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:
  \textit{<s> the cat ran . </s>}

• for center word \texttt{cat}, do not increment count for \texttt{(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 $\mathbf{x}$ to classification labels $y$
• one simple type of classifier:
  – for any input $\mathbf{x}$, assign a score to each label $y$
    
    $$\text{score}(\mathbf{x}, y, \mathbf{w})$$

  – classify by choosing highest-scoring label:

    $$\text{classify}(\mathbf{x}, \mathbf{w}) = \arg\max_y \text{score}(\mathbf{x}, y, \mathbf{w})$$
Modeling, Inference, Learning

\[ \text{classify}(\mathbf{x}, \mathbf{w}) = \arg\max_y \text{ score}(\mathbf{x}, y, \mathbf{w}) \]
Modeling, Inference, Learning

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

\[
\text{classify}(x, w) = \arg\max_y \text{score}(x, y, w)
\]
Modeling, Inference, Learning

**inference**: solve argmax

\[ \text{classify}(x, w) = \arg\max_y \text{score}(x, y, w) \]

**modeling**: define score function

• **Inference**: How do we efficiently search over the space of all labels?
Modeling, Inference, Learning

**inference**: solve $\arg\max$

$$\text{classify}(x, w) = \arg\max_y \text{score}(x, y, w)$$

**modeling**: define score function

**learning**: choose $w$

- **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 $\mathbf{w}$:

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

• $\mathbf{f}$ : vector of feature functions
• all features look at the label $y$!

$$f_1(x, y) = \mathbb{I}[y = \text{positive}] \land \mathbb{I}[x \text{ contains } great]$$

$$f_2(x, y) = \mathbb{I}[y = \text{negative}] \land \mathbb{I}[x \text{ contains } great]$$

• 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 $y$!

$$f_1(x, y) = \mathbb{I}[y = \text{positive}] \land \mathbb{I}[x \text{ contains great}]$$

$$f_2(x, y) = \mathbb{I}[y = \text{negative}] \land \mathbb{I}[x \text{ contains great}]$$

• 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:

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

  – 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 $\leq 3$ 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

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

```
\text{classify}(x, w) = \underset{y}{\text{argmax}} \quad \text{score}(x, y, w)
```

- **Inference**: solve \( \text{argmax} \)
- **Modeling**: define score function
- **Learning**: choose \( w \)
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

\[ \alpha \text{ ease}_\text{of}_\text{use} + \beta \text{ effectiveness} + \gamma \text{ applicability} \]

(for some positive constants \( \alpha, \beta, \gamma \))

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

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

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

• for classification, what cost should we use?

\[
\text{cost}(y, y') = \mathbb{1}[y \neq y']
\]

• how about for other NLP tasks?
Risk Minimization

• given training data: \( \mathcal{T} = \{ \langle x^{(i)}, y^{(i)} \rangle \}_{i=1}^{\mathcal{T}} \)
where each \( y^{(i)} \in \mathcal{L} \) is a label

• assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution \( P(x, y) \)
Risk Minimization

- given training data: \( \mathcal{T} = \{ \langle x^{(i)}, y^{(i)} \rangle \}_{i=1}^{\mathcal{T}} \)

where each \( y^{(i)} \in \mathcal{L} \) is a label

- assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution \( P(x, y) \)

- we want to solve the following:

\[
\hat{w} = \arg \min_w \mathbb{E}_{P(x,y)} [\text{cost}(y, \text{classify}(x, w))]
\]
Risk Minimization

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

\[
\hat{w} = \arg\min_w \mathbb{E}_{P(x,y)} [\text{cost}(y, \text{classify}(x, w))] 
\]

**problem: \( P \) is unknown**
Empirical Risk Minimization
(Vapnik et al.)

• replace expectation with sum over examples:

\[
\hat{w} = \arg \min_w \mathbb{E}_{P(x,y)} [\text{cost}(y, \text{classify}(x, w))]
\]

\[
\hat{w} = \arg \min_w \sum_{i=1}^{|T|} \text{cost}(y^{(i)}, \text{classify}(x^{(i)}, w))
\]
Empirical Risk Minimization
(Vapnik et al.)

• replace expectation with sum over examples:

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

\[
\hat{w} = \text{argmin}_{w} \quad \sum_{i=1}^{\lvert \mathcal{T} \rvert} \text{cost}(y^{(i)}, \text{classify}(x^{(i)}, w))
\]

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

\[ \hat{w} = \arg\min_w \sum_{i=1}^{\mathcal{T}} \text{cost}(y^{(i)}, \text{classify}(x^{(i)}, w)) \]

generalize to permit any loss function

\[ \hat{w} = \arg\min_w \sum_{i=1}^{\mathcal{T}} \text{loss}(x^{(i)}, y^{(i)}, w) \]
solution: replace “cost loss” (also called “0-1” loss) with a surrogate function that is easier to optimize

\[
\hat{w} = \arg\min_w \sum_{i=1}^{T} \text{cost}(y^{(i)}, \text{classify}(x^{(i)}, w))
\]

generalize to permit any loss function

\[
\hat{w} = \arg\min_w \sum_{i=1}^{T} \text{loss}(x^{(i)}, y^{(i)}, w)
\]

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

• modeling
• inference
• learning
  – empirical risk minimization
  – surrogate loss functions
  – gradient-based optimization
Surrogate Loss Functions

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

why is this so difficult to optimize?
Surrogate Loss Functions

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

why is this so difficult to optimize?
not necessarily continuous, can’t use gradient-based optimization
Surrogate Loss Functions

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

max-score loss:

\[ \text{loss}_{\text{maxscore}}(x, y, w) = -\text{score}(x, y, w) \]
Surrogate Loss Functions

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

max-score loss:
\[ \text{loss}_{\text{maxscore}}(x, y, w) = -\text{score}(x, y, w) \]

this is continuous, but what are its drawbacks?
Surrogate Loss Functions

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

max-score loss:
\[
\text{loss}_{\text{maxscore}}(x, y, w) = -\text{score}(x, y, w)
\]

perceptron loss:
\[
\text{loss}_{\text{perc}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} \text{score}(x, y', w)
\]
Surrogate Loss Functions

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

max-score loss:
\[
\text{loss}_{\text{maxscore}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w})
\]

perceptron loss:
\[
\text{loss}_{\text{perc}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \mathbf{w})
\]

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

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

perceptron loss:
\[
\text{loss}_{\text{perc}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} \text{score}(x, y', w)
\]

hinge loss:
\[
\text{loss}_{\text{hinge}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} (\text{score}(x, y', w) + \text{cost}(y, y'))
\]
Surrogate Loss Functions

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

perceptron loss:

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

hinge loss:

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

loss function underlying support vector machines
Surrogate Loss Functions

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

perceptron loss:
\[ \text{loss}_{\text{perc}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} \text{score}(x, y', w) \]

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

hinge loss for our classification setting:
\[ \text{loss}_{\text{hinge}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} (\text{score}(x, y', w) + \delta \mathbb{I}(y \neq y')) \]

tunable hyperparameter
Visualization for a single input $x$

five possible outputs

scores

$y_1, y_2, y_3, y_4, y_5$
Visualisation
for a single input $x$

five possible outputs
Visualization for a single input $x$
Visualization
for a single input $x$

cost

cost($y_5, y_1$)
cost($y_5, y_2$)

$y_1$  $y_2$  $y_3$  $y_4$  $y_5$

gold standard
Visualization
for a single input $x$
\[ \text{loss}_{\text{maxscore}}(x, y, w) = -\text{score}(x, y, w) \]
$$\text{loss}_{\text{maxscore}}(x, y, w) = -\text{score}(x, y, w)$$
\[
\text{loss}_{\text{maxscore}}(x, y, w) = -\text{score}(x, y, w)
\]
\[ \text{loss}_{\text{maxscore}}(y, w) = -\text{score}(x, y, w) \]

**effect of learning:** score of gold standard will go to infinity
perceptron loss:

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

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

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

\[
\text{loss}_{\text{perc}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \mathbf{w})
\]
perceptron loss:

\[
\text{loss}_{\text{perc}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} \text{score}(x, y', w)
\]
hinge loss:

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

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

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

\[
\text{loss}_{\text{hinge}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\mathbf{x}, y', \mathbf{w}) + \text{cost}(y, y'))
\]
hinge loss:

\[
\text{loss}_{\text{hinge}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} (\text{score}(x, y', w) + \text{cost}(y, y'))
\]
Log Loss

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

• 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 $\rightarrow$ Probability

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

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

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

• this is often called a “softmax” function
Log Loss

\[
\text{loss}_{\text{log}}(\mathbf{x}, y, \mathbf{w}) = - \log p_{\mathbf{w}}(y | \mathbf{x})
\]

\[
= - \log \frac{\exp\{\text{score}(\mathbf{x}, y, \mathbf{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\mathbf{x}, y', \mathbf{w})\}}
\]

\[
= - \text{score}(\mathbf{x}, y, \mathbf{w}) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(\mathbf{x}, y', \mathbf{w})\}
\]

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

\[
\text{loss}_{\text{perc}}(\mathbf{x}, y, \mathbf{w}) = -\text{score}(\mathbf{x}, y, \mathbf{w}) + \max_{y' \in \mathcal{L}} \text{score}(\mathbf{x}, y', \mathbf{w})
\]
Log Loss

\[
\text{loss}_\text{log}(x, y, w) = -\log p_w(y \mid x)
\]

\[
= -\log \frac{\exp\{\text{score}(x, y, w)\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(x, y', w)\}}
\]

\[
= -\text{score}(x, y, w) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(x, y', w)\}
\]

- similar to perceptron loss!
- just replace max with \(\text{sopmax}\)

log loss is used in:
- logistic regression classifiers,
- conditional random fields,
- maximum entropy ("maxent") models
Log Loss

\[
\text{loss}_{\text{log}}(x, y, w) = - \log p_w(y \mid x)
\]

\[
= - \log \frac{\exp\{\text{score}(x, y, w)\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(x, y', w)\}}
\]

\[
= - \text{score}(x, y, w) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(x, y', w)\}
\]

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

\[
\text{loss}_{\text{perc}}(x, y, w) = -\text{score}(x, y, w) + \max_{y' \in \mathcal{L}} \text{score}(x, y', w)
\]
Empirical Risk Minimization

\[ \hat{w} = \arg \min_w \mathbb{E}_{P(x, y)} \left[ \text{cost}(y, \text{classify}(x, w)) \right] \]

\[ \hat{w} = \arg \min_w \sum_{i=1}^{\mathcal{T}} \text{loss}(x^{(i)}, y^{(i)}, w) \]
Regularized Empirical Risk Minimization

\[ \hat{w} = \arg\min_w \mathbb{E}_{P(x,y)} \left[ \text{cost}(y, \text{classify}(x, w)) \right] \]

\[ \hat{w} = \arg\min_w \sum_{i=1}^{\mid\mathcal{T}\mid} \text{loss}(x^{(i)}, y^{(i)}, w) + \lambda R(w) \]
Regularized Empirical Risk Minimization

$$\hat{w} = \arg\min_w \mathbb{E}_{P(x,y)} \left[ \text{cost}(y, \text{classify}(x, w)) \right]$$

$$\hat{w} = \arg\min_w \sum_{i=1}^{\mathcal{T}} \text{loss}(x^{(i)}, y^{(i)}, w) + \lambda R(w)$$

regularization strength

regularization term
Regularization Terms

\[ \hat{w} = \arg\min_w \sum_{i=1}^{|T|} \text{loss}(x^{(i)}, y^{(i)}, w) + \lambda R(w) \]

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

• examples:

  \textbf{L}_2 \textit{regularization}:
  (also called Tikhonov regularization or ridge regression)

  \textbf{L}_1 \textit{regularization}:
  (also called basis pursuit or LASSO)
Regularization Terms

$L_2$ regularization: \[ R_{L2}(w) = \|w\|_2^2 = \sum_i w_i^2 \]
differentiable, widely-used

$L_1$ regularization: \[ R_{L1}(w) = \|w\|_1 = \sum_i |w_i| \]
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:

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

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

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

$\eta^{(t)}$ : stepsize at iteration $t$
$\nabla F(\theta^{(t)})$ : 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:

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

$\eta^{(t)}$ : stepsize at iteration $t$
$\nabla F(\theta^{(t)})$ : gradient of objective function

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

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

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

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

$\eta^{(t)}$: stepsize at iteration $t$
$\nabla F(\theta^{(t)})$: gradient of objective function

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

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

• can use line search to choose better stepsizes; also see L-BFGS
Gradient Descent

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

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

- $\eta^{(t)}$: stepsize at iteration $t$
- $\nabla F(\theta^{(t)})$: gradient of the objective function

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

- for efficiency, $F$ is a sum over all training examples!
  every parameter update requires iterating through entire training set
Gradient Descent

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

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

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

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

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

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

every parameter update requires iterating through entire training set

"batch" algorithm
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:

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

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

- 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

\[ f(x) = |x| = \max(x, -x) \]

\[ x < 0: \quad \partial f(x) = \{-1\} \]
\[ x > 0: \quad \partial f(x) = \{1\} \]
\[ x = 0: \quad \partial f(x) = \]
Subgradient Examples

\[ f(x) = |x| = \max(x, -x) \]

- 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

\[
\begin{align*}
x < 0: & \quad \partial f(x) = \{-1\} \\
x > 0: & \quad \partial f(x) = \{1\} \\
x = 0: & \quad \partial f(x) = [-1, 1]
\end{align*}
\]