STA 314: Statistical Methods for Machine Learning I

Lecture 6 - Linear Classification II

Chris J. Maddison

University of Toronto

Very Important Announcement

- Despite my previous announcement, you do not have to be in a group for HW2.
- I have extended the HW2 deadline by 24 hours for everyone to make up for my mess.
- Please double check your submission to avoid confusion. Some students have been overwriting other's work, so check that your submission is as expected. I won't penalize students based on this issue, but please please please check your submission so that there are no surprises in a few weeks.
- If you still need to submit as a group and there are no groups, email me.

Today

- Support vector machines, elegant binary linear classifiers that generalize very well.
- Multiclass classification: predicting a discrete(> 2)-valued target.
- Stochastic gradient descent, which lets us scale up gradient descent to large data sets.

Binary Classification with a Linear Model (Small Change)

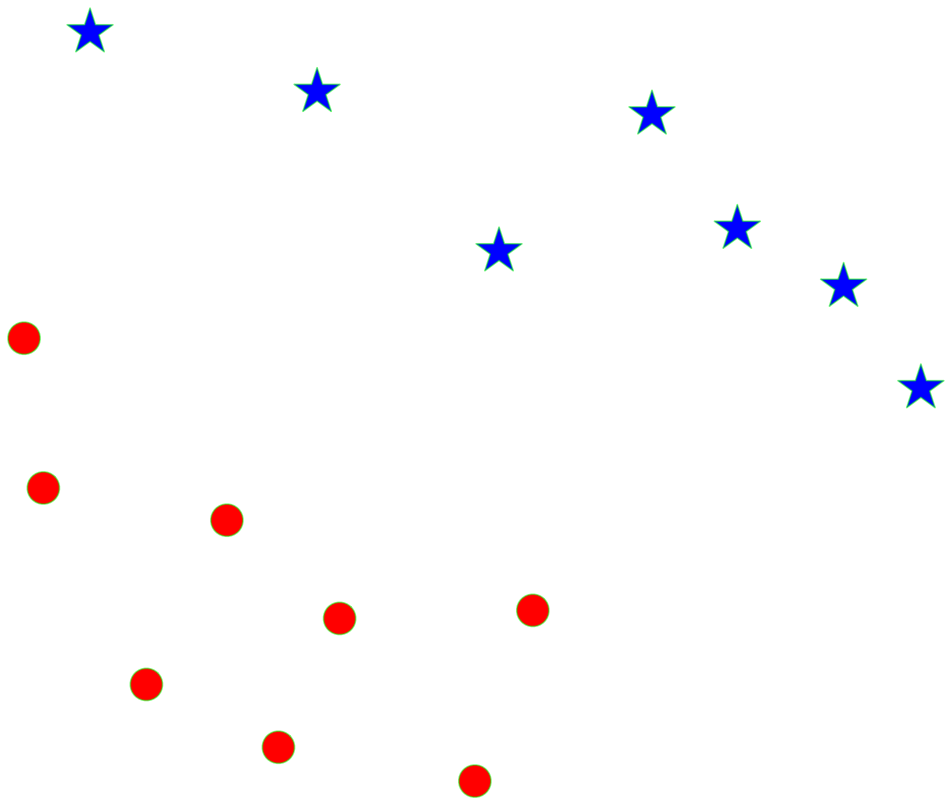
- Binary classification: predicting a target with two values
- Targets (small change from last week): $t \in \{-1, +1\}$
- Linear model (small change from last week):

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = sign(z)$$

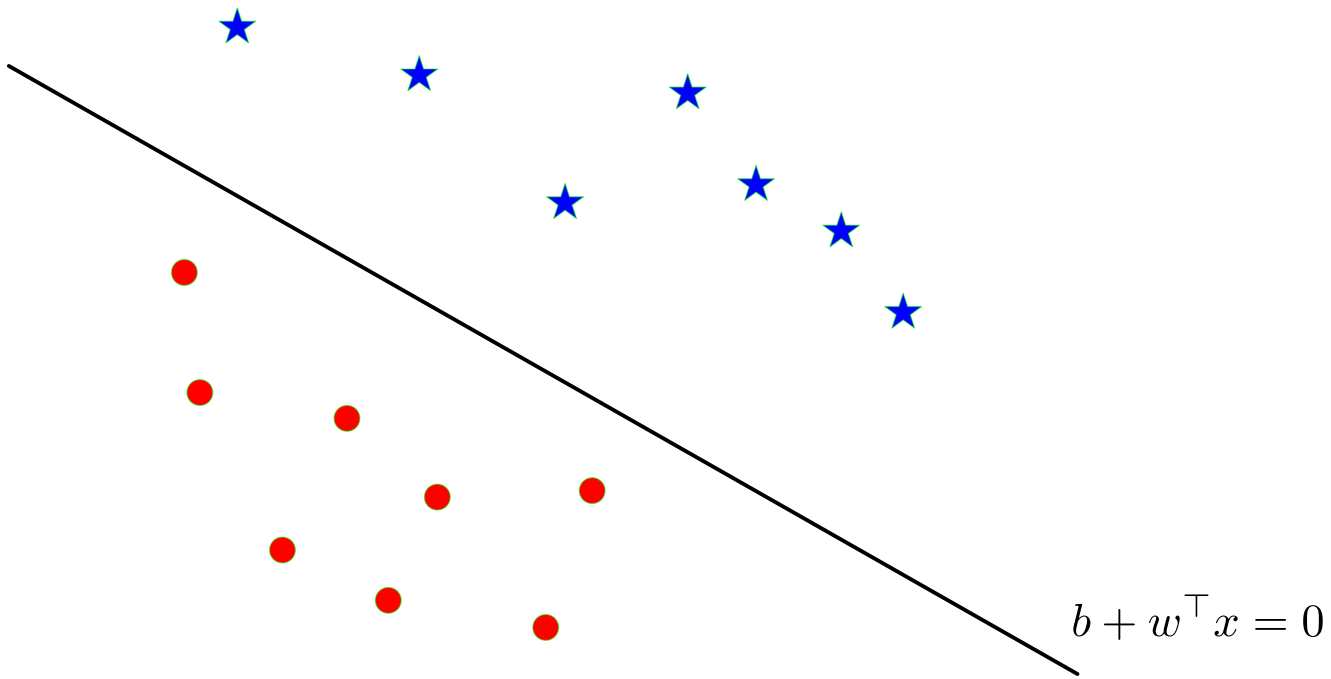
- This is an equivalent formulation of binary linear classification.
- Last week we considered how to get any $\mathbf{w}$ and b that minimized the cost on the training set.
- Question: How should we choose $\mathbf{w}$ and b to get the best generalization?

Separating Hyperplanes

Suppose we are given these data points from two different classes and want to find a linear classifier that separates them.

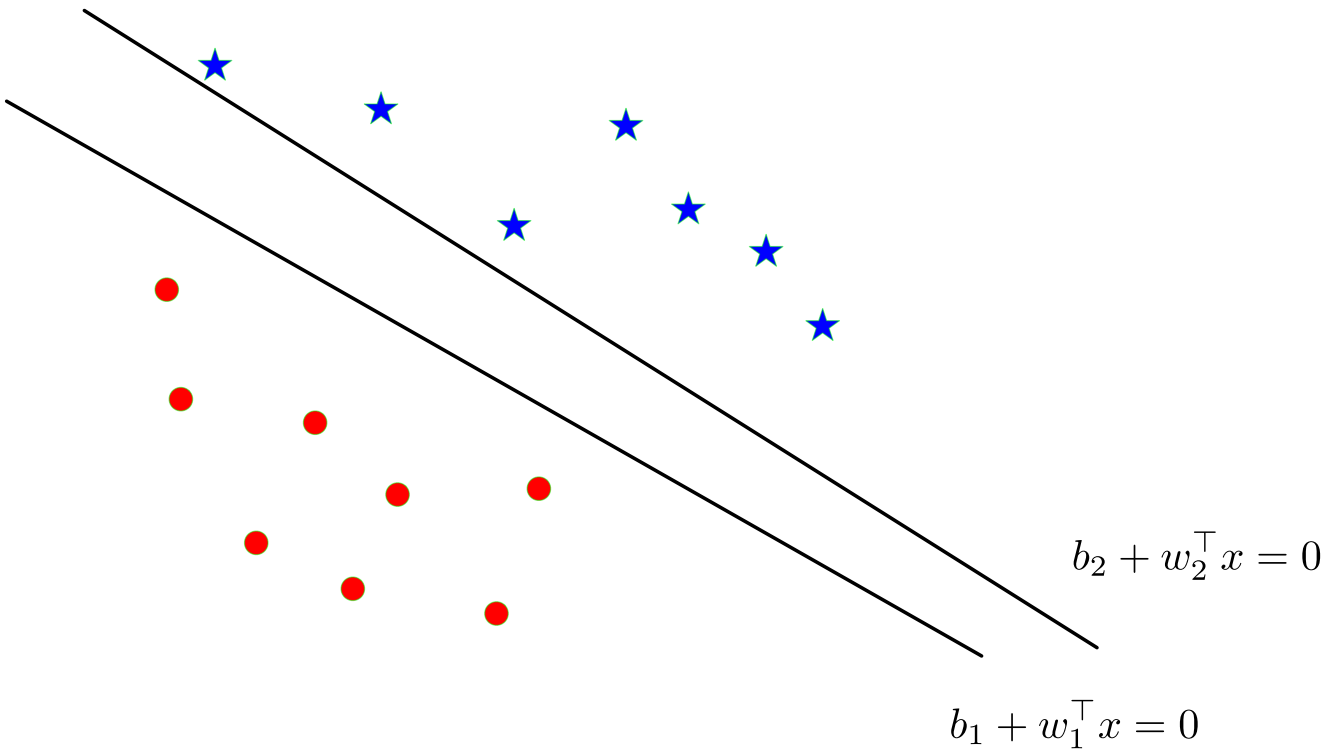


Separating Hyperplanes



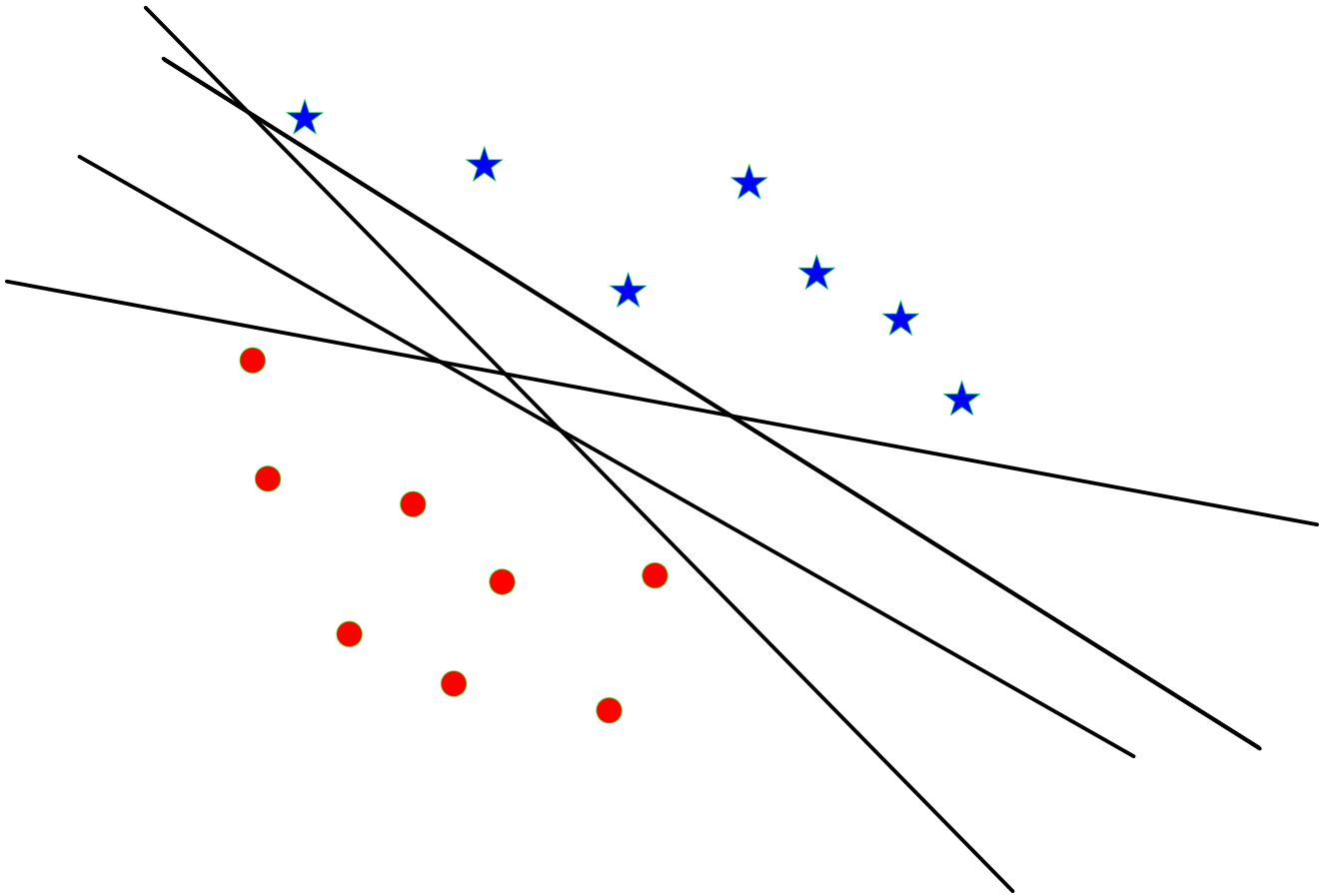
- The decision boundary looks like a line because $\mathbf{x} \in \mathbb{R}^2$, but think about it as a $D-1$ dimensional hyperplane.
- Recall that a hyperplane is described by points $\mathbf{x} \in \mathbb{R}^D$ such that $f(\mathbf{x}) = \mathbf{w}^\top x + b = 0$.

Separating Hyperplanes



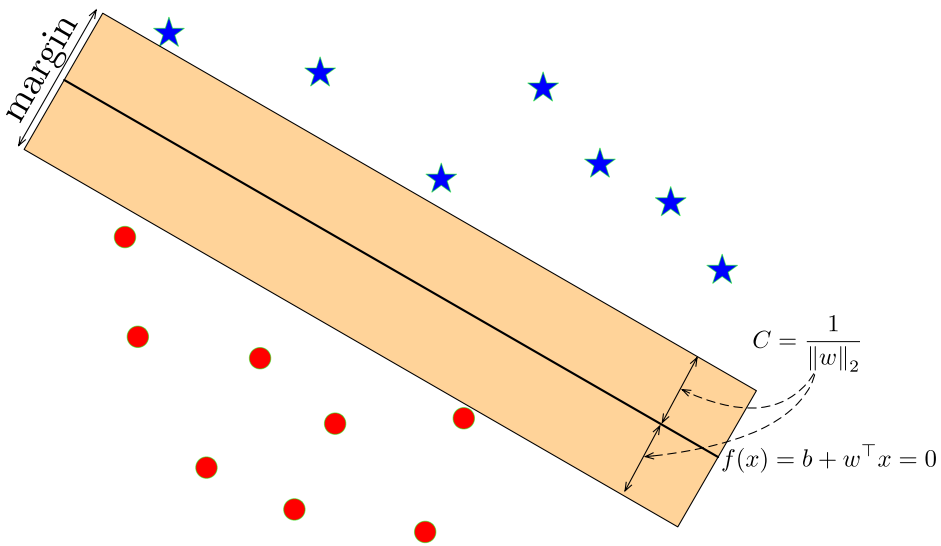
- There are multiple separating hyperplanes, described by different parameters $(\mathbf{w}, b)$.

Separating Hyperplanes



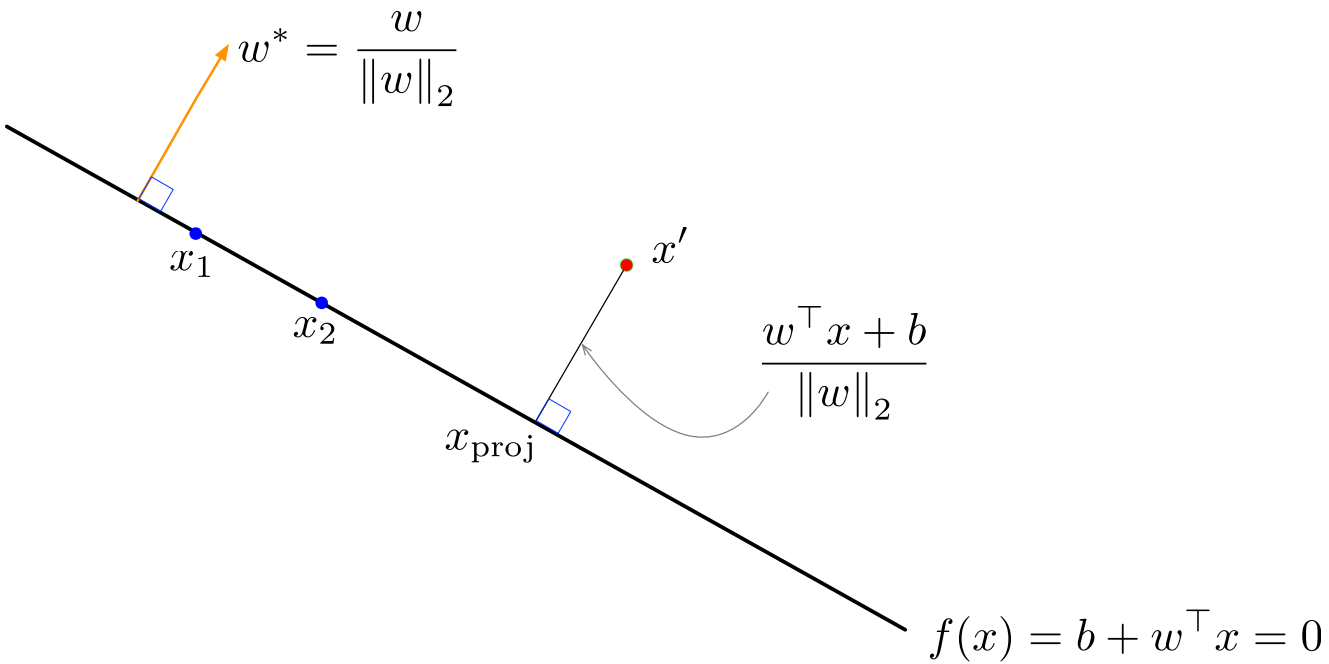
Optimal Separating Hyperplane

Optimal Separating Hyperplane: A hyperplane that separates two classes and maximizes the distance to the closest point from either class, i.e., maximize the margin of the classifier.



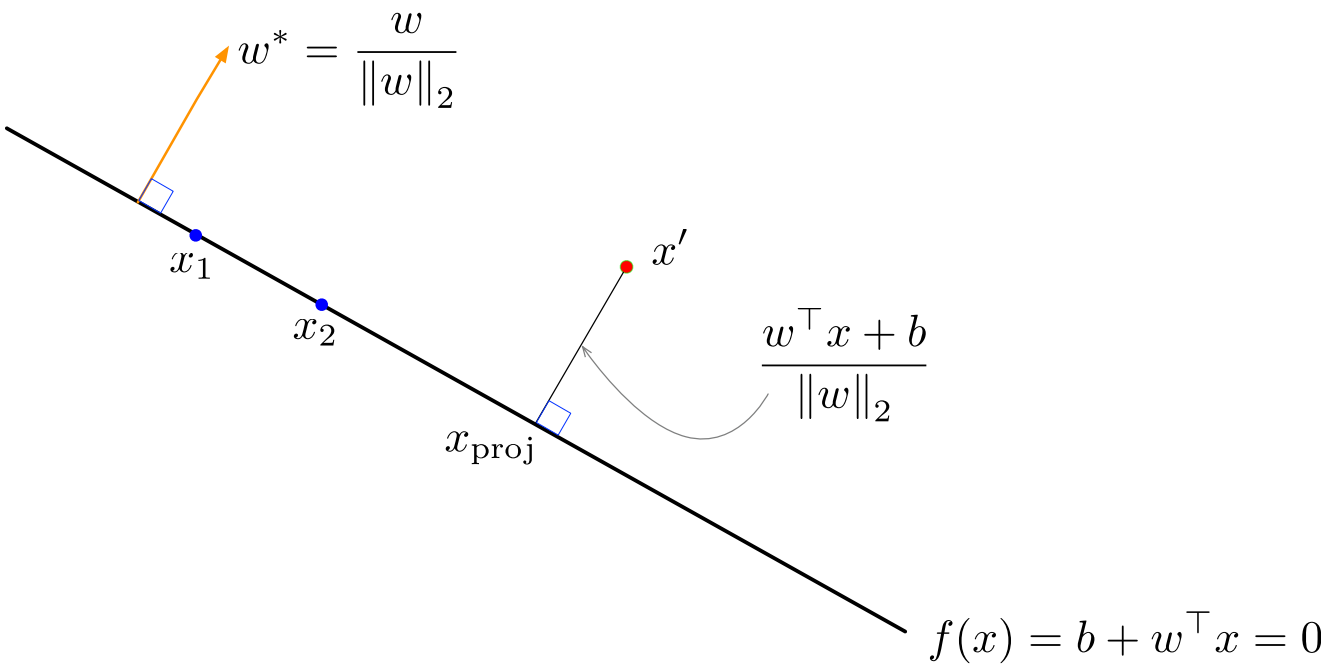
Intuitively, ensuring that a classifier is not too close to any data points leads to better generalization on the test data.

Geometry of Points and Planes



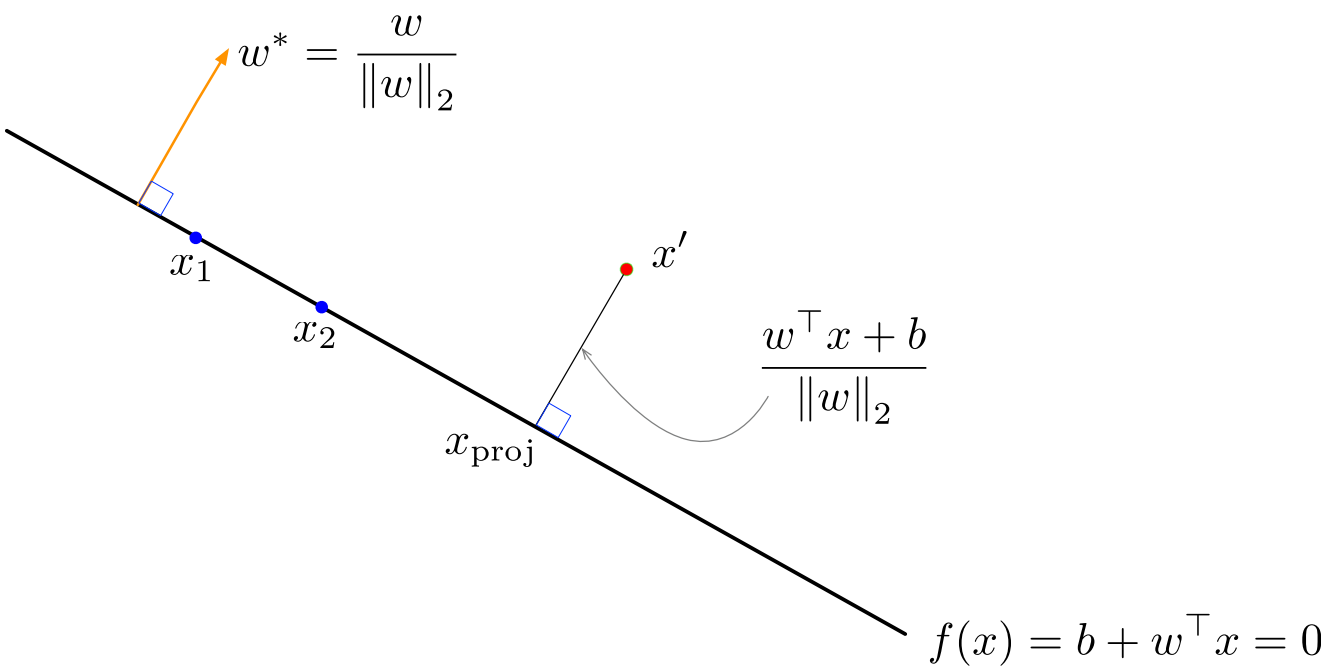
- Recall that the decision hyperplane is orthogonal (perpendicular) to $\mathbf{w}$. I.e., for any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ on the decision hyperplane we have that $\mathbf{w}^\top(\mathbf{x}_1 - \mathbf{x}_2) = 0$.

Geometry of Points and Planes



- The vector $\mathbf{w}^* = \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$ is a unit vector pointing in the same direction as $\mathbf{w}$.
- The same hyperplane could equivalently be defined in terms of $\mathbf{w}^*$.

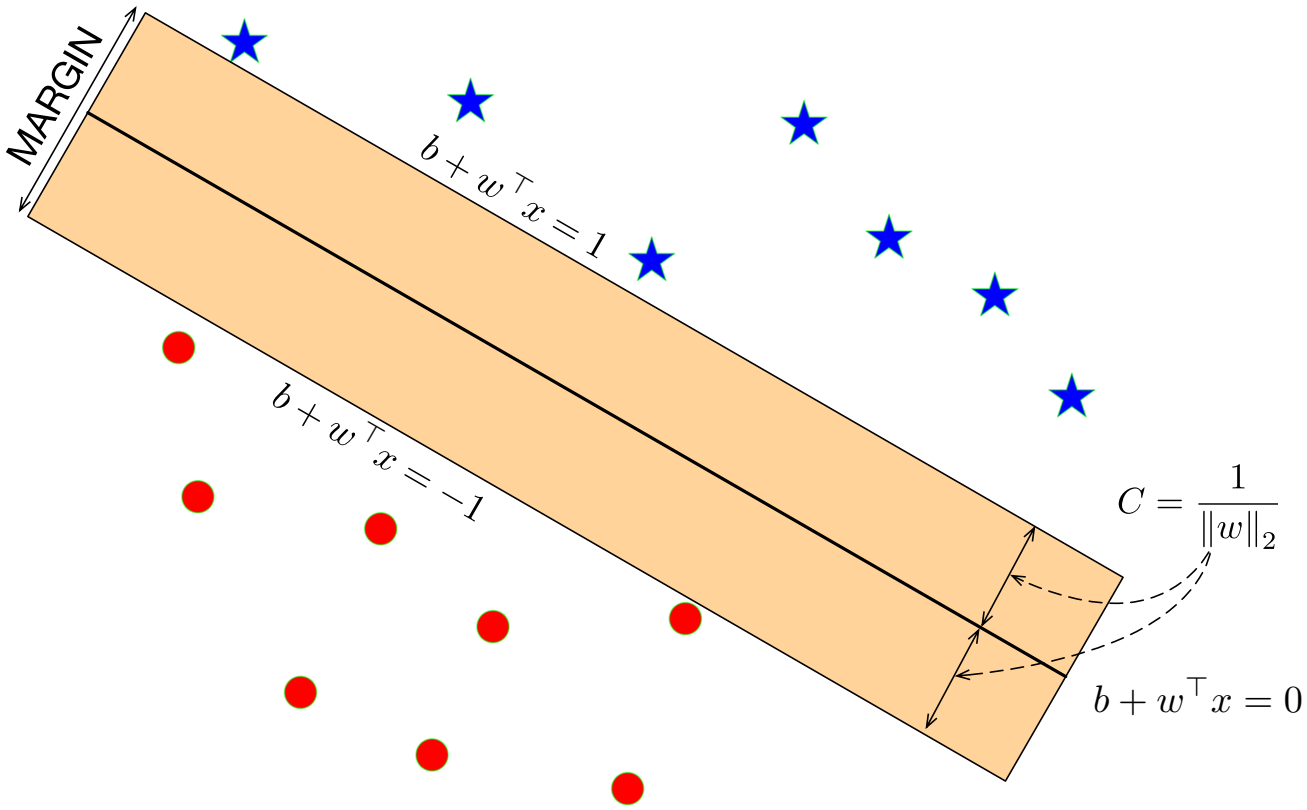
Geometry of Points and Planes



- To get the distance from a point $\mathbf{x}$ to the hyperplane, take the closest point $\mathbf{x}_{\text{proj}}$ on the hyperplane and project $\mathbf{x} - \mathbf{x}_{\text{proj}}$ onto $\mathbf{w}/\|\mathbf{w}\|_2$:

$$\left|(\mathbf{x} - \mathbf{x}_{\text{proj}})^\top \frac{\mathbf{w}}{\|\mathbf{w}\|_2}\right| = \frac{\left|\mathbf{x}^\top \mathbf{w} - \mathbf{x}_{\text{proj}}^\top \mathbf{w}\right|}{\|\mathbf{w}\|_2} = \frac{\left|\mathbf{x}^\top \mathbf{w} + b\right|}{\|\mathbf{w}\|_2}$$

Maximizing Margin as an Optimization Problem



- Now consider the two parallel hyperplanes

$$\mathbf{w}^\top \mathbf{x} + b = 1 \qquad \mathbf{w}^\top \mathbf{x} + b = -1$$

- Using the distance formula, can see that the margin is $2/\|\mathbf{w}\|_2$.

Maximizing Margin as an Optimization Problem

- Recall: to correctly classify all points we require that

$$\text{sign}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) = t^{(i)} \qquad \text{for all } i$$

- Let's impose a stronger requirement: correctly classify all points *and* prevent them from falling in the margin.

$$\mathbf{w}^\top \mathbf{x}^{(i)} + b \geq 1 \qquad \text{if } t^{(i)} = 1$$
$$\mathbf{w}^\top \mathbf{x}^{(i)} + b \leq -1 \qquad \text{if } t^{(i)} = -1$$

- This is equivalent to

$$t^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b \right) \geq 1 \qquad \text{for all } i$$

which we call the margin constraints.

Maximizing Margin as an Optimization Problem

- Now, we want to pick $\mathbf{w}, b$ that maximize the size of the margin (the region where we do not allow points to fall), while ensuring all points are correctly classified.
  - Margin has width $2/\left\|\mathbf{w}\right\|_2$, so maximizing this is equivalent to minimizing $\left\|\mathbf{w}\right\|_2^2$.
- This leads to the max-margin objective:

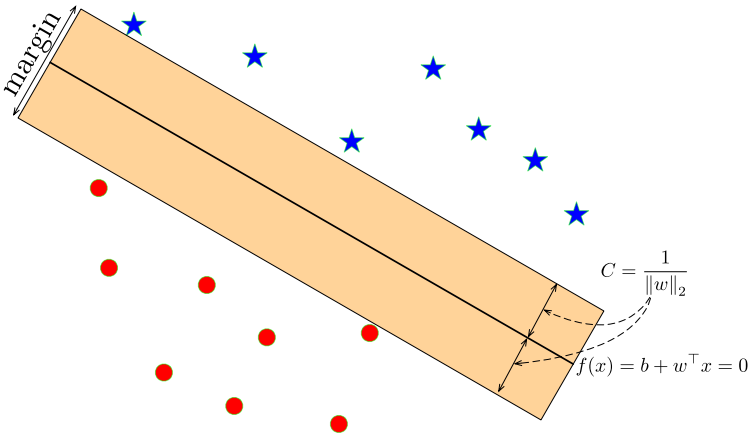
$$\min_{\mathbf{w},b} \left\|\mathbf{w}\right\|_2^2$$
$$\text{s.t. } t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \qquad i = 1, \ldots, N$$

Maximizing Margin as an Optimization Problem

Max-margin objective:

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_2^2$$

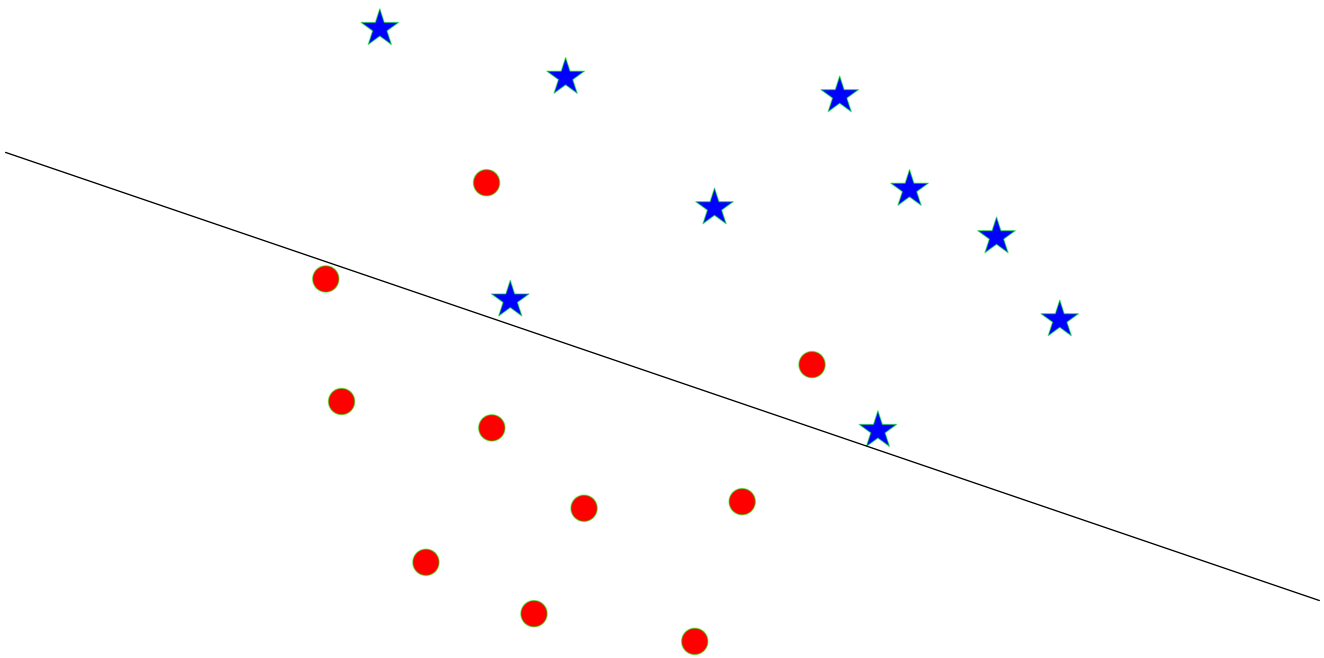
$$\text{s.t. } t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \qquad i = 1, \ldots, N$$



- Observe: if the margin constraint is not tight for $\mathbf{x}^{(i)}$, we could remove it from the training set and the optimal $\mathbf{w}$ would be the same.
- The important training examples are the ones with algebraic margin 1, and are called support vectors.
- Hence, this algorithm is called the (hard) Support Vector Machine (SVM) (or Support Vector Classifier).
- SVM-like algorithms are often called max-margin or large-margin.

Non-Separable Data Points

How can we apply the max-margin principle if the data are **not** linearly separable?



Hinge Loss

- There is an elegant relaxation of the max-margin objective called soft-margin SVM for the non-separable case. We won't cover it carefully, but let's motivate it.
- We can measure the extent to which $\mathbf{x}^{(i)}$ violates its margin constraint by the magnitude of

$$1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$$

  - If this is very positive, then $t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) < 1$.
  - If this is not positive, then $t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1$.

Hinge Loss

- We want $1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$ to be small, but if it is negative, we don't care how negative it is.
- This motivates the hinge loss for $z^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$:

$$L_{\text{hinge}}(z^{(i)}) = \max\left(0, 1 - t^{(i)} z^{(i)}\right)$$

Soft-margin SVM

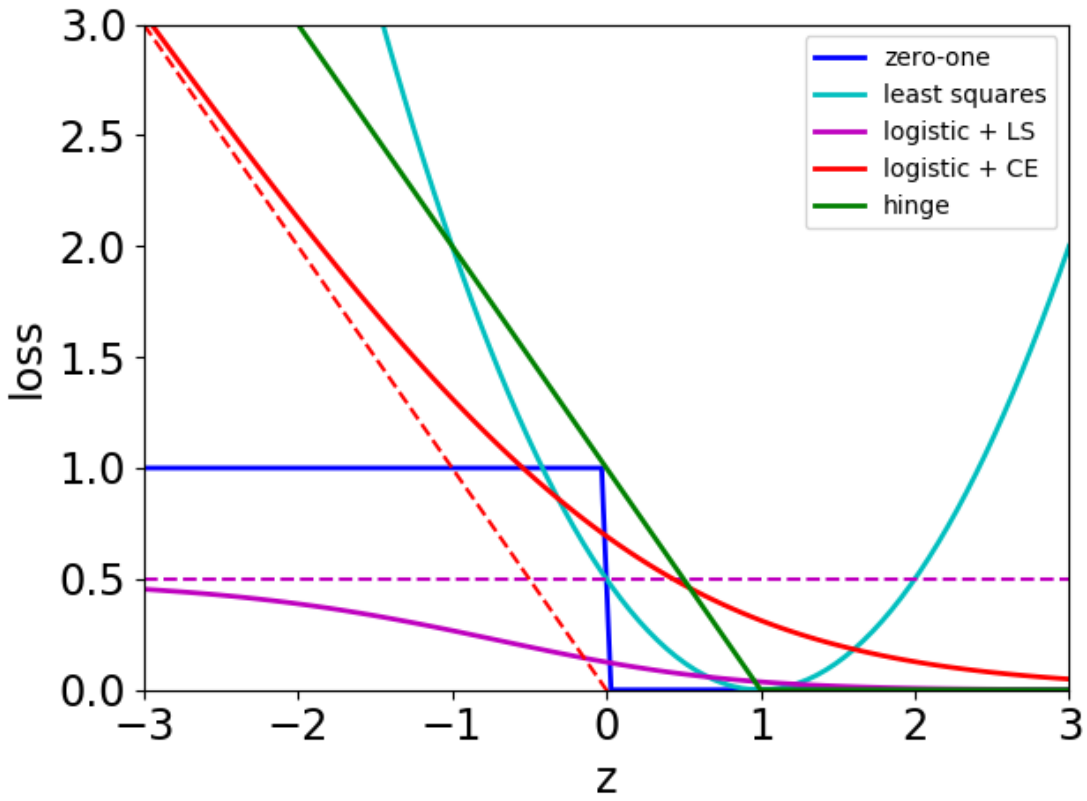
- Soft-margin SVM minimizes the average hinge losses plus the norm of the weights, where $z^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$:

$$\min_{\mathbf{w},b} \sum_{i=1}^{N} \frac{1}{N} \max\left(0, 1 - t^{(i)} z^{(i)}\right) + \lambda \left\|\mathbf{w}\right\|_2^2$$

- Hence, the soft-margin SVM can be seen as a linear classifier with hinge loss and an L_2 regularizer.

Revisiting Loss Functions for Classification

Hinge loss compared with other loss functions



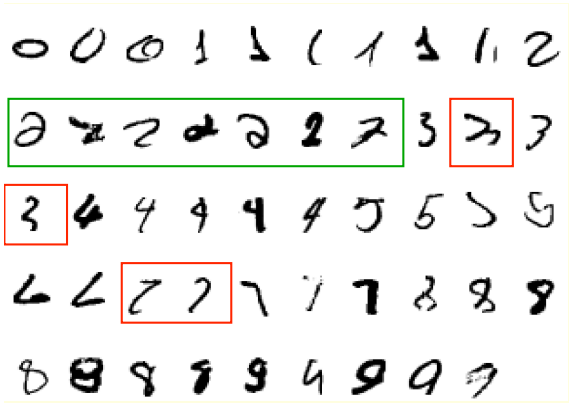
SVMs: What we Left Out

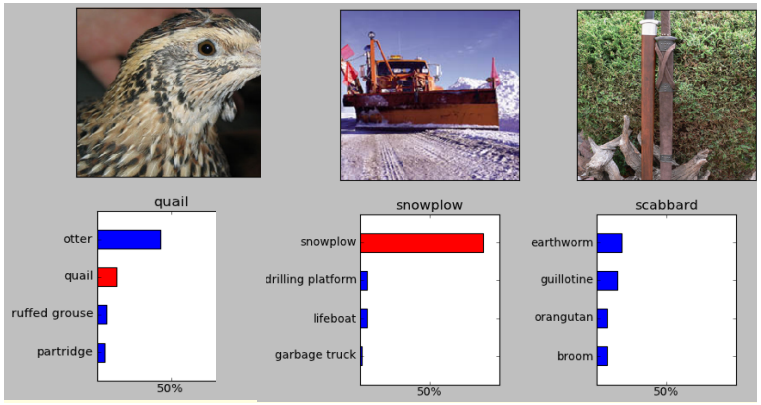
What we left out:

- How to fit $\mathbf{w}$ for the max-margin SVM:
  - One option: gradient descent
- Classic results from learning theory show that a large margin implies good generalization.

Multiclass Classification

- So far, we've only talked about binary linear 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:

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

- We will build multiclass linear classifiers by generalizing binary linear classifiers and logistic regression (not SVMs).

Multiclass Linear Classification

- We can start with a linear function of the inputs.
- D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W} \in \mathbb{R}^{K \times D}$.
- Also, we have a vector $\mathbf{b} \in \mathbb{R}^K$ of bias parameters.
- A linear function of an input $\mathbf{x} \in \mathbb{R}^D$:

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

- Eliminate the bias parameters by taking $\mathbf{W} \in \mathbb{R}^{K \times (D+1)}$ and adding a dummy variable $x_0 = 1$.
- So, vectorized we have the vector $\mathbf{z} \in \mathbb{R}^K$:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad \text{or with dummy } x_0 = 1 \quad \mathbf{z} = \mathbf{W}\mathbf{x}$$

Multiclass Linear Classification

- How can we turn this linear prediction into a one-hot prediction?
- We can interpret the magnitude of z_k as an measure of how much the model prefers k as its prediction.
- If we do this, we should set

$$y_i = \begin{cases} 1 & i = \arg\max_{k=1}^{K} z_k \\ 0 & \text{otherwise} \end{cases}$$

- **Exercise:** how does the case of $K = 2$ relate to the prediction rule in binary linear classifiers?

Softmax Regression

- As with binary classification, we need to soften our predictions for the sake of optimization.
- We want predictions that are like probabilities, i.e., $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.
- We can use the softmax function, a multivariable generalization of the logistic function:

$$y_k = \mathrm{softmax}(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

  - Outputs can be interpreted as probabilities (positive and sum to 1)
  - If z_k is much larger than the others, then $\mathrm{softmax}(\mathbf{z})_k \approx 1$ and it behaves like argmax.
  - **Exercise:** how does the case of $K = 2$ relate to the logistic function?
- The inputs z_k are called the logits.

Softmax Regression

- If a model outputs a vector $\mathbf{y} \in \mathbb{R}^K$ of class probabilities, we can use cross-entropy as the loss function:

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

  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.

Softmax Regression

- Softmax regression (with dummy $x_0 = 1$):

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$
$$\mathbf{y} = \mathrm{softmax}(\mathbf{z})$$
$$L_{\mathrm{CE}} = -\mathbf{t}^\top(\log \mathbf{y})$$

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

$$\frac{\partial L_{\mathrm{CE}}}{\partial \mathbf{w}_k} = \frac{\partial L_{\mathrm{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) \mathbf{x}^{(i)}$$

- Similar to linear/logistic reg (no coincidence) (verify the update)

What about SVMs?

- Not trivial to generalize the notion of a margin to multiclass setting.
- Many different proposals for multi-class SVMs, but outside of the scope of this course.

Batch Gradient Descent

- Let's imagine we have a prediction function $y(\mathbf{x}, \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$, e.g. $\boldsymbol{\theta} = \mathbf{w}$ in logisitic regression.
- So far, the cost function $\hat{\mathcal{R}}$ has been the average loss over the training examples:

$$\hat{\mathcal{R}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} L^{(i)} = \frac{1}{N} \sum_{i=1}^{N} L(y(\mathbf{x}^{(i)}, \boldsymbol{\theta}), t^{(i)}).$$

- By linearity,

$$\frac{\partial \hat{\mathcal{R}}}{\partial \boldsymbol{\theta}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L^{(i)}}{\partial \boldsymbol{\theta}}.$$

- Computing the gradient requires summing over *all* of the training examples. This is known as batch training.

Stochastic Gradient Descent

- Batch training is impractical if you have a large dataset (e.g. millions of training examples)!
- Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example,

$$\begin{aligned} &1- \quad \text{Choose } i \text{ uniformly at random,} \\ &2- \quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial L^{(i)}}{\partial \boldsymbol{\theta}} \end{aligned}$$

- 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:

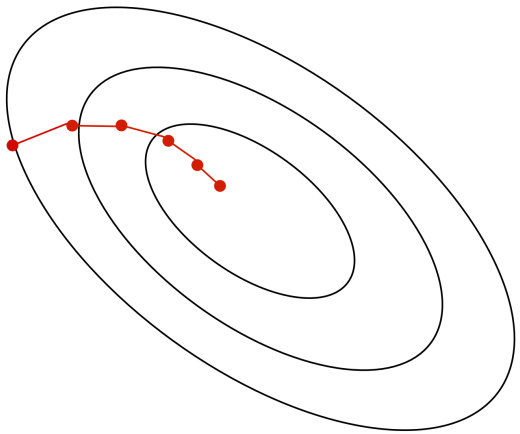
$$\mathbb{E}\left[\frac{\partial L^{(i)}}{\partial \boldsymbol{\theta}}\right] = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial L^{(i)}}{\partial \boldsymbol{\theta}} = \frac{\partial \hat{\mathcal{R}}}{\partial \boldsymbol{\theta}}.$$

Stochastic Gradient Descent

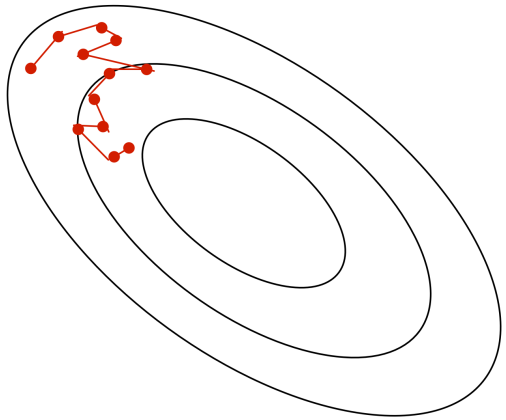
- Problems with using single training example to estimate gradient:
  - Variance in the estimate may be high
  - We can't exploit efficient vectorized operations
- Compromise approach:
  - compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \ldots, N\}$, called a mini-batch.
- Stochastic gradients computed on larger mini-batches have smaller variance.
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.
  - Too large: requires more compute; e.g., it 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 $|\mathcal{M}| = 100$.

Stochastic Gradient Descent

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



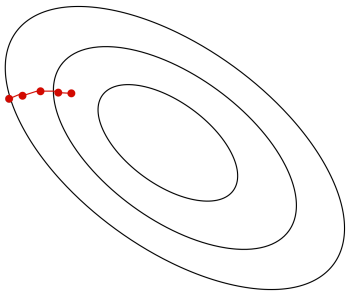
batch gradient descent



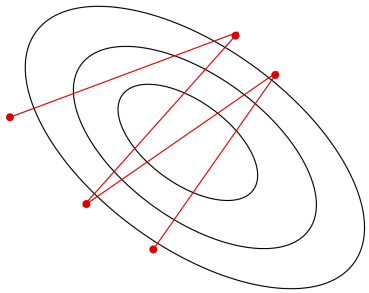
stochastic gradient descent

Learning Rate

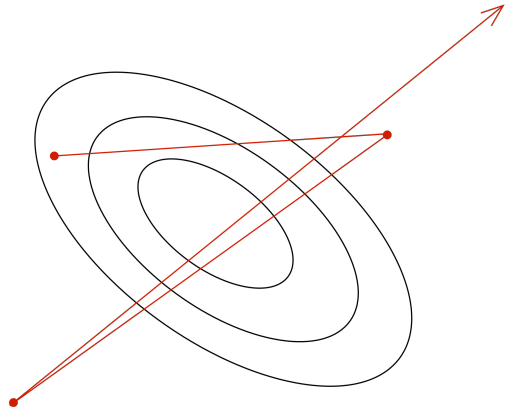
- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:



α too small:
slow progress



α too large:
oscillations



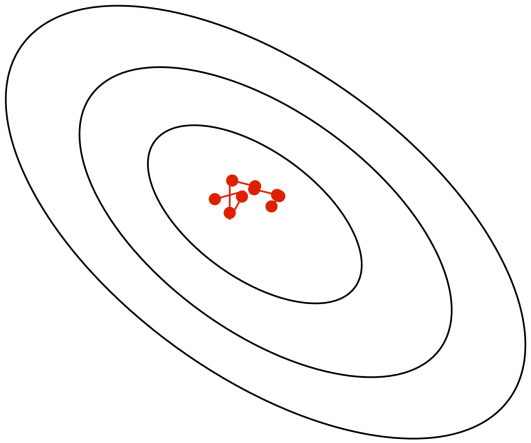
α much too large:
instability

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

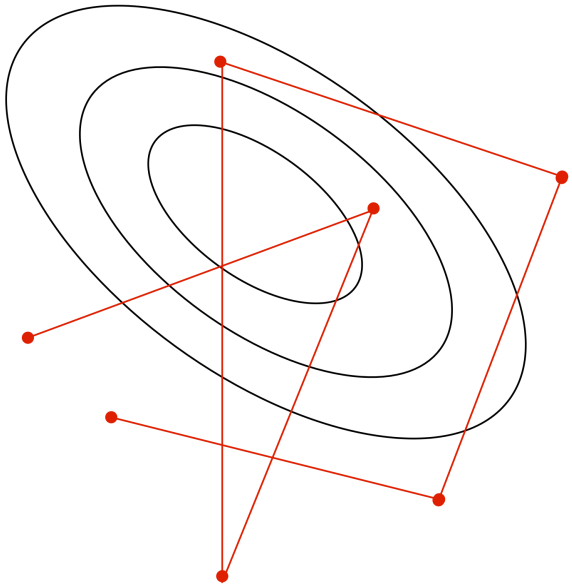
SGD Learning Rate

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

small learning rate



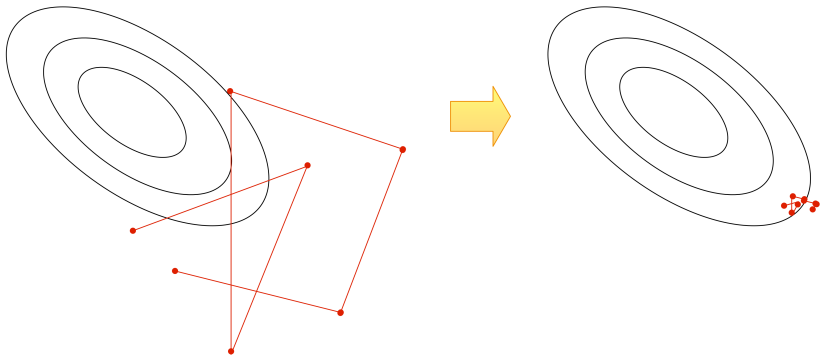
large learning rate

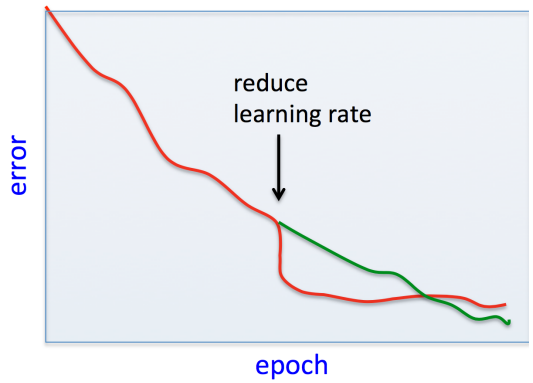


- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations

SGD Learning Rate

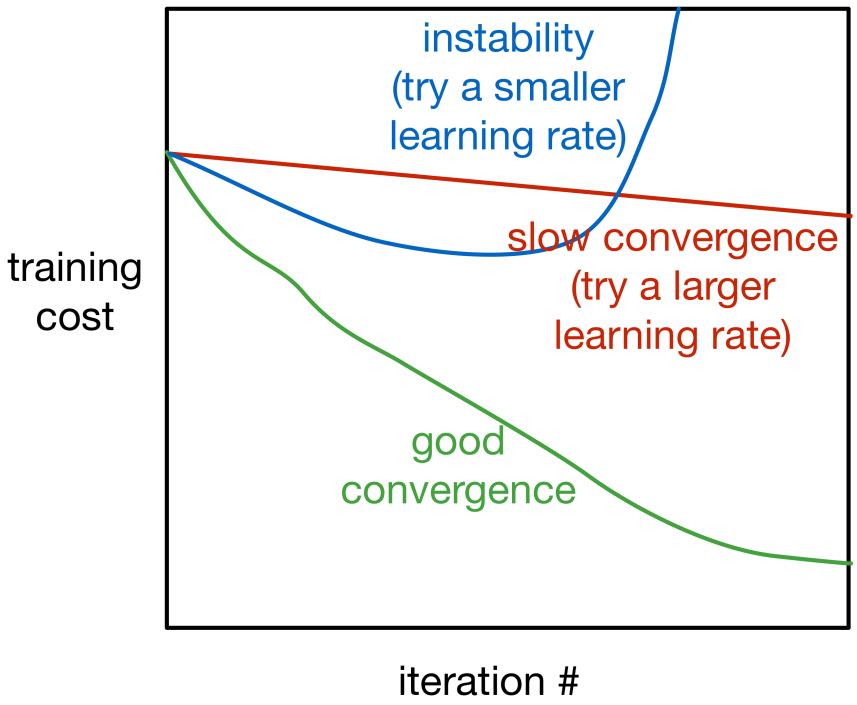
- Warning: by reducing the learning rate, you reduce the fluctuations, which can appear to make the loss drop suddenly. But this can come at the expense of long-run performance.





Training Curves

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



- Warning: it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

The End of Supervised Learning

- This is the end of supervised learning in this course (sort of), next week we will move on to unsupervised learning.

The following slides are optional and will not be tested.

Gradient Checking

- We've derived a lot of gradients so far. How do we know if they're correct?
- Recall the definition of the partial derivative:

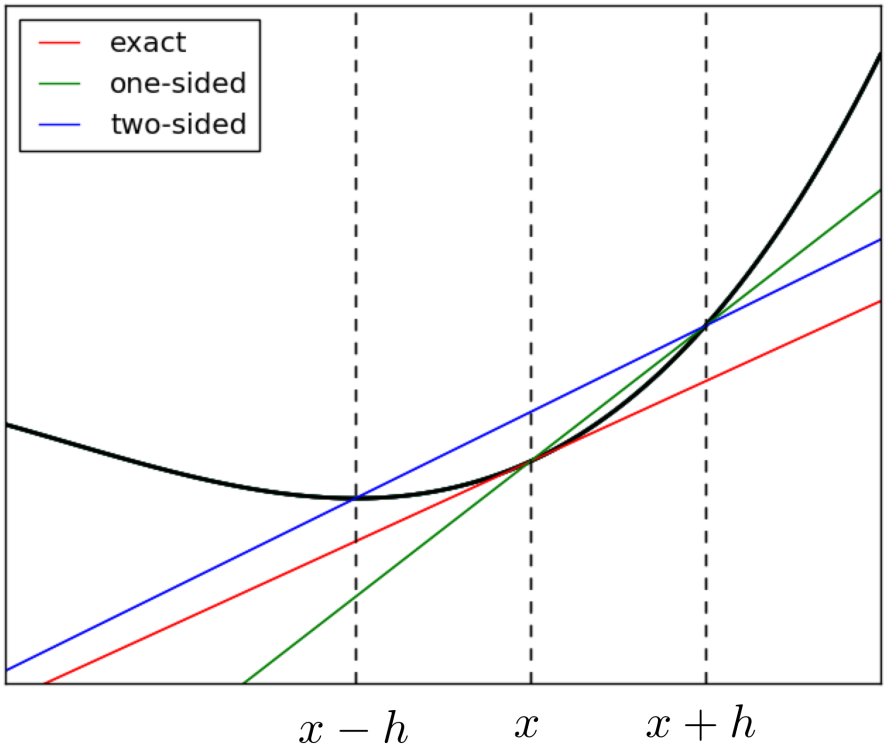
$$\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N) = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_N) - f(x_1, \ldots, x_i, \ldots, x_N)}{h}$$

- Check your derivatives numerically by plugging in a small value of h, e.g. 10^{-10}. This is known as finite differences.

Gradient Checking

- Even better: the two-sided definition

$$\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N) = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_N) - f(x_1, \ldots, x_i - h, \ldots, x_N)}{2h}$$



Gradient Checking

- Run gradient checks on small, randomly chosen inputs
- Use double precision floats (not the default for TensorFlow, PyTorch, etc.!)
- Compute the relative error:

$$\frac{|a-b|}{|a|+|b|}$$

- The relative error should be very small, e.g. 10^{-6}

Gradient Checking

- Gradient checking is really important!
- Learning algorithms often appear to work even if the math is wrong.
- **But:**
  - They might work much better if the derivatives are correct.
  - Wrong derivatives might lead you on a wild goose chase.
- If you implement derivatives by hand, gradient checking is the single most important thing you need to do to get your algorithm to work well.