Stochastic Gradient Descent for Structures

CS 6355: Structured Prediction

Where are we?

- Structural Support Vector Machine
  - How it naturally extends multiclass SVM
- Empirical Risk Minimization
  - Or: how structural SVM and CRF are solving very similar problems
- Training Structural SVM via stochastic gradient descent
  - The algorithm and loss augmented inference
  - Comparison to Perceptron
  - Practical tips

Training structural SVM

- We want to solve the minimization problem:

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max_{\mathbf{y}}\left(\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i)\right)$$

- The function being minimized is convex in $\mathbf{w}$
  - Why?

- Many different algorithms exist
  - Let's look at a gradient based one

Recall: General gradient descent

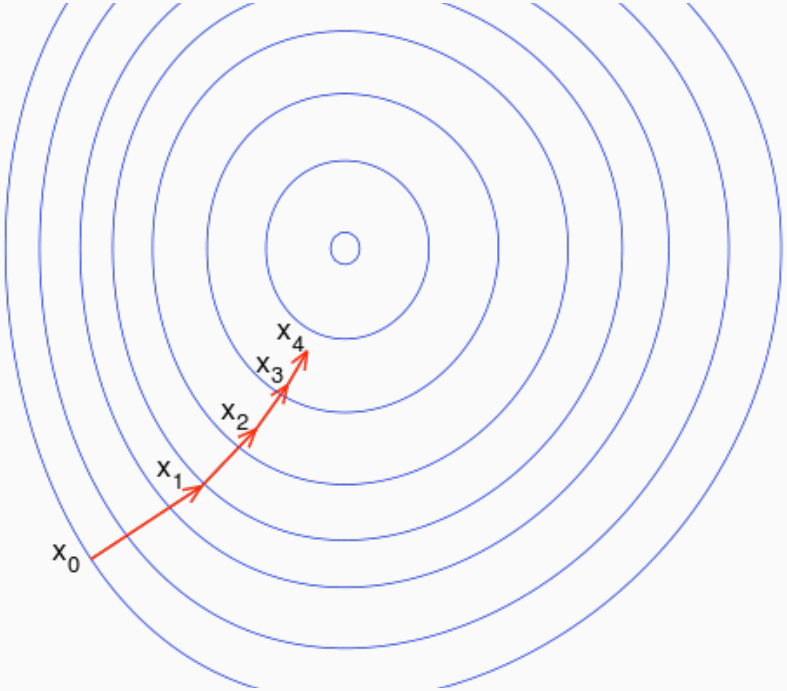
To minimize a differentiable convex function g(x)

- Initialize x_0 to any value
- Iterate until convergence
  - Find gradient of g at x_t: rg(x_t)
  - Update: $x_{t+1} \leftarrow x_t - \gamma_t \nabla g(x_t)$

Recall: General gradient descent

To minimize a differentiable convex function g(x)

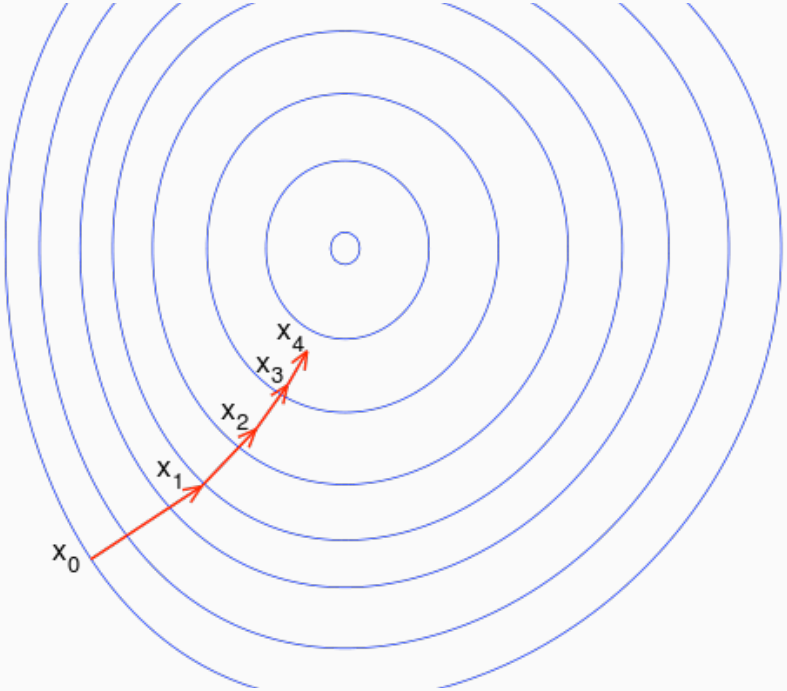
- Initialize x_0 to any value
- Iterate until convergence
  - Find gradient of g at x_t: $rg(x_t)$
  - Update: $x_{t+1} \leftarrow x_t - \gamma_t \nabla g(x_t)$



Recall: General gradient descent

To minimize a differentiable convex function g(x)

- Initialize x_0 to any value
- Iterate until convergence
  - Find gradient of g at x_t: $rg(x_t)$
  - Update: $x_{t+1} \leftarrow x_t - \gamma_t \nabla g(x_t)$



What we know:

$g(x_0) > g(x_1) > g(x_2) \ldots$

Guarantee: Will converge to local minimum (with some assumptions)

Minimizing averages of functions

Suppose the function f we want to minimize is the average of differentiable functions

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x)$$

Minimizing averages of functions

Suppose the function f we want to minimize is the average of differentiable functions

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x)$$

Gradient descent

Initialize x_0

Iterate until convergence:

$$x_{t+1} \leftarrow x_t - \gamma_t \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(x_t)$$

Minimizing averages of functions

Suppose the function f we want to minimize is the average of differentiable functions

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x)$$

Gradient descent

Initialize x_0

Iterate until convergence:

$$x_{t+1} \leftarrow x_t - \gamma_t \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(x_t)$$

Stochastic gradient descent

Initialize x_0

Iterate until convergence:

1. Pick a random g_i and compute its gradient at x_t: $\nabla g_i(x_t)$
2. Update: $x_{t+1} \leftarrow x_t - \gamma_t \nabla g_i(x_t)$

Minimizing averages of functions

Suppose the function f we want to minimize is the average of differentiable functions

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x)$$

Gradient descent

Initialize x_0

Iterate until convergence:

$$x_{t+1} \leftarrow x_t - \gamma_t \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(x_t)$$

Stochastic gradient descent

Initialize x_0

Iterate until convergence:

1. Pick a random g_i and compute its gradient at x_t: $\nabla g_i(x_t)$
2. Update: $x_{t+1} \leftarrow x_t - \gamma_t \nabla g_i(x_t)$

General idea: Replace the gradient with a noisy estimate

Minimizing averages of functions

Suppose the function f we want to minimize is the average of differentiable functions

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x)$$

Gradient descent

Initialize x_0

Iterate until convergence:

$$x_{t+1} \leftarrow x_t - \gamma_t \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(x_t)$$

Stochastic gradient descent

Initialize x_0

Iterate until convergence:

1. Pick a random g_i and compute its gradient at x_t: $\nabla g_i(x_t)$
2. Update: $x_{t+1} \leftarrow x_t - \gamma_t \nabla g_i(x_t)$

General idea: Replace the gradient with a noisy estimate

Stochastic gradient descent

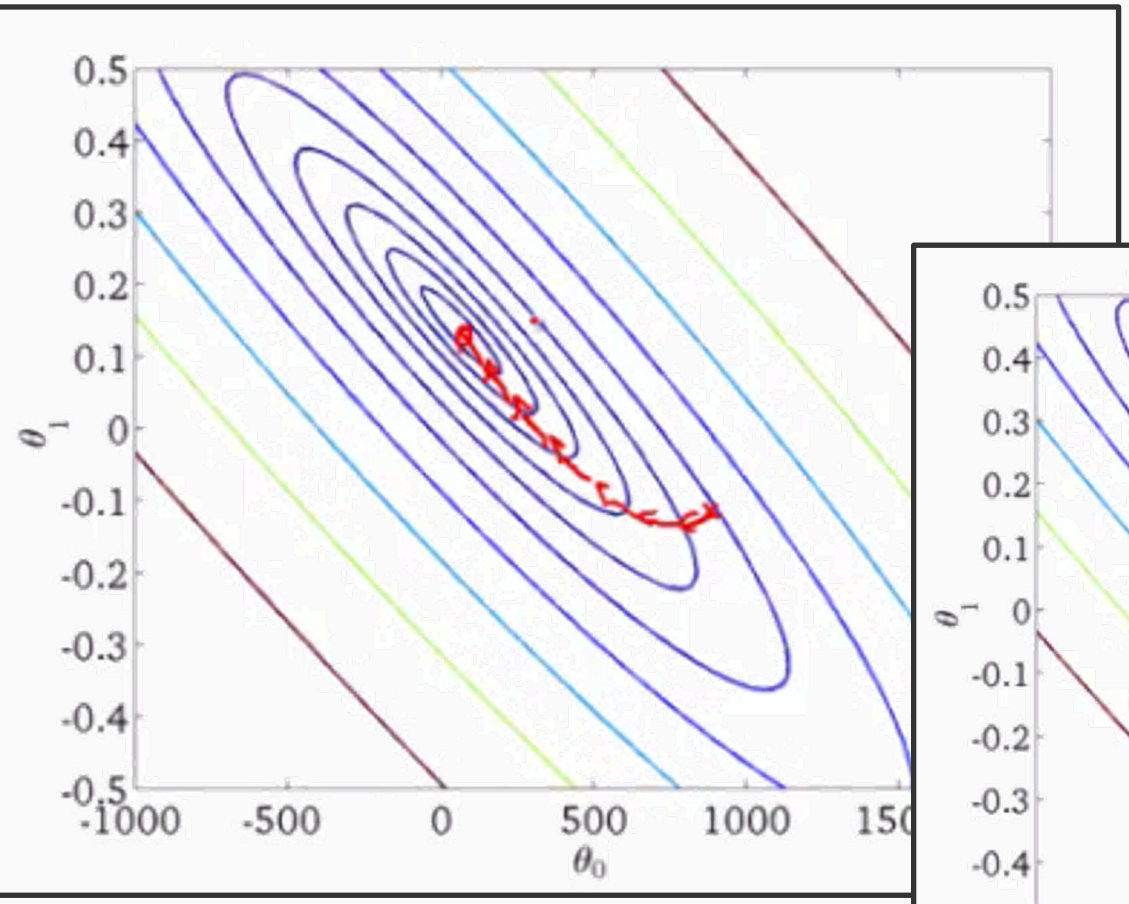
Suppose the function f we want to minimize is the average of differentiable functions

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x)$$

- Initialize x_0
- Iterate until convergence
  - Pick a random g_i and compute its gradient at x_t: $\nabla g_i(x_t)$
  - Update: $x_{t+1} \leftarrow x_t - \gamma_t \nabla g_i(x_t)$

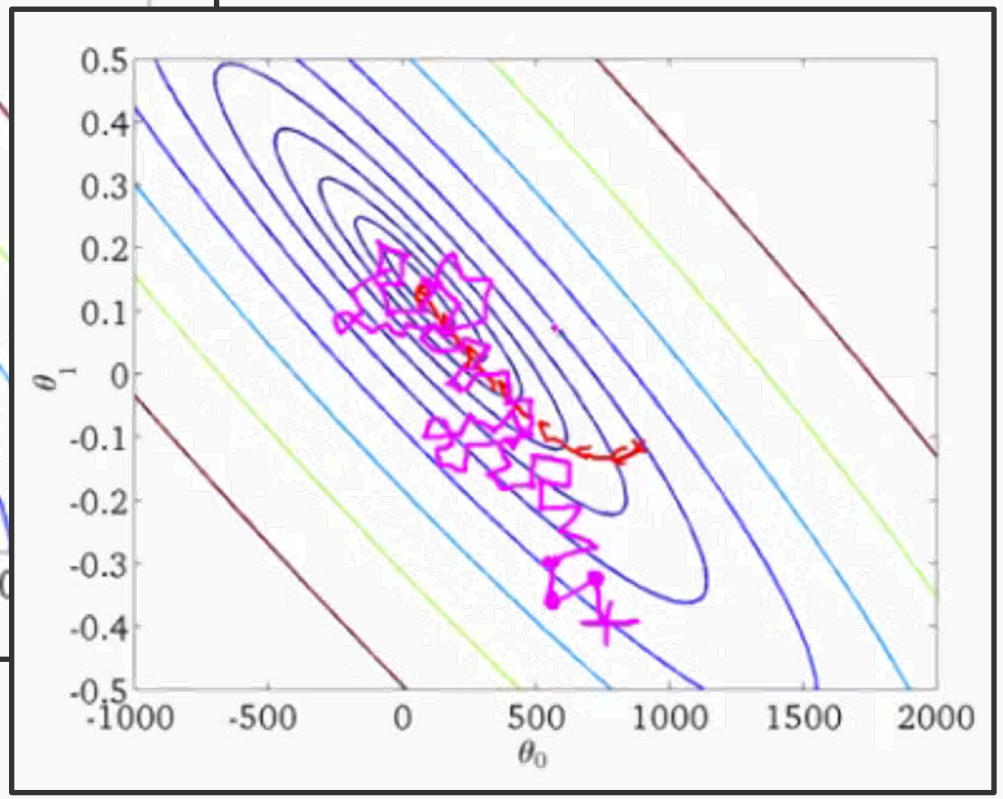
General idea: Replace the gradient with a noisy estimate

Gradient descent vs SGD



Gradient descent

Stochastic Gradient descent



[Images from Ng]

Stochastic gradient descent

We want to solve the following optimization problem:

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max_{\mathbf{y}}\left(\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i)\right)$$

Stochastic gradient descent

We want to solve the following optimization problem:

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max_{\mathbf{y}}\left(\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i)\right)$$

General SGD strategy:

Repeat till convergence:

- Pretend our training set has only one example (say $\mathbf{x}_i$, $\mathbf{y}_i$)

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\max_{\mathbf{y}}\left(\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i)\right)$$

- Compute gradient of this objective and update $\mathbf{w}$

Subgradients

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \max_{\mathbf{y}} \left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

Not differentiable! What is the gradient?

Answer: Use ***sub***gradients

Definition: A vector **g** is a subgradient of a function f (not necessarily convex) at **x** if

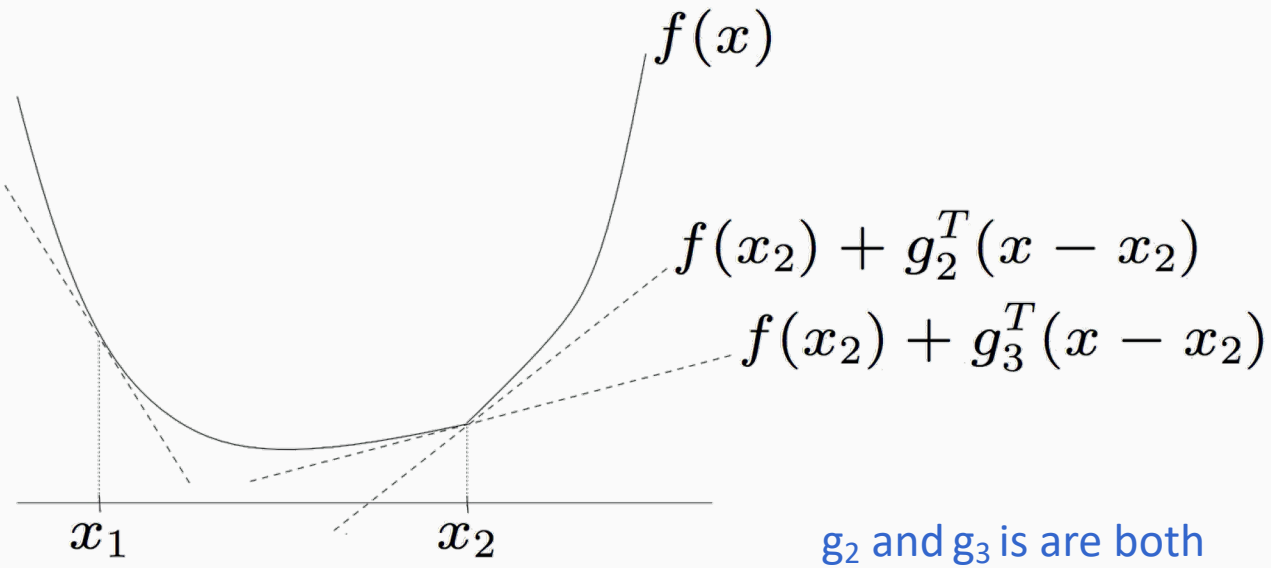
$$f(y) \geq f(x) + g^T(y - x) \quad \text{for all } y$$

Subgradients

$$f(y) \geq f(x) + g^T(y - x) \quad \text{for all } y$$

f is differentiable at x_1
Tangent at this point

$f(x_1) + g_1^T(x - x_1)$



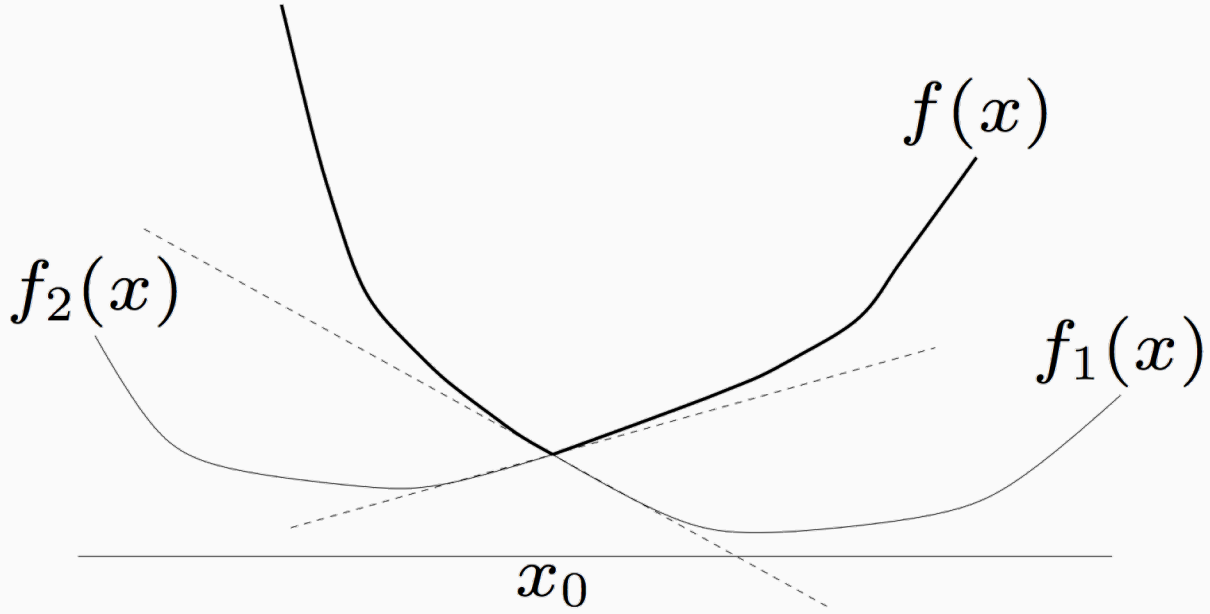
g_1 is a gradient at x_1

g_2 and g_3 is are both subgradients at x_2

[Example from Boyd]

Subgradients of max

Consider the function $f(x) = \max\{f_1(x), f_2(x)\}$



- $f_1(x) > f_2(x)$, unique subgradient r $f_1(x)$
- $f_2(x) > f_1(x)$, unique subgradient r $f_2(x)$
- $f_1(x) = f_2(x)$, subgradients in [r $f_1(x)$, r $f_2(x)$]

Strategy: Solve the max first, then compute gradient of whichever function is argmax

Stochastic subgradient descent

We want to solve the following optimization problem:

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max_{\mathbf{y}}\left(\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i)\right)$$

General SGD strategy:

Repeat till convergence:

- Pretend our training set has only one example (say $\mathbf{x}_i$, $\mathbf{y}_i$)

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\max_{\mathbf{y}}\left(\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) + \Delta(\mathbf{y},\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i)\right)$$

- Compute *sub*gradient of this objective and update $\mathbf{w}$

Subgradient of the structured hinge loss

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \max_{\mathbf{y}} \left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

Subgradient of the structured hinge loss

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \max_{\mathbf{y}} \left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

1. Solve the max. Suppose solution is $\mathbf{y}'$
   - The loss-augmented/loss-sensitive/cost-augmented inference step

Subgradient of the structured hinge loss

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \max_{\mathbf{y}} \left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

1. Solve the max. Suppose solution is $\mathbf{y}'$
   - The loss-augmented/loss-sensitive/cost-augmented inference step

2. Compute gradient of

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}') + \Delta(\mathbf{y}', \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

Subgradient of the structured hinge loss

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \max_{\mathbf{y}} \left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

1. Solve the max. Suppose solution is $\mathbf{y}'$
   - The loss-augmented/loss-sensitive/cost-augmented inference step

2. Compute gradient of

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}') + \Delta(\mathbf{y}', \mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

3. The subgradient is

$$\mathbf{w} + C\left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$$

SGD for structural SVM: The update

- Gradient: $\mathbf{w} + C\left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$

- At each step, go down the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \cdot gradient$$

SGD for structural SVM: The update

- Gradient: $\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)$

- At each step, go down the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \cdot \Big(\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)\Big)$$

SGD for structural SVM: The update

- Gradient: $\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)$
- At each step, go down the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \cdot \Big(\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)\Big)$$

- Or equivalently
  - If $\mathbf{y}' = \mathbf{y}_i$: $\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w}$

SGD for structural SVM: The update

- Gradient: $\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)$
- At each step, go down the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \cdot \Big(\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)\Big)$$

- Or equivalently
  - If $\mathbf{y}' = \mathbf{y}_i$: $\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w}$
  - Otherwise:

$$\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w} - C\gamma_t\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)$$

SGD for structural SVM: The update

- Gradient: $\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)$
- At each step, go down the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \cdot \Big(\mathbf{w} + C\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)\Big)$$

- Or equivalently
  - If $\mathbf{y}' = \mathbf{y}_i$: $\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w}$

Even of loss-augmented inference gives the true answer, shrink $\mathbf{w}$. (Increase margin!)

  - Otherwise:

$$\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w} - C\gamma_t\big(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\big)$$

Training structural SVM with SGD

Given a training set $\mathrm{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:

2. Return $\mathbf{w}$

Training structural SVM with SGD

Given a training set $\mathrm{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data

2. Return $\mathbf{w}$

Training structural SVM with SGD

Given a training set $\mathrm{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:

2. Return $\mathbf{w}$

Training structural SVM with SGD

Given a training set $\mathrm{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        1. Let $\mathbf{y}' = \max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}_i) \right)$

(loss-augmented) Inference step

2. Return $\mathbf{w}$

Training structural SVM with SGD

Given a training set $\mathrm{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        1. Let $\mathbf{y}' = \max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$
        2. If $\mathbf{y}' = \mathbf{y}_i$
            Shrink $\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w}$
           Else:
            Update $\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w} - C\gamma_t\left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$
2. Return $\mathbf{w}$

Model update

Loss augmented inference

$$\max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}_i) \right)$$

- Recall: $\Delta(\mathbf{y}, \mathbf{y}_i)$ is the Hamming distance between $\mathbf{y}$ and $\mathbf{y}_i$
- Last term in the inference is constant. So effectively

$$\max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) \right)$$

Loss augmented inference

$$\max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}_i) \right)$$

- Recall: $\Delta(\mathbf{y}, \mathbf{y}_i)$ is the Hamming distance between $\mathbf{y}$ and $\mathbf{y}_i$
- Last term in the inference is constant. So effectively

$$\max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) \right)$$

- How difficult is this?
  - Computationally, identical complexity to inference
  - You need to implement inference anyway
    - Minor tweak will give you loss-augmented inference

Loss augmented inference

$$\max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}_i) \right)$$

- Recall: $\Delta(\mathbf{y}, \mathbf{y}_i)$ is the Hamming distance between $\mathbf{y}$ and $\mathbf{y}_i$
- Last term in the inference is constant. So effectively

$$\max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) \right)$$

- How difficult is this?
  - Computationally, identical complexity to inference
  - You need to implement inference anyway
    - Minor tweak will give you loss-augmented inference
- Exercise: Why does it help?

Where are we?

- Structural Support Vector Machine
  - How it naturally extends multiclass SVM
- Empirical Risk Minimization
  - Or: how structural SVM and CRF are solving very similar problems
- Training Structural SVM via stochastic gradient descent
  - ✓ The algorithm and loss augmented inference
  - Comparison to Perceptron
  - Practical tips

Recall: Structured Perceptron algorithm

Given a training set D = $\{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        1. Let $\mathbf{y}' = \max_{\mathbf{y}} \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y})$
        2. If $\mathbf{y}' \neq \mathbf{y}_i$

            Update $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$
2. Return $\mathbf{w}$

Recall: Structured Perceptron algorithm

Given a training set D = $\{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        1. Let $\mathbf{y}' = \max_{\mathbf{y}} \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y})$ Inference within the training loop
        2. If $\mathbf{y}' \neq \mathbf{y}_i$

            Update $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t\left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$
2. Return $\mathbf{w}$

Recall: Structured Perceptron algorithm

Given a training set D = $\{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        1. Let $\mathbf{y}' = \max_{\mathbf{y}} \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y})$
        2. If $\mathbf{y}' \neq \mathbf{y}_i$
            Update $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t\left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i)\right)$
2. Return $\mathbf{w}$

Update only on an error.
Structured Perceptron is an mistake-driven algorithm.
If there is a mistake, promote **y** and demote **y'**

SGD for Structural SVM

Given a training set D = $\{(\mathbf{x}_i, \mathbf{y}_i)\}$

Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$

1. For epoch = 1 … T:
    1. Shuffle data
    2. For each training example $(\mathbf{x}_i, \mathbf{y}_i) \in D$:
        1. Let $\mathbf{y}' = \max_{\mathbf{y}} \left(\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}_i) - \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}_i) \right)$
        2. If $\mathbf{y}' = \mathbf{y}_i$

            Shrink $\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w}$

            Else:

            Update $\mathbf{w} \leftarrow (1 - \gamma_t)\mathbf{w} - C\gamma_t \left(\phi(\mathbf{x}_i, \mathbf{y}') - \phi(\mathbf{x}_i, \mathbf{y}_i) \right)$
2. Return $\mathbf{w}$

Update is a lot like the Perceptron update. Two differences:

1. Loss augmented inference instead of standard inference
2. Shrink $\mathbf{w}$, even if it the inference is correct

SGD for structural SVM and Perceptron

SGD update is a lot like the Perceptron update

Two differences:

1. Loss augmented inference

    If Δ is defined to be uniformly zero, this disappears

2. Shrink w, even if it the inference is correct

    If no regularization (or C is very large), this disappears

Where are we?

- Structural Support Vector Machine
  - How it naturally extends multiclass SVM
- Empirical Risk Minimization
  - Or: how structural SVM and CRF are solving very similar problems
- Training Structural SVM via stochastic gradient descent
  - ✓ The algorithm and loss augmented inference
  - ✓ Comparison to Perceptron
  - Practical tips

1. Convergence and learning rates

With enough iterations, it will converge in expectation

Provided the step sizes are square summable, but not summable

- Step sizes γ_t are positive
- Sum of squares of step sizes over t = 1 to 1 is not infinite
- Sum of step sizes over t = 1 to 1 is infinity

- Examples: $\gamma_t = \frac{\gamma_0}{1+t}$ or $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0 t}{c}}$

1. Convergence and learning rates

- Number of iterations to get to accuracy within ϵ
- For strongly convex functions, N examples, d dimensional features:
  - Gradient descent: $O\left(Nd \ln \frac{1}{\epsilon}\right)$
  - Stochastic gradient descent: $O\left(\frac{d}{\epsilon}\right)$
- More subtleties involved, but SGD is generally preferable when the data size is huge

2. Extensions of the simple SGD update

- Several extensions to the vanilla SGD algorithm

  See the survey by Ruder for a long discussion. Available online at https://ruder.io/optimizing-gradient-descent

- Intuitions for the extensions:
  - How do we choose learning rates? How should we decay them?
  - If a certain dimension has very few updates, while another has many more updates, should their learning rates be the same?
  - Can we keep a history of gradients to get more stable updates?

2a. The AdaGrad update

Intuition:

- Frequently updated features should get smaller learning rates
- Pay more attention to infrequent, possibly informative features
- Each weight (indexed by i) gets a separate learning rate $\gamma_{t,i}$ at step t

- Suppose the gradient is $\mathbf{g}$
  - The shrinking of $\mathbf{w}$ is separate
  - For structured SVM, $\mathbf{g} = \phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \boldsymbol{y}')$

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. JMLR, 2011.

2a. The AdaGrad update

Intuition:

- Frequently updated features should get smaller learning rates
- Pay more attention to infrequent, possibly informative features
- Each weight (indexed by i) gets a separate learning rate $\gamma_{t,i}$ at step t

- Suppose the gradient is $\mathbf{g}$
  - The shrinking of **w** is separate
  - For structured SVM, $\mathbf{g} = \phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \boldsymbol{y}')$

- Set learning rate for weight i as

$$\gamma_{t,i} = \frac{\gamma_0}{\sqrt{\sum_{j=0}^{t} \mathbf{g}_{j,i}^2}}$$

Typically computed as a vector operation.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. JMLR, 2011.

2a. Implementing the AdaGrad update

- In practice, the denominator is accumulated

- Initialize $v_{0,i} = 0$ for all weights (indexed by i)
- At each step:
  - Compute the gradient as always. Call the gradient vector $\mathbf{g}$
  - For every dimension (indexed by i), update the denominator for the learning rate:

$$v_{t,i} \leftarrow v_{t-1,i} + \boldsymbol{g}_i^2$$

  - The learning rate for dimension i is

$$\gamma_{t,i} \leftarrow \frac{\gamma_0}{\sqrt{v_{t,i}}}$$

Easy to implement, especially with vector operators
Has theoretical guarantees of being faster than SGD (See paper for details)

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. JMLR, 2011.

2b. RMSProp

The AdaGrad decay can be too aggressive: the denominator may grow too fast.

Uses a hyperparameter β that is between 0 and 1

- Initialize $v_{0,i} = 0$ for all weights (indexed by i)
- At each step:
  - Compute the gradient as always. Call the gradient vector $\mathbf{g}$
  - Decay the denominator instead of just accumulating to it (i.e, compute an exponential moving average):

$$v_{t,i} = \beta v_{t-1,i} + (1-\beta)\mathbf{g}_{t,i}^2$$

  - Then compute the learning rate for each weight as before

$$\gamma_{t,i} \leftarrow \frac{\gamma_0}{\sqrt{v_{t,i}}}$$

Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, Coursera: Neural Networks for Machine Learning.

2c. Adam (Adaptive Moment Estimation)

- Can be seen as an extension of RMSProp (and also AdaGrad)
- General idea:
  - Keep track of the history of *gradients* as an exponential moving average
  - Keep an exponential moving average of the squares of individual gradients as well (as in RMSProp)
  - Perform an update using the average gradient (not just the currently computed one), and use the feature-specific learning rates as with RMSProp
    - Has a bias correction step prior to this (see paper for details)
- Commonly used in practice

Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

3. Mini-batches for SGD

- What we saw: Approximate the gradient with a ***single*** example
- Instead, select small set (e.g. 10 or 100 or more) of examples (called mini-batch) at each step and use those to approximate the gradient
- Compute subgradient of this mini-batch and update **w**
- Converges faster, more stable
- Exercise: Work out the details of the mini-batch update

4. Miscellaneous tips

- Shuffle the dataset at the beginning of each epoch
- Monitor training cost and validation error
  - Training error should "generally" decrease
  - Stop training when validation error doesn't change for some time
- Very important: Check your gradient computation
  - Even small errors will make SGD erratic
  - Finite differences:
    - See Leon Bottou's "Stochastic Gradient Descent Tricks"
- Experiment with learning rates using a small dataset

Summary: Optimizing the SVM objective

- We have seen a solver for structural SVMs
  - Stochastic sub-gradient descent and some variants
  - Easy to implement
- Other approaches exist
  - Dual co-ordinate ascent/descent, cutting planes, etc.
- To use in your problem (for all solvers):
  - Need to define the model (independence assumptions between predictions), the features, inference *and* loss-augmented inference
- Important: SGD is a general strategy, not just for SVMs
  - Use for other objective functions, neural networks, etc.

Big picture

- Structural Support Vector Machine
  - How it naturally extends multiclass SVM
- Empirical Risk Minimization
  - Or: how structural SVM and CRF are solving very similar problems
- Training via stochastic gradient descent
  - Broadly applicable
  - For structural SVM, the computationally difficult step is computing the gradient (needs loss-augmented inference). The rest is standard SGD.