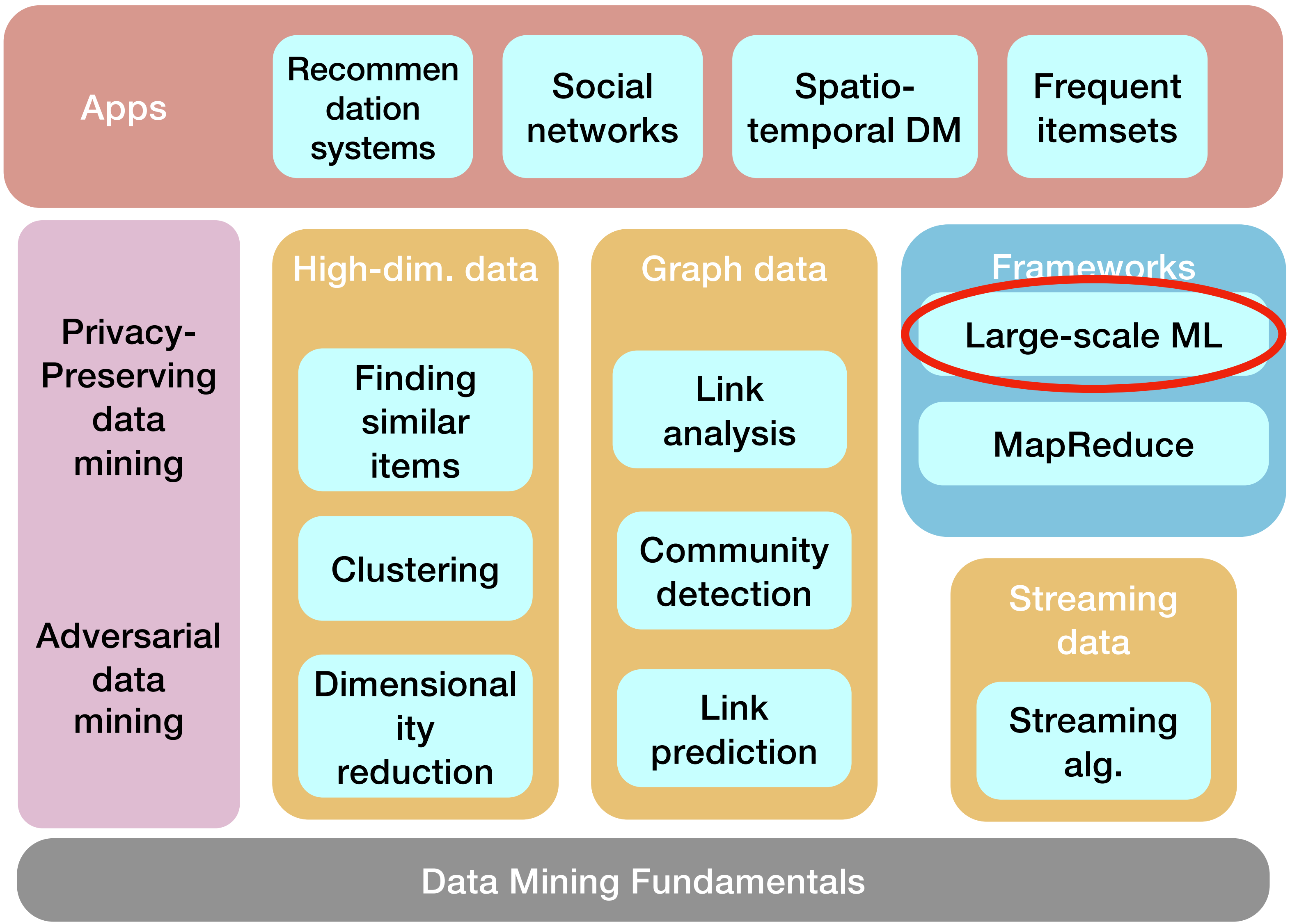
EE359 Data Mining Lecture 10

Large-Scale Machine Learning

Liyao Xiang
http://xiangliyao.cn/
Shanghai Jiao Tong University

Course Landscape



Outline

- Motivation
- Support Vector Machines
    - Optimal Margin Classifier
    - Slack Penalty
    - SGD SVM
- Distributed Deep Learning

Supervised Learning

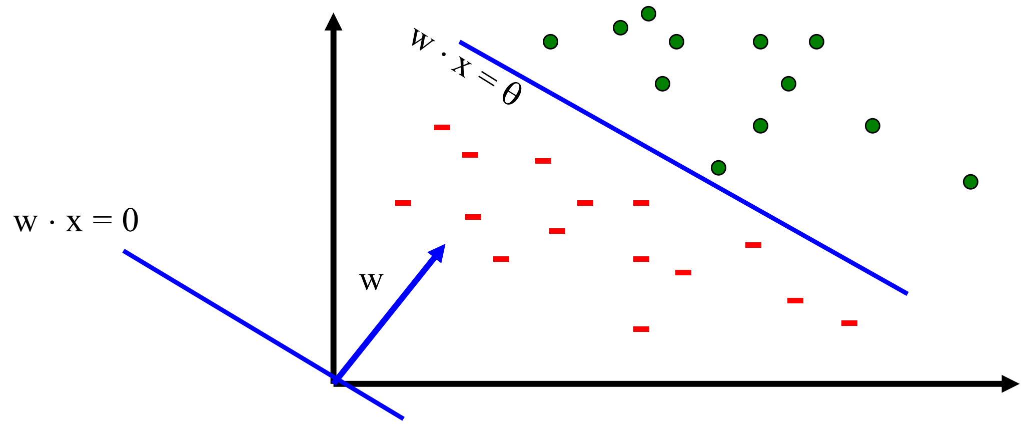
- Example: Spam filtering

	viagra	learning	the	dating	nigeria	*spam*?
$\vec{x}_1 = ($	1	0	1	0	0 $)$	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0 $)$	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1 $)$	$y_3 = 1$

- Instance space $x \in X$ ($|X|$= n data points)
  - Binary or real-valued feature vector x of word occurrences
  - d features (words + other things, d~100,000)
- Class $y \in Y$
  - y: Spam (+1), Not-Spam (-1)
- Goal: Estimate a function f(x) so that y = f(x)

Linear models for classification

- Binary classification: $f(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad \mathbf{w}^{(1)}\mathbf{x}^{(1)} + \mathbf{w}^{(2)}\mathbf{x}^{(2)} + \ldots \mathbf{w}^{(d)}\mathbf{x}^{(d)} \geq \theta \\ -1 & \text{otherwise} \end{cases}$
- Input: Vectors $\boldsymbol{x_j}$ and labels $\boldsymbol{y_j}$
  - Vectors $\boldsymbol{x_j}$ are real valued where $\|\boldsymbol{x}\|_2 = 1$
- Goal: Find vector w = (w(1), w(2) ,... , w(d))
- Each w(i) is a real number



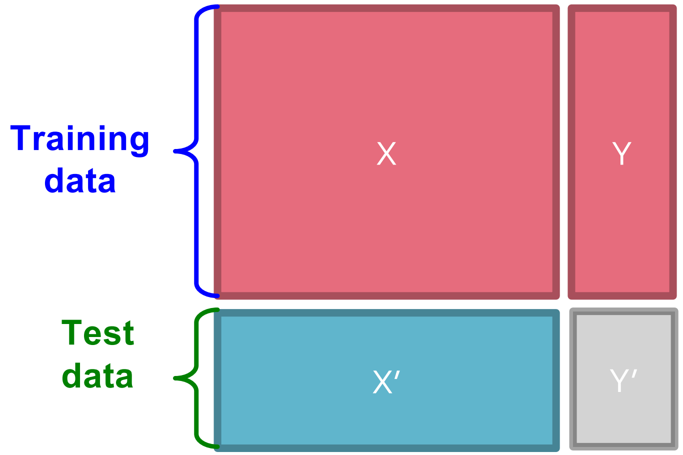
Decision boundary is linear

Supervised Learning

- Would like to do prediction: **estimate** a function f(x) so that y = f(x)
- Where y can be:
  - Real number: Regression
  - Categorical: Classification
  - Complex object:
    - Ranking of items, Parse tree, etc.
- Data is labeled:
  - Have many pairs {(x, y)}
    - x … vector of binary, categorical, real valued features
    - y … class ({+1, -1}, or a real number)

Supervised Learning

- Task: Given data (X,Y) build a model f() to predict Y' based on X'
- Strategy: Estimate $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ on $(\boldsymbol{X}, \boldsymbol{Y})$.
  Hope that the same also works to predict unknown Y'
  - The "hope" is called generalization
    - **Overfitting**: If f(x) predicts well Y but is unable to predict Y'
  - We want to build a model that generalizes well to unseen data
    - But how can we predict well on data we have never seen before?

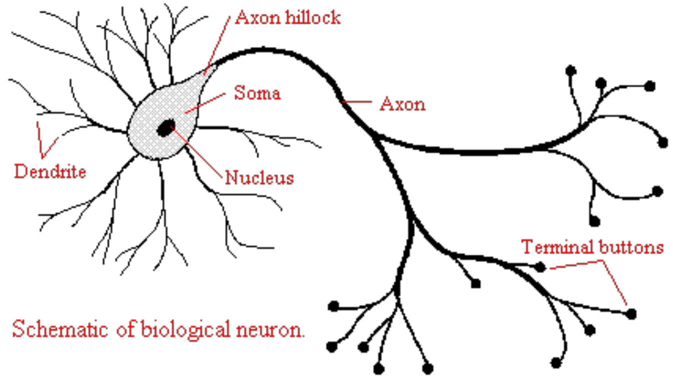


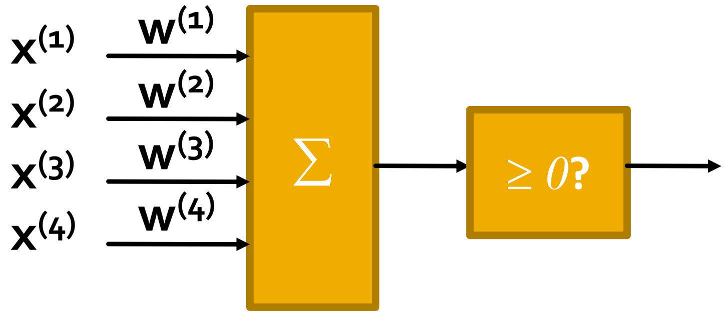
Perceptron [Rosenblatt '58]

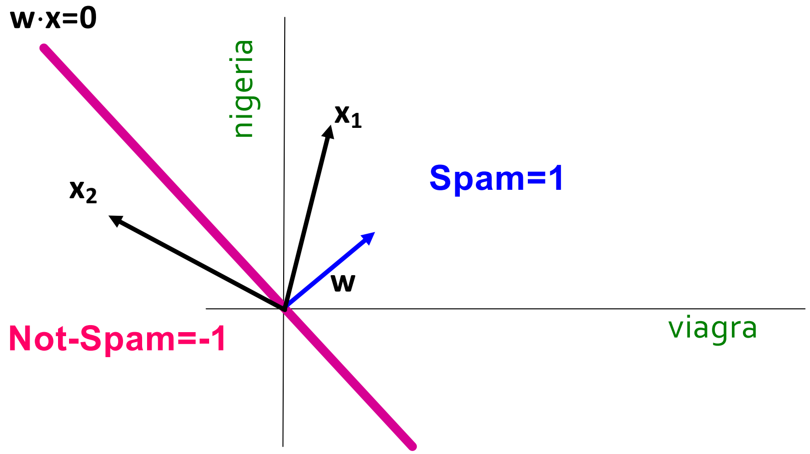
- (Very) loose motivation: Neuron
- Inputs are feature values
- Each feature has a weight $\boldsymbol{w_i}$
- Activation is the sum:

$$\boldsymbol{f(x) = \sum_i^d w^{(i)} x^{(i)} = w \cdot x}$$

- If the f(x) is:
  - Positive: Predict +1
  - Negative: Predict -1





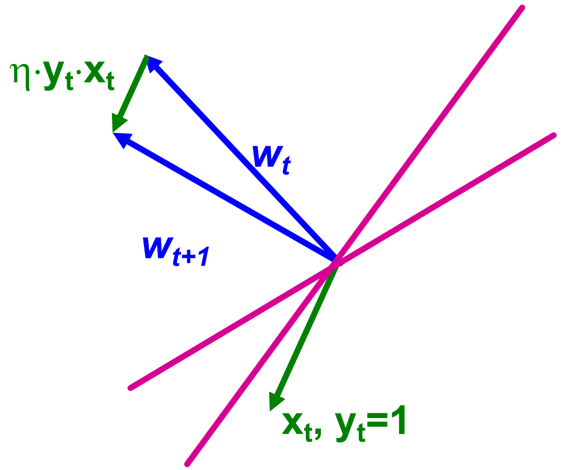


Perceptron

- Perceptron: y' = sign(w · x)
- How to find parameters w?
  - Start with $\boldsymbol{w_0 = 0}$
  - Pick training examples $\boldsymbol{x_t}$ one by one
  - Predict class of $\boldsymbol{x_t}$ using current $\boldsymbol{w_t}$
    - $y' = \text{sign}(w_t \cdot x_t)$
  - If y' is correct (i.e., $\boldsymbol{y_t} = y'$)
    - No change: $\boldsymbol{w_{t+1} = w_t}$
  - If y' is wrong: Adjust w_t

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} + \eta \cdot \boldsymbol{y_t} \cdot \boldsymbol{x_t}$$

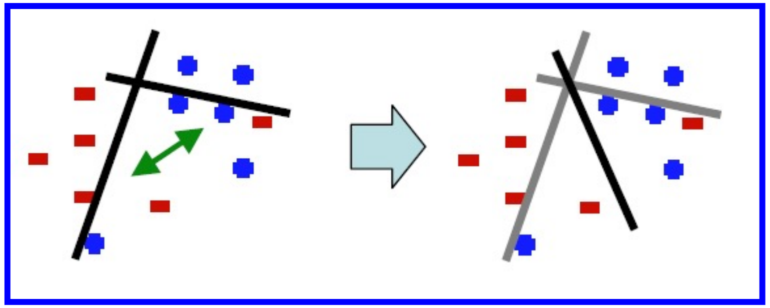
Note that the Perceptron is a conservative algorithm: it ignores samples that it classifies correctly.



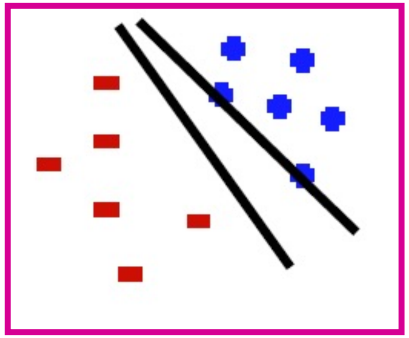
η is the learning rate parameter
$\boldsymbol{x_t}$ is the t-th training example
$\boldsymbol{y_t}$ is true t-th class label ({+1, -1})

Perceptron: The Good and the Bad

- Good: Perceptron convergence theorem:
  - If there exist a set of weights that are consistent (i.e., the data is linearly separable) the Perceptron learning algorithm will converge



- Bad: Never converges:
  If the data is not separable weights dance around indefinitely



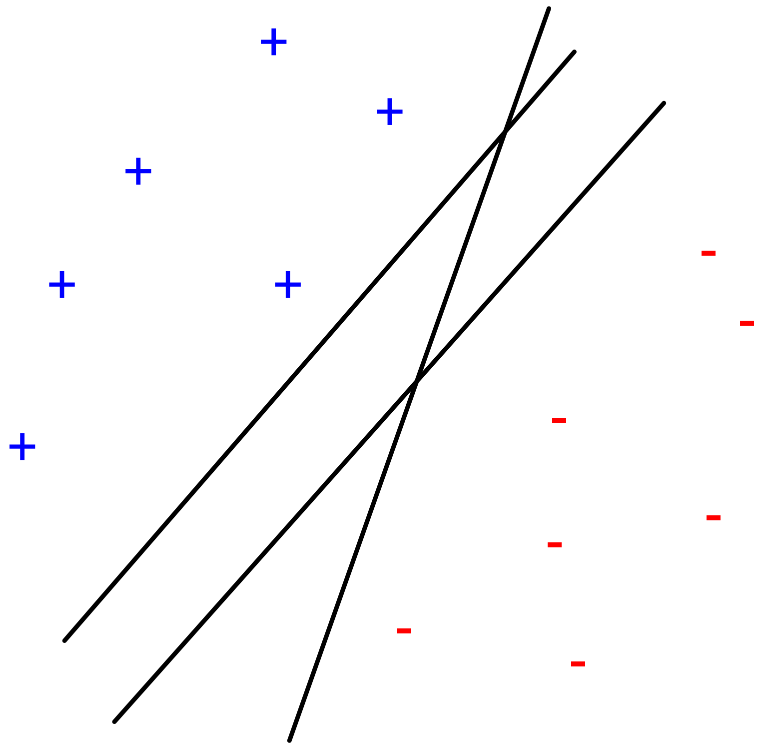
- Bad: Mediocre generalization:
  - Finds a “barely” separating solution

Outline

- Motivation
- Support Vector Machines
  - Optimal Margin Classifier
  - Slack Penalty
  - SGD SVM
- Distributed Deep Learning

Support Vector Machines

- Want to separate "+" from "-" using a line



Data:

Training examples:

$(x_1, y_1) \ldots (x_n, y_n)$

Each example i:

$x_i = (x_i^{(1)}, \ldots , x_i^{(d)})$

$x_i^{(j)}$ is real valued

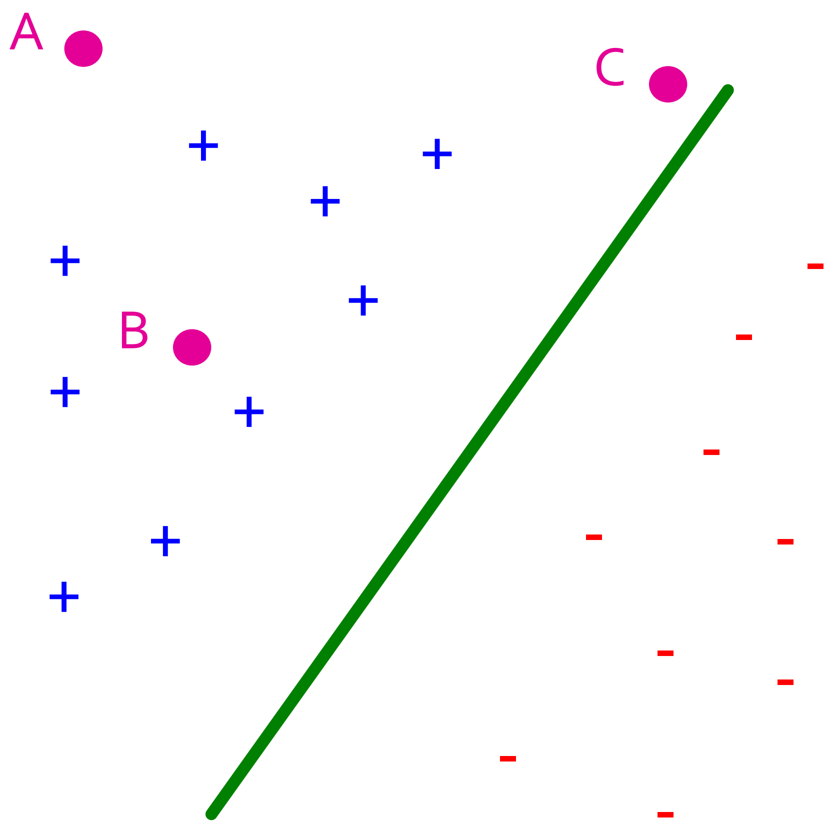
$y_i \in \{ -1, +1 \}$

Inner product:

$$\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{d} w^{(j)} \cdot x^{(j)}$$

Which is best linear separator (defined by w)?

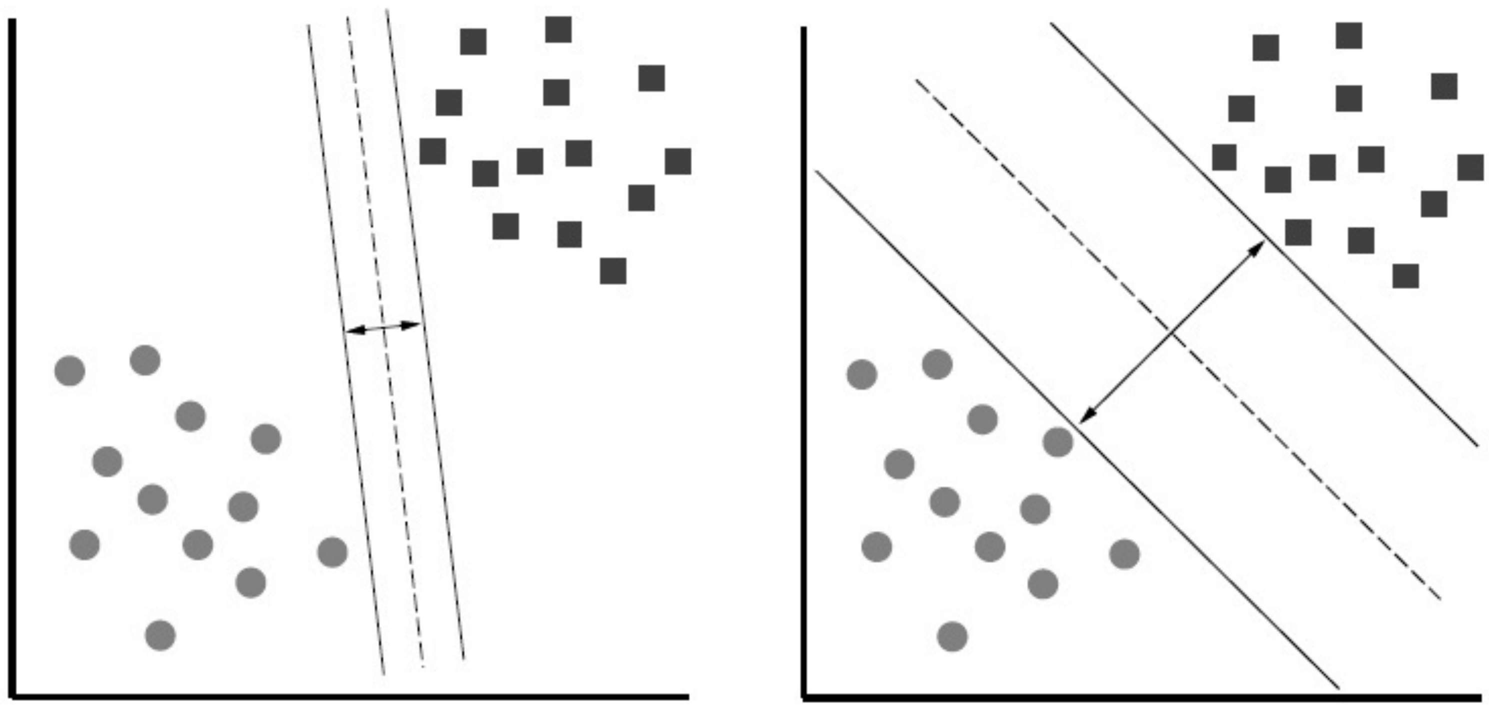
Largest Margin



- Distance from the separating hyperplane corresponds to the "**confidence**" of prediction
- Example:
  - We are more sure about the class of A and B than of C

Largest Margin

- Margin: Distance of **closest** example from the decision line/hyperplane

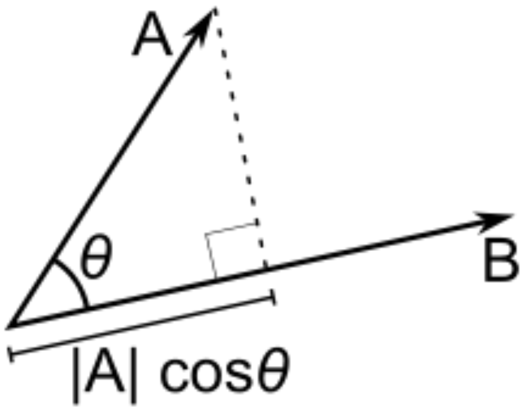


The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.

Why maximizing γ a good idea?

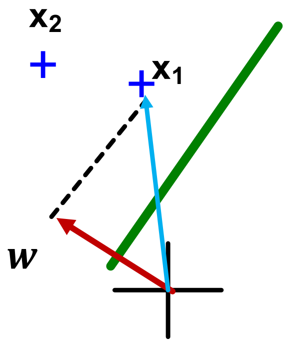
- Remember: Dot product

$$\boldsymbol{A} \cdot \boldsymbol{B} = \|\boldsymbol{A}\| \cdot \|\boldsymbol{B}\| \cdot \cos\boldsymbol{\theta}$$

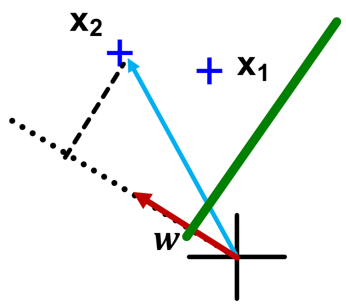


$$||A|| = \sqrt{\sum_{j=1}^{d}(A^{(j)})^2}$$

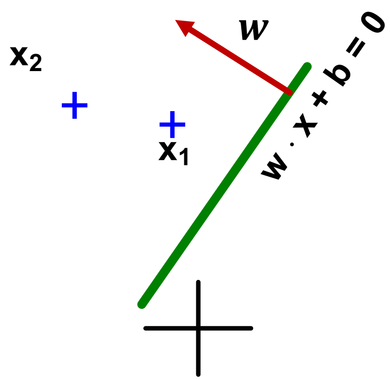
- What is $\boldsymbol{w} \cdot \boldsymbol{x_1}$, $\boldsymbol{w} \cdot \boldsymbol{x_2}$?



In this case
$\gamma_1 \approx \|w\|^2$



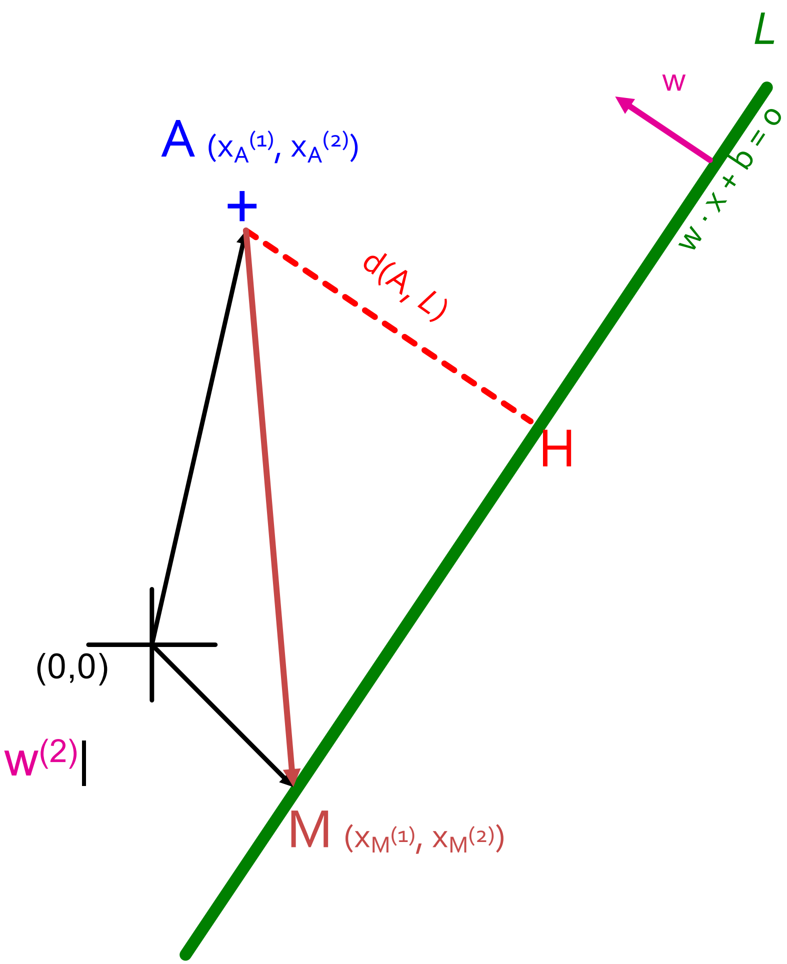
In this case
$\gamma_2 \approx 2\|w\|^2$



So, $\boldsymbol{\gamma}$ roughly corresponds to the margin:
Bigger $\boldsymbol{\gamma}$ bigger the separation

What is the margin?

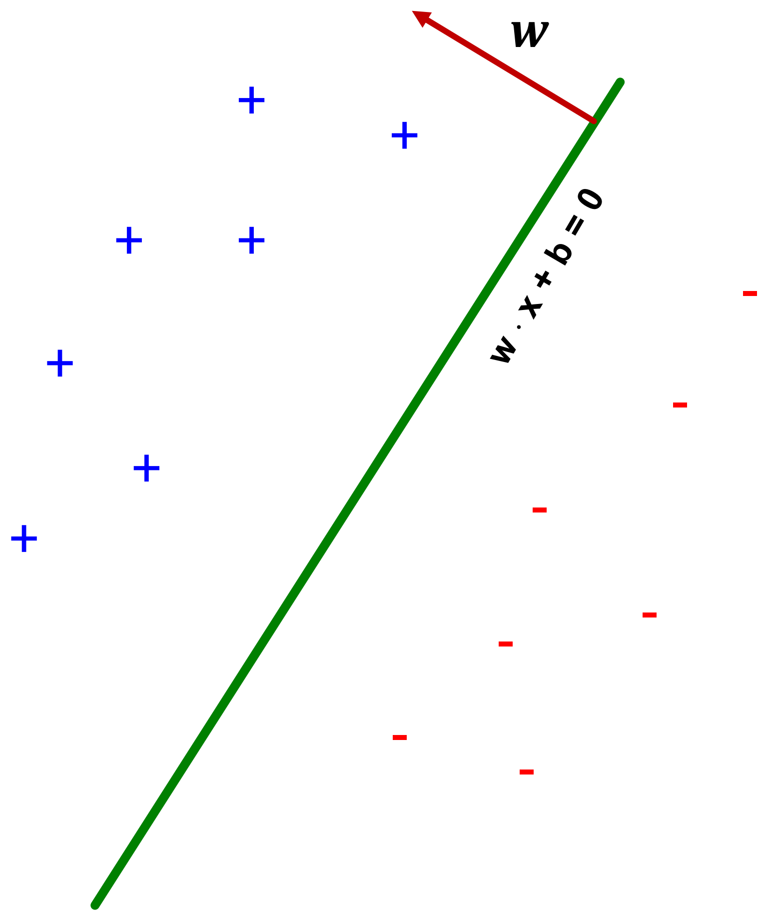
- Let (assume $||w||_2 = 1$):
  - Line L: $w \cdot x + b = w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + b = 0$
  - $w = (w^{(1)}, w^{(2)})$
  - Point A = $(x_A^{(1)}, x_A^{(2)})$
  - Point M on a line = $(x_M^{(1)}, x_M^{(2)})$



$$
\begin{aligned}
d(\mathbf{A}, \boldsymbol{L}) &= |AH| \\
&= |(A-M) \cdot w| \\
&= |(x_A^{(1)} - x_M^{(1)})\, w^{(1)} + (x_A^{(2)} - x_M^{(2)})\, w^{(2)}| \\
&= x_A^{(1)}\, w^{(1)} + x_A^{(2)}\, w^{(2)} + b \\
&= w \cdot \mathbf{A} + b
\end{aligned}
$$

Remember $\mathbf{x_M^{(1)} w^{(1)} + x_M^{(2)} w^{(2)} = - b}$
since **M** belongs to line **L**

Largest Margin



- Prediction = **sign($w \cdot x + b$)**
- "Confidence" = **$(w \cdot x + b)\, y$**
- For i-th datapoint:

$$\gamma_i = (w\, x_i + b) y_i$$

- Want to solve:
- $\max_{w,b} \min_i \gamma_i$
- Can rewrite as

$$\max_{w,\gamma} \gamma$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq \gamma$$

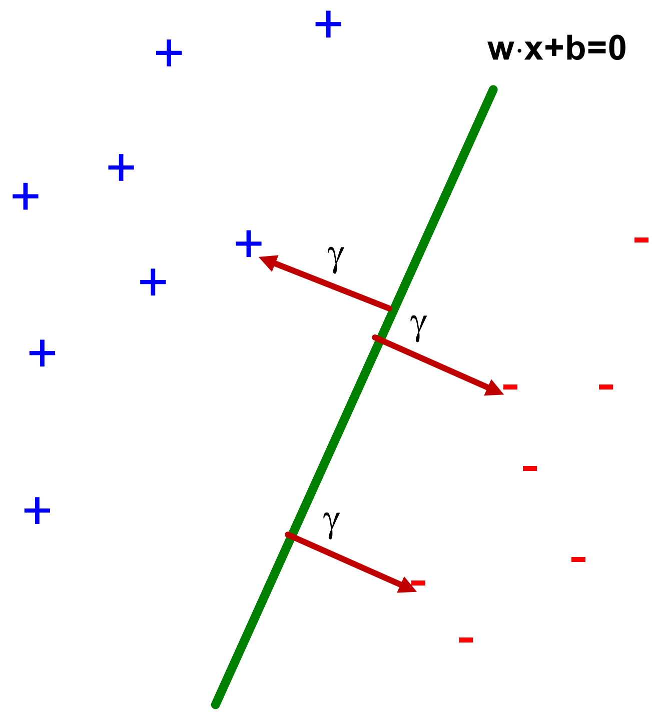
Support Vector Machine

- Maximize the margin:
  - Good according to intuition, theory (VC dimension) & practice

$$\max_{w,\gamma} \gamma$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq \gamma$$

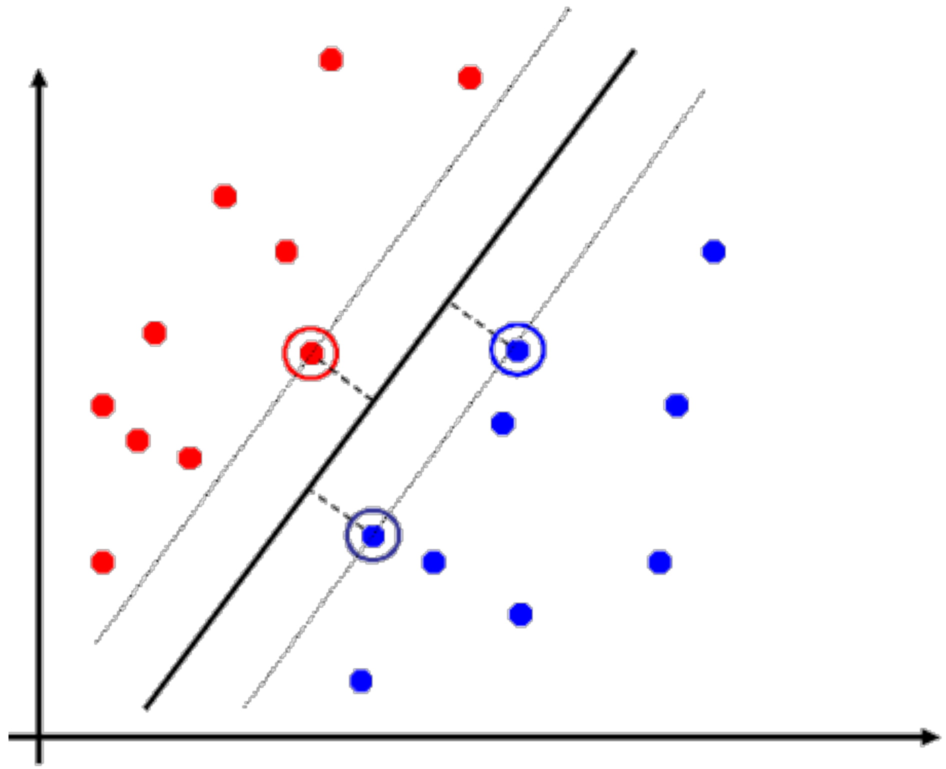
- $\boldsymbol{\gamma}$ is margin … distance from the separating hyperplane



Maximizing the margin

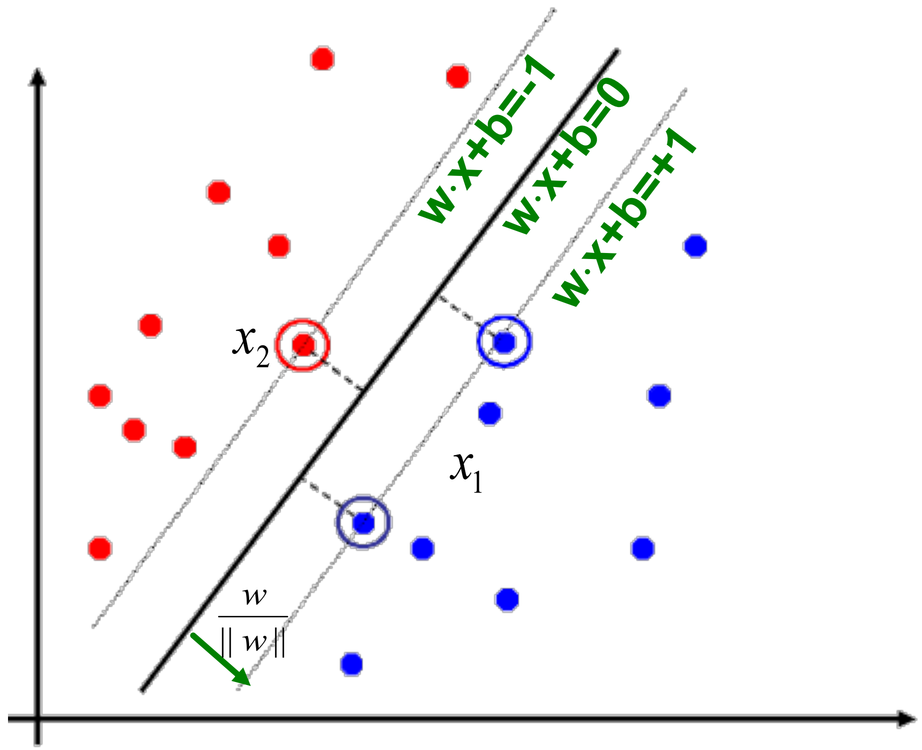
Support Vector Machines

- Separating hyperplane is defined by the **support vectors**
  - Points on +/- planes from the solution
  - If you knew these points, you could ignore the rest
  - Generally, d+1 support vectors (for d dim. data)



Canonical Hyperplane: Problem

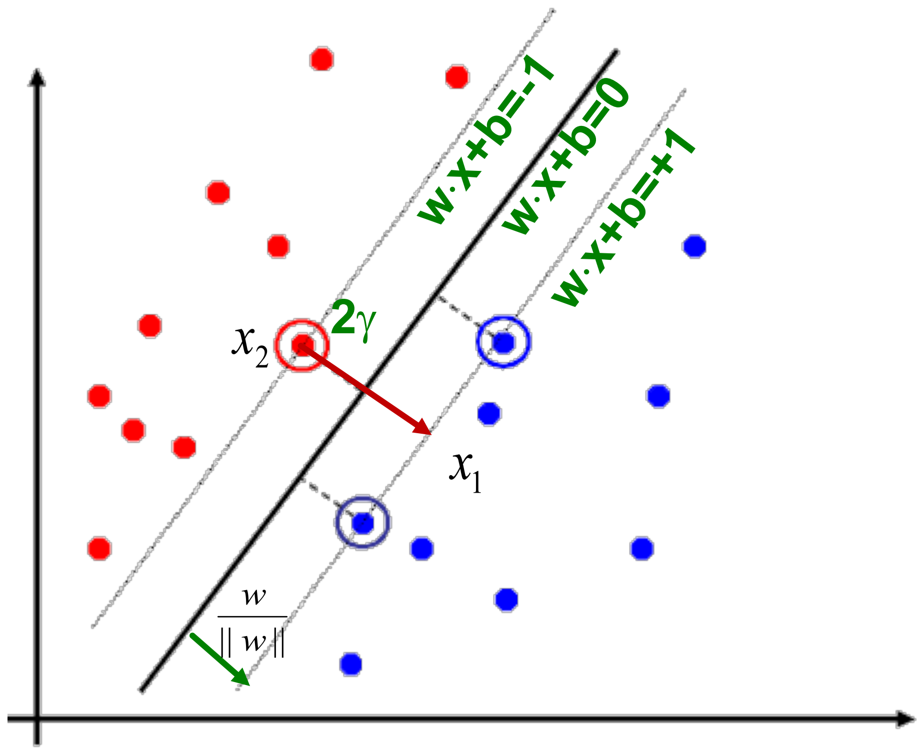
- Problem:
  - Let $(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b})\boldsymbol{y} = \boldsymbol{\gamma}$
    then $(2\boldsymbol{w} \cdot \boldsymbol{x} + 2\boldsymbol{b})\boldsymbol{y} = 2\boldsymbol{\gamma}$
    - Scaling w increases margin!
- Solution:
  - Work with normalized w:
  - $\boldsymbol{\gamma} = \left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \cdot \boldsymbol{x} + \boldsymbol{b}\right)\boldsymbol{y}$
  - Let's also require support vectors $\boldsymbol{x_j}$ to be on the plane defined by: $\boldsymbol{w} \cdot \boldsymbol{x_j} + \boldsymbol{b} = \pm\boldsymbol{1}$



$$||w|| = \sqrt{\sum_{j=1}^{d} (w^{(j)})^2}$$

Canonical Hyperplane: Solution

- Want to maximize margin γ!
- What is the relation between x_1 and x_2?
  - $x_1 = x_2 + 2\gamma \frac{w}{||w||}$
  - We also know:
    - $w \cdot x_1 + b = +1$
    - $w \cdot x_2 + b = -1$
- So:
  - $w \cdot x_1 + b = +1$
  - $w\left(x_2 + 2\gamma \frac{w}{||w||}\right) + b = +1$
  - $\underbrace{w \cdot x_2 + b}_{-1} + 2\gamma \frac{w \cdot w}{||w||} = +1$



$$\boxed{\Rightarrow \gamma = \frac{\|w\|}{w \cdot w} = \frac{1}{\|w\|}}$$

Note:
$w \cdot w = \|w\|^2$

Maximizing the Margin

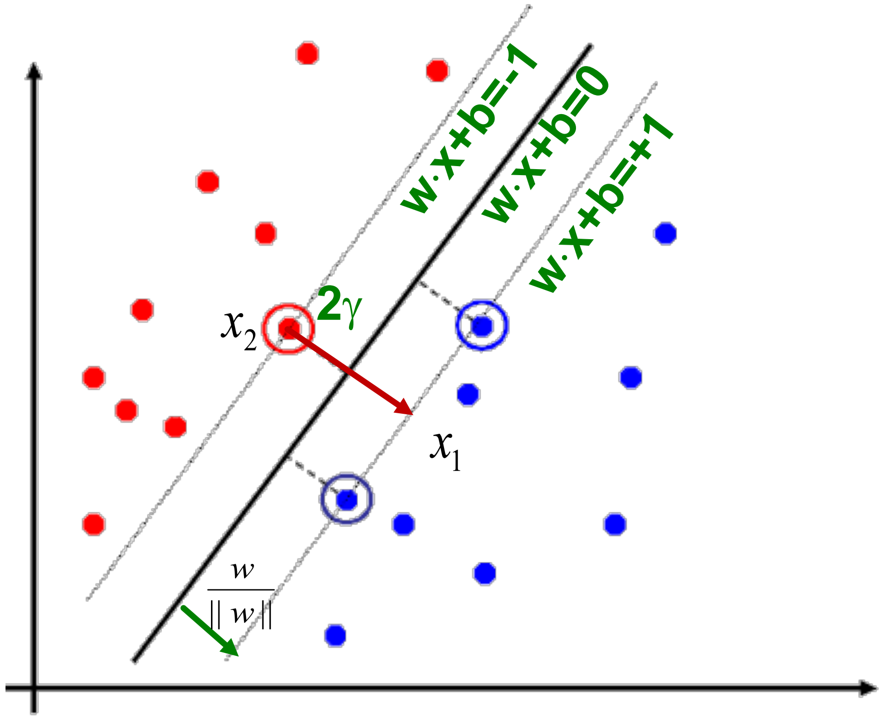
- We started with

$$\max_{w,\gamma} \gamma$$
$$s.t. \forall i, y_i(w \cdot x_i + b) \geq \gamma$$

But *w* can be arbitrarily large!

- We normalized and...

$$\arg\max \gamma = \arg\max \frac{1}{\|w\|} = \arg\min \|w\| = \arg\min \tfrac{1}{2}\|w\|^2$$



- Then:

$$\min_w \tfrac{1}{2} \| w \|^2$$
$$s.t. \forall i, y_i(w \cdot x_i + b) \geq 1$$

SVM with “hard” constraints

Optimal Margin Classifier

Back to our problem:

$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m$$

We write the constraint as:

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0$$

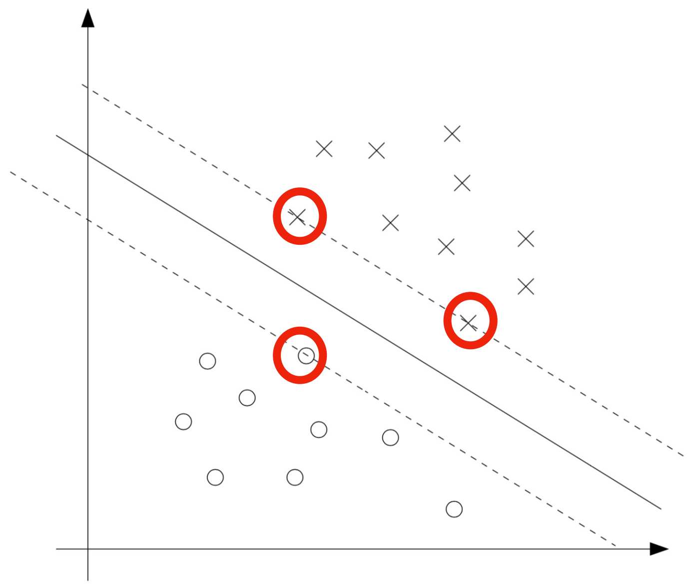
Lagrangian:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1 \right]$$

From KKT dual complementarity condition,
$\boldsymbol{\alpha}_i > 0$ only for $g_i(w) = 0$

training examples whose functional margin exactly equal to 1

Support Vectors!



We only need to care about the points with the smallest margins to the decision boundary!

Optimal Margin Classifier

- Rewrite our problem
- Solve the dual form of the problem by $\text{minimize } \mathcal{L}(w, b, \alpha)$ with respect to w and b
- Setting derivatives of L w.r.t. w and b to 0

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$$

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)} (w^T x^{(i)} + b) - 1 \right]$$

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}$$

Optimal Margin Classifier

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}$$

Dual optimization problem:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

Can check the conditions of “max min” == “min max” are satisfied, so solve the dual optimization problem to obtain **α*** instead!

substitute into $w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$ to get w*

Optimal Margin Classifier

Assume we have found (w*, b*), we want to make a prediction at a new input x

need to calculate: $w^T x + b$

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$$= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b$$

$$= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b.$$

the prediction only depends on the inner product between x and the points in the training set!

αi ≠ 0 for the support vectors, only need to consider the inner products between x and the support vectors

Outline

- Motivation
- Support Vector Machines
  - Optimal Margin Classifier
  - Slack Penalty
  - SGD SVM
- Distributed Deep Learning

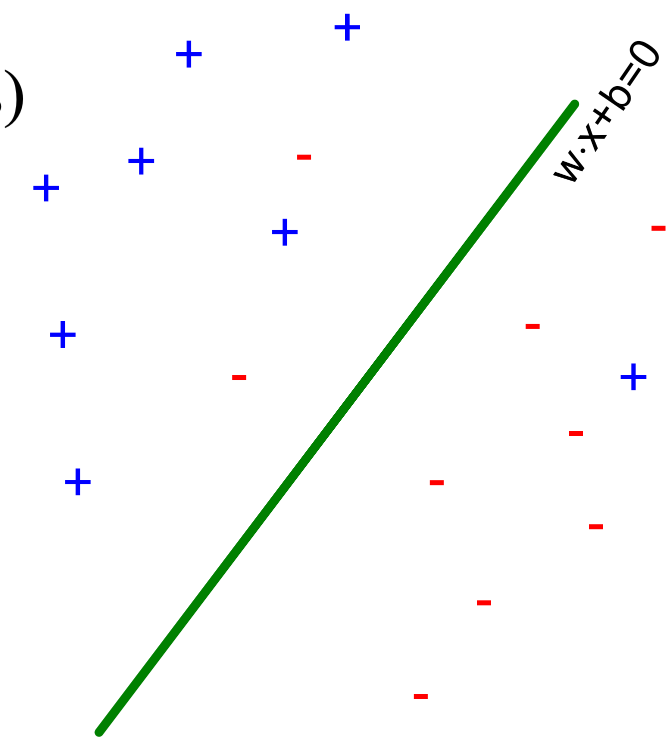
Non-linearly Separable Data

- If data is not separable introduce penalty:

$$\min_w \tfrac{1}{2}\|w\|^2 + \mathrm{C} \cdot (\#\text{number of mistakes})$$
$$s.t. \forall i, y_i(w \cdot x_i + b) \geq 1$$

  - Minimize $\|w\|^2$ plus the number of training mistakes
  - Set C using cross validation

- How to penalize mistakes?
  - All mistakes are not equally bad!



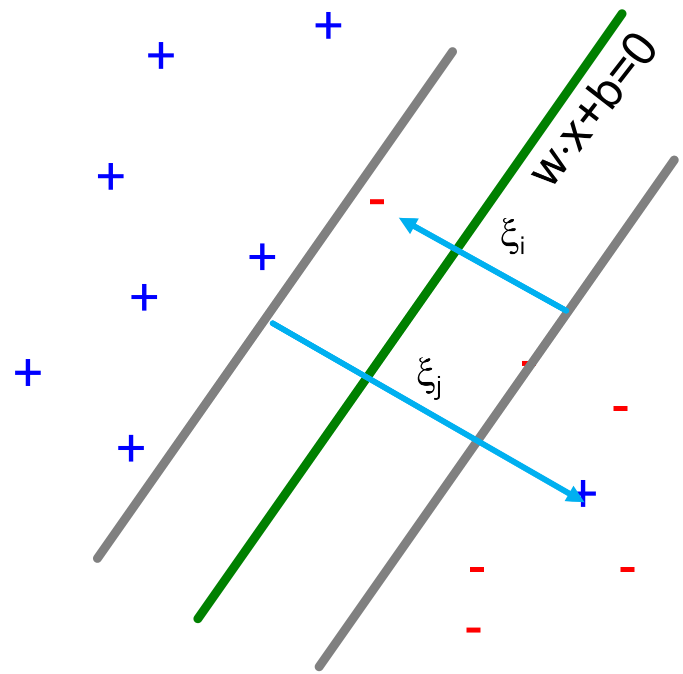
Support Vector Machines

- **Introduce slack variables ξ_i**

$$\min_{w,b,\xi_i \geq 0} \ \tfrac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq 1 - \xi_i$$

- If point $\boldsymbol{x_i}$ is on the wrong side of the margin then get penalty ξ_i

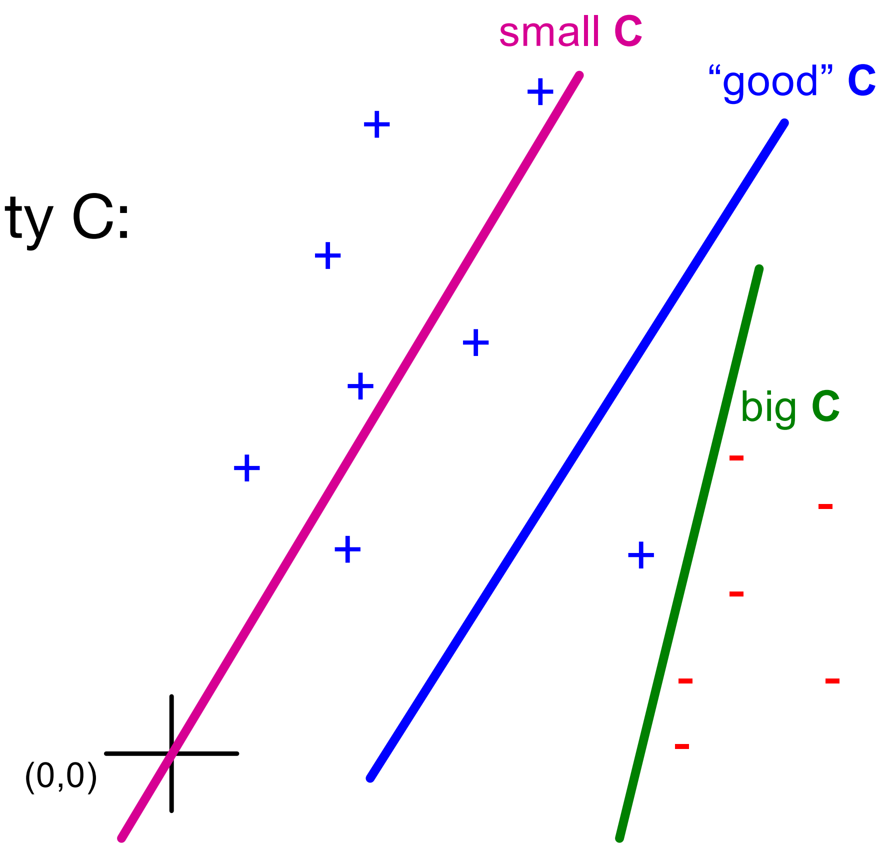


For each data point:
If margin ≥ 1, don't care
If margin < 1, pay linear penalty

Slack Penalty

$$\min_w \tfrac{1}{2}\|w\|^2 + \mathrm{C} \cdot (\#\text{number of mistakes})$$
$$s.t. \forall i, y_i(w \cdot x_i + b) \geq 1$$

- What is the role of slack penalty C:
  - C=∞: Only want **w, b** that separate the data
  - C=0: Can set ξ_i to anything, then **w=0** (basically ignores the data)



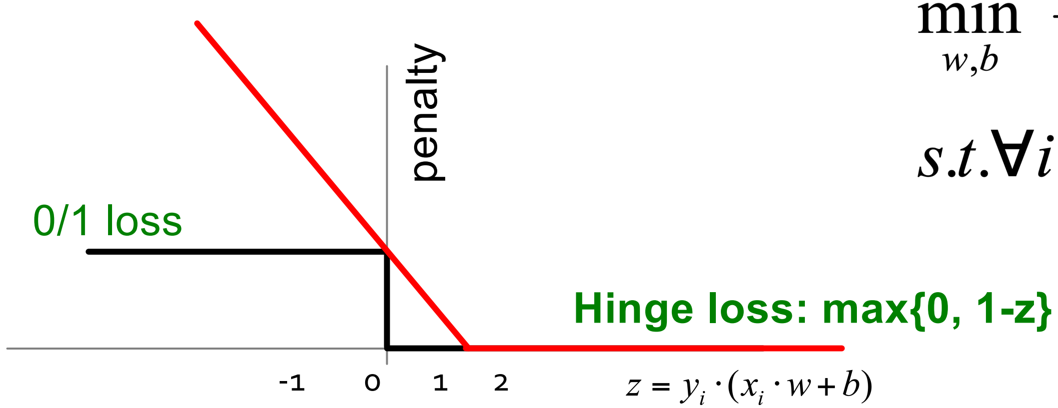
Support Vector Machines

- SVM in the "natural" form

$$\arg\min_{w,b} \underbrace{\tfrac{1}{2} w \cdot w}_{\text{Margin}} + C \cdot \sum_{i=1}^{n} \underbrace{\max\left\{0, 1 - y_i (w \cdot x_i + b)\right\}}_{\text{Empirical } \textbf{loss L} \text{ (how well we fit training data)}}$$

Regularization parameter

- SVM uses "Hinge Loss":



$$\min_{w,b} \tfrac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (w \cdot x_i + b) \geq 1 - \xi_i$$

Outline

- Motivation
- Support Vector Machines
  - Optimal Margin Classifier
  - Slack Penalty
  - SGD SVM
- Distributed Deep Learning

SVM: How to estimate w?

$$\min_{w,b} \ \tfrac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$$

- Want to estimate $\boldsymbol{w}$ and $\boldsymbol{b}$!
  - Standard way: Use a **solver**!
  - Solver: software for finding solutions to "common" optimization problems
- Use a quadratic solver:
  - Minimize quadratic function
  - Subject to linear constraints
- Problem: Solvers are inefficient for big data!

SVM: How to estimate w?

- Want to minimize *f(w,b)*:

$$f(w,b) = \tfrac{1}{2}\sum_{j=1}^{d}\left(w^{(j)}\right)^2 + C\sum_{i=1}^{n} \underbrace{\max\left\{0, 1 - y_i\left(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b\right)\right\}}_{\textbf{Empirical loss } L(x_i\ y_i)}$$

- Compute the gradient ∇(j) w.r.t. $w^{(j)}$

$$\nabla f^{(j)} = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

$$\frac{\partial L(x_i, y_i)}{\partial w^{(j)}} = 0 \qquad \text{if } y_i(\mathrm{w}\cdot x_i + b) \geq 1$$

$$= -y_i x_i^{(j)} \qquad \text{else}$$

SVM: How to estimate w?

- Gradient descent:

Iterate until convergence:
- **For j = 1 … d**
  - **Evaluate:** $$\nabla f^{(j)} = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$
  - **Update:**
    $\mathbf{w}^{(j)} \leftarrow \mathbf{w}^{(j)} - \eta \nabla \mathbf{f}^{(j)}$

η…learning rate parameter
C… regularization parameter

- Problem:
  - Computing $\nabla f^{(j)}$ takes O(n) time!
  - n … size of the training dataset

SVM: How to estimate w?

- Stochastic Gradient Descent
  - Instead of evaluating gradient over all examples evaluate it for each individual training example

We just had:

$$\nabla f^{(j)} = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

$$\nabla f^{(j)}(x_i) = w^{(j)} + C \cdot \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

Notice: no summation over *i* anymore

- Stochastic gradient descent:

Iterate until convergence:
- **For i = 1 … n**
  - **For j = 1 … d**
    - **Compute: $\nabla f^{(j)}(x_i)$**
    - **Update: $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}(x_i)$**

Other Variations of GD

- Batch Gradient Descent
  - Calculates error for each example in the training dataset, but updated model only after all examples have been evaluated (i.e., end of training epoch)
  - PROS: few updates, more stable error gradient
  - CONS: usually requires whole dataset in memory, slower than SGD
- Mini-Batch Gradient Descent
  - Like BGD, but using smaller batches of training data. Balance between robustness of SGD, and efficiency of BGD

Example: Text categorization

- Example by Leon Bottou:
  - Reuters RCV1 document corpus
  - Predict a category of a document
    - One vs. the rest classification
  - n = 781,000 training examples (documents)
  - 23,000 test examples
  - d = 50,000 features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

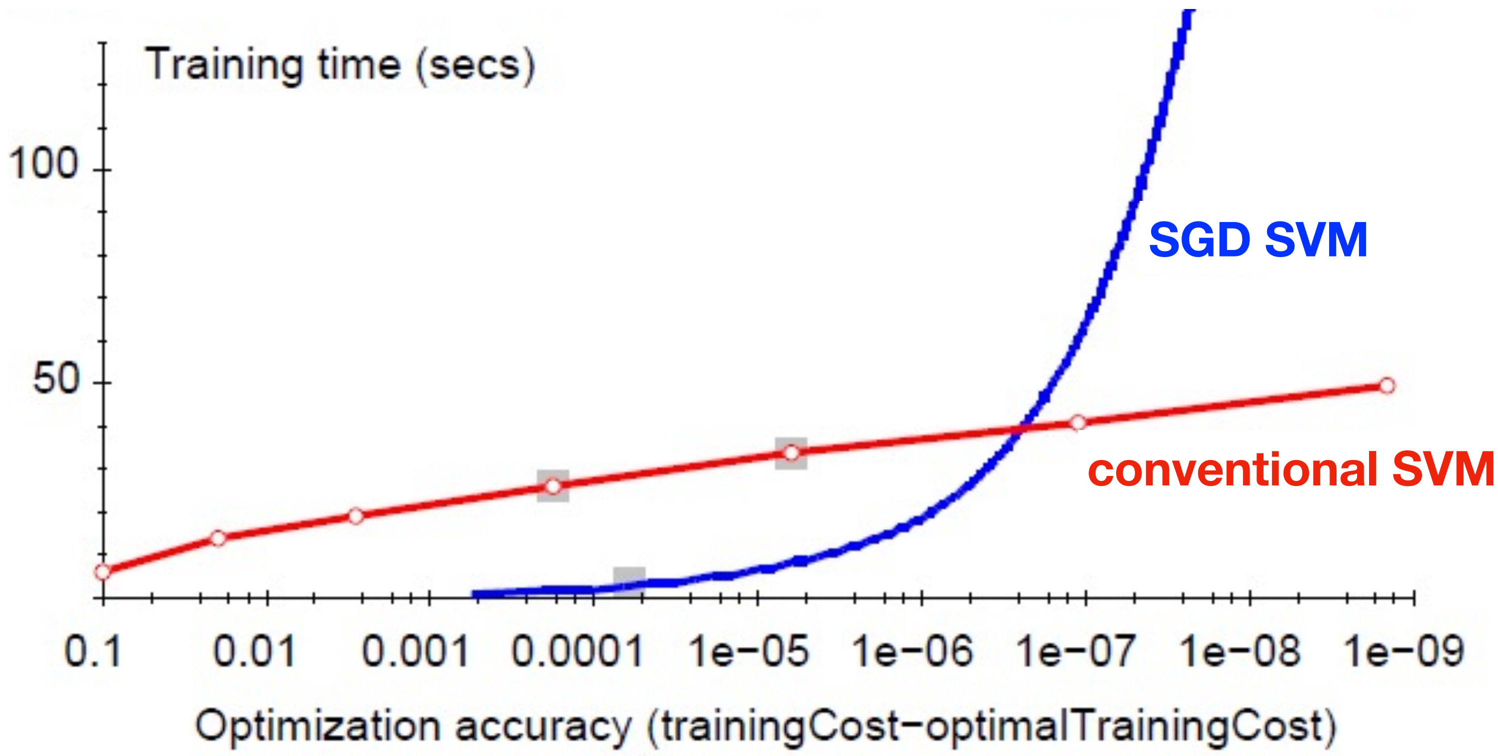
Example: Text categorization

- Questions:
  - (1) Is **SGD** successful at minimizing ***f(w,b)***?
  - (2) How quickly does **SGD** find the min of ***f(w,b)***?
  - (3) What is the error on a test set?

	Training time	***Value of f(w,b)***	***Test error***
Standard SVM	23,642 secs	*0.2275*	*6.02%*
"Fast SVM"	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	*0.2275*	*6.02%*

(1) SGD-SVM is successful at minimizing the value of ***f(w,b)***
(2) SGD-SVM is super fast
(3) SGD-SVM test set error is comparable

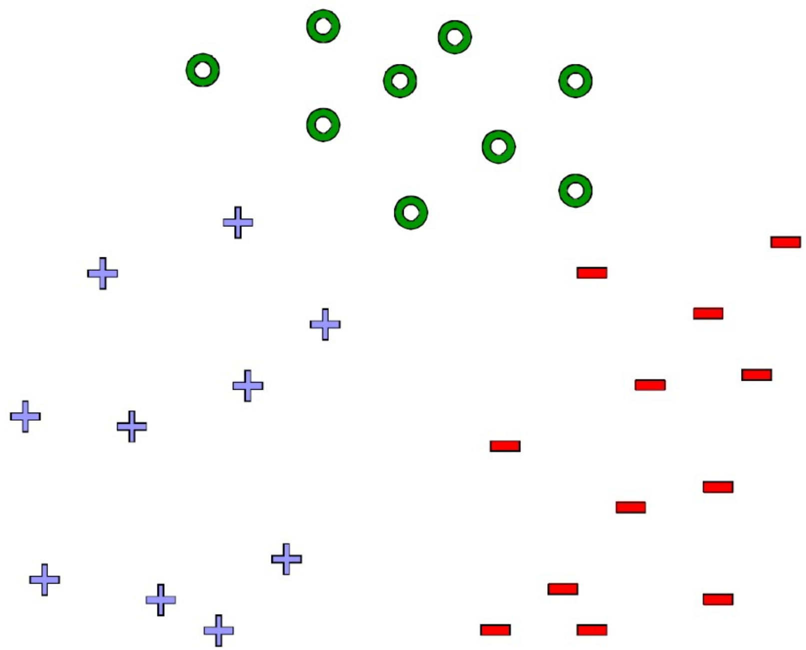
Optimization “Accuracy”



Optimization quality: | $f(w,b) - f(w^{opt},b^{opt})$ |

For optimizing *f(w,b) within reasonable* quality
SGD-SVM is super fast

What about multiple classes?



- **Idea 1:**

  One against all

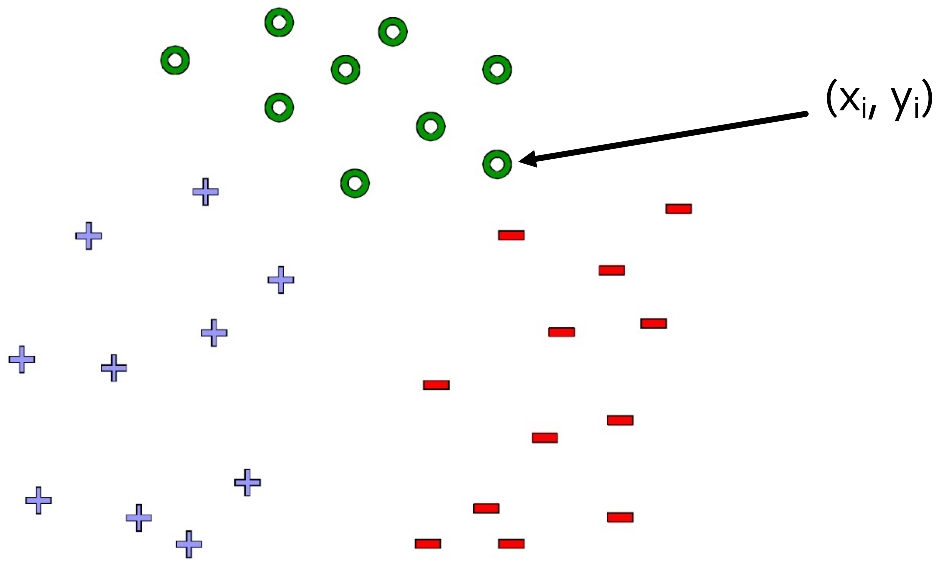
  Learn 3 classifiers

  - **+ vs. {o, -}**
  - **- vs. {o, +}**
  - **o vs. {+, -}**
  - Obtain:
  - $\mathbf{w_+ \, b_+, \;\; w_- \, b_-, \;\; w_o \, b_o}$
- **How to classify?**
- Return class *c*

  $\mathbf{\arg\max_c \; w_c x + b_c}$

Learn 1 classifier: Multiclass SVM

- **Idea 2: Learn 3 sets of weights simultaneously!**
  - For each class ***c*** estimate $\boldsymbol{w_c}$, $\boldsymbol{b_c}$
  - **Want the correct class to have highest margin:**
  - $\mathbf{w_{y_i} x_i + b_{y_i} \geq 1 + w_c x_i + b_c \quad \forall c \neq y_i \;, \forall i}$



Multiclass SVM

- **Optimization problem:**

$$\min_{w,b} \; \tfrac{1}{2}\sum_{c}\left\|w_c\right\|^2 + C\sum_{i=1}^{n}\xi_i \qquad \forall c \neq y_i, \forall i$$

$$w_{y_i} \cdot x_i + b_{y_i} \geq w_c \cdot x_i + b_c + 1 - \xi_i \qquad \xi_i \geq 0, \forall i$$

  - To obtain parameters $\boldsymbol{w_c}$, $\boldsymbol{b_c}$ (for each class $\boldsymbol{c}$) we can use similar techniques as for 2 class **SVM**

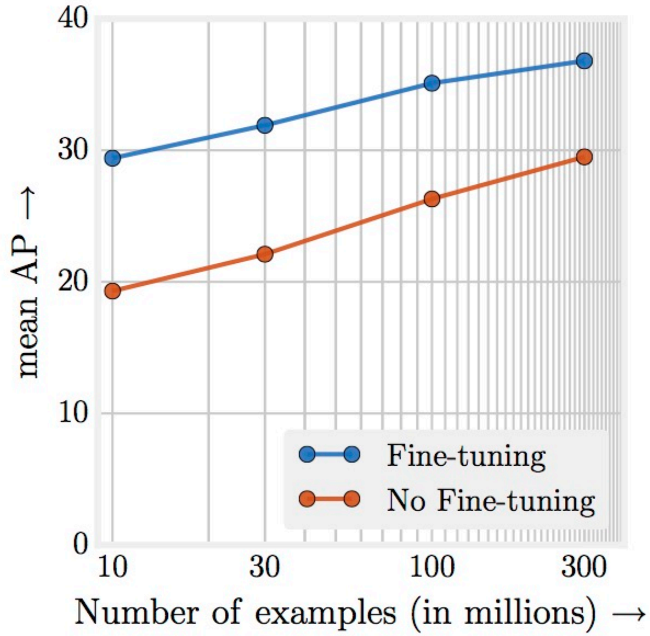
- **SVM is widely perceived a very powerful learning algorithm**

Outline

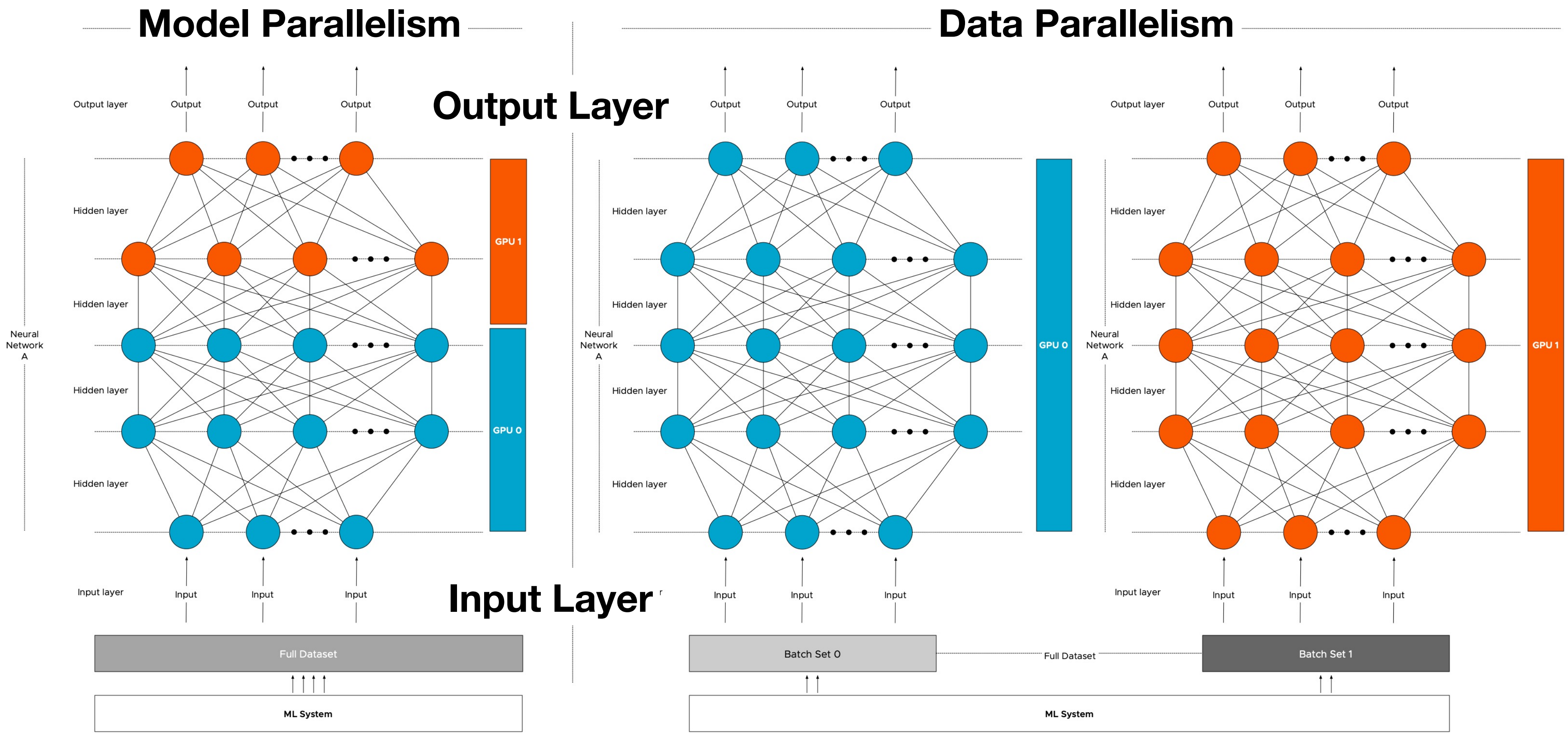
- Motivation
- Support Vector Machines
  - Optimal Margin Classifier
  - Slack Penalty
  - SGD SVM
- Distributed Deep Learning

Why Large-Scale ML?

- The Unreasonable Effectiveness of Data
    - In 2017, Google revisited a 15-year-old experiment on the effect of data and model size in ML, focusing on the latest Deep Learning models in computer vision
- Findings:
    - Performance increases logarithmically based on volume of training data
    - Complexity of modern ML models (i.e., deep neural nets) allows for even further performance gains
- Large datasets + large ML models => amazing results!

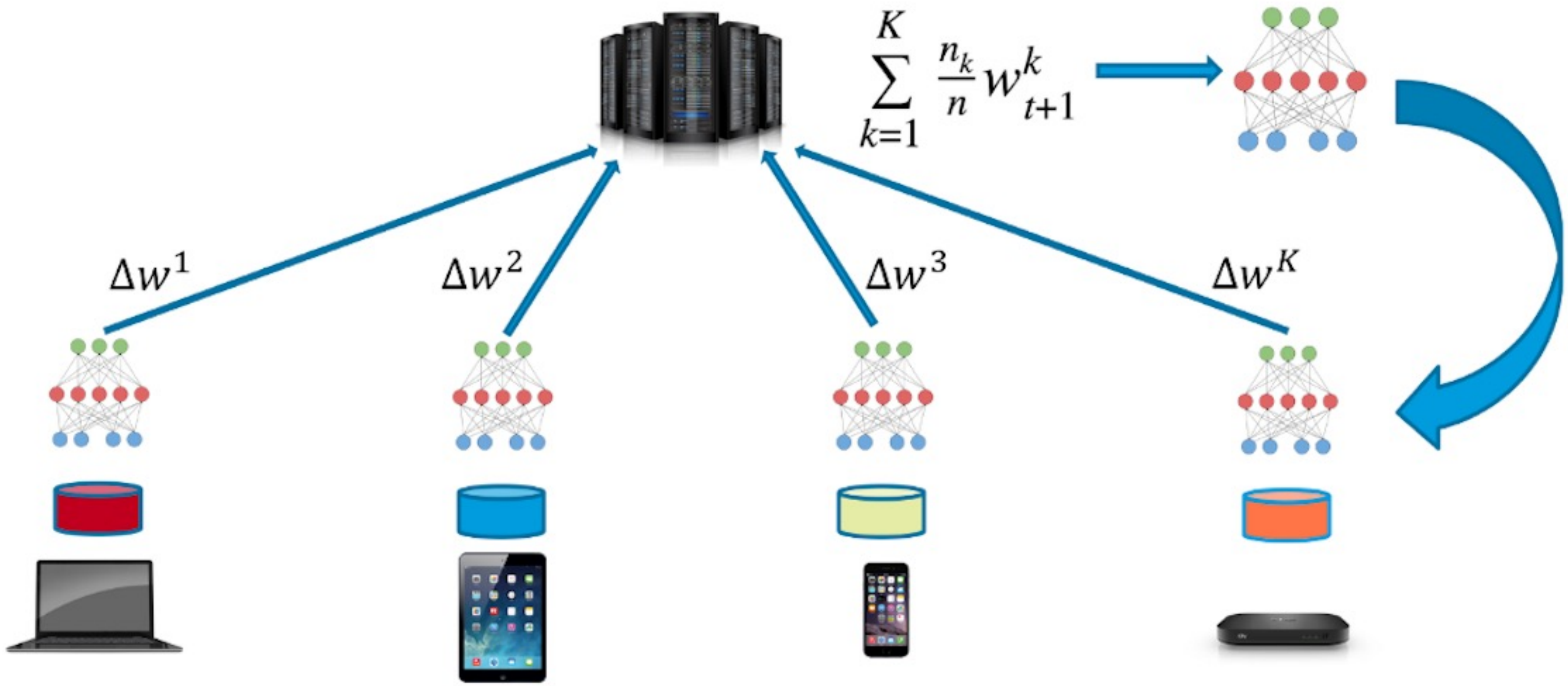


Parallelization Overview



Data Parallelism

- Divide training data into a number of subsets and run a copy of the model on each subset
- Model replicas asks the parameter server for an updated copy of model params
- Model replicas run minibatch SGD independently to compute gradients and send gradients to parameter server
- Parameter server applies the **averaged gradients** to the current model params



Sync SGD

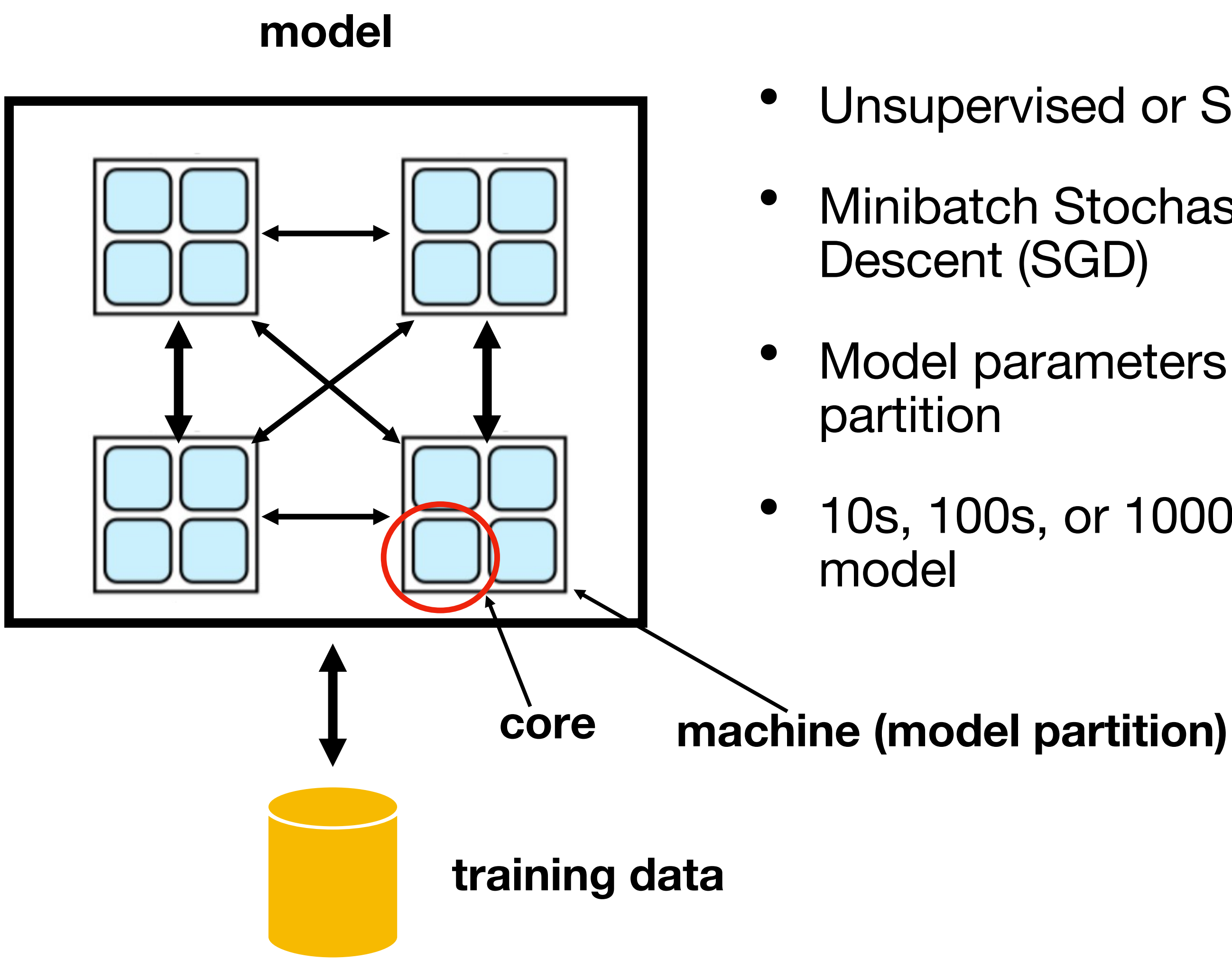
Sync-SGD Alg: worker k

Input: Dataset $\mathcal{X}$
Input: B mini-batch size
for $t = 0, 1, \ldots$ **do**
  Wait to read $\theta^{(t)} = (\theta^{(t)}[0], \ldots, \theta^{(t)}[M])$ from parameter servers.
  $G_k^{(t)} := 0$
  for $i = 1, \ldots, B$ **do**
    Sample datapoint $\tilde{x}_{k,i}$ from $\mathcal{X}$.
    $G_k^{(t)} \leftarrow G_k^{(t)} + \frac{1}{B}\nabla F(\tilde{x}_{k,i}; \theta^{(t)})$.
  end
  Send $(G_k^{(t)}, t)$ to parameter servers.
end

Sync-SGD Alg: Parameter Server j

Input: $\gamma_0, \gamma_1, \ldots$ learning rates.
Input: α decay rate.
Input: N number of mini-batches to aggregate.
Input: $\theta^{(0)}$ model initialization.
for $t = 0, 1, \ldots$ **do**
  $\mathcal{G} = \{\}$
  while $|\mathcal{G}| < N$ **do**
    Wait for (G, t') from any worker.
    if $t' == t$ **then** $\mathcal{G} \leftarrow \mathcal{G} \cup \{G\}$.
    else Drop gradient G.
  end
  $\theta^{(t+1)}[j] \leftarrow \theta^{(t)}[j] - \frac{\gamma_t}{N} \sum_{G \in \mathcal{G}} G[j]$.
  $\bar{\theta}^{(t)}[j] = \alpha\bar{\theta}^{(t-1)}[j] + (1-\alpha)\theta^{(t)}[j]$.
end

Model Parallelism

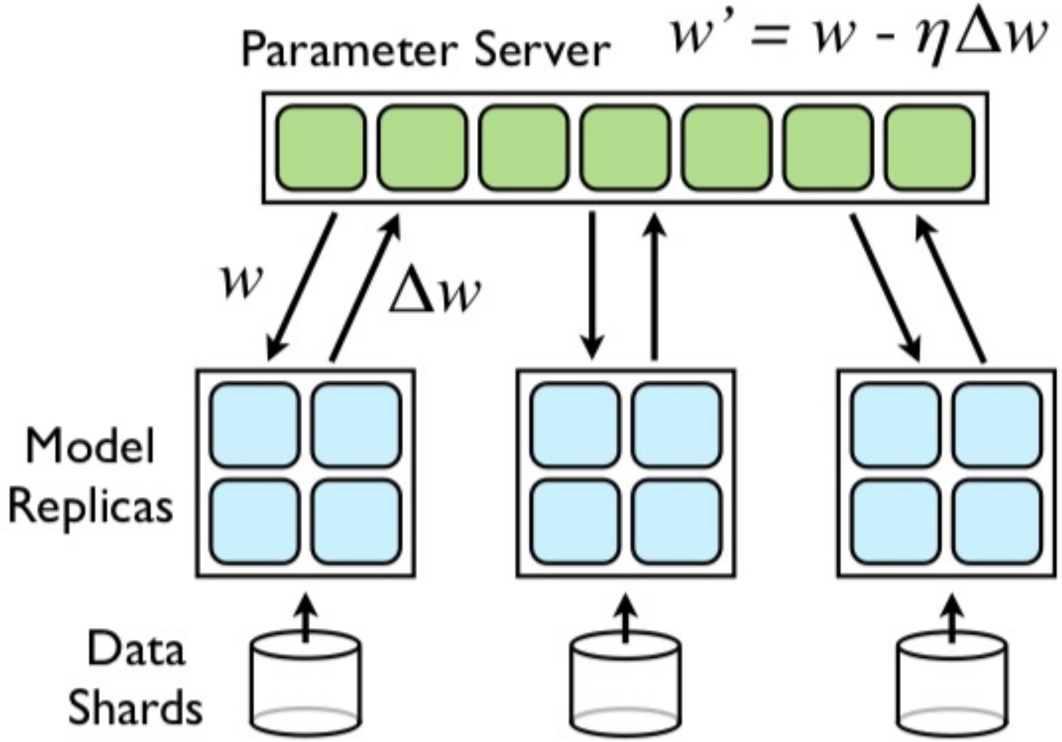


- Unsupervised or Supervised Objective
- Minibatch Stochastic Gradient Descent (SGD)
- Model parameters sharded by partition
- 10s, 100s, or 1000s of cores per model

Model Parallelism

Async SGD: an asynchronous stochastic gradient descent procedure

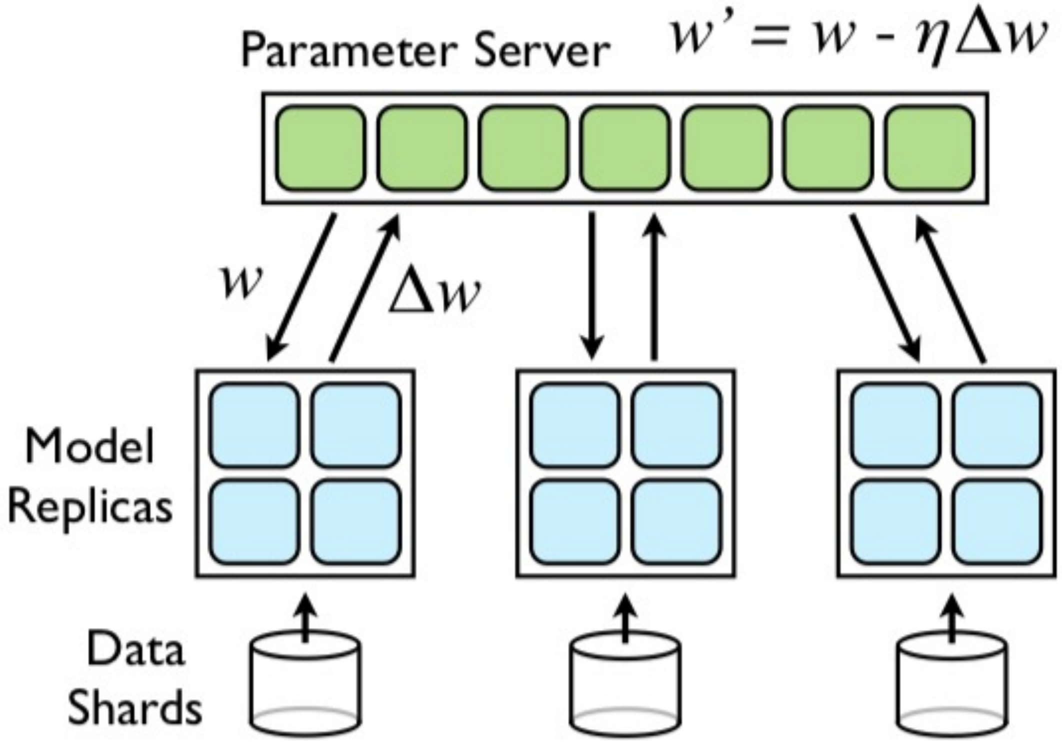
Key/Value store



Google, "Large Scale Distributed Deep Networks" [2012]

- Divide training data into a number of subsets and run a copy of the model on each subset
- Model replicas asks the parameter server for an updated copy of model params
- Model replicas run minibatch SGD independently to compute gradients and send gradients to parameter server
- Parameter server applies the gradients to the current model params

Model Parallelism



Systems challenges:

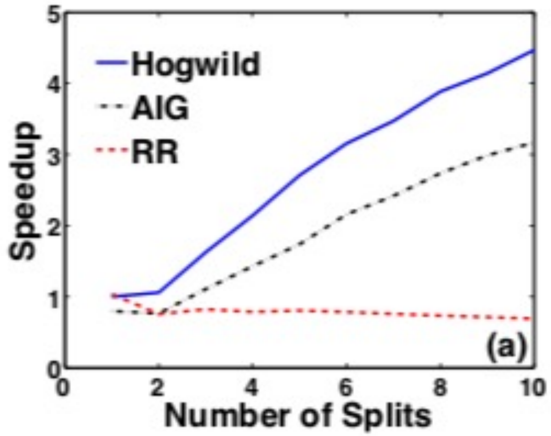
- High bandwidth
- Convergence
- Fault tolerance

Why do parallel updates work?

Async SGD

- Key idea: don't synchronize, just overwrite parameters opportunistically from multiple workers (i.e., servers)
  - Same implementation as SGD, just without blocking!
- In theory, Async SGD converges, but a slower rate than the serial version
- In practice, when gradient updates are sparse (i.e., high dimensional data), same convergence!

RR: a round-robin approach



Recht et al. "HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent"

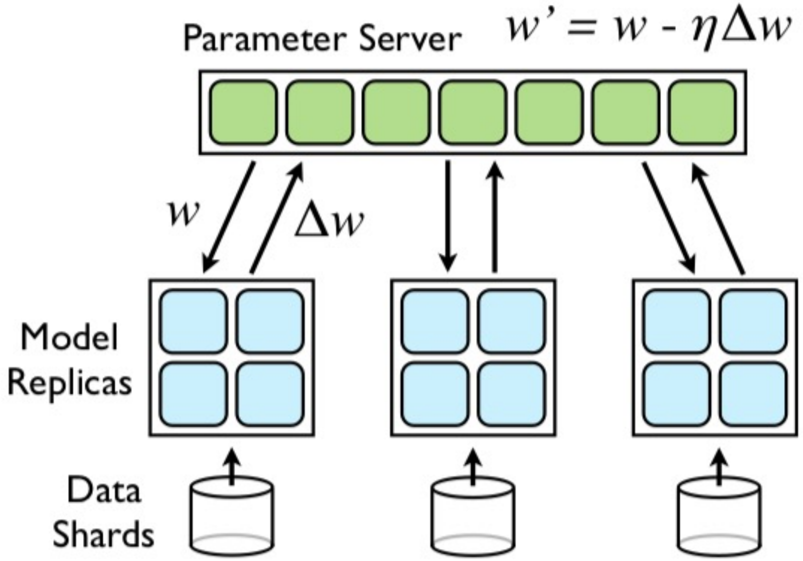
HOGWILD!

```
Initialize w in shared memory // in parallel
for i = 1, ..., p do
    while TRUE do
        if stopping criterion met then
            break
        end
        Sample j from 1, ..., n uniformly at random
        Compute f_j(w) and ∇f_j(w) using whatever w is currently available
        Let e_j denote non-zero indices of x_i
        for k ∈ e_j do
            w_k ← w_k − α[∇F_j(w)]_(k)
        end
    end
end
```

SGD

component-wise gradient updates
(relies on sparsity)

Async Distributed SGD



- All ingredients together:
  - Model and Data parallelism
  - Async SGD
- Dawn of modern deep learning

- From an engineering view, this is much better than a single model with the same number of total machines:
  - Synchronization boundaries involve fewer machines
  - Better robustness to individual slow machines
  - Makes forward progress even during evictions/restarts

Reading

- Jure Leskovec, Anand Raj, Jeff Ullman, “Mining of Massive Datasets,” Cambridge University Press, Chapter 12