

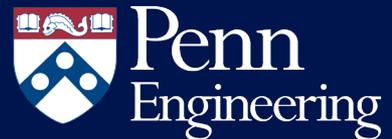


# On-line Learning, Perceptron, Kernels

Dan Roth

danroth@seas.upenn.edu | <http://www.cis.upenn.edu/~danroth/> | 461C, 3401 Walnut

Slides were created by Dan Roth (for CIS519/419 at Penn or CS446 at UIUC), or borrowed from other authors who have made their ML slides available.



# Administration (10/5/20)

Are we recording? YES!

Available on the web site

- Remember that all the lectures are available on the website **before the class**
  - **Go over it and be prepared**
  - We will add written notes next to each lecture, with some more details, examples and, (when relevant) some code.
- **HW 1** is due tonight.
  - Covers: SGD, DT, Feature Extraction, Ensemble Models, & Experimental Machine Learning
- **HW 2** is out tonight
  - Covers: Linear classifiers; learning curves; an application to named entity recognition; domain adaptation.
  - **Start working on it now. Don't wait until the last day (or two) since it could take a lot of your time**
- PollEv: If you haven't done so, please register <https://pollev.com/cis519/register>
  - We still have some problems with missing participation for students
- Quizzes: <https://canvas.upenn.edu/courses/1546646/quizzes/2475913/statistics>
- Go to the recitations and office hours
- Questions? **Please ask/comment during class; give us feedback**

# Comments on HW1

---

- Learning Algorithms
  - Decision Trees (Full, limited depth)
  - Stochastic Gradient Descent (over original and a richer feature space)
- Should we get different results?
  - Why, why not?
- Results on what?
  - Training data?
  - Future data (Data that is not used in training; a.k.a Test data)?
    - Note that the use of **cross-validation** just allows us to get a better estimate to the performance on future data;

# What are the factors that dictate the results of the different classifiers? (Choose up to 3)

Size of Training Set

The Hypothesis Space (what functions we learn)

The Data Set (Training and Test data)

The hidden function

The tuning of the models

The size of the test data

The efficiency of our programs

# Comments on HW1 (Cont.)

- Hypothesis Spaces:

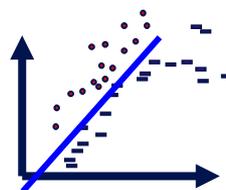
- DT (full) >> DT(4) ?> SGD << SGD(DT(4))

Should really call this “linear”

Should really call this “linear” over a richer feature set

- Training data performance:

- Directly dictated by the size of the hypothesis space
- But, depending on the data, learning with simpler hypothesis spaces can also get 100% on training



- Test data performance

- Learning using hypothesis spaces that are too expressive leads to overfitting
- Recall an example I gave in the Intro class, using deep NN for this dataset

Our expressive model learned the dataset

- Data set or the hidden function?

- In principle, it's the hidden function that determines what will happen – is it too expressive/expressive enough for the hypothesis space we are using?
- But, if we don't see a lot of data, our data may not be representative of the hidden function.
  - In training, we will actually learn another function, and thus not do well at test time.

- At this point, we are developing an intuition, but we will quantify some of it soon.

# A Guide

---

- Learning Algorithms
  - (Stochastic) Gradient Descent (with LMS)
  - Decision Trees
- \* • Importance of hypothesis space (representation)
- How are we doing?
  - We quantified in terms of cumulative # of mistakes (on test data)
  - But our algorithms were driven by a different metric than the one we care about.
  - Today, we are going to start discussing quantification.

# A Guide

- Versions of Perceptron
  - How to deal better with large features spaces & sparsity?
  - Variations of Perceptron
    - Dealing with overfitting
  - Closing the loop: Back to Gradient Descent
  - Dual Representations & Kernels
- Multilayer Perceptron
- Beyond Binary Classification?
  - Multi-class classification and Structured Prediction
- More general ways to quantify learning performance (PAC)
  - New Algorithms (SVM, Boosting)

Today:

Take a more general perspective and think more about learning, learning protocols, quantifying performance, etc.

This will motivate some of the ideas we will see next.

# Two Key Goals

---

- (1) Learning Protocols
  - Most of the class is devoted to Supervised Learning
  - Most of machine learning today is spent doing supervised learning.
- But, there is more to ML than supervised learning
  - We'll introduce some ideas that can motivate other protocols, like active learning and teaching (human in the loop)
- (2) We will do it in the context of Linear Learning algorithms
  - Why? Don't we only care about neural networks?
  - No.
    - First, the algorithmic ideas we'll introduce are the same used in neural networks.
    - More importantly, there are good reasons to want to know about simpler learning algorithms.

# Example (Named Entity Recognition)

## Named Entity Recognition Demo

23,847 views

About This Demo

If you wish to cite this work, please cite the following publications: (1) [Design Challenges and Misconceptions in Named Entity Recognition](#), (2) [Illinois Named Entity Recognizer: Addendum to Ratinov and Roth '09 reporting improved results](#).

BOONE, N.C. — Since last Monday, when a sophomore at his school died from suspected Covid-19 complications, Chase Sturgis says he has been thinking about his own bout with the coronavirus — and his own mortality. Mr. Sturgis, 21, had been avoiding socializing over the summer, but as students at his school, Appalachian State University, began returning to campus in August, he yielded to temptation.

Submit

The Named Entity Recognizer has identified the following named entities.

[GPE BOONE] , [GPE N.C.] — Since [DATE last Monday] , when a sophomore at his school died from suspected [NORP Covid-19] complications, [PERSON Chase Sturgis] says he has been thinking about his own bout with the coronavirus — and his own mortality. Mr. [PERSON Sturgis] , [DATE 21] , had been avoiding socializing over [DATE the summer] , but as students at his school, [ORG Appalachian State University] , began returning to campus in [DATE August] , he yielded to temptation.

- (A “standard” dataset will only care about PER, LOC, ORG, MISC)

# Linear Models? Consider Performance

- Named Entity Recognition in 22 languages
- CCG: a state-of-the-art **linear model** (over expressive features)
- fastText, mBERT: state of the art **neural models** based on bi-LSTM-CRF, with different embeddings

Language	CCG	fastText	Wiki	mBERT
aka	72.46	68.87	<b>76.37</b>	
amh	<b>69.15</b>	60.74	52.01	
ara	55.22	54.21	<b>61.43</b>	
ben	<b>77.10</b>	73.48	76.50	
cmn	<b>74.18</b>	70.06	68.93	
fas	61.72	60.50	<b>66.36</b>	
hin	71.63	68.69	<b>72.27</b>	
hun	62.80	59.29	<b>78.30</b>	
ind	66.64	56.80	<b>75.57</b>	
rus	67.18	66.07	<b>77.35</b>	
som	76.75	70.93	<b>77.37</b>	
spa	67.50	58.51	<b>76.37</b>	
swa	<b>76.77</b>	70.34	76.35	
tam	62.85	66.80	<b>69.55</b>	
tgl	81.61	76.35	<b>86.65</b>	
tha	73.49	<b>77.29</b>	75.76	
tur	76.33	69.14	<b>84.97</b>	
uzb	78.17	73.46	<b>80.87</b>	
vie	56.49	51.37	<b>68.72</b>	
wol	<b>76.57</b>	71.38	73.68	
yor	73.08	66.23	<b>73.84</b>	
zul	75.64	78.32	<b>81.55</b>	
avg	70.61	66.77	<b>74.13</b>	

Table 17: Results from the Monolingual Experiments

# But,...Consider Cost Effectiveness

- The linear model is
  - more than 50 times faster to train.
  - ~20 times faster to evaluate.
- Companies that need to run it over millions of documents will consider the cost (time & money) and would often prefer linear models.

Language	Codes	Training (hr:min:sec)			Evaluation (sec)		
		XLM-RoBERTa	mBERT	Cogcomp	XLM-RoBERTa	mBERT	Cogcomp
Akan (Twi)	aka	1:44:55	0:57:04	0:01:12	39	40	4
Amharic	amh	1:41:10	0:36:34	0:01:45	38	41	3
Arabic	ara	1:41:16	0:52:38	0:01:33	38	41	4
Bengali	ben	2:10:58	1:23:21	0:02:37	46	48	6
Mandarin	cmn	1:54:38	1:14:08	0:02:17	42	40	4
Farsi	fas	1:03:31	0:42:56	0:00:57	37	28	0.75
Hindi	hin	1:31:41	0:56:15	0:01:29	40	42	2
Hungarian	hun	1:12:50	0:44:59	0:01:25	39	40	4
Indonesian	ind	1:34:25	0:56:33	0:01:30	38	40	4
Russian	rus	1:34:25	0:53:49	0:01:24	38	41	5
Somali	som	1:15:04	0:34:07	0:00:50	36	27	3
Spanish	spa	1:26:26	0:59:28	0:01:05	36	20	1
Swahili	swa	1:45:48	1:06:21	0:01:23	42	26	3
Tamil	tam	2:40:42	2:07:46	0:02:57	46	32	7
Tagalog	tgl	1:39:55	1:11:22	0:01:55	43	41	4
Thai	tha	2:52:03	2:54:49	0:04:08	49	39	8
Turkish	tur	1:24:19	1:12:30	0:01:32	40	28	5
Uzbek	uzb	2:04:26	1:53:27	0:02:57	42	29	5
Vietnamese	vie	1:21:48	0:53:42	0:01:37	37	28	3
Wolof	wol	2:00:43	1:24:01	0:01:21	37	25	2
Yoruba	yor	1:22:45	0:53:18	0:01:06	35	24	3
Zulu	zul	1:50:22	1:24:36	0:02:30	40	43	4
Hausa	hau	1:21:56	0:57:18	0:01:18	39	43	2
Kinyarwanda	kin	0:46:02	0:31:31	0:00:16	32	47	2
Oromo	orm	1:14:11	0:49:47	0:00:53	37	41	2
Sinhala	sin	1:23:55	0:32:50	0:00:13	31	45	1
Tigrinya	tir	1:52:57	0:35:37	0:00:40	36	42	2
Uyghur	uig	2:15:54	1:05:21	0:00:46	35	51	2

# Quantifying Performance

---

- We want to be able to say something rigorous about the performance of our learning algorithm.
- We will concentrate on discussing the number of examples one needs to see before we can say that our learned hypothesis is good.

# Learning Conjunctions

(recall that these are linear functions, and you can use SGD and many other functions to learn them; but the lessons of today are clearer with a simpler representation)

# Learning Conjunctions

- There is a hidden (monotone) conjunction the learner (you) is to learn
- $f(x_1, x_2, \dots, x_{100}) = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
- How many examples are needed to learn it? How?
  - **Protocol I:** The learner proposes instances as queries to the teacher
  - **Protocol II:** The teacher (who knows  $f$ ) provides training examples
  - **Protocol III:** Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels ( $f(x)$ )

# Learning Conjunctions (I)

---

- Protocol I: The learner proposes instances as queries to the teacher
- Note: we know we are after a monotone conjunction:



**Propose the first and second example you would use, or  
concisely describe your algorithm.**

# Learning Conjunctions (I)

- Protocol I: The learner proposes instances as queries to the teacher
- Since we know we are after a monotone conjunction:
- Is  $x_{100}$  in?  $\langle (1,1,1 \dots, 1,0), ? \rangle f(x) = 0$  (conclusion: Yes)
- Is  $x_{99}$  in?  $\langle (1,1, \dots 1,0,1), ? \rangle f(x) = 1$  (conclusion: No)
- Is  $x_1$  in?  $\langle (0,1, \dots 1,1,1), ? \rangle f(x) = 1$  (conclusion: No)
  
- A straight forward algorithm requires  $n = 100$  queries, and will produce as a result the hidden conjunction (exactly).
  - $h(x_1, x_2, \dots, x_{100}) = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

What happens here if the conjunction is not known to be monotone?  
If we know of a positive example, the same algorithm works.

# Learning Conjunctions(II)

---

- Protocol II: The teacher (who knows  $f$ ) provides training examples

# Learning Conjunctions (II)

---

- Protocol II: The teacher (who knows  $f$ ) provides training examples
- $\langle (0,1,1,1,1,0, \dots, 0,1), 1 \rangle$

# Learning Conjunctions (II)

---

- Protocol II: The teacher (who knows  $f$ ) provides training examples
- $\langle (0,1,1,1,1,0, \dots, 0,1), 1 \rangle$  (We learned a superset of the good variables)

# Learning Conjunctions (II)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol II: The teacher (who knows  $f$ ) provides training examples



**What would your strategy be if you are asked to TEACH this concept via examples? Describe your first example (and next, if possible)**

# Learning Conjunctions (II)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol II: The teacher (who knows  $f$ ) provides training examples
- $\langle (0,1,1,1,1,0, \dots, 0,1), 1 \rangle$  (We learned a superset of the good variables)
- To show you that all these variables are required...
  - $\langle (0,0,1,1,1,0, \dots, 0,1), 0 \rangle$  need  $x_2$
  - $\langle (0,1,0,1,1,0, \dots, 0,1), 0 \rangle$  need  $x_3$
  - ...
  - $\langle (0,1,1,1,1,0, \dots, 0,0), 0 \rangle$  need  $x_{100}$
- A straight forward algorithm requires  $k = 6$  examples to produce the hidden conjunction (exactly).
- $$h(x_1, x_2, \dots, x_{100}) = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Modeling Teaching Is tricky

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
- Teacher (Nature) provides the labels ( $f(x)$ )
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle$
  - $\langle (1,0,1,1,1,0, \dots, 0,1,1), 0 \rangle$
  - $\langle (1,1,1,1,1,0, \dots, 0,0,1), 1 \rangle$
  - $\langle (1,0,1,0,0,0, \dots, 0,1,1), 0 \rangle$
  - $\langle (1,1,1,1,1,1, \dots, 0,1), 1 \rangle$
  - $\langle (0,1,0,1,0,0, \dots, 0,1,1), 0 \rangle$
- How should we learn?



**Suggest a learning strategy: How would you use these examples to recover the hidden function? Are you guaranteed success?**

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels ( $f(x)$ )
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels ( $f(x)$ )
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle$

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels ( $f(x)$ )
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle$  ← learned nothing:  $h = x_1 \wedge x_2, \dots, \wedge x_{100}$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle$

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels ( $f(x)$ )
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle \leftarrow$  learned nothing:  $h = x_1 \wedge x_2, \dots, \wedge x_{100}$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle \leftarrow h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels ( $f(x)$ )
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle \leftarrow$  learned nothing:  $h = x_1 \wedge x_2, \dots, \wedge x_{100}$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle \leftarrow h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
  - $\langle (1,0,1,1,0,0, \dots, 0,0,1), 0 \rangle \leftarrow$  learned nothing
  - $\langle (1,1,1,1,1,0, \dots, 0,0,1), 1 \rangle$

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels ( $f(x)$ )
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle \leftarrow$  learned nothing:  $h = x_1 \wedge x_2, \dots, \wedge x_{100}$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle \leftarrow h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
  - $\langle (1,0,1,1,0,0, \dots, 0,0,1), 0 \rangle \leftarrow$  learned nothing
  - $\langle (1,1,1,1,1,0, \dots, 0,0,1), 1 \rangle \leftarrow h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels  $f(x)$
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle \leftarrow$  learned nothing:  $h = x_1 \wedge x_2, \dots, \wedge x_{100}$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle \leftarrow h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
  - $\langle (1,0,1,1,0,0, \dots, 0,0,1), 0 \rangle \leftarrow$  learned nothing
  - $\langle (1,1,1,1,1,0, \dots, 0,0,1), 1 \rangle \leftarrow h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
  - $\langle (1,0,1,0,0,0, \dots, 0,1,1), 0 \rangle$
  - $\langle (1,1,1,1,1,1, \dots, 0,1), 1 \rangle$
  - $\langle (0,1,0,1,0,0, \dots, 0,1,1), 0 \rangle$

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels  $f(x)$
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle$  ← learned nothing:  $h = x_1 \wedge x_2, \dots, \wedge x_{100}$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle$  ←  $h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$
  - $\langle (1,0,1,1,0,0, \dots, 0,0,1), 0 \rangle$  ← learned nothing
  - $\langle (1,1,1,1,1,0, \dots, 0,0,1), 1 \rangle$  ←  $h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
  - $\langle (1,0,1,0,0,0, \dots, 0,1,1), 0 \rangle$
  - $\langle (1,1,1,1,1,1, \dots, 0,1), 1 \rangle$
  - $\langle (0,1,0,1,0,0, \dots, 0,1,1), 0 \rangle$

- Is it good
- Performance ?
- # of examples ?

Final hypothesis:

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels  $f(x)$
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle$
  - $\langle (1,0,1,1,0,0, \dots, 0,0,1), 0 \rangle$
  - $\langle (1,1,1,1,1,0, \dots, 0,0,1), 1 \rangle$
  - $\langle (1,0,1,0,0,0, \dots, 0,1,1), 0 \rangle$
  - $\langle (1,1,1,1,1,1, \dots, 0,1), 1 \rangle$
  - $\langle (0,1,0,1,0,0, \dots, 0,1,1), 0 \rangle$

- Is it good
- Performance ?
- # of examples ?

- With the given data, we only learned an “approximation” to the true concept
- We don’t know **how many examples** we need to see to learn **exactly**. (do we care?)
- But we know that we can make a limited **# of mistakes**.

Final hypothesis:

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

# Two Directions

- Can continue to analyze the probabilistic intuition:
  - Never saw  $x_1 = 0$  in positive examples, maybe we'll never see it?
  - And if we will, it will be with small probability, so the concepts we learn may be pretty **good**
  - **Good**: in terms of performance on future data
  - PAC framework
- Mistake Driven Learning algorithms/On line algorithms
  - Now, we can only reason about #(mistakes), not #(examples)
    - any relations?
  - Update your hypothesis only when you make mistakes
    - Not all on-line algorithms are mistake driven, so performance measure could be different.

# On-Line Learning

---

- New learning algorithms
- (all learn a linear function over the feature space)
  - Perceptron (+ many variations)
  - General Gradient Descent view
- Issues:
  - Importance of Representation
  - Complexity of Learning
  - Idea of Kernel Based Methods
  - More about features

# You run an online learning algorithm on consistent data, is it clear that you can stop making mistakes at some point?

Yes, if the hypothesis space is finite (e.g., space of all conjunctions over  $n$  variables)

No. As more examples appear you will necessarily make more mistakes

Yes for finite hypothesis spaces, no for infinite hypothesis spaces (e.g, linear function)

Yes, also for some infinite hypothesis spaces

No idea how to think about it

# Administration (10/7/20)

Are we recording? YES!

Available on the web site

- Remember that all the lectures are available on the website **before the class**
  - **Go over it and be prepared**
  - **A new set of written notes will accompany most lectures, with some more details, examples and, (when relevant) some code. The first one is available today for this lecture.**
- **HW 1:** Was due on Monday.
  - We plan to complete grading it this week, in case you want to consider it before the drop deadline next week.
- **HW 2 is out.**
  - Covers: Linear classifiers; learning curves; an application to named entity recognition; domain adaptation.
  - **Start working on it now. Don't wait until the last day (or two) since it could take a lot of your time**
- PollEv: If you haven't done so, please register <https://pollev.com/cis519/register>
  - We still have some problems with missing participation for students
- Quizzes: <https://canvas.upenn.edu/courses/1546646/quizzes/2475913/statistics>
- Go to the recitations and office hours
- Questions? **Please ask/comment during class; give us feedback**

# How many hours did you spend on HW1?

Less than 5 hours

5-10 hours

10-15 hours

15-20 hours

More than 20 hours

# Practical Machine Learning

The following question came up in my office hour:  
Why did you talk about the Teaching and Active Learning protocols. After all, this is not what Machine Learning is about, right? **WRONG!**

- You work for an E-discovery company that handles documents for large corporations.
- A corporation asks you for a reliable Text Classification program that can identify harassing e-mail (**H-mail**) messages sent by/to their employees within the organization (they don't want to be sued).
- **Problem:** The harassing emails constitute a **tiny fraction** of the 10k/day email traffic.
  - How will you develop a reliable classifier?
- **Active Learning Protocol:**
- Assume that you already have a “reasonable” model that can classify an e-mail message as **H-mail/Not**
  - Get some experts from the HR office in the company. Get them to use your model.
  - The model will output its guesses; hopefully, many of them will indeed “reasonable”. The HR experts will classify them. This will allow you to, relatively quickly get a good enough collection of H-mails, along with hard negative examples.
  - Now you can train a model.
  - **Questions:**
    - How will your learning algorithm decide what examples to present to the HR experts? (key Active Learning Question)
    - How will you initialize the model so it doesn't present only negative examples (which is the vast majority of examples)?
- **Teaching Protocol:**
  - Ask the HR experts to “dream up” some examples of the rare H-mails, and maybe some hard-negative examples.
  - They can use real examples they have seen, but can also invent some, given that they understand the concepts.
  - This “**teaching**” process is essential to start the learning process.
  - **Questions:**
    - Can we teach in a more efficient way? (e.g., via definitions)?

The example protocols discussed last time were simplified and idealized, but very realistic in applications.

# Learning Conjunctions (III)

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- Protocol III: Some random source (e.g., Nature) provides training examples
  - Teacher (Nature) provides the labels  $f(x)$
- Algorithm: Elimination
  - Start with the set of all literals as candidates
  - Eliminate a literal that is not active (0) in a positive example
  - $\langle (1,1,1,1,1,1, \dots, 1,1), 1 \rangle$
  - $\langle (1,1,1,0,0,0, \dots, 0,0), 0 \rangle$
  - $\langle (1,1,1,1,1,0, \dots, 0,1,1), 1 \rangle$
  - $\langle (1,0,1,1,0,0, \dots, 0,0,1), 0 \rangle$
  - $\langle (1,1,1,1,1,0, \dots, 0,0,1), 1 \rangle$
  - $\langle (1,0,1,0,0,0, \dots, 0,1,1), 0 \rangle$
  - $\langle (1,1,1,1,1,1, \dots, 0,1), 1 \rangle$
  - $\langle (0,1,0,1,0,0, \dots, 0,1,1), 0 \rangle$

- Is it good
- Performance ?
- # of examples ?

- With the given data, we only learned an “approximation” to the true concept
- We don’t know **how many examples** we need to see to learn **exactly**. (do we care?)
- But we know that we can make a limited **# of mistakes**.

Final hypothesis:

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

# Two Directions

---

- Can continue to analyze the probabilistic intuition:
  - Never saw  $x_1 = 0$  in positive examples, maybe we'll never see it?
  - And if we will, it will be with small probability, so the concepts we learn may be pretty **good**
  - **Good**: in terms of performance on future data
  - PAC framework
- Mistake Driven Learning algorithms/On line algorithms
  - Now, we can only reason about #(mistakes), not #(examples)
    - any relations?
  - Update your hypothesis only when you make mistakes
    - Not all on-line algorithms are mistake driven, so performance measure could be different.

# Generic Mistake Bound Algorithms

- Is it clear that we can bound the number of mistakes ?
- Let  $\mathcal{C}$  be a finite concept class. Learn  $f \in \mathcal{C}$
- **CON algorithm:**
  - In the  $i$ th stage of the algorithm:
    - $\mathcal{C}_i$ : all concepts in  $\mathcal{C}$  consistent with all  $i - 1$  previously seen examples
  - Choose randomly  $f \in \mathcal{C}_i$  and use it to predict the next example
  - Clearly,  $\mathcal{C}_{i+1} \subseteq \mathcal{C}_i$  and, if a mistake is made on the  $i$ -th example, then  $|\mathcal{C}_{i+1}| < |\mathcal{C}_i|$ , so progress is made.
- The CON algorithm makes at most  $|\mathcal{C}| - 1$  mistakes
- Can we do better ?

The goal of the following discussion is to think about hypothesis spaces, and some “optimal” algorithms, as a way to understand what might be possible.

For this reason, here we think about the class of possible target functions and the class of hypothesis as the same.

# The Halving Algorithm

- Let  $\mathcal{C}$  be a finite concept class. Learn  $f \in \mathcal{C}$
- **Halving Algorithm:**
  - In the  $i$ th stage of the algorithm:
    - $\mathcal{C}_i$ : all concepts in  $\mathcal{C}$  consistent with all  $i - 1$  previously seen examples
  - Given an example  $e_t$  consider the value  $f_j(e_t)$  for all  $f_j \in \mathcal{C}_i$  and predict by majority.
  - Predict 1 iff
    - $|\{f_j \in \mathcal{C}_i ; f_j(e_i) = 0\}| < |\{f_j \in \mathcal{C}_i ; f_j(e_i) = 1\}|$
  - Clearly,  $\mathcal{C}_{i+1} \subseteq \mathcal{C}_i$  and, if a mistake is made on the  $i$ -th example, then  $|\mathcal{C}_{i+1}| < \frac{1}{2} |\mathcal{C}_i|$ , so progress is made
- The Halving algorithm makes at most  $\log(|\mathcal{C}|)$  mistakes
  - Of course, this is a theoretical algorithm; can this be achieved with an efficient algorithm?

# Learning Conjunctions

Can this bound be achieved?

- There is a hidden conjunctions the learner is to learn
- $f(x_1, x_2, \dots, x_{100}) = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$
- The number of (all; not monotone) conjunctions:  $|C| = 3^n$
- $\log(|C|) = n$
- The elimination algorithm makes  $n$  mistakes
- ...
- k-conjunctions:
  - Assume that only  $k \ll n$  attributes occur in the disjunction
- The number of k-conjunctions:  $|C| = \binom{n}{k} 2^k$ 
  - $\log(|C|) = k \log n$
  - Can we learn efficiently with this number of mistakes ?

Can mistakes be bounded in the **non-finite case**?

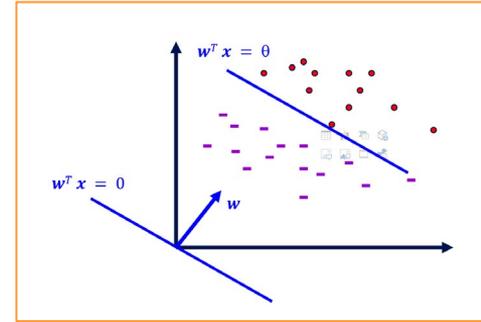
Earlier:

- Talked about various learning protocols & on algorithms for conjunctions.
- Discussed the performance of the algorithms in terms of bounding the **number of mistakes** that algorithm makes.
- Gave a “theoretical” algorithm with  $\log |C|$  mistakes.

# Linear Threshold Functions

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \cdot \mathbf{x} - \theta) = \text{sgn}\left\{\sum_{i=1}^n w_i x_i - \theta\right\}$$

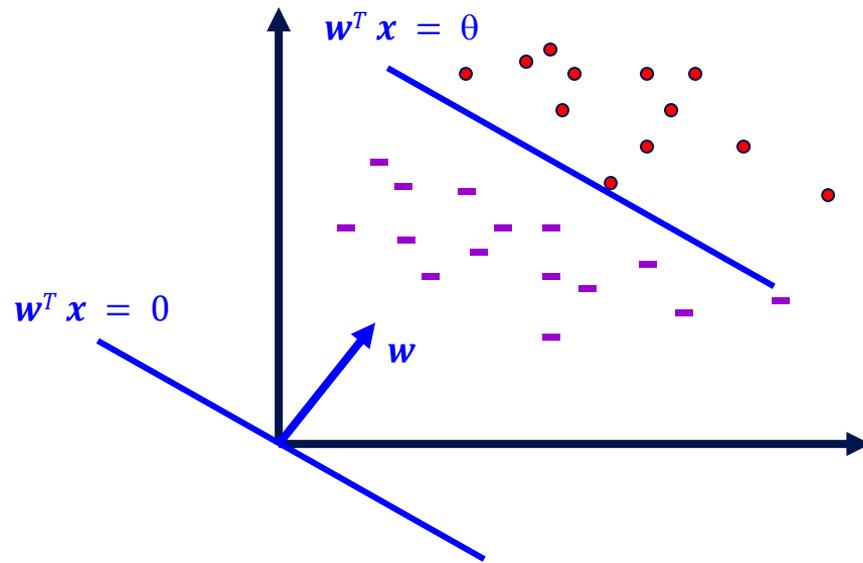
- Many functions are Linear
  - Conjunctions:
    - $y = x_1 \wedge x_3 \wedge x_5$
    - $y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 3\}$   $\mathbf{w} = (1, 0, 1, 0, 1)$   $\theta = 3$
  - At least m of n:
    - $y = \text{at least } 2 \text{ of } \{x_1, x_3, x_5\}$
    - $y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 2\}$   $\mathbf{w} = (1, 0, 1, 0, 1)$   $\theta = 2$
- Many functions are not
  - Xor:  $y = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$
  - Non trivial DNF:  $y = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$
- But can be made linear
- Note: all the variables above are Boolean variables



# Linear Threshold Functions

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \cdot \mathbf{x} - \theta) = \text{sgn}\{\sum_{i=1}^n w_i x_i - \theta\}$$

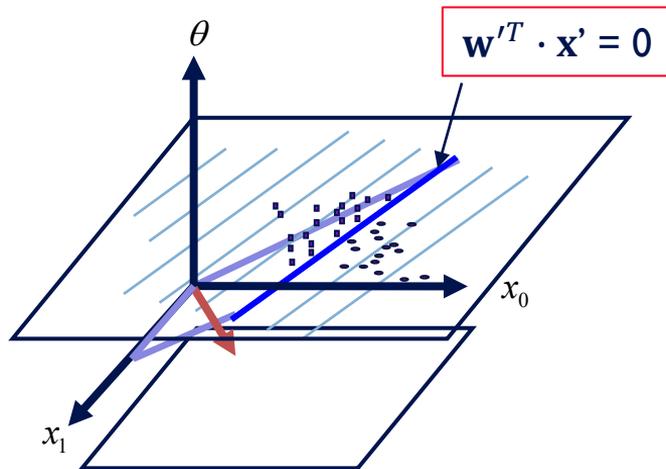
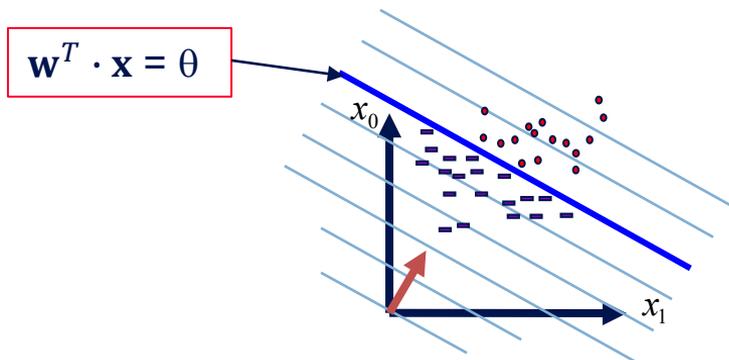
- Many functions are Linear
  - Conjunctions:
    - $y = x_1 \wedge x_3 \wedge x_5$
    - $y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 3\}$   $\mathbf{w} = (1, 0, 1, 0, 1)$   $\theta = 3$
- In our Elimination Algorithm we started with:
  - $\mathbf{w} = (1, 1, 1, 1, 1, n)$  and changed  $w_i$  from 1  $\rightarrow$  0 following a mistake
- In general, when learning linear functions, we will change  $w_i$  more carefully
  - $y = \text{sgn}\{w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4 + w_5 \cdot x_5 - \theta\}$



# Canonical Representation

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \cdot \mathbf{x} - \theta) = \text{sgn}\{\sum_{i=1}^n w_i x_i - \theta\}$$

- **Note:**  $\text{sgn}(\mathbf{w}^T \cdot \mathbf{x} - \theta) = \text{sgn}\{\mathbf{w}'^T \cdot \mathbf{x}'\}$ 
  - Where  $\mathbf{x}' = (\mathbf{x}, -1)$  and  $\mathbf{w}' = (\mathbf{w}, \theta)$
- Moved from an  $n$  dimensional representation to an  $(n + 1)$  dimensional representation, but now can look for hyperplanes that go through the origin.
- Basically, that means that we learn both  $\mathbf{w}$  and  $\theta$



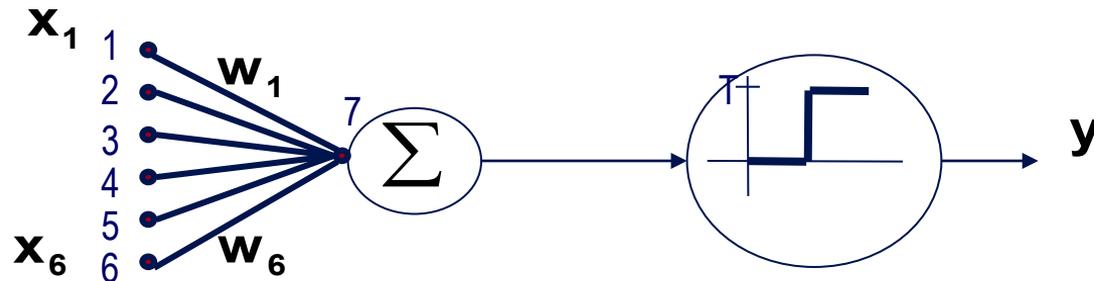
# Perceptron

## Earlier:

- Talked about various learning protocols & on algorithms for conjunctions.
- Measured performance of algorithms in terms of bounding the **number of mistakes** that algorithm makes.
- Showed that in the case of a **finite concept** class we will stop making mistakes at some point (a “theoretical” algorithm had  $\log |C|$  mistakes).

# 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)



# Perceptron learning rule

- We learn  $f: \mathbf{X} \rightarrow Y = \{-1, +1\}$  represented as  $f = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\}$
- Where  $\mathbf{X} = \{0, 1\}^n$  or  $\mathbf{X} = \mathbf{R}^n$  and  $\mathbf{w} \in \mathbf{R}^n$
- Given Labeled examples:  $\{\{\mathbf{x}_1, y_1\}, \{\mathbf{x}_2, y_2\}, \dots, \{\mathbf{x}_m, y_m\}\}$

1. Initialize  $\mathbf{w} \in \mathbf{R}^n$

2. Cycle through all examples [multiple times]

- a. Predict the label of instance  $\mathbf{x}$  to be  $y' = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\}$

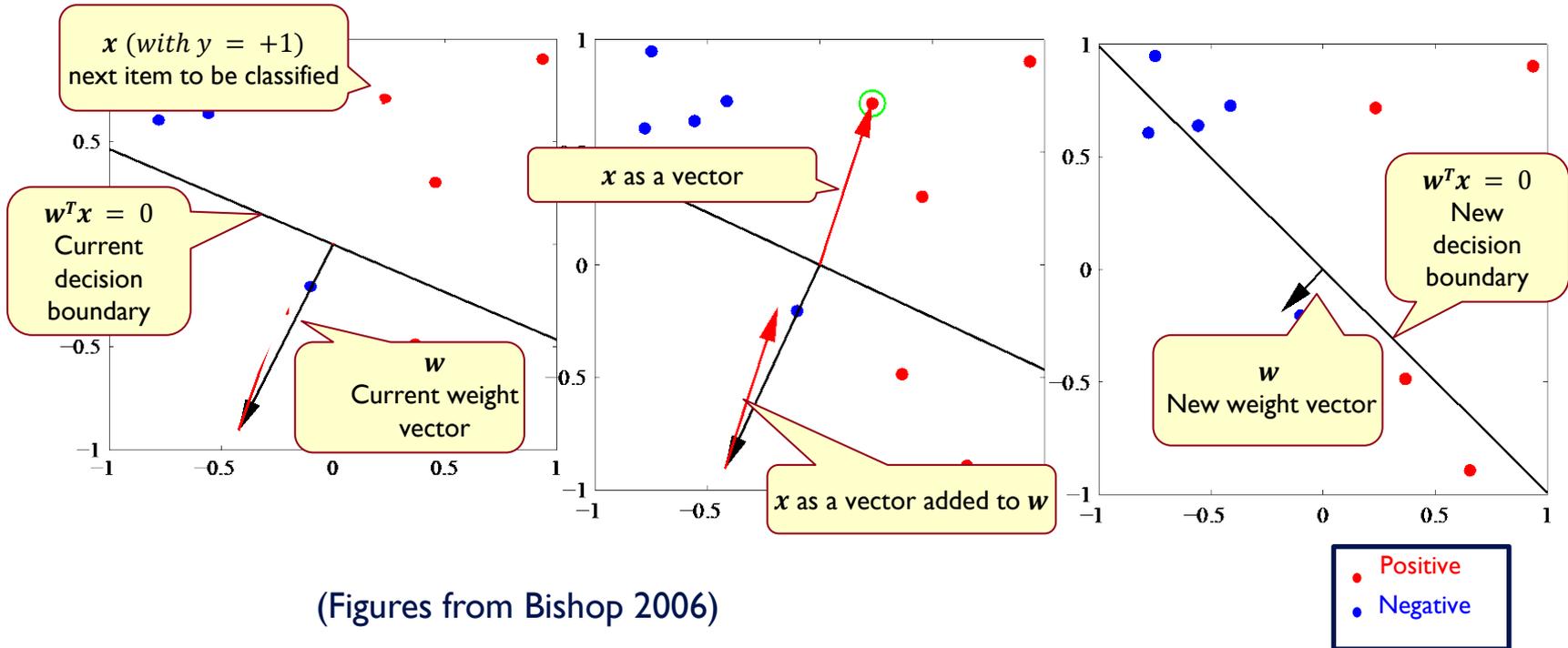
- b. If  $y' \neq y$ , **update** the weight vector:

$$\mathbf{w} = \mathbf{w} + r y \mathbf{x} \quad (r - \text{a constant, learning rate})$$

Otherwise, if  $y' = y$ , leave weights unchanged.

# Perceptron in action

1. Initialize  $w \in R^n$
2. Cycle through all examples [multiple times]
  - a. Predict the label of instance  $x$  to be  $y' = \text{sgn}\{w^T \cdot x\}$
  - b. If  $y' \neq y$ , update the weight vector:  
 $w = w + ryx$  ( $r$  - a constant, learning rate)  
Otherwise, if  $y' = y$ , leave weights unchanged.

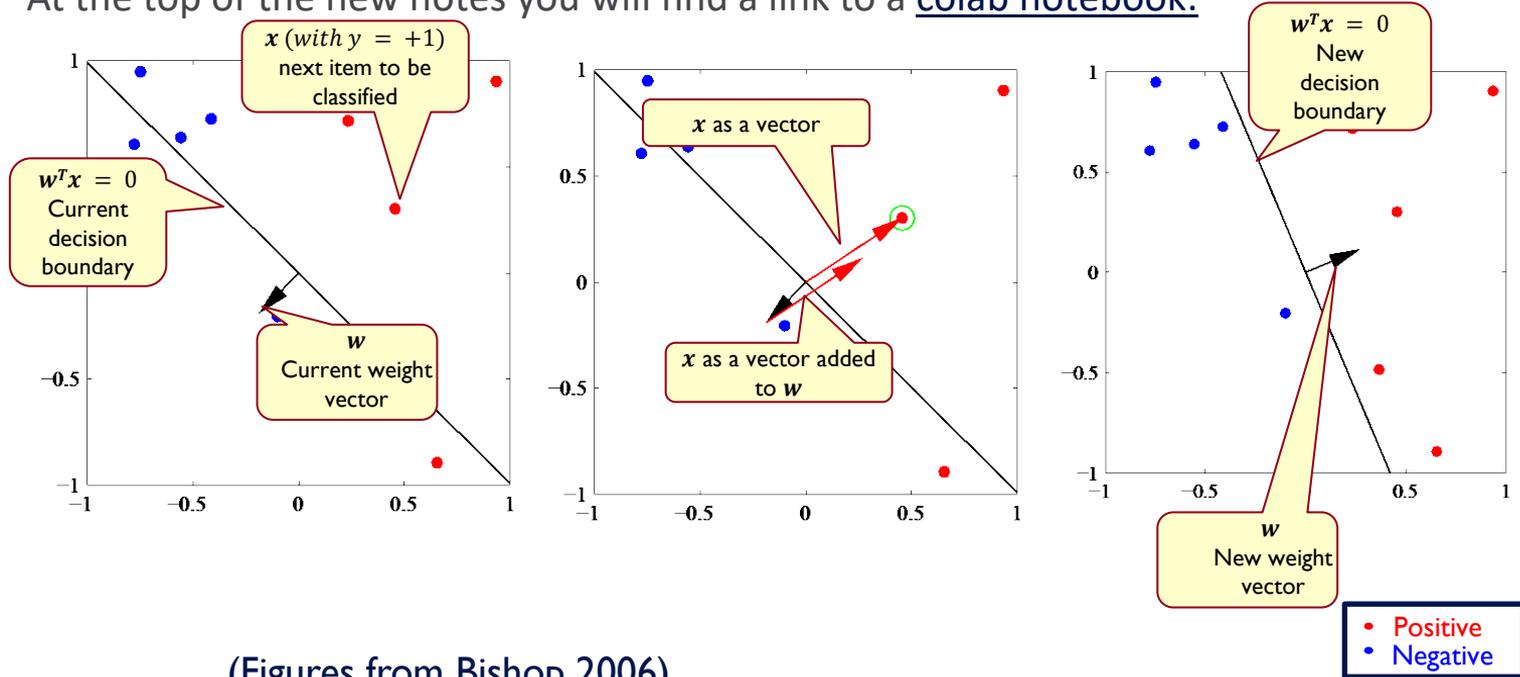


(Figures from Bishop 2006)

# Perceptron in action

1. Initialize  $w \in R^n$
2. Cycle through all examples [multiple times]
  - a. Predict the label of instance  $x$  to be  $y' = \text{sgn}\{w^T \cdot x\}$
  - b. If  $y' \neq y$ , **update** the weight vector:  
 $w = w + ryx$  ( $r$  - a constant, learning rate)  
Otherwise, if  $y' = y$ , leave weights unchanged.

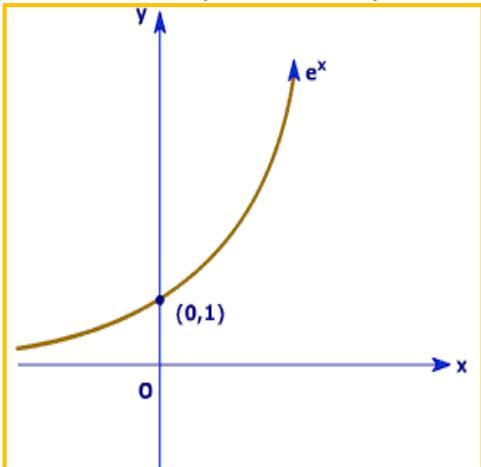
- At the top of the new notes you will find a link to a [colab notebook](#):



(Figures from Bishop 2006)

# Perceptron learning rule

- If  $x$  is Boolean, only weights of **active features** are updated
- Why is this important?



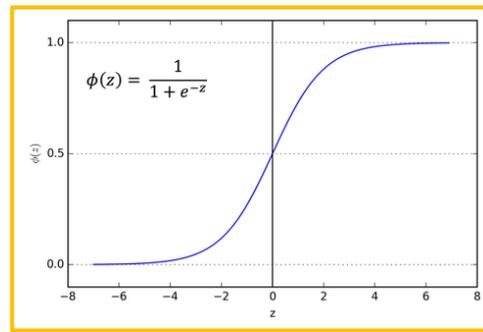
1. Initialize  $w \in \mathbb{R}^n$
2. Cycle through all examples [multiple times]
  - a. Predict the label of instance  $x$  to be  $y' = \text{sgn}\{w^T \cdot x\}$
  - b. If  $y' \neq y$ , **update** the weight vector:

$$w = w + ryx \quad (r - \text{a constant, learning rate})$$

Otherwise, if  $y' = y$ , leave weights unchanged.

- $w^T \cdot x > 0$  is equivalent to:

$$\frac{1}{1+e^{-w^T x}} > \frac{1}{2}$$



# Perceptron Learnability

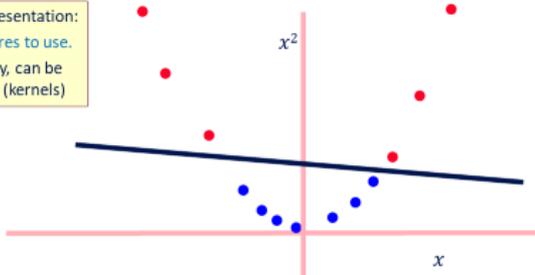
---

- Obviously can't learn what it can't represent (???)
  - Only linearly separable functions
- Minsky and Papert (1969) wrote an influential book demonstrating Perceptron's representational limitations
  - Parity functions can't be learned (XOR)
  - In vision, if patterns are represented with local features, can't represent symmetry, connectivity
- Research on Neural Networks stopped for years
- Perceptron
- Rosenblatt himself (1959) asked,
  - “What pattern recognition problems can be transformed so as to become linearly separable?”

## Blown Up Feature Space

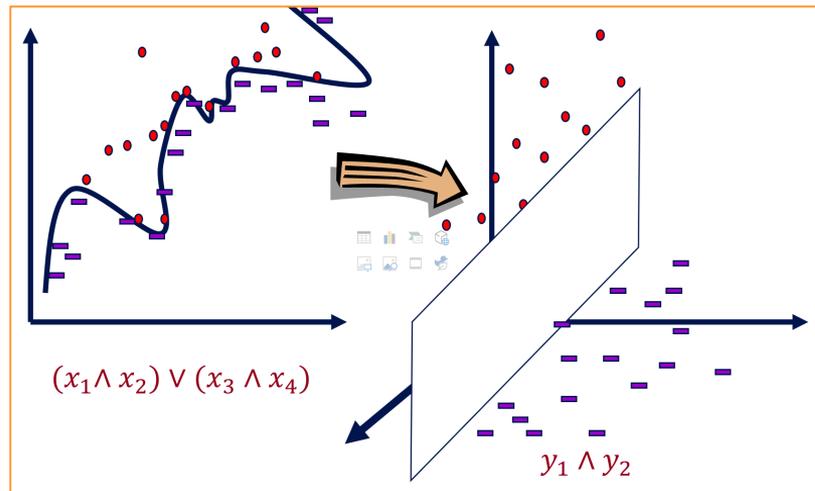
- But, we can change the way we **represent** the data
- Data are separable in  $\langle x, x^2 \rangle$  space

- Key issue: Representation:
  - what features to use.
- Computationally, can be done implicitly (kernels)



CIS 419/519 Fall'2020

78



# Perceptron Convergence

---

## – Perceptron Convergence Theorem:

- If there exist a set of weights that are consistent with the data (i.e., the data is linearly separable), the perceptron learning algorithm will converge
- How long would it take to converge ?

## – Perceptron Cycling Theorem:

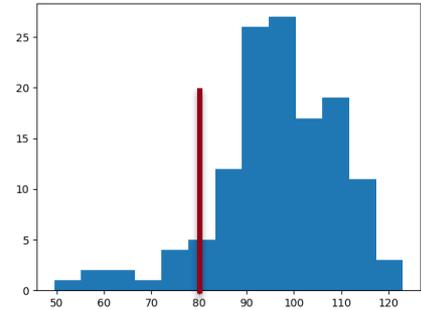
- If the training data is not linearly separable the perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop.
- How to provide robustness, more expressivity ?

# Administration (10/12/20)

Are we recording? YES!

Available on the web site

- Remember that all the lectures are available on the website **before the class**
  - Go over it and be prepared
  - A new set of written notes will accompany most lectures, with some more details, examples and, (when relevant) some code. The first one is available today for this lecture.
- **HW 1:** Avg. 90.5/100; overall, 96.8 (with EC)
- **HW 2:** Due date extended to 10/22
- **No class on Wednesday** (see my Piazza message last night)
- Quizzes: <https://canvas.upenn.edu/courses/1546646/quizzes/2480748/statistics>
- **Mid-term is on 10/28**; at the class time. If you have a time zone problem – email me.
- Questions? Please ask/comment during class; give us feedback



# Where Are We?

---

- Discussed On-Line Learning
  - Learning by updating the hypothesis after each observed example
  - Specifically: Perceptron
    - Mistake driven learning: updates are done only when a mistake is made
- Today:
  - Continue with Perceptron and some variations
  - Understand Perceptron as a special case of SGD
  - More advanced update rules

# Perceptron learning rule

- We learn  $f: \mathbf{X} \rightarrow Y = \{-1, +1\}$  represented as  $f = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\}$
- Where  $\mathbf{X} = \{0, 1\}^n$  or  $\mathbf{X} = \mathbf{R}^n$  and  $\mathbf{w} \in \mathbf{R}^n$
- Given Labeled examples:  $\{\{\mathbf{x}_1, y_1\}, \{\mathbf{x}_2, y_2\}, \dots \{\mathbf{x}_m, y_m\}\}$

1. Initialize  $\mathbf{w} \in \mathbf{R}^n$

2. Cycle through all examples [multiple times]

- a. Predict the label of instance  $\mathbf{x}$  to be  $y' = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\}$

- b. If  $y' \neq y$ , **update** the weight vector:

$$\mathbf{w} = \mathbf{w} + r y \mathbf{x} \quad (r - \text{a constant, learning rate})$$

Otherwise, if  $y' = y$ , leave weights unchanged.

# Perceptron in action

1. Initialize  $w \in R^n$

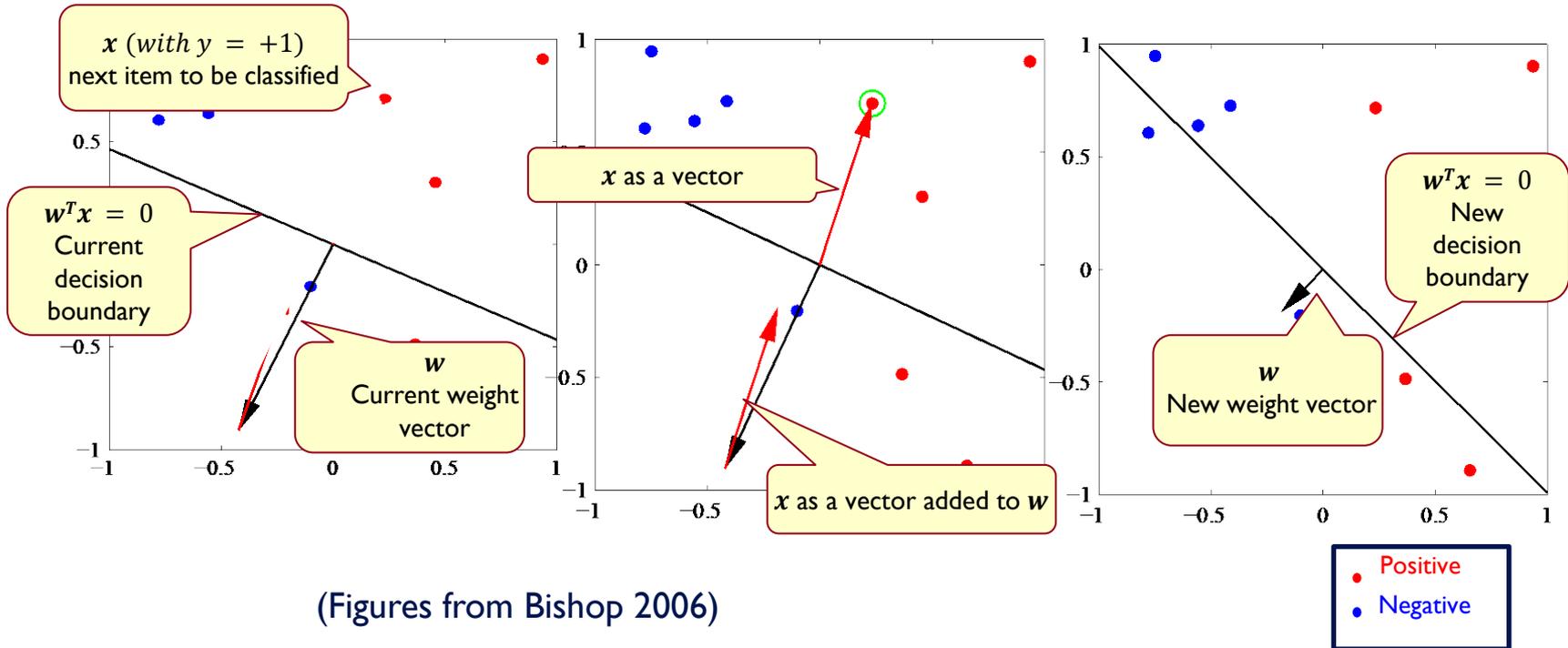
2. Cycle through all examples [multiple times]

a. Predict the label of instance  $x$  to be  $y' = \text{sgn}\{w^T \cdot x\}$

b. If  $y' \neq y$ , update the weight vector:

$$w = w + ryx \quad (r - \text{a constant, learning rate})$$

Otherwise, if  $y' = y$ , leave weights unchanged.



# Perceptron

Input set of examples and their labels

$$Z = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)) \in \mathbf{R}^n \times \{-1, 1\}^m, \eta, \theta_{Init}$$

- Initialize  $\mathbf{w} \leftarrow 0$  and  $\theta \leftarrow \theta_{Init}$
- For every training epoch:
- for every  $\mathbf{x}_j \in X$ :
  - $\hat{y} \leftarrow \text{sign}(\langle \mathbf{w}, \mathbf{x}_j \rangle - \theta)$
  - If  $(\hat{y} \neq y_j)$ 
    - $\mathbf{w} \leftarrow \mathbf{w} + \eta y_j \mathbf{x}_j$
    - $\theta \leftarrow \theta + \eta y_j$

Just to make sure we understand that we learn both  $\mathbf{w}$  and  $\theta$

# Perceptron: Mistake Bound Theorem

- Maintains a weight vector  $\mathbf{w} \in \mathbf{R}^n$ ,  $\mathbf{w}_0 = (0, \dots, 0)$ .
- Upon receiving an example  $\mathbf{x} \in \mathbf{R}^n$
- Predicts according to the linear threshold function  $\mathbf{w}^T \cdot \mathbf{x} \geq 0$ .
- **Theorem [Novikoff,1963]**

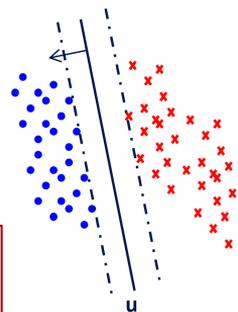
size of  $\mathbf{u}$ ;  $L_2$  norm

$$\|\mathbf{u}\| = \sqrt{\sum_{i=1}^n u_i^2}$$

Let  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_t, y_t)$  be a sequence of labeled examples with  $\mathbf{x}_i \in \mathbf{R}^n$ ,  $\|\mathbf{x}_i\| \leq R$  and  $y_i \in \{-1, 1\}$  for all  $i$ . Let  $\mathbf{u} \in \mathbf{R}^n$ ,  $\gamma > 0$  be such that,  $\|\mathbf{u}\| = 1$  and  $y_i \mathbf{u}^T \cdot \mathbf{x}_i \geq \gamma$  for all  $i$

complexity parameter  
(margin)

- Then Perceptron makes at most  $\frac{R^2}{\gamma^2}$  mistakes on this example sequence.
- (see additional notes)



# Perceptron-Mistake Bound

**Proof:** Let  $v_k$  be the hypothesis after the  $k$ -th mistake. Assume that the  $k$ -th mistake occurs on the input example  $(x_i, y_i)$ .

$$\therefore y_i(\vec{v}_k \cdot \vec{x}_i) \leq 0.$$

Assumptions

$$\vec{v}_1 = \mathbf{0}$$

$$\|\vec{u}\| = 1$$

$$y_i \vec{u}^T \cdot \vec{x}_i \geq \gamma$$

$$\vec{v}_{k+1} = \vec{v}_k + y_i \vec{x}_i$$

$$\begin{aligned} \vec{v}_{k+1} \cdot \vec{u} &= \vec{v}_k \cdot \vec{u} + y_i(\vec{u} \cdot \vec{x}_i) \\ &\geq \vec{v}_k \cdot \vec{u} + \gamma \end{aligned}$$

$$\therefore \vec{v}_{k+1} \cdot \vec{u} \geq k\gamma$$

$$\begin{aligned} \|\vec{v}_{k+1}\|^2 &= \|\vec{v}_k\|^2 + 2y_i(\vec{v}_k \cdot \vec{x}_i) + \|\vec{x}_i\|^2 \\ &\leq \|\vec{v}_k\|^2 + R^2 \end{aligned}$$

$$\therefore \|\vec{v}_{k+1}\|^2 \leq kR^2$$

Therefore,

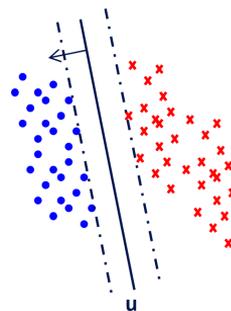
$$\underline{\sqrt{k}R} \geq \|\vec{v}_{k+1}\| \geq \vec{v}_{k+1} \cdot \vec{u} \geq \underline{k\gamma}.$$



$$k < R^2 / \gamma^2$$

The second inequality follows because  $\|\vec{u}\| \leq 1$ .

1. Note that the bound does not depend on the **dimensionality** nor on the **number of examples**.
2. Note that we place **weight vectors** and **examples** in the same space.
3. Interpretation of the theorem



# Robustness to Noise

- In the case of non-separable data, the extent to which a data point fails to have margin  $\gamma$  via the hyperplane  $\mathbf{w}$  can be quantified by a slack variable

$$\xi_i = \max(0, \gamma - y_i \mathbf{w}^T \mathbf{x}_i)$$

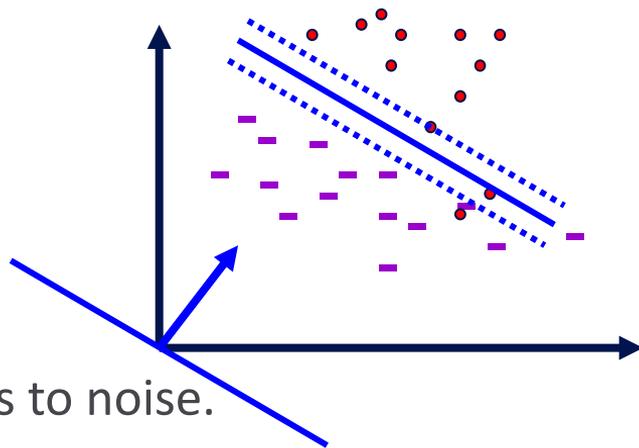
- Observe that when  $\xi_i = 0$ , the example  $\mathbf{x}_i$  has margin at least  $\gamma$ . Otherwise, it grows linearly with  $-y_i \mathbf{w}^T \mathbf{x}_i$

- Denote:  $D_2 = [\sum \{\xi_i\}^2]^{\frac{1}{2}}$

- Theorem:**

- The perceptron is guaranteed to make no more than  $(\frac{R+D_2}{\gamma})^2$  mistakes on any sequence of examples satisfying  $\|\mathbf{x}_i\|^2 < R$

- Perceptron is expected to have some robustness to noise.



# Perceptron for Boolean Functions

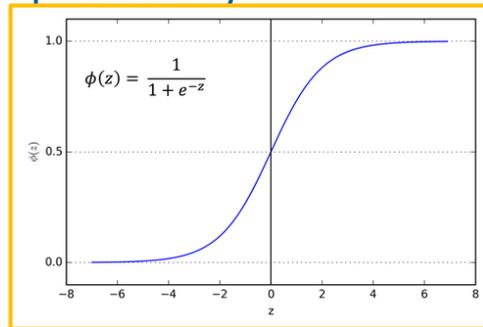
- How many mistakes will the Perceptron algorithms make when learning a  $k$ -disjunction?
- Try to figure out the bound
- Find a sequence of examples that will cause Perceptron to make  $O(n)$  mistakes on  $k$ -disjunction on  $n$  attributes.
  - (Where is  $n$  coming from?)
- Recall that **halving** suggested the possibility of a better bound –  $k \log(n)$
- This can be achieved by Winnow
  - A multiplicative update algorithm [Littlestone'88]
  - See HW2

# Practical Issues and Extensions

- There are many extensions that can be made to these basic algorithms.
- Some are necessary for them to perform well
  - Regularization (next; will be motivated in the next section, COLT)
- Some are for ease of use and tuning
  - Converting the output of a Perceptron to a conditional probability

$$P(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-Aw^T\mathbf{x}}}$$

- The parameter  $A$  can be tuned on a development set



$$\frac{1}{1 + e^{-w^T\mathbf{x}}} > \frac{1}{2}$$

# A Learning Scenario

---

- You are running Perceptron on a stream of random examples.
- $h_1$  made a mistake on the first example it saw.
- $h_2$  ran through 9 examples and only made a mistake on the 10<sup>th</sup>.
- $h_3$  .....
- $h_{98}$  made a mistake only on the 1000<sup>th</sup> example it saw.
- $h_{99}$  made a mistake on the first example it saw.
- $h_{100}$  made a mistake on the 10th example it saw.
- Now, you need to stop training and output the final hypothesis.
- Which one will you output?
-



**You need to stop training and output your final hypothesis;  
which one will you output?**

# Regularization Via Averaged Perceptron

- An **Averaged Perceptron Algorithm** is motivated by the following considerations:
  - In real life, we want [more] guarantees from our learning algorithm
  - In the mistake bound model:
    - We don't know **when** we will make the mistakes.
    - In a sequential/on-line scenario, which hypothesis will you choose...??
    - Being consistent with more examples is better (why?)
  - Ideally, we want to quantify the **expected performance** as a function of the number of examples seen and **not** number of mistakes. (“a global guarantee”; the PAC model does that)
  - Every Mistake-Bound Algorithm can be converted efficiently to yield such a guarantee.
- To convert a given Mistake Bound algorithm (into a global guarantee algorithm):
  - Wait for a long stretch w/o mistakes (there must be one)
  - Use the hypothesis at the end of this stretch.
  - Its PAC behavior is relative to the length of the stretch.
- Averaged Perceptron returns a weighted average of earlier hypotheses; the weights are a function of the length of no-mistakes stretch.

# Regularization Via Averaged Perceptron

- **Variables:**
  - $m$ : number of examples
  - $k$ : number of mistakes
  - $c_i$ : consistency count for hypothesis  $\mathbf{v}_i$
  - $T$ : number of epochs
- **Input:** a labeled training  $\{\{\mathbf{x}_1, y_1\}, \{\mathbf{x}_2, y_2\}, \dots, \{\mathbf{x}_m, y_m\}\}$
- **Output:** a list of weighted perceptrons  $\{\{\mathbf{v}_1, c_1\}, \dots, \{\mathbf{v}_k, c_k\}\}$
- **Initialize:**  $k = 0, \mathbf{v}_1 = 0, c_1 = 0$
- Repeat  $T$  times:
  - For  $i = 1, \dots, m$ ;
  - Compute prediction  $y' = \text{sgn}(\mathbf{v}_k^T \cdot \mathbf{x}_i)$
  - If  $y' = y$ , then  $c_k = c_k + 1$   
else:  $\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i; c_{k+1} = 1; k = k + 1$
- **Prediction:**
  - **Given:** a list of weighted perceptrons  $\{\{\mathbf{v}_1, c_1\}, \dots, \{\mathbf{v}_k, c_k\}\}$ ; a new example  $\mathbf{x}$
  - **Predict** the label ( $\mathbf{x}$ ) as follows:  $y(\mathbf{x}) = \text{sgn}[\sum_1^k c_i (\mathbf{v}_i^T \mathbf{x})]$

- This can be done on top of any online mistake driven algorithm.
- In HW 2 you will run it over three different algorithms.
- The implementation requires thinking.

Averaged version of Perceptron/Winnow is as good as any other linear learning algorithm, if not better.

# Perceptron with Margin

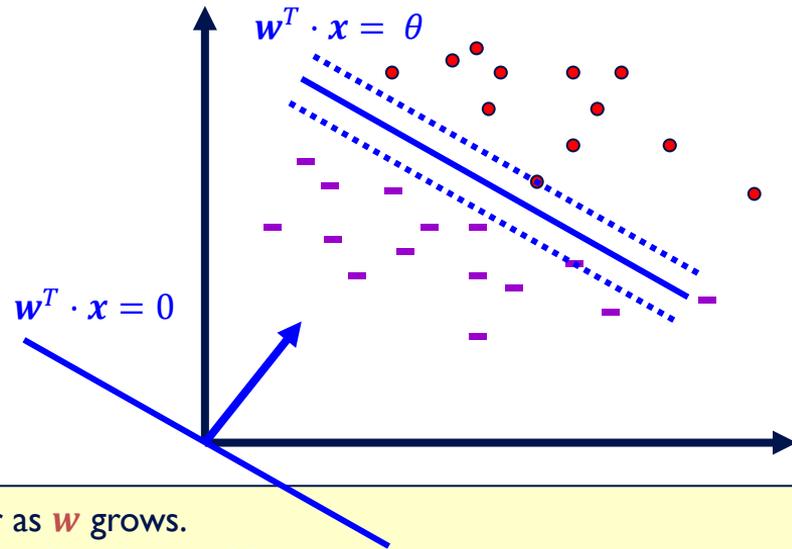
- Thick Separator (aka as Perceptron with Margin)  
(Applies both for Perceptron and Winnow)

- Promote weights if:

- $w^T \cdot x - \theta < \gamma$

- Demote weights if:

- $w^T \cdot x - \theta > \gamma$



Note:  $\gamma$  is a functional margin. Its effect could disappear as  $w$  grows.

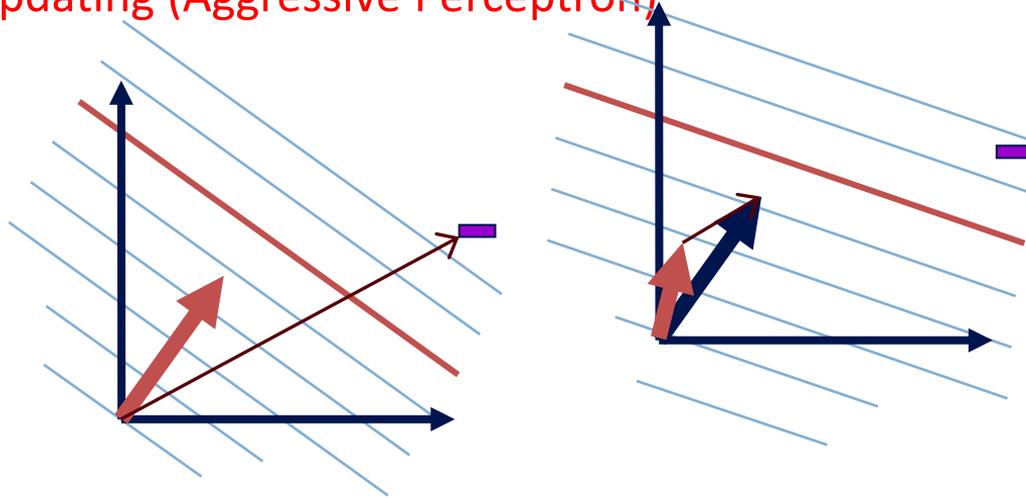
Nevertheless, this has been shown to be a very effective algorithmic addition. (Grove & Roth 98,01; Karov et. al 97)

# Other Extensions

- Assume you made a mistake on example  $x$ .
- You then see example  $x$  again; will you make a mistake on it?
- **Threshold relative updating (Aggressive Perceptron)**

- $w \leftarrow w + rx$

- $r = \frac{\theta - w^T x}{\|x\|^2}$



- Equivalent to updating on the same example multiple times

# So Far

---

- We have presented several **on-line** linear learning algorithms
  - **SGD**
    - Remember: this algorithm depended on the loss function chosen.
  - **Perceptron**
- What is the difference between the versions we have seen?
- Are there any similarities?

# General Stochastic Gradient Algorithms

- Given examples  $\{z = (\mathbf{x}, y)\}_{1,m}$  from a distribution over  $X \times Y$ , we are trying to learn a linear function, parameterized by a weight vector  $\mathbf{w}$ , so that we minimize the expected risk function

The loss  $Q$ : a function of  $\mathbf{x}$ ,  $\mathbf{w}$  and  $y$

$$J(\mathbf{w}) = E_z Q(\mathbf{z}, \mathbf{w}) \approx \frac{1}{m} \sum_{i=1}^m Q(\mathbf{z}_i, \mathbf{w}_i)$$

- In Stochastic Gradient Descent Algorithms we approximate this minimization by incrementally updating the weight vector  $\mathbf{w}$  as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r_t \mathbf{g}_w Q(\mathbf{z}_t, \mathbf{w}_t) = \mathbf{w}_t - r_t \mathbf{g}_t$$

- Where  $\mathbf{g}_t = \mathbf{g}_w Q(\mathbf{z}_t, \mathbf{w}_t)$  is the gradient with respect to  $\mathbf{w}$  at time  $t$ .
- The difference between algorithms now amounts to choosing a different loss function  $Q(\mathbf{z}, \mathbf{w})$

# General Stochastic Gradient Algorithms

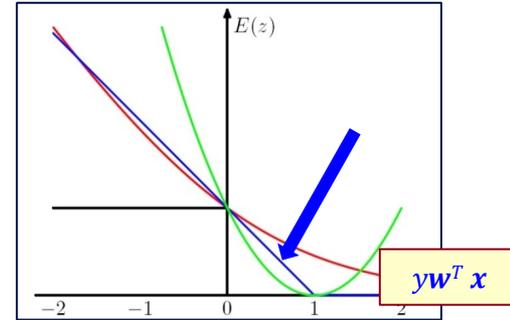
Learning rate

gradient

The loss  $Q$ : a function of  $x, w$  and  $y$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r_t \mathbf{g}_w Q(\mathbf{x}_t, y_t, \mathbf{w}_t) = \mathbf{w}_t - r_t \mathbf{g}$$

- **LMS**:  $Q((\mathbf{x}, y), \mathbf{w}) = \frac{1}{2}(y - \mathbf{w}^T \mathbf{x})^2$
- Computing the gradient leads to the update rule (Also called Widrow's Adaline):  
$$\mathbf{w}_{t+1} = \mathbf{w}_t + r(y_t - \mathbf{w}_t^T \mathbf{x}_t) \mathbf{x}_t$$
- Here, even though we make binary **predictions** based on  $\text{sgn}(\mathbf{w}^T \mathbf{x})$  we do not take the sign of the dot-product into account in the loss.
- Another common loss function is:
- **Hinge loss**:  
$$Q((\mathbf{x}, y), \mathbf{w}) = \max(0, 1 - y \mathbf{w}^T \mathbf{x})$$
- Computing the gradient leads to the perceptron update rule:
- $\mathbf{g} = 0$  if  $y_i \cdot \mathbf{w}_i^T \mathbf{x}_i > 1$ ; otherwise,  $\mathbf{g} = -y \mathbf{x}$
- If  $y_i \cdot \mathbf{w}_i^T \mathbf{x}_i > 1$  (No mistake, by a margin): No update
- Otherwise, (Mistake, relative to margin):  $\mathbf{w}_{t+1} = \mathbf{w}_t + r y_t \mathbf{x}_t$



Here  $\mathbf{g} = -y\mathbf{x}$   
Good to think about the case of Boolean examples

# Perceptron on Boolean Examples

- $\mathbf{w}_{t+1} = \mathbf{w}_t - r_t \mathbf{g}_w Q(\mathbf{x}_t, y_t, \mathbf{w}_t) = \mathbf{w}_t - r_t \mathbf{g}_t$
- Perceptron:  $\mathbf{g} = \mathbf{0}$  if  $y_i \cdot \mathbf{w}_i^T \mathbf{x}_i > 1$ ; otherwise,  $\mathbf{g} = -y \mathbf{x}$
- Consider the performance of Perceptron on Boolean Examples:
  - $\mathbf{x} = \langle (0, 0, 1, 1, 0), + \rangle$
  - Assume that you made a mistake on  $\mathbf{x}$ :
    - what will happen to  $w_2$ ?
    - what will happen to  $w_3$ ?
  - Assume that you made another mistake on  $\mathbf{x}$ :
    - what will happen to  $w_2$ ?
    - what will happen to  $w_3$ ?
  - And another mistake on  $\mathbf{x}$ :
    - what will happen to  $w_2$ ?
    - what will happen to  $w_3$ ?

# What will happen to the two weights $w_2$ , $w_3$ on these three consecutive?

$w_2$  never changes,  $w_3$  grows by 1 in the first time, then stops changing

each time  $w_2$  goes down by 1 and  $w_3$  goes up by 1,

$w_2$  never changes,  $w_3$  grows by 1 all three times

None of the above

# New Stochastic Gradient Algorithms

$$\mathbf{w}_{t+1} = \mathbf{w}_t - r_t \mathbf{g}_w Q(\mathbf{z}_t, \mathbf{w}_t) = \mathbf{w}_t - r_t \mathbf{g}_t$$

(notice that this is a vector, each coordinate (feature) has its own  $w_{t,j}$  and  $g_{t,j}$ )

- So far, we used fixed learning rates  $r = r_t$ , but this can change.
- **AdaGrad** alters the update to adapt based on historical information
  - Frequently occurring features in the gradients get small learning rates and infrequent features get higher ones.
  - The idea is to “learn slowly” from frequent features but “pay attention” to rare but informative features.
- Define a “per feature” learning rate for the feature  $j$ , as:

$$r_{t,j} = r / (G_{t,j})^{1/2}$$

- where  $G_{t,j} = \sum_{k=1}^t g_{k,j}^2$  the sum of squares of gradients at feature  $j$  until time  $t$ .
- Overall, the update rule for Adagrad is:

$$w_{t+1,j} = w_{t,j} - g_{t,j} r / (G_{t,j})^{1/2}$$

Easy to think about the case of Perceptron, and on Boolean examples ( $x=0,0,1,\dots$ ). Check that in Perceptron, if  $x_{t,j} = 0$  then  $w_{t,j}$  does not change. Now, what happens when the  $j$ -th coordinate of  $x$  is 1 for a few consecutive examples?

- This algorithm is supposed to update weights faster than Perceptron or LMS when needed.

# Administration (10/19/20)

Are we recording? YES!

Available on the web site

- Remember that all the lectures are available on the website **before the class**
  - **Go over it and be prepared**
  - **A new set of written notes will accompany most lectures, with some more details, examples and, (when relevant) some code. The first one is available today for this lecture.**
- **HW 2: Due date extended to 10/22**
- Quizzes: Quiz 5 Statistics
- **Mid-term is on 10/28**; at the class time.
  - If you have a time zone problem – email me.
- Questions? Please ask/comment during class; give us feedback

# What have we done?

- Perceptron with Extensions

## New Stochastic Gradient Algorithms

$$w_{t+1} = w_t - r_t g_w Q(z_t, w_t) = w_t - r_t g_t$$

(notice that this is a vector, each coordinate (feature) has its own  $w_{t,j}$  and  $g_{t,j}$ )

- So far, we used fixed learning rates  $r = r_t$ , but this can change.
- AdaGrad** alters the update to adapt based on historical information
  - Frequently occurring features in the gradients get small learning rates and infrequent features get higher ones.
  - The idea is to "learn slowly" from frequent features but "pay attention" to rare but informative features.
- Define a "per feature" learning rate for the feature  $j$ , as:

$$r_{t,j} = r / (G_{t,j})^{1/2}$$

- where  $G_{t,j} = \sum_{k=1}^t g_{k,j}^2$  the sum of squares of gradients at feature  $j$  until time  $t$ .

- Overall, the update rule for Adagrad is:

$$w_{t+1,j} = w_{t,j} - g_{t,j} r / (G_{t,j})^{1/2}$$

- This algorithm is supposed to update weights faster than Perceptron or LMS when needed.

CIS 419/519 Fall'20

Some additional comments on [optimization with adaptive learning rates](#)

86

Easy to think about the case of Perceptron, and on Boolean examples ( $x \in \{0,1,\dots\}$ ). Check that in Perceptron, if  $x_{t,j} = 0$  then  $w_{t,j}$  does not change. Now, what happens when the  $j$ -th coordinate of  $x$  is 1 for a few consecutive examples?

## General Stochastic Gradient Algorithms

Learning rate      gradient      The loss  $Q$ : a function of  $x$ ,  $w$  and  $y$

$$w_{t+1} = w_t - r_t g_w Q(x_t, y_t, w_t) = w_t - r_t g_t$$

- LMS:  $Q((x,y), w) = \frac{1}{2} (y - w^T x)^2$
- Computing the gradient leads to the update rule (Also called Widrow's Adaline):  

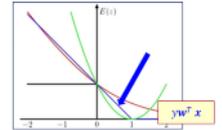
$$w_{t+1} = w_t + r (y_t - w_t^T x_t) x_t$$
- Here, even though we make binary predictions based on  $\text{sgn}(w^T x)$  we do not take the sign of the dot-product into account in the loss.

- Another common loss function is:

$$Q((x,y), w) = \max(0, 1 - y w^T x)$$

- Computing the gradient leads to the perceptron update rule:  

$$g = 0 \text{ if } y_t \cdot w_t^T x_t > 1; \text{ otherwise, } g = -y x$$



- If  $y_t \cdot w_t^T x_t > 1$  (No mistake, by a margin): No update
- Otherwise, (Mistake, relative to margin):  $w_{t+1} = w_t + r y_t x_t$

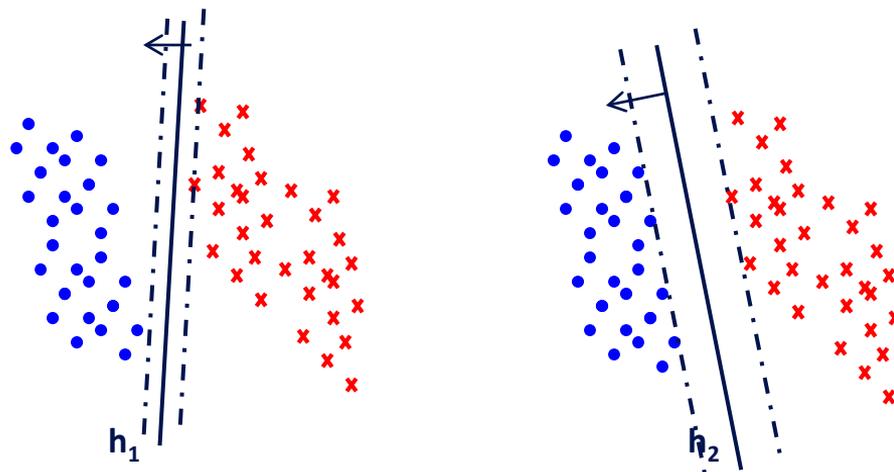
CIS 419/519 Fall'20

- Perceptron is an SGD Algorithm:

- With hinge loss

- Adagrad: Adaptive learning rate

# Preventing Overfitting



# Regularization

- The more general formalism adds a regularization term to the risk function, and minimize:

$$J(\mathbf{w}) = \frac{1}{m} \sum_1^m Q(\mathbf{z}_i, \mathbf{w}_i) + \lambda R_i(\mathbf{w}_i)$$

- Where  $R$  is used to enforce “simplicity” of the learned functions.

- LMS case:**  $Q((x, y), \mathbf{w}) = (y - \mathbf{w}^T \mathbf{x})^2$

- $R(\mathbf{w}) = \|\mathbf{w}\|_2^2$  gives the optimization problem called Ridge Regression.
- $R(\mathbf{w}) = \|\mathbf{w}\|_1$  gives a problem called the LASSO problem

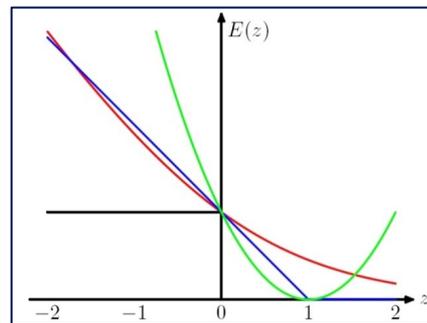
- Hinge Loss case:**  $Q((x, y), \mathbf{w}) = \max(0, 1 - y \mathbf{w}^T \mathbf{x})$

- $R(\mathbf{w}) = \|\mathbf{w}\|_2^2$  gives the problem called Support Vector Machines

- Logistics Loss case:**  $Q((x, y), \mathbf{w}) = \log(1 + \exp\{-y \mathbf{w}^T \mathbf{x}\})$

- $R(\mathbf{w}) = \|\mathbf{w}\|_2^2$  gives the problem called Logistics Regression

- These are convex optimization problems and, in principle, the same gradient descent mechanism can be used in all cases.
- We will see later why it makes sense to use the “size” of  $\mathbf{w}$  as a way to control “simplicity”.



# Algorithmic Approaches

---

- Focus: Two families of algorithms (one of the on-line representative)
  - **Additive** update algorithms: Perceptron
    - SVM is a close relative of Perceptron
  - **Multiplicative** update algorithms: Winnow
    - Close relatives: Boosting, Max entropy/Logistic Regression

# Summary of Algorithms

- Examples:  $\mathbf{x} \in \{0,1\}^n$ ; or  $\mathbf{x} \in \mathbf{R}^n$  (indexed by  $k$ ); Hypothesis:  $\mathbf{w} \in \mathbf{R}^n$
- Prediction:  $y \in \{-1, +1\}$ : Predict:  $y = 1$  iff  $\mathbf{w} \cdot \mathbf{x} > \theta$
- Update: Mistake Driven
- Additive weight update algorithm:  $\mathbf{w} \leftarrow \mathbf{w} + r y_k \mathbf{x}_k$ 
  - (Perceptron, Rosenblatt, 1958. Variations exist)
  - In the case of Boolean features:

*If Class = 1 but  $\mathbf{w} \cdot \mathbf{x} \leq \theta$ ,  $w_i \leftarrow w_i + 1$  (if  $x_i = 1$ )(promotion)  
If Class = 0 but  $\mathbf{w} \cdot \mathbf{x} \geq \theta$ ,  $w_i \leftarrow w_i - 1$  (if  $x_i = 1$ )(demotion)*

- Multiplicative weight update algorithm  $w_i \leftarrow w_i \alpha^{y_k x_i}$  (component-wise update)
- (Winnow, Littlestone, 1988. Variations exist)
  - Boolean features: (as an example,  $\alpha=2$ )

*If Class = 1 but  $\mathbf{w} \cdot \mathbf{x} \leq \theta$ ,  $w_i \leftarrow 2w_i$  (if  $x_i = 1$ )(promotion)  
If Class = 0 but  $\mathbf{w} \cdot \mathbf{x} \geq \theta$ ,  $w_i \leftarrow w_i/2$  (if  $x_i = 1$ )(demotion)*

# Which algorithm is better? How to Compare?

- Generalization
  - Since we deal with linear learning algorithms, we know (???) that they will all converge eventually to a perfect representation.
    - All can represent the data
- So, how do we compare:
  - How many **examples** are needed to get to a given level of accuracy?
  - How many **mistakes** will an algorithm make before getting to a give level of accuracy?
  - Efficiency: How long does it take to learn a hypothesis and evaluate it (per-example)?
  - Robustness (to noise);
  - Adaptation to a new domain, ....
- One key issue here turns out to be the **characteristics of the data**

# Sentence Representation

---

- Assume that our problem is context sensitive spelling:  
 $S =$  I don't know **weather** to laugh or cry
- Define a set of features:
  - features are relations that hold in the sentence
- Map a sentence to its feature-based representation
  - The feature-based representation will give some of the information in the sentence
- Use this feature-based representation as an example to your algorithm

# Sentence Representation

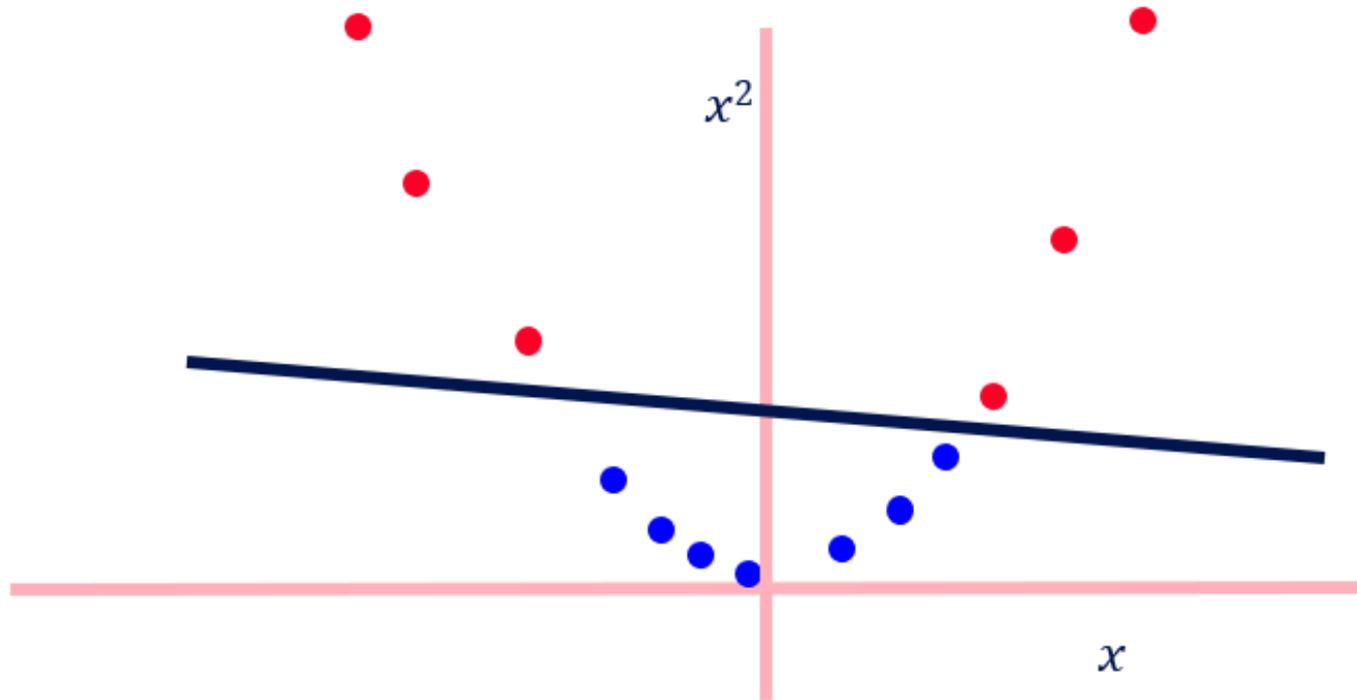
$S$  = I don't know **whether** to laugh or cry

- Define a set of features:
  - features are **properties** that hold in the sentence
- Conceptually, there are two steps in coming up with a feature-based representation
  - What are the information sources available?
    - Sensors: words, order of words, properties (?) of words
  - What features to construct based on these?

Why is this distinction needed?

Think about the features you use  
in the NER HW2 problem

# Blow Up Feature Space



# Domain Characteristics

---

- The number of potential features (dimensionality) is **very large**
  - The instance space is **sparse**
- Decisions depend on a small set of features
  - the function space is **sparse**
- Want to learn from a number of examples that is small relative to the dimensionality
  
- A comment on deep neural networks:
  - Have things changed with DNN? Not really; in many respects we can think about the DNNs as the feature extractors and learning of (relatively simple functions) is done over these feature extractors. We pay in training time, since it takes time to learn these feature extractors. We'll discuss this more later.

# Generalization

- Dominated by the sparseness of the function space
  - Most features are irrelevant
- # of examples required by multiplicative algorithms depends mostly on # of relevant features
  - (Generalization bounds depend on the target  $\|u\|$  )
- # of examples required by additive algorithms depends heavily on sparseness of features space:
  - Advantage to additive. Generalization depend on input  $\|x\|$ 
    - (Kivinen/Warmuth 95).
- Nevertheless, today most people use additive algorithms.

# Which Algorithm to Choose?

- Generalization (in terms of # of mistakes made)

The  $l_1$  norm:  $\|x\|_1 = \sum_i |x_i|$       The  $l_2$  norm:  $\|x\|_2 = (\sum_1^n |x_i|^2)^{1/2}$

The  $l_p$  norm:  $\|x\|_p = (\sum_1^n |x_i|^p)^{1/p}$       The  $l_\infty$  norm:  $\|x\|_\infty = \max_i |x_i|$

- **Multiplicative algorithms:**

- Bounds depend on  $\|u\|$ , the separating hyperplane;  $i$ : example #)
- $M_w = \frac{2 \ln n}{\|u\|_1^2} \max_i \|x^{(i)}\|_\infty^2 / \min_i (u x^{(i)})^2$
- Do not care much about data; advantage with sparse target  $u$

- **Additive algorithms:**

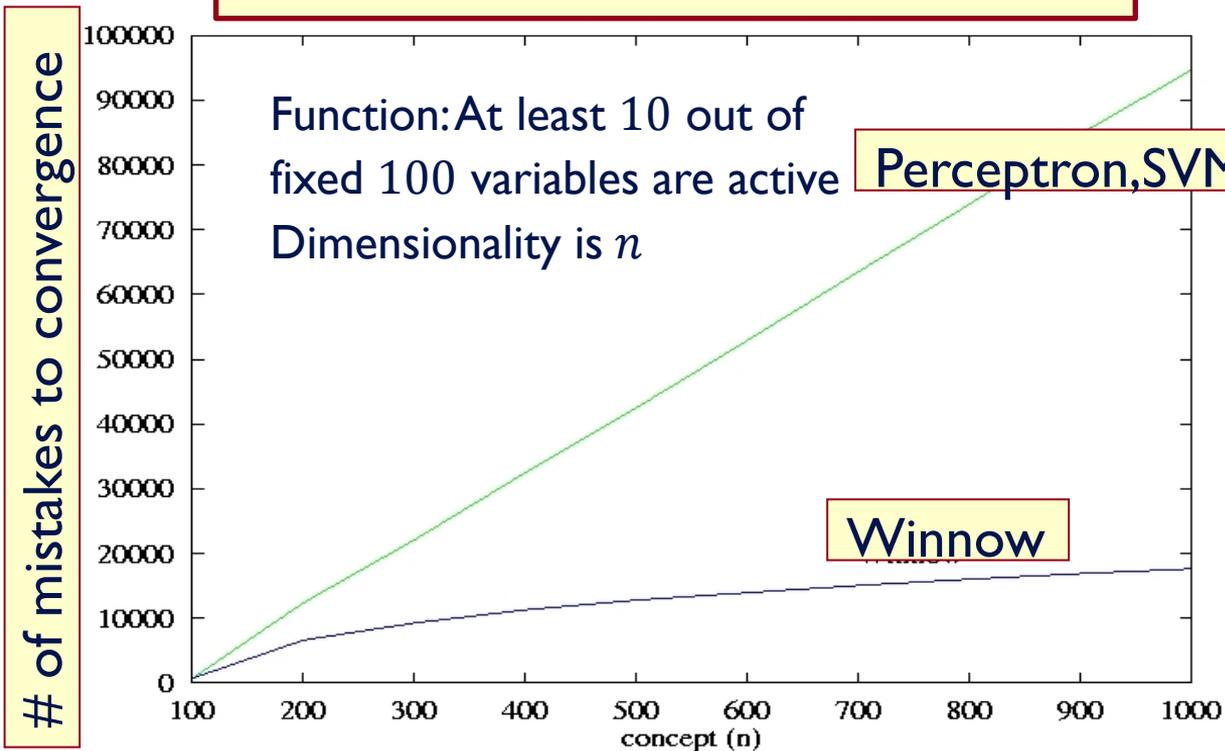
- Bounds depend on  $\|x\|$  (Kivinen / Warmuth, '95)
- $M_p = \|u\|_2^2 \max_i \|x^{(i)}\|_2^2 / \min_i (u x^{(i)})^2$
- Advantage with few active features per example

# Examples

$$M_w = 2 \ln n \|u\|_1^2 \max_i \|x^{(i)}\|_\infty^2 / \min_i (u x^{(i)})^2$$
$$M_p = \|u\|_2^2 \max_i \|x^{(i)}\|_2^2 / \min_i (u x^{(i)})^2$$

- Extreme Scenario 1: Assume the  $u$  has exactly  $k$  active features, and the other  $n - k$  are 0. That is, only  $k$  input features are relevant to the prediction. Then:
  - $\|u\|_2 = \sqrt{k}$ ;  $\|u\|_1 = k$ ;  $\max \|x\|_2 = \sqrt{n}$ ;  $\max \|x\|_\infty = 1$
  - We get that:  $M_p = kn$ ;  $M_w = 2k^2 \ln n$
  - Therefore, if  $k \ll n$ , Winnow behaves much better.
- Extreme Scenario 2: Now assume that  $u = (1, 1, \dots, 1)$  and the instances are very sparse, the rows of an  $n \times n$  unit matrix. Then:
  - $\|u\|_2 = \sqrt{n}$ ;  $\|u\|_1 = n$ ;  $\max \|x\|_2 = 1$ ;  $\max \|x\|_\infty = 1$
  - We get that:  $M_p = n$ ;  $M_w = 2n^2 \ln n$
  - Therefore, Perceptron has a better bound.

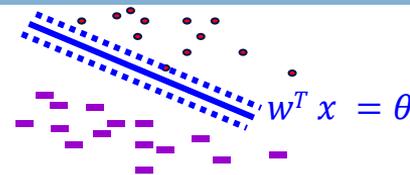
## Mistakes bounds for 10 of 100 of $n$



$n$ : Total # of Variables (Dimensionality)

# Summary

- Introduced multiple versions of on-line algorithms
- Most turned out to be Stochastic Gradient Algorithms
  - For different loss functions
- Some turned out to be mistake driven
- We suggested generic improvements via regularization:
  - Adding a term that forces a “simple hypothesis”
  - $J(\mathbf{w}) = \sum_{1, m} Q(z_i, w_i) + \lambda R_i(w_i)$
  - Regularization via the Averaged Trick
    - “Stability” of a hypothesis is related to its ability to generalize
  - An improved, adaptive, learning rate (Adagrad)
- Dependence on **function space** and the **instance space** properties.
- Now:
  - A way to deal with non-linear target functions (Kernels)
  - Beginning of Learning Theory.

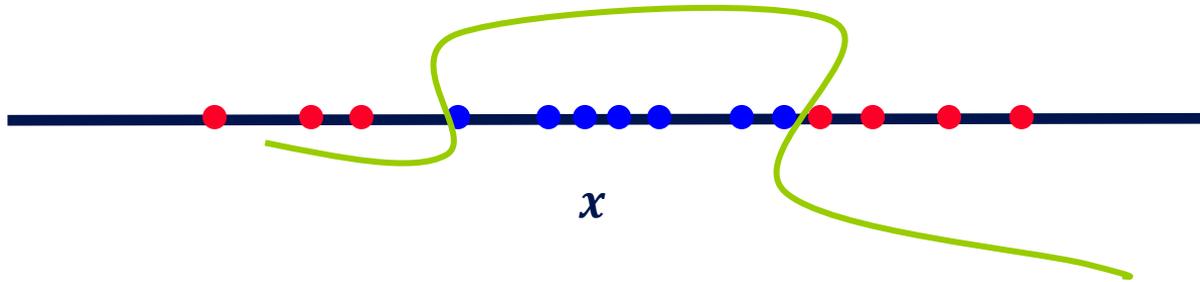


A term that minimizes error on the training data

A term that forces simple hypothesis

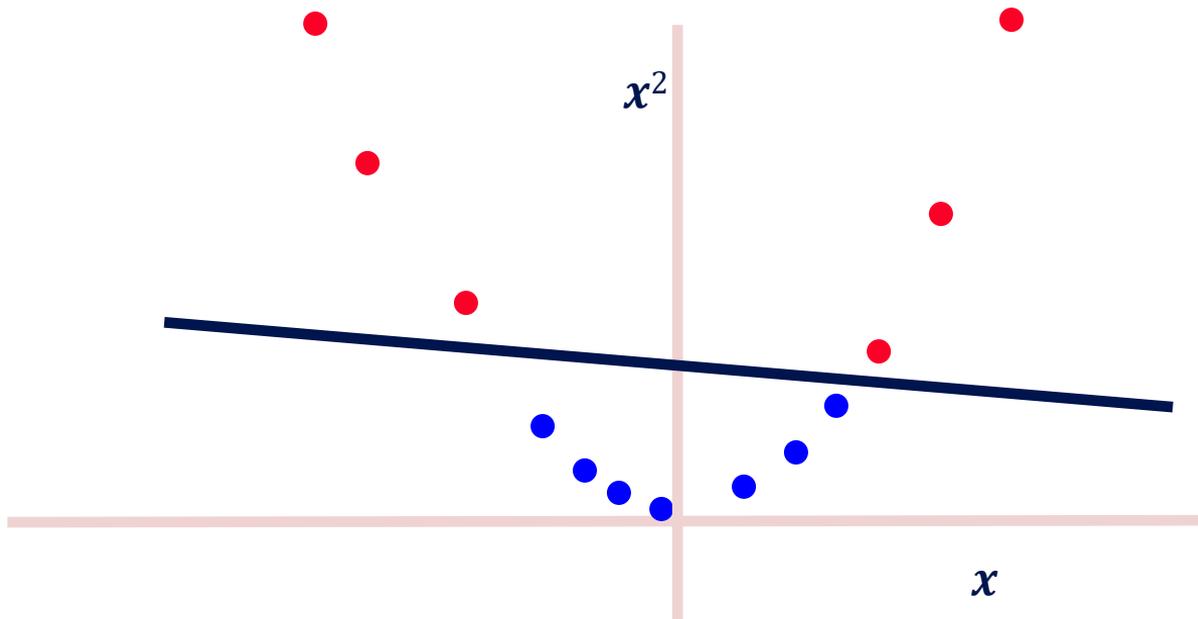
# Functions Can be Made Linear

- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions

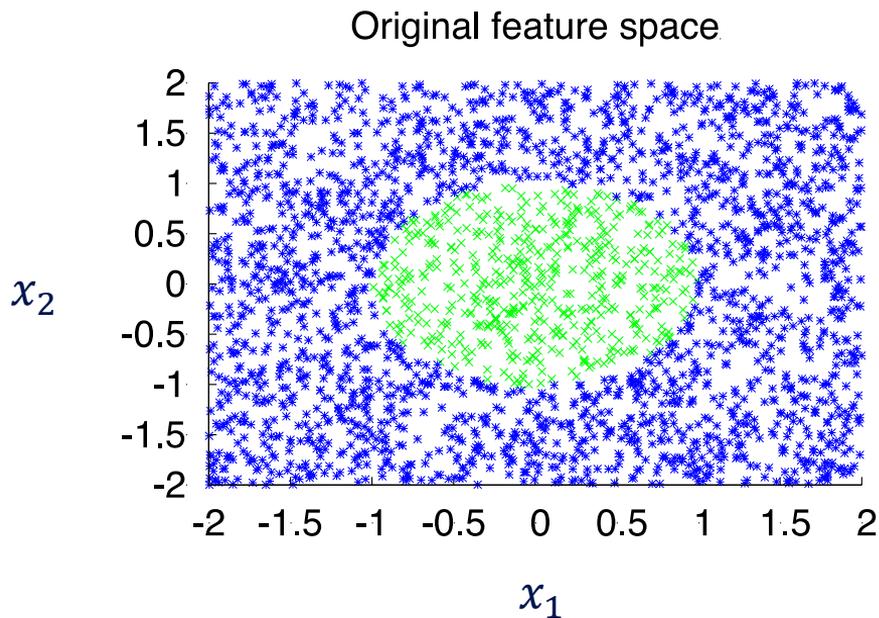


# Blown Up Feature Space

- Data are separable in  $\langle x, x^2 \rangle$  space



# Making data linearly separable



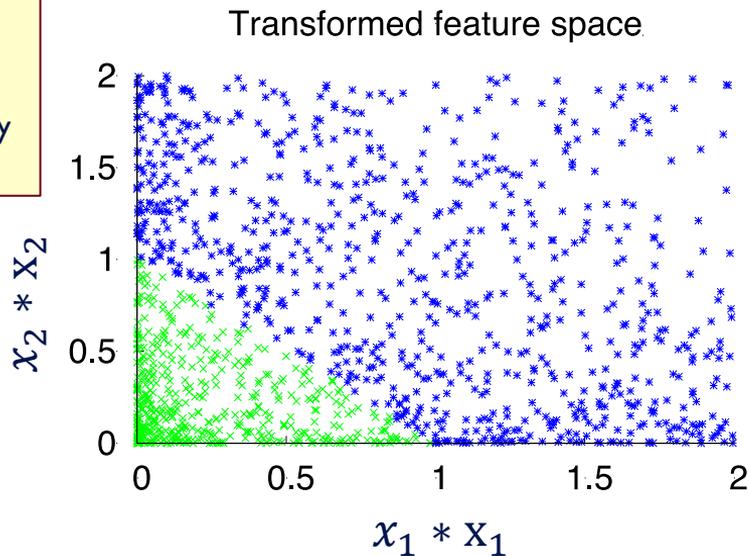
$$f(\mathbf{x}) = 1 \text{ iff } x_1^2 + x_2^2 \leq 1$$

# Making data linearly separable

■ In order to deal with this, we introduce two new concepts:

- Dual Representation
- Kernel (& the kernel trick)

The key question is: how to systematically transform the feature space?



$$\text{Transform data: } \mathbf{x} = (x_1, x_2) \rightarrow \mathbf{x}' = (x_1^2, x_2^2)$$
$$f(\mathbf{x}') = 1 \text{ iff } x'_1 + x'_2 \leq 1$$

# Dual Representation

Examples  $\mathbf{x} \in \{0,1\}^N$ ; Learned hypothesis  $\mathbf{w} \in \mathbb{R}^N$

$$f(\mathbf{x}) = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\} = \text{sgn}\{\sum_{i=1}^n w_i x_i\}$$

Perceptron Update:

If  $y' \neq y$ , update:  $\mathbf{w} = \mathbf{w} + ry\mathbf{x}$

- Let  $\mathbf{w}$  be an initial weight vector for perceptron. Let  $(\mathbf{x}^1, +)$ ,  $(\mathbf{x}^2, +)$ ,  $(\mathbf{x}^3, -)$ ,  $(\mathbf{x}^4, -)$  be examples and assume mistakes are made on  $\mathbf{x}^1, \mathbf{x}^2$  and  $\mathbf{x}^4$ .
- What is the resulting weight vector?

Let  $(x^1,+)$ ,  $(x^2,+)$ ,  $(x^3,-)$ ,  $(x^4,-)$  be examples and assume mistakes are made on  $x^1$ ,  $x^2$  and  $x^4$ . What is the resulting weight vector (assume  $r=1$ )?

# Administration (10/21/20)

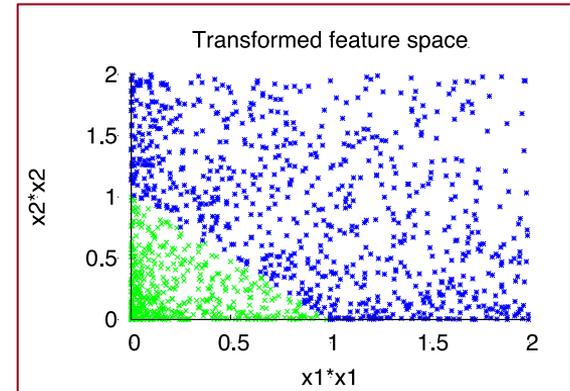
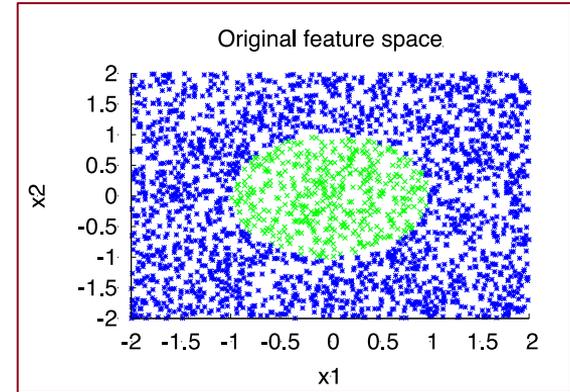
Are we recording? YES!

Available on the web site

- Remember that all the lectures are available on the website **before the class**
  - **Go over it and be prepared**
  - **A new set** of written notes will accompany most lectures, with some more details, examples and, (when relevant) some code.
- **HW 2:** Due date extended to 10/22
- Quizzes: Quiz 5 Statistics
- **Mid-term is on 10/28;** at the class time.
  - Example mid-terms are available on the website
  - If you have a time zone problem – email me.
- Questions? Please ask/comment during class; give us feedback

# Kernels

- Dealing with non-linearly-separable data using linear classifier.
- Method: Systematically enrich the feature space
  - How to avoid the computation cost of working with a much larger feature space?
  - Done by moving the dual space
  - Instead of predicting by computing a dot product with a weight vector  $w$ , the prediction  $f(x)$  is done by computing dot-products  $\sum g(x_i^T x)$
  - of the target point  $x$  with other examples  $x_i^T$  (observed earlier)



# Dual Representation

Examples  $\mathbf{x} \in \{0,1\}^N$ ; Learned hypothesis  $\mathbf{w} \in \mathbb{R}^N$

$$f(\mathbf{x}) = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\} = \text{sgn}\{\sum_{i=1}^n w_i x_i\}$$

Note: We care about the dot product:  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} =$

Perceptron Update:

If  $y' \neq y$ , update:  $\mathbf{w} = \mathbf{w} + r y \mathbf{x}$

- Let  $\mathbf{w}$  be an initial weight vector for perceptron. Let  $(\mathbf{x}^1, +)$ ,  $(\mathbf{x}^2, +)$ ,  $(\mathbf{x}^3, -)$ ,  $(\mathbf{x}^4, -)$  be examples and assume mistakes are made on  $\mathbf{x}^1$ ,  $\mathbf{x}^2$  and  $\mathbf{x}^4$ .

- What is the resulting weight vector?

$$\mathbf{w} = \mathbf{w} + \mathbf{x}^1 + \mathbf{x}^2 - \mathbf{x}^4$$

- (here  $r = 1$ )
- In general, the weight vector  $\mathbf{w}$  can be written as a linear combination of examples:

$$\mathbf{w} = \sum_1^m r \alpha_i y_i \mathbf{x}^i$$

- Where  $\alpha_i$  is the number of mistakes made on  $\mathbf{x}^i$ .

# Kernel Based Methods $f(x) = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\} = \text{sgn}\{\sum_{i=1}^n w_i x_i\}$

- Representing the model in the dual space will allow us to use Kernels.
- A method to run Perceptron on a very large feature set, without incurring the cost of keeping a very large weight vector.
- Computing the dot product can be done in the original feature space.
  - Notice: this pertains only to efficiency: The classifier is identical to the one you get by blowing up the feature space.
  - Generalization is still relative to the real dimensionality (or, related properties).
- Kernels were popularized by SVMs, but many other algorithms can make use of them (== run in the dual).
  - Linear Kernels: no kernels; stay in the original space. A lot of applications actually use linear kernels.

# Kernel Base Methods

- Recall the NER problem in HW2.
- You added conjunctive features.
- We will do it systematically now, but we'll add all the conjunctive features, not only conjunctions of small size.

Examples  $\mathbf{x} \in \{0,1\}^N$ ; Learned hypothesis  $\mathbf{w} \in \mathbf{R}^N$

$$f(\mathbf{x}) = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\} = \text{sgn}\{\sum_{i=1}^n w_i x_i(\mathbf{x})\}$$

- Let's transform the space to an expressive one.
- Let  $I$  be the set  $t_1, t_2, t_3 \dots$  of monomials (conjunctions) over the feature space  $x_1, x_2 \dots x_n$ .
- Then we can write a linear function over this **new feature space**.

New features (indexed by  $I$ )

$$f(\mathbf{x}) = \text{sgn}\{\mathbf{w}^T \cdot \mathbf{x}\} = \text{sgn}\{\sum_I w_i t_i(\mathbf{x})\}$$

Example:  $x_1 x_2 x_4(11010) = 1$   $x_3 x_4(11010) = 0$   $x_1 x_2(11010) = 1$

# Kernel Based Methods

Examples  $\mathbf{x} \in \{0,1\}^N$ ; Learned hypothesis  $\mathbf{w} \in \mathbf{R}^N$

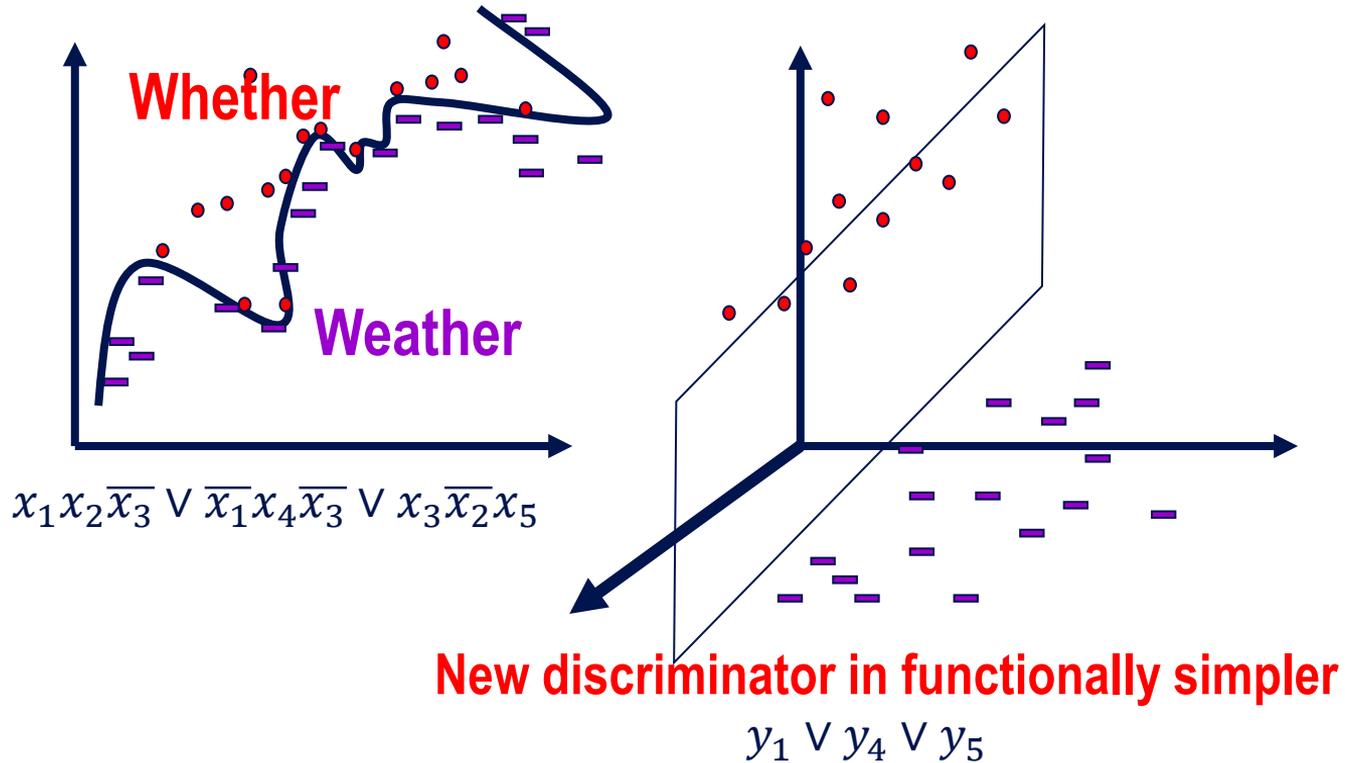
$$f(\mathbf{x}) = \text{sgn} \{ \mathbf{w}^T \cdot \mathbf{x} \} = \text{sgn} \{ \underline{\sum_{i=1}^n w_i t_i(\mathbf{x})} \}$$

Perceptron Update:

$$\text{If } y' \neq y, \text{ update: } \mathbf{w} = \mathbf{w} + r y \mathbf{x}$$

- Great Increase in expressivity
- Can run Perceptron (and Winnow) but the convergence bound may suffer exponential growth.
- Exponential number of monomials are true in each example.
- Also, will have to keep many weights.

# Embedding



# The Kernel Trick(1)

Let  $\Phi(\mathbf{x})$  be the representation of  $\mathbf{x}$  in the new feature space. We already showed that:

$$\begin{aligned} f(\Phi(\mathbf{x})) &= \mathbf{w}^T \Phi(\mathbf{x}) = (\sum_1^n r \alpha_i y_i \Phi(\mathbf{x}^i))^T \Phi(\mathbf{x}) \\ &= \sum_1^n r \alpha_i y_i (\Phi(\mathbf{x}^i)^T \Phi(\mathbf{x})) \end{aligned}$$

Examples  $\mathbf{x} \in \{0,1\}^N$ ; Learned hypothesis  $\mathbf{w} \in \mathbf{R}^N$

$$f(\mathbf{x}) = \text{sgn} \{ \mathbf{w}^T \cdot \mathbf{x} \} = \text{sgn} \{ \sum_{i=1}^n w_i t_i(\mathbf{x}) \}$$

Perceptron Update:

New features (n)

If  $y' \neq y$ , update:  $\mathbf{w} = \mathbf{w} + r y \mathbf{x}$

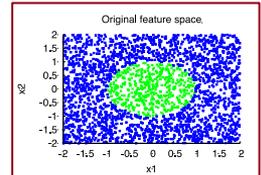
We will not show it in a more explicit way.

# Example: Polynomial Kernel

We called this a “dual representation:

- Prediction with respect to a separating hyper planes  $\mathbf{w}$  (produced by Perceptron, SVM) can be computed **instead**, as a function of dot products of feature based representation of (some of) the examples.
- We want to define a dot product in a **high** dimensional space.
- Given two examples  $x = (x_1, x_2, \dots, x_n)$  and  $y = (y_1, y_2, \dots, y_n)$  we want to map them to a high dimensional space [example- quadratic]:
- $\Phi(x_1, x_2, \dots, x_n) = (1, x_1, \dots, x_n, x_1^2, \dots, x_n^2, x_1x_2, \dots, x_{n-1}x_n)$
- $\Phi(y_1, y_2, \dots, y_n) = (1, y_1, \dots, y_n, y_1^2, \dots, y_n^2, y_1y_2, \dots, y_{n-1}y_n)$   
and compute the dot product  $A = \Phi(x)^T \Phi(y)$  [higher dimensionality, takes time ]
- Instead, in the original space, compute
  - $B = k(x, y) = [1 + (x_1, x_2, \dots, x_n)^T (y_1, y_2, \dots, y_n)]^2$
- **Theorem:**  $A = B$
- Exercise: Compute A and B and prove the theorem. (Coefficients do not really matter)

Sq(2)



# The Kernel Trick(1)

Let  $T(\mathbf{x})$  be the representation of  $\mathbf{x}$  in the new feature space. We already showed that:

$$\begin{aligned} f(T(\mathbf{x})) &= \mathbf{w}^T T(\mathbf{x}) = (\sum_1^n r \alpha_i y_i T(\mathbf{x}^i))^T T(\mathbf{x}) \\ &= \sum_1^n r \alpha_i y_i (T(\mathbf{x}^i)^T T(\mathbf{x})) \end{aligned}$$

Examples  $\mathbf{x} \in \{0,1\}^N$ ; Learned hypothesis  $\mathbf{w} \in \mathbf{R}^N$

$$f(\mathbf{x}) = \text{sgn} \{ \mathbf{w}^T \cdot \mathbf{x} \} = \text{sgn} \{ \sum_{i=1}^n w_i t_i(\mathbf{x}) \}$$

Perceptron Update:

New features (n)

If  $y' \neq y$ , update:  $\mathbf{w} = \mathbf{w} + r y \mathbf{x}$

We will not show it in a more explicit way.

- Consider the value of  $\mathbf{w}$  used in the prediction.
- Each previous mistake on example  $z$ , makes an additive contribution of  $+/-1$  to some of the coordinates of  $\mathbf{w}$ .
  - Note: examples are Boolean, so only coordinates of  $\mathbf{w}$  that correspond to ON terms in the example  $z$  ( $t_i(z) = 1$ ) are being updated.
- The value of  $\mathbf{w}$  is determined by the number and type of mistakes.

# The Kernel Trick(2)

Examples  $\mathbf{x} \in \{0,1\}^N$ ; Learned hypothesis  $\mathbf{w} \in \mathbf{R}^N$

$$f(\mathbf{x}) = \text{sgn} \{ \mathbf{w}^T \cdot \mathbf{x} \} = \text{sgn} \{ \sum_{i=1}^n w_i t_i(\mathbf{x}) \}$$

Perceptron Update:

$$\text{If } y' \neq y, \text{ update: } \mathbf{w} = \mathbf{w} + r y \mathbf{x}$$

- $P$  – set of examples on which we Promoted
- $D$  – set of examples on which we Demoted
- $M = P \cup D$

$$\begin{aligned} f(\mathbf{x}) &= \text{sgn} \sum_I w_i t_i(\mathbf{x}) = \sum_I \left[ \sum_{z \in P, t_i(z)=1} 1 - \sum_{z \in D, t_i(z)=1} 1 \right] t_i(\mathbf{x}) = \\ &= \sum_I \left[ \sum_{z \in M} S(z) t_i(z) t_i(\mathbf{x}) \right] \end{aligned}$$

# The Kernel Trick(3)

$$f(\mathbf{x}) = \text{sgn} \{ \mathbf{w}^T \cdot \mathbf{x} \} = \text{sgn} \{ \sum_I w_i t_i(\mathbf{x}) \}$$

- $P$  – set of examples on which we Promoted
- $D$  – set of examples on which we Demoted
- $M = P \cup D$
- $f(\mathbf{x}) = \text{sgn} \sum_I w_i t_i(\mathbf{x}) = \sum_I \left[ \sum_{z \in P, t_i(z)=1} 1 - \sum_{z \in D, t_i(z)=1} 1 \right] t_i(\mathbf{x})$
- $= \text{sgn} \{ \sum_I [ \sum_{z \in M} S(z) t_i(z) t_i(\mathbf{x}) ] \}$
- Where  $S(z) = 1$  if  $z \in P$  and  $S(z) = -1$  if  $z \in D$ .
- Reordering:

$$f(\mathbf{x}) = \text{sgn} \{ \sum_{z \in M} S(z) \sum_I t_i(z) t_i(\mathbf{x}) \}$$

# The Kernel Trick(4)

$$f(x) = \operatorname{sgn} \sum_{i \in I} w_i t_i(x)$$

- $S(y) = 1$  if  $y \in P$  and  $S(y) = -1$  if  $y \in D$ .

$$f(x) = \operatorname{sgn} \sum_{z \in M} S(z) \sum_{i \in I} t_i(z) t_i(x)$$

- A mistake on  $z$  contributes the value  $+/-1$  to all monomials satisfied by  $z$ . The total contribution of  $z$  to the sum is equal to the number of monomials that satisfy both  $x$  and  $z$ .
- Define a dot product in the  $t$ -space:

$$K(x, z) = \sum_{i \in I} t_i(z) t_i(x)$$

- We get the standard notation:

$$f(x) = \operatorname{sgn} \left( \sum_{z \in M} S(z) K(x, z) \right)$$

# Kernel Based Methods

$$f(x) = \text{sgn}\left(\sum_{z \in M} S(z)K(x, z)\right)$$

- What does this representation give us?

$$K(x, z) = \sum_{i \in I} t_i(z)t_i(x)$$

- We can view this Kernel as the distance between  $x, z$  in the  $t$ -space.
- So far, all we did is algebra; as written above  $I$  is huge.
- **But,  $K(x, z)$  can be measured in the original space, without explicitly writing the  $t$ -representation of  $x, z$**

# Kernel Trick

$$\begin{aligned}x_1 x_3 (001) &= x_1 x_3 (011) = 1 \\x_1 (001) &= x_1 (011) = 1; \quad x_3 (001) = x_3 (011) = 1 \\ \Phi(001) &= \Phi(011) = 1\end{aligned}$$

If any other variables appears in the monomial, it's evaluation on  $x, z$  will be different.

$$f(x) = \operatorname{sgn}\left(\sum_{z \in M} S(z)K(x, z)\right) \quad K(x, z) = \sum_{i \in I} t_i(z)t_i(x)$$

- Consider the space of all  $3^n$  monomials (allowing both positive and negative literals). Then,

- **Claim:**  $K(x, z) = \sum_{i \in I} t_i(z)t_i(x) = 2^{\operatorname{same}(x, z)}$

- Where  $\operatorname{same}(x, z)$  is the number of features that have the same value for both  $x$  and  $z$ .

- We get:

$$f(x) = \operatorname{sgn}\left(\sum_{z \in M} S(z)(2^{\operatorname{same}(x, z)})\right)$$

- Example: Take  $n = 3$ ;  $x = (001), z = (011)$ , features are monomials of size 0, 1, 2, 3.
- **Proof:** let  $k = \operatorname{same}(x, z)$ ; construct a “surviving” monomials by: (1) choosing to include one of these  $k$  literals with the right polarity in the monomial, or (2) choosing to not include it at all. Monomials with literals outside this set disappear.

# Example

$$f(x) = \text{sgn}\left(\sum_{z \in M} S(z)K(x, z)\right) \quad K(x, z) = \sum_{i \in I} t_i(z)t_i(x)$$

- Take  $X = \{x_1, x_2, x_3, x_4\}$
- $I$  = The space of all  $3^n$  monomials;  $|I| = 81$
- Consider  $x = (1100), z = (1101)$
- Write down  $I(x), I(z)$ , the representation of  $x, z$  in the  $I$  space.
- Compute  $I(x) \cdot I(z)$ .
- Show that:
  - $K(x, z) = I(x) \cdot I(z) = \sum_I t_i(z)t_i(x) = 2^{\text{same}(x, z)} = 8$
- Try to develop another kernel, e.g., where  $I$  is the space of all conjunctions of size 3 (exactly).

# Implementation: Dual Perceptron

$$f(x) = \text{sgn}\left(\sum_{z \in M} S(z)K(x, z)\right) \quad K(x, z) = \sum_{i \in I} t_i(z)t_i(x)$$

- Simply run Perceptron in an on-line mode, **but keep track of the set  $M$** .
- Keeping the set  $M$  allows us to keep track of  $S(z)$ .
- Rather than remembering the weight vector  $w$ , remember the set  $M$  ( $P$  and  $D$ ) – all those examples on which we made mistakes.
  
- Dual Representation

# Example: Polynomial Kernel

We called this a “dual representation:

$$f(x) = \text{sgn}\left(\sum_{z \in M} S(z)K(x, z)\right)$$

- Prediction with respect to a separating hyper planes  $\mathbf{w}$  (produced by Perceptron, SVM) can be computed **instead**, as a function of dot products of feature based representation of (some of) the examples.
- We want to define a dot product in a **high** dimensional space. Sq(2)
- Given two examples  $x = (x_1, x_2, \dots, x_n)$  and  $y = (y_1, y_2, \dots, y_n)$  we want to map them to a high dimensional space [example- quadratic]:
- $\Phi(x_1, x_2, \dots, x_n) = (1, x_1, \dots, x_n, x_1^2, \dots, x_n^2, x_1x_2, \dots, x_{n-1}x_n)$
- $\Phi(y_1, y_2, \dots, y_n) = (1, y_1, \dots, y_n, y_1^2, \dots, y_n^2, y_1y_2, \dots, y_{n-1}y_n)$   
and compute the dot product  $A = \Phi(x)^T \Phi(y)$  [takes time ]
- Instead, in the original space, compute
  - $B = k(x, y) = [1 + (x_1, x_2, \dots, x_n)^T (y_1, y_2, \dots, y_n)]^2$
- **Theorem:**  $A = B$  (Coefficients do not really matter)

# Kernels – General Conditions

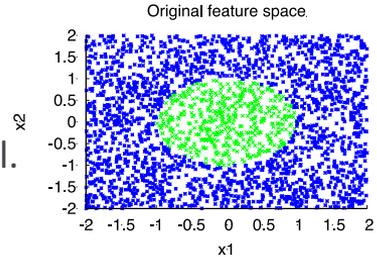
Definition: When is  $k(x,z)$  a kernel?

**Kernel Trick:** You want to work with degree 2 polynomial features  $\Phi(x)$ . Then, your dot product will be in a space of dimensionality  $n(n + 1)/2$ . The kernel trick allows you to save and compute dot products in an  $n$  dimensional space.

$$f(x) = \text{sgn}\left(\sum_{z \in M} S(z)K(x,z)\right) \quad K(x,z) = \sum_{i \in I} t_i(z)t_i(x)$$

- Can we use any  $K(.,.)$ ?
  - **A function  $K(x,z)$  is a valid kernel** if it corresponds to a dot (an inner) product in some (perhaps infinite dimensional) feature space.
- Take the **quadratic function**:  $k(x,z) = (x^T z)^2$ 
  - Is it a kernel?
- Example: Direct construction (2 dimensional, for simplicity):
- $K(x,z) = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2$   
 $= (x_1^2, \sqrt{2}x_1x_2, x_2^2)(z_1^2, \sqrt{2}z_1z_2, z_2^2)^T$   
 $= \Phi(x)^T \Phi(z) \rightarrow$  A dot product in an expanded space; consequently, a kernel.
- Comment:
- In general, it is not necessary to explicitly show the feature function  $\Phi$ .
- General condition: construct the kernel matrix  $\{k(x_i, z_j)\}$  (indices are over the examples); check that it's positive semi definite.

We proved that  $K$  is a valid kernel by explicitly showing that it corresponds to a dot product.



# Polynomial kernels

---

- Linear kernel:  $k(\mathbf{x}, \mathbf{z}) = \mathbf{xz}$
- Polynomial kernel of degree  $d$ :  $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz})^d$   
(only  $d$ -th-order interactions)
- Polynomial kernel up to degree  $d$ :  $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz} + c)^d$  ( $c > 0$ )  
(all interactions of order  $d$  or lower)

# Constructing New Kernels

- You can construct new kernels  $k'(\mathbf{x}, \mathbf{x}')$  from existing ones:

- Multiplying  $k(\mathbf{x}, \mathbf{x}')$  by a constant  $c$ :

$$k'(\mathbf{x}, \mathbf{x}') = ck(\mathbf{x}, \mathbf{x}')$$

- Multiplying  $k(\mathbf{x}, \mathbf{x}')$  by a function  $f$  applied to  $\mathbf{x}$  and  $\mathbf{x}'$ :

$$k'(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$$

- Applying a polynomial (with non-negative coefficients) to  $k(\mathbf{x}, \mathbf{x}')$ :

$$k'(\mathbf{x}, \mathbf{x}') = P(k(\mathbf{x}, \mathbf{x}')) \text{ with } P(z) = \sum_i a_i z_i \text{ and } a_i \geq 0$$

- Exponentiating  $k(\mathbf{x}, \mathbf{x}')$ :

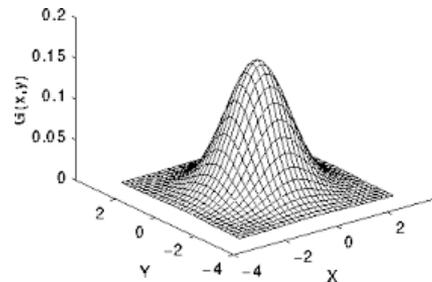
$$k'(\mathbf{x}, \mathbf{x}') = \exp(k(\mathbf{x}, \mathbf{x}'))$$

# Constructing New Kernels (2)

- You can construct  $k'(\mathbf{x}, \mathbf{x}')$  from  $k_1(\mathbf{x}, \mathbf{x}')$ ,  $k_2(\mathbf{x}, \mathbf{x}')$  by:
  - Adding  $k_1(\mathbf{x}, \mathbf{x}')$  and  $k_2(\mathbf{x}, \mathbf{x}')$ :
$$k'(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$$
  - Multiplying  $k_1(\mathbf{x}, \mathbf{x}')$  and  $k_2(\mathbf{x}, \mathbf{x}')$ :
$$k'(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$$
- Also:
  - If  $\phi(\mathbf{x}) \in \mathbf{R}^m$  and  $k_m(\mathbf{z}, \mathbf{z}')$  a valid kernel in  $\mathbf{R}^m$ ,  
 $k(\mathbf{x}, \mathbf{x}') = k_m(\phi(\mathbf{x}), \phi(\mathbf{x}'))$  is also a valid kernel
  - If  $A$  is a symmetric positive semi-definite matrix,  
 $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}A\mathbf{x}'$  is also a valid kernel
- In all cases, it is easy to prove these directly by construction.

# Gaussian Kernel (Radial Basis Function kernel)

- $k(\mathbf{x}, \mathbf{z}) = \exp(-(\mathbf{x} - \mathbf{z})^2 / c)$ 
  - $(\mathbf{x} - \mathbf{z})^2$ : squared Euclidean distance between  $\mathbf{x}$  and  $\mathbf{z}$
  - $c = 2\sigma^2$ : a free parameter
  - very small  $c$ :  $K \approx$  identity matrix (every item is different)
  - very large  $c$ :  $K \approx$  unit matrix (all items are the same)
- $k(\mathbf{x}, \mathbf{z}) \approx 1$  when  $\mathbf{x}, \mathbf{z}$  close
- $k(\mathbf{x}, \mathbf{z}) \approx 0$  when  $\mathbf{x}, \mathbf{z}$  dissimilar



# Gaussian Kernel

- $k(x, z) = \exp(-(x - z)^2/c)$ 
  - Is this a kernel?
- $k(x, z) = \exp(-(x - z)^2/2\sigma^2)$ 
  - $= \exp(-(xx + zz - 2xz)/2\sigma^2)$
  - $= \exp(-xx/2\sigma^2) \exp(xz/\sigma^2) \exp(-zz/2\sigma^2)$
  - $= f(x) \exp(xz/\sigma^2) f(z)$
- $\exp(xz/\sigma^2)$  is a valid kernel:
  - $xz$  is the linear kernel;
  - we can multiply kernels by constants ( $1/\sigma^2$ )
  - we can exponentiate kernels
- Unlike the discrete kernels discussed earlier, here you cannot easily explicitly blow up the feature space to get an identical representation.

# Summary – Kernel Based Methods

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{\mathbf{z} \in M} S(\mathbf{z})K(\mathbf{x}, \mathbf{z})\right)$$

- A method to run Perceptron on a very large feature set, without incurring the cost of keeping a very large weight vector.
- Computing the weight vector can be done in the original feature space.
- Notice: this pertains only to efficiency: the classifier is identical to the one you get by blowing up the feature space.
- Generalization is still relative to the real dimensionality (or, related properties).
- Kernels were popularized by SVMs but apply to a range of models, Perceptron, Gaussian Models, PCAs, etc.

# Explicit & Implicit Kernels: Complexity

- Is it always worthwhile to define kernels and work in the dual space?
- Computationally:
  - $m$ : # of examples,
  - $t_1$ : dimensionality of the original space, and therefore the dimensionality used in the Dual space
  - $t_2$ : the dimensionality of the Primal space (blown up features space ( $O(n^2)$  or  $3^n$ , in earlier examples)
    - (That is,  $t_1, t_2$  is the cost of a dot product in the respective spaces)
  - Then, computational cost is:
    - Dual space –  $t_1 m^2$  vs, Primal Space –  $t_2 m$  (think about it)
  - Typically,  $t_1 \ll t_2$ , so it boils down to the number of examples one needs to consider relative to the growth in dimensionality.
- Rule of thumb:
  - A lot of examples  $\rightarrow$  use Primal space
  - High dimensionality, not so many examples, use dual space
- Most applications today: People use explicit kernels. That is, they blow up the feature space explicitly.

# Kernels: Generalization

---

- Do we want to use the most expressive kernels we can?
  - (e.g., when you want to add quadratic terms, do you really want to add all of them?)
- No; this is equivalent to working in a larger feature space, and will lead to overfitting.
- It's possible to give simple arguments that show that simply adding irrelevant features does not help.

# Conclusion- Kernels

---

- The use of Kernels to learn in the dual space is an important idea
  - Different kernels may expand/restrict the hypothesis space in useful ways.
  - Need to know the benefits and hazards
- To justify these methods we must embed in a space much larger than the training set size.
  - Can affect generalization
- Expressive structures in the input data could give rise to specific kernels, designed to exploit these structures.
  - E.g., people have developed kernels over parse trees: corresponds to features that are sub-trees.
  - It is always possible to trade these with explicitly generated features, but it might help one's thinking about appropriate features.

# How are we doing?

We are moving too slow

We are moving a bit too slow

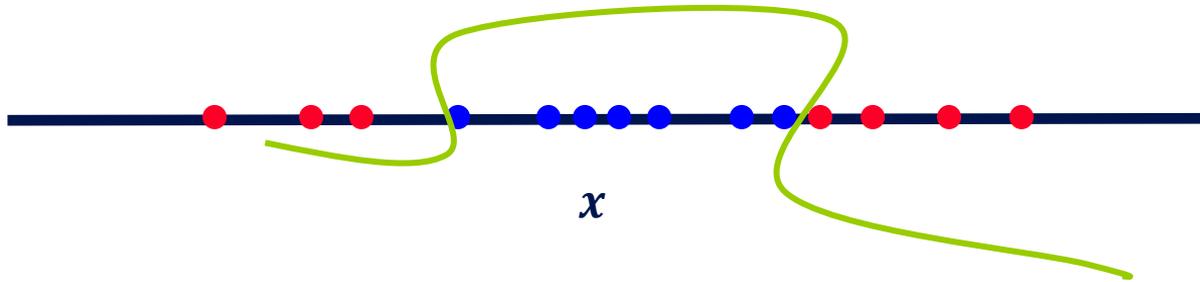
We are about right

We are moving a bit too fast

We are moving way too fast

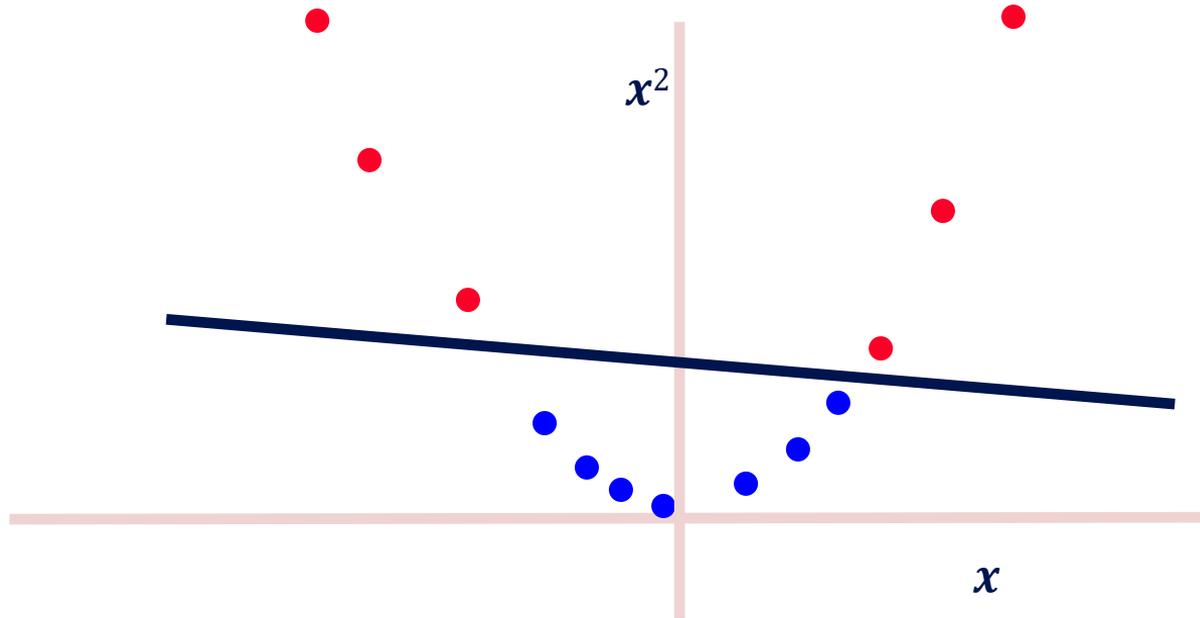
# Functions Can be Made Linear

- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions



# Blown Up Feature Space

- Data are separable in  $\langle x, x^2 \rangle$  space

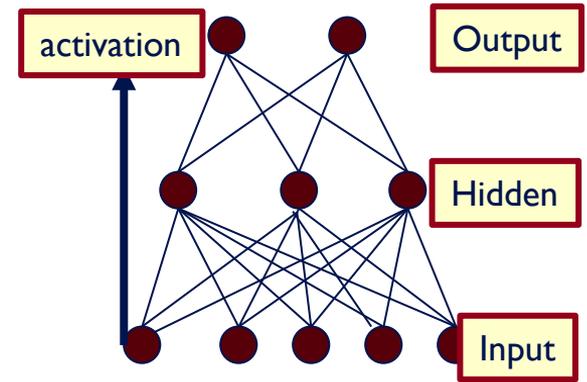


End



# Multi-Layer Neural Network

- Multi-layer networks were designed to overcome the computational (expressivity) limitation of a single threshold element.
- The idea is to stack several layers of threshold elements, each layer using the output of the previous layer as input.
- Multi-layer networks can represent arbitrary functions, but building effective learning methods for such networks was [thought to be] difficult.

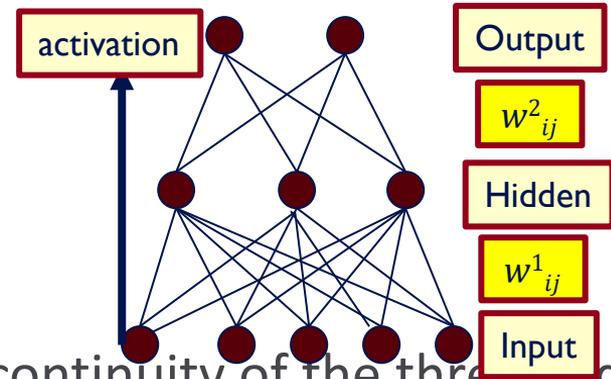


# Basic Units

- **Linear Unit:** Multiple layers of linear functions  $o_j = \mathbf{w} \cdot \mathbf{x}$  produce linear functions. We want to represent nonlinear functions.
- **Need to do it in a way that facilitates learning**
- **Threshold units:**  $o_j = \text{sgn}(\mathbf{w} \cdot \mathbf{x})$  are not differentiable, hence unsuitable for gradient descent.
- The key idea was to notice that the discontinuity of the threshold element can be represented by a smooth non-linear approximation:

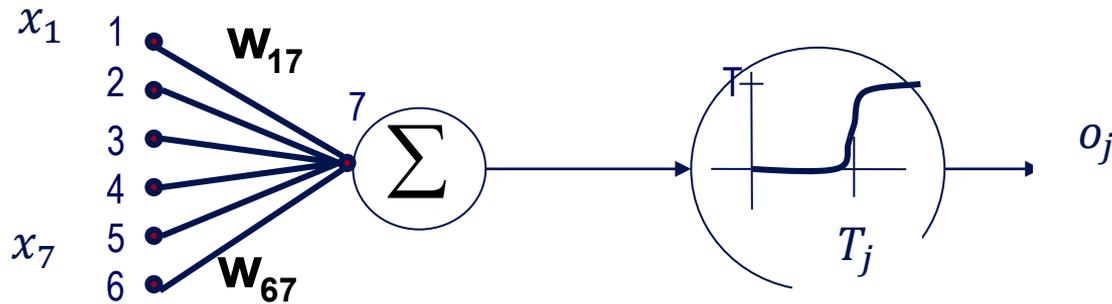
$$o_j = [1 + \exp\{-\mathbf{w} \cdot \mathbf{x}\}]^{-1}$$

(Rumelhart, Hinton, Williams, 1986), (Linnainmaa, 1970), see: <http://people.idsia.ch/~juergen/who-invented-backpropagation.html> )



# Model Neuron (Logistic)

- Use a non-linear, differentiable output function such as the sigmoid or logistic function



- Net input to a unit is defined as:  $net_j = \sum w_{ij} \cdot x_i$
- Output of a unit is defined as:  $o_j = \frac{1}{1 + e^{-(net_j - T_j)}}$