

DSC 240: Machine Learning

Continuous Optimization and SGD

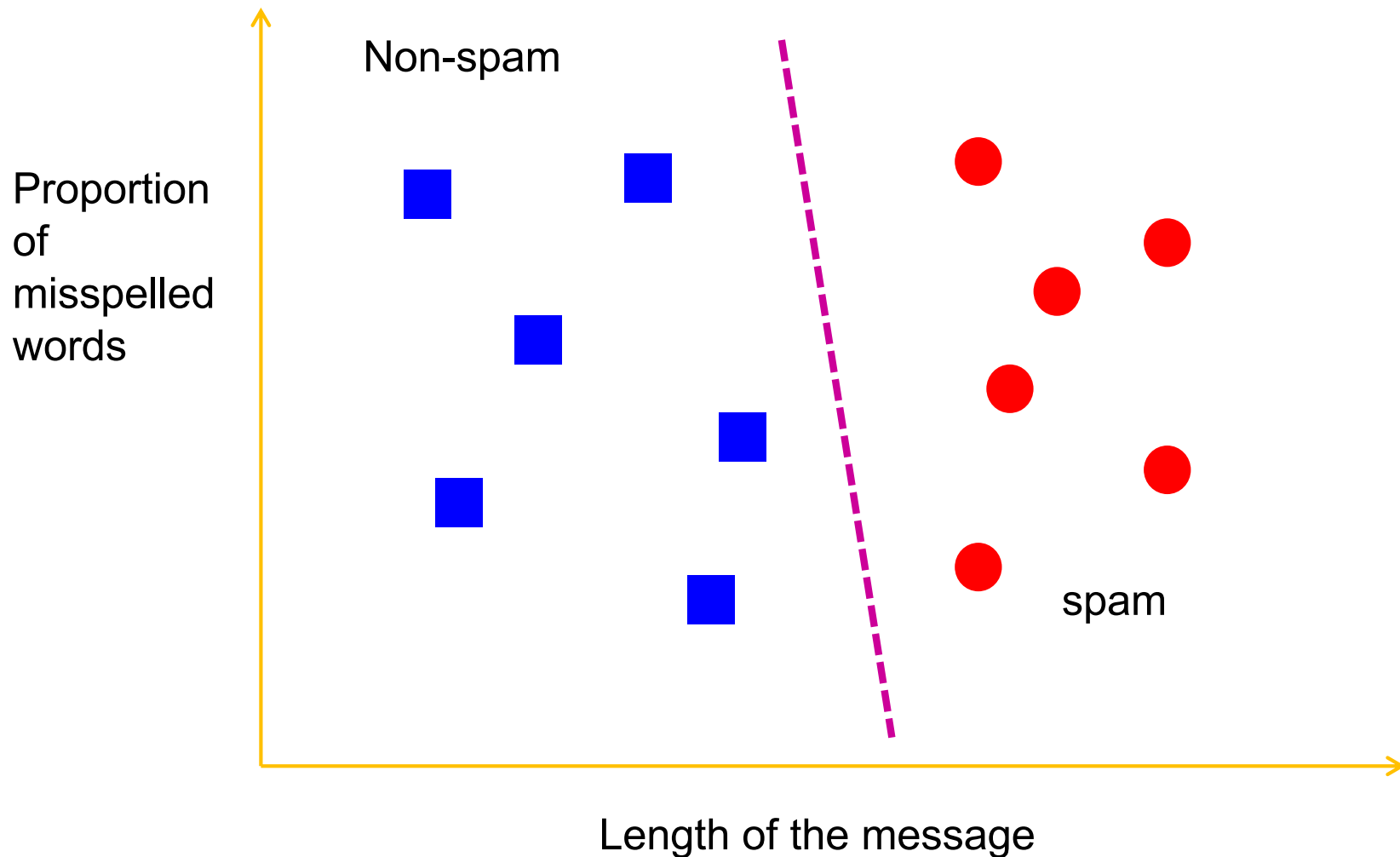
Jan 23, 2025

Instructor: Prof. Yu-Xiang Wang

Today

- Non-convergence of Perceptron in non-linear-separable cases
- Surrogate loss and continuous optimization
- Gradient Descent and SGD

Last lecture: How to learn a linear classifier? Focus on the “linearly separable” case

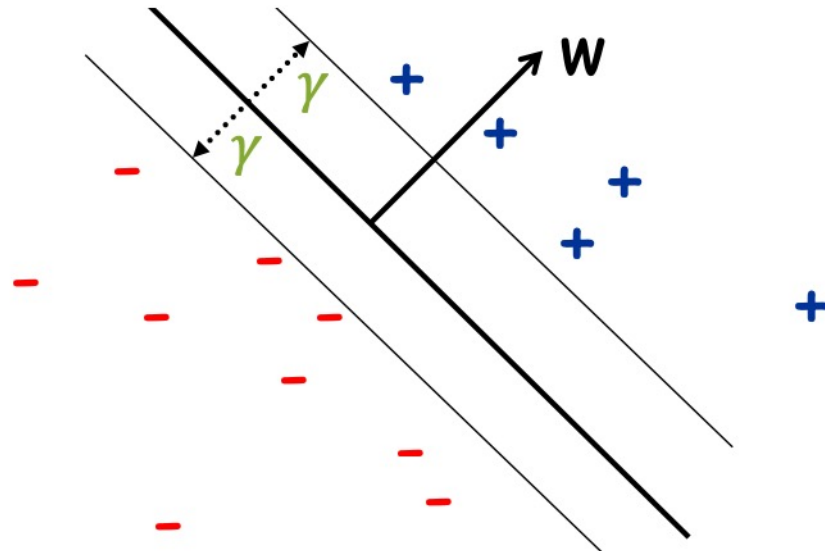


Recap: Geometric Margins

Definition: The **margin** of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The **margin** γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.

Definition: The **margin** γ of a set of examples S is the **maximum** γ_w over all linear separators w .



Recap: The perceptron algorithm

The Perceptron algorithm

Initialize $w_1 = [0, 0, \dots, 0]$

while 1:

1. Nature selects feature vector x
2. Agent makes prediction $\hat{y} \leftarrow \text{sign}(w_t \cdot x)$
3. Nature reveals label y
4. **If** $\hat{y} \neq y$:
 - a. $w_{t+1} \leftarrow w_t + yx$
 - b. $t \leftarrow t + 1$

Recap: Perceptron Algorithm makes finite number of mistakes! (when the data is **linearly separable with margin > 0**)

Theorem (Novikoff, 1962): Assume there exists w^* such that every input satisfies that $\frac{|x \cdot w^*|}{\|w^*\|_2} \geq \gamma$ and $y = \text{sign}(x \cdot w^*)$

Also, assume that $\|x\|_2 \leq R$. Then the **total number of mistakes** Perceptron makes is smaller than R^2 / γ^2

Remarks:

- The algorithm can run infinitely long => No mistake on average.
- No assumption on data distribution (can be fully adversarial, given the margin constraint)
- Every new data point to predict on is new.
- No hyperparameters to choose. The alg does not need to know R, γ .
- Dimension-free. Can support infinitely many features!

Using Perceptron in batch (offline) mode

$$\min_{w \in \mathbb{R}^d} \text{Error}(w) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}(h_w(x_i) \neq y_i)$$

- Recall: Perceptron can take any sequence of data
- Loop over the data again and again until there is no mistakes.

```
while not converged:  
    converged = true  
    for over data set:  
        make prediction  
        If "Mistake": 1. Update weights. 2. set converged = false
```

- Converge in after at most $\frac{nR^2}{\gamma^2}$ inner loop iterations.

Limitations of the Perceptron Algorithm

- What if the margin is small?
 - The perceptron algorithm will be slower
- What if non-linearly separable?
 - The perceptron algorithm is known to not converge!

Demo of the perceptron algorithms

- Large-margin
- Small-margin
- Non-separable case

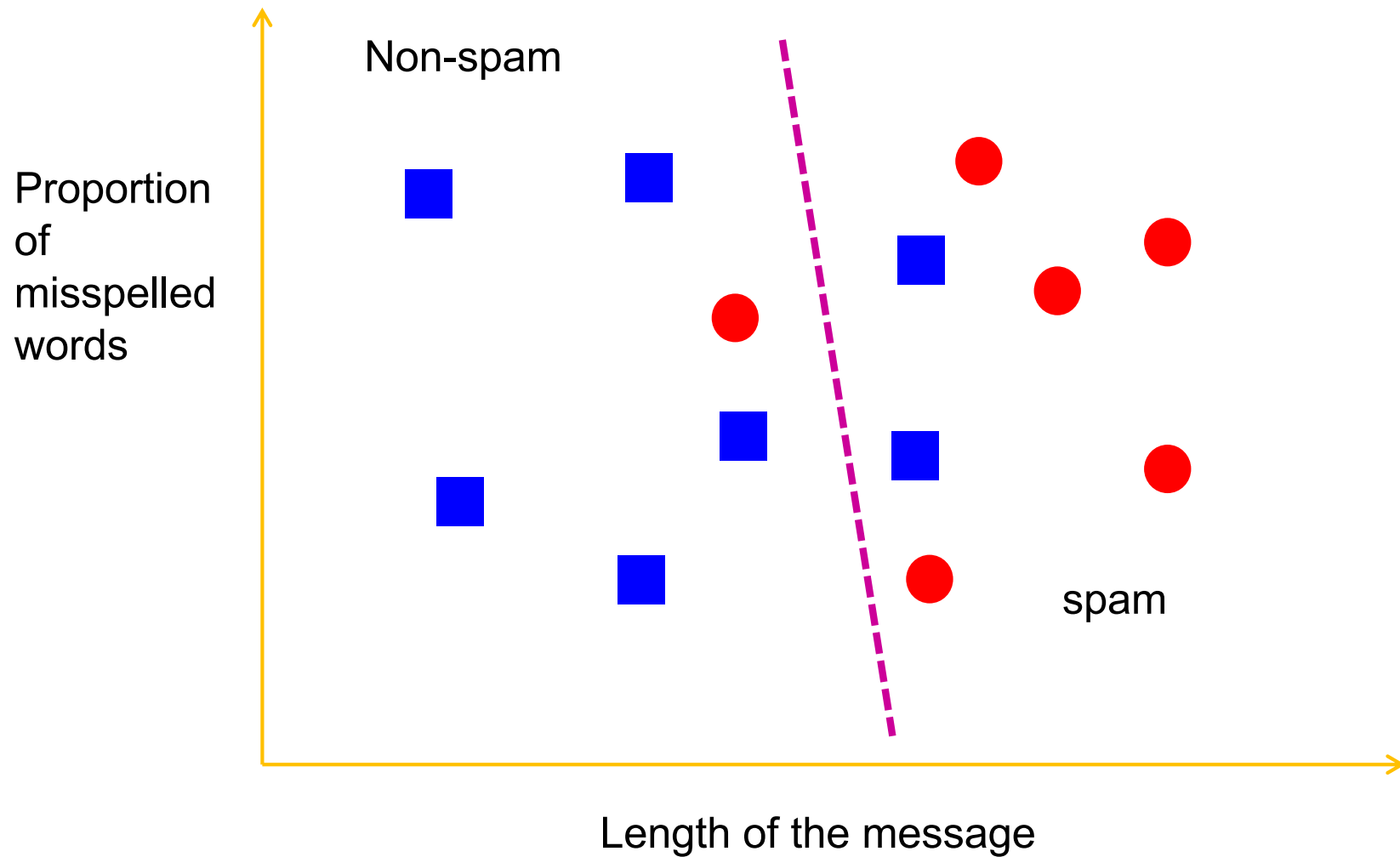
Black & White game's hyped learning AI used the Perceptron... but is unstable... And it is ruining the game



**Avatar learns from
your behavior**

**Black & White
Lionsgate Studios**

Best linear separator in general (linearly non-separable cases) is NP-hard.



What do we do? The general idea is to **just relax...**

- Maybe we can minimize an approximation to the objective function that is computationally easier
- So we can focus on modelling (e.g., better features, better hypothesis class)

Just “relax”: relaxing a hard problem into an easier one

$$\min_{w \in \mathbb{R}^d} \text{Error}(w) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}(\text{sign}(w^T x_i) \neq y_i)$$

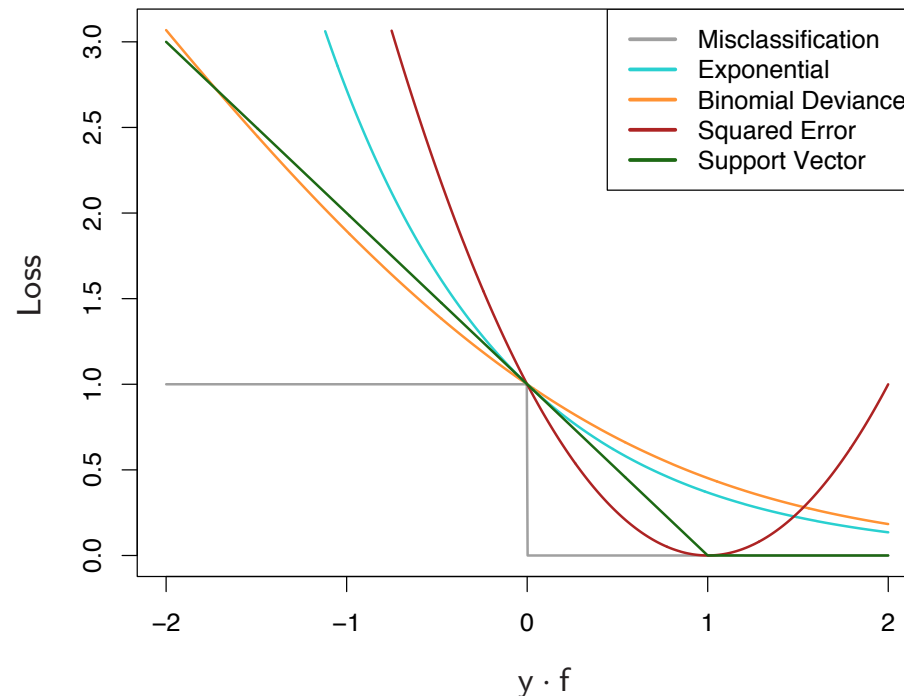


$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \ell(w^T x_i, y_i).$$

Loss functions and surrogate losses

- 0-1 loss: $\mathbf{1}(h_w(x) \neq y) = \mathbf{1}(\text{sign}(S_w(x)) \neq y)$
- Square loss: $(y - S_w(x))^2$
- Logistic loss: $\log_2(1 + \exp(-y \cdot S_w(x)))$
- Hinge loss: $\max(0, 1 - y \cdot S_w(x))$

Visualizing the relaxed “surrogate loss” functions



** “Binomial deviance” is the “logistic loss” from the previous slide.

FIGURE 10.4. Loss functions for two-class classification. The response is $y = \pm 1$; the prediction is f , with class prediction $\text{sign}(f)$. The losses are misclassification: $I(\text{sign}(f) \neq y)$; exponential: $\exp(-yf)$; binomial deviance: $\log(1 + \exp(-2yf))$; squared error: $(y - f)^2$; and support vector: $(1 - yf)_+$ (see Section 12.3). Each function has been scaled so that it passes through the point $(0, 1)$.

Intuition of the logistic loss (3 min discussion)

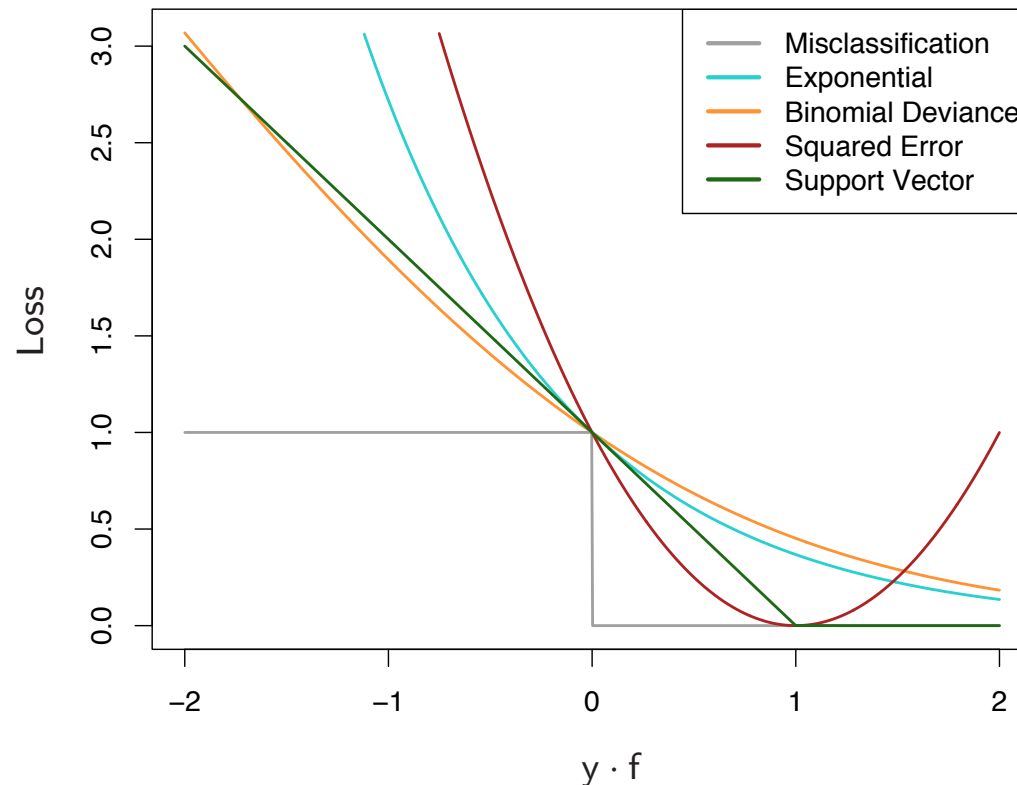
$$\log_2(1 + \exp(-y \cdot S_w(x)))$$

Try plotting the logistic loss as a function of $y \cdot S_w(x)$

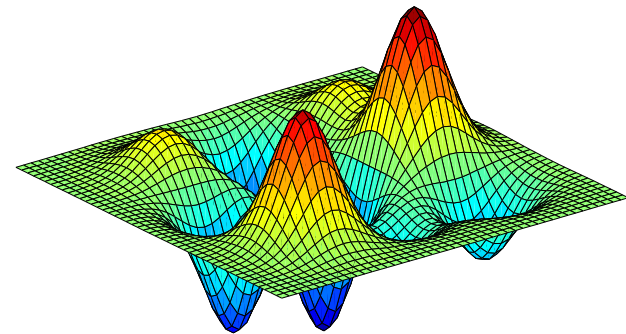
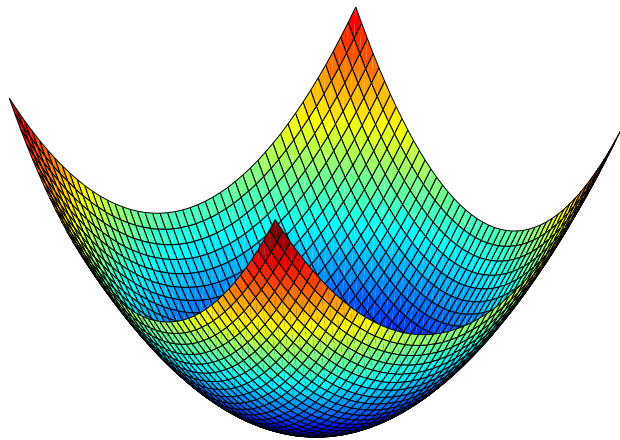
1. What happens when the classifier predicts correctly?
2. What happens when the classifier correct?
3. What role does the magnitude of the score function play?

Why are “surrogate losses” easier to minimize?

- They are continuous.
- Differentiable (except hinge loss, i.e., “support vector” in the figure below).
- Convex



Convex vs Nonconvex optimization



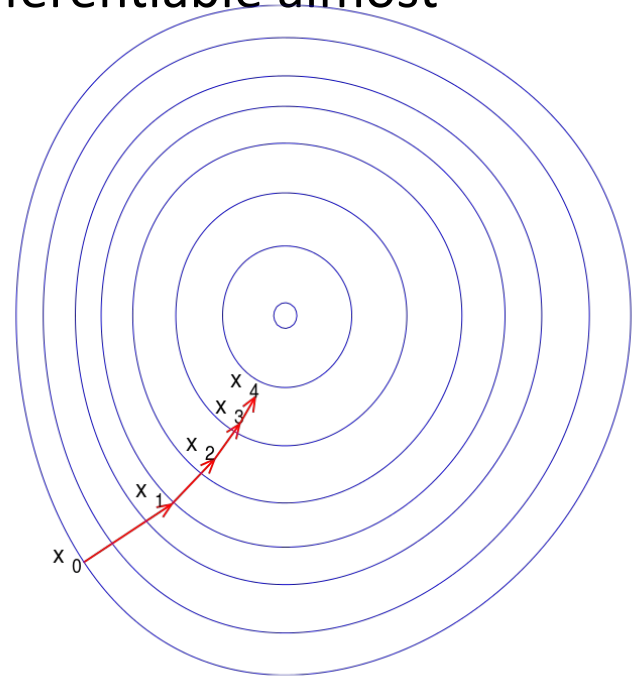
- Unique optimum: global/local.
- Multiple local optima
- In high dimensions possibly exponential local optima

* Be careful: The surrogate loss being convex does not imply all ML problems using surrogate losses are convex. Linear classifiers are, but non-linear classifiers are usually not. Take “convex optimization” to know more.

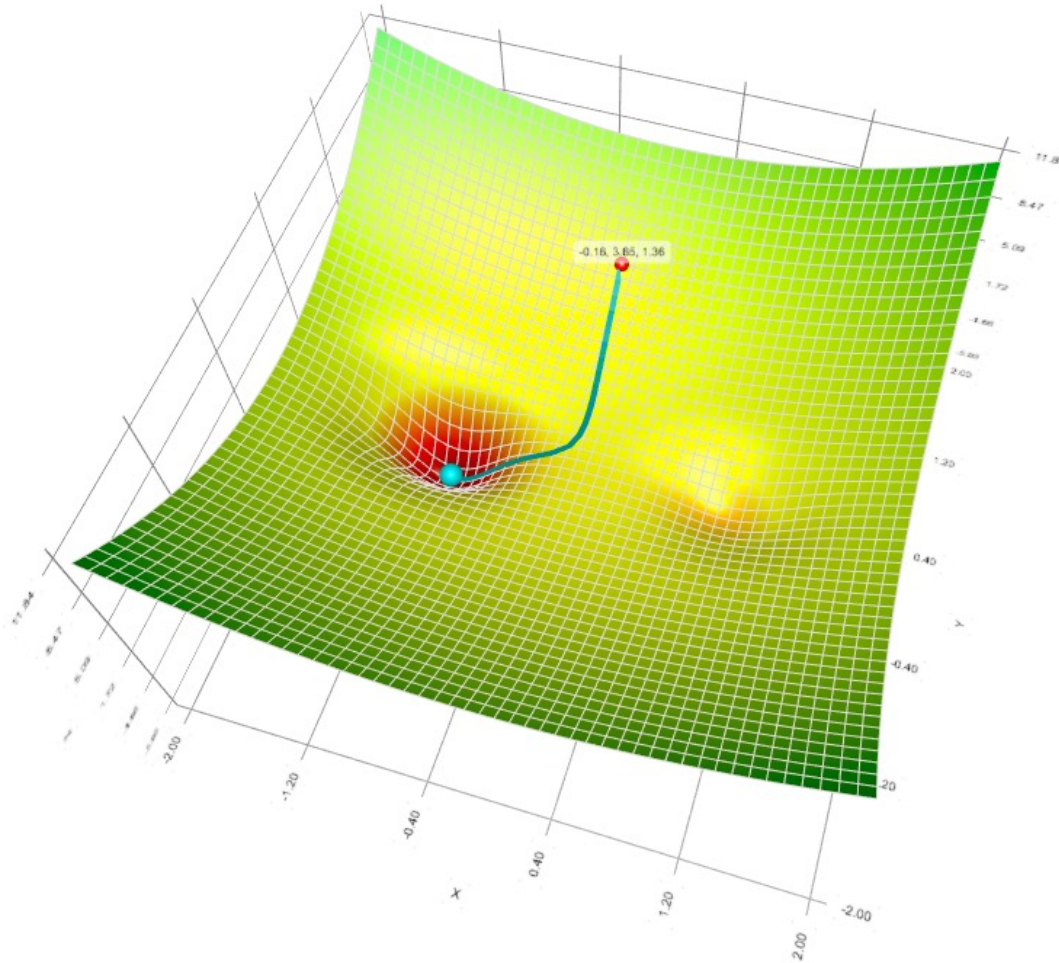
How do we optimize a continuously differentiable function in general?

- The problem: $\min_{\theta} f(\theta)$
- Let's just optimize it anyway!
 - With gradient descent.
- Assumption: The objective function is differentiable almost everywhere.

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$



Gradient Descent Demo

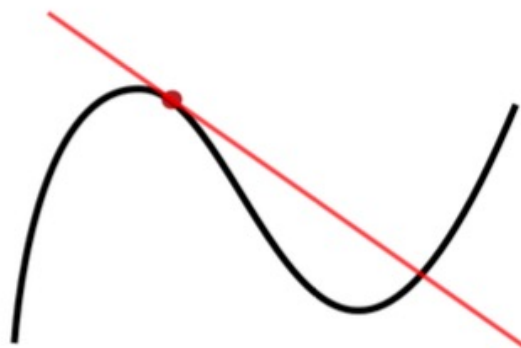


- Play with this excellent tool yourself to build intuition:

https://github.com/lilipads/gradient_descent_viz

What is the “gradient” of a (multivariate functions) function anyways?

- We use differentiation to compute derivatives of functions in Calculus.



- Example $f(x, y) = 3x^2 + xy$, $\frac{\partial f(x,y)}{\partial x} = 6x + y$, $\frac{\partial f(x,y)}{\partial y} = x$.
- In many machine learning problems, the objective involves a function that takes a vector of variables as input, e.g., $f(w) = w^T x$ where $w \in \mathbb{R}^d$.
- How to take derivatives on such functions?

Gradient is the concatenation of partial derivatives into a vector.
The shape of the vector is the same as that of the parameter vector (or matrix).

(1 min discussion) Gradient of a linear function

- What is the gradient of

$$f(x) = w^T x$$

Take out a piece of paper and work on it!

Gradient of logistic loss for learning a linear classifier

- The function to minimize is

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \log(1 + \exp(-y_i \cdot x_i^T w))$$

- How to calculate the gradient?
 - Take out a piece of paper and work on it!
 - (you have 3 min)

Hint:

- Apply the chain rule.
- $d \log(x) / dx = 1/x$
- $d \exp(x) / dx = \exp(x)$

Gradient of logistic loss for learning a linear classifier

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^n \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

- What is the time complexity of computing this gradient?

Can we do better?

Stochastic Gradient Descent (Robbins-Monro 1951)

- Gradient descent

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$

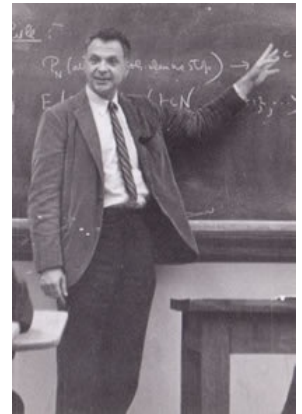
- Stochastic gradient descent

$$\theta_{t+1} = \theta_t - \eta_t \hat{\nabla} f(\theta_t)$$

- Using a stochastic approximation of the gradient:

$$\mathbb{E}[\hat{\nabla} f(\theta_t) | \theta_t] = \nabla f(\theta_t)$$

$$\mathbf{Var}[\hat{\nabla} f(\theta_t) | \theta_t] \leq \sigma^2$$



Herbert Robbins
1915 - 2001

One natural stochastic gradient to consider in machine learning

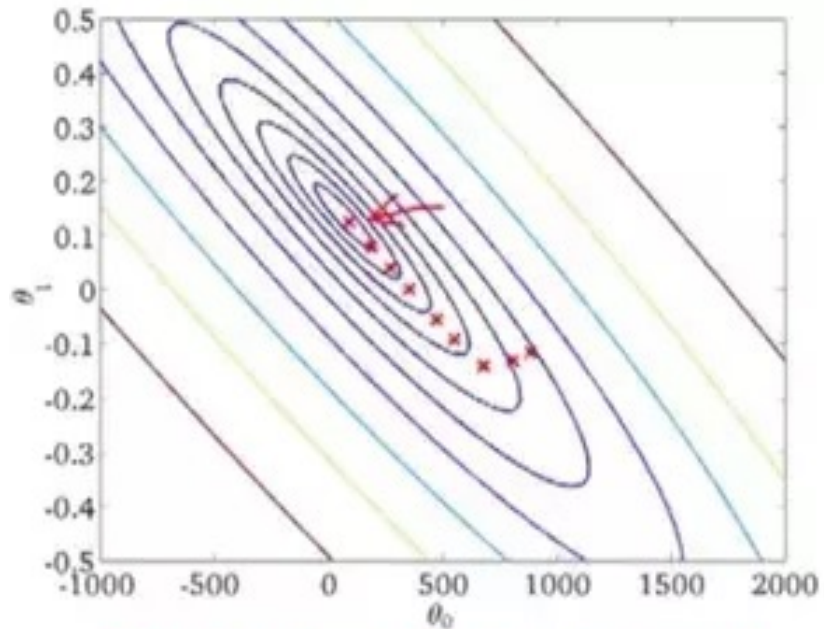
- Recall that
$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \ell(\theta, (x_i, y_i))$$

- Pick a **single** data point i uniformly at random

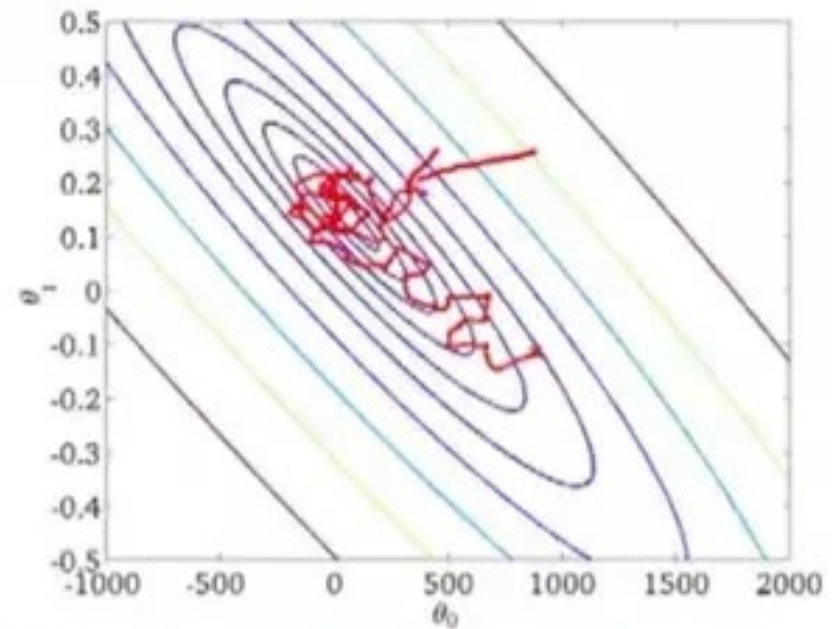
- Use
$$\nabla_{\theta} \ell(\theta, (x_i, y_i))$$

- What is the time complexity of SGD with this choice of “Stochastic Gradient”?
- Show that this is an unbiased estimator!

Illustration of GD vs SGD



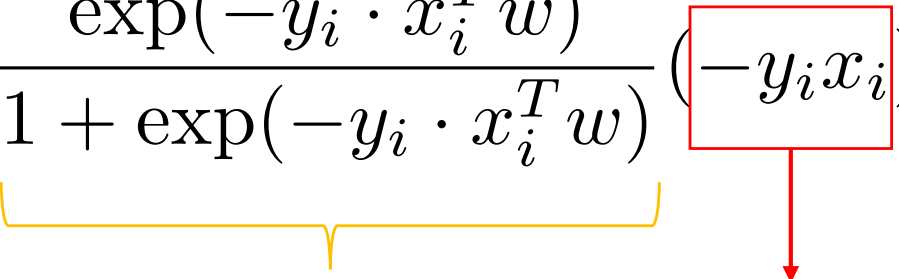
Batch Gradient Descent



Stochastic Gradient Descent

Observation: With the time gradient descent taking one step. SGD would have already moved many steps.

Intuition of the SGD algorithm on the “Spam Filter” example

$$\nabla \ell(w, (x_i, y_i)) = \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} \begin{pmatrix} -y_i x_i \end{pmatrix}$$


Scalar > 0 :
 ≈ 0 if the prediction
is correct
 ≈ 1 otherwise

Vector of dimension d :
provides the direction
of the gradient

If we receive an example $[1, 0, 0.0375, 80]$ like the one before.
And a label $y = 1$ saying that this is a spam.

How will the SGD update change the weight vector?

Then by moving w towards the negative gradient direction, we are changing the weight vector by increasing the weights. i.e., increasing the amount they contribute to the score function (if currently the classifier is making a mistake on this example)

Demo of GD and SGD in non-separable problems

- What do we observe about GD?
 - Stable and steady convergence to the optimal solution in hundreds of iterations. Each iteration is expensive.

- What do we observe about SGD?
 - Converges in a few data passes (and many more iterations than GD). But each iteration is a lot cheaper.
 - More sensitive to learning rate.. And jittering a bit.
 - A lot more stable than the Perceptron Algorithm.

How to choose the step sizes / learning rates?

- In theory:

- Gradient decent: $1/L$ where L is the **Gradient Lipschitz constant** of the function we minimize.

- SGD:
$$\sum_t \eta_t = \infty, \sum_t \eta_t^2 < \infty$$

- e.g.
$$\eta_t \in [1/t, 1/\sqrt{t})$$

- In practice:

- Use cross-validation on a subsample of the data.
- Fixed learning rate for SGD is usually fine.
- If it diverges, decrease the learning rate.
- If for extremely small learning rate, it still diverges, check if your gradient implementation is correct.

The power of SGD

- Extremely general:
 - Specify an end-to-end differentiable score function, e.g., a complex neural network.
 - Beyond the context of machine learning
- Extremely simple: a few lines of code.
- Extremely scalable
 - Just a few pass of the data, no need to store the data
- People are continuing to discover that many methods are special cases of SGD.

Perceptron Algorithm can be viewed as a variant of SGD

- It uses a shifted version of “Hinge loss” --- aka “Perceptron loss”
- **Exercise (high chance to be in midterm):** Calculate the gradient of the “Perceptron loss”. Show that the update rule after a single data point is selected is the same as the Perceptron’s algorithm when learning rate = 1.
- Important differences to SGD
 1. Perceptron does not require the update direction to be an “unbiased” estimate. It could go with the iid sampling version or shuffle-then-“for”-over version.
 2. It does not require a choice of learning rate.
 3. It does not aim at optimizing an objective function (thus does not necessarily succeed in doing so).

Summary of the lecture

- Learning a linear classifier:
 - Surrogate losses and linear logistic regression
- Gradient descent
 - Calculating gradient / making sense of gradient
 - Improving GD with Stochastic Gradient Descent
- Next Lecture:
 - Linear regression

Don't forget to turn-in your HW1 and MP1!