COMP4901K/Math4824B
Machine Learning for Natural Language Processing

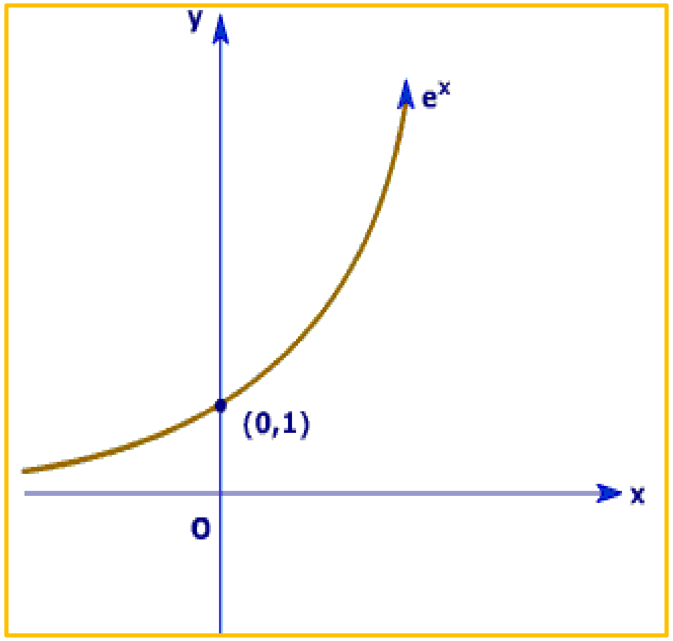
Lecture 9: Logistic Regression

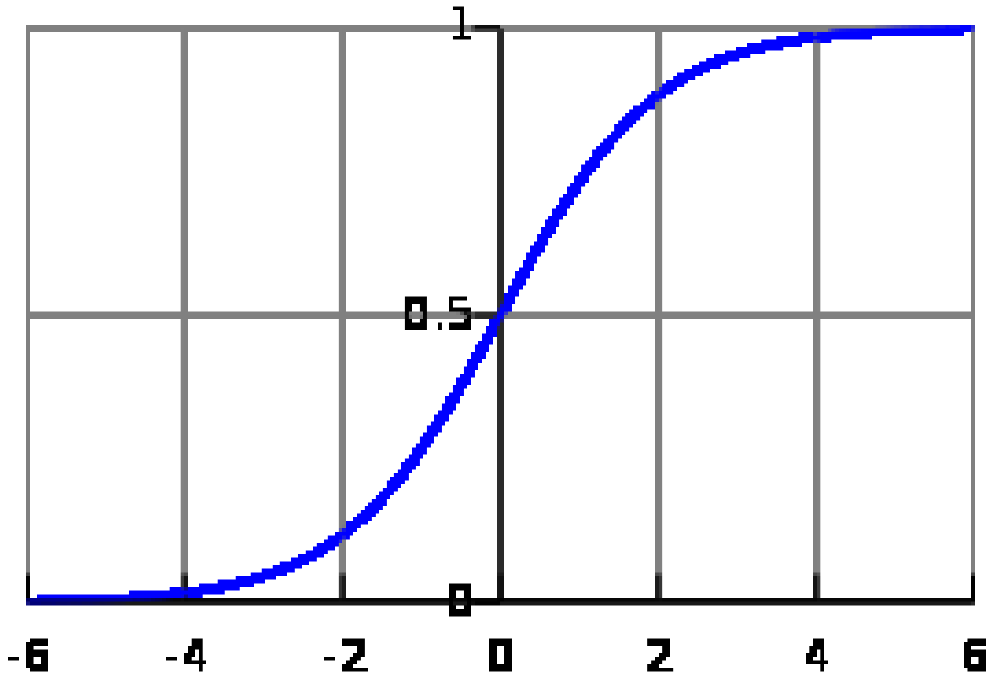
Instructor: Yangqiu Song

Slides credits: John Canny

Revisit Perceptron Prediction

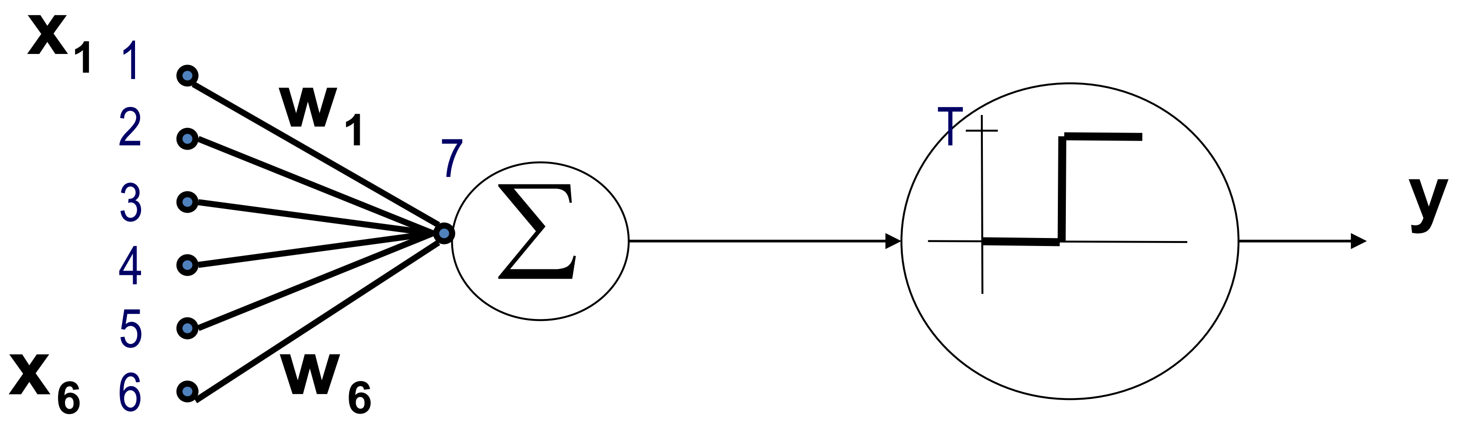
$$\mathbf{w}^{\mathrm{T}}\mathbf{x} > 0 \text{ is equivalent to } \frac{1}{1+\exp\{-(\mathbf{w}^{\mathrm{T}}\mathbf{x})\}} > 1/2$$



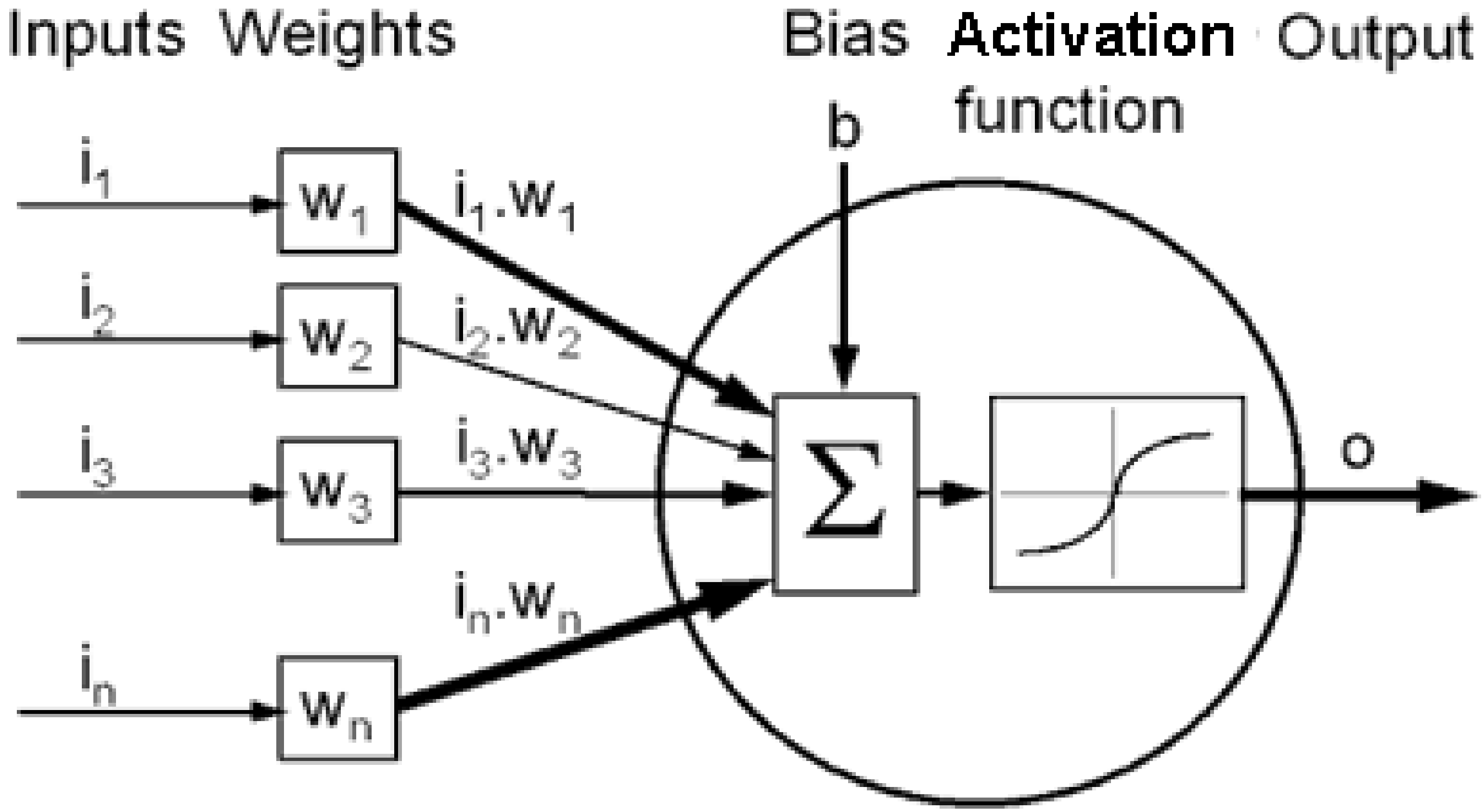


Perceptron learning rule

- On-line, mistake driven algorithm.
- Rosenblatt (1959) suggested that when a target output value is provided for a single neuron with fixed input, it can incrementally change weights and learn to produce the output using the Perceptron learning rule
- (Perceptron == Linear Threshold Unit)



What if I directly optimize the output probability?



Logistic Regression

- Logistic regression is designed as a **binary classifier** (output say {0,1} or {-1,1}) but actually **outputs the probability** that the input instance is in the "1" class.

- A logistic classifier has the form:

$$p(y=1|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}^\top \boldsymbol{x})}$$

where $\boldsymbol{x} = \left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(d)}\right)$ is a vector of features.

Training

For training, we start with a collection of input values $\boldsymbol{x}_i$ and corresponding output labels $y_i \in \{-1,1\}$. Let p_i be the predicted output on input $\boldsymbol{x}_i$, so

$$p_i = \frac{1}{1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)}$$

Logistic regression minimize the loss: negative sum of the log accuracy (the total accuracy), i.e.,

$$f(\boldsymbol{w}) = -\sum_{i=1}^{N} \log p_i = \sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))$$

Logistic Regression

- Logistic regression is probably the most widely used general-purpose classifier.

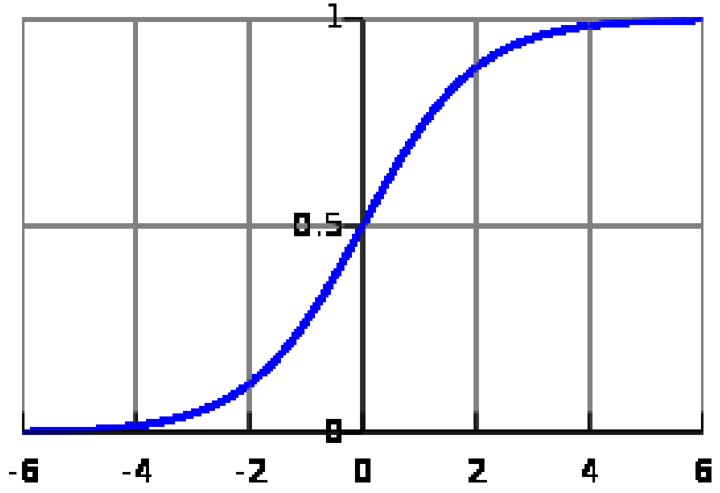
- Its very scalable and can be very fast to train. It’s used for
  - Spam filtering
  - News message classification
  - Web site classification
  - Most classification problems with large, sparse feature sets.

- The only caveat is that it can overfit on very sparse data, so its often used with Regularization

Logistic Regression

- Logistic regression maps the "regression" value $-\boldsymbol{w}^\top \boldsymbol{x}$ in (-∞,∞) to the range [0,1] using a "logistic" function:

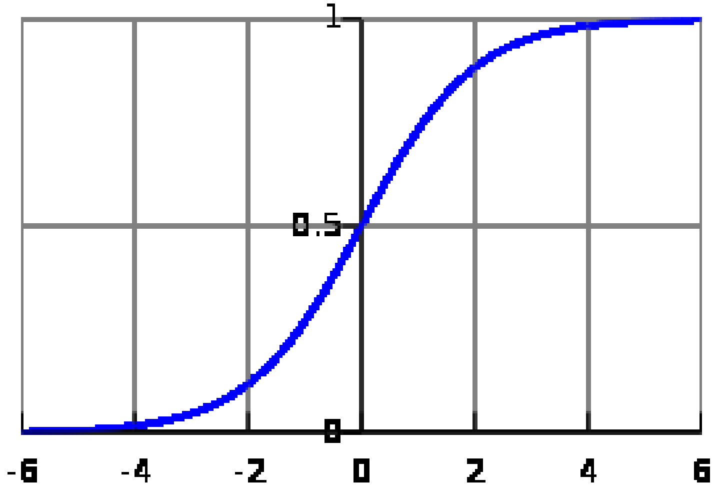
$$p(y = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})}$$



- i.e. the logistic function maps any value on the real line to a probability in the range [0,1]

Logistic Regression

- Where did the logistic function come from?



Logistic Regression and Naïve Bayes

- Logistic regression is actually **a generalization of Naïve Bayes with binary features**.
- Logistic Regression can model a Naïve Bayes classifier when the binary features are **independent**.
- Bayes rule for two classes c and $\neg c$:

$$\mathrm{P}(c|\boldsymbol{x}) = \frac{\mathrm{P}(\boldsymbol{x}|c)\,\mathrm{P}(c)}{\mathrm{P}(\boldsymbol{x})} = \frac{\mathrm{P}(\boldsymbol{x}|c)\,\mathrm{P}(c)}{\mathrm{P}(\boldsymbol{x}|c)\,\mathrm{Pr}(c) + \mathrm{P}(\boldsymbol{x}|\neg c)\,\mathrm{P}(\neg c)}$$

- Dividing by the numerator:

$$= \frac{1}{1 + \frac{\mathrm{P}(\boldsymbol{x}|\neg c)\,\mathrm{P}(\neg c)}{\mathrm{P}(\boldsymbol{x}|c)\,\mathrm{P}(c)}}$$

Logistic Regression and Naïve Bayes

We have

$$\mathrm{P}(c|X) = \frac{1}{1 + \frac{\mathrm{P}(X|\neg c)\,\mathrm{Pr}(\neg c)}{\mathrm{P}(X|c)\,\mathrm{Pr}(c)}} = \frac{1}{1 + \exp(-\boldsymbol{w}^{\top}\boldsymbol{x})}$$

which matches if

$$\frac{\mathrm{P}(X|\neg c)\,\mathrm{P}(\neg c)}{\mathrm{P}(X|c)\,\mathrm{P}(c)} = \exp(-\boldsymbol{w}^{\top}\boldsymbol{x})$$

and assuming feature independence, the two sides factor as:

$$\frac{\mathrm{P}(\neg c)}{\mathrm{P}(c)} \prod_{j=1}^{d} \frac{\mathrm{P}\left(\boldsymbol{x}^{(j)}|\neg c\right)}{\mathrm{P}\left(\boldsymbol{x}^{(j)}|c\right)} = \frac{\pi_{\neg c}}{\pi_c} \prod_{j=1}^{d} \frac{\theta_{\neg c,j}^{\boldsymbol{x}^{(j)}}}{\theta_{c,j}^{\boldsymbol{x}^{(j)}}}$$

$$= \exp\log\frac{\pi_{\neg c}}{\pi_c} \exp\left(\sum_j \log\frac{\theta_{\neg c,j}^{\boldsymbol{x}^{(j)}}}{\theta_{c,j}^{\boldsymbol{x}^{(j)}}}\right)$$

$$e^{x+y} = e^x e^y$$
$$\log xy = \log x + \log y$$
$$e^{\log x} = x$$

$$= \exp\log\frac{\pi_{\neg c}}{\pi_c} \exp\left(\sum_j \boldsymbol{x}^{(j)}(\log\theta_{\neg c,j} - \log\theta_{c,j})\right) = \exp(-w_0) \prod_{i=1}^{d} \exp\left(-\boldsymbol{x}^{(j)}\boldsymbol{w}^{(j)}\right)$$

And we can match corresponding (j)th terms to define $\boldsymbol{w}$.

Logistic Regression and Naïve Bayes

Summary: Logistic regression has this form:

Models Naïve Bayes formula with two classes
after dividing through by one of them

$$\mathrm{P}(c|X) = \frac{1}{1 + \exp(-\boldsymbol{w}^{\top}\boldsymbol{x})}$$

Models product of contributions
from different (independent) features

Results on UCI datasets (Ng and Jordan (2001))

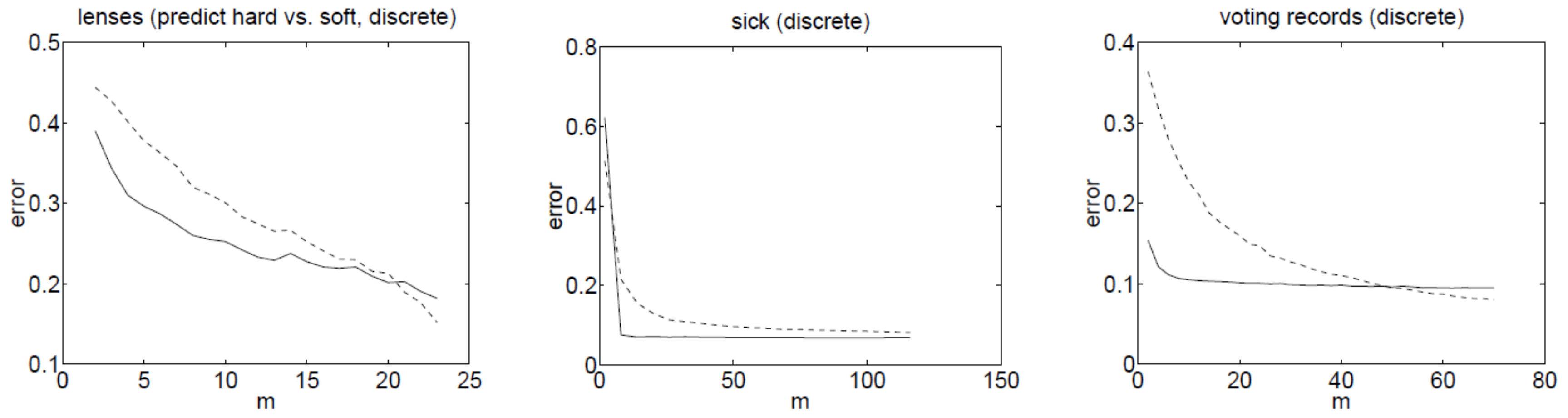
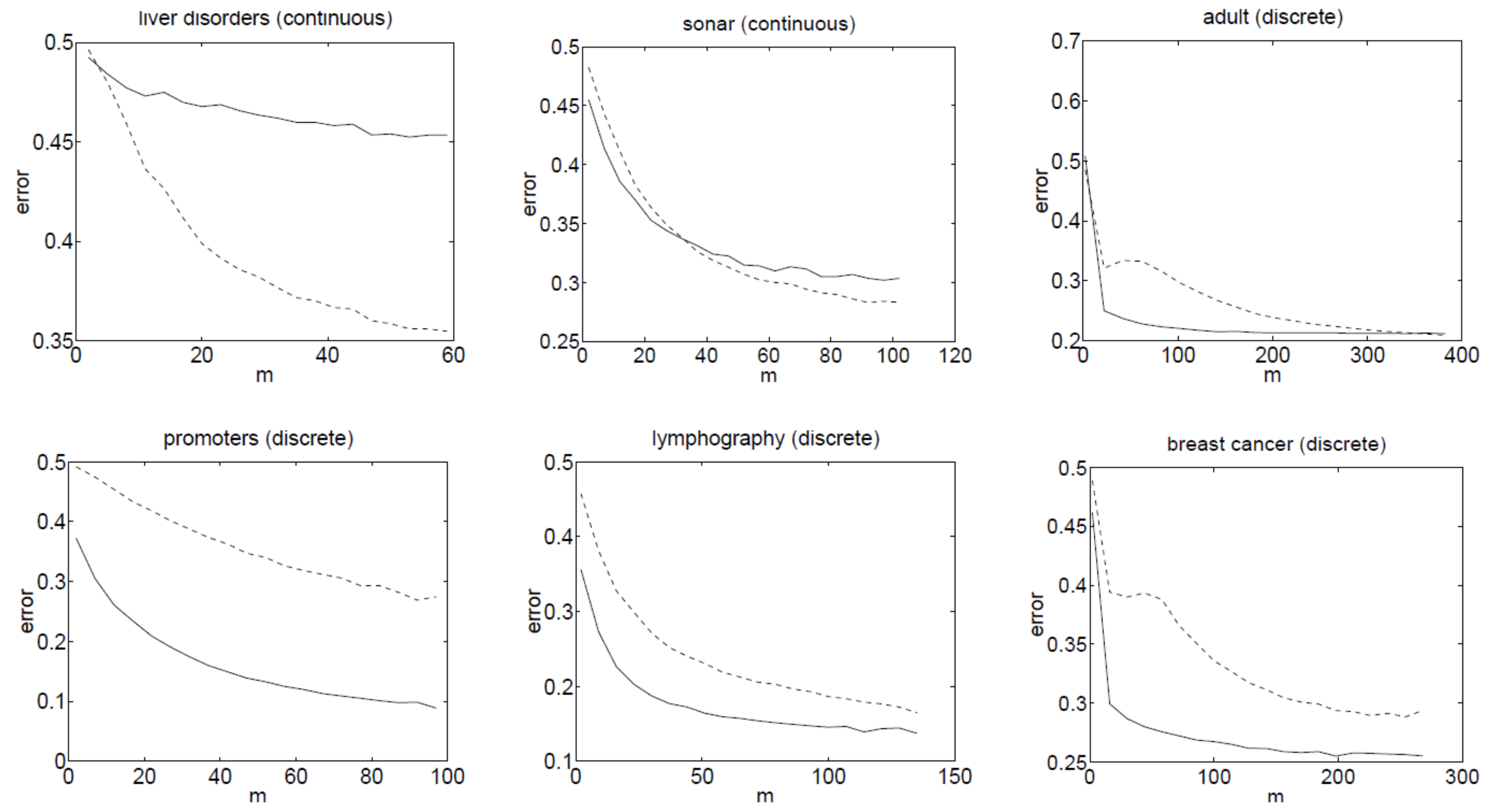
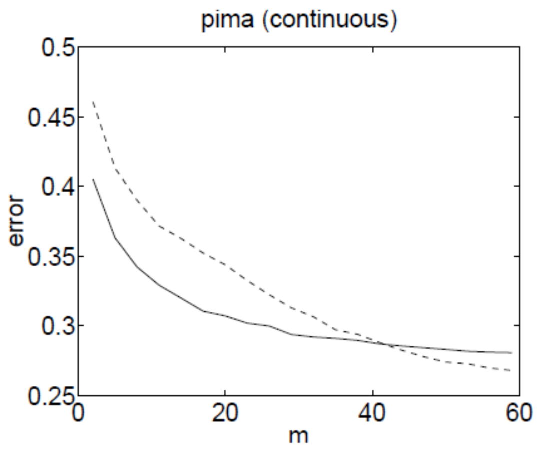


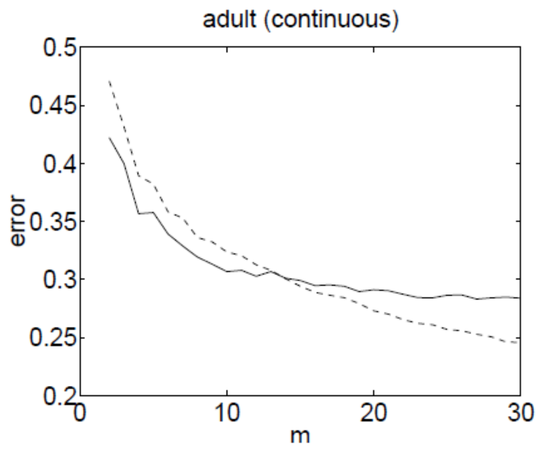
Figure 1: Results of 15 experiments on datasets from the UCI Machine Learning repository. Plots are of generalization error vs. m (averaged over 1000 random train/test splits). Dashed line is logistic regression; solid line is naive Bayes.

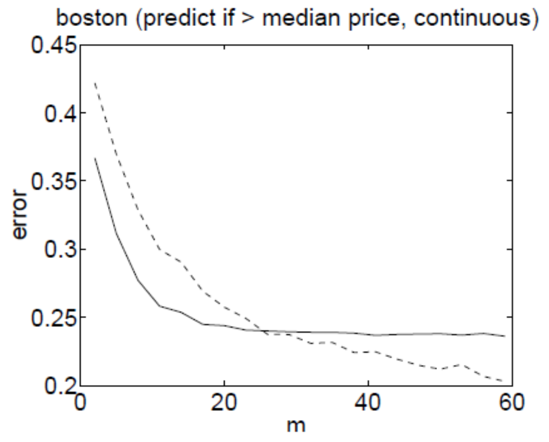
Results on UCI datasets (Ng and Jordan (2001))

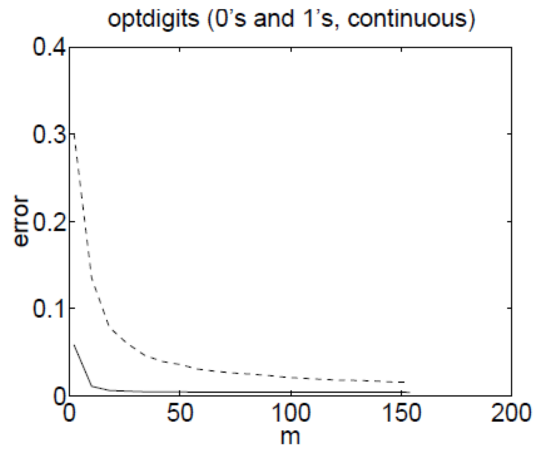


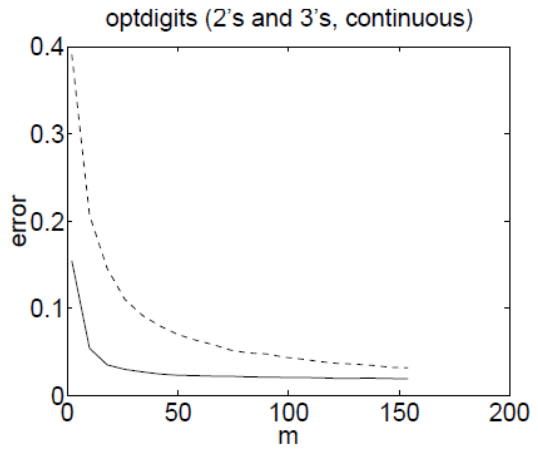
Results on UCI datasets (Ng and Jordan (2001))

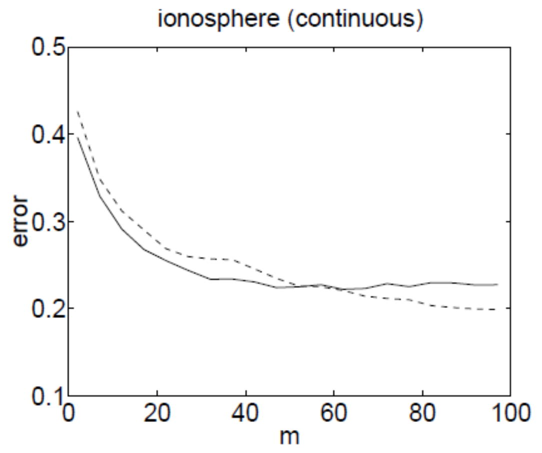












Outline

- Logistic Regression
- Gradient-based optimization

Training

For training, we start with a collection of input values $\boldsymbol{x}_i$ and corresponding output labels $y_i \in \{-1,1\}$. Let p_i be the predicted output on input $\boldsymbol{x}_i$, so

$$p_i = \frac{1}{1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)}$$

Logistic regression minimize the loss: negative sum of the log accuracy (the total accuracy), i.e.,

$$f(\boldsymbol{w}) = -\sum_{i=1}^{N} \log p_i = \sum_{i=1}^{N} \log(1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))$$

Gradient based methods

Our goal is to minimize $f(\boldsymbol{w})$

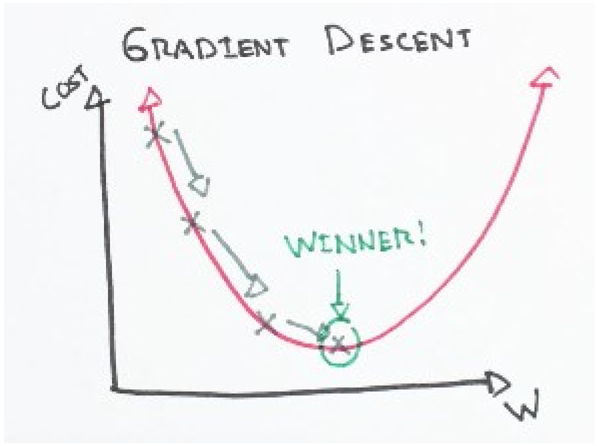
Gradient based method use

$$\boldsymbol{w}' = \boldsymbol{w} - \alpha\, \boldsymbol{d}$$

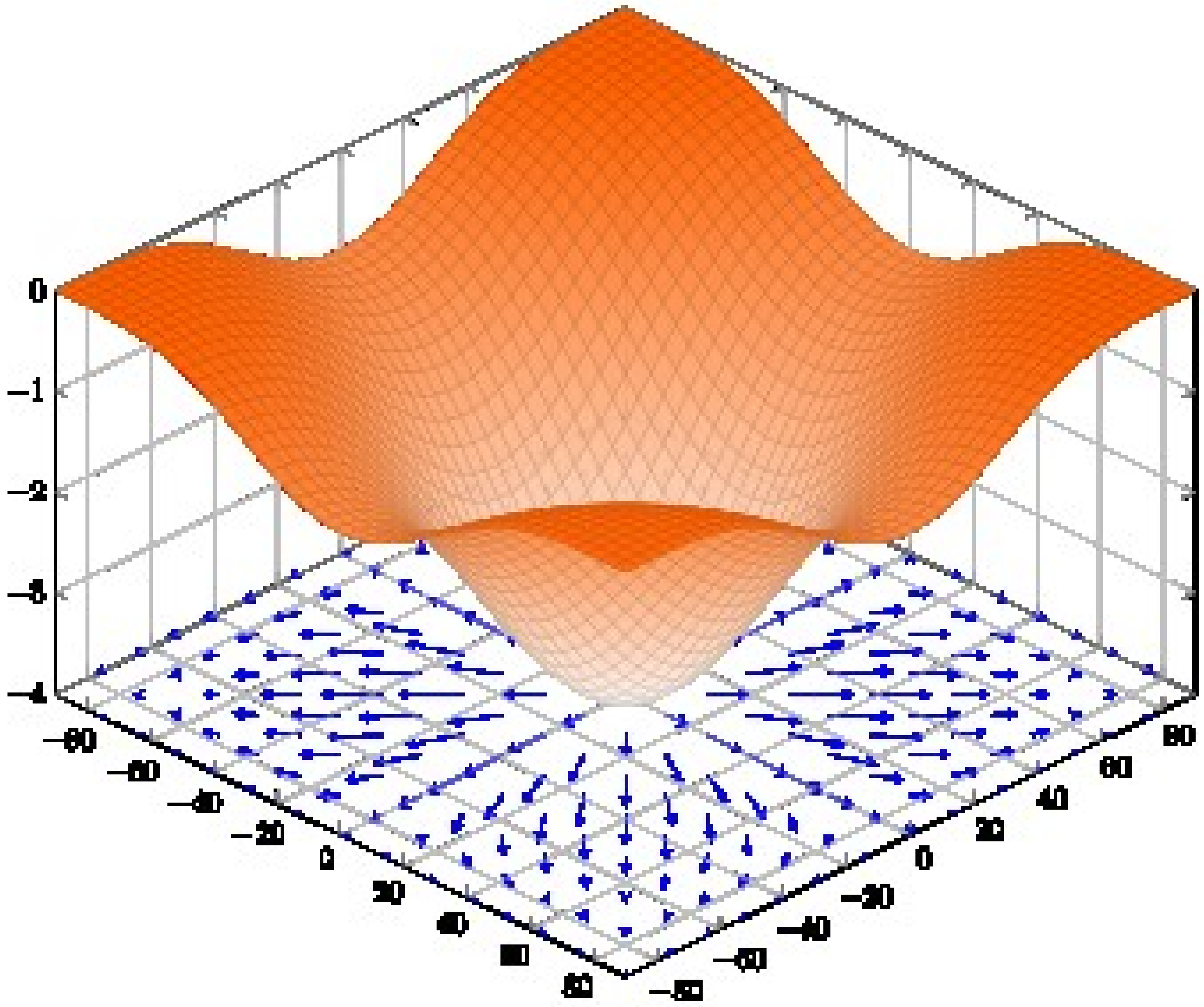
to update the weight vector where

$$\boldsymbol{d} \propto \nabla f(\boldsymbol{w})$$

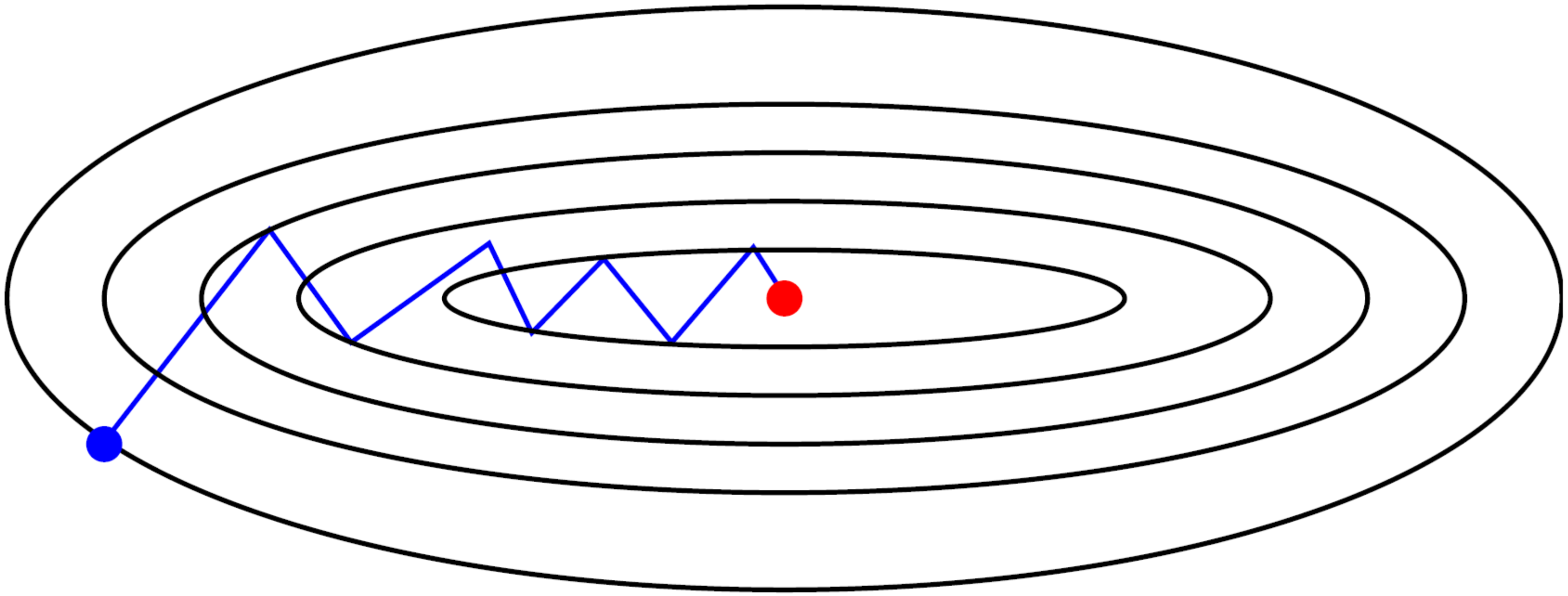
$\nabla f(\boldsymbol{w}) = \left[\frac{df}{d\boldsymbol{w}^{(1)}}, \ldots, \frac{df}{d\boldsymbol{w}^{(d)}}\right]^T$ is the gradient



The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.



Gradient based methods



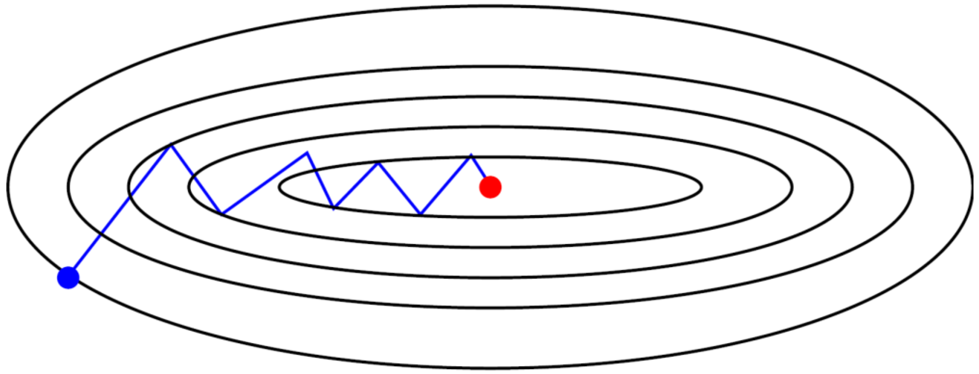
Incremental Training - SGD

- A very efficient way to train logistic models is with Stochastic Gradient Descent (SGD) – we keep updating the model with gradients one example

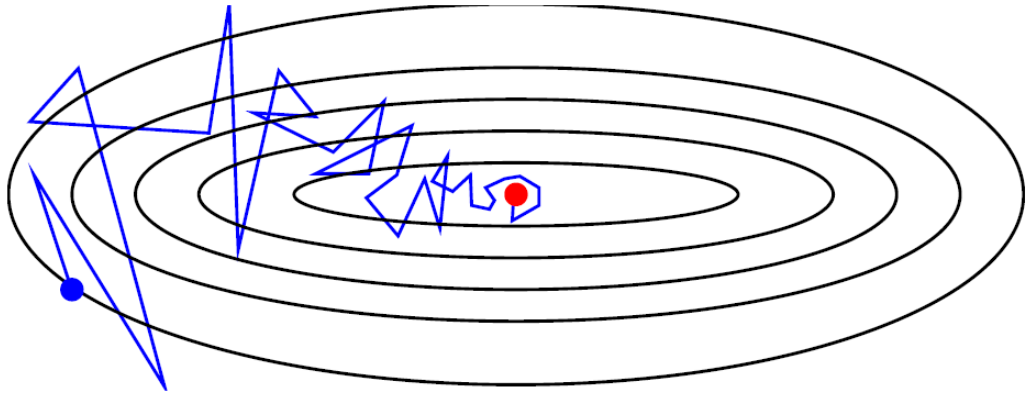
- One challenge with training on power law data (i.e. most data) is that the terms in the gradient can have very different strengths (because of the power law distribution). We'll discuss this soon…

Deterministic Gradient vs Stochastic Gradient

- Deterministic Gradient method



- Stochastic Gradient method



Stochastic Gradient for Logistic Regression

- Logistic regression minimize the negative sum of the log accuracy (the total accuracy),

$$f(\boldsymbol{w}) = -\sum_{i=1}^{N} \log p_i = \sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))$$

- Taking one example $\boldsymbol{x}_i$, computing the gradient of $f_i(\boldsymbol{w})$:

$$\begin{aligned}\nabla f_i(\boldsymbol{w}) &= \nabla \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) \\ &= \frac{1}{1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)} \nabla(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) \\ &= \frac{1}{1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)} \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)\,(-y_i \boldsymbol{x}_i)\end{aligned}$$

- SGD becomes

$$\boldsymbol{w}' = \boldsymbol{w} - \alpha \nabla f_i(\boldsymbol{w}) = \boldsymbol{w} + \alpha \frac{1}{1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)} \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)\,(y_i \boldsymbol{x}_i)$$

Where

$$\frac{1}{1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)} \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i) = \frac{1}{1 + \exp(y_i \boldsymbol{w}^\top \boldsymbol{x}_i)} = 1 - \frac{1}{1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)}$$

is the probability of $\boldsymbol{x}_i$ being not predicted as y_i

Compared to Perceptron

- For perceptron:
  - if $y_i' != y_i$, update the weight vector

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + a\, y_i \boldsymbol{x}_i$$

- For logistic regression:

$$\boldsymbol{w}' = \boldsymbol{w} - \alpha\, \boldsymbol{d} = \boldsymbol{w} + \alpha P(y_i' != y_i)(y_i \boldsymbol{x}_i)$$

Stochastic Gradient with Mini-Batch

A very important set of iterative algorithms use stochastic gradient updates.

They use a small subset or mini-batch of the data, and use it to compute a gradient which is added to the model

$$\boldsymbol{w}' = \boldsymbol{w} - \alpha\, \boldsymbol{d}$$

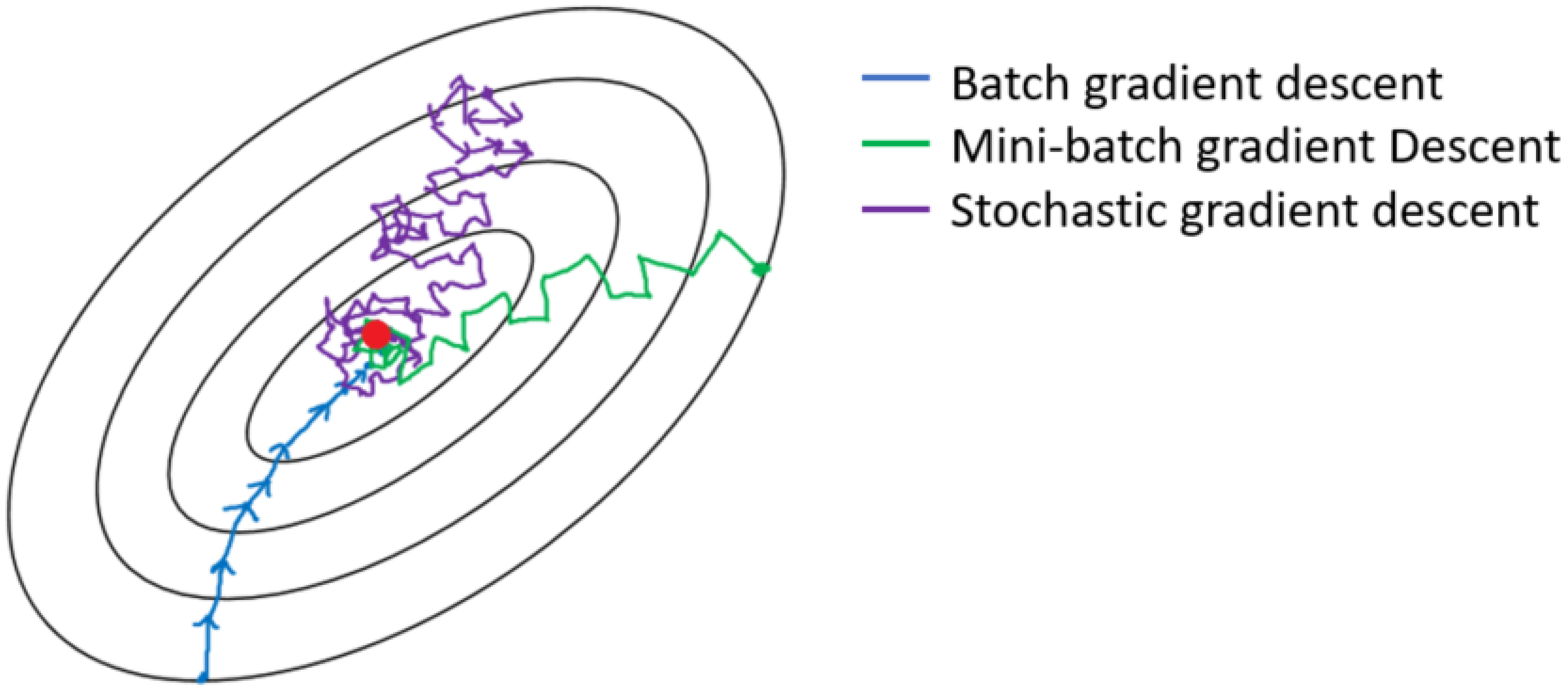
$$\boldsymbol{d} \propto \sum_{i \in B} \nabla f_i(\boldsymbol{w})$$

where α is the learning rate.

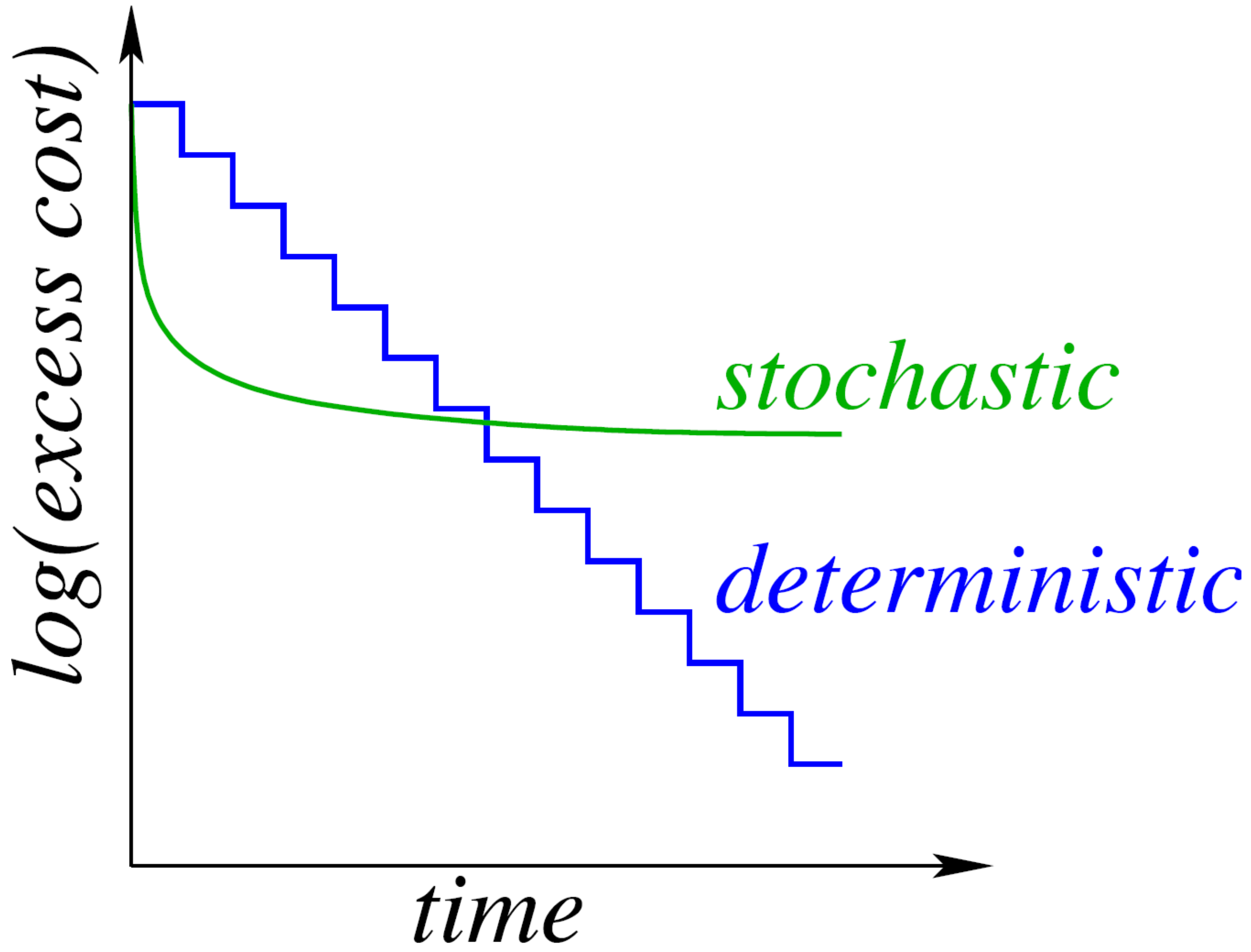
These updates happen many times in one pass over the dataset.

It's possible to compute high-quality models with very few passes, sometime with less than one pass over a large dataset.

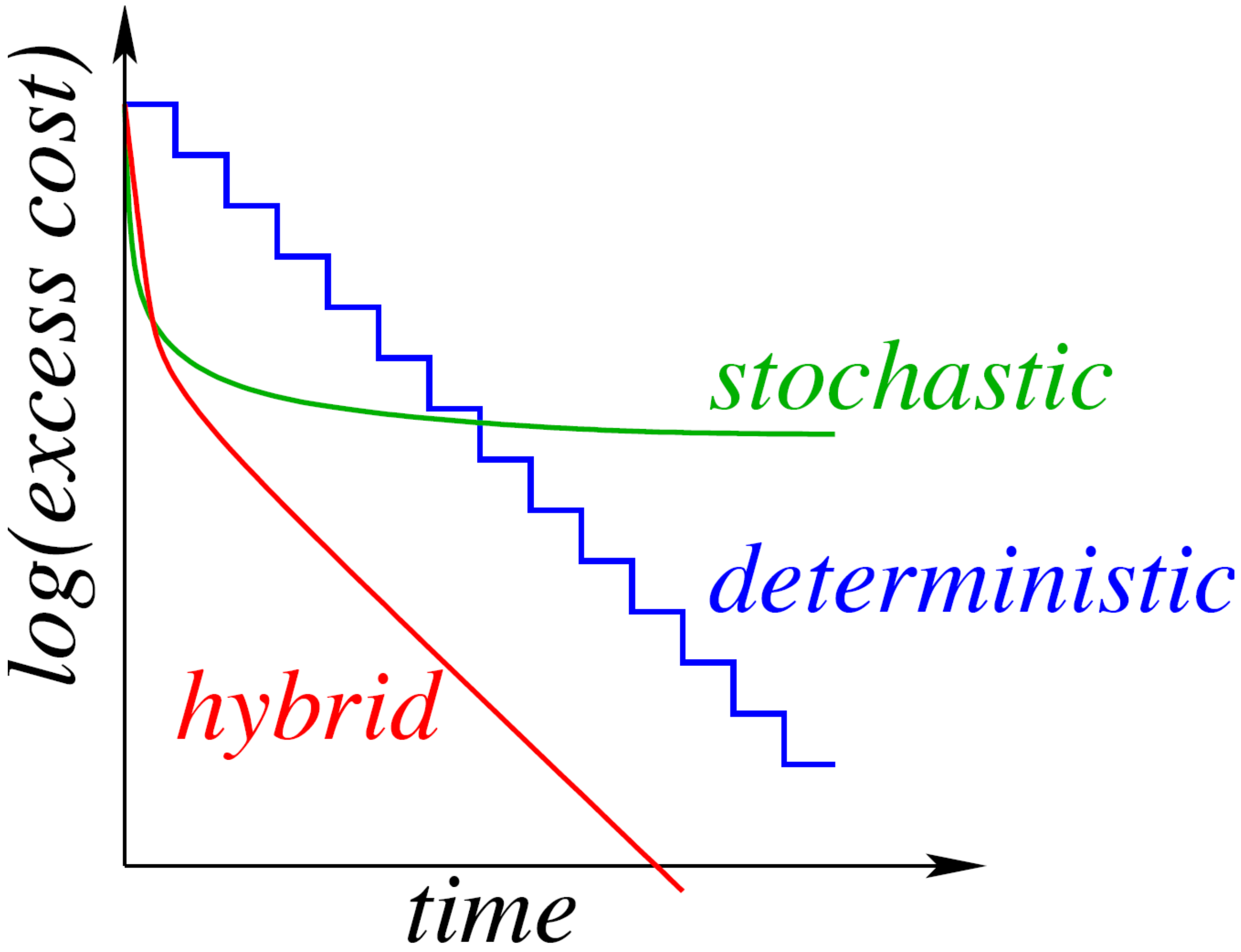
Stochastic Gradient with Mini-Batch



Mini-Batch Methods



Mini-Batch Methods



Challenges for Stochastic Gradient

- Stochastic gradient has some serious limitations however, especially if the **gradients vary widely in magnitude**. Some coefficients change very fast, others very slowly.

- This happens for **text, user activity and social media data** (and other power-law data), because gradient magnitudes scale with feature frequency, i.e. over several orders of magnitude.

- It's not possible to set a single learning rate that trains the frequent and infrequent features at the same time.

ADAGRAD – Adaptive-rate SGD

- ADAGRAD is a particularly simple and fast approach to this problem. Let $\mathbf{g}^t = \frac{1}{|\mathcal{B}^t|}\sum_{i\in\mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t)$
- ADAGRAD updates weights as

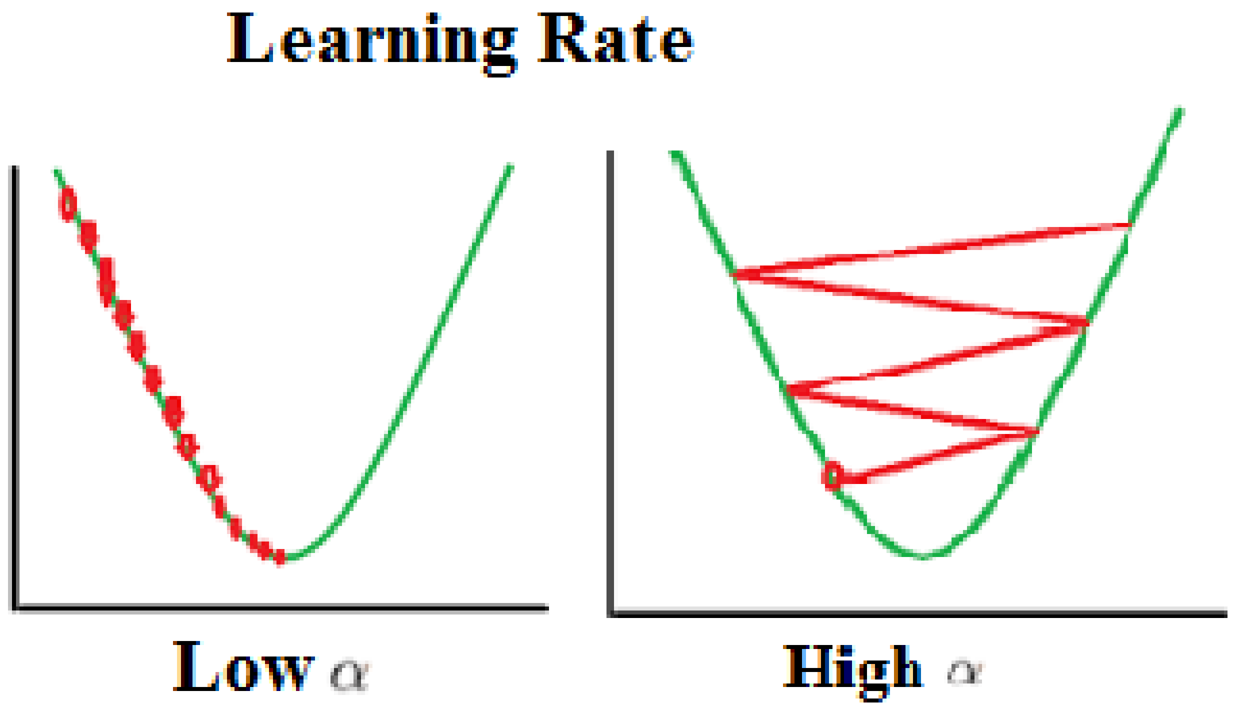
$$\mathbf{r}^{t+1} = \mathbf{r}^t + \mathbf{g}^t \odot \mathbf{g}^t$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta\frac{1}{\delta+\sqrt{\mathbf{r}^t}} \odot \mathbf{g}^t$$

Division and square root applied element-wise

- This corrects for feature scale factors, and all ADAGRAD-scaled gradient components have similar magnitudes.

- ADAGRAD often improves SGD convergence on power-law data by orders of magnitude.

Learning Rate



SGD learning rate schedules

For harder (non-convex) learning problems, its common to use learning rate schedules that decay more slowly, or not at all.

Exponential schedule:$\alpha = \alpha_0 \exp(-{}^{t}/_{\tau})$ starting with an initial learning rate α_0, the rate decreases by a factor of e for each interval of learning τ.

Constant schedule:$\alpha = \ \alpha_0$ is good for very complex learning problems.

Linear schedule:$\alpha = \ \alpha_0 \left(1 - {}^{t}/_{T_{final}}\right)$ where T_{final} is the final value of t. Uses a fast initial rate to move close to the final optimum, and then slows to reduce variance.

Regularization

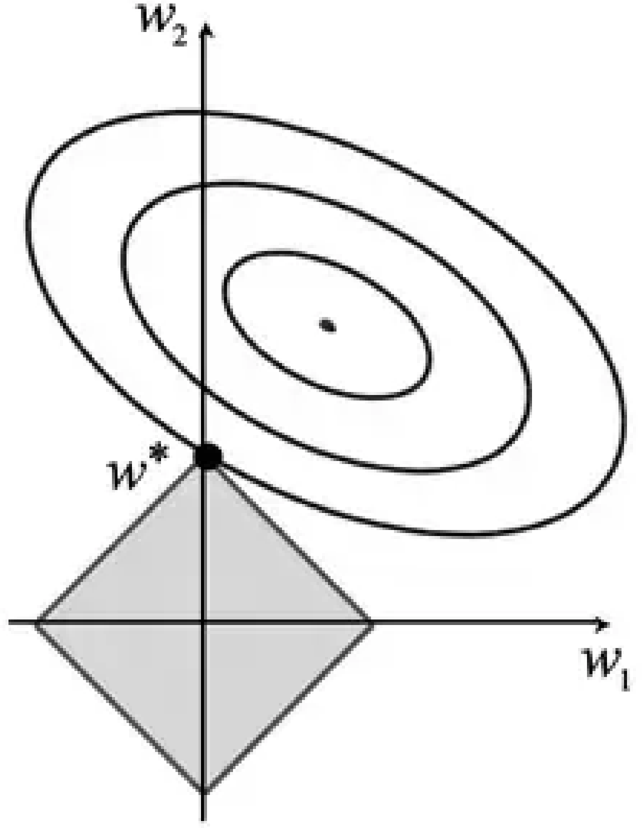
- To avoid the learned weights to be too large, we can put regularization terms to the loss function

$$f(\boldsymbol{w}) = \sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) + \frac{\lambda}{2} \left|\left|\boldsymbol{w}\right|\right|_2^2$$

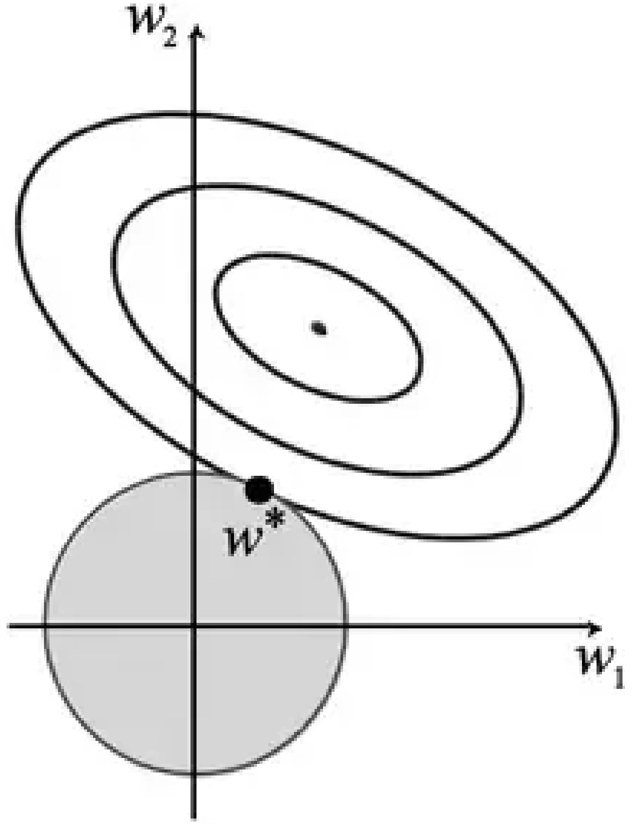
Or

$$f(\boldsymbol{w}) = \sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) + \lambda \left|\left|\boldsymbol{w}\right|\right|_1$$

Regularization: Geometric Interpretation



L1



L2

Regularization

The new loss

$$f(\boldsymbol{w}) = \sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)) + \frac{\lambda}{2} ||\boldsymbol{w}||_2^2$$

SGD optimization:

$$\boldsymbol{w}' = \boldsymbol{w} - \alpha\, \boldsymbol{d} = \boldsymbol{w} + \alpha(P(y_i' != y_i)(y_i \boldsymbol{x}_i) - \lambda \boldsymbol{w})$$

The L2 parameter norm penalty commonly is known as weight decay

More Reading

- Kevin Clark. Computing Neural Network Gradients
  - http://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf
- Daniel Jurafsky and James H. Martin. Logistic regression (Jurafsky and Martin (2017))
  - https://web.stanford.edu/~jurafsky/slp3/5.pdf

- More about optimization
  - I added some slides for PG on canvas
  - Only for whom are really interested in machine learning optimization
  - Will not include in final exams

- We may come back to related concepts when introducing NN and deep learning algorithms