TTIC 31190: Natural Language Processing

Kevin Gimpel

Spring 2018

Lecture 5: Text Classification (Learning)

Assignment 1

- do not increment counts for the center word when counting context words:

`<s> the cat ran . </s>`

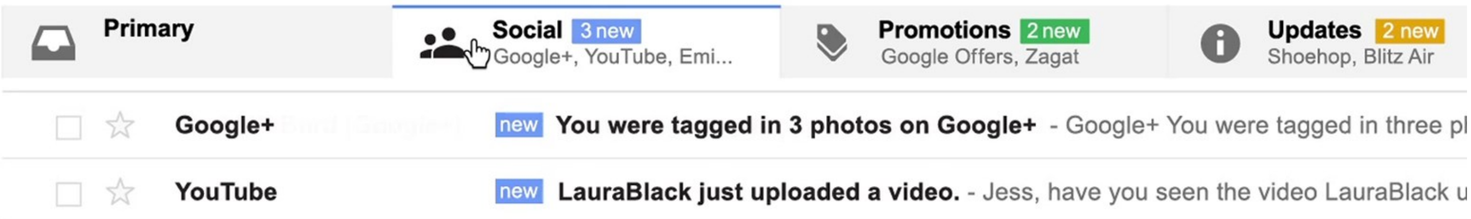
- for center word `cat`, do not increment count for `(cat, cat)`
- sorry for not making this clearer in the assignment

Roadmap

- words, morphology, lexical semantics
- text classification
- simple neural methods for NLP
- language modeling and word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

Text Classification

- simplest user-facing NLP application
- email (spam, priority, categories):



- sentiment:

- topic classification
- others?

Text Classification

- datasets
- classification
  - modeling
  - inference
  - learning

What is a classifier?

- a function from inputs ***x*** to classification labels *y*
- one simple type of classifier:
  - for any input ***x***, assign a score to each label *y*

$$\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

  - classify by choosing highest-scoring label:

$$\mathrm{classify}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname*{argmax}_{y} \ \mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

Modeling, Inference, Learning

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \ \ \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

Modeling, Inference, Learning

modeling: define score function

$$\mathrm{classify}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname*{argmax}_{y} \ \mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

- **Modeling**: How do we assign a score to an (***x***,*y*) pair using parameters ***w***?

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\operatorname{argmax}} \; \operatorname{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

- **Inference**: How do we efficiently search over the space of all labels?

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname*{argmax}_{y} \ \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

learning: choose $\boldsymbol{w}$

- **Learning**: How do we choose the weights ***w***?

Text Classification

- datasets
- classification
  - modeling
  - inference
  - learning

Linear Models

- parameters are arranged in a vector $\mathbf{w}$
- score function is linear in $\mathbf{w}$:

$$\text{score}(\boldsymbol{x}, y, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y) = \sum_i w_i f_i(\boldsymbol{x}, y)$$

- $\mathbf{f}$: vector of feature functions

- all features look at the label *y*!

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

- this may be different from what you’re used to
  - when using dense feature vectors in machine learning, the machine learning algorithm may create weights for every output label for every feature

- all features look at the label *y*!

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

- this may be different from what you're used to
  - when using dense feature vectors in machine learning, the machine learning algorithm may create weights for every output label for every feature
  - e.g., you specify a feature like:

$$f_3(\boldsymbol{x}, y) = \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

  - and then the ML code creates weights for all labels for this feature: $w_3^{\text{positive}}, w_3^{\text{negative}}$

Example: Part-of-Speech Tagging

- there are 45 part-of-speech (POS) tags in the Penn Treebank
- we don't want to create features for all 45*|V| combinations of tags and words
  - too many features to store in memory and too many feature weights to learn
- most words appear with <= 3 unique POS tags in the training set
- so we use feature count cut-offs and only create features for combinations that appear enough times in the training data

Text Classification

- datasets
- classification
  - modeling
  - inference
  - learning

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \; \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

learning: choose $\boldsymbol{w}$

- **Learning**: How should we choose values for the weights?

Text Classification

- modeling
- inference
- learning
  - empirical risk minimization
  - surrogate loss functions
  - gradient-based optimization

Learning: Empirical Risk Minimization

- In a machine learning course, you learn about many different learning frameworks

Learning: Empirical Risk Minimization

- In a machine learning course, you learn about many different learning frameworks
- Since we have limited time, we will be greedy and focus on a single framework that maximizes

$$\alpha \text{ ease\_of\_use} + \beta \text{ effectiveness} + \gamma \text{ applicability}$$

(for some positive constants α, β, γ)

We will start it today but continue to add to it later

Cost Functions

- cost function: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task
- usual conventions: $\text{cost}(y, y) = 0$

$$\text{cost}(y, y') = \text{cost}(y', y)$$

Cost Functions

- cost function: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task

- for classification, what cost should we use?

Cost Functions

- cost function: scores outputs against a gold standard

$$\mathrm{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task

- for classification, what cost should we use?

$$\mathrm{cost}(y, y') = \mathbb{I}[y \neq y']$$

- how about for other NLP tasks?

Risk Minimization

- given training data: $\mathcal{T} = \{\langle \boldsymbol{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$
  where each $y^{(i)} \in \mathcal{L}$ is a label
- assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution $P(\boldsymbol{x}, y)$

Risk Minimization

- given training data: $\mathcal{T} = \{\langle \boldsymbol{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$
  where each $y^{(i)} \in \mathcal{L}$ is a label
- assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution $P(\boldsymbol{x}, y)$
- we want to solve the following:

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \ \mathbb{E}_{P(\boldsymbol{x},y)} \left[\mathrm{cost}(y, \mathrm{classify}(\boldsymbol{x}, \boldsymbol{w})) \right]$$

Risk Minimization

- given training data: $\mathcal{T} = \{\langle \boldsymbol{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$
  where each $y^{(i)} \in \mathcal{L}$ is a label
- assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution $P(\boldsymbol{x}, y)$
- we want to solve the following:

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \ \mathbb{E}_{P(\boldsymbol{x},y)} [\text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))]$$

problem: *P* is unknown

Empirical Risk Minimization
(Vapnik et al.)

- replace expectation with sum over examples:

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w})) \right]$$

$$\downarrow$$

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \; \sum_{i=1}^{|\mathcal{T}|} \text{cost}(y^{(i)}, \text{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

Empirical Risk Minimization
(Vapnik et al.)

- replace expectation with sum over examples:

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))\right]$$

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \; \sum_{i=1}^{|\mathcal{T}|} \text{cost}(y^{(i)}, \text{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

problem: NP-hard even for binary classification with linear models solution: replace "cost loss" (also called "0-1" loss) with a **surrogate** function that is easier to optimize

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \mathrm{cost}(y^{(i)}, \mathrm{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

generalize to permit any loss function

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \mathrm{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w})$$

solution: replace "cost loss" (also called "0-1" loss) with a **surrogate** function that is easier to optimize

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \text{cost}(y^{(i)}, \text{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

generalize to permit any loss function

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{|\mathcal{T}|} \text{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w})$$

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

Text Classification

- modeling
- inference
- learning
  - empirical risk minimization
  - surrogate loss functions
  - gradient-based optimization

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

why is this so difficult to optimize?

Surrogate Loss Functions

cost loss / 0-1 loss: $\mathrm{loss}_{\mathrm{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \mathrm{cost}(y, \mathrm{classify}(\boldsymbol{x}, \boldsymbol{w}))$

why is this so difficult to optimize?
not necessarily continuous, can't use
gradient-based optimization

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

this is continuous, but what are its drawbacks?

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

loss function underlying perceptron algorithm
(Rosenblatt, 1957-58)

Surrogate Loss Functions

cost loss / 0-1 loss: $\mathrm{loss}_{\mathrm{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \mathrm{cost}(y, \mathrm{classify}(\boldsymbol{x}, \boldsymbol{w}))$

perceptron loss:

$$\mathrm{loss}_{\mathrm{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \ \mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

hinge loss:

$$\mathrm{loss}_{\mathrm{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \mathrm{cost}(y, y'))$$

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$

loss function underlying support vector machines

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$

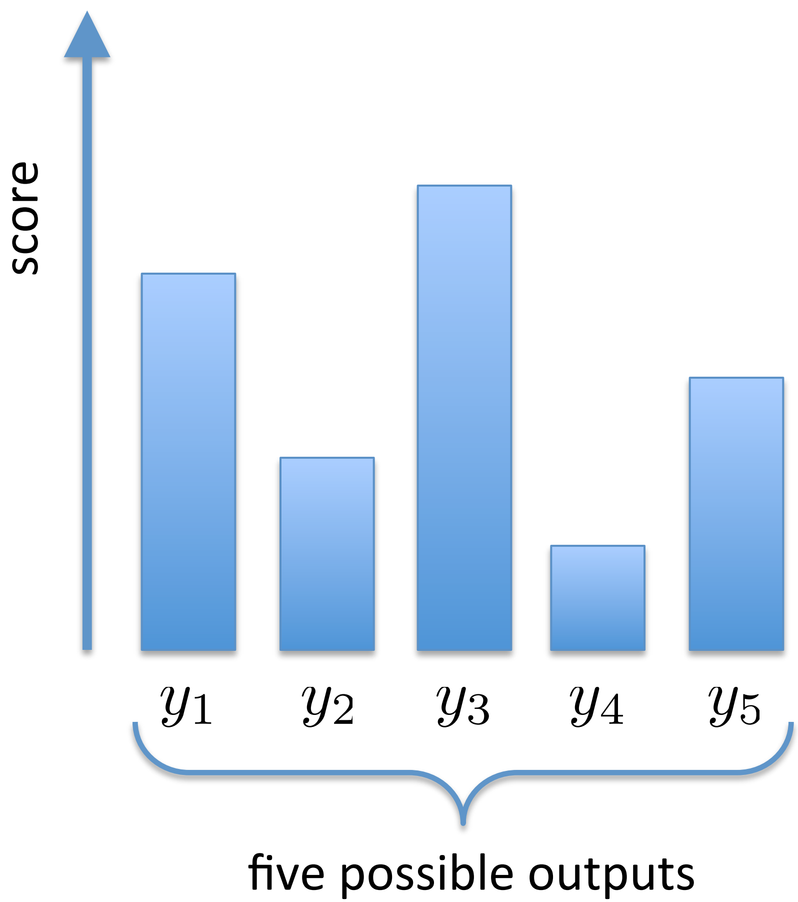
hinge loss for our classification setting:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \delta \mathbb{I}(y \neq y'))$$

tunable hyperparameter

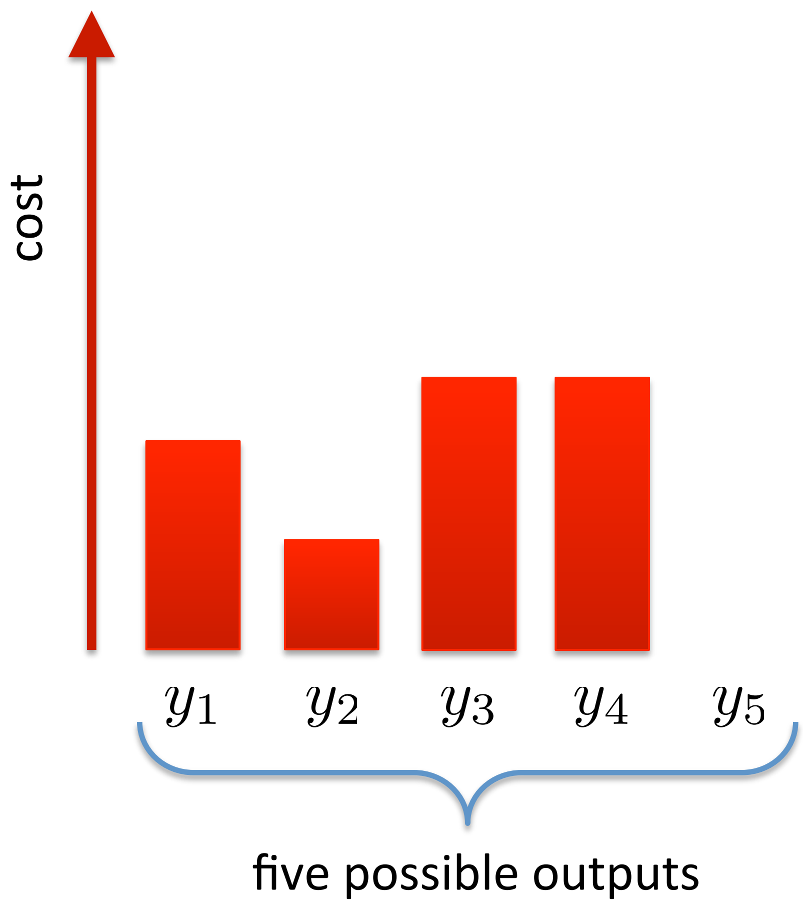
Visualization

for a single input *x*



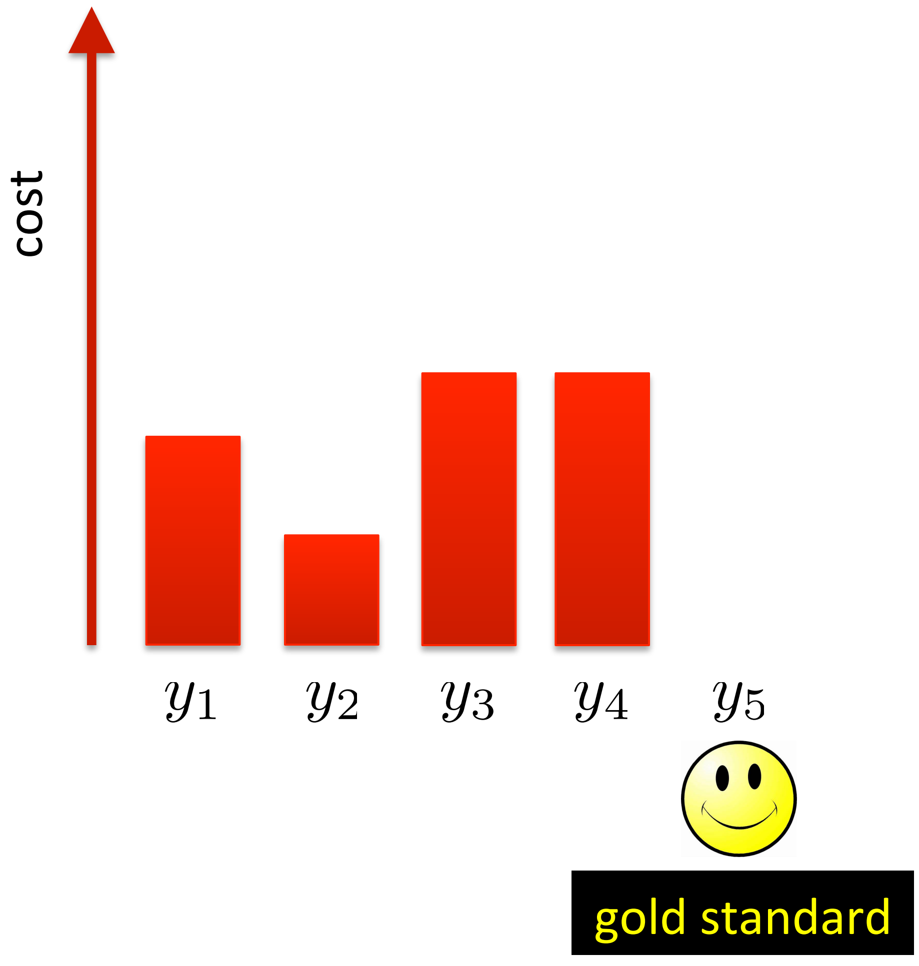
Visualization

for a single input ***x***



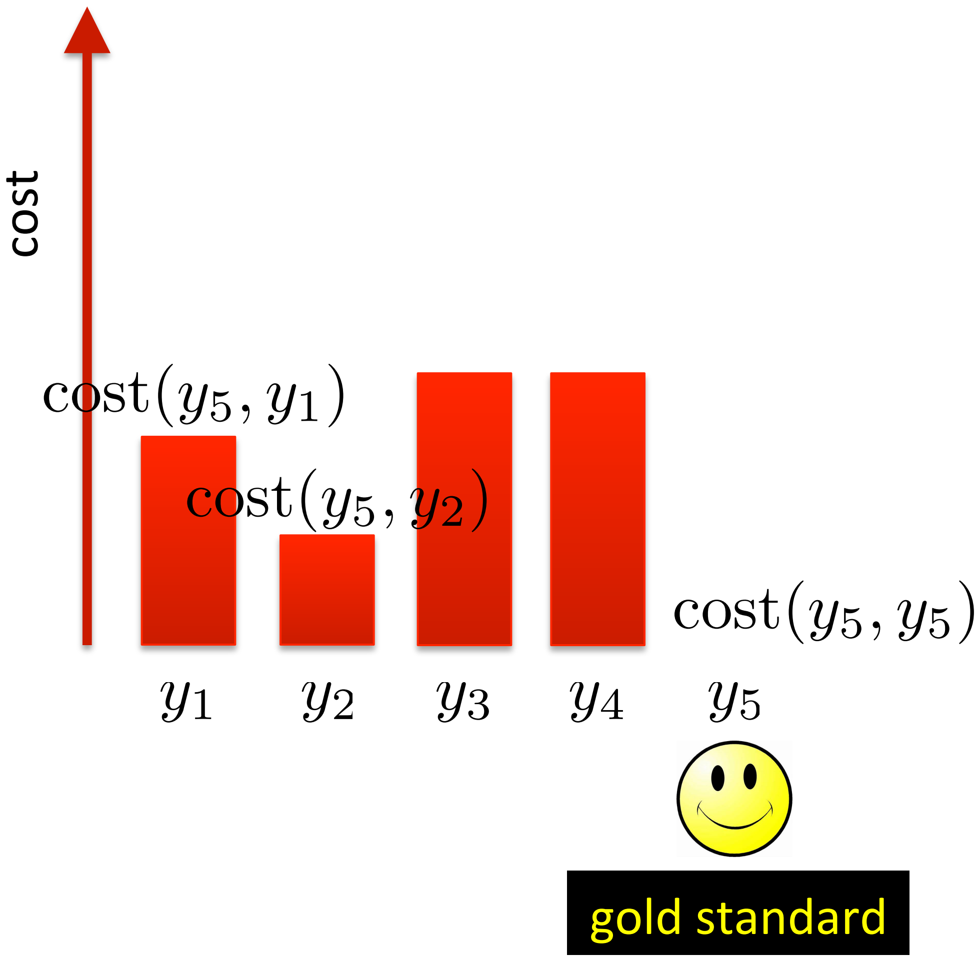
Visualization

for a single input ***x***



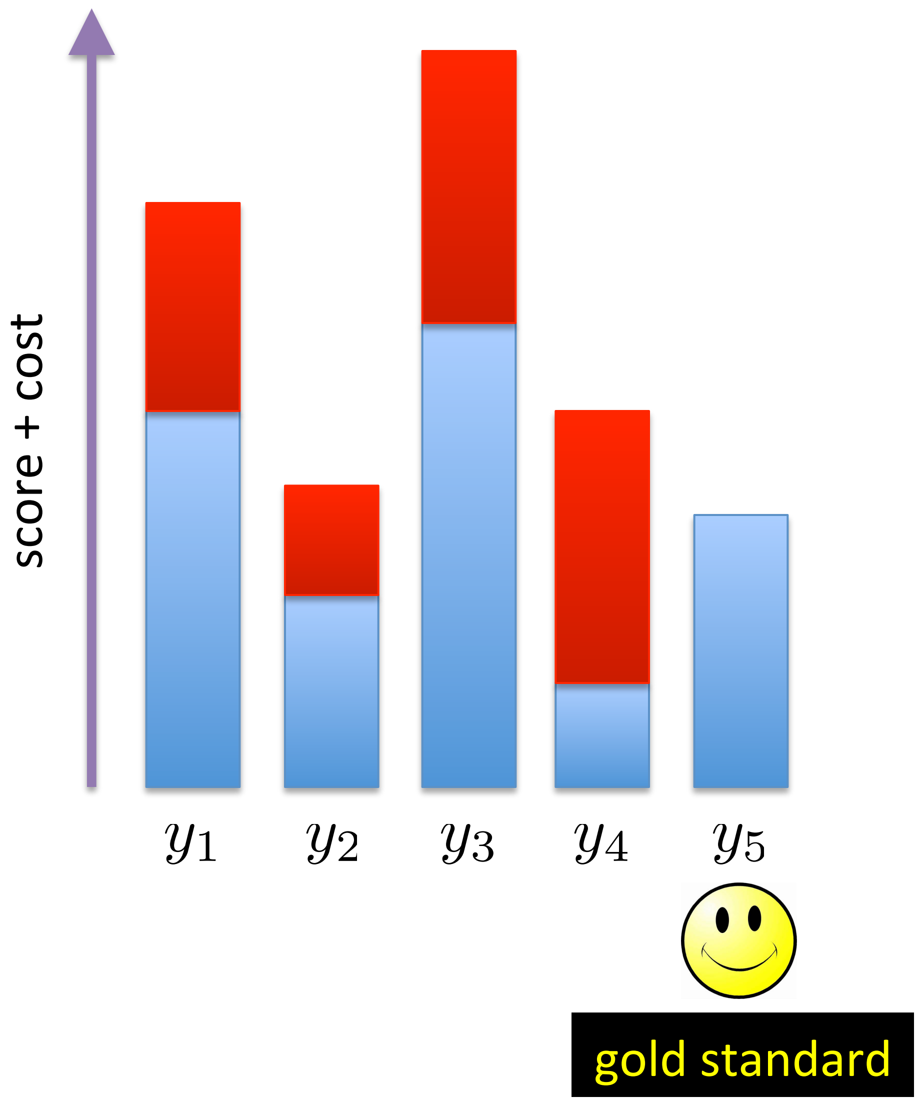
Visualization

for a single input ***x***



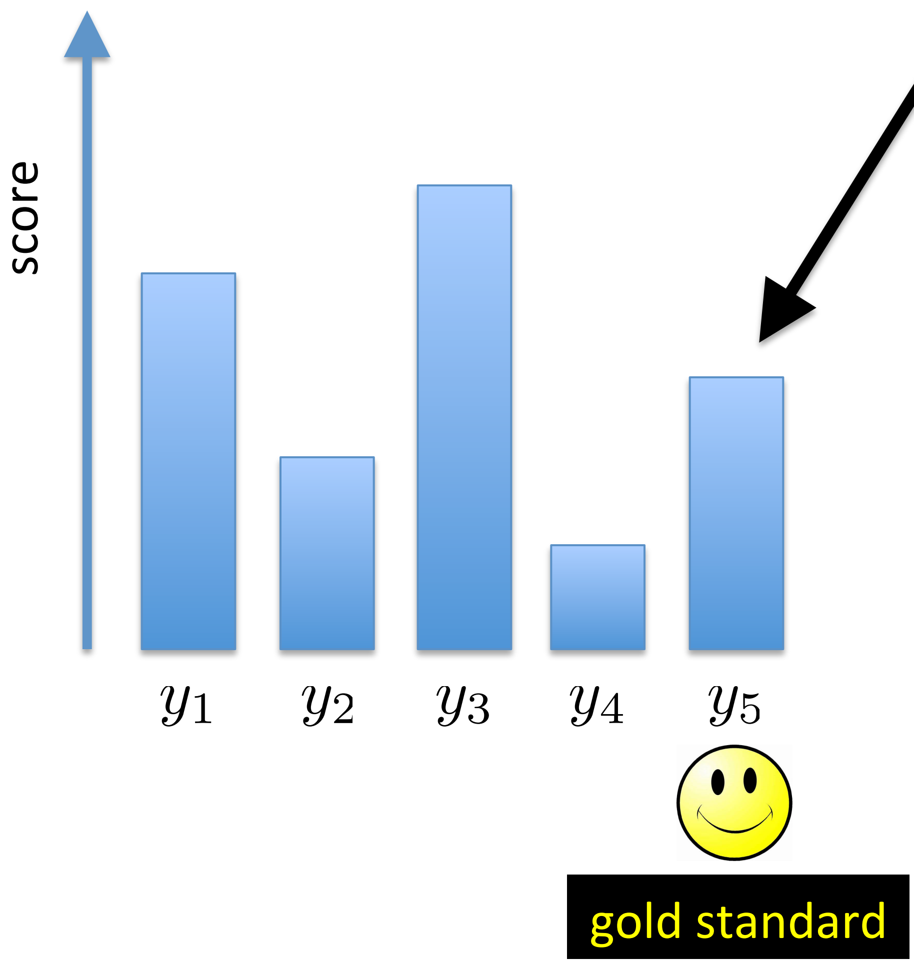
Visualization

for a single input ***x***

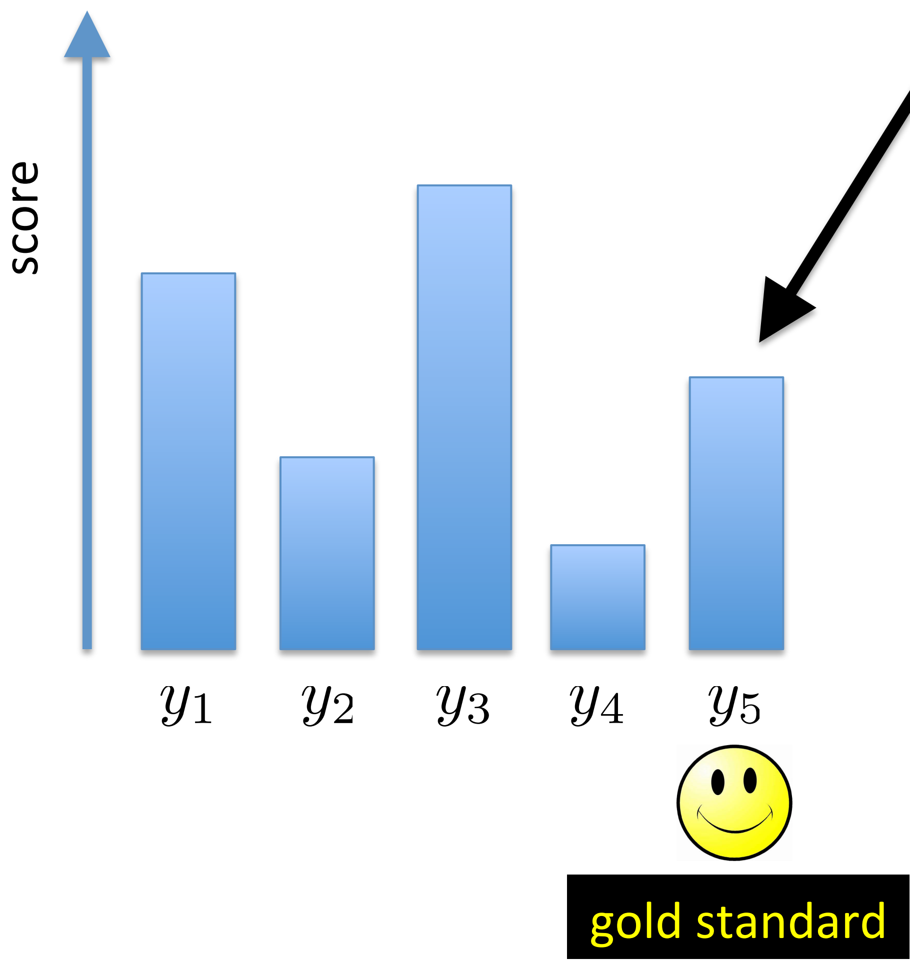


$$\mathrm{loss}_{\mathrm{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

$$\mathrm{loss}_{\mathrm{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

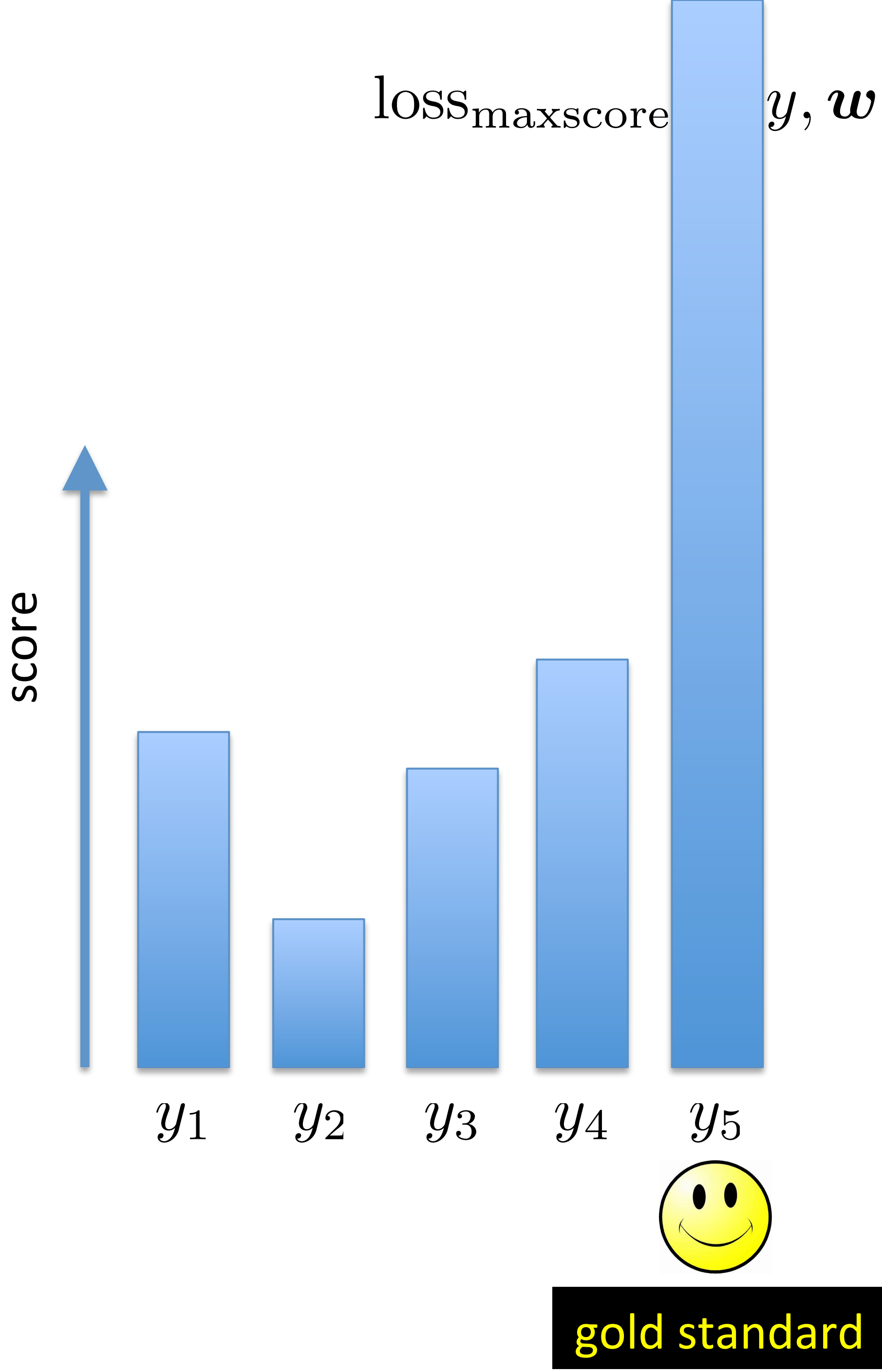


$$\mathrm{loss}_{\mathrm{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w})$$



effect of learning?

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$



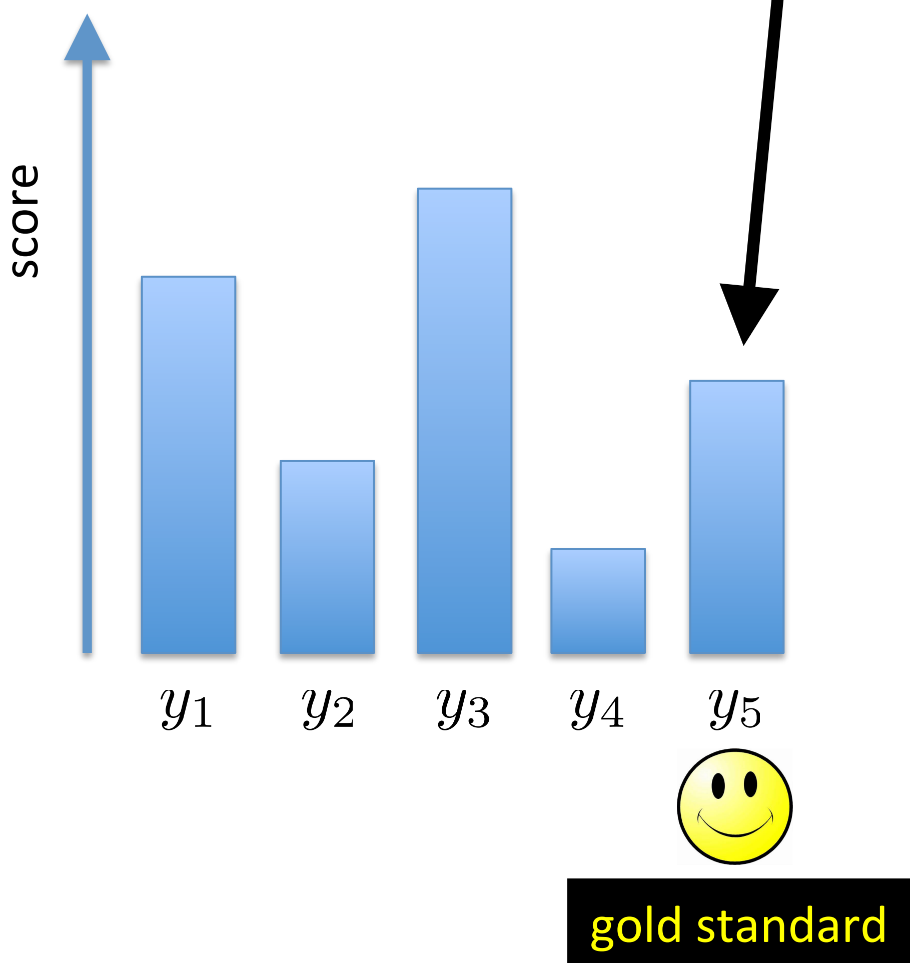
effect of learning:
score of gold standard
will go to infinity

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \ \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

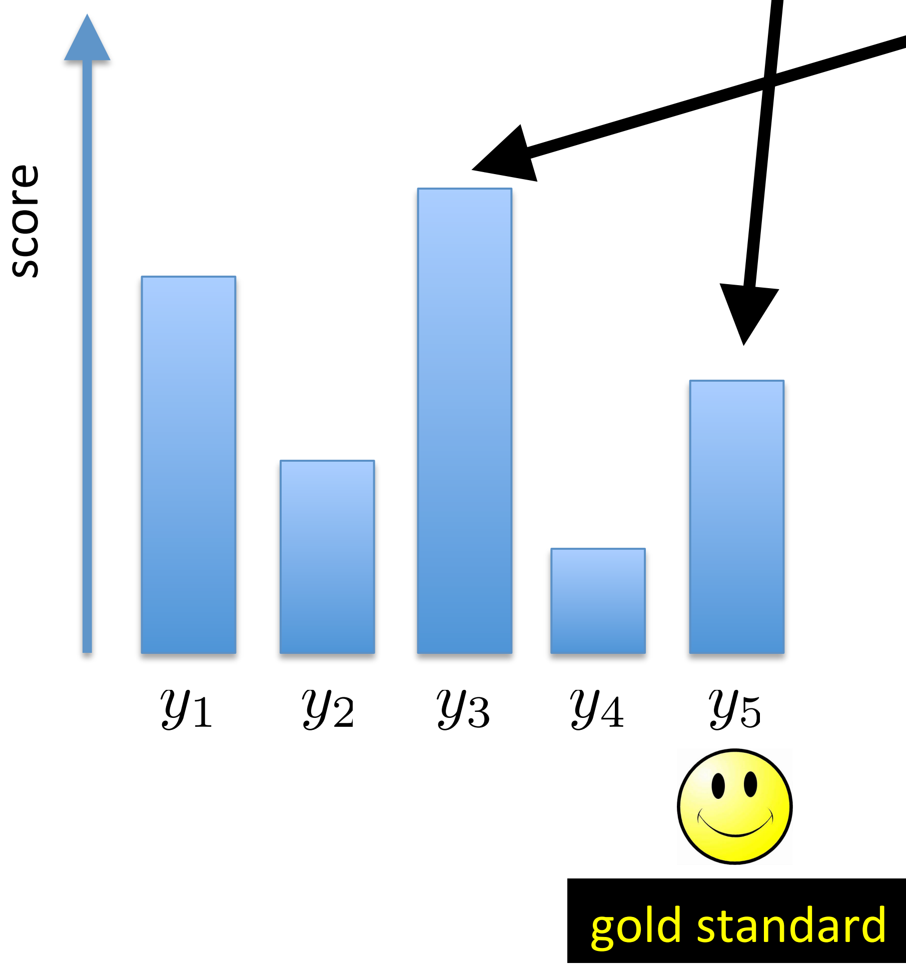
perceptron loss:

$$\mathrm{loss}_{\mathrm{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \underbrace{\max_{y' \in \mathcal{L}} \mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w})}$$



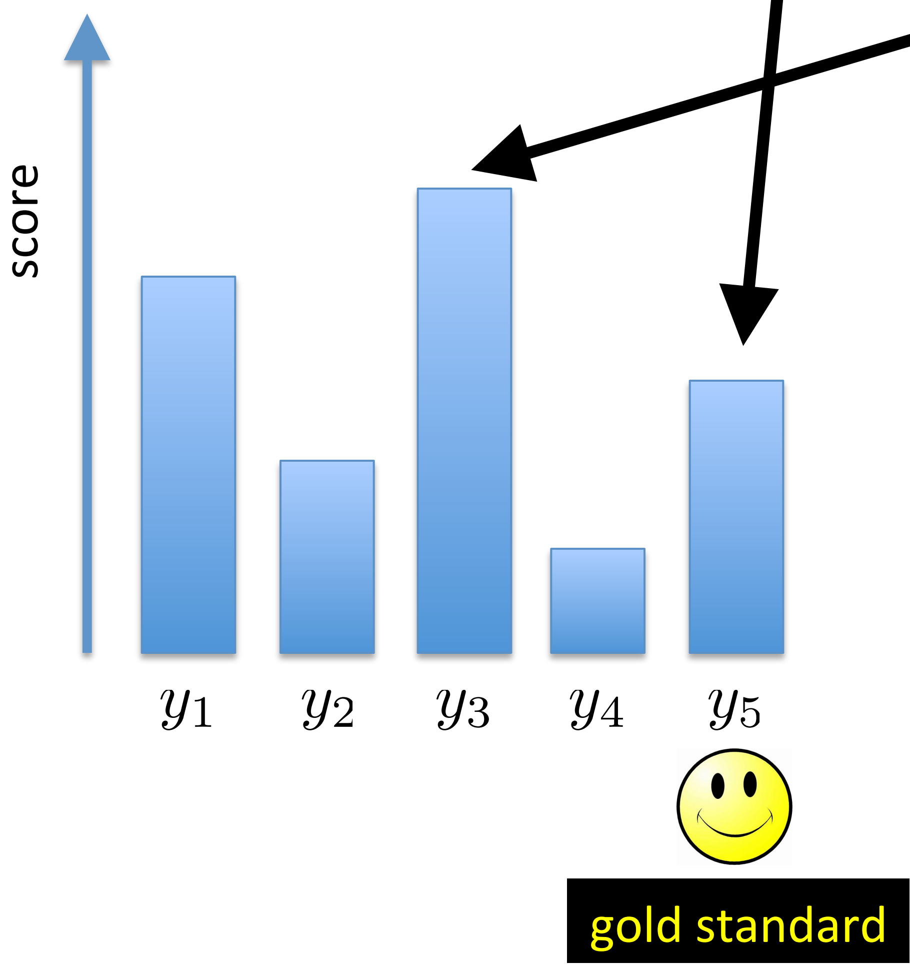
perceptron loss:

$$\mathrm{loss}_{\mathrm{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \underbrace{\max_{y' \in \mathcal{L}} \mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w})}$$



perceptron loss:

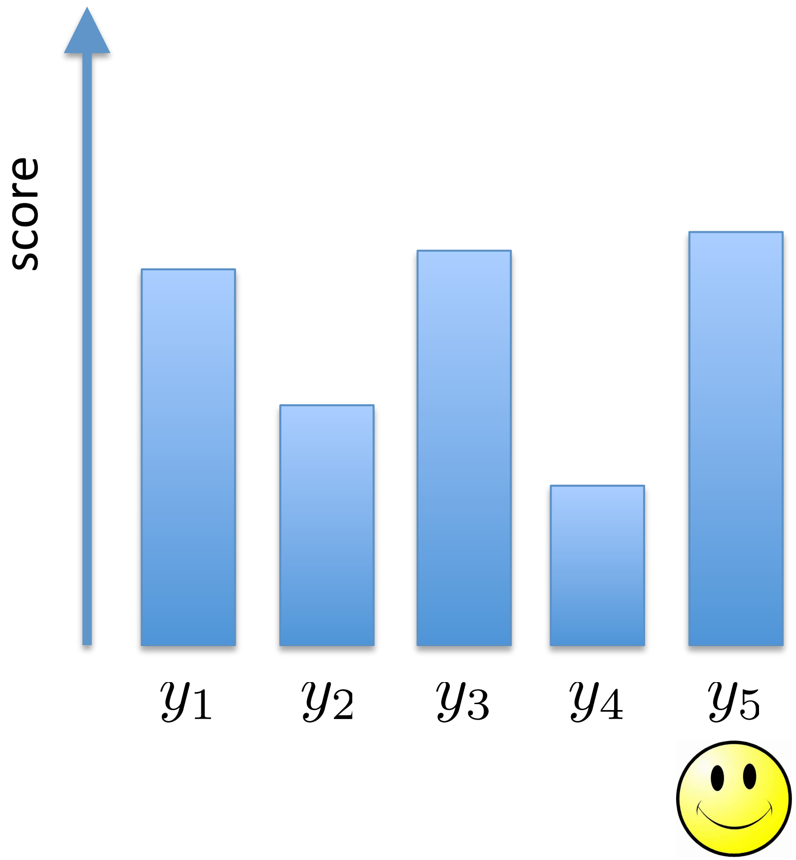
$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \underbrace{\max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})}$$



effect of learning?

perceptron loss:

$$\mathrm{loss}_{\mathrm{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \ \mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w})$$



gold standard

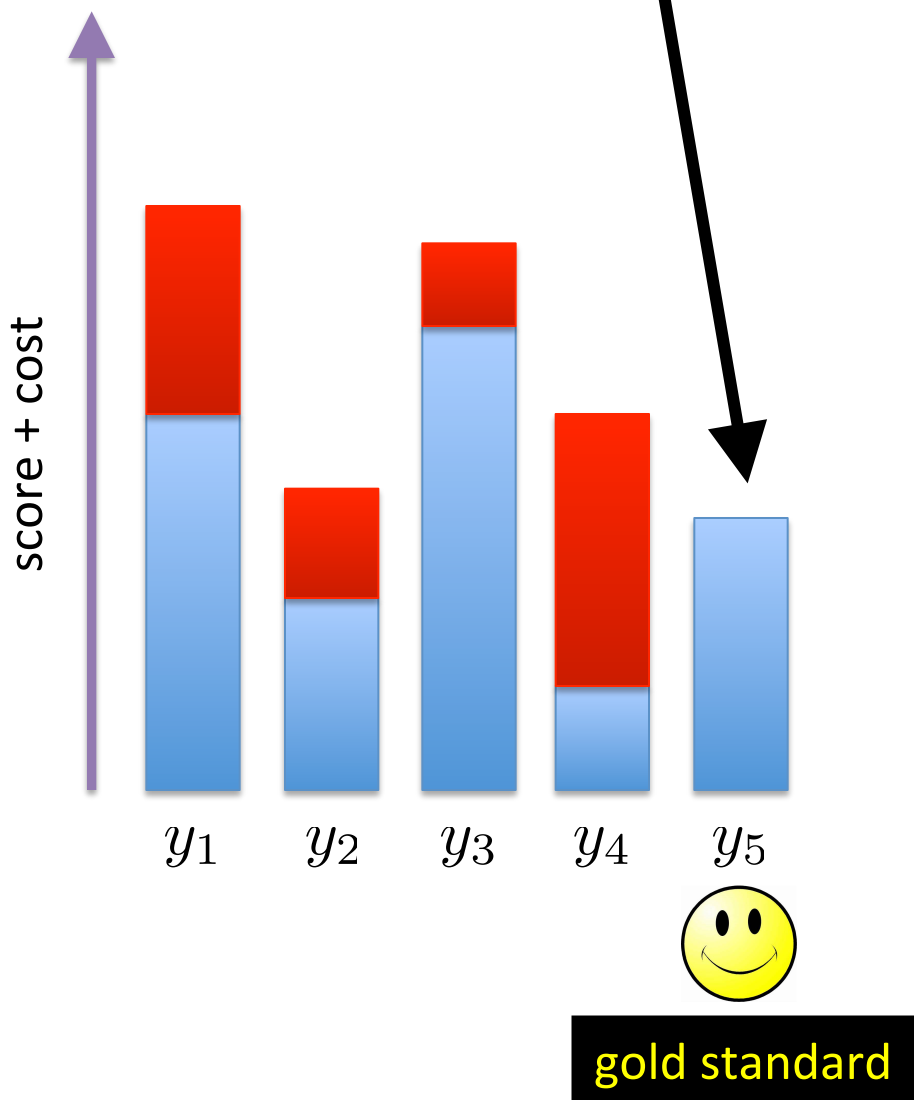
effect of learning:
gold standard will have highest score

hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$

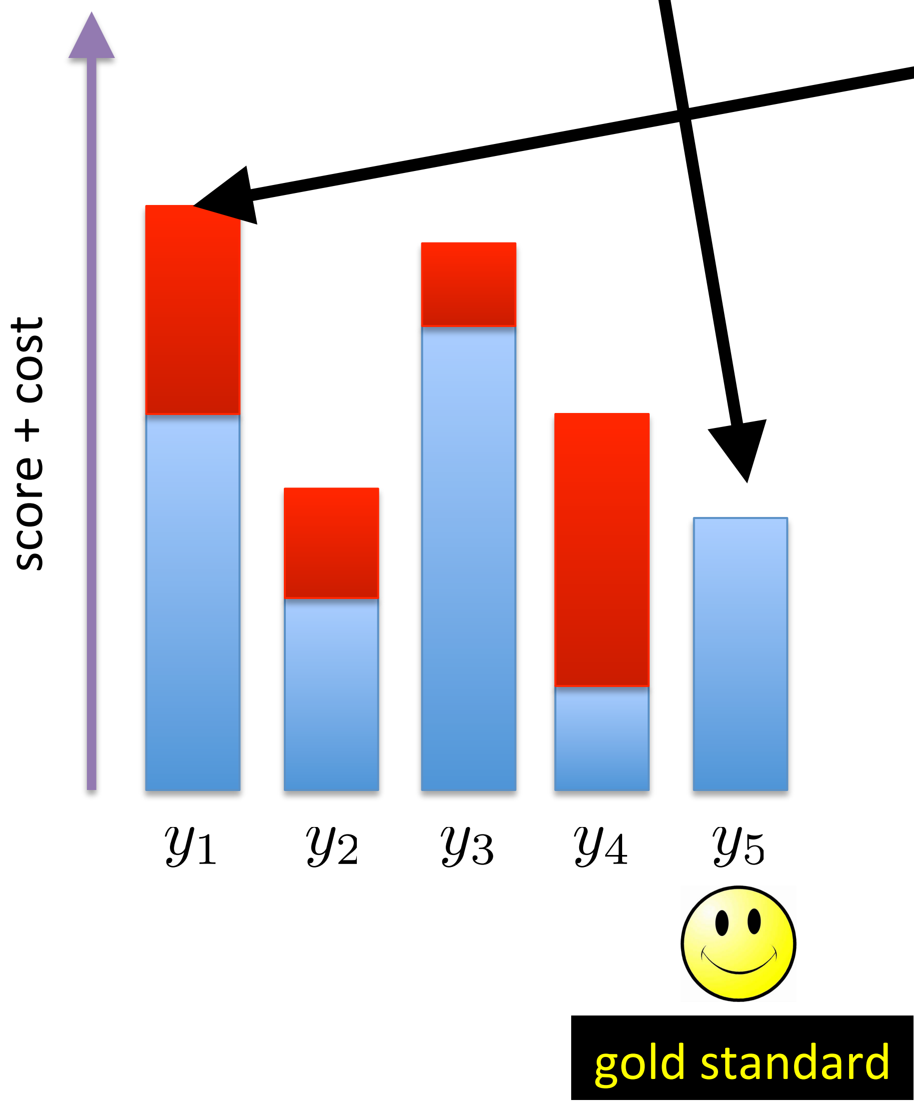
hinge loss:

$$\mathrm{loss}_{\mathrm{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \underbrace{\max_{y' \in \mathcal{L}} \left(\mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \mathrm{cost}(y, y')\right)}$$



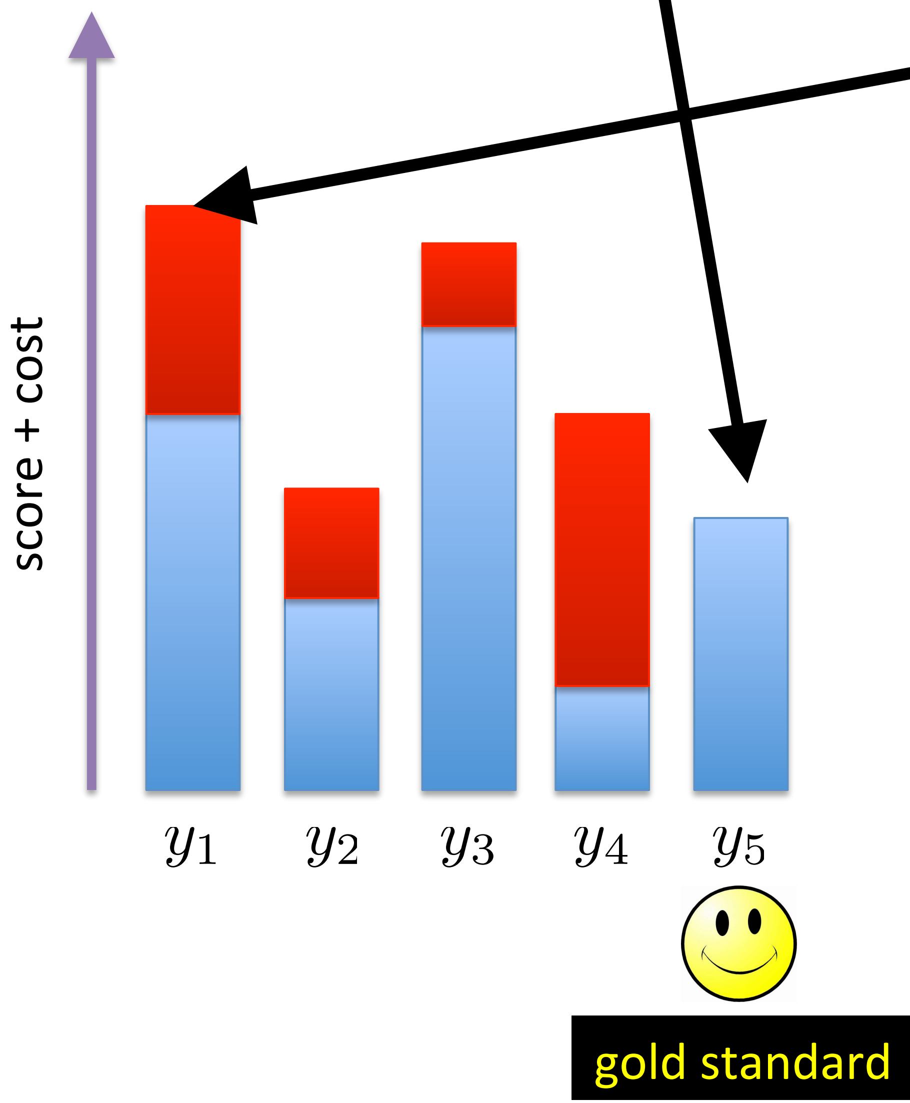
hinge loss:

$$\mathrm{loss}_{\mathrm{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \underbrace{\max_{y' \in \mathcal{L}} \left(\mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \mathrm{cost}(y, y')\right)}$$



hinge loss:

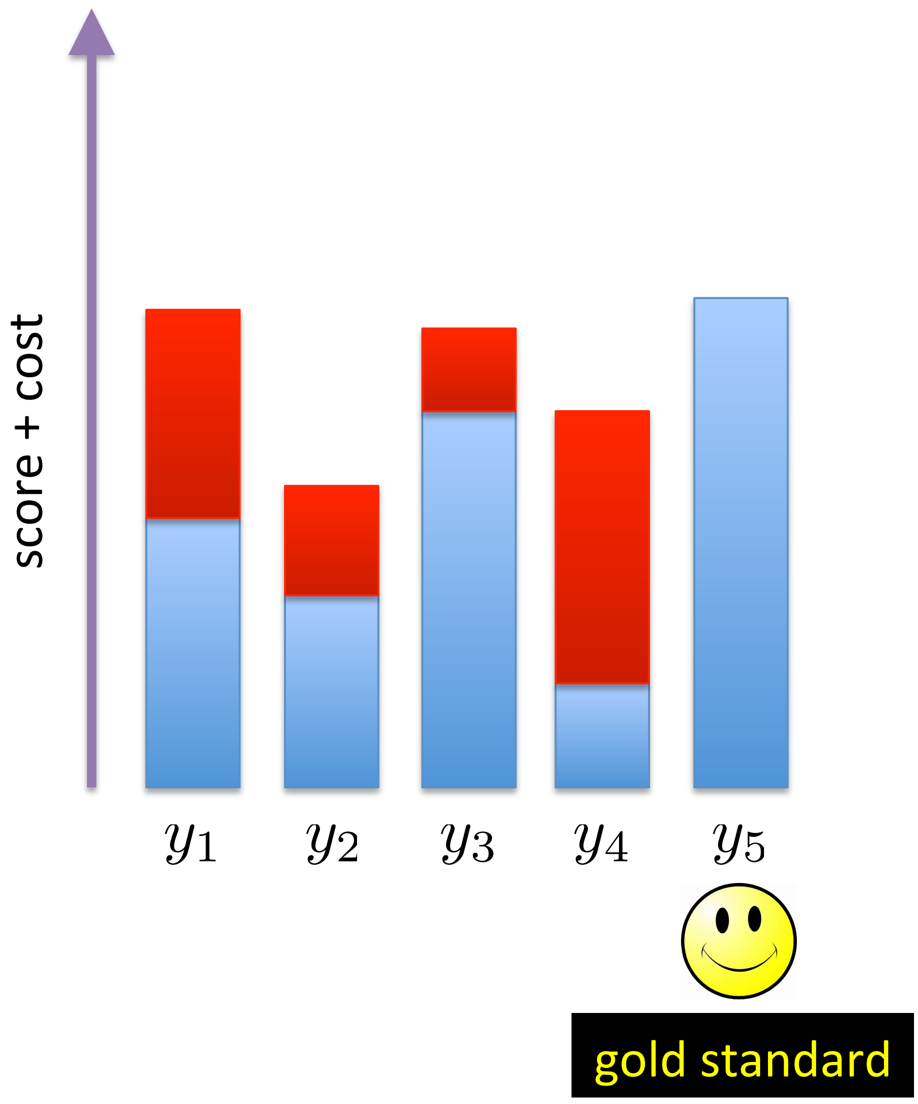
$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \underbrace{\max_{y' \in \mathcal{L}} \left(\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y')\right)}$$



effect of learning?

hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \left(\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y')\right)$$



effect of learning:
score of gold standard will be higher than score+cost of all others

Log Loss

$$\text{loss}_{\text{log}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x})$$

- minimize negative log of conditional probability of output given input
  - sometimes called "maximizing conditional likelihood"
- but wait, we don't have a probabilistic model, we just have a score function

Score → Probability

- can turn score into probability by exponentiating (to make it positive) and normalizing:

$$p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) \propto \exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}$$

$$p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) = \frac{\exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}}$$

- this is often called a “softmax” function

Log Loss

$$\begin{aligned}\mathrm{loss}_{\mathrm{log}}(\boldsymbol{x}, y, \boldsymbol{w}) &= -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) \\ &= -\log \frac{\exp\{\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w})\}} \\ &= -\,\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \log \sum_{y' \in \mathcal{L}} \exp\{\mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w})\}\end{aligned}$$

- similar to perceptron loss!
- replace max with “softmax” (a different kind of softmax)

$$\mathrm{loss}_{\mathrm{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\mathrm{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}}\ \mathrm{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

Log Loss

$$\begin{aligned}
\text{loss}_{\text{log}}(\boldsymbol{x}, y, \boldsymbol{w}) &= -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) \\
&= -\log \frac{\exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}} \\
&= -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}
\end{aligned}$$

- si
- ju

log loss is used in:
logistic regression classifiers,
conditional random fields,
maximum entropy (“maxent”) models

los $\ldots \; y' \in \mathcal{L} \; \ldots \; w)$

Log Loss

$$\begin{aligned}
\text{loss}_{\text{log}}(\boldsymbol{x}, y, \boldsymbol{w}) &= -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) \\
&= -\log \frac{\exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}} \\
&= -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}
\end{aligned}$$

- issue: can be very expensive due to summation over all possible outputs!

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \;\; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w})) \right]$$

$$\downarrow$$

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \;\; \sum_{i=1}^{|\mathcal{T}|} \text{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w})$$

Regularized Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \ \ \mathbb{E}_{P(\boldsymbol{x},y)}\left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}))\right]$$

$$\downarrow$$

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \ \ \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

Regularized Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \ \mathbb{E}_{P(\boldsymbol{x},y)}\left[\text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))\right]$$

regularization strength

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \ \sum_{i=1}^{|\mathcal{T}|} \text{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda \underbrace{R(\boldsymbol{w})}_{\textbf{regularization term}}$$

Regularization Terms

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

- most common: penalize large parameter values
- intuition: large parameters might be instances of overfitting
- examples:

***L_2* regularization**:
(also called Tikhonov regularization
or ridge regression)

***L_1* regularization**:
(also called basis pursuit or LASSO)

Regularization Terms

L₂ **regularization**: $R_{\mathrm{L2}}(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_i w_i^2$

differentiable, widely-used

L₁ **regularization**: $R_{\mathrm{L1}}(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_i |w_i|$

not differentiable (but is subdifferentiable)

leads to sparse solutions (many parameters become zero!)

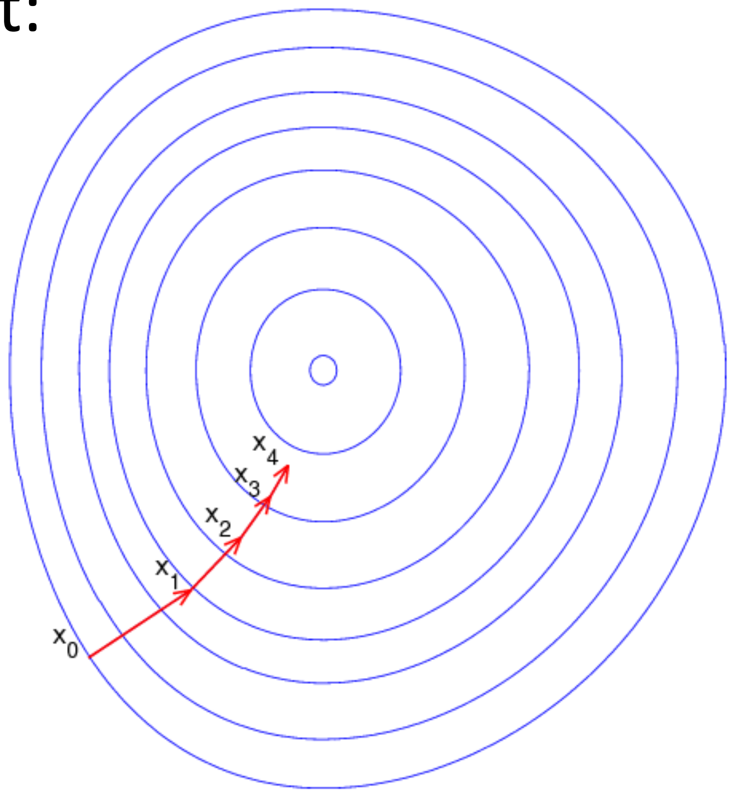
Text Classification

- modeling
- inference
- learning
  - empirical risk minimization
  - surrogate loss functions
  - gradient-based optimization

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

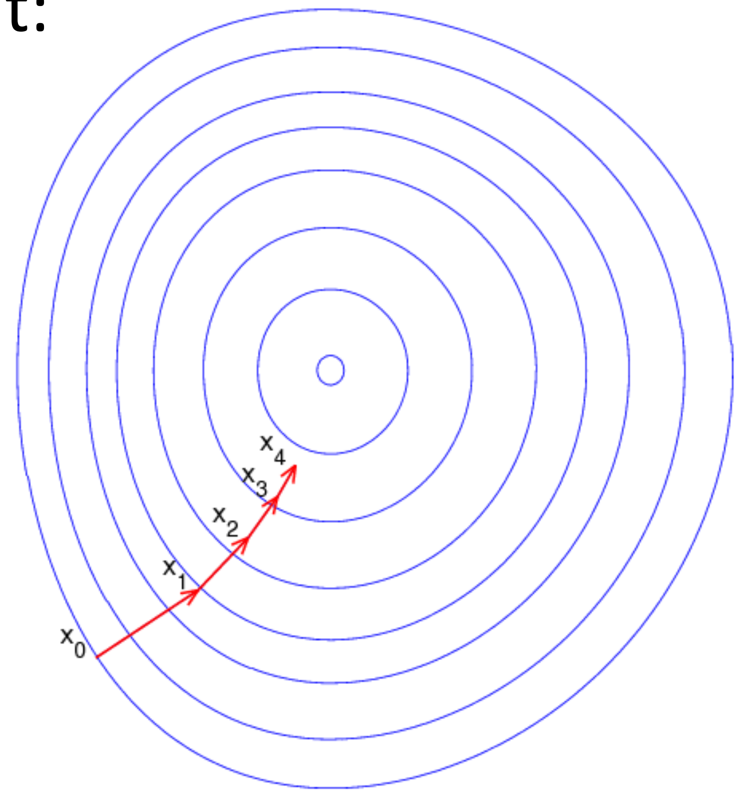
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$



Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$



$\eta^{(t)}$: stepsize at iteration *t*

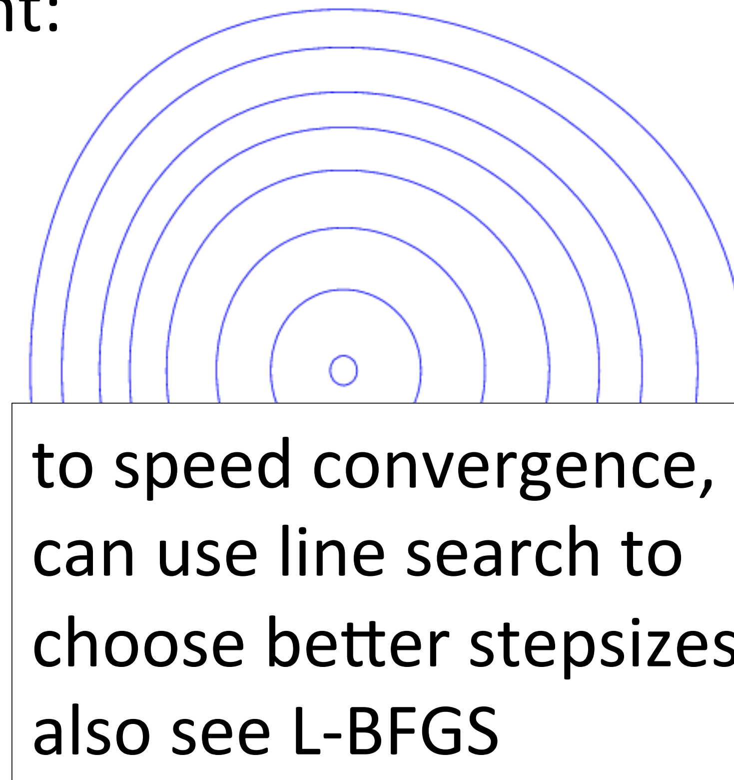
$\boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$: gradient of objective function

- with conditions on stepsize and objective function, will converge to local minimum

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$



$\eta^{(t)}$: stepsize at iteration *t*

$\boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$: gradient of objective function

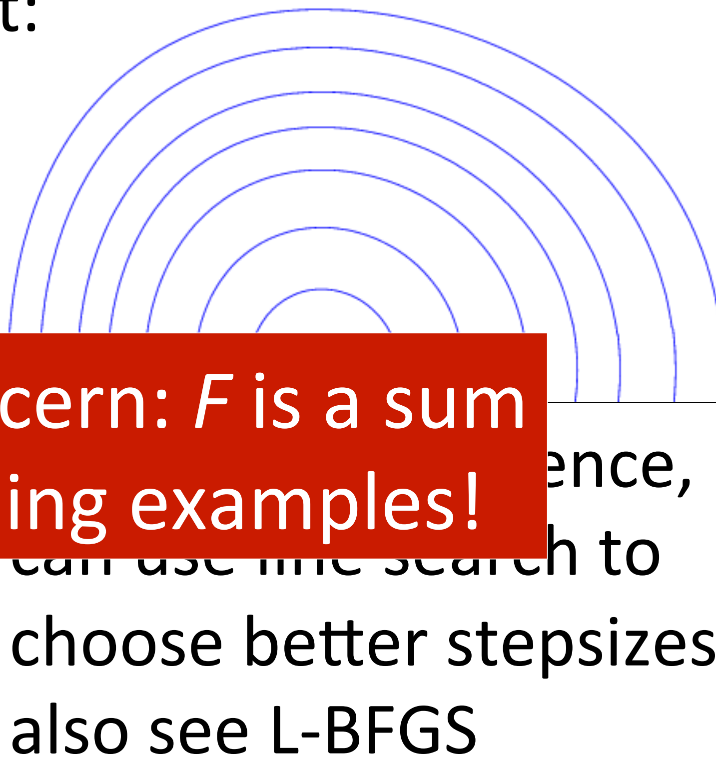
to speed convergence, can use line search to choose better stepsizes; also see L-BFGS

- with conditions on stepsize and objective function, will converge to local minimum

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$



$\eta^{(t)}$: stepsize

$\boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$: grad

objective function

efficiency concern: *F* is a sum over all training examples!

ence,

h to

choose better stepsizes;

also see L-BFGS

- with conditions on stepsize and objective function, will converge to local minimum

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$

$\eta^{(t)}$: stepsize a

$\boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$: grad

objective funct

ence,

h to

psizes;

efficiency concern: *F* is a sum over all training examples!

every parameter update requires iterating through entire training set

- with conditio on, will converge to local minimum

Gradient Descent

- minimizes a function F by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla F(\boldsymbol{\theta}^{(t)})$$

$\eta^{(t)}$: stepsize a

$\nabla F(\boldsymbol{\theta}^{(t)})$: grad

objective funct

ence,

h to

psizes;

efficiency concern: *F* is a sum over all training examples!

every parameter update requires iterating through entire training set

“batch” algorithm

- with conditio

on,

will converge

Stochastic Gradient Descent

- applicable when objective function is a sum
- like gradient descent, except calculates gradient on a single example at a time ("online") or on a small set of examples ("mini-batch")

Stochastic Gradient Descent

- applicable when objective function is a sum
- like gradient descent, except calculates gradient on a single example at a time ("online") or on a small set of examples ("mini-batch")
- converges much faster than (batch) gradient descent
- with conditions on stepsize and objective function, will converge to local minimum
- there are many popular variants:

  SGD+momentum, AdaGrad, AdaDelta, Adam, RMSprop, etc.

What if *F* is not differentiable?

- some loss functions are not differentiable:

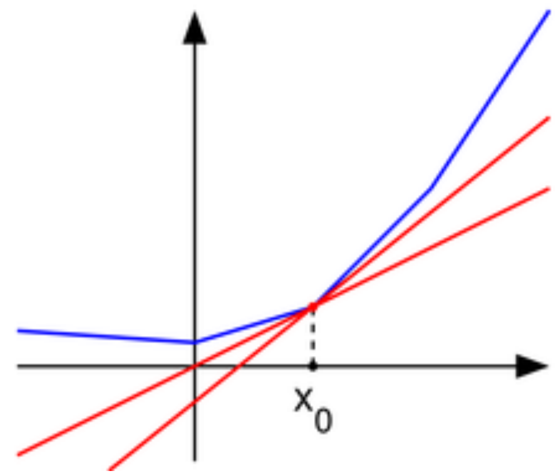
$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \ \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$

- but they *are* subdifferentiable, so we can compute subgradients and use (stochastic) subgradient descent

Subderivatives

- subderivative: generalization of derivative for nondifferentiable, convex functions



- there may be multiple subderivatives at a point (red lines)

- this set is called the subdifferential
- a convex function *g* is differentiable at point x_0 if and only if the subdifferential of *g* at x_0 contains only the derivative of *g* at x_0

Stochastic Subgradient Descent

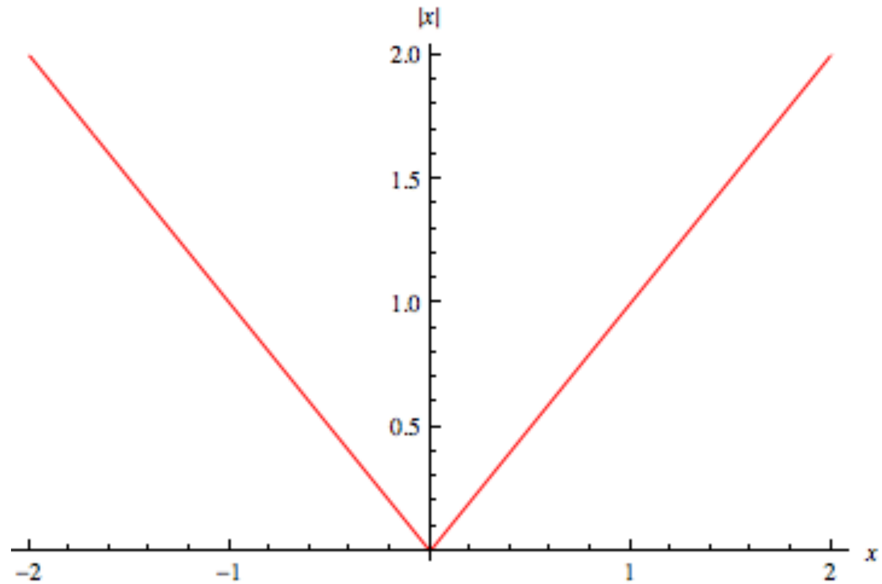
- just like stochastic gradient descent, except replace gradients with subgradients
- similarly strong theoretical guarantees

Calculating Subgradients

- at points of differentiability, just use your rules for calculating gradients
- at points of nondifferentiability, just find a single subgradient; *any* subgradient will do

Subgradient Examples

$$f(x) = |x| = \max(x, -x)$$



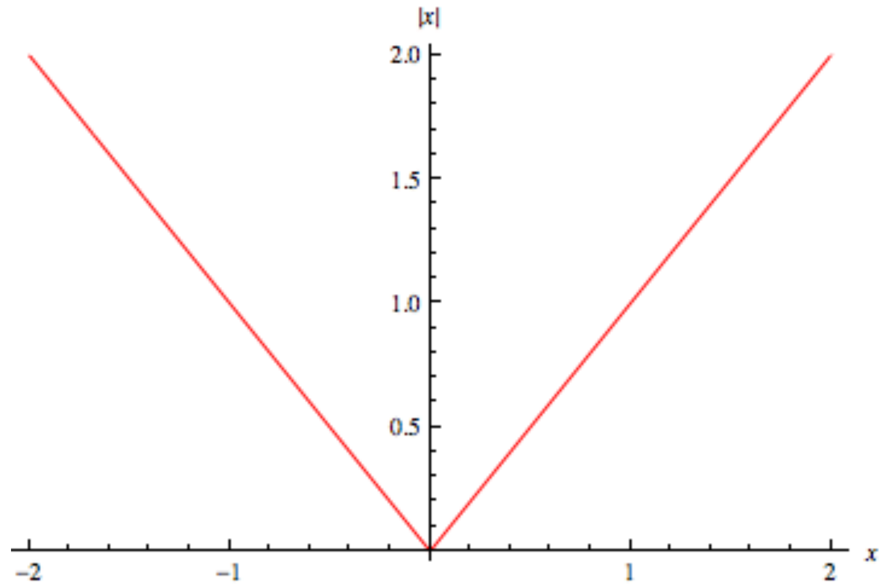
$$x < 0: \;\; \partial f(x) =$$

$$x > 0: \;\; \partial f(x) =$$

$$x = 0: \;\; \partial f(x) =$$

Subgradient Examples

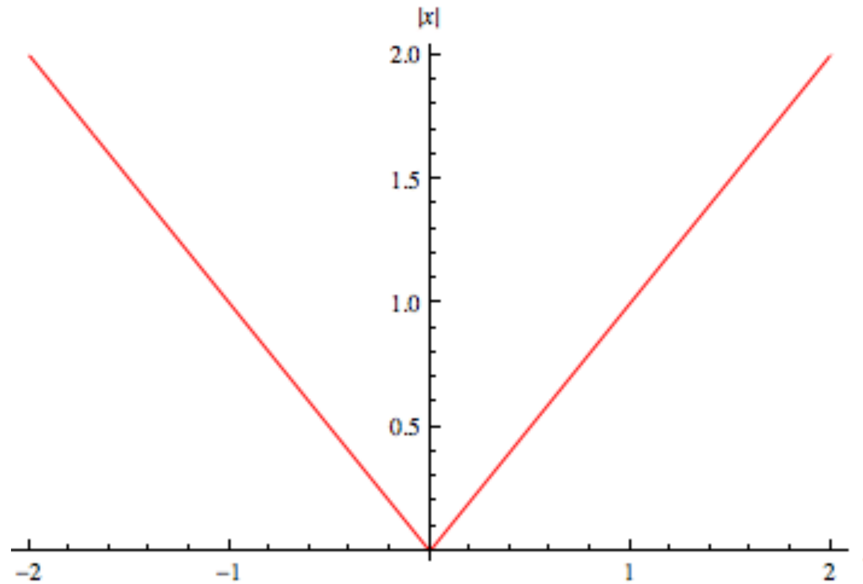
$$f(x) = |x| = \max(x, -x)$$



$$x < 0: \;\; \partial f(x) = \{-1\}$$
$$x > 0: \;\; \partial f(x) = \{1\}$$
$$x = 0: \;\; \partial f(x) =$$

Subgradient Examples

$$f(x) = |x| = \max(x, -x)$$



$$x < 0: \;\; \partial f(x) = \{-1\}$$
$$x > 0: \;\; \partial f(x) = \{1\}$$
$$x = 0: \;\; \partial f(x) = [-1, 1]$$

- to find a subgradient of max of convex functions at a point, choose one function that achieves the max at that point and choose any of its subgradients at the point