5: Regression

CS1420: Machine Learning

Michael L. Littman

Spring 2018

http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/ . Sections 9.2, 13.1, 23.1, 14, 20, 23.2

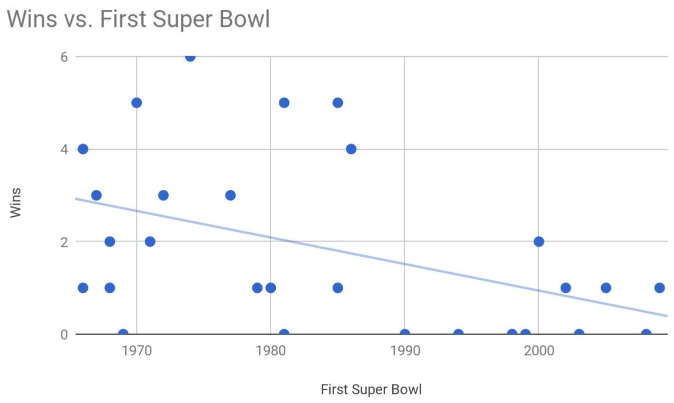
Linear Regression

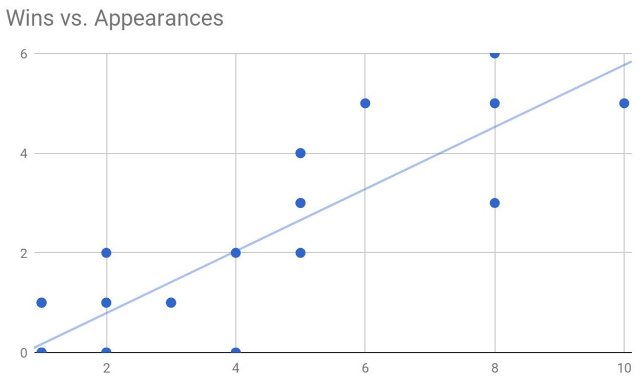
$\mathcal{X} = \mathcal{R}^d$, $\mathcal{Y} = \mathcal{R}$.

Distinct from classification, regression is concerned with continuous valued outputs. Linear regression considers at hypotheses that are linear functions of their inputs

$H = \{ \mathbf{x} \rightarrow \langle\mathbf{w},\mathbf{x}\rangle \}$.

We'll assume again that vectors have a spare dimension always set to one to handle the intercept.





Loss for Regression

Squared loss, mean squared error (MSE):

$$L_S(h) = 1/m \sum_{(\mathbf{x},\, y) \in S} (h(\mathbf{x}) - y)^2 \,.$$

Absolute value loss:

$$L_S(h) = 1/m \sum_{(\mathbf{x},\, y) \in S} |h(\mathbf{x}) - y| \,.$$

ERM: Least Squares Derivation

$\operatorname{argmin}_h L_S(h)$
$= \operatorname{argmin}_h 1/m \sum_{(\mathbf{x},y) \in S} (h(\mathbf{x}) - y)^2$
$= \operatorname{argmin}_{\mathbf{w}} 1/m \sum_{(\mathbf{x},y) \in S} (h_{\mathbf{w}}(\mathbf{x}) - y)^2$
$= \operatorname{argmin}_{\mathbf{w}} 1/m \sum_{(\mathbf{x},y) \in S} (<\mathbf{w},\mathbf{x}> - y)^2$

Set derivative (wrt w_i) to zero:

$\forall i, 0 = 2/m \sum_{(\mathbf{x},y) \in S} (<\mathbf{w},\mathbf{x}> - y)x_i$
$\forall i, \sum_{(\mathbf{x},y) \in S} y\, x_i = \sum_{(\mathbf{x},y) \in S} <\mathbf{w},\mathbf{x}>x_i$

System of equations, one for each w_i.

Put in matrix form $A\mathbf{w}=b$ (so, $\mathbf{w}=A^{-1}b$):

$A = \sum_{(\mathbf{x},\, y) \in S} \mathbf{x}\, \mathbf{x}^{\mathrm{T}}$

$b = \sum_{(\mathbf{x},\, y) \in S} y\, \mathbf{x}$

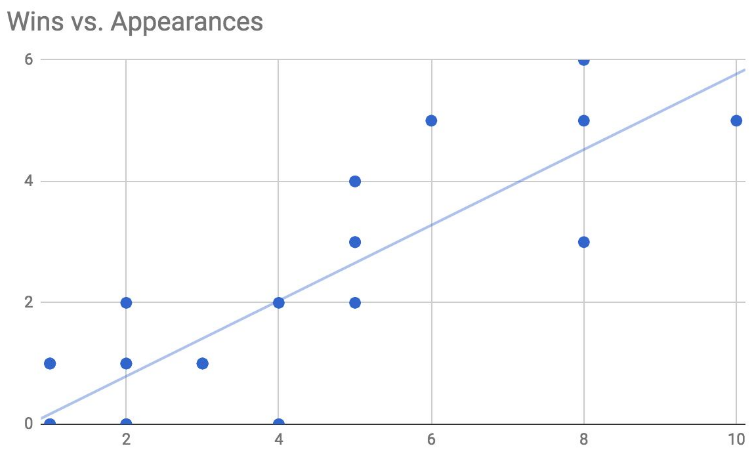
$\forall i, (A\mathbf{w})_i = b_i$
$\Leftrightarrow \forall i, (\sum_{(\mathbf{x},\, y) \in S} (\mathbf{x}\, \mathbf{x}^{\mathrm{T}})\mathbf{w})_i = (\sum_{(\mathbf{x},\, y) \in S} y\, \mathbf{x})_i$
$\Leftrightarrow \forall i, (\sum_{(\mathbf{x},\, y) \in S} \mathbf{x} <\mathbf{x},\mathbf{w}>)_i = (\sum_{(\mathbf{x},\, y) \in S} y\, \mathbf{x})_i$
$\Leftrightarrow \forall i, \sum_{(\mathbf{x},\, y) \in S} x_i <\mathbf{x},\mathbf{w}> = \sum_{(\mathbf{x},\, y) \in S} y\, x_i$

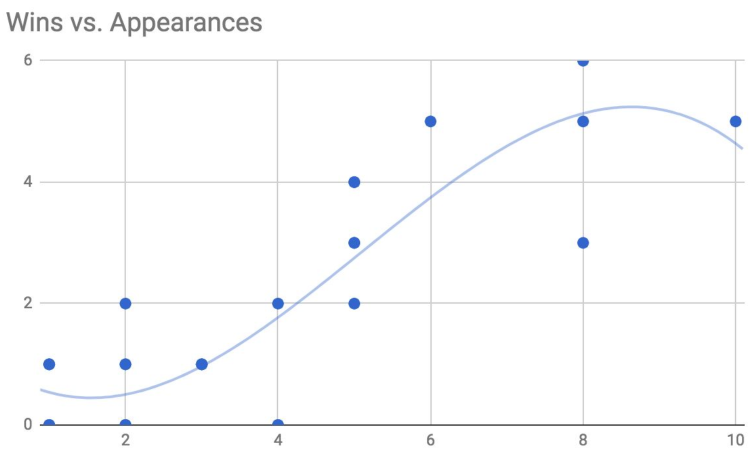
Polynomial Regression

By expanding the feature set, can also fit a more general polynomial function of the original features.

Team	y	1	x	x^2	x^3
Packers	**4**	1	5	25	125
Panthers	**0**	1	2	4	8
Patriots	**5**	1	10	100	1000

$$h(x) = a + bx + cx^2 + dx^3$$

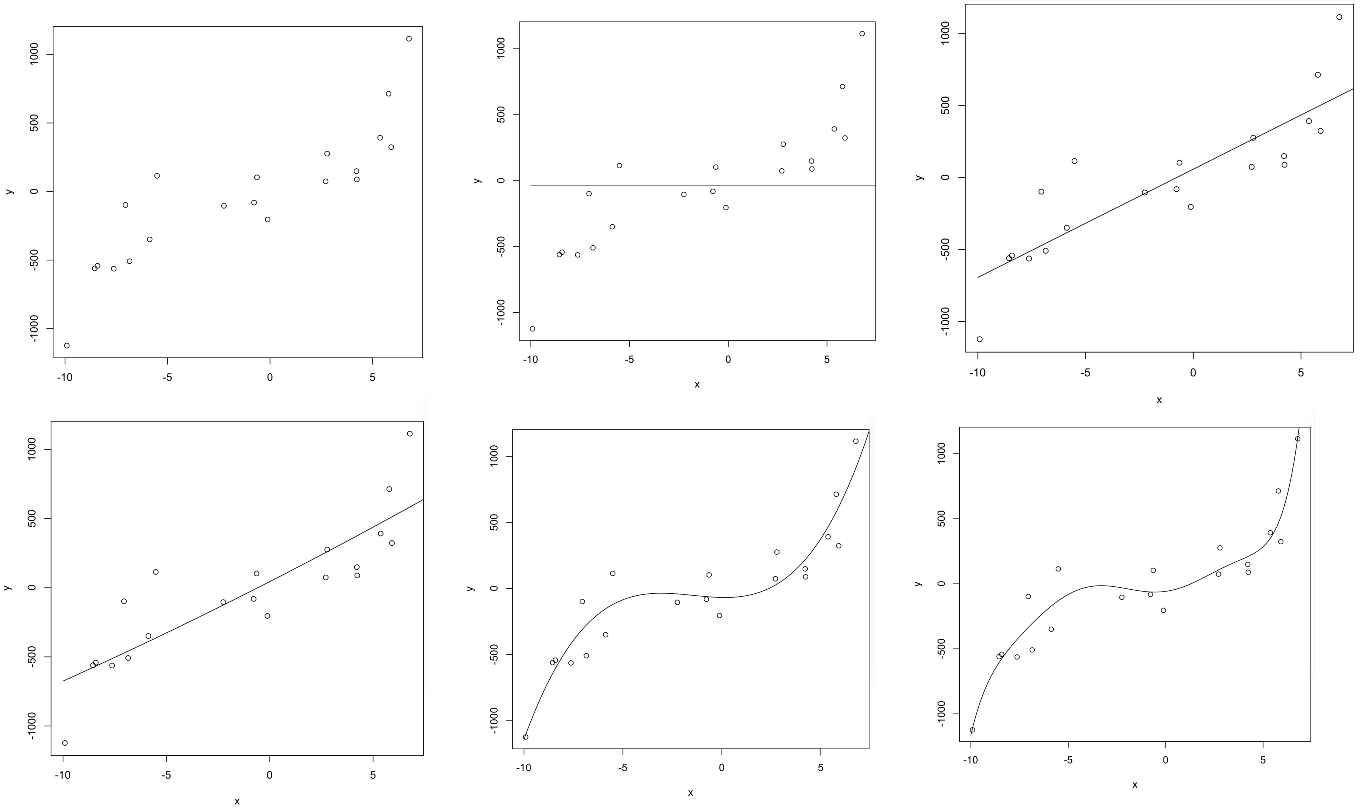




Risk of Increasing The Degree of Polynomial

a. Increased computational cost to derive weights for the new features.
b. More features learned, so more memory needed.
c. Number of features is d choose k (d is number of dimensions, k is the degree of the polynomial), blows up exponentially with k.
d. Additional parameters can lead to overfitting.
e. All of above.

Polynomial Regression Demo




```
k=0: 4749711
k=1: 1192544
k=2: 1190601
k=3:  509179
...
k=7:  411436
```

Fighting Overfitting While Fitting

RLM: Regularized loss minimization.

Minimize loss, but also minimize the “complexity” of the hypothesis.

If our hypothesis class is parameterized by a weight vector $\mathbf{w}$, RLM is

$$\mathrm{argmin}_{\mathbf{w}} \left(L_S(\mathbf{w}) + R(\mathbf{w})\right)$$

where L is our loss function and R is the “regularization function” penalizing complex hypotheses.

ERM is RLM with a constant regularizer: $R(\mathbf{w})=0$.

Simple Regularization Function

$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + \ldots + w_k x^k.$

$R(\mathbf{w}) = \lambda \max \{ i \mid w_i \neq 0 \}.$

In words? In practice? Advantages? Challenges?

Computational approach?

Tikhonov Regularization

$R(\mathbf{w}) = \lambda \, ||\mathbf{w}||^2.$

Penalize the use of big values in the vector. Related somewhat to previous ideas:

- Can’t do much in terms of high order terms if the weights are small.
- Can’t have high Lipschitz constant if the weights are small.

Ridge regression = linear regression + Tikhonov regularization:

$$\operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \left(\lambda \|\mathbf{w}\|_2^2 + \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}(\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 \right)$$

Solving Ridge Regression

Could do gradient descent, but things work out a bit better than that:

RLM: Least Squares Derivation

$\text{argmin}_{\mathbf{w}} (L_S(\mathbf{w}) + R(\mathbf{w}))$
$= \text{argmin}_{\mathbf{w}} (1/m \sum_{(\mathbf{x},\, y) \in S} (h_{\mathbf{w}}(\mathbf{x}) - y)^2 + \lambda \, ||\mathbf{w}||^2)$
$= \text{argmin}_{\mathbf{w}} (1/m \sum_{(\mathbf{x},\, y) \in S} (<\mathbf{w}, \mathbf{x}> - y)^2 + \lambda$
$<\mathbf{w}, \mathbf{w}>)$

Set derivative (wrt w_i) to zero:

$\forall\, i,\, 0 = 2/m \sum_{(\mathbf{x},\, y) \in S} (<\mathbf{w}, \mathbf{x}> - y) x_i + \lambda 2 w_i$
$\forall\, i, \sum_{(\mathbf{x},\, y) \in S} y\, x_i = \sum_{(\mathbf{x},\, y) \in S} <\mathbf{w}, \mathbf{x}> x_i + \lambda m w_i$

System of equations, one for each w_i.

$A = \sum_{(\mathbf{x},\, y) \in S} \mathbf{x}\,\mathbf{x}^{\mathrm{T}},\, b = \sum_{(\mathbf{x},\, y) \in S} y\,\mathbf{x}$

Solve: $(\lambda m I + A)\mathbf{w} = b$

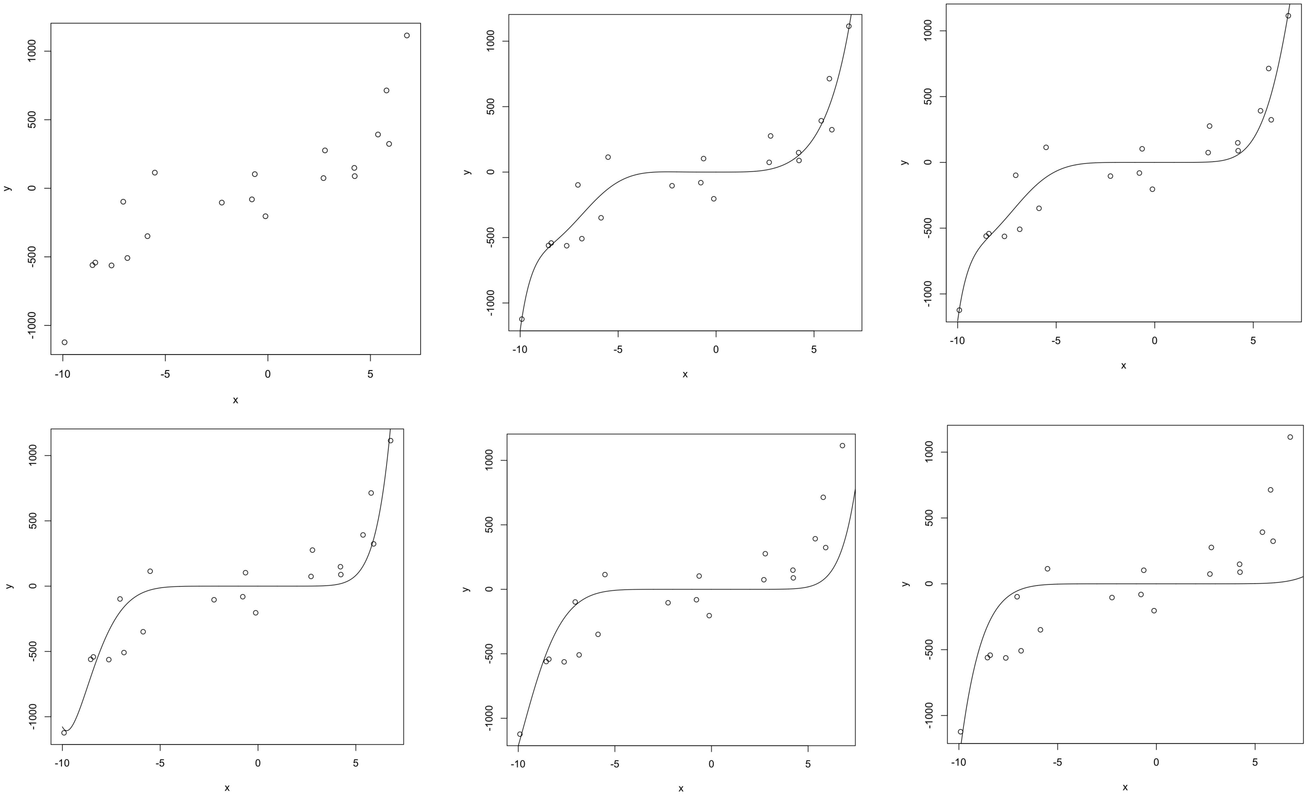
$\mathbf{w} = (\lambda m I + A)^{-1}\, b$

Just like regular regression, but we've added some weight to the diagonal.

FYI: "Ridge" is a description of the error surface, not this raised diagonal.

Bigger λ, more like the identity.

Ridge Regression Demo (k=10)



Regularizer

10^5

10^10

10^12

10^16

10^17

10^22: flat

Regularization: Finite Hypothesis Case

H: main hypothesis space, possibly infinite (even in terms of VC dimension)

Consider nested subsets of hypothesis spaces:

$H_1 \subseteq H_2 \subseteq H_3 \subseteq H_4 \subseteq \ ... \subseteq H$

Examples?

Error Quantities

H_i: hypothesis class i ($H_i \subseteq H_{i+1}$).

h_i: ERM hypothesis in H_i.

h_i*: true error minimizer in H_i.

err$_i$: empirical risk of h_i.

err*$_i$: true risk of h_i.

err**$_i$: true risk of h_i^*.

What can we say about

- err$_i$ vs err$_{i+1}$?
- err*$_i$ vs err*$_{i+1}$?
- err**$_i$ vs err**$_{i+1}$?

Can we use these relationships to define a rule for picking i and returning h_i?

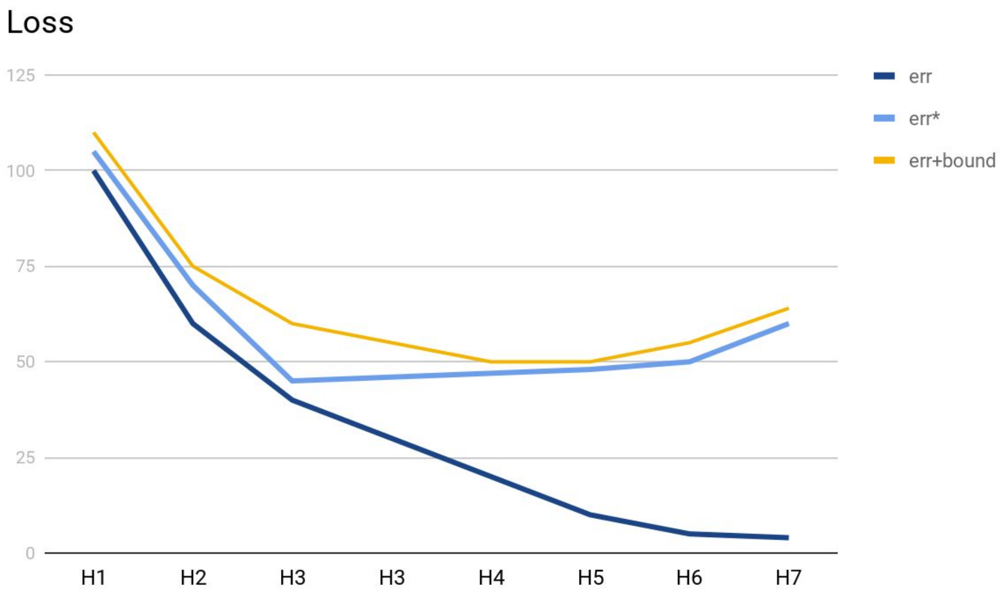
Justify?

Error Graph (on board)

How Choose?

Want the min of err*.

Maybe close enough to choose min of err + bound.



Which Hypothesis Should We Choose?

Relate data and accuracy: $m \geq \log(2|H|/\square)/2\varepsilon^2$.

With m samples, $1-\square$ sure we're within ε of best hypothesis in H.

How good is the hypothesis? err - ε ≤ err* ≤ err + ε, where $\varepsilon = \sqrt{}\log(2|H|/\square)/2m$.

$\text{err}_i - \varepsilon_i \leq \text{err}_i^* \leq \text{err}_i + \varepsilon_i$, where $\varepsilon_i = \sqrt{}\log(2|H_i|/\square_i)/2m$ (grows with H_i).

$\text{armin}_i\ (\text{err}_i + \sqrt{}\log(2|H_i|/\square_i)/2m)$.

If just a finite set (k) of hypotheses, $\square_i = \square/k$. If infinite, maybe $\square_i = \square/2^i$.

Viewed As Regularizer (RLM)

armin$_i$ (err$_i$ + $\sqrt{}\log(2|H_i|/\square_i)/2m$).

Recall RLM: $\mathrm{argmin}_h\ (L_S(h) + R(h))$.

Define $R(h) = \sqrt{}\log(2|H_i|/\square_i)/2m$ where $i = \mathrm{argmin}_i\ \{i \text{ s.t. } h \text{ in } H_i\}$.

The regularizer returns the same value for all hypotheses in a given strata. So, only need to compare the lowest loss hypothesis in each strata.

So, using the lowest upper bounded quantity can be viewed as RLM!

Formal guarantee: Best hypothesis in the worst case given the sample size.

Dimension Reduction

$\mathcal{X} = \mathcal{R}^d \rightarrow \mathcal{R}^k$, where $k < d$.

Why?

- Compress data.
- Improve running time.
- Improve interpretability.
- Enable visualization.
- Denoise.
- Create generalizations by establishing better similarities.

Dimension Reduction Made Easy

I have data in 1000 dimensions and I want to make it 100 dimensional.

What’s the fastest way to do that?

A. Use singular value decomposition.

B. Use an variational autoencoder in a deep neural network.

C. Delete the final 900 dimensions.

D. Use eigenvalue decomposition.

E. All of the above.

Need a Loss Function

Linear dimension reduction goes like this.

Find a matrix $E \in \mathcal{R}^{k\text{x}d}$ that induces the mapping $\mathbf{x} \rightarrow E\mathbf{x}$.

One idea (linear “autoencoding”):

- $E\mathbf{x}$ should retain information about $\mathbf{x}$: if we tried to reconstruct $\mathbf{x}$ from $E\mathbf{x}$, our reconstruction would be optimal in a least squares sense.

Form of unsupervised learning: no labels, just $\mathbf{x}$.

Principal Component Analysis

PCA is not

- PCR (polymerase chain reaction)
- PCP (probabilistically checkable proofs)

Define $E \in \mathcal{R}^{k\mathrm{x}d}$, but also $D \in \mathcal{R}^{d\mathrm{x}k}$. E compresses $\mathbf{x}$, and D tries to reconstruct it.

$\mathbf{y} = E\mathbf{x}$, $\mathbf{x}^{\mathrm{hat}} = D\mathbf{y}$, want $\mathbf{x}$ and $\mathbf{x}^{\mathrm{hat}}$ to be similar.

PCA, given target dimension n: $\mathrm{argmin}_{U,W} \sum_{\mathbf{x} \in S} ||\mathbf{x} - DE\mathbf{x}||^2$.

How Do We Solve It?

Gradient descent is always an answer, but usually the worst one.

Linear algebra!

Claim: $D^{T}D = I$.

Fix D, E. Note that $DE\mathbf{x}$ is an k-dimensional linear subspace of $\mathcal{R}^{d}$. Let $V \in \mathcal{R}^{d\text{x}k}$ be an orthonormal basis that spans this subspace ($V^{T}V = I$). That means any vector $DE\mathbf{x}$ can be written as $V\mathbf{y}$. $||\mathbf{x}\text{-}V\mathbf{y}||^2 = ||\mathbf{x}||^2 - 2\ ||\mathbf{x}V\mathbf{y}|| + ||V\mathbf{y}||^2 = ||\mathbf{x}||^2 + ||\mathbf{y}||^2 - 2\ ||\mathbf{x}V\mathbf{y}||$.

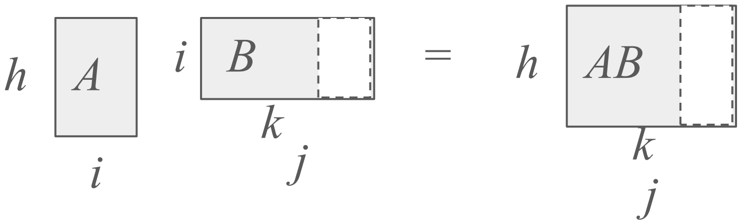
To find min, take derivative/solve: $\mathbf{y} = V^{T}\mathbf{x}$. (Same trick as with linear regression.)

That means, for each $\mathbf{x}$, $\min_{D,E} ||\mathbf{x}\text{-}DE\mathbf{x}|| = ||\mathbf{x}\text{-}VV^{T}\mathbf{x}||$. Let $D = V$, $E = V^{T}$.

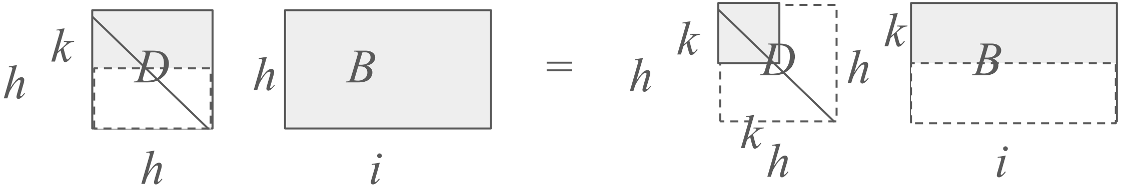
Matrix Truncation Algebra

If A (ixj) is a matrix, $\underline{A}$ (ixk) is the matrix with all but the first k columns removed and $A|$ (kxi) is the matrix with all but the top k rows removed.

$A\,\underline{B} = \underline{A\,B}$, $A|\,B = (AB)|$, $\underline{A}^{\mathrm{T}} = (A^{\mathrm{T}})|$



If D is diagonal, $D|\,B = \underline{D|}\,B|$.



Singular Value Decomposition

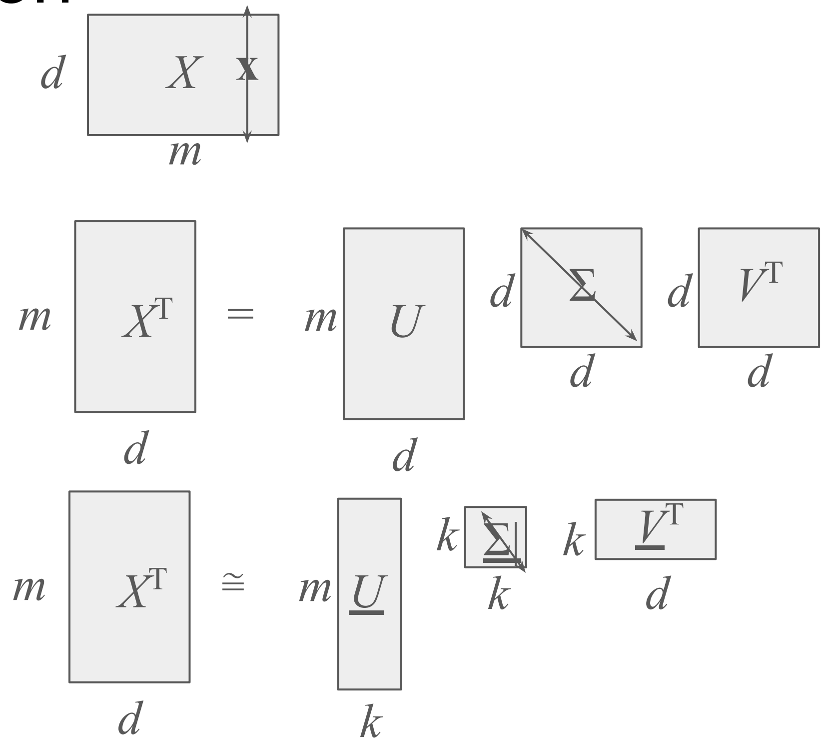
Let X be the matrix of input vectors.

$X^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}$, where $U^{\mathrm{T}}U = I$, $V^{\mathrm{T}}V = I$,
Σ diagonal, if $d<m$, $VV^{\mathrm{T}} = I$.

Drop the d-k smallest values of Σ and adjust the matrices: $X^{\mathrm{T}} \neq \underline{U}\,\underline{\Sigma|}\,\underline{V}^{\mathrm{T}}$.

Still: $\underline{U}^{\mathrm{T}}\underline{U} = I$, $\underline{V}^{\mathrm{T}}\underline{V} = I$, $\underline{\Sigma|}$ diagonal.

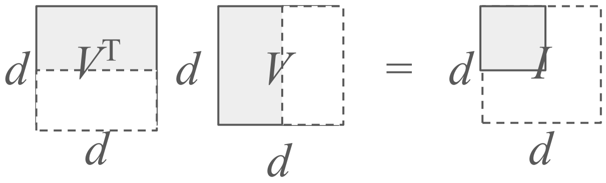
SVD of X^{hat}, the best (least squares) reconstruction of X in k dimensions.



SVD for PCA

$\mathrm{argmin}_{D,E} \sum_{\mathbf{x}\in S} ||\mathbf{x} - DE\mathbf{x}||^2 = \mathrm{argmin}_{D,E} ||X - DEX||^2 = ||X - \underline{V}\ \underline{\Sigma|}\ \underline{U}^{\mathrm{T}}||^2$

Want $DEX = X^{\mathrm{hat}} = \underline{U}\ \underline{\Sigma|}\ \underline{V}^{\mathrm{T}}$.



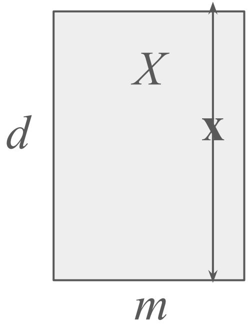
What about: $D = \underline{V}$, $E = \underline{V}^{\mathrm{T}}$?

Then, $DEX = \underline{V}\ \underline{V}^{\mathrm{T}} V \Sigma\ U^{\mathrm{T}} = \underline{V}\ (V^{\mathrm{T}})|\ V \Sigma\ U^{\mathrm{T}} = \underline{V}\ (V^{\mathrm{T}} V)|\ \Sigma\ U^{\mathrm{T}} = \underline{V}\ \mathrm{I}|\ \Sigma\ U^{\mathrm{T}} = \underline{V}\ \Sigma|\ U^{\mathrm{T}} = \underline{V}\ \underline{\Sigma|}\ \underline{U}^{\mathrm{T}}$

Note: If $X^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}$, $XX^{\mathrm{T}} = V\Sigma U^{\mathrm{T}} U\Sigma V^{\mathrm{T}} = V\Sigma^2 V^{\mathrm{T}}$. $VV^{\mathrm{T}} = V^{\mathrm{T}} V = I$, so $V^{\mathrm{T}} = V^{-1}$. Eigenvalues!

Classically, PCA first “centers” the data: M is mxm matrix of $1/m$, $X = X^{\mathrm{orig}} - X^{\mathrm{orig}} M$ (subtract off the average vector).

PCA in High Dimensions



What happens if $d>m$?

$X^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}$, where $U^{\mathrm{T}}U = I$, $V^{\mathrm{T}}V = I$, Σ diagonal, if $d>m$, $UU^{\mathrm{T}} = I$.

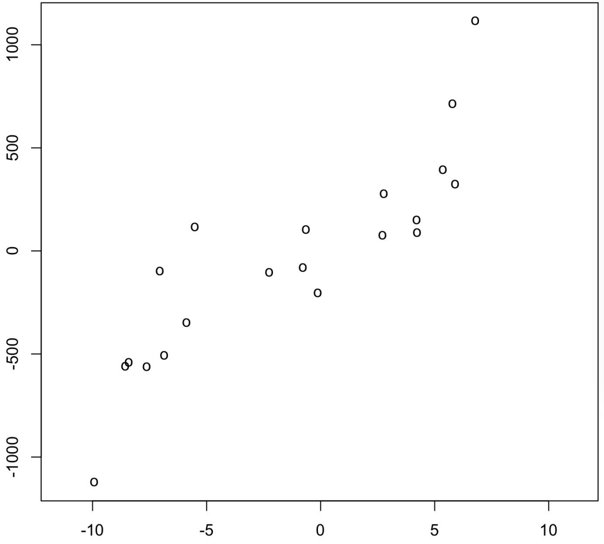
If use: $D = \underline{U}\,\underline{\Sigma|}^{-1}$, $E = \underline{\Sigma|}\,\underline{U}^{\mathrm{T}}$?

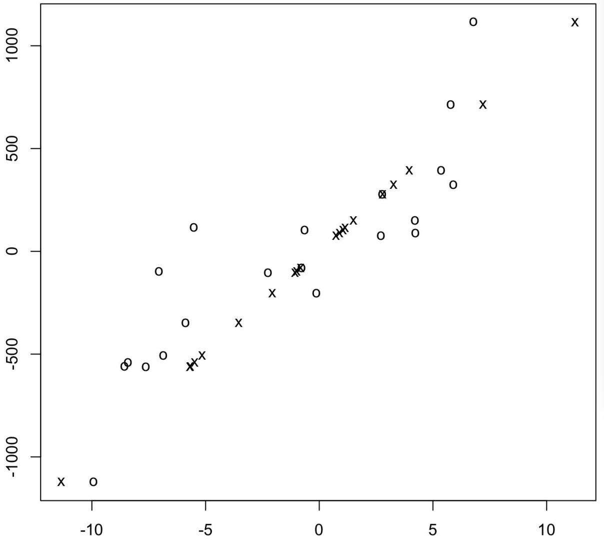
Then, $DEX = \underline{U}\,\underline{\Sigma|}^{-1}\,\underline{\Sigma|}\,\underline{U}^{\mathrm{T}}\,V\Sigma U^{\mathrm{T}} = \underline{V}\,\underline{\Sigma|}\,\underline{U}^{\mathrm{T}}$.

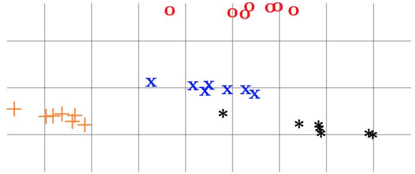
But, can compute $\underline{U}\,\underline{\Sigma|}^{-1}$ from eigenvalues/SVD of $X^{\mathrm{T}}X$. That's a matrix of dot products of the data. Only use dot products… kernel PCA!

Reconstruction Example

2 to 1, 2500 to 10 (to 2) from book







Text Applications

Latent semantic indexing: Can discover the color wheel and the ordering of integers just from word usages!

Non-linear Autoencoding

Can be more powerful…

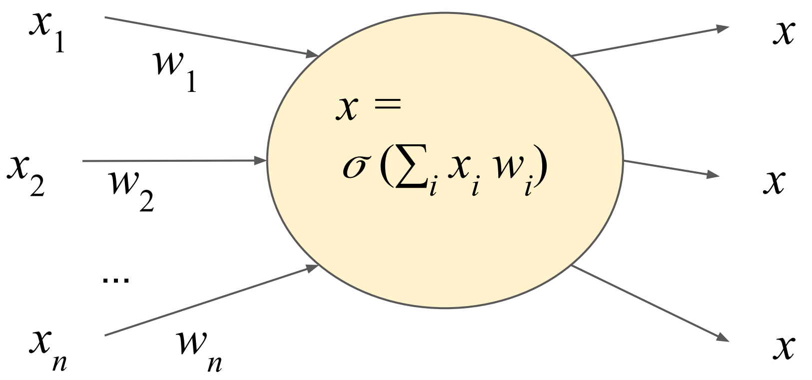
How can we make that happen, though? Need a flexible, yet simple, representation for computations.

Computational Neurons and Their Networks

Analogous to biological neurons.

Activiations (like spike trains) transmitted along edges (like axons).

Activations modulated by weights (like synaptic strengths).



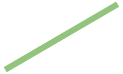
Feedforward neural networks: Neurons arranged into acyclic graph. Neurons with no incoming edges are inputs. Neurons with no outcoming edges are outputs.

In ML: Graph typically held fixed, weights set via learning.

Activation Functions

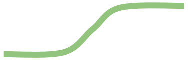
More input leads to more output.

Linear: $\sigma(a) = a$

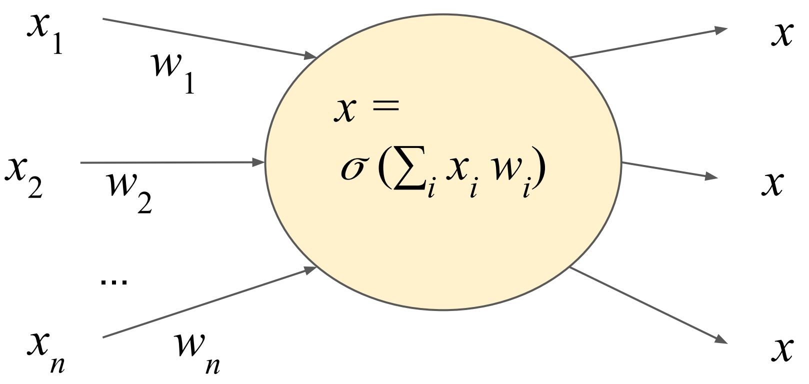


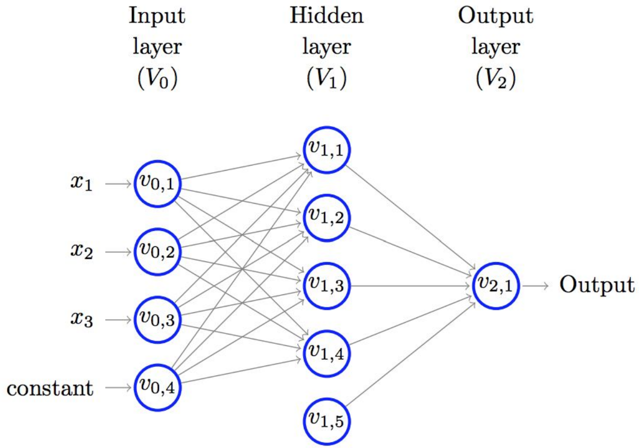
Step: $\sigma(a) = 1$ if $a > 0$, 0 otherwise.

Sigmoid: $\sigma(a) = 1/(1+\exp(-a))$



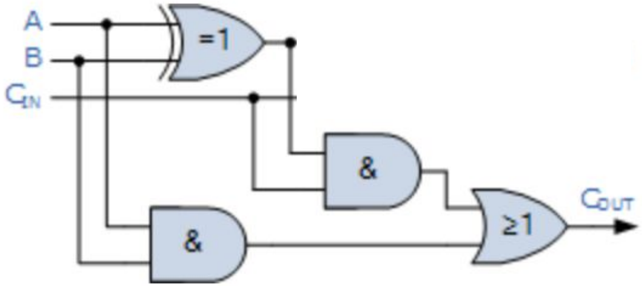
Relu: $\sigma(a) = a$ if $a > 0$, 0 otherwise.





NNs Can Represent Any Boolean Function

Boolean function:

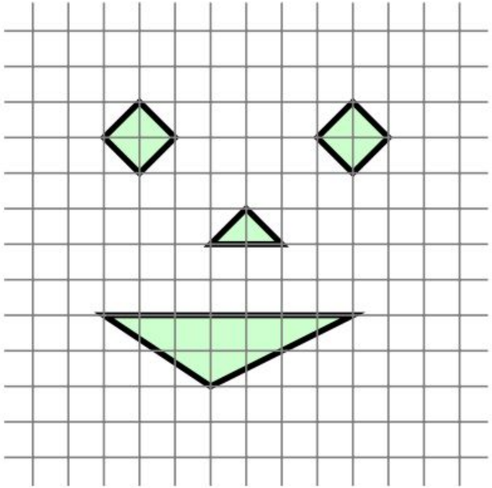


Representational Power

Using step functions and a network with one (giant) hidden layer, can capture any Boolean function we want.

Can convert polynomial-time computations into polynomial-size networks.

Also universal approximator for regression problems.



Stuff We Know

Universal approximators: For any function, there's a big enough network that can have its weights set to approximate that function to arbitrary accuracy.

VC dimension of network with E edges & threshold activation function: $E \log E$.

Computationally hard to find ERM, however.

Can use SGD, with a few caveats (non-convex):

- start with small random weights (symmetry breaking)
- use variable η_t

Gradients (relatively) efficiently computed via backpropagation.

Recall: Gradient Descent

gradient f: $\mathcal{R}^d \rightarrow \mathcal{R}$: $\nabla f(\mathbf{w}) = (\partial f(\mathbf{w})/\partial w_1, \ldots, \partial f(\mathbf{w})/\partial w_d)$.

Gradient descent:

set: η = something smallish

initialize: $\mathbf{w}(0) = \mathbf{0}$; $t = 0$.

iterate T times: $\mathbf{w}(t{+}1) = \mathbf{w}(\mathrm{t}) - \eta \nabla f(\mathbf{w}(t))$; $t = t + 1$.

return (A) $\mathbf{w}(T)$

(B) argmin$_{\mathbf{w}(t)} \|\nabla f(\mathbf{w}(t))\|^2$

(C) $\sum^{T}_{\mathrm{t}=1} \mathbf{w}(t)/T$

(D) argmin$_{\mathbf{w}(t)} f(\mathbf{w}(t))$

(E) argmin$_{\mathbf{w}(t)} \|\mathbf{w}(t)\|^2$

How Does This Help Minimize Loss?

The gradient points in the direction of the greatest growth of f around $\mathbf{w}$, the algorithm makes a small step in the opposite direction, thus decreasing f.

First order Taylor approximation of $f(\mathbf{u}) \approx f(\mathbf{w}) + <\mathbf{u} - \mathbf{w}, \nabla f(\mathbf{w})>$. When f is convex, $f(\mathbf{u}) \geq f(\mathbf{w}) + <\mathbf{u} - \mathbf{w}, \nabla f(\mathbf{w})>$.

$$\mathbf{w}(t+1) = \text{argmin}_{\mathbf{w}} \left(\tfrac{1}{2}\|\mathbf{w}\text{-}\mathbf{w}(\mathrm{t})\|^2 + \eta \left(f(\mathbf{w}(t)) + <\mathbf{w} - \mathbf{w}(t), \nabla f(\mathbf{w}(t))>\right)\right).$$

η trades off distance from previous weights and estimated function value. Minimize $\mathbf{w}$ (how?) and get same update!

Analysis of GD for Convex-Lipschitz Functions

$\mathbf{w}^* = \text{argmin}_{\mathbf{w}:\ \|\mathbf{w}\| \leq B}\ f(\mathbf{w})$.

$\mathbf{w}^{\text{avg}} = \sum_{t=1}^{T} \mathbf{w}(t)/T$.

$f(\mathbf{w}^{\text{avg}}) - f(\mathbf{w}^*)$

$= f(1/T \sum_{t=1}^{T} \mathbf{w}(t)) - f(\mathbf{w}^*)$ — definition of $\mathbf{w}^{\text{avg}}$

$\leq\ 1/T \sum_{t=1}^{T} f(\mathbf{w}(t))\ - f(\mathbf{w}^*)$ — Jensen's inequality (convex f)

$=\ 1/T \sum_{t=1}^{T} (f(\mathbf{w}(t))\ - f(\mathbf{w}^*))$ — independent of t

$\leq\ 1/T \sum_{t=1}^{T} \langle \mathbf{w}(t) - \mathbf{w}^*, \nabla f(\mathbf{w}(t)) \rangle$ — convex f

Theorem: f is convex, ρ Lipschitz. $\mathbf{w}^*$ as above. B is a chosen distance bound. Run gradient descent for T steps with $\eta = B/\rho\ 1/\sqrt{T}$, then $f(\mathbf{w}^{\text{avg}}) \leq f(\mathbf{w}^*) + \eta$.

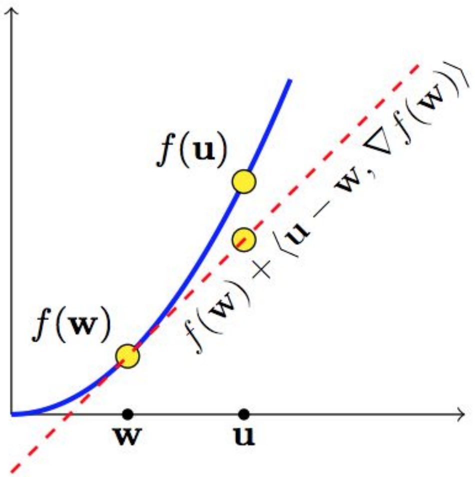
Time Complexity

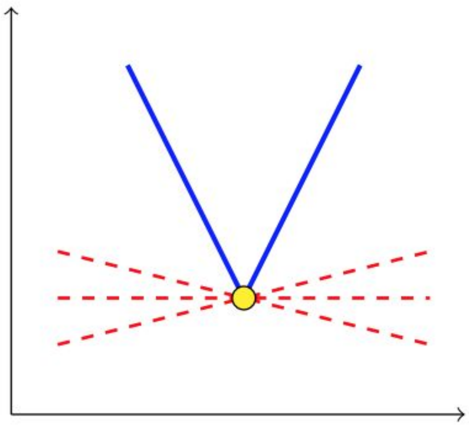
Theorem: f is convex, ρ Lipschitz. $\mathbf{w}^*$ as above. B is a chosen distance bound. Run gradient descent for T steps with $\eta = B/\rho\ 1/\sqrt{T}$, then $f(\mathbf{w}^{\text{avg}}) \leq f(\mathbf{w}^*) + \eta$.

Assuming B and ρ are constants, how many iterations for ε optimal?

A. $O(\log(1/\varepsilon))$
B. $O(1/\sqrt{\varepsilon})$
C. $O(1/\varepsilon)$
D. $O(1/\varepsilon \log(1/\varepsilon))$
E. $O(1/\varepsilon^2)$

Subgradients





For convex function, just need gradient to stay below it.

Can do that, even if function isn't differentiable.

Subgradient $\mathbf{v}$:

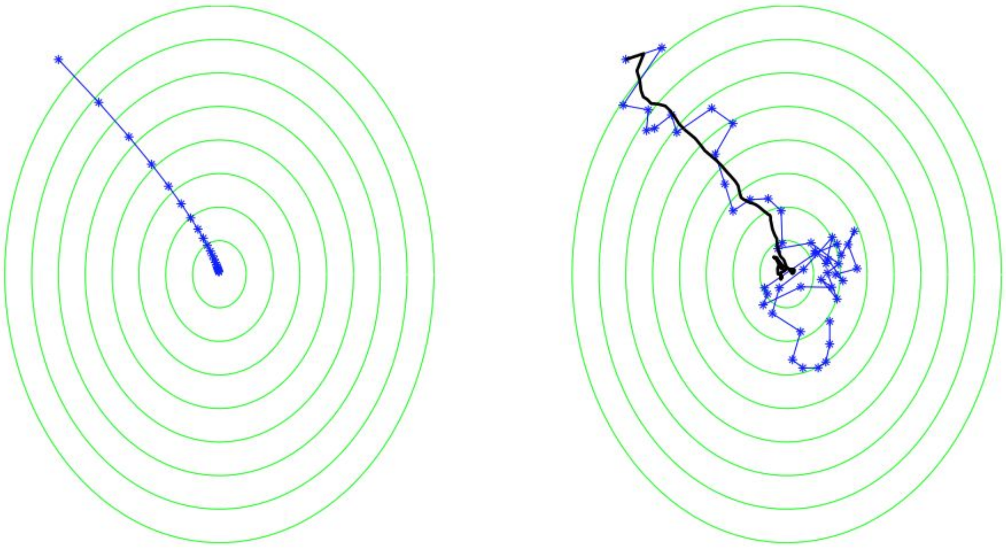
$$f(\mathbf{u}) \geq f(\mathbf{w}) + \langle \mathbf{u}-\mathbf{w}, \mathbf{v} \rangle$$

Stochastic Gradient Descent

Update direction is a random vector $\mathbf{v}$ such that $\mathcal{E}[\mathbf{v}] = \nabla f(\mathbf{w}(t))$ or a subgradient.

$\mathcal{E}[f(\mathbf{w}^{\mathrm{avg}})] \leq f(\mathbf{w}^*) + B/\rho \; 1/\sqrt{T}$.

$\mathbf{v}$ might come from computing gradient on a subset of the data chosen at random.



SGD Variants

Project back into H.

$\underline{\mathbf{w}}(t+1) = \mathbf{w}(\mathrm{t}) - \eta\ \nabla f(\mathbf{w}(t))$

$\mathbf{w}(t+1) = \mathrm{argmin}_{\mathbf{w}:\ \|\mathbf{w}\| \leq B} \|\underline{\mathbf{w}}(t+1) - \mathbf{w}\|;\ t = t + 1$

Still converges (for convex, Lipschitz).

Variable step size: $\eta_t = B/\rho\ 1/\sqrt{t}$.

Directly minimize risk. Draw one fresh sample at each update, expectation is gradient of true (not empirical!) loss.

Back to Feedforward NNs

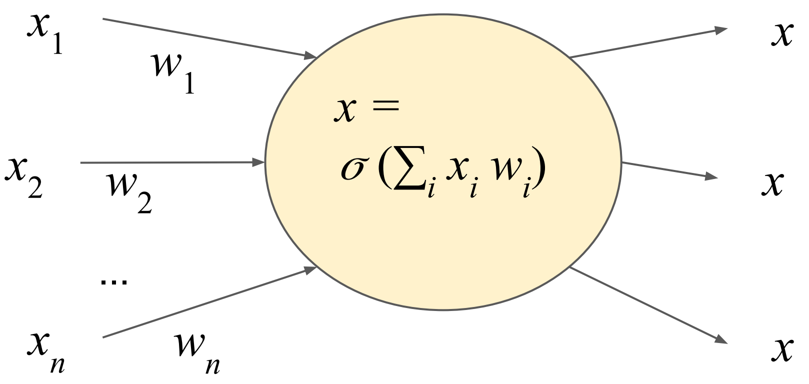
d: dimension of input, n nodes, $w_{i,j}$: weight of edge i,j,

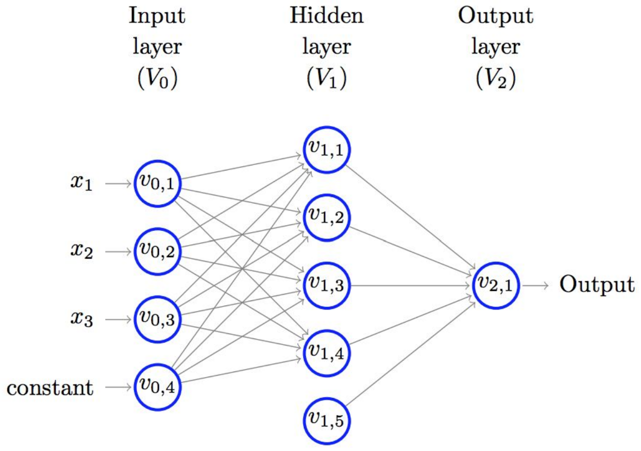
Since graph is acyclic, assume topological sort so all $i<j$ for all $w_{i,j}$. Nodes 1 to d correspond to inputs, node n to output.

To do SGD, we need, for all i,j,

$$\nabla f(\mathbf{w}) = (\partial f(\mathbf{w})/\partial w_{ij})$$

where f is the loss function.





Backpropagation

for $i = 1$ to d: $o_i = x_i$

for $j = d+1$ to n:

$$a_j = \sum_{i<j} w_{i,j}\, o_i$$

$$o_j = \sigma(a_j)$$

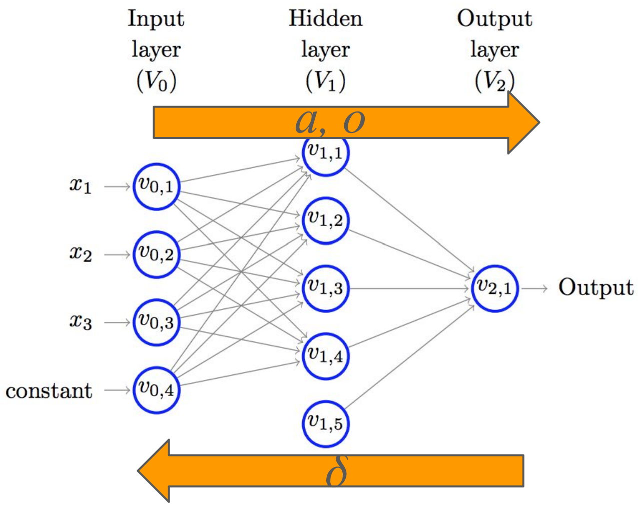
$\delta_n = 2(o_n - y)$ (assuming squared loss)

for $j = n-1$ down to 1:

$$\delta_j = \sum_{k>j} w_{j,k}\, \delta_k\, \sigma'(a_k)$$

for all i^*, j^*:

partial derivative for w_{i^*,j^*}: $\delta_{j^*}\, \sigma'(a_{j^*})\, o_{i^*}$



Derivatives of Activation Functions

Linear: $\sigma(a) = a;\ \sigma'(a) = 1$.

Step: $\sigma(a) = 1$ if $a > 0, 0$ otherwise; $\sigma'(a) = 0$.

Relu: $\sigma(a) = a$ if $a > 0, 0$ otherwise; $\sigma'(a) = 1$ if $a > 0, 0$ otherwise.

Sigmoid: $\sigma(a) = 1/(1+\exp(-a));\ \sigma'(a) = \sigma(a)(1-\sigma(a))$.

$$
\begin{aligned}
&\sigma(a)(1-\sigma(a)) \\
&= 1/(1+\exp(-a))\ (1-1/(1+\exp(-a))) \\
&= 1/(1+\exp(-a))\ ((1+\exp(-a)-1)/(1+\exp(-a))) \\
&= 1/(1+\exp(-a))\ (\exp(-a)/(1+\exp(-a))) \\
&= \exp(-a)/(1+\exp(-a))^2
\end{aligned}
$$

$$
\begin{aligned}
&\sigma'(a) \\
&= -D(1+\exp(-a))/(1+\exp(-a))^2 \\
&= \exp(-a)/(1+\exp(-a))^2
\end{aligned}
$$

Recursive Derivatives of Weights

We want to compute $\partial f(\mathbf{w})/\partial w_{i*,j*}$.

Let p_j be the partial derivative of o_j with respect to $w_{i*,j*}$. Compute recursively via:

$$p_j = D(o_j) = D(\sigma(a_j)) = \sigma'(a_j)\, D(a_j) = \sigma'(a_j)\, D(\textstyle\sum_{i<j} w_{i,j}\, o_i) = \sigma'(a_j) \sum_{i<j} D(w_{i,j}\, o_i)$$
$$= \sigma'(a_j) \textstyle\sum_{i<j} (D(w_{i,j})\, o_i + w_{i,j}\, D(o_i)) = \sigma'(a_j) \sum_{i<j} (D(w_{i,j})\, o_i + w_{i,j}\, p_i)$$

if $j<j*$, $p_j = 0$ since o_j can't be influenced by $w_{i*,j*}$.
if $j=j*$, $p_j = \sigma'(a_j)\, o_{i*}$ since $p_i = 0$ as $i<j=j*$.
else, $p_j = \sigma'(a_j) \sum_{i<j} w_{i,j}\, p_i$

$$\partial f(\mathbf{w})/\partial w_{i*,j*} = D((o_n - y)^2) = 2(o_n - y) D(o_n) = 2(o_n - y)\, p_n$$

Backprop Proof

Run time: For each n^2 weights, compute n p values, each in n time: $O(n^4)$.

Observe: $p_j = o_{i*} \sum_{\text{all paths } \varrho \text{ from } j* \text{ to } j} \prod_{i=2 \text{ to len}(\varrho)} w_{\varrho[i-1],\varrho[i]} \sigma' (a_{\varrho[i]})$.

And, we don't need all the p_j values, just p_n.

Define: $\delta_j = 2(o_n - y) \sum_{\text{all paths } \varrho \text{ from } j \text{ to } n} \prod_{i=2 \text{ to len}(\varrho)} w_{\varrho[i-1],\varrho[i]} \sigma' (a_{\varrho[i]})$.

Then, $\partial f(\mathbf{w})/\partial w_{i*,j*} = D((o_n - y)^2) = 2(o_n - y)D(o_n) = 2(o_n - y)\, p_n = \delta_{j*}\, \sigma' (a_{j*})\, o_{i*}$.

Ah, but δ_j is exactly what backprop produces! $\delta_j = \sum_{k>j} w_{j,k}\, \delta_k\, \sigma' (a_k)$

Run time: Compute n δ values, each in n time. For each n^2 weights $O(1)$: $O(n^2)$.

Flexible “Programming Language” for ML

$$\underset{r \in R}{\hat{\min}}\ L_S(r)$$

Expressive representation.

Flexible loss encoding.

Powerful optimization.

Dimension Reduction, Revisited

Recall PCA.

Define $E \in \mathfrak{R}^{n\mathrm{x}d}$, but also $D \in \mathfrak{R}^{d\mathrm{x}n}$. E compresses $\mathbf{x}$, and D tries to reconstruct it.

$\mathbf{y} = E\mathbf{x}$, $\mathbf{x}^{\mathrm{hat}} = D\mathbf{y}$, want $\mathbf{x}$ and $\mathbf{x}^{\mathrm{hat}}$ to be similar.

PCA, given target dimension n: $\mathrm{argmin}_{D,E} \sum_{\mathbf{x} \in S} \|\mathbf{x} - DE\mathbf{x}\|^2$.

Here's another metric with no decoding: $\|E\,\mathbf{x}_1 - E\,\mathbf{x}_2\| \,/\, \|\mathbf{x}_1 - \mathbf{x}_2\|$.

What does it mean? Strengths? Weaknesses?

Distortion

Metric: $\|E\,\mathbf{x}_1 - E\,\mathbf{x}_2\| \,/\, \|\mathbf{x}_1 - \mathbf{x}_2\|$.

$\|\mathbf{x}_1 - \mathbf{x}_2\|^2 = (\mathbf{x}_1 - \mathbf{x}_2)^T(\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{x}_1^T\mathbf{x}_1 - 2\,\mathbf{x}_2^T\mathbf{x}_1 + \mathbf{x}_2^T\mathbf{x}_2$

$\|E\,\mathbf{x}_1 - E\,\mathbf{x}_2\|^2 = (E\,\mathbf{x}_1 - E\,\mathbf{x}_2)^T(E\,\mathbf{x}_1 - E\,\mathbf{x}_2) = \mathbf{x}_1^T E^T E\,\mathbf{x}_1 - 2\,\mathbf{x}_2^T E^T E\,\mathbf{x}_1 + \mathbf{x}_2^T E^T E\,\mathbf{x}_2$

If result close to 1, distances in projected space similar to the original space.

Doesn’t require decoding.

PCA Distortion

Let X be the matrix of input vectors. $X^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}$. PCA: $E = \underline{V}^{\mathrm{T}}$.

$E^{\mathrm{T}}E = \underline{V}\,\underline{V}^{\mathrm{T}}$. Approximates the identity matrix. Leaves things relatively undistorted.

Random Projection

Fix some $\mathbf{x}$ in $\mathcal{R}^d$. Assume unit vector ($||\mathbf{x}||=1$).

Let W be a nxd matrix where W_{ij} is an independent normal random variable.

$$\mathbb{P}\left[(1-\epsilon)n \le \|W\mathbf{x}\|^2 \le (1+\epsilon)n\right] \ge 1 - 2e^{-\epsilon^2 n/6}$$

Follows from a concentration inequality (like Hoeffding, but different).

Johnson-Lindenstrauss Lemma

Let Q be a finite set of vectors in $\mathcal{R}^d$. Let 0<δ<1 and

$$\epsilon = \sqrt{\frac{6\ \log(2|Q|/\delta)}{n}} \leq 3$$

then, with probability 1-δ over choice of random matrix W where elements are normally distributed with zero mean and variance of $1/n$,

$$\sup_{\mathbf{x}\in Q}\left|\frac{||W\mathbf{x}||^2}{||\mathbf{x}||^2} - 1\right| < \epsilon$$

Observations

Uses union bound.

Doesn't depend on d.

Also retains inner products.

If data was linearly separable, most likely remains so.

Can actually expand dimensionality and get new features for learning.