta\|^2$

set $\theta = 0$
Random select $t \in [1 \ldots n]$
$\theta^{k+1} = \theta^k + \eta_k \underbrace{(y^{(t)} - \theta \cdot x^{(t)}) x^{(t)}}_{\nabla\theta}$

Ridge Regression (w/ regularization): w/ reg $\to$ $\theta^{k+1} = (1 - \lambda\eta_k)\theta^{(k)} + \eta_k (y^{(t)} - \theta \cdot x^{(t)}) x^{(t)}$

- Structural error - when you have linear model for poly solution
- Estimation error: test error. For less dim in feature vectors, $\downarrow$
- Regularization prevents overfitting

Recommender: Predict $Y_{ai} \to u_{a} v_i = X_{ai}$

$$J = \sum_{i:(a,i) \in D} \frac{(Y_{ai} - u^{(a)} v^{(i)})^2}{2} + \frac{\lambda}{2} \|u^{(a)}\|^2 \text{ for optimizing } u^{(a)}$$

$u_a v_i$ thou at $b v_i$

known y values

$\frac{\lambda}{2} \|v^{(i)}\|^2$ for optimizing $v^{(i)}$

$\to$ Alternating Optimizing

w/ offsets: $J(u, v) = \sum \frac{(Y_{ai} - u^{(a)} v^{(i)} - b_u^{(a)} - b_v^{(i)})^2}{2} + \frac{\lambda}{2} \sum \|u_a\|^2 + \frac{\lambda}{2} \sum \|v_i\|^2$

Matrix Shit - Learning

Y_{ai} - givens
B is every v that appears in a_i
Z is all Y_{ai} in a column
To solve for optimal U's: set $\nabla_u J = 0$
$0 = -(B)^T (Z - BU_a) + \lambda \cdot U_a$
or $0 = -(B)^T (Z - BU_a - b_v - b_u^{(a)}) + \lambda u^{(a)}$

all b_v's that show up in a, i

Ex
$Y_{12} = 1$
$U = [u, 1]$
Solve for $[V_1, V_2]$
$J(U, V) = \frac{1}{2} \sum [Y_{ai} - [UV]]^2 + \frac{\lambda}{2}(V_1^2 + V_1^2)$
$0 = \nabla J_v = \frac{1}{2}(2 \cdot (-U)[Y_{ai} - UV]) + 2\lambda V_1$
$0 = \nabla J_{v_1} = \lambda V_1 = 0$
$V_1 = 0$
$0 = \nabla J_{v_2} = -u[1 - uV_2] = 0$
$V_2 = \frac{1}{u}$

Kernels $\to \phi(x) = [x_1 x_2, x_1 x_2]^T$

- $K(x,x') = \phi(x)\cdot\phi(x') = K(x',x)$ • $K(x,x') = f(x)K'(x,x')f(x')$
- $K(x,x) \geq 0$ • Radial Basis $\to e^{-\|x-x'\|^2/2}$

- Predict x: $\theta\cdot\phi(x)+\theta_0 = \sum a_i y^{(i)}[K(x^{(i)},x')+1]$
  or $\theta\cdot\phi(x) = \sum a_i y^{(i)}[K(x^{(i)},x')]$
  $\theta\cdot\phi(x) = \sum a_i [K(x^{(i)},x)]$

Perceptron $\alpha_t = [0..0]$ $\to$ w/ offset

If $y^{(t)}\left(\sum_{i=1}^n \alpha_i y^{(i)}[K(x^{(i)},x^{(t)})+1]\right) \leq 0$

return αs

$\alpha_t \leftarrow \alpha_t + 1$

Signed for perceptron

Ex. $a[0,0,0]_i$

$K = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ $i \leftarrow \alpha_i s$

$Y = [1,1,-1]$

Building $\alpha \to$

$t=1$ $y^{(1)}(0\cdot y^{(1)}K(x^1,x^1)$
$+0\cdot y^{(2)}K(x^1,x^2)$
$+0\cdot y^{(3)}K(x^1,x^3))$
$=0$

$\alpha_1 \pm 1$ $\alpha = [1,0,0]$

$t=2$ $y^{(2)}(1\cdot y^{(1)}K(x^1,x^2)+$
$0\cdot y^{(2)}K(x^2,x^2)+$
$0\cdot y^{(3)}K(x_2,x_3))$
$=$
$1(1\cdot1\cdot1) = 1$ ✓

$t=3$ $y^{(3)}(1\cdot y^{(1)}K(x^1,x^3)+$
$0\cdot y^{(2)}K(x^2,x^3)+$
$0\cdot y^{(3)}K(x^3,x^3))$
$= -1(1\cdot1\cdot0+0) = 0$

$\alpha_3 \pm 1$ $\alpha = [1,0,1]$

Regression: Loss function is to the effect of

return αs

$J(\theta) = \frac{1}{n}\sum_{t=1}^n \frac{(y^{(t)}-\theta\cdot\phi(x^{(t)}))^2}{2} + \frac{\lambda}{2}\|\theta\|^2$

* Set gradient to 0

$\vec{\alpha} = \vec{y}(n\lambda I + K)^{-1}$ $\to$ $y^{(t)}-\theta\cdot\phi(x^{(t)}) = n\lambda\alpha_t$

$\to$ Gram Matrix

$\nabla_\theta J(\theta) = -\lambda\sum\alpha_t\phi(x^{(t)}) + \lambda\theta = 0$

CNN – Filters, pooling

where things are — what is in each region

CNN w/ 100 units; 10 filters each w/ 5 inputs w/o bias $\to$

100 units $\times$ 10 feature maps = 1000 output

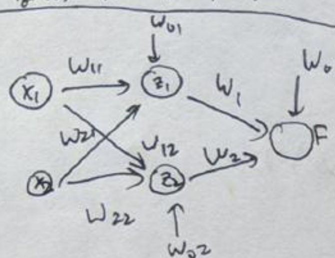
$\to$ needs $5\cdot10+10$ weights

Act funcs — Loss fxns

hidden units, nonlinearity: • Relu • tanh

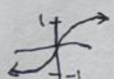
outputs:
- linear — Quadratic
- Sigmoid (prob) — NLL
- softmax (prob) — NLL
  $\downarrow$ multiclass

NN

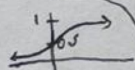


$z_1 = [w_{11}, w_{21}]\cdot[x_1, x_2] + w_{01}$

Activations
- $f(z) = z \to$ Linear
- $f(z) = \text{Relu}(z)\begin{cases} 0 & z\leq 0 \\ z & \text{ow}\end{cases}$
- $f(z) = \tanh(z)$



- $f(z) = \text{sigmoid}(z)$ $\frac{1}{1+e^{-z}}$



- $f(z) = \text{softmax}(z) = \frac{e^{z_j}}{\sum_k^n e^{z_k}}$ (Probability) – outputs a probability dist vector

Ex $X = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 1 & 2\end{bmatrix}$

$Y = [0,1,0]$

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

$w_{01} = -0.5$ $w_{1,1} = 1$
$w_{2,1} = -1$ $w_{0,2} = 1.5$
$w_{1,2} = -1$ $w_{2,2} = 1$

$z_1 = \vec{w}\cdot\vec{x}_1 = [1,-1]\cdot[1,0] = 1$
$f(z_1) = 1$

$f(z) = \begin{array}{c} x_1\ x_2\ x_3 \\ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1\end{bmatrix}\end{array}$

$v_1 + v_2 + v_0 = +$
$v_0 + v_1 = +$
$v_2 + v_0 = -$
$v_2 = -2$
$v_0 = 1$
$v_1 = 3$
could be these values

Ex Ridge Regression w/ kernels

$J(\theta) = \frac{\lambda}{2}\|\theta\|^2 + \frac{1}{n}\sum\frac{(y^{(t)}-\theta\cdot\phi(x^{(t)}))^2}{2}$

- $\nabla_\theta \to \lambda\hat{\theta} - \frac{1}{n}\sum(y^{(t)}-\theta\cdot\phi(x^{(t)}))\phi(x^{(t)}) = 0$
- $J(\theta) \geq J(\hat{\theta})$ for all $\theta\in\mathbb{R}^d$
- if $\lambda\uparrow$, $\|\theta\|\downarrow$
- Adding features to $\phi(x)$ will $\downarrow$ training error

Predict: $\hat{\theta}\cdot\phi(x)\begin{cases} 1 & \geq 0.5 \\ 0 & \text{ow}\end{cases}$

Learning/Backprop

$\vec{a} = \text{sm}(\vec{z})$

$\boxed{\text{Loss}} = \begin{cases}(1-yz) & yz<1 \\ 0 \end{cases}$ or $\boxed{(1-yf)}$ $yf<1$

usu you take ∇ Loss wrt weight or ∇ Loss wrt w_{02}

$= -y\frac{\partial f}{\partial w} = -y\frac{\partial f}{\partial z}\left(\frac{\partial z}{\partial w}\right)$
$= -y\frac{\partial f}{\partial z}\left(\frac{\partial}{\partial w}\sum w\cdot x + b\right)$

For simple NN w/ linear act.

$w_i \leftarrow w_i + \eta y^{(i)}x^{(i)}$
$w_0 \leftarrow w_0 + \eta y^{(i)}$

$\text{NLL}(a,y) = -\sum_{j=1}^{n^L} y_j \log a_j^L$ – Objective for probability dist $\vec{a},\vec{y}$

$\frac{\partial}{\partial w_{jk}^L}\text{NLL}(a^L,y) = x_k(a_j^L - y_j)$

probability is positive; a from sigmoid function

Sigmoid deriv $\to$
$y = \frac{1}{1+e^{-z}}$
$y' = y(1-y) = \frac{\partial y}{\partial z}$

* larger weights cause sigmoid to saturate

Relu deriv
$f_1 = \text{Relu}[w_1 x + w_0]$
$\frac{\partial f}{\partial w_1} = x[[f_1 > 0]]$

MDP | Horizon stuff

$\pi(s) = \max_{a\in A}\left[R(s,a) + \sum_{s'} T(s,a,s')\cdot V_{H-1}(s')\right]$

Val Iteration – Fin
$Q_0(s,a) = 0$
For h in 1:H:
For s,a in S,A:
$Q_h(s,a) = R(s,a) + \sum_{s'} T(s,a,s')\underbrace{\max_{a'} Q_{h-1}(s',a')}_{V_{H-1}(s')}$

Inf
$Q(s,a) = R(s,a) + \gamma\sum_{s'} T(s,a,s')\underbrace{\max_{a'} Q_{old}(s',a')}_{V(s)}$

until $\max_{s,a}\|Q_{old}(s,a) - Q_{new}(s,a)\| < \epsilon$

MDP cont

$Q(s,a) := (1-\alpha)Q(s,a) + \alpha\left(R + \gamma \max_{a'} Q(s_{t+1}, a)\right)$

Value Iteration w/ $R(s,a,s')$

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

Goal-reward – init 0, all movements then are random until goal state is found

SSP: init -1, 0 for reward → all movements are away from prev bc -1 reward
↳ can discover faster

Deep Q Learning

- Bellman error: $R(s_t,a_t) + \gamma \max_{a'} \left[Q(s_{t+1}, a', \theta) - Q(s_t, a_t; \theta)\right]^2$
- Fitted Q learn:

```
For i in iters:
    for k in #trajectories:          ↗ ε-greedy
        t = episode(policy)
        experiences.append(random-select t)
    for e in experiences:
        data, labels = experiences   ↗ Goal labels = r+γ max_a' Q(s',a,θ)
        model.fit(data, labels)
```

RNN

$s_t = f_1(W^{sx}x_t + W^{ss}s_{t-1})$

$z_t = f_2(W^o s_t)$

Let $z_t = \sum_{i=0}^{t} (0.9^{t-i} + 0.8^{t-i}) x_i$

Smallest dimensionality: 2

$W^{ss} = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.8 \end{bmatrix}$

$y_t = y_{t-1} - 2y_{t-2} + 3y_{t-3}$

$x_t = y_{t-1}$

$s_t = f_1(W^{sx}x_t + W^{ss}s_{t-1})$

$y_t = f_2(W^o s_t)$

$W_{ss} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

$W_{sx} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$W^o = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$

$\boxed{\text{Ex}}$ $R(s,a) \to$ 1 if $s=1$, 2 if $s=3$, 0 ow — 4 states, 2 actions a,b, $\gamma = 0.9$

Init $Q = 0 \ \forall s,a$

Q first iter → Rewards

$[0, 0, 1, 1, 0, 0, 2, 2]$

$\boxed{\text{2nd}}$ $Q(0,a) = R(0,a) + \gamma[0.9 \cdot 1 + 0.1 \cdot 0] = 0.81$

⋮

$\boxed{\text{Ex}}$ Horizons $R = -1$, 0 ow

Goal: 10, $H = 15$, $\gamma = 1$ $V^*(1) = -9$

ori = 1 $H = \infty$ $V^*(1) = -9$

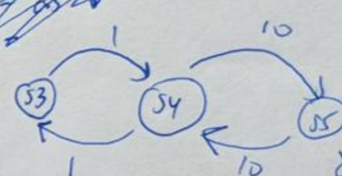
If $R = 0$, 1 ow

$V^*(1) = 0$, $H = 1$

$V^*(1) = 6$, $H = 15$

$V^*(1) = \infty$ $H = \infty$

$\boxed{\text{Ex MDP}}$



$\gamma = 0.5$

$Q\,S3_1 = 1$

$Q\,S3_2 = 1 + 0.5 \cdot 10 = 6$

$Q\,S3_3 = 1 + 0.5 \cdot 0.5 \cdot 10 + 0.5 \cdot 10 =$

$Q\,S3_\infty = 1 + \underbrace{\sum_{i=1}^{\infty} (0.5^i)}_{1} \cdot 10 = 11$

$\boxed{\text{CNN Ex}}$

8x8 Image – Layer 1

Layer 2: 4 filters w/ 3x3 field, stride 2

Layer 3: Max-pooling filter 2x2, stride 2

Layer 4: 4 filters with 3x3x4 field w/ stride 1 (↳ channels)

Layer 5: Max Pooling w/ filter size 2x2, stride 2

→ Weights b/w 1,2 inc. biases: $3 \times 3 \times 4 + 4 = 40$

→ Each output is 8x8 for padding

→ 9 pixels in original image contribute to layer 2 pixel

→ output of layer 3 is 4 images 4x4 bc stride 2

→ 4x4 pixels in original image contributed to one pixel in layer 3

→ $3 \times 3 \times 4 \times 4 + 4 = 148$ b/w 3 and 4

→ 2x2 large images from Layer 5 bc stride 2.

Loss Objective

$$J(\theta, \theta_0) = \overbrace{\frac{1}{n}\sum_{i=1}^{n} \text{Loss}(x^{(i)}, y^{(i)}, \theta, \theta_0)}^{\text{average loss on training examples}} + \lambda \overbrace{R(\theta, \theta_0)}^{\text{regularizer}}$$

$\gamma = \frac{y^{(i)}(\theta^T x^{(i)})}{\|\theta\|}$

margin

SVM Objective - optimize margin, margin size

$$J(\theta,\theta_0) = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) + \frac{\lambda}{2}\|\theta\|^2$$

Maximize $\rightarrow \gamma = \frac{1}{\|\theta\|}$ $\downarrow$ minimize

You can use this w/ perceptron classification

* geometric margin ⇒ margin on graph from correct points

Hinge Loss

taking the agreement as its argument

$$\text{Loss}_h(y(\theta \cdot x + \theta_0)) = \max\{0, 1 - y(\theta \cdot x + \theta_0)\} = \begin{cases} 1 - y(\theta \cdot x + \theta_0) & \text{if } y(\theta \cdot x + \theta_0) \le 1 \\ 0, & \text{o.w.} \end{cases} \quad (8)$$

By introducing this loss, we do not exclude any linear classifier from being selected but

→ takes in same X, Y as perceptron, returns optimized Decision boundary

Algorithm 1 Pegasos Algorithm (without offset)

```
procedure PEGASOS({(x^(i), y^(i)), i = 1, ..., n}, λ, T)
    θ = 0 (vector)
    for t = 1, ..., T do
        Select i ∈ {1, ..., n} at random
        η = 1/t
        if y^(i) θ · x^(i) ≤ 1 then
            θ = (1 − ηλ)θ + η y^(i) x^(i)
        else
            θ = (1 − ηλ)θ
    return θ
```

$\theta - \eta \nabla = \boxed{\cancel{\lambda}\nabla_\theta = -XY + \lambda\theta}$

else: just add regularization

$\theta \leftarrow \theta + \eta \nabla$

* keep in mind what Loss is for: optimizing θ
  - when you evaluate a θ: no regularization
  - When you select a θ and optimize, use regularization
  - when you select for λ, this is when you're trying to figure out how good your loss func was, and you test it on your test data.