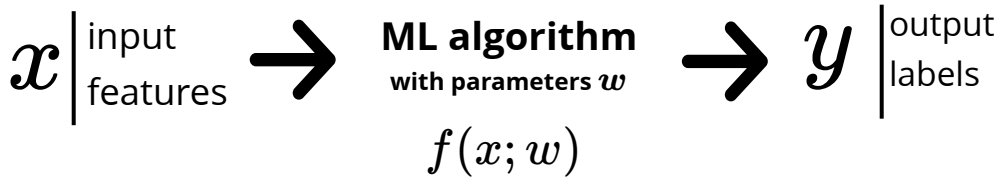
Applied Machine Learning

Review

Reihaneh Rabbany

ML Models



The core problem in ML is parameter estimation (aka model fitting), which requires solving an optimization problem of the loss/cost function

$$J(w) = \frac{1}{N}\sum_{n=1}^{N} l(y^{(n)}, f(x^{(n)}; w))$$

$$w^* = \arg\min_w J(w)$$

Linear Regression:

model: $\hat{y} = f_w(x) = w^\top x \ : \mathbb{R}^D \to \mathbb{R}$

cost function: $J_w = \frac{1}{N}\sum_n \frac{1}{2}(y^{(n)} - \hat{y}^{(n)})^2$

partial derivatives: $\frac{\partial}{\partial w_d} J_w = \frac{1}{N}\sum_n (\hat{y}^{(n)} - y^{(n)}) x_d^{(n)}$

gradient: vector of all partial derivatives: $\nabla J(w) = \frac{1}{N}\sum_n (\hat{y}^{(n)} - y^{(n)}) x^{(n)}$

Dx1

Logistic Regression:

$\hat{y} = f_w(x) = \sigma(w^\top x) : \mathbb{R}^D \to \{0, 1\}$

$J_w = \frac{1}{N}\sum_n -y \log(\hat{y}^{(n)}) - (1 - y^{(n)}) \log(1 - \hat{y}^{(n)})$

use SGD to find w^* given $\nabla J(w)$

Linear model of regression

$$f_w(x) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

model parameters or weights

bias or intercept

simplification

concatenate a 1 to x $\longrightarrow$ $x = [1, x_1, \ldots, x_D]^\top$

$$f_w(x) = w^\top x$$

$$w = [w_0, w_1, \ldots, w_D]^\top$$

Logistic regression: model

recall the way we included a **bias** parameter $x = [1, x_1]$

the input feature is generated uniformly in [-5,5]
for all the values less than 2 we have y=1 and y=0 otherwise

a good fit to this data is the one shown (green)

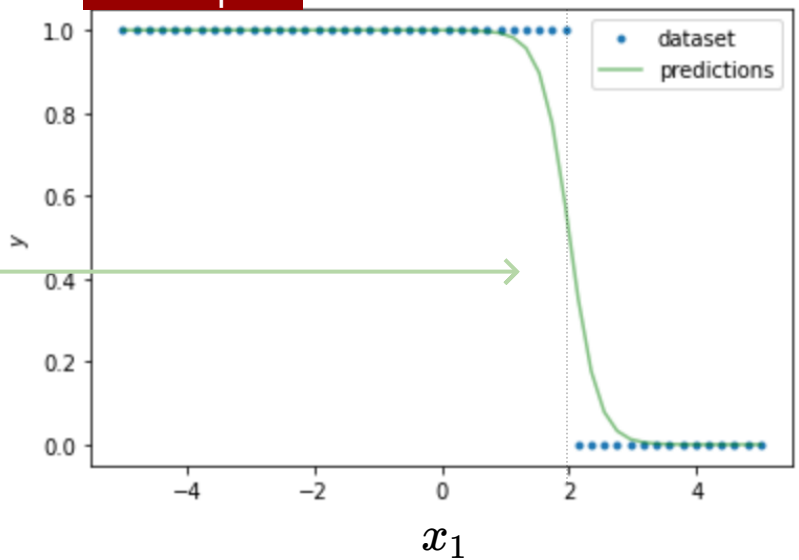
$$f_w(x) = \sigma(w^\top x) = \frac{1}{1+e^{-w^\top x}}$$

in the model shown $w \approx [9.1, -4.5]$

that is $\hat{y} = \sigma(-4.5x_1 + 9.1)$

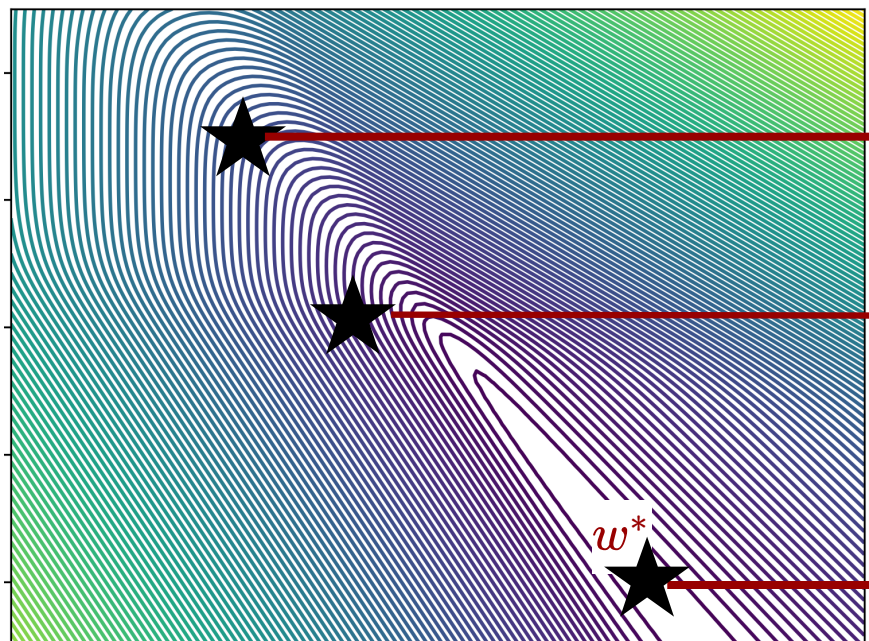
what is our model's decision boundary?

example

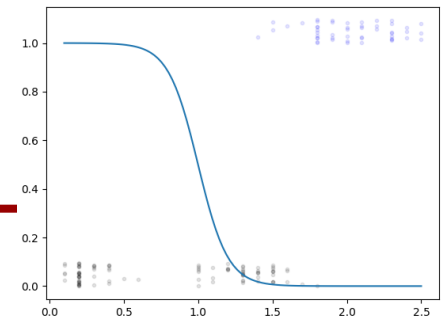


Example: binary classification

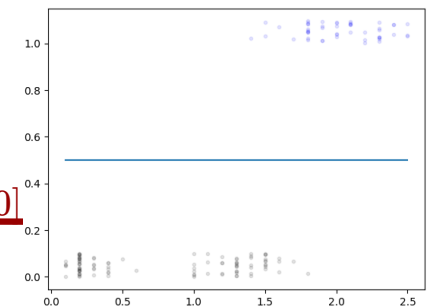
$J(w)$ as a function of these weights

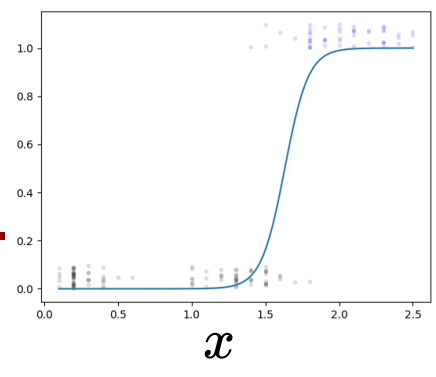


we have two weights associated with bias + petal width



$w = [0, 0]$





$\leftarrow \sigma(w_0^* + w_1^* x)$

(petal width)

Gradient descent

an iterative algorithm for optimization

- starts from some $w^{\{0\}}$
- update using gradient $w^{\{t+1\}} \leftarrow w^{\{t\}} - \alpha \nabla J(w^{\{t\}})$

t indicates the iteration step

steepest descent direction

learning rate

cost function

(for maximization : objective function)

converges to a local minima

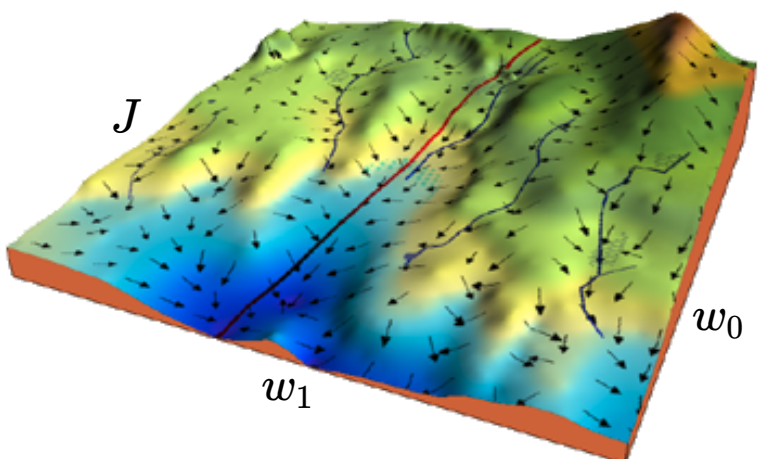


image from here

$$\nabla J(w) = [\frac{\partial}{\partial w_1} J(w), \cdots \frac{\partial}{\partial w_D} J(w)]^T$$

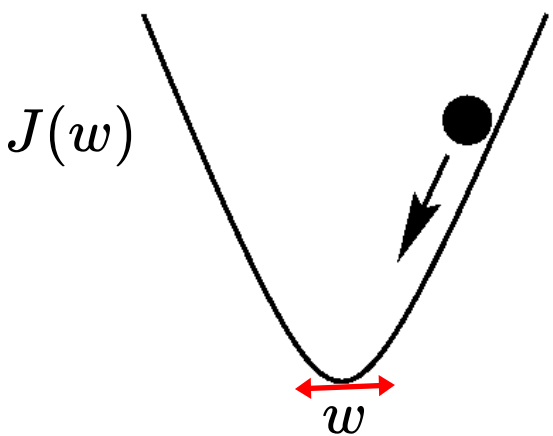
Minimum of a convex function

Convex functions are easier to minimize:

- critical points are global minimum
- gradient descent can find it

$$w^{\{t+1\}} \leftarrow w^{\{t\}} - \alpha \nabla J(w^{\{t\}})$$

convex



non-convex: gradient descent may find a local optima

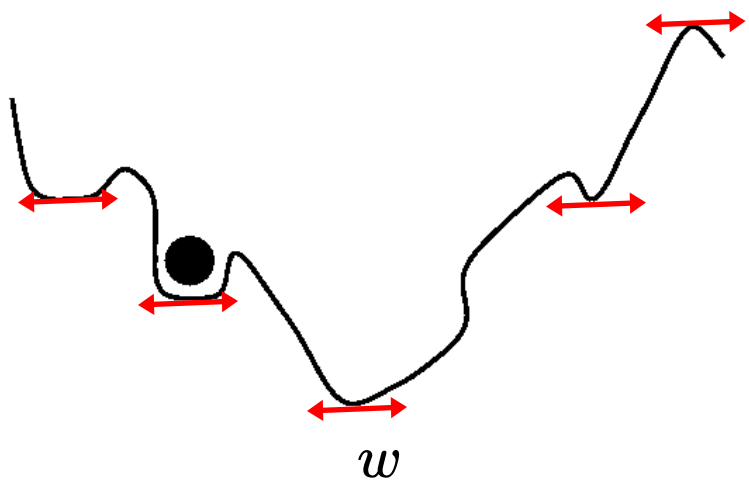


image from here

Recognizing convex functions

the logistic regression cost function is convex in model parameters (w)

$$J(w) = \frac{1}{N}\sum_{n=1}^{N} y^{(n)} \log\left(1 + e^{-w^\top x}\right) + \left(1 - y^{(n)}\right) \log\left(1 + e^{w^\top x}\right)$$

same argument

non-negative

linear

checking second derivative

$$\frac{\partial^2}{\partial z^2} \log(1 + e^z) = \frac{e^{-z}}{(1+e^{-z})^2} \geq 0$$

sum of convex functions

Gradient for linear and logistic regression

recall

in both cases: $\nabla J(w) = \frac{1}{N}\sum_n x^{(n)}(\hat{y}^{(n)} - y^{(n)}) = \underset{D\times N}{\frac{1}{N}X^\top}\underset{N\times 1}{(\hat{y}}\underset{N\times 1}{- y)}$

linear regression: $\hat{y} = w^\top x$

logistic regression: $\hat{y} = \sigma(w^\top x)$

```
def gradient(x, y, w):
    N,D = x.shape
    yh = logistic(np.dot(x, w))
    grad = np.dot(x.T, yh - y) / N
    return grad
```

time complexity: $\mathcal{O}(ND)$

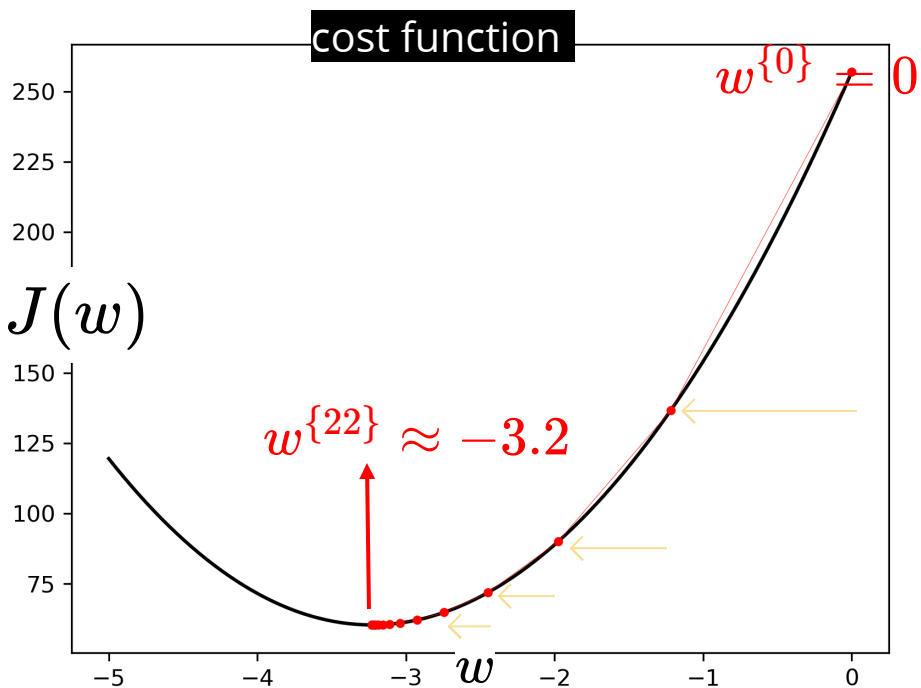
(two matrix multiplications)

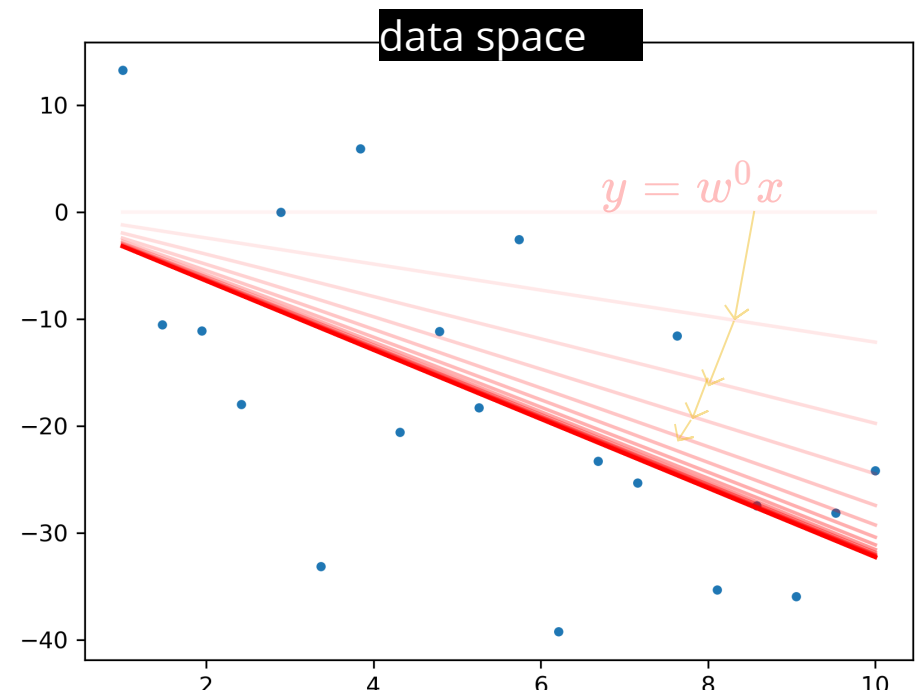
compared to the direct solution for linear regression: $\mathcal{O}(ND^2 + D^3)$

gradient descent can be much faster for large D

example GD for linear regression

After 22 steps $w^{\{t+1\}} \leftarrow w^{\{t\}} - .01 \nabla J(w^{\{t\}})$

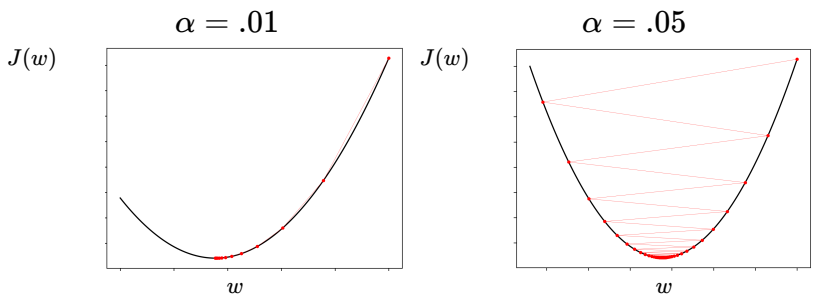




Learning rate α

Learning rate has a significant effect on GD

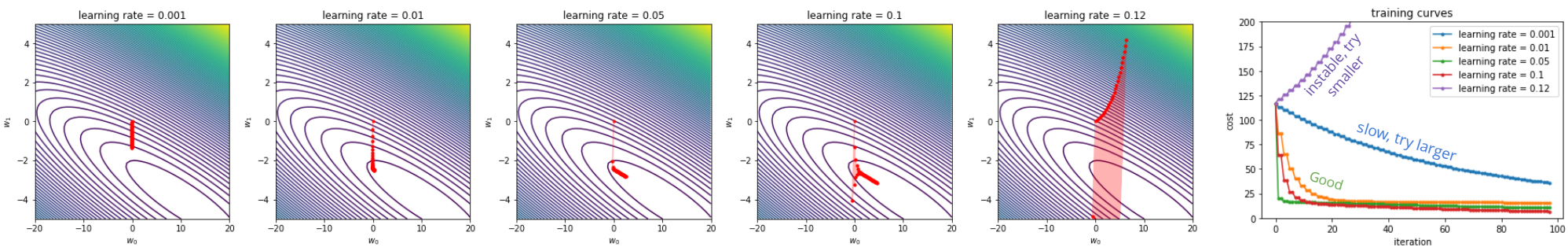
example, D=1
linear regression



too **small**: may take a long time to converge

too **large**: it overshoots or even diverges

example, D=2
linear regression
50 gradient steps



do a grid search usually between 0.001 to .1 to find the right value, look at the training curves

Stochastic Gradient Descent

we can write the cost function as an average over instances

$$J(w) = \frac{1}{N} \sum_{n=1}^{N} J_n(w)$$

cost for a single data-point

e.g. for linear regression

$$J_n(w) = \frac{1}{2}(w^T x^{(n)} - y^{(n)})^2$$

the same is true for the partial derivatives $\frac{\partial}{\partial w_j} J(w) = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial w_j} J_n(w)$

therefore $\nabla J(w) = \mathbb{E}_{\mathcal{D}}[\nabla J_n(w)]$

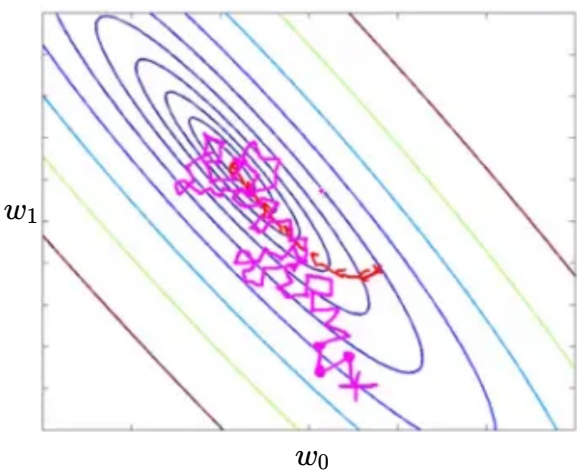
Stochastic Gradient Descent

Idea: use stochastic approximations $\nabla J_n(w)$ in gradient descent

stochastic gradient update

$$w \leftarrow w - \alpha \nabla J_n(w)$$

the steps are "on average" in the right direction



each step is using gradient of a different cost, $J_n(w)$

each update is (1/N) of the cost of batch gradient

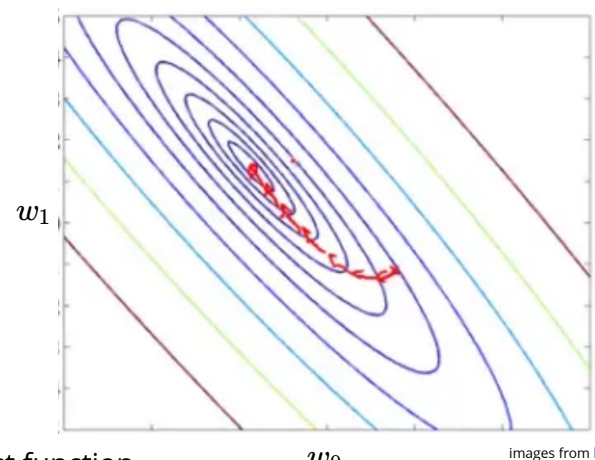
e.g., for linear regression $\mathcal{O}(D)$

$$\nabla J_n(w) = x^{(n)}(w^\top x^{(n)} - y^{(n)})$$

batch gradient update

$$w \leftarrow w - \alpha \nabla J(w)$$

with small learning rate: guaranteed improvement at each step



contour plot of the cost function

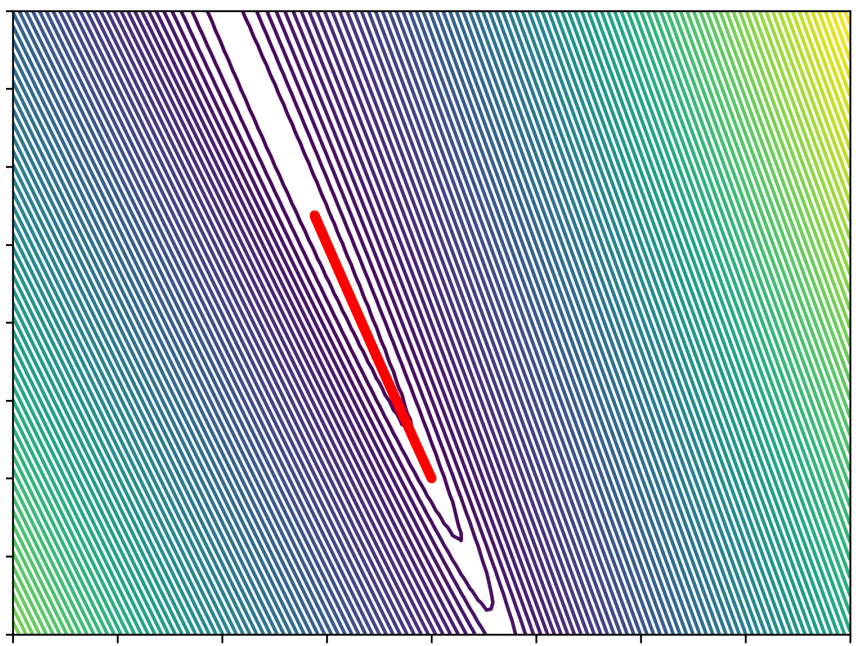
images from here

SGD for logistic regression

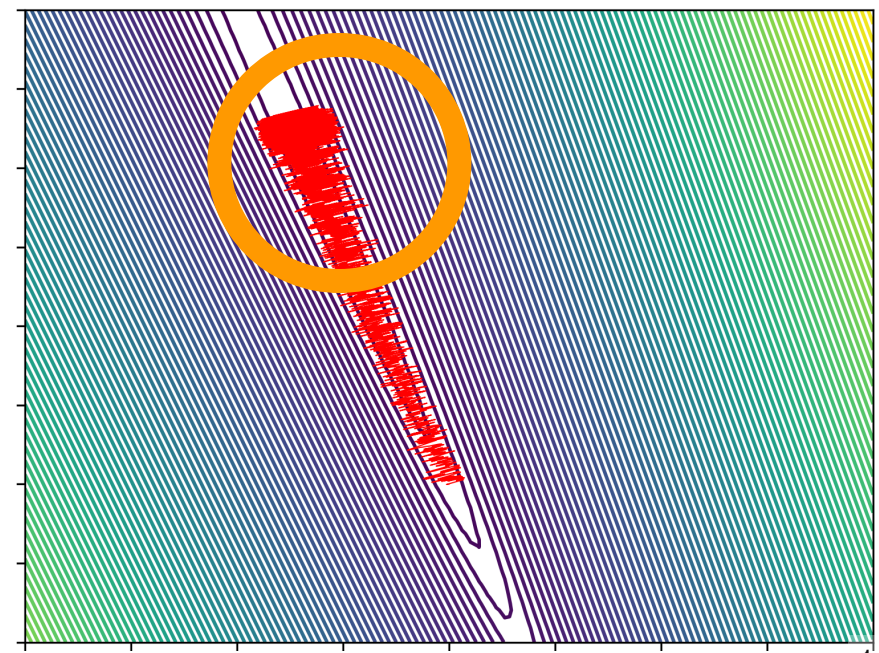
example

logistic regression for Iris dataset (D=2 , $\alpha = .1$)

batch gradient



stochastic gradient

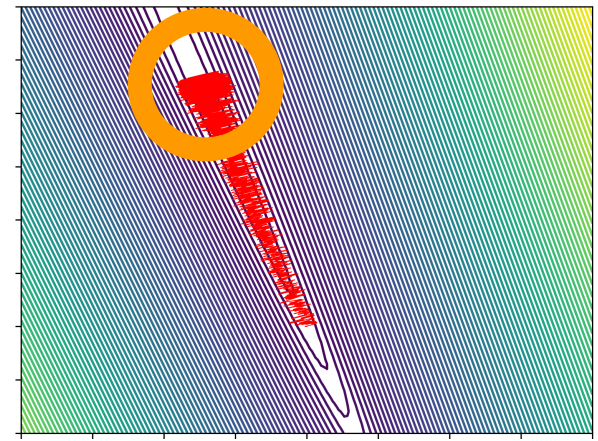


Convergence of SGD

stochastic gradients are not zero even at the optimum w

how to guarantee convergence?

idea: schedule to have a smaller learning rate over time



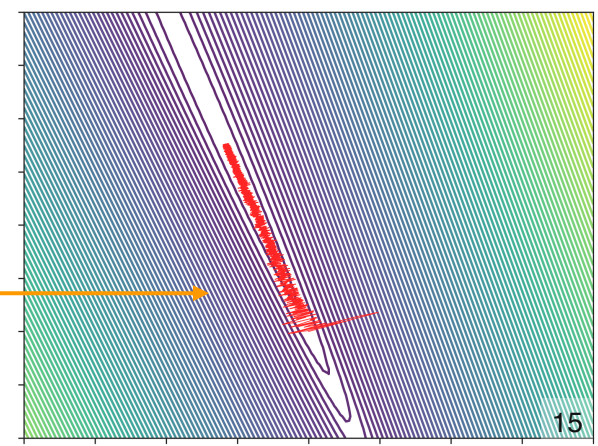
Robbins Monro

the sequence we use should satisfy: $\sum_{t=0}^{\infty} \alpha^{\{t\}} = \infty$

& otherwise for large $||w^{\{0\}} - w^*||$ we can't reach the minimum

the steps should go to zero $\sum_{t=0}^{\infty} (\alpha^{\{t\}})^2 < \infty$

example $\alpha^{\{t\}} = \frac{10}{t}, \alpha^{\{t\}} = t^{-.51}$



read more here

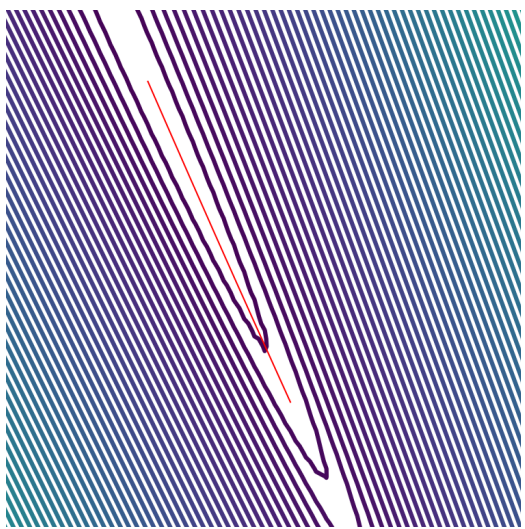
Minibatch SGD

use a minibatch to produce gradient estimates

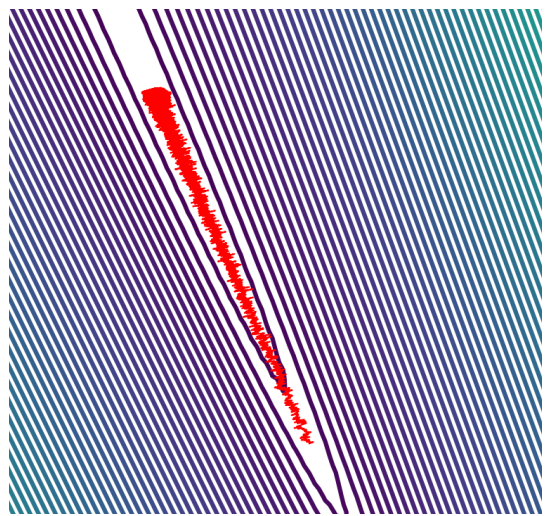
$$\nabla J_{\mathbb{B}} = \frac{1}{|\mathbb{B}|} \sum_{n \in \mathbb{B}} \nabla J_n(w)$$

$\mathbb{B} \subseteq \{1, \ldots, N\}$ a subset of the dataset

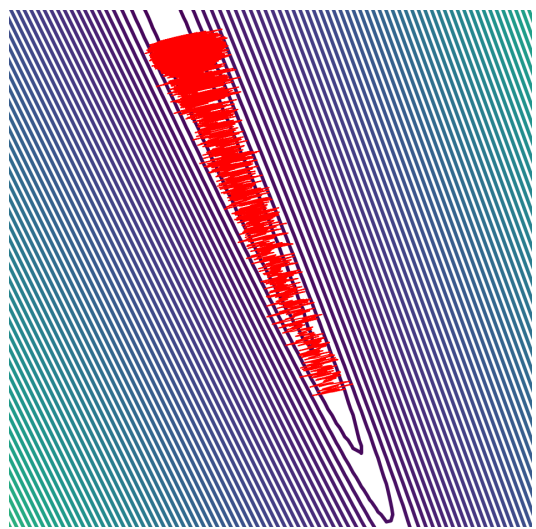
GD full batch



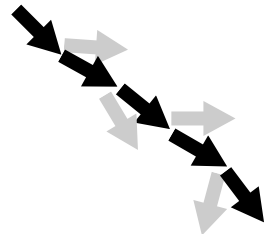
SGD minibatch-size=16



SGD minibatch-size=1



Momentum



to help with oscillations:

- use a **running average** of gradients
- more recent gradients should have higher weights

$$\Delta w^{\{t\}} \leftarrow \beta \Delta w^{\{t-1\}} + (1-\beta)\nabla J_{\mathbb{B}}(w^{\{t-1\}})$$

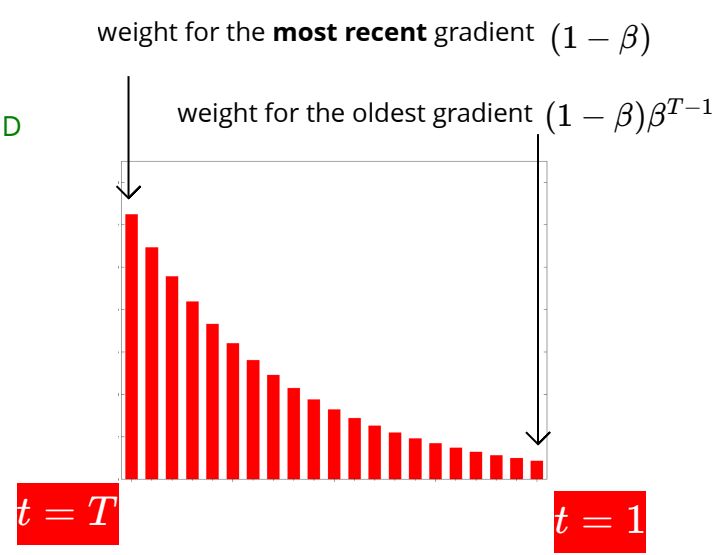
$$w^{\{t\}} \leftarrow w^{\{t-1\}} - \alpha \Delta w^{\{t\}}$$

momentum of 0 reduces to SGD
common value > .9

is effectively an exponential moving average

$$\Delta w^{\{T\}} = \sum_{t=1}^{T} \beta^{T-t}(1-\beta)\nabla J_{\mathbb{B}}(w^{\{t\}})$$

there are other variations of momentum with similar idea

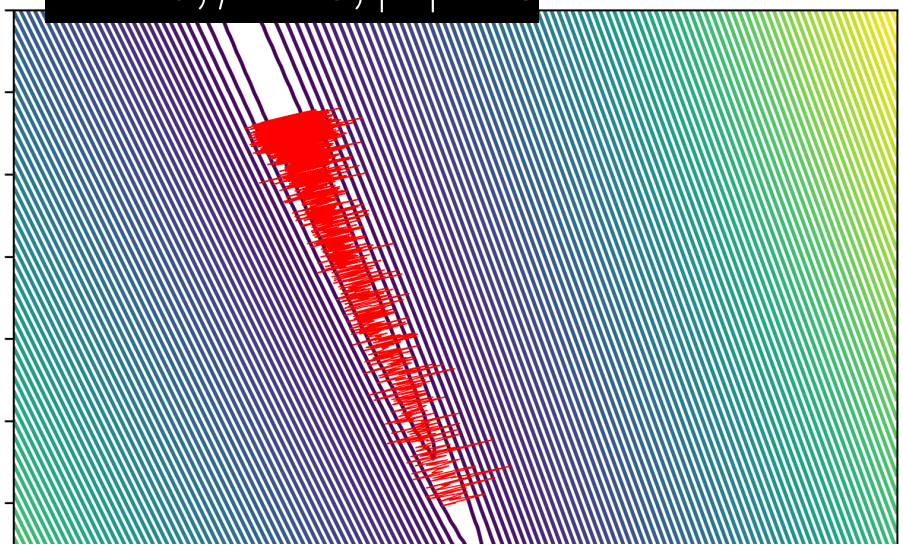


Momentum

Example: logistic regression

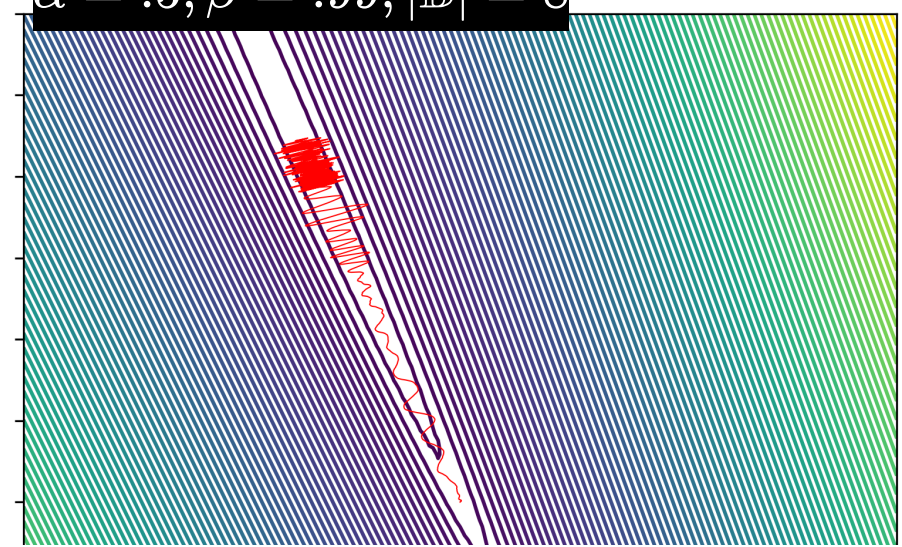
no momentum

$\alpha = .5, \beta = 0, |\mathbb{B}| = 8$



with momentum

$\alpha = .5, \beta = .99, |\mathbb{B}| = 8$



see the beautiful demo at Distill https://distill.pub/2017/momentum/

RMSprop (Root Mean Squared propagation)

solve the problem of diminishing step-size with Adagrad

- use exponential moving average instead of sum (similar to momentum)

instead of Adagrad: $S_d^{\{t\}} \leftarrow S_d^{\{t-1\}} + \frac{\partial}{\partial w_d} J(w^{\{t-1\}})^2$

$$S^{\{t\}} \leftarrow \gamma S^{\{t-1\}} + (1-\gamma) \nabla J(w^{\{t-1\}})^2$$

$$w^{\{t\}} \leftarrow w_{\{t-1\}} - \frac{\alpha}{\sqrt{S^{\{t\}}+\epsilon}} \nabla J(w^{\{t-1\}})$$

identical to Adagrad

note that $S^{\{t\}}$ here is a vector and with the square root is element-wise

Adam (Adaptive Moment Estimation)

the default algorithm in practice

two ideas so far:
1. use momentum to smooth out the oscillations
2. adaptive per-parameter learning rate

both use exponential moving averages

Adam **combines the two**:

$$M^{\{t\}} \leftarrow \beta_1 M^{\{t-1\}} + (1-\beta_1)\nabla J(w^{\{t-1\}})$$

identical to method of momentum
(moving average of the first moment)

$$S^{\{t\}} \leftarrow \beta_2 S^{\{t-1\}} + (1-\beta_2)\nabla J(w^{\{t-1\}})^2$$

identical to RMSProp
(moving average of the second moment)

$$w^{\{t\}} \leftarrow w^{\{t-1\}} - \frac{\alpha}{\sqrt{\hat{S}^{\{t\}}}+\epsilon}\hat{M}^{\{t\}}$$

since M and S are initialized to be zero, at early stages they are biased towards zero

$$\hat{M}^{\{t\}} \leftarrow \frac{M^{\{t\}}}{1-\beta_1^t} \qquad \hat{S}^{\{t\}} \leftarrow \frac{S^{\{t\}}}{1-\beta_2^t}$$

for large time-steps it has no effect
for small t, it scales up numerator

Loss, cost and generalization

assume we have a **model** $f : x \mapsto y$ for example $f :$  $\mapsto 3$

and we have a **loss function** that measures the error in our prediction $\ell : y, \hat{y} \to \mathbb{R}$

for example

$$\begin{cases} \ell(y, \hat{y}) = (y - \hat{y})^2 & \text{for regression} \\ \ell(y, \hat{y}) = \mathbb{I}(y \neq \hat{y}) & \text{for classification} \end{cases}$$

we train our models to minimize **the cost function**:

$$J = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{x,y \in \mathcal{D}_{\text{train}}} \ell(y, f(x))$$

We can drop this, why?

how to estimate this?

what we really care about is the **generalization error:** $\mathbb{E}_{x,y \sim p}\, \ell(y, f(x))$

we can not measure this, why?

we can set aside part of the given data and use it to estimate generalization error

Bias-variance decomposition: Setup

decompose the generalization error to see the effect of bias and variance (for L2 loss)

assume a true distribution $p(x, y)$

best prediction given L2 loss $f(x) = \mathbb{E}_p[y|x]$

assume that a dataset $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_n$ is sampled from $p(x, y)$

let $\hat{f}_{\mathcal{D}}$ be our model based on the dataset

what we care about is the generalization error (aka expected loss, expected risk)

$$\mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - y)^2]$$

all blue items are random variables

Bias-variance decomposition

what we care about is the generalization error

$$\mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - y)^2] = \mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)] - y + \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2]$$

y: $f(x) + \epsilon$

$\hat{f}_{\mathcal{D}}(x)$: $\hat{f}_{\mathcal{D}}(x) + \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)] - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)]$ add and subtract a term

above simplifies to the following (the remaining terms are going to be zero)

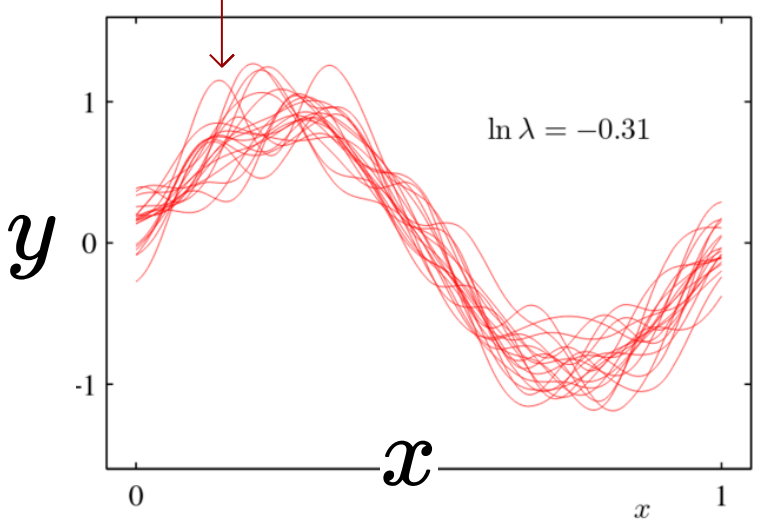
$$= \mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2] + \mathbb{E}[(f(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2] + \mathbb{E}[\epsilon^2]$$

variance — bias^2 — unavoidable noise error

Example: bias vs. variance

models for different datasets $\hat{f}_{\mathcal{D}}$
using Gaussian bases

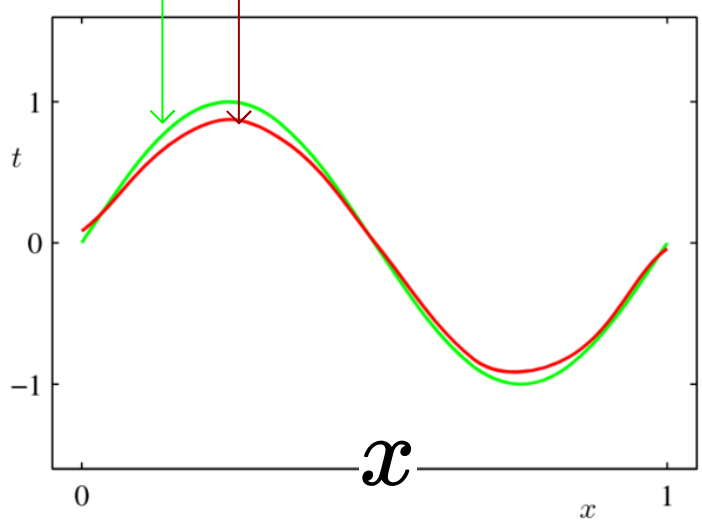
random datasets of size N=25 instances are not shown



variance is the average difference (in squared L2 norm) between these curves and their average

true model f

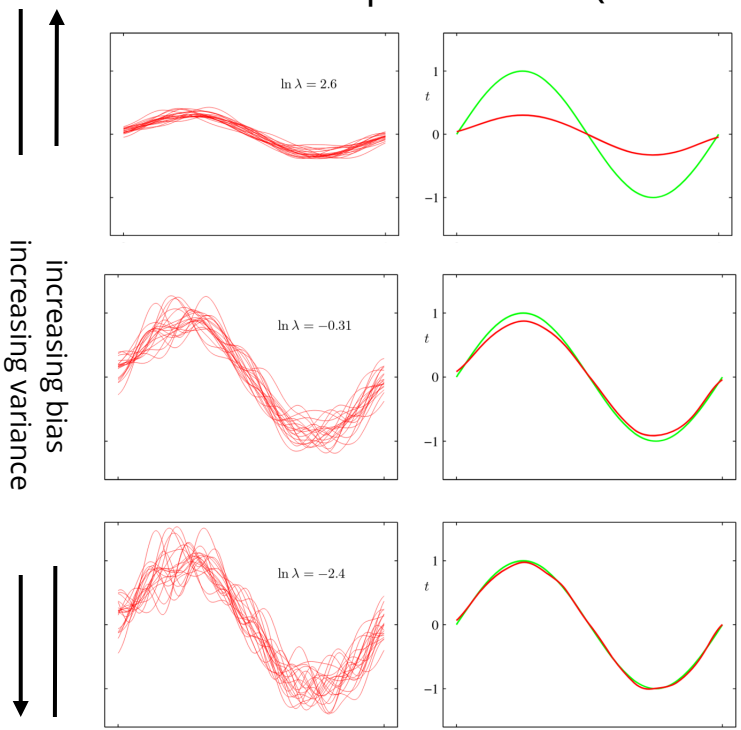
their average $\mathbb{E}[\hat{f}_{\mathcal{D}}]$

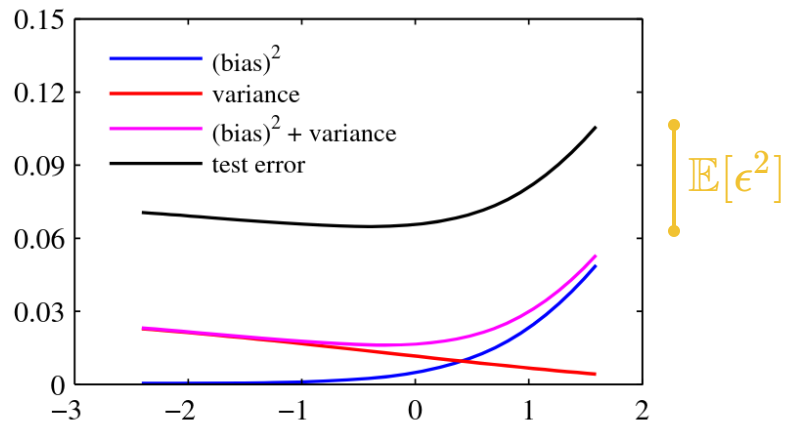


bias is the difference (in L2 norm) between two curves

Example: bias vs. variance

the lowest expected loss (test error) is somewhere between the two extremes

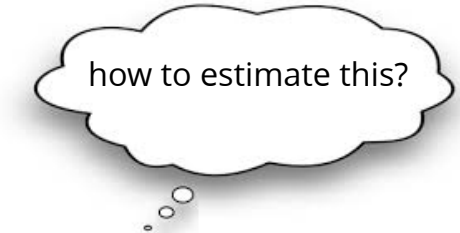




$$= \mathbb{E}[(\hat{f}_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2] \quad + \mathbb{E}[(f(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)])^2] \quad + \mathbb{E}[\epsilon^2]$$

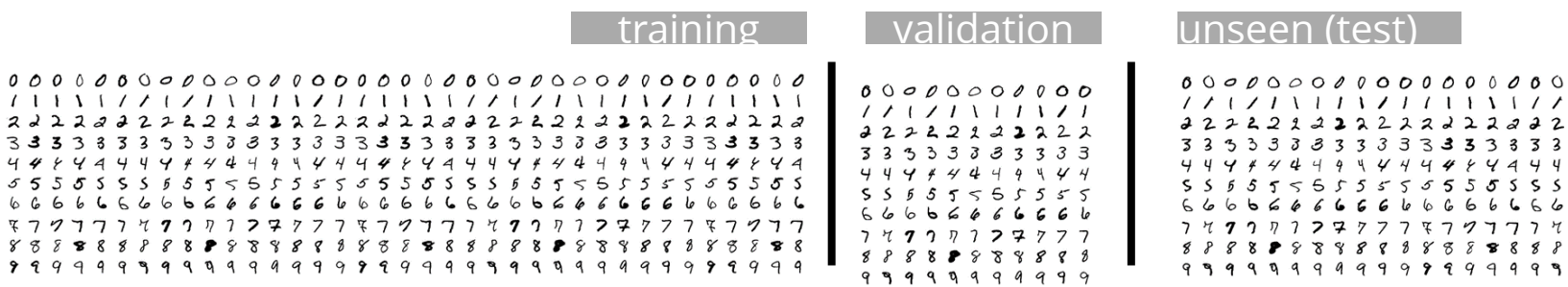
in practice, how to decide which model to use?

Validation set



what we really care about is the **generalization error:** $\mathbb{E}_{x,y\sim p}\ \ell(y, f(x))$

we can set aside part of the training data and use it to **estimate** the generalization error

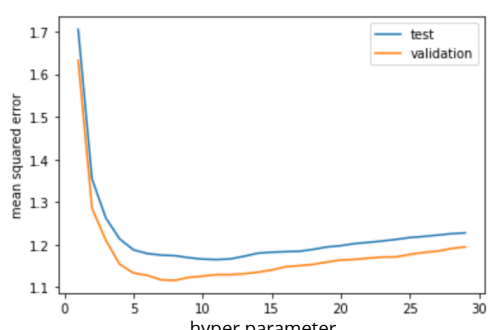


pick a hyper-parameter that gives us the best **validation error**

at the very end, we report the error on **test set**

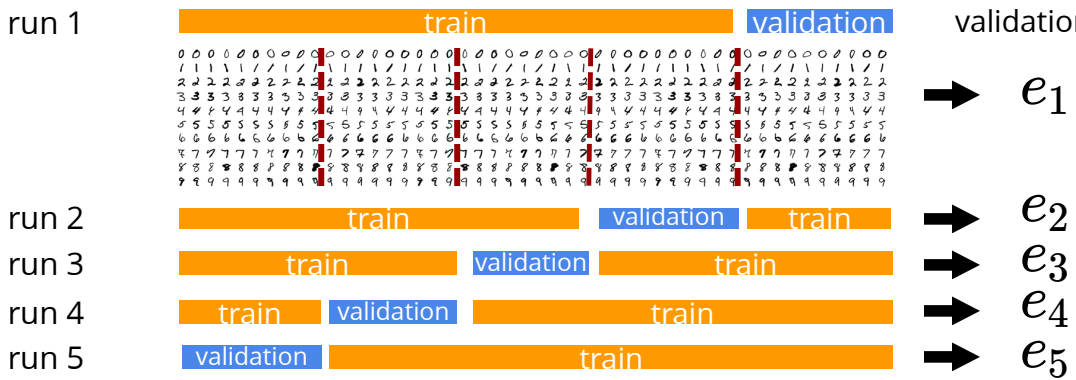
validation and test error could be different

because they use limited amount of data



Cross validation

- divide the (training + validation) data into L parts
- use one part for validation and L-1 for training



validation errc.

e_1

e_2

e_3

e_4

e_5

$$\bar{e} = \frac{1}{5}\sum_{i=1}^{5} e_i$$

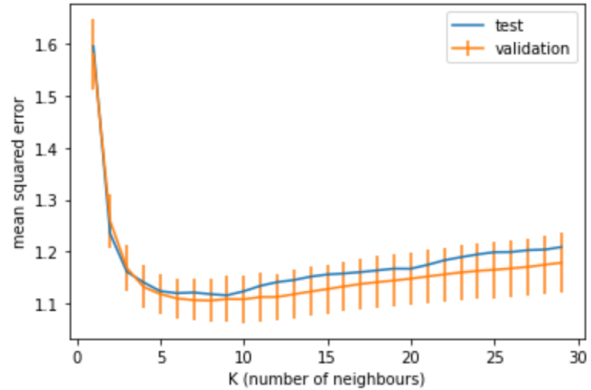
- use the **average** validation error and its variance (uncertainty) to pick the best model

- report the test error for the final model

e_t

this is called **L-fold** cross-validation

in **leave-one-out** cross-validation L=N (only one instance is used for validation)



Performance metrics for classification

Not all errors are the same

In particular in classification, we have different types of mistakes

false positive (type I) and false negative (type II)

example:

patient does not have disease but received positive diagnostic (Type I error)
patient has disease but it was not detected (Type II error)

a message that is not spam is assigned to the spam folder (Type I error)

a message that is spam appears in the regular folder (Type II error)

Performance metrics for classification

confusion matrix

		Truth		Σ
Result		TP	FP	RP
		FN	TN	RN
Σ		P	N	

example:

		Truth		Σ
Result		14	2	16
		3	11	14
Σ		17	13	

$$Precision = \frac{TP}{RP} = \frac{14}{16}$$

$$Recall = \frac{TP}{P} = \frac{14}{17}$$

$$\frac{1}{F_\beta score} = \frac{1}{1+\beta^2}\frac{1}{Precision} + \frac{\beta^2}{1+\beta^2}\frac{1}{Recall}$$

$$Accuracy = \frac{TP+TN}{P+N}$$

$$Precision = \frac{TP}{RP}$$

$$Recall = \frac{TP}{P}$$ sensitivity

$$F_1 score = 2\frac{Precision \times Recall}{Precision + Recall}$$ {Harmonic mean}

$$F_\beta score = (1+\beta^2)\frac{Precision \times Recall}{\beta^2 Precision + Recall}$$

recall is β times more important compared to precision

less common

$$Miss\ rate = \frac{FN}{P}$$

$$Fallout = \frac{FP}{N}$$ false positive rate

$$False\ discovery\ rate = \frac{FP}{RP}$$

$$Selectivity = \frac{TN}{N}$$ specificity

$$False\ omission\ rate = \frac{FN}{RN}$$

$$Negative\ predictive\ value = \frac{TN}{RN}$$

Trade-off between precision and recall

goal: evaluate class scores/probabilities (independent of choice of threshold)

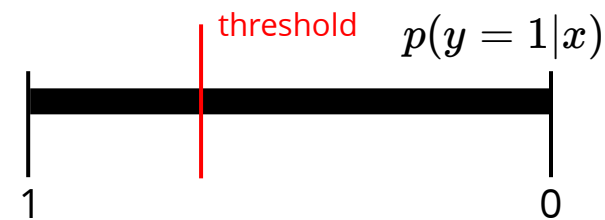
Receiver Operating Characteristic **ROC curve,** a function of threshold t

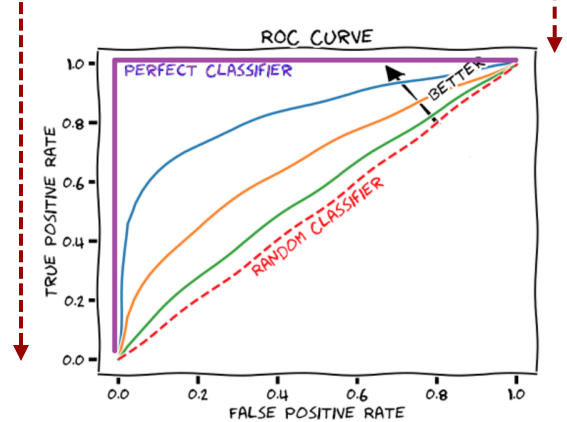
TPR(t) = TP(t)/P (**recall**, sensitivity at t, hit rate)
FPR(t) = FP(t)/N (**fallout**, false alarm at t, type I error rate)

Area Under the Curve (**AUC**) is used as a threshold independent measure of quality of the classifier

$AUC = \sum_t TPR(t)(FPR(t) - FPR(t-1))$, box-rule approximation

Most ML algorithm produces class score or probability





Nonlinear basis functions

replace original features in $f_w(x) = \sum_d w_d x_d$

with nonlinear bases $f_w(x) = \sum_d w_d \, \phi_d(x)$

linear least squares solution $(\Phi^\top \Phi) w^* = \Phi^\top y$

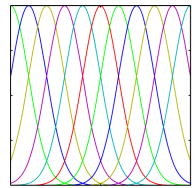
replacing X with Φ

a (nonlinear) feature

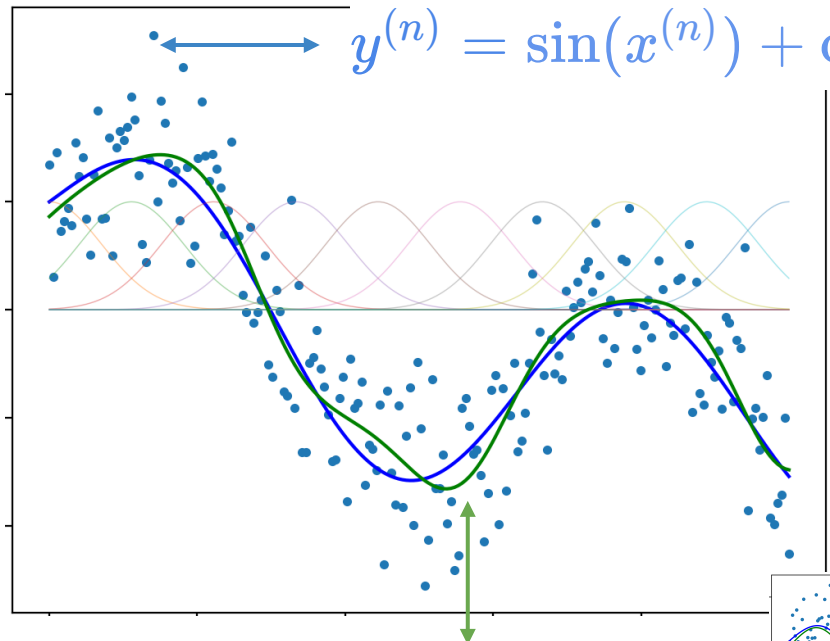
$$\Phi = \begin{bmatrix} \phi_1(x^{(1)}), & \phi_2(x^{(1)}), & \cdots, & \phi_D(x^{(1)}) \\ \phi_1(x^{(2)}), & \phi_2(x^{(2)}), & \cdots, & \phi_D(x^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x^{(N)}), & \phi_2(x^{(N)}), & \cdots, & \phi_D(x^{(N)}) \end{bmatrix}$$

one instance

Example: Gaussian bases

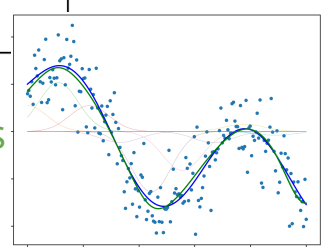


$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

our fit to data using 10 Gaussian bases



prediction for a new instance

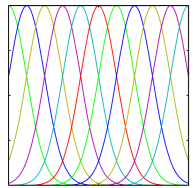
$$f(x') = \phi(x')^\top (\Phi^\top \Phi)^{-1} \Phi^\top y$$

x': new instance

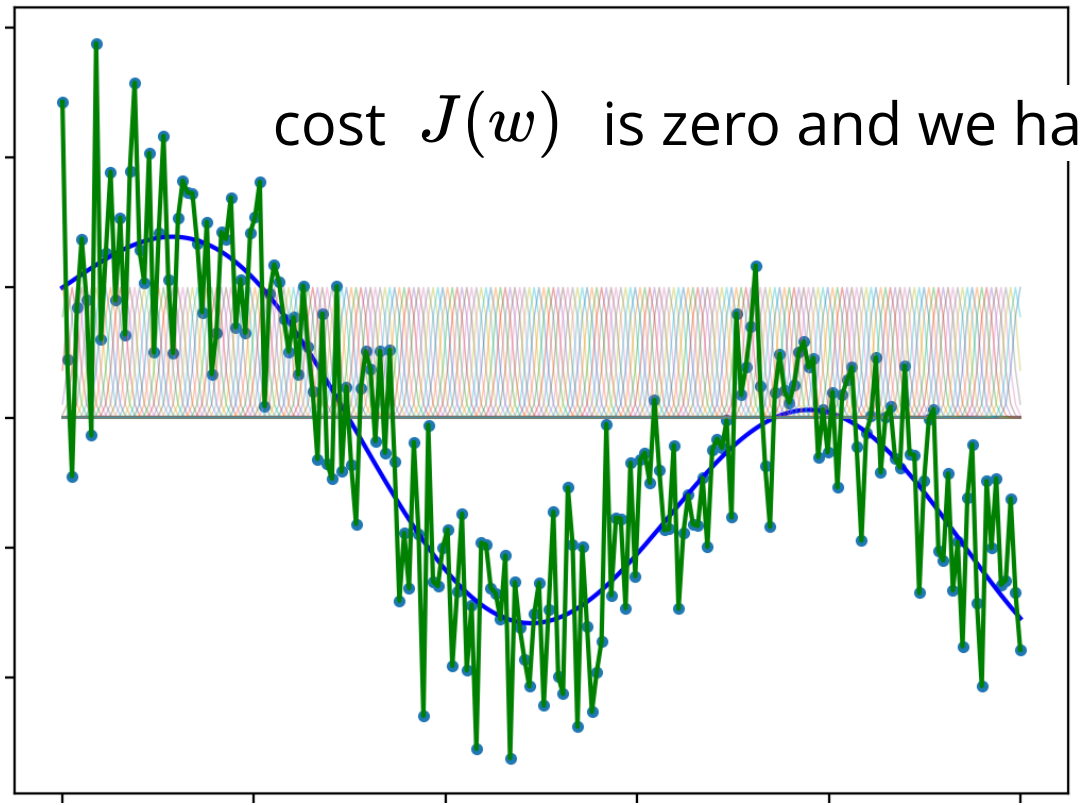
$\phi(x')$: features evaluated for the new point

w found using LLS

Example: Gaussian bases



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

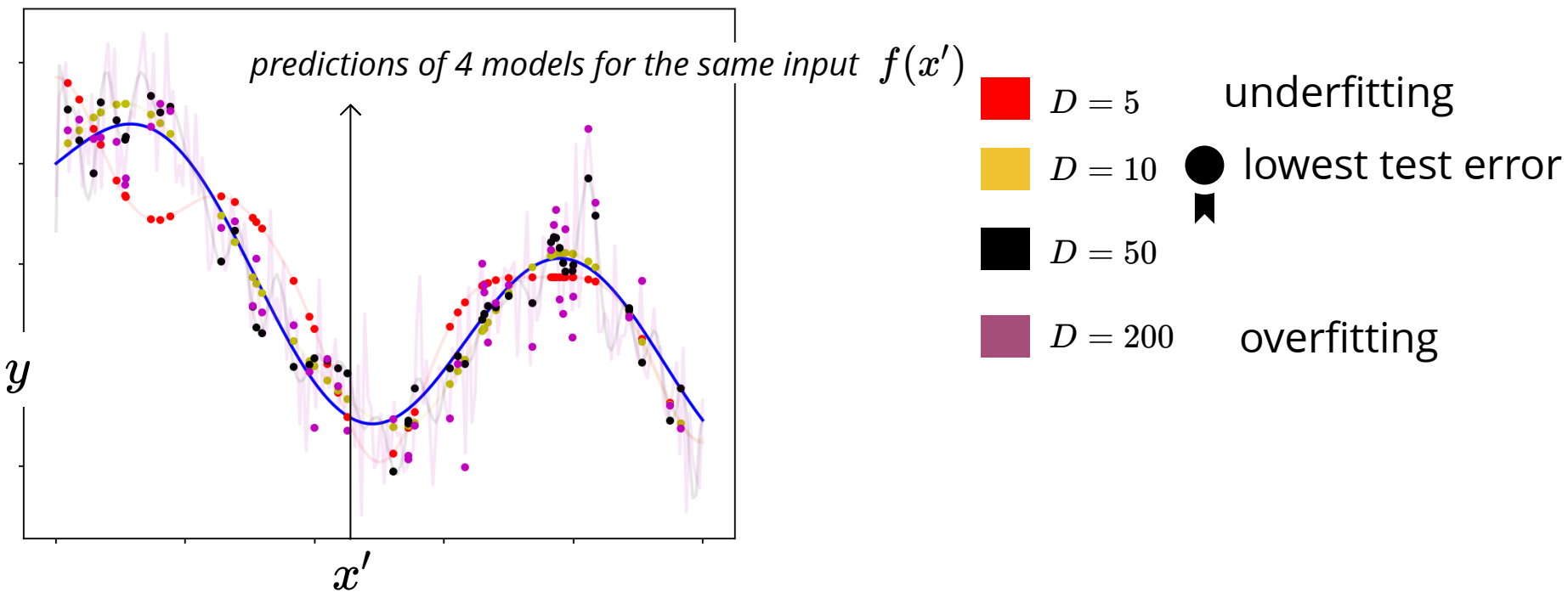


cost $J(w)$ is zero and we have a "*perfect*" fit!

using 200, thinner bases (s=.1)

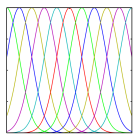
Overfitting

which one of these models performs better at test time?



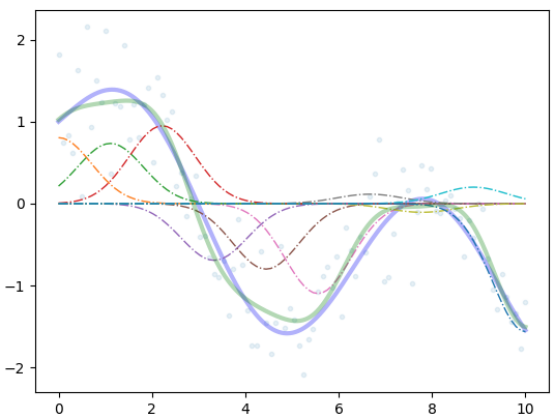
An observation

when overfitting, we sometimes see large weights

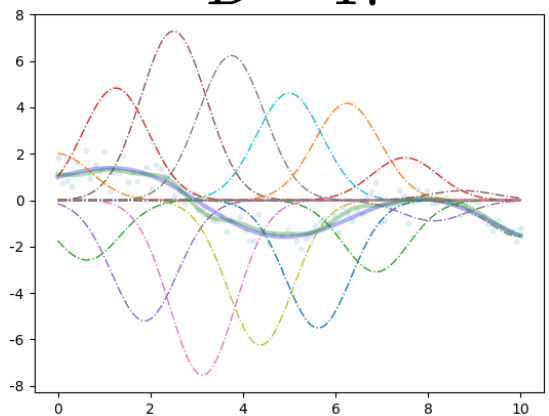


dashed lines are $w_d \phi_d(x) \quad \forall d$ $\qquad f_w(x) = \sum_d w_d \, \phi_d(x)$

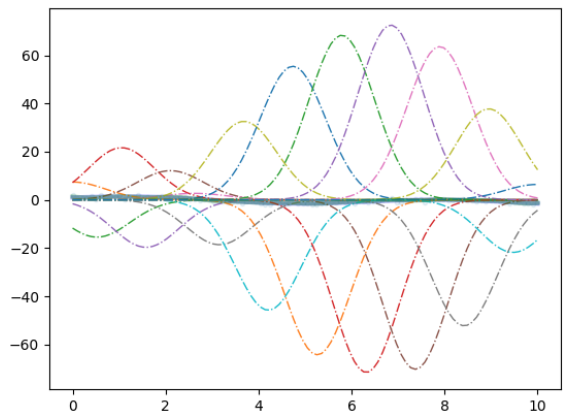
$D = 10$



$D = 17$



$D = 20$



idea: penalize large parameter values

Ridge regression

also known as

L2 regularized linear least squares regression:

$$J(w) = \frac{1}{2}||Xw - y||_2^2 + \frac{\lambda}{2}||w||_2^2$$

sum of squared error
$\frac{1}{2}\sum_n (y^{(n)} - w^\top x)^2$

squared L2 norm of w
$w^T w = \sum_d w_d^2$

regularization parameter $\lambda > 0$ controls the strength of regularization

a good practice is to **not** penalize the intercept $\lambda(||w||_2^2 - w_0^2)$

λ is a hyper-parameter (use a validation set or cross-validation to pick the best value)

Ridge regression

set the derivative to zero $J(w) = \frac{1}{2}\sum_{x,y\in\mathcal{D}}(y - w^\top x)^2 + \frac{\lambda}{2}w^\top w$

$$\nabla J(w) = \sum_{x,y\in\mathcal{D}} x(w^\top x - y) + \lambda w$$

$$= X^\top(Xw - y) + \lambda w = 0$$

when using gradient descent, this term reduces the weights at each step **(weight decay)**

linear system of equations $(X^\top X + \lambda \mathbf{I})w = X^\top y$

$$w = (X^\top X + \lambda \mathbf{I})^{-1} X^\top y$$

the only part different due to regularization

λI makes it invertible, adds a small value to the diagonals $X^\top X$

we can have linearly dependent features

the solution will be unique!

Probabilistic interpretation

recall linear regression & logistic regression maximize log-likelihood

$$w^{MLE} = \arg\max_w p(y|X, w)$$

linear regression $$w^{MLE} = \arg\max_w \prod_{x,y\in\mathcal{D}} \mathcal{N}\left(y|w^\top x, \sigma^2\right)$$

logistic regression $$w^{MLE} = \arg\max_w \prod_{x,y\in\mathcal{D}} \text{Bernoulli}\left(y|\sigma(w^\top x)\right)$$

can we do Bayesian inference instead of maximum likelihood?

$$p(w|y, X) \propto p(w)p(y|w, X)$$

posterior prior likelihood

Gaussian Prior

MAP estimate $w^{MAP} = \arg\max_w \log p(y|X,w) + \log p(w)$

prior

assume independent zero-mean Gaussians

$$\mathcal{N}(\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

$$\log p(w) = \log \prod_{d=1}^{D} \mathcal{N}(w_d|0,\tau^2) = -\sum_d \frac{w^2}{2\tau^2} + \text{const.}$$

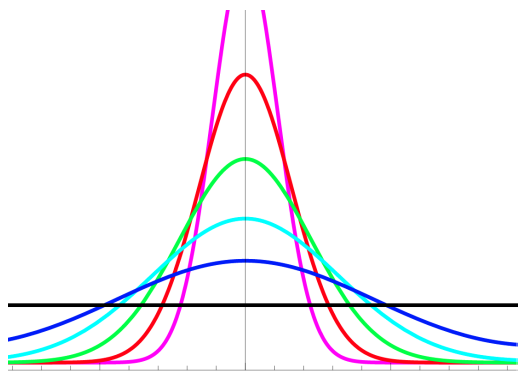
does not depend on w

so it doesn't affect the optimization

lets call $\frac{1}{\tau^2} \rightarrow \lambda$

then we get the L2 regularization penalty $\frac{\lambda}{2}||w||_2^2$

smaller variance of the prior τ gives larger regularization λ



Probabilistic interpretation for Linear Regression

idea

given the dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$

learn a probabilistic model $p(y|x; w)$

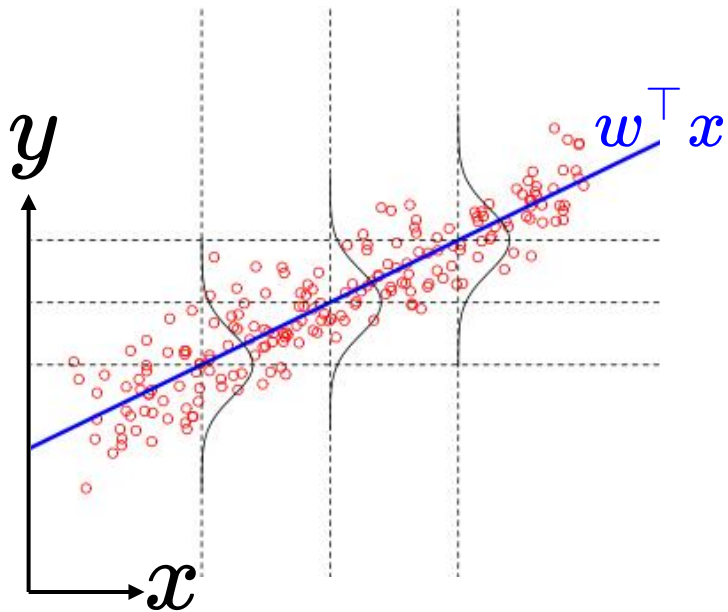


image from here

consider $p(y|x; w)$ with the following form

$$p_w(y \mid x) = \mathcal{N}(y \mid w^\top x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y - w^\top x)^2}{2\sigma^2}}$$

assume a fixed variance, say $\sigma^2 = 1$

Q: how to fit the model?

A: maximize the conditional likelihood!

Maximum likelihood & linear regression

cond. probability $p(y \mid x; w) = \mathcal{N}(y \mid w^\top x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y - w^\top x)^2}{2\sigma^2}}$

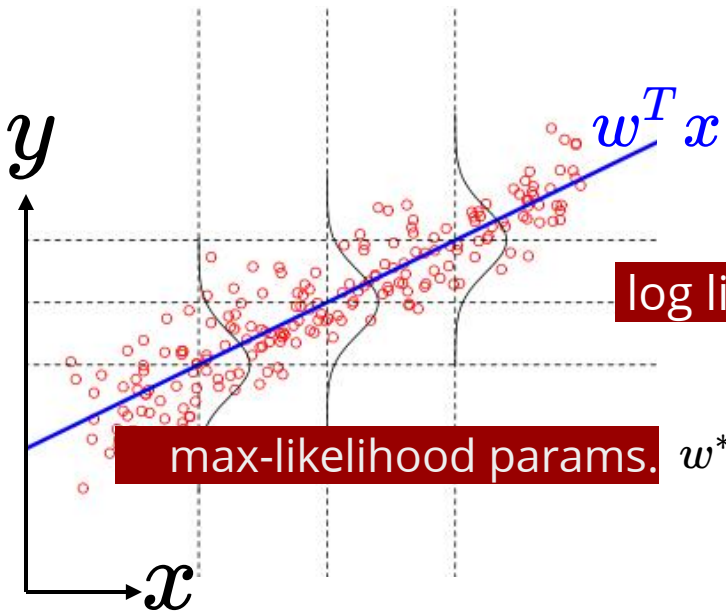


image from here

likelihood $L(w) = \prod_{n=1}^{N} p(y^{(n)} \mid x^{(n)}; w)$

log likelihood $\ell(w) = \sum_n -\frac{1}{2\sigma^2}(y^{(n)} - w^\top x^{(n)})^2 + \text{constants}$

max-likelihood params. $w^* = \arg\max_w \ell(w) = \arg\min_w \frac{1}{2} \sum_n (y^{(n)} - w^\top x^{(n)})^2$

linear least squares!

whenever we use square loss, we are assuming Gaussian noise!

Laplace prior

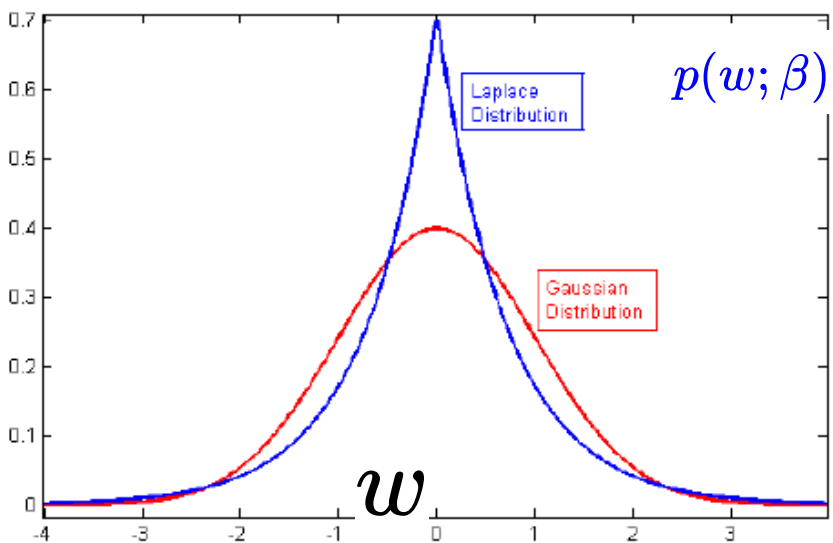
another notable choice of prior is the Laplace distribution

minimizing negative log-likelihood → $\sum_d \log p(w_d) = -\sum_d \frac{1}{\beta}|w_d| = -\frac{1}{\beta}||w||_1$

L1 norm of w

L1 regularization: $J(w) \leftarrow J(w) + \lambda||w||_1$ also called lasso

(least absolute shrinkage and selection operator)



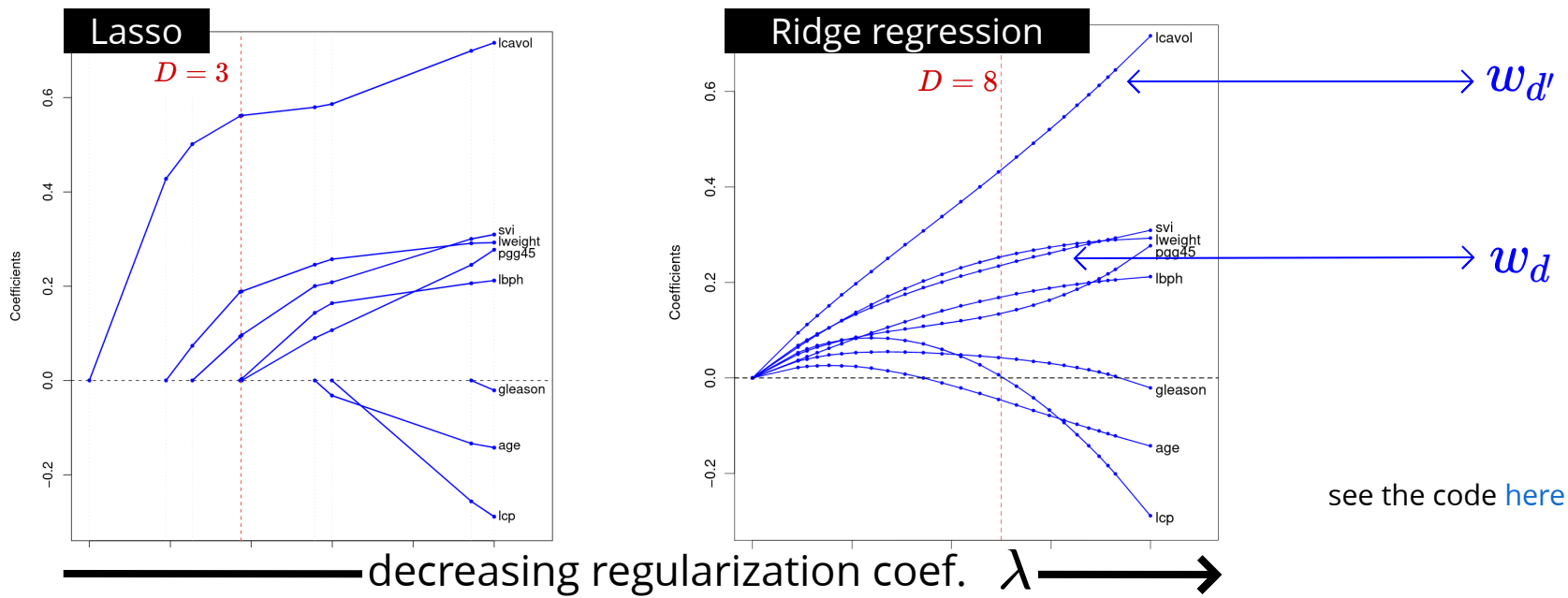
$p(w;\beta) = \frac{1}{2\beta}e^{-\frac{|w|}{\beta}}$ notice the peak around zero

image from here

L_1 vs L_2 regularization

regularization path shows how $\{w_d\}$ change as we change λ
Lasso produces sparse weights (many are zero, rather than small)

Example
$D = 8$



see the code here

red-line is the optimal λ from cross-validation, for lasso the model uses only 3 of the 8 features

$\Rightarrow$ lasso results in sparse models

Probabilistic view for Logistic Regression

Interpret the prediction as class probability $\hat{y} = p_w(y = 1 \mid x) = \sigma(w^\top x)$

the log-ratio of class probabilities is linear

$$\log \frac{\hat{y}}{1-\hat{y}} = \log \frac{\sigma(w^\top x)}{1-\sigma(w^\top x)} = \log \frac{1}{e^{-w^\top x}} = w^\top x$$

logit function
is the inverse of logistic, read more here

We have a Bernoulli likelihood

$$p(y^{(n)} \mid x^{(n)}; w) = \text{Bernoulli}(y^{(n)}; \sigma(w^\top x^{(n)})) = \hat{y}^{(n)^{y^{(n)}}} (1 - \hat{y}^{(n)})^{1-y^{(n)}}$$

conditional likelihood of the labels given the inputs

$$L(w) = \prod_{n=1}^{N} p(y^{(n)} \mid x^{(n)}; w) = \prod_{n=1}^{N} \hat{y}^{(n)^{y^{(n)}}} (1 - \hat{y}^{(n)})^{1-y^{(n)}}$$

Maximum likelihood & logistic regression

likelihood $L(w) = \prod_{n=1}^{N} p(y^{(n)} \mid x^{(n)}; w) = \prod_{n=1}^{N} \hat{y}^{(n)^{y^{(n)}}} (1 - \hat{y}^{(n)})^{1-y^{(n)}}$

find w that maximizes log likelihood

$$w^* = \max_w \sum_{n=1}^{N} \log p_w(y^{(n)} \mid x^{(n)}; w)$$

$$= \max_w \sum_{n=1}^{N} y^{(n)} \log(\hat{y}^{(n)}) + (1 - y^{(n)}) \log(1 - \hat{y}^{(n)})$$

$= \min_w J(w)$ the cross entropy cost function!

so using cross-entropy loss in logistic regression is maximizing conditional likelihood

we saw a similar interpretation for linear regression (L2 loss maximizes the conditional Gaussian likelihood)

Multiclass classification

using this probabilistic view we extend logistic regression to multiclass setting

binary classification: Bernoulli likelihood:

$$\text{Bernoulli}(y \mid \hat{y}) = \hat{y}^y (1-\hat{y})^{1-y} \quad \text{subject to} \quad \hat{y} \in [0,1]$$

$$\begin{cases} \hat{y} & y = 1 \\ 1-\hat{y} & y = 0 \end{cases}$$

↓

using logistic function to ensure this $\hat{y} = \sigma(z) = \sigma(w^T x)$

C classes: categorical likelihood

$$\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{I}(y=c)} \quad \text{subject to} \quad \sum_c \hat{y}_c = 1$$

$$\begin{cases} \hat{y}_1 & y = 1 \\ \hat{y}_2 & y = 2 \\ \cdots & \\ \hat{y}_C & y = C \end{cases}$$

↓

achieved using softmax function

Softmax

generalization of logistic to > 2 classes:

- **logistic**: $\sigma : \mathbb{R} \to (0, 1)$ produces a single probability
  - probability of the second class is $(1 - \sigma(z))$
- **softmax:** $\mathbb{R}^C \to \Delta_C$ recall: probability simplex $p \in \Delta_c \to \sum_{c=1}^{C} p_c = 1$

$$\hat{y}_c = \text{softmax}(z)_c = \frac{e^{z_c}}{\sum_{c'=1}^{C} e^{z_{c'}}} \quad \text{so} \quad \sum_c \hat{y} = 1$$

example $\text{softmax}([1, 1, 2, 0]) = [\frac{e}{2e+e^2+1}, \frac{e}{2e+e^2+1}, \frac{e^2}{2e+e^2+1}, \frac{1}{2e+e^2+1}]$

$$\text{softmax}([10, 100, -1]) \approx [0, 1, 0]$$

if input values are large, softmax becomes similar to argmax

similar to logistic this is also a squashing function

Multiclass classification

C classes: categorical likelihood

$\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{I}(y=c)}$ using softmax to enforce sum-to-one constraint

$$\hat{y}_c = \text{softmax}([w_1{}^\top x, \ldots, w_C{}^\top x])_c = \frac{e^{w_c{}^\top x}}{\sum_{c'} e^{w_{c'}{}^\top x}}$$

so we have on parameter **vector** for each class

$w_1 = [w_{1,1}, w_{1,2}, \ldots w_{1,D}]$

to simplify equations we write $z_c = w_c{}^\top x$

$$\hat{y}_c = \text{softmax}([z_1, \ldots, z_C])_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}}$$

Likelihood for multiclass classification

C classes: categorical likelihood

$\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{I}(y=c)}$ using softmax to enforce sum-to-one constraint

$$\hat{y}_c = \text{softmax}([z_1, \ldots, z_C])_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}} \quad \text{where} \quad z_c = {w_c}^\top x$$

substituting softmax in Categorical likelihood:

likelihood $L(\{w_c\}) = \prod_{n=1}^{N} \prod_{c=1}^{C} \text{softmax}([z_1^{(n)}, \ldots, z_C^{(n)}])_c^{\mathbb{I}(y^{(n)}=c)}$

$$= \prod_{n=1}^{N} \prod_{c=1}^{C} \left(\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}} \right)^{\mathbb{I}(y^{(n)}=c)}$$

One-hot encoding

likelihood $L(\{w_c\}) = \prod_{n=1}^{N} \prod_{c=1}^{C} \left(\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}} \right)^{\mathbb{I}(y^{(n)}=c)}$

log-likelihood $\ell(\{w_c\}) = \sum_{n=1}^{N} \sum_{c=1}^{C} \mathbb{I}(y^{(n)} = c)(z_c^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}})$

one-hot encoding for labels $y^{(n)} \rightarrow [\mathbb{I}(y^{(n)} = 1), \ldots, \mathbb{I}(y^{(n)} = C)]$

convert that feature into C binary features

Example: $y^{(n)} \in \{1, 2, 3\} \Rightarrow y^{(n)} \in \{[1, 0, 0], [0, 1, 0], [0, 0, 1]\}$

side note

we can also use this encoding for categorical **features**

$$x_d^{(n)} \rightarrow [\mathbb{I}(x_d^{(n)} = 1), \ldots, \mathbb{I}(x_d^{(n)} = C)]$$

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

one-hot example from here

One-hot encoding

likelihood $L(\{w_c\}) = \prod_{n=1}^{N} \prod_{c=1}^{C} \left(\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}} \right)^{\mathbb{I}(y^{(n)}=c)}$

log-likelihood $\ell(\{w_c\}) = \sum_{n=1}^{N} \sum_{c=1}^{C} \mathbb{I}(y^{(n)} = c)(z_c^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}})$

one-hot encoding for labels $y^{(n)} \rightarrow [\mathbb{I}(y^{(n)} = 1), \ldots, \mathbb{I}(y^{(n)} = C)]$

$z^{(n)} = [z_1^{(n)}, z_2^{(n)}, \ldots z_C^{(n)}], \quad z_c^{(n)} = w_c^\top x^{(n)}$

using this encoding from now on

log-likelihood $\ell(\{w_c\}) = \sum_{n=1}^{N} \left({y^{(n)}}^\top z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}} \right)$

Optimization

given the training data $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_n$

find the best model parameters $\{w_c\}_c$

by minimizing the cost (maximizing the likelihood of $\mathcal{D}$)

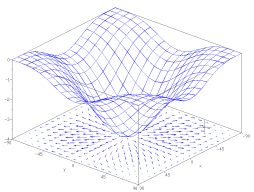
$$J(\{w_c\}) = -\sum_{n=1}^{N} (y^{(n)\top} z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}}) \text{ where } \quad z_c = w_c{}^\top x$$

need to use gradient descent (for now calculate the gradient)

$$\nabla J(w) = [\frac{\partial}{\partial w_{1,1}} J, \ldots \frac{\partial}{\partial w_{1,D}} J, \ldots, \frac{\partial}{\partial w_{C,D}} J]^\top$$

length $C \times D$

Gradient



need to use gradient descent (for now calculate the gradient)

$$J(\{w_c\}) = -\sum_{n=1}^{N}(y^{(n)\top} z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}}) \quad \text{where } z_c = w_c{}^\top x$$

using chain rule $\frac{\partial}{\partial w_{c,d}} J = \sum_{n=1}^{N} \frac{\partial J}{\partial z_c^{(n)}} \frac{\partial z_c^{(n)}}{\partial w_{c,d}} = \sum_n (\hat{y}_c^{(n)} - y_c^{(n)}) x_d^{(n)}$

this looks familiar!

$x_d^{(n)}$

$-y_c^{(n)} + \frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z^{(n)}{}_{c'}}}$ so the derivative of log-sum-exp is softmax

$\hat{y}_c^{(n)}$