CSC 2515: Introduction to Machine Learning

Lecture 3: Regression and Classification with Linear Models

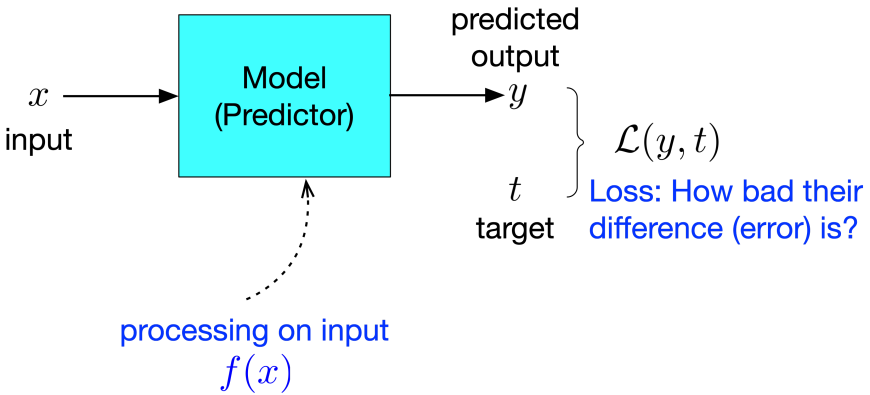
Amir-massoud Farahmand[1]

University of Toronto and Vector Institute

[1] Credit for slides goes to many members of the ML Group at the U of T, and beyond, including (recent past): Roger Grosse, Murat Erdogdu, Richard Zemel, Juan Felipe Carrasquilla, Emad Andrews, and myself.

Table of Contents

Modular Approach to ML Algorithm Design



Modular Approach to ML Algorithm Design

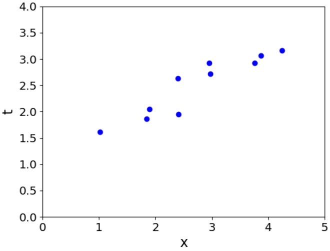
- So far, we have talked about *procedures* for learning.
  - KNN and decision trees.
- For the remainder of this course, we will take a more modular approach:
  - choose a model describing the relationships between variables of interest
  - define a loss function quantifying how bad the fit to the data is
  - (possibly) choose a regularizer saying how much we prefer different candidate models (or explanations of data), before (prior to) seeing the data
  - fit the model that minimizes the loss function and satisfy the constraint/penalty imposed by the regularizer, possibly using an optimization algorithm
- Mixing and matching these modular components gives us a lot of new ML methods.

Skills to Learn

Understanding

- The modular approach to ML
- The role of a model
  - Linear models
  - How can we make them more powerful and flexible?
- Regularization
- Loss function
  - The relation of loss function and the decision problem we want to solve
  - Some loss functions suitable for regression and classification
  - Maximum Likelihood interpretation
- Optimization using Gradient Descent and Stochastic Gradient Descent

The Supervised Learning Setup



Recall that in supervised learning:

- There is a target $t \in \mathcal{T}$ (also called response, outcome, output, class)
- There are features $\mathbf{x} \in \mathcal{X}$ (also called inputs or covariates)
- The goal is to learn a function $f : \mathcal{X} \rightarrow \mathcal{T}$ such that

$$t \approx y = f(x),$$

based on given data $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)}) \text{ for } i = 1, 2, ..., N\}$.

Regression with Linear Models

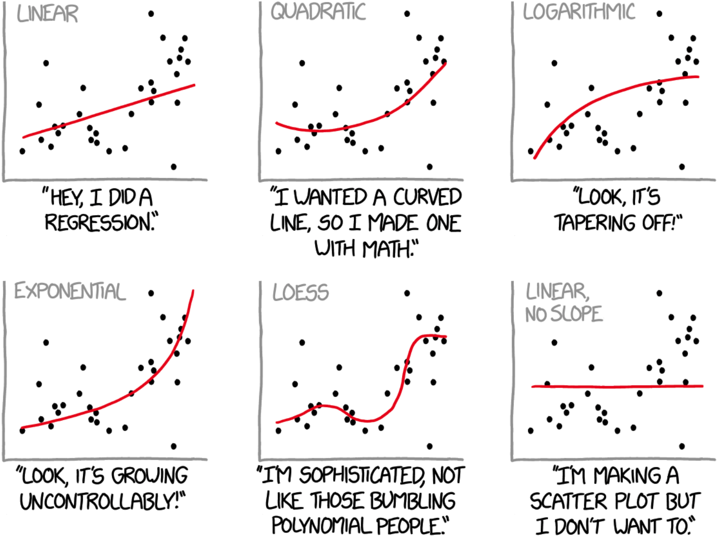


Image credit: xkcd (cropped)

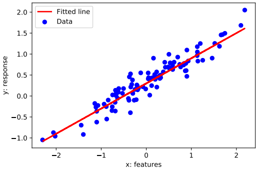
Linear Regression – Model

- Model: In linear regression, we use linear functions of the inputs $\mathbf{x} = (x_1, \ldots, x_D)$ to make predictions y of the target value t:

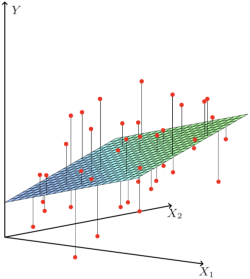
$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

  - y is the prediction
  - $\mathbf{w}$ is the weights
  - b is the bias (or intercept) (do not confuse with the bias-variance tradeoff in the next lecture)
- $\mathbf{w}$ and b together are the parameters
- We hope that our prediction is close to the target: $y \approx t$.

What is Linear? 1 Feature vs. D Features



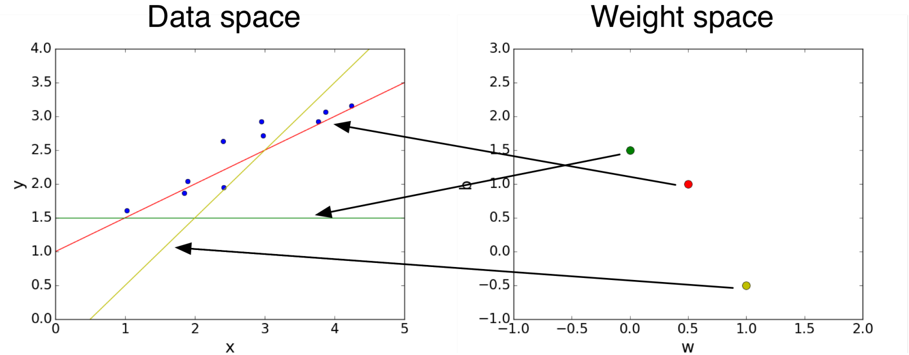
- If we have only 1 feature:
  $y = wx + b$ where $w, x, b \in \mathbb{R}$.
- y is linear in x.



- If we have D features:
  $y = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D$,
  $b \in \mathbb{R}$
- y is linear in $\mathbf{x}$.

Relation between the prediction y and inputs $\mathbf{x}$ is linear in both cases.

Weight Space vs. Data Space



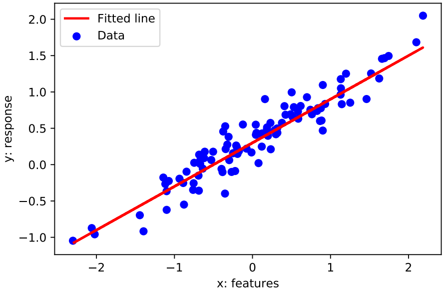
Recall that

$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

Linear Regression

We have a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{N}$ where,

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_D^{(i)})^\top \in \mathbb{R}^D$ are the inputs, e.g., age, height,
- $t^{(i)} \in \mathbb{R}$ is the target or response, e.g., income,
- predict $t^{(i)}$ with a linear function of $\mathbf{x}^{(i)}$:



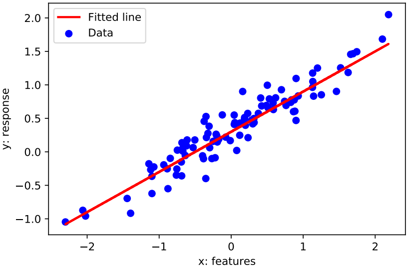
- $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
- Find the "best" line $(\mathbf{w}, b)$.
- Q: How should we define the best line?

Linear Regression – Loss Function

- How to quantify the quality of the fit to data?
- A loss function $\mathcal{L}(y, t)$ defines how bad it is if, for some input $\mathbf{x}$, the algorithm predicts y, but the target is actually t.
- Squared error loss function:

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

- $y - t$ is the residual, and we want to make its magnitude small
- The $\frac{1}{2}$ factor is just to make the calculations convenient.



Linear Regression – Loss Function

- Cost function: loss function averaged over all training examples

$$\begin{aligned}\mathcal{J}(\mathbf{w}, b) &= \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y^{(i)}, t^{(i)}) \\ &= \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2 \\ &= \frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{w}^\top\mathbf{x}^{(i)} + b - t^{(i)}\right)^2\end{aligned}$$

- To find the best fit, we find a model (parameterized by its weights $\mathbf{w}$ and b) that minimizes the cost:

$$\underset{(\mathbf{w},b)}{\text{minimize}}\mathcal{J}(\mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y^{(i)}, t^{(i)}).$$

- The terminology is not universal. Some might call "loss" pointwise loss and the "cost function" the empirical loss or average loss.

Vector Notation

- We can organize all the training examples into a design matrix $\mathbf{X}$ with one row per training example, and all the targets into the target vector $\mathbf{t}$.

one feature across all training examples

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{pmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 8 \end{pmatrix}$$

one training example (vector)

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^\top \mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

Vectorization

- Computing the squared error cost across the whole dataset:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

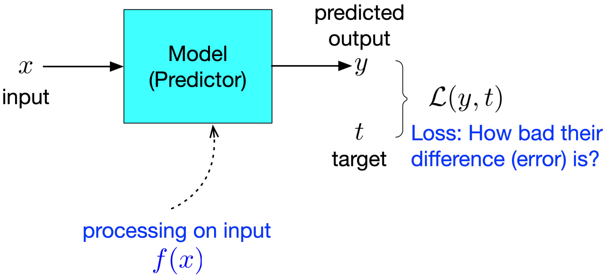
$$\mathcal{J} = \frac{1}{2N}\|\mathbf{y} - \mathbf{t}\|^2$$

- Note that sometimes we may use $\mathcal{J} = \frac{1}{2}\|\mathbf{y} - \mathbf{t}\|^2$, without $\frac{1}{N}$ normalizer. That would correspond to the sum of losses, and not the average loss. That does not matter as the minimizer does not depend on N.
- We can also add a column of 1s to the design matrix, combine the bias and the weights, and conveniently write

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ 1 & [\mathbf{x}^{(2)}]^\top \\ 1 & \vdots \end{bmatrix} \in \mathbb{R}^{N \times D+1} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

Then, our predictions reduce to $\mathbf{y} = \mathbf{X}\mathbf{w}$.

Solving the Minimization Problem



- We defined a model (linear).
- We defined a loss and the cost function to be minimized.
- Q: How should we solve this minimization problem?

Solving the Minimization Problem

- Recall from your calculus class: minimum of a differentiable function (if it exists) occurs at a critical point, i.e., point where the derivative is zero.
- Multivariate generalization: set the partial derivatives to zero (or equivalently the gradient).
- We would like to find a point where the gradient is (close to) zero. How can we do it?
- Sometimes it is possible to directly find the parameters that make the gradient zero in a closed-form. We call this the direct solution.
- We may also use optimization techniques that iteratively get us closer to the solution. We will get back to this soon.

Direct Solution

- Partial derivatives: derivatives of a multivariate function with respect to (w.r.t.) one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- To compute, take the single variable derivatives, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y with respect to weight w_j and bias b:

$$\begin{aligned} \frac{\partial y}{\partial w_j} &= \frac{\partial}{\partial w_j} \left[\sum_{j'} w_{j'} x_{j'} + b \right] \\ &= x_j \\ \frac{\partial y}{\partial b} &= \frac{\partial}{\partial b} \left[\sum_{j'} w_{j'} x_{j'} + b \right] \\ &= 1 \end{aligned}$$

Direct Solution

- The derivative of loss: We apply the chain rule: first we take the derivative of the loss $\mathcal{L}$ w.r.t. output y of the model, and then the derivative of the output y w.r.t. a parameter of the model such as w_j or b:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\frac{\partial y}{\partial w_j} \\
&= \frac{\mathrm{d}}{\mathrm{d}y}\left[\frac{1}{2}(y-t)^2\right]\cdot x_j \\
&= (y-t)x_j \\
\frac{\partial \mathcal{L}}{\partial b} &= y-t
\end{aligned}$$

- Cost derivatives (average over data points):

$$\frac{\partial \mathcal{J}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)}-t^{(i)})\,x_j^{(i)}$$

$$\frac{\partial \mathcal{J}}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)}-t^{(i)})$$

Direct Solution

- Recall that the output y is a function of the parameters as $y = \mathbf{w}^\top \mathbf{x}$.
- The minimum of the cost function must occur at a point where the partial derivatives are zero, i.e.,

$$\nabla_{\mathbf{w}} \mathcal{J} = 0 \Leftrightarrow \frac{\partial \mathcal{J}}{\partial w_j} = 0 \quad (\forall j), \qquad \frac{\partial \mathcal{J}}{\partial b} = 0.$$

- If $\partial \mathcal{J} / \partial w_j \neq 0$, you could reduce the cost by changing w_j.

Direct Solution

If we follow this recipe, we get that we have to set the gradient of $\mathcal{J} = \frac{1}{2N}\|\mathbf{y} - \mathbf{t}\|^2$, with $\mathbf{y} = \mathbf{X}\mathbf{w}$ (bias absorbed in $\mathbf{X}$) equal to zero. We have

$$\mathcal{J} = \frac{1}{2N}(\mathbf{X}\mathbf{w} - \mathbf{t})^\top(\mathbf{X}\mathbf{w} - \mathbf{t}),$$

so

$$\nabla_{\mathbf{w}}\mathcal{J} = \frac{1}{N}\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{t}) = 0 \Rightarrow (\mathbf{X}^\top\mathbf{X})\mathbf{w} = \mathbf{X}^\top\mathbf{t}.$$

This is a linear system of equations.

- Q: What are the dimensions of each component?

Assuming that $\mathbf{X}^\top\mathbf{X}$ is invertible, the optimal weights are

$$\mathbf{w}^{\mathrm{LS}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{t}.$$

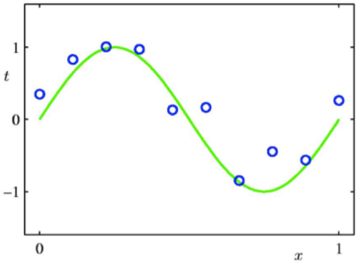
This solution is also called Ordinary Least Squares (OLS) solution.

At an arbitrary point $\mathbf{x}$, our prediction is $y = \mathbf{w}^{\mathrm{LS}\top}\mathbf{x}$.

- Q: What happens if $\mathbf{X}^\top\mathbf{X}$ is not invertible?

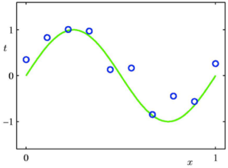
Basis Expansion (Feature Mapping)

- The relation between the input and output may not be linear.



- We can still use linear regression by mapping the input feature to another space using basis expansion (or feature mapping) $\boldsymbol{\psi}(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^d$ and treat the mapped feature (in $\mathbb{R}^d$) as the input of a linear regression procedure.
- Let us see how it works when $\mathbf{x} \in \mathbb{R}$ and we use polynomial feature mapping.

Polynomial Feature Mapping



Fit the data using a degree-M polynomial function of the form:

$$y = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{i=0}^{M} w_i x^i$$

- The feature mapping is $\boldsymbol{\psi}(x) = [1, x, x^2, ..., x^M]^\top$.
- We can still use the linear regression framework with least squares loss to find $\mathbf{w}$ since $y = \boldsymbol{\psi}(x)^\top \mathbf{w}$ is linear in $w_0, w_1, ...$.
- In general, $\boldsymbol{\psi}$ can be any function. Another example: Fourier map $\boldsymbol{\psi} =$ $[1, \sin(2\pi x), \cos(2\pi x), \sin(4\pi x), \cos(4\pi x), \sin(6\pi x), \cos(6\pi x), \cdots]^\top$.
- Q: Other examples?

Polynomial Feature Mapping with $M = 0$

$$y = w_0$$

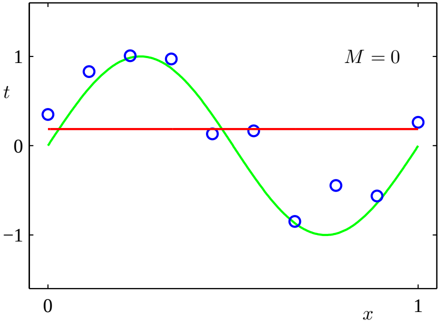


Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Polynomial Feature Mapping with $M = 1$

$$y = w_0 + w_1 x$$

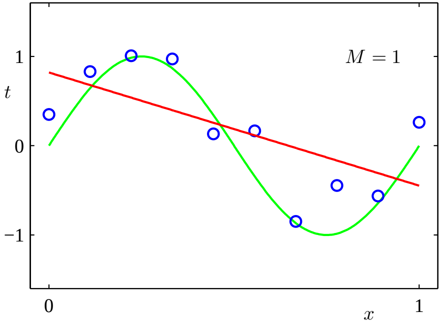


Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Polynomial Feature Mapping with $M = 3$

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

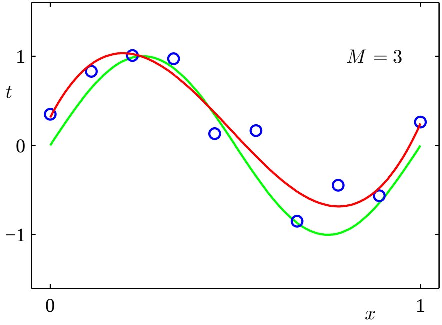


Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Polynomial Feature Mapping with $M = 9$

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$$

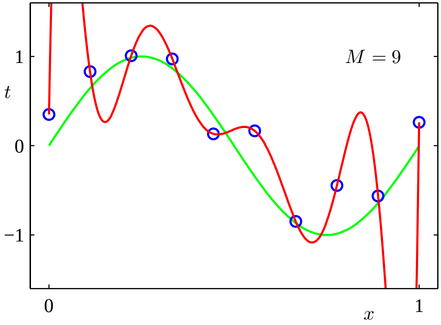


Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Model Complexity and Regularization

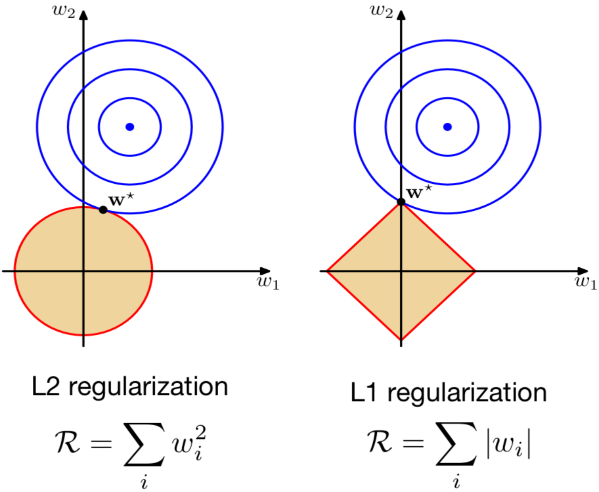
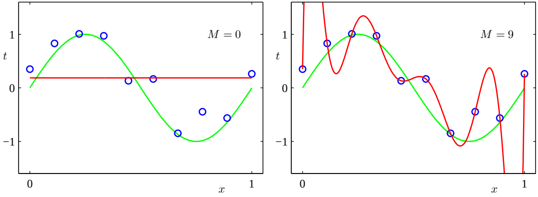


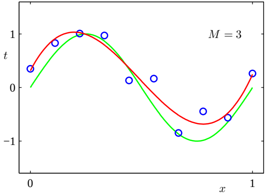
Image credit: Pattern Recognition and Machine Learning (Chapter 3), Christopher Bishop.

Model Complexity and Generalization

Underfitting (M=0): model is too simple — does not fit the data.
Overfitting (M=9): model is too complex — fits perfectly.

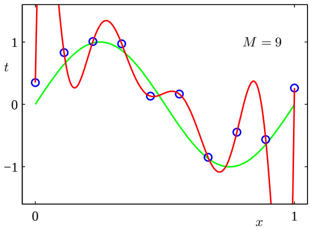


Good model (M=3): Achieves small test error (generalizes well).



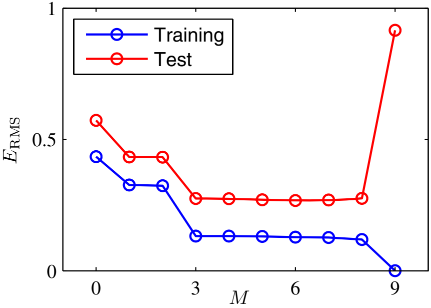
Model Complexity and Generalization

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43



- As M increases, the magnitude of coefficients gets larger.
- For $M = 9$, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

Model Complexity and Generalization

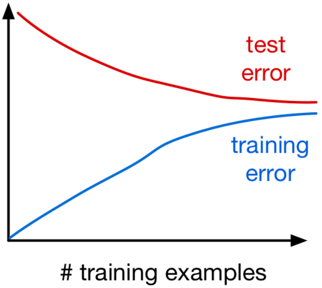


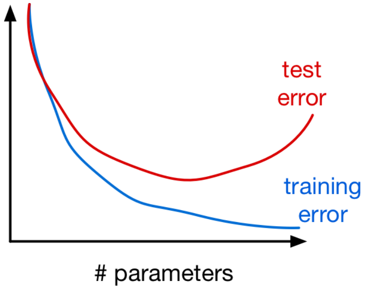
As the degree M of the polynomial increases

- the training errors decreases;
- the test error, however, initially decreases, but then increases.

Model Complexity and Generalization

- Training and test error as a function of # training examples and # parameters:





Regularization for Controlling the Model Complexity

- The degree of the polynomial M controls the complexity of the model.
- The value of M is a hyperparameter for polynomial expansion, just like K in KNN or the depth of a tree in a decision tree. We can tune it using a validation set.
- Restricting the number of parameters of a model (M here) is a crude approach to control the complexity of the model.
- A better solution: keep the number of parameters of the model large, but enforce "simpler" solutions within the same space of parameters.
- This is done through regularization or penalization.
  - Regularizer (or penalty): a function that quantifies how much we prefer one hypothesis vs. another, prior to seeing the data.
- Q: How?!

ℓ_2 (or L^2) Regularization

- We can encourage the weights to be small by choosing the ℓ_2 (or L^2) of the weights as our regularizer or penalty:

$$\mathcal{R}(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\sum_j w_j^2.$$

  - Note: To be precise, we are regularizing the *squared* ℓ_2 norm.

- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights:

$$\mathcal{J}_{\mathrm{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda\mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2}\sum_j w_j^2.$$

ℓ_2 (or L^2) Regularization

- The regularized cost function:

$$\mathcal{J}_{\text{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda \mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2} \sum_j w_j^2.$$

- The basic idea is that "simpler" functions have weights $\mathbf{w}$ with smaller ℓ_2-norm and we prefer them to functions with larger ℓ_2-norms.
  - Intuition: Large weights makes the function f have more abrupt changes as a function of the input $\mathbf{x}$; it will be less smooth.
- If you fit training data poorly, $\mathcal{J}$ is large. If the fitted weights have high values, $\mathcal{R}$ is large.
- Large λ penalizes weight values more.
- Here, λ is a hyperparameter that we can tune with a validation set.

ℓ_2 Regularized Least Squares: Ridge Regression

For the least squares problem, we have $\mathcal{J}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$.

- When $\lambda > 0$ (with regularization), regularized cost gives

$$\mathbf{w}_{\lambda}^{\text{Ridge}} = \operatorname*{argmin}_{\mathbf{w}} \mathcal{J}_{\text{reg}}(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$
$$= (\mathbf{X}^T\mathbf{X} + \lambda N\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}.$$

- The case of $\lambda = 0$ (no regularization) reduces to the least squares solution!
- Q: What happens when $\lambda \to \infty$?
- Note that it is also common to formulate this problem as $\operatorname{argmin}_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$ in which case the solution is $\mathbf{w}_{\lambda}^{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{t}$.

Lasso and the ℓ_1 Regularization

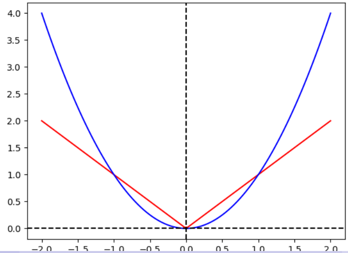
- The ℓ_1 norm, or sum of absolute values, is another regularizer:

$$\mathcal{R}(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_j |w_j|.$$

- The Lasso (Least Absolute Shrinkage and Selection Operator) is

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \lambda \|\mathbf{w}\|_1 .$$

- It can be shown that Lasso encourages weights to be exactly zero.
  - Q: When is this helpful?



Ridge vs. Lasso – Geometric Viewpoint

- We presented regularization as a penalty on the weights, in which we solve

$$\min_{\mathbf{w}} \mathcal{J}(\mathbf{w}) + \lambda \mathcal{R}(\mathbf{w})$$

- We can also write an equivalent form as a constraint optimization:

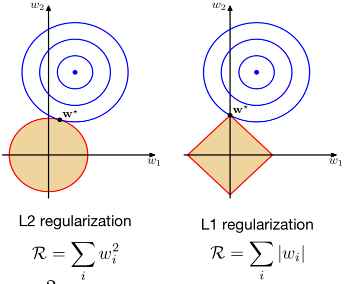
$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{w}} \mathcal{J}(\mathbf{w}) \\ &\quad \text{s.t. } \mathcal{R}(\mathbf{w}) \leq \mu, \end{aligned}$$

  for a corresponding value of μ.

- The Ridge regression and the Lasso can then be written as

$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 \\ &\quad \text{s.t. } \|\mathbf{w}\|_p \leq \mu \qquad (\text{Lasso: } p = 1; \text{Ridge: } p = 2) \end{aligned}$$

Ridge vs. Lasso – Geometric Viewpoint

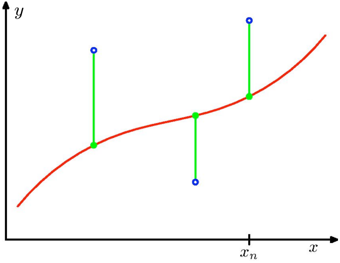


- The set $\{\mathbf{w} : \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 \leq \varepsilon\}$ defines ellipsoids of ε cost in the weights space.
- The set $\{\mathbf{w} : \|\mathbf{w}\|_p \leq \mu\}$ defines the constraint on weights defined by the regularizer.
- The solution would be the smallest ε for which these two sets intersects.
- For $p = 1$, the diamond-shaped constraint set has corners. When the intersection happens at a corner, some of the weights are zero.
- For $p = 2$, the disk-shaped constraint set does not have corners. It does not induce any zero weights.

Probabilistic Interpretation of the Squared Error

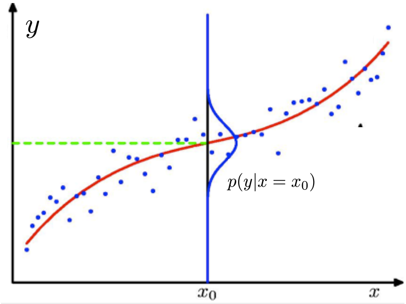
For the least squares: we minimize the sum of the squares of the errors between the predictions for each data point $\mathbf{x}^{(i)}$ and the corresponding target values $t^{(i)}$, i.e.,

$$\underset{(\mathbf{w},\mathbf{w}_0)}{\text{minimize}} \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)})^2$$



- $t \approx \mathbf{x}^\top \mathbf{w} + b$, $(\mathbf{w}, b) \in \mathbb{R}^D \times \mathbb{R}$
- We measure the quality of the fit using the squared error loss. Why?
- Even though the squared error loss is intuitive, we did not justify it.
- We provide a probabilistic perspective here.
- There are other justifications too; we get to them in the Bias-Variance decomposition lecture.

Probabilistic Interpretation of the Squared Error



- Suppose that our model arose from a statistical model ($b=0$ for simplicity):

$$y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + \epsilon^{(i)},$$

where $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ is independent of the input $\mathbf{x}^{(i)}$.

- Thus, $y^{(i)}|\mathbf{x}^{(i)} \sim p(y|\mathbf{x}^{(i)}, \mathbf{w}) = \mathcal{N}(\mathbf{w}^\top \mathbf{x}^{(i)}, \sigma^2)$.

Probabilistic Interpretation of the Squared Error: Maximum Likelihood Estimation

- Suppose that the input data $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$ are given and the outputs are independently drawn from

$$t^{(i)} \sim p(y|\mathbf{x}^{(i)}, \mathbf{w}),$$

  with an unknown parameter $\mathbf{w}$. So the dataset is $\mathcal{D} = \{(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(N)}, t^{(N)})\}$.
- The likelihood function is $\Pr(\mathcal{D}|\mathbf{w})$.
- The maximum likelihood estimation (MLE) is based on the "principle" suggesting that we have to find a parameter $\hat{\mathbf{w}}$ that maximizes the likelihood, i.e.,

$$\hat{\mathbf{w}} \leftarrow \operatorname*{argmax}_{\mathbf{w}} \Pr(\mathcal{D}|\mathbf{w}).$$

Maximum likelihood estimation: after observing the data samples $(\mathbf{x}^{(i)}, t^{(i)})$ for $i = 1, 2, ..., N$, we should choose $\mathbf{w}$ that maximizes the likelihood.

Probabilistic Interpretation of the Squared Error: Maximum Likelihood Estimation

- For independent samples, the likelihood function of samples $\mathcal{D}$ is the product of their likelihoods

$$p\left(t^{(1)}, t^{(2)}, \ldots, t^{(N)} | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}, \mathbf{w}\right) = \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}, \mathbf{w}) = L(\mathbf{w}).$$

- Product of N terms is not easy to minimize.
- Taking log reduces it to a sum. Two objectives are equivalent since log is strictly increasing.
- Maximizing the likelihood is equivalent to minimizing the negative log-likelihood:

$$\ell(\mathbf{w}) = -\log L(\mathbf{w}) = -\log \prod_{i=1}^{N} p(t^{(i)} | \mathbf{x}^{(i)}; \mathbf{w}) = -\sum_{i=1}^{n} \log p(t^{(i)} | \mathbf{x}^{(i)}; \mathbf{w})$$

Probabilistic Interpretation of the Squared Error: Maximum Likelihood Estimation

Maximum Likelihood Estimator (MLE)

After observing $z^{(i)} = (\mathbf{x}^{(i)}, t^{(i)})$ for $i = 1, ..., N$ independent and identically distributed (i.i.d.) samples from $p(z, \mathbf{w})$, MLE is

$$\mathbf{w}^{\text{MLE}} = \underset{\mathbf{w}}{\operatorname{argmin}} \quad l(\mathbf{w}) = -\sum_{i=1}^{N} \log p(t^{(i)} | \mathbf{x}^{(i)}; \mathbf{w}).$$

Probabilistic Interpretation of the Squared Error: From MLE to Squared Error

- Suppose that our model arose from a statistical model:

$$y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + \epsilon^{(i)}$$

  where $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ is independent of anything else.
- $p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)})^2\right\}$
- $\log p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = -\frac{1}{2\sigma^2}(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)})^2 - \log(\sqrt{2\pi\sigma^2})$
- The MLE solution is

$$\mathbf{w}^{\text{MLE}} = \underset{\mathbf{w}}{\operatorname{argmin}}\ \mathcal{L}(\mathbf{w}) = \frac{1}{2\sigma^2} \sum_{i=1}^{N} (t^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)})^2 + C.$$

- As C and σ do not depend on $\mathbf{w}$, they do not contribute to the minimization.

$\mathbf{w}^{\text{MLE}} = \mathbf{w}^{\text{LS}}$ when we work with Gaussian densities.

Probabilistic Interpretation of the Squared Error: From MLE to Squared Error

- Suppose that our model arose from a statistical model:

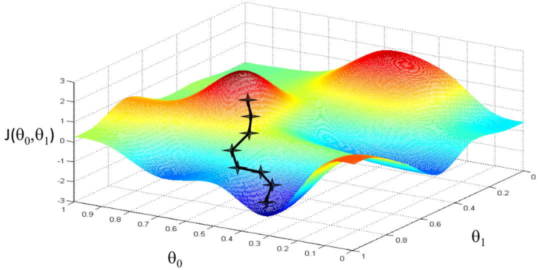
$$y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + \epsilon^{(i)}$$

where $\epsilon^{(i)}$ comes from the Laplace distribution, that is, the distribution of $\epsilon^{(i)}$ has density

$$\frac{1}{2b} \exp\left(\frac{|y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)}|}{2b} \right).$$

- Q: What is the loss in the MLE?
  - Choice 1: $\frac{1}{N} \sum_{i=1}^{N} |t^{(i)} - w^\top x^{(i)}|^{1/2}$
  - Choice 2: $\frac{1}{N} \sum_{i=1}^{N} (t^{(i)} - w^\top x^{(i)})$
  - Choice 3: $\frac{1}{N} \sum_{i=1}^{N} |t^{(i)} - w^\top x^{(i)}|$
  - Choice 4: $\frac{1}{N} \left| \sum_{i=1}^{N} (t^{(i)} - w^\top x^{(i)}) \right|$
- Q: Can you think of an application area with non-Gaussian probabilistic model?

Gradient Descent for Optimization



Gradient Descent

- Now let's see a second way to minimize the cost function which is more broadly applicable: gradient descent.
- Gradient descent is an iterative algorithm, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g., all zeros) and repeatedly adjust them in the direction of steepest descent.

Gradient Descent

- Observe:
  - if $\partial \mathcal{J} / \partial w_j > 0$, then increasing w_j increases $\mathcal{J}$.
  - if $\partial \mathcal{J} / \partial w_j < 0$, then increasing w_j decreases $\mathcal{J}$.
- The following update decreases the cost function:

$$\begin{aligned} w_j &\leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j} \\ &= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, x_j^{(i)} \end{aligned}$$

- α is the learning rate or step size. The larger it is, the faster $\mathbf{w}$ changes.
  - We'll see later how to tune the learning rate, but values are typically small, e.g., 0.1 or 0.001.

Gradient Descent

- The method gets its name from the gradient:

$$\nabla_{\mathbf{w}} \mathcal{J} = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

  - This is the direction of fastest increase in $\mathcal{J}$. (Q: Why?)
- Update rule in vector form:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \\ &= \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)} \end{aligned}$$

- Hence, gradient descent updates the weights in the direction of fastest *decrease*.
- Observe that once it converges, we get a critical point: $\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = 0$.

Gradient Descent for Linear regression

- Even for linear regression, where there is a direct solution, we sometimes need to use GD.
- Why gradient descent, if we can find the optimum directly?
  - GD can be applied to a much broader set of models
  - GD can be easier to implement than direct solutions
  - For regression in high-dimensional spaces, GD is more efficient than direct solution
    - Linear regression solution: $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$
    - matrix inversion is an $\mathcal{O}(D^3)$ algorithm
    - each GD update costs $O(ND)$
    - Huge difference if $D \gg 1$

Gradient Descent under the ℓ_2 Regularization

- Recall the gradient descent update:

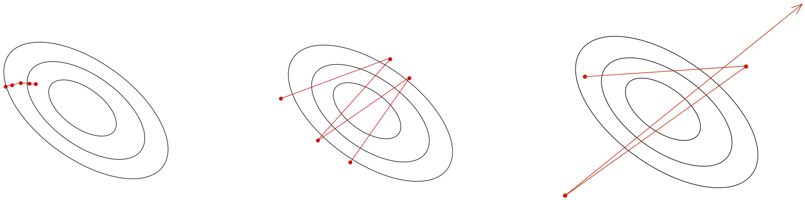
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}$$

- The gradient descent update of the regularized cost $\mathcal{J} + \lambda \mathcal{R}$ has an interesting interpretation as weight decay (for the ℓ_2 regularizer):

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \frac{\partial \mathcal{R}}{\partial \mathbf{w}} \right) \\ &= \mathbf{w} - \alpha \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \mathbf{w} \right) \\ &= (1 - \alpha\lambda)\mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \end{aligned}$$

Learning Rate (Step Size)

- In gradient descent, the learning rate α is a hyperparameter we need to tune. If we do not choose it right, the procedure may have undesirable convergence properties:



α too small: slow progress

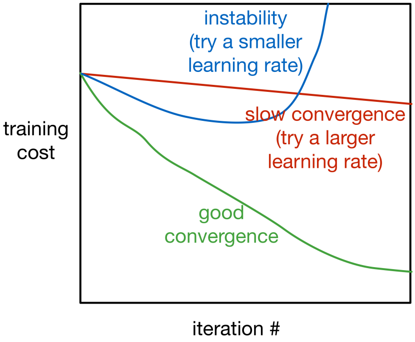
α too large: oscillations

α much too large: instability

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance, i.e., try $0.1, 0.03, 0.01, \ldots$.

Training Curves

- To diagnose optimization problems, it is useful to look at training curves: plot the training cost as a function of iteration.



Brief Matrix and Vector Calculus

- For a function $f : \mathbb{R}^p \to \mathbb{R}$, $\nabla f(z)$ denotes the gradient at z which points in the direction of the greatest rate of increase.
- $\nabla f(x) \in \mathbb{R}^p$ is a vector with $[\nabla f(x)]_i = \frac{\partial}{\partial x_i} f(x)$.
- $\nabla^2 f(x) \in \mathbb{R}^{p \times p}$ is a matrix with $[\nabla^2 f(x)]_{ij} = \frac{\partial^2}{\partial x_i \partial x_j} f(x)$
- At any minimum of a function f, we have $\nabla f(\mathbf{w}) = 0$, $\nabla^2 f(\mathbf{w}) \succeq 0$.
- Consider the problem $\underset{\mathbf{w}}{\text{minimize}}\ \ell(\mathbf{w}) = \frac{1}{2}\|y - X\mathbf{w}\|_2^2$,
- $\nabla \ell(\mathbf{w}) = X^\top (X\mathbf{w} - y) = 0 \implies \hat{\mathbf{w}} = (X^\top X)^{-1} X^\top y$ (assuming $X^\top X$ is invertible)

Vectorization

- Computing the prediction using a for loop:

```
y = b
for j in range(M):
    y += w[j] * x[j]
```

- For-loops in Python are slow, so we vectorize algorithms by expressing them in terms of vectors and matrices.

$$\mathbf{w} = (w_1, \ldots, w_D)^T \qquad \mathbf{x} = (x_1, \ldots, x_D)^T$$

$$y = \mathbf{w}^T \mathbf{x} + b$$

- This is simpler and much faster: `y = np.dot(w,x) + b`

Vectorization

Why vectorize?

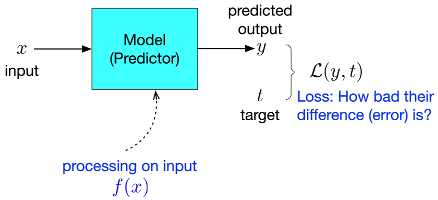
- The equations, and the code, will be simpler and more readable. Gets rid of dummy variables/indices!
- Vectorized code is much faster
  - Cut down on Python interpreter overhead
  - Use highly optimized linear algebra libraries
  - Matrix multiplication is very fast on a Graphics Processing Unit (GPU)

Classification with Linear Models

Classification Problem

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
- Examples
  - predict whether a patient has a disease, given the presence or absence of various symptoms
  - classify e-mails as spam or non-spam
  - predict whether a financial transaction is fraudulent
  - find out whether a picture is a cat or dog

Binary Linear Classification



- **classification:** predict a discrete-valued target
- **binary:** predict a binary target $t \in \{0, 1\}$
  - Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples.
  - $t \in \{0, 1\}$ or $t \in \{-1, +1\}$ is for computational convenience.
- **linear:** model is a linear function of $\mathbf{x}$, followed by a threshold r:

$$z = \mathbf{w}^T \mathbf{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

Some Simplifications

Eliminating the threshold

- We can assume without loss of generality (w.l.o.g.) that the threshold is $r = 0$:

$$\mathbf{w}^T\mathbf{x} + b \geq r \quad \Longleftrightarrow \quad \mathbf{w}^T\mathbf{x} + \underbrace{b - r}_{\triangleq w_0} \geq 0.$$

Eliminating the bias

- Add a dummy feature x_0 which always takes the value 1. The weight $w_0 = b$ is equivalent to a bias (same as linear regression)

Simplified model

$$z = \mathbf{w}^T\mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Examples

- Let us consider some simple examples to examine the properties of our model
- Forget about generalization and suppose we just want to learn Boolean functions

Examples

NOT

x_0	x_1	t
1	0	1
1	1	0

- This is our "training set"
- What conditions are needed on w_0, w_1 to classify all examples?
  - When $x_1 = 0$, need: $z = w_0 x_0 + w_1 x_1 > 0 \iff w_0 > 0$
  - When $x_1 = 1$, need: $z = w_0 x_0 + w_1 x_1 < 0 \iff w_0 + w_1 < 0$
- Example solution: $w_0 = 1, w_1 = -2$
- Is this the only solution?

Examples

AND

x_0	x_1	x_2	t
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$$z = w_0x_0 + w_1x_1 + w_2x_2$$

need: $w_0 < 0$

need: $w_0 + w_2 < 0$

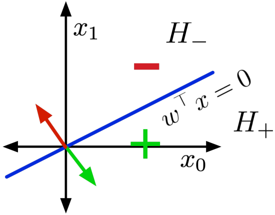
need: $w_0 + w_1 < 0$

need: $w_0 + w_1 + w_2 > 0$

Example solution: $w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$

Geometric Picture

Input Space, or Data Space for NOT example

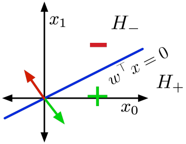


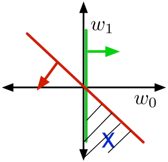
x_0	x_1	t
1	0	1
1	1	0

- This is the input space. Training examples are points in that space.
- Any weight (hypothesis) $\mathbf{w}$ defines half-spaces
  - $H_+ = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} \geq 0\}$
  - $H_- = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} < 0\}$

  in the input space.
  - The boundaries of these half-spaces pass through the origin (why?)
- The boundary is the decision boundary: $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0\}$
  - In 2-D, it is a line, but think of it as a hyperplane in general.
- If the training examples can be perfectly separated by a linear decision rule, we say that the data is linearly separable.

Weight Space





$w_0 > 0$

$w_0 + w_1 < 0$

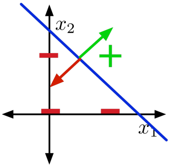
- The left figure is the input space; the right figure is the weight (hypothesis) space.
- To correctly classify each training example $\mathbf{x}$, weights $\mathbf{w}$ should belong to a particular half-space in the weight space such that $\mathbf{w}^T\mathbf{x} > 0$ if $t = 1$ (and $\mathbf{w}^T\mathbf{x} < 0$ if $t = 0$).
- For NOT example:
  - $x_0 = 1, x_1 = 0, t = 1 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 > 0\}$
  - $x_0 = 1, x_1 = 1, t = 0 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 + w_1 < 0\}$
- The region satisfying all the constraints is the feasible region; if this region is nonempty, the problem is feasible, otherwise it is infeasible.

Geometric Picture

- The **AND** example requires three dimensions, including the dummy one.
- To visualize data space and weight space for a 3-D example, we can look at a 2-D slice.
- The visualizations are similar.
  - Feasible set will always have a corner at the origin.

Visualizations of the **AND** example

Data Space

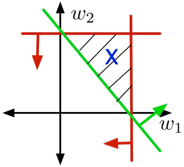


- Slice for $x_0 = 1$
- example sol: $w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$
- decision boundary:

$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$

$\implies -1.5 + x_1 + x_2 = 0$

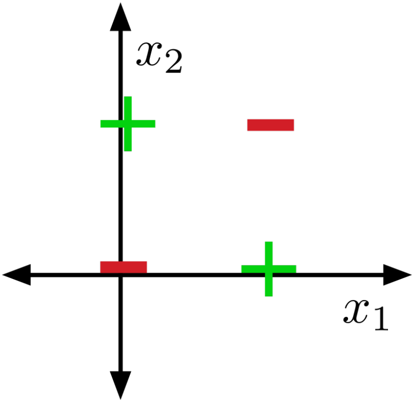
Weight Space



- Slice for $w_0 = -1.5$ for the constraints
- $w_0 < 0$
- $w_0 + w_2 < 0$
- $w_0 + w_1 < 0$
- $w_0 + w_1 + w_2 > 0$

Geometric Picture

Some datasets are not linearly separable, e.g. **XOR**



Finding the Weight Vector

- **Recall: binary linear classifiers.** Targets $t \in \{0, 1\}$

$$z = \mathbf{w}^T \mathbf{x} + b$$
$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- How can we find good values for $\mathbf{w}, b$?
- If training set is separable, we can solve for $\mathbf{w}, b$ using Linear Programming (Q: How?).
- If it is not separable, the problem is harder
  - data is almost never separable in real life.

Loss Functions for Classification

- Define loss function, then try to minimize the resulting cost function
  - Recall: cost is loss averaged (or summed) over the training set
- What loss function is suitable for classification?
- Seemingly obvious loss function: 0-1 loss

$$
\begin{aligned}
\mathcal{L}_{0-1}(y,t) &= \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases} \\
&= \mathbb{I}\{y \neq t\}
\end{aligned}
$$

Attempt 1: 0-1 Loss

- Usually, the cost $\mathcal{J}$ is the averaged loss over training examples; for 0-1 loss, this is the misclassification rate/error:

$$\mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{0-1}(y^{(i)}, t^{(i)})$$
$$= \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\{y^{(i)} \neq t^{(i)}\}.$$

Attempt 1: 0-1 Loss

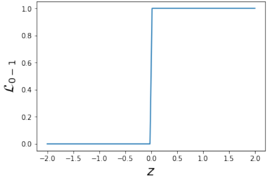
- Challenge: How to optimize?
- In general, a hard problem (can be NP-hard)
- This is due to the step function (0-1 loss) not being nice (continuous/smooth/convex etc)

Attempt 1: 0-1 Loss

- Minimum of a function will be at its critical points.
- Let's try to find the critical point of 0-1 loss.
- Consider $\mathcal{L}_{0-1}(y, t = 0)$. Recall that $y = y(\mathbf{w}) = \mathbb{I}\{z(w) \geq 0\}$ with $z = \mathbf{w}^T x$. By the chain rule:

$$\frac{\partial \mathcal{L}_{0-1}(y, 0)}{\partial w_j} = \frac{\partial \mathcal{L}_{0-1}}{\partial z} \frac{\partial z}{\partial w_j}$$

- But $\partial \mathcal{L}_{0-1} / \partial z$ is zero everywhere it is defined!



  - $\partial \mathcal{L}_{0-1} / \partial w_j = 0$ means that changing the weights by a very small amount has no effect on the loss.
  - Almost any point has 0 gradient!

Attempt 2: Linear Regression

- Sometimes we can replace the loss function we care about with another that is easier to optimize. This is known as relaxation with a smooth surrogate loss function.
- A problem with $\mathcal{L}_{0-1}$ is that it is defined in terms of final prediction (that is, after thresholding), which inherently involves a discontinuity
- Instead, define loss in terms of value of $\mathbf{w}^T\mathbf{x} + b$ (that is, before thresholding) directly
  - Redo notation for convenience: $z = \mathbf{w}^T\mathbf{x} + b$

Attempt 2: Linear Regression

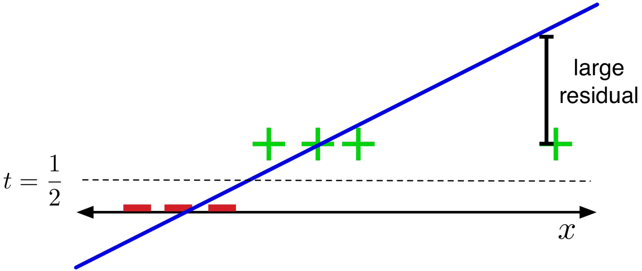
- We already know how to fit a linear regression model using the squared error loss. Can we use the same squared error loss instead?

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$\mathcal{L}_{\text{SE}}(z,t) = \frac{1}{2}(z-t)^2$$

- Doesn't matter that the targets are actually binary. Treat them as continuous values.
- For this loss function, it makes sense to make final predictions by thresholding z at $\frac{1}{2}$ (Q: Why?)

Attempt 2: Linear Regression

The problem:



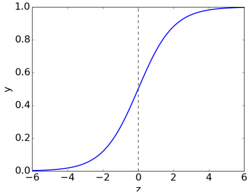
- The loss function penalizes you when you make correct predictions with high confidence!
- If $t = 1$, the loss is larger when $z = 10$ than when $z = 0$.

Attempt 3: Logistic Activation Function with Squared Error

- There is no reason to predict values outside [0, 1]. Let's squash y into this interval.

- The logistic function is a kind of sigmoid, or S-shaped function:

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



- $\sigma^{-1}(y) = \log(y/(1-y))$ is called the logit.
- A linear model with a logistic nonlinearity is known as log-linear:

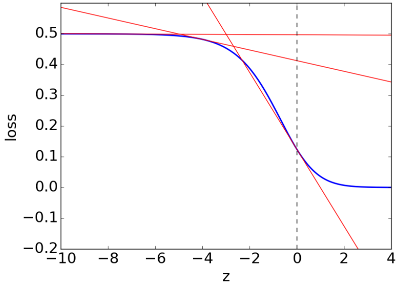
$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$y = \sigma(z)$$

$$\mathcal{L}_{\mathrm{SE}}(y, t) = \frac{1}{2}(y - t)^2.$$

- Used in this way, σ is called an activation function.

Attempt 3: Logistic Activation Function with Squared Error

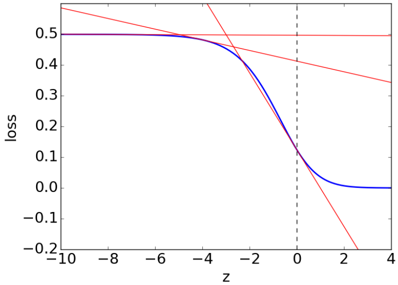


$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w_j}$$

Plot of $\mathcal{L}_{\text{SE}}$ as a function of z with $t = 1$.

- When $z \gg 0$, the prediction $\sigma(z) = \frac{1}{1+e^{-z}} \approx 1$, which is the correct prediction.
- When $z \ll 0$, we have $\sigma(z) \approx 0$. This is an incorrect prediction.
- To fix it, we would like to use the gradient to update the weights.

Attempt 3: Logistic Activation Function with Squared Error



$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w_j}$$

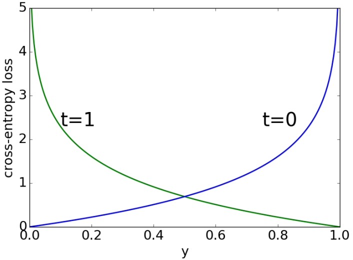
Plot of $\mathcal{L}_{\mathrm{SE}}$ as a function of z with $t = 1$.

- But $\frac{\partial \mathcal{L}}{\partial z} \approx 0$ (check!) $\implies \frac{\partial \mathcal{L}}{\partial w_j} \approx 0 \implies$ derivative w.r.t. w_j is small $\implies w_j$ is like a critical point
- If the prediction is really wrong, you should be far from a critical point and the gradient should show that.
- The gradient of this loss, however, does not indicate that.

Attempt 4: Logistic Regression

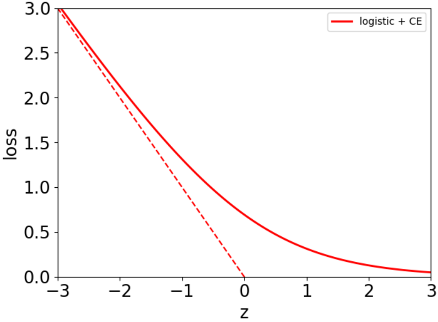
- Because $y \in [0, 1]$, we can interpret it as the estimated probability that $t = 1$.
- The pundits who were 99% confident Clinton would win were much more wrong than the ones who were only 90% confident.
- Cross-entropy loss (aka log loss) captures this intuition:

$$\mathcal{L}_{\mathrm{CE}}(y, t) = \begin{cases} -\log y & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases}$$
$$= -t \log y - (1 - t) \log(1 - y)$$



Logistic Regression

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = \sigma(z)$$
$$= \frac{1}{1 + e^{-z}}$$
$$\mathcal{L}_{\text{CE}} = -t \log y - (1 - t) \log(1 - y)$$



The plot is for target $t = 1$.

Logistic Regression

- Problem: what if $t = 1$ but you're really confident it's a negative example ($z \ll 0$)?
- If y is small enough, it may be numerically zero. This can cause very subtle and hard-to-find bugs.

$$
\begin{aligned}
y &= \sigma(z) && \Rightarrow y \approx 0 \\
\mathcal{L}_{\mathrm{CE}} &= -t \log y - (1-t)\log(1-y) && \Rightarrow \text{ computes } \log 0
\end{aligned}
$$

- Instead, we combine the activation function and the loss into a single logistic-cross-entropy function.

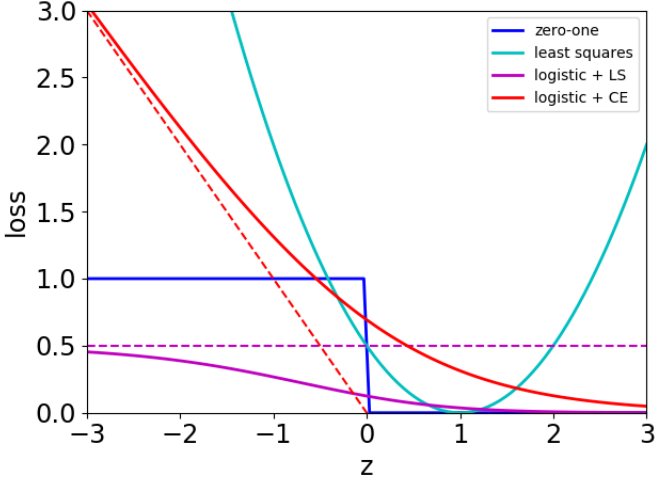
$$
\mathcal{L}_{\mathrm{LCE}}(z,t) = \mathcal{L}_{\mathrm{CE}}(\sigma(z),t) = t\log(1+e^{-z}) + (1-t)\log(1+e^{z})
$$

Q: Why do we get $\log(1+e^z)$?

- Numerically stable computation:

```
E = t * np.logaddexp(0, -z) + (1-t) * np.logaddexp(0, z)
```

Comparison of loss functions (for $t = 1$):



Probabilistic Interpretation of the Logistic Regression

- Suppose that our model arose from the statistical model

$$p(t = 1|\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^\top x}},$$

  and $p(t = 0|\mathbf{x}; \mathbf{w}) = 1 - p(t = 1|\mathbf{x}; \mathbf{w}) = \frac{e^{-\mathbf{w}^\top x}}{1+e^{-\mathbf{w}^\top x}}$.
- Consider the dataset $\mathcal{D} = \{(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(N)}, t^{(N)})\}$.
- The MLE is based on finding $\mathbf{w}$ that maximizes $\Pr(\mathcal{D}|\mathbf{w})$.
- Assume that the inputs are independent. So

$$p(t^{(1)}, \ldots, t^{(N)}|\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}, \mathbf{w}) = \prod_{i=1}^{N} p(t^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = L(\mathbf{w}).$$

- Maximizing the likelihood is equivalent to minimizing the negative log-likelihood:

$$\ell(\mathbf{w}) = -\log L(\mathbf{w}) = -\log \prod_{i=1}^{N} p(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w}) = -\sum_{i=1}^{N} \log p(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w})$$

Probabilistic Interpretation of the Logistic Regression

- So the MLE solves

$$\min_{\mathbf{w}} -\sum_{i=1}^{N} \log p(t^{(i)}|\mathbf{x}^{(i)};\mathbf{w}) = -\sum_{i:t^{(i)}=1} \log \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}^{(i)}}} - \sum_{i:t^{(i)}=0} \log \frac{e^{-\mathbf{w}^\top \mathbf{x}^{(i)}}}{1+e^{-\mathbf{w}^\top \mathbf{x}^{(i)}}}.$$

- The output of a linear model with logistic activation is $y(\mathbf{x};\mathbf{w}) = \sigma(\mathbf{x};\mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$.
- We can substitute the terms with $\log \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}^{(i)}}}$ with $\log y(\mathbf{x}^{(i)};\mathbf{w})$ and the terms with $\log \frac{e^{-\mathbf{w}^\top \mathbf{x}^{(i)}}}{1+e^{-\mathbf{w}^\top \mathbf{x}^{(i)}}}$ with $\log(1 - y(\mathbf{x}^{(i)};\mathbf{w}))$.
- The MLE would be

$$\min_{\mathbf{w}} -\sum_{i:t^{(i)}=1} \log y(\mathbf{x}^{(i)};\mathbf{w}) - \sum_{i:t^{(i)}=0} \log(1 - y(\mathbf{x}^{(i)};\mathbf{w})) =$$

$$\min_{\mathbf{w}} -\sum_{i=1}^{N} t^{(i)} \log y(\mathbf{x}^{(i)};\mathbf{w}) + (1 - t^{(i)}) \log(1 - y(\mathbf{x}^{(i)};\mathbf{w})).$$

- This is the same loss that we got for logistic regression.
- So LR is MLE with a particular probabilistic model.

Gradient Descent

- How do we minimize the cost $\mathcal{J}$ in this case? No direct solution.
  - Taking derivatives of $\mathcal{J}$ w.r.t. $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.
- We can use the gradient descent instead.

Gradient Descent for Logistic Regression

Back to logistic regression:

$$\mathcal{L}_{\text{CE}}(y,t) = -\,t\log(y) - (1-t)\log(1-y)$$
$$y = 1/(1+e^{-z}) \quad \text{and} \quad z = \mathbf{w}^T\mathbf{x} + b$$

Therefore

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_j} = \frac{\partial \mathcal{L}_{\text{CE}}}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1-t}{1-y}\right) \cdot y(1-y) \cdot x_j$$
$$= (y-t)x_j$$

Exercise: Verify this!

Gradient descent update to find the weights of logistic regression (expressed only for the w_j term):

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$
$$= w_j - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})\,x_j^{(i)}$$

Gradient Descent for Logistic Regression vs Linear Regression

Comparison of gradient descent updates:

- Linear regression (verify!):

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)}$$

- Logistic regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)}$$

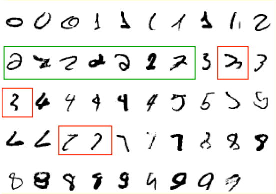
- Not a coincidence! These are both examples of generalized linear models. But we won't go in further detail.
- Notice $\frac{1}{N}$ in front of sums due to averaged losses. This is why you need smaller learning rate when we optimize the sum of losses ($\alpha' = \alpha/N$).

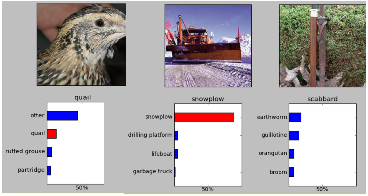
Multiclass Classification

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(> 2)-valued target
- Examples of multi-class classification
  - predict the value of a handwritten digit
  - classify e-mails as spam, travel, work, personal
  - find out whether a picture is a cat, dog, coyote, or fox

Multiclass Classification

- Classification tasks with more than two categories:





Multiclass Classification

- Targets form a discrete set $\{1, \ldots, K\}$.
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1} \in \mathbb{R}^K$$

Multiclass Classification

- There are D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- We have a K-dimensional vector $\mathbf{b}$ of biases too.
- Linear predictions:

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k \quad \text{for} \quad k = 1, 2, ..., K$$

- Vectorized:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Multiclass Classification

- Predictions are like probabilities: we want them to satisfy $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$
- A suitable activation function is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \text{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

- The inputs z_k are called the logits.
- Properties:
  - Outputs are positive and sum to 1. So they can be interpreted as probabilities.
  - If one of the z_k is much larger than the others, $\text{softmax}(\mathbf{z})_k \approx 1$. It approximately behaves like argmax.
  - **Exercise:** how does the case of $K = 2$ relate to the logistic function?
- Note: sometimes $\sigma(\mathbf{z})$ is used to denote the softmax function; in this class, it will denote the logistic function applied element-wise.

Multiclass Classification

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\begin{aligned}\mathcal{L}_{\text{CE}}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{K} t_k \log y_k \\ &= -\mathbf{t}^\top (\log \mathbf{y}),\end{aligned}$$

  where the log is applied elementwise.
- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

Multiclass Classification

- Softmax regression:

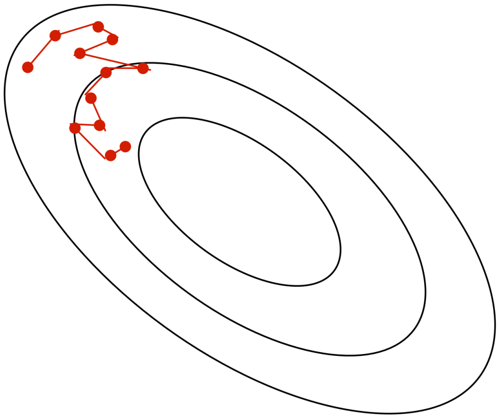
$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{y} = \text{softmax}(\mathbf{z})$$
$$\mathcal{L}_{\text{CE}} = -\mathbf{t}^\top(\log \mathbf{y})$$

- Gradient descent updates can be derived for each row of $\mathbf{W}$:

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{\text{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$
$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) \mathbf{x}^{(i)}$$

- Similar to linear/logistic regression.
- Verify the update.

Stochastic Gradient Descent



Stochastic Gradient Descent

- So far, the cost function $\mathcal{J}$ has been the average loss over the training examples:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}^{(i)} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y(\mathbf{x}^{(i)}, \mathbf{w}), t^{(i)}).$$

- By linearity,

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{w}}.$$

- Computing the gradient requires summing over *all* of the training examples. This is known as batch training.
- Batch training is impractical if you have a large dataset $N \gg 1$ (think about millions of training examples)!

Stochastic Gradient Descent

- Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example,
  1. Choose i uniformly at random
  2. $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{w}}$
- Cost of each SGD update is independent of N.
- SGD can make significant progress before even seeing all the data!
- Mathematical justification: if you sample a training example uniformly at random, the stochastic gradient is an unbiased estimate of the batch gradient:

$$\mathbb{E}\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{w}}\right] = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{w}} = \frac{\partial \mathcal{J}}{\partial \mathbf{w}}.$$

- Problems:
  - Variance in this estimate may be high
  - If we only look at one training example at a time, we can't exploit efficient vectorized operations.

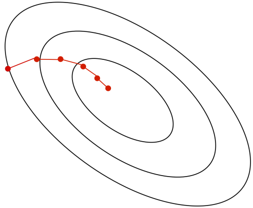
Stochastic Gradient Descent

- Compromise approach: compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \ldots, N\}$, called a mini-batch.
- Stochastic gradients computed on larger mini-batches have smaller variance.

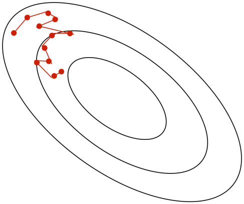
$$\mathrm{Var}\left[\frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}}\frac{\partial\mathcal{L}^{(i)}}{\partial\mathbf{w}_j}\right] = \frac{1}{|\mathcal{M}|^2}\sum_{i\in\mathcal{M}}\mathrm{Var}\left[\frac{\partial\mathcal{L}^{(i)}}{\partial\mathbf{w}_j}\right] = \frac{1}{|\mathcal{M}|}\mathrm{Var}\left[\frac{\partial\mathcal{L}^{(1)}}{\partial\mathbf{w}_j}\right]$$

  - Here we used the independence of data points in the first equality, and their having identical distribution in the second equality.
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.
  - Too large: takes more computation, i.e. takes more memory to store the activations, and longer to compute each gradient update
  - Too small: can't exploit vectorization; has high variance
  - A reasonable value might be $|\mathcal{M}| = 100$.

- Batch gradient descent moves directly downhill. SGD takes steps in a noisy direction, but moves downhill on average.



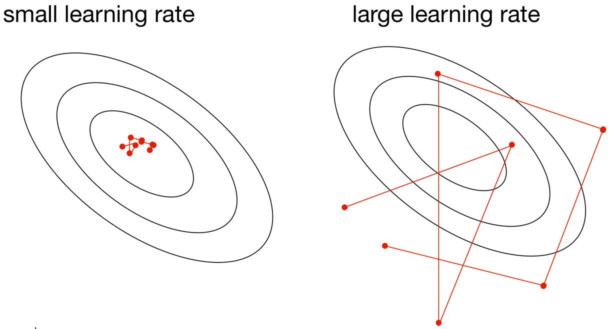
batch gradient descent



stochastic gradient descent

SGD Learning Rate

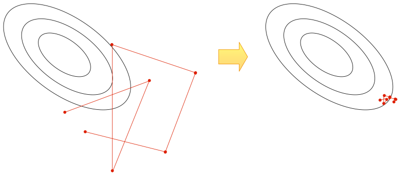
- In stochastic training, the learning rate also influences the fluctuations due to the stochasticity of the gradients.

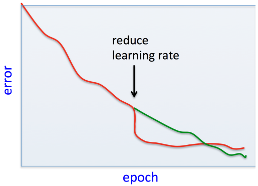


- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations

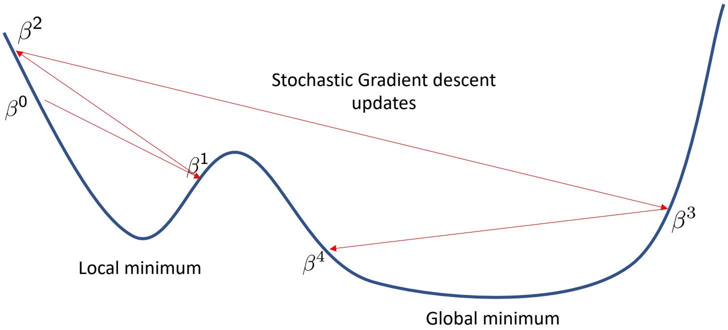
SGD Learning Rate

- Warning: by reducing the learning rate, you reduce the fluctuations, which can appear to make the loss drop suddenly. But this can come at the expense of long-run performance.





SGD and Non-convex optimization



- Stochastic methods have a chance of escaping from bad minima.
- Gradient descent with small step-size converges to first minimum it finds.

Conclusion

- A modular approach to ML
  - choose a model
  - choose a loss function suitable for the problem
  - formulate an optimization problem
  - solve the minimization problem

Conclusion

- Regression with linear models:
  - Solution method: direct solution or gradient descent
  - vectorize the algorithm, i.e., use vectors and matrices instead of summations
  - make a linear model more powerful using feature mapping (or basis expansion)
  - improve the generalization by adding a regularizer
  - Probabilistic Interpretation as MLE with Gaussian noise model
- Classification with linear models:
  - $0 - 1$ loss is the difficult to work with
  - Use of surrogate loss functions such as the cross-entropy loss lead to computationally feasible solutions
  - Logistic regression as the result of using cross-entropy loss with a linear model going through logistic nonlinearity
  - No direct solution, but gradient descent can be used to minimize it
  - Probabilistic interpretation as MLE
- Gradient Descent and Stochastic Gradient Descent (SGD)