

Probability and Structure in Natural Language Processing

Noah Smith

Heidelberg University, November 2014

Learning

- Assume a collection of N pairs (\mathbf{x}, \mathbf{y}) ; supervised learning with complete data.

Loss

- Let h be a hypothesis (an instantiated, predictive model).
- $\text{loss}(\mathbf{x}, \mathbf{y}; h)$ = a measure of how badly h performs on input \mathbf{x} if \mathbf{y} is the correct output.
- How to decide what “loss” should be?
 1. computational expense
 2. knowledge of actual costs of errors
 3. formal foundations enabling theoretical guarantees

Risk

- There is some true distribution p^* over input, output pairs (\mathbf{X}, \mathbf{Y}) .
- Under that distribution, what do we expect h 's loss to be?

$$\mathbb{E}_{p^*(\mathbf{X}, \mathbf{Y})}[\text{loss}(\mathbf{X}, \mathbf{Y}; h)]$$

- We don't have p^* , but we have the empirical distribution, giving **empirical risk**:

$$\mathbb{E}_{\tilde{p}(\mathbf{X}, \mathbf{Y})}[\text{loss}(\mathbf{X}, \mathbf{Y}; h)] = \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h)$$

Empirical Risk Minimization

- Provides a criterion to decide on h :

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h)$$

- Background preferences over h can be included in **regularized** empirical risk minimization:

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h) + R(h)$$

Parametric Assumptions

- Typically we do not move in “h-space,” but rather in the space of continuously-parameterized predictors.

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h) + R(h)$$
$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

Three Kinds of Loss Functions

- Error
 - Could be zero-one, or task-specific.
 - Mean squared error makes sense for *continuous* predictions and is used in *regression*.
- Log loss
 - Probabilistic interpretation (“likelihood”)
- Hinge loss
 - Geometric interpretation (“margin”)

Log Loss (First Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$$

- Maximum likelihood estimation:
 $R(\mathbf{w})$ is 0 for models in the family, $+\infty$ for other models.
- Maximum *a posteriori* (MAP) estimation:
 $R(\mathbf{w})$ is $-\log p(\mathbf{w})$
- Often called **generative** modeling.

Log Loss (First Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$$

Examples:

- N-gram language models
- Supervised HMM taggers
- Charniak, Collins, and Stanford parsers

Log Loss (First Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$$

Computationally ...

- Convex and differentiable.
- Closed form for directed, multinomial-based models $p_{\mathbf{w}}$.
 - Count and normalize!
- In other cases, requires posterior inference, which can be expensive depending on the model's structure.
- Linear decoding (for some parameterizations).

Log Loss (First Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$$

Error ...

- No notion of error.
- Learner wins by moving as much probability mass as possible to training examples.

Log Loss (First Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$$

Guarantees...

- Consistency: if the true model is in the right family, enough data will lead you to it.

Log Loss (First Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$$

Different parameterizations ...

- Multinomials (BN-like): $-\sum_e \text{freq}(e; \mathbf{x}, \mathbf{y}) \underbrace{\log p_e}_{w_e}$
- Global log-linear (MN-like): $-\mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}) + \log \sum_{\mathbf{x}', \mathbf{y}'} \exp \mathbf{w}^\top \mathbf{g}(\mathbf{x}', \mathbf{y}')$
- Locally normalized log-linear: $-\sum_e \text{freq}(e; \mathbf{x}, \mathbf{y}) \left(\mathbf{w}^\top \mathbf{g}(e) - \log \sum_{e' \in \mathcal{C}(e)} \exp \mathbf{w}^\top \mathbf{g}(e') \right)$

Reflections on Generative Models

- Most early solutions are generative.
- Most unsupervised approaches are generative.
- Some people only believe in generative models.
- Sometimes estimators are not as easy as they seem (“deficiency”).
- Advice: start here if there’s a sensible generative story.
 - You can always use a “better” loss function with the same linear model later on.

Zero-One Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = \mathbf{1}\{h_{\mathbf{w}}(\mathbf{x}) \neq \mathbf{y}\}$$

Zero-One Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = \mathbf{1}\{h_{\mathbf{w}}(\mathbf{x}) \neq \mathbf{y}\}$$

Computationally:

- Piecewise constant. ☹️

Error: 😊

Guarantees: none

Error as Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = \text{error}(h_{\mathbf{w}}(\mathbf{x}); \mathbf{y})$$

Generalizes zero-one, same difficulties.

Example: Bleu-score maximization in machine translation, with “MERT” line search.

Comparison

	Generative (Log Loss)	Error as Loss
Computation	Convex optimization.	Optimizing a piecewise constant function.
Error-awareness	None	😊
Guarantees	Consistency.	None.

Discriminative Learning

- Various loss functions between log loss and error.
- Three commonly used in NLP:
 - Conditional log loss (“max ent,” CRFs)
 - Hinge loss (structural SVMs)
 - Perceptron’s loss
- We’ll discuss each, compare, and unify.

Log Loss (Second Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$$

- Can be understood as a generative model over **Y**, but does not model **X**.
 - “Conditional” model

Log Loss (Second Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$$

Examples:

- Logistic regression (for classification)
- MEMMs
- CRFs

Log Loss (Second Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$$

Computationally ...

- Convex and differentiable.
- Requires posterior inference, which can be expensive depending on the model's structure.
- Linear decoding (for some parameterizations).

Log Loss (Second Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$$

Error ...

- No notion of error.
- Learner wins by moving as much probability mass as possible to training examples' correct outputs.

Log Loss (Second Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{y} | \mathbf{x})$$

Guarantees...

- Consistency: if the true conditional model is in the right family, enough data will lead you to it.

Log Loss (Second Version)

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\log p_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$$

Different parameterizations ...

- Global log-linear (CRF):

$$-\mathbf{w}^{\top} \mathbf{g}(\mathbf{x}, \mathbf{y}) + \log \sum_{\mathbf{y}'} \exp \mathbf{w}^{\top} \mathbf{g}(\mathbf{x}', \mathbf{y}')$$

- Locally normalized log-linear (MEMM):

$$-\sum_e \text{freq}(\mathbf{e}; \mathbf{x}, \mathbf{y}) \left(\mathbf{w}^{\top} \mathbf{g}(\mathbf{e}) - \log \sum_{\mathbf{e}' \in \mathcal{C}(\mathbf{e})} \exp \mathbf{w}^{\top} \mathbf{g}(\mathbf{e}') \right)$$

Comparing the Two Log Losses

	$-\log p_w(\mathbf{x}, \mathbf{y})$	$-\log p_w(\mathbf{y} \mathbf{x})$
Parameterization	Usually multinomials (BN-like).	Almost always log-linear (MN-like).
Under the usual parameterization ...		
Computation	Count and normalize.	Convex optimization.
Error-awareness	None.	Aware of the \mathbf{Y} -prediction task, (approximates zero-one).
Guarantees	Consistency of joint.	Consistency of cond.

Hinge Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}) + \max_{\mathbf{y}'} \mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}') + \text{error}(\mathbf{y}', \mathbf{y})$$

- Penalizes the model for letting competitors get close (in terms of score) to the correct answer \mathbf{y} .
 - Can penalize them in proportion to their error.

Hinge Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}) + \max_{\mathbf{y}'} \mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}') + \text{error}(\mathbf{y}', \mathbf{y})$$

Examples ...

- Perceptron (including Collins' structured version)
 - Classic version ignores *error* term
- SVM and some structured variants:
 - Max-margin Markov networks (Taskar et al.)
 - MIRA (1-best, k-best)

Hinge Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}) + \max_{\mathbf{y}'} \mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}') + \text{error}(\mathbf{y}', \mathbf{y})$$

Computationally ...

- Convex, not everywhere differentiable.
 - Many specialized techniques now available.
- Requires MAP or “cost-augmented” MAP inference.
- Linear decoding.

Hinge Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}) + \max_{\mathbf{y}'} \mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}') + \text{error}(\mathbf{y}', \mathbf{y})$$

Error ...

- Built in.
- Most convenient when error function factors similarly to features \mathbf{g} .

Hinge Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N \text{loss}(\mathbf{x}_i, \mathbf{y}_i; h_{\mathbf{w}}) + R(\mathbf{w})$$

$$\text{loss}(\mathbf{x}, \mathbf{y}; h_{\mathbf{w}}) = -\mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}) + \max_{\mathbf{y}'} \mathbf{w}^\top \mathbf{g}(\mathbf{x}, \mathbf{y}') + \text{error}(\mathbf{y}', \mathbf{y})$$

Guarantees ...

- Generalization bounds.
 - Not clear how seriously to take these in NLP; may not be tight enough to be meaningful.
- Often you will find *convergence* guarantees (and rates) for optimization techniques.

They Are All Related

$$\frac{1}{\beta} \log \sum_{\mathbf{y}'} \exp [\beta (\mathbf{w}^\top (\mathbf{g}(\mathbf{x}, \mathbf{y}') - \mathbf{g}(\mathbf{x}, \mathbf{y})) + \gamma error(\mathbf{y}', \mathbf{y}))]$$

	β	γ
Conditional log loss (Lafferty et al., 2001)	1	0
Perceptron's hinge loss (Collins, 2002)	∞	0
Structural SVM's hinge loss (Taskar et al., 2004)	∞	> 0
Softmax-margin (Gimpel and Smith, 2010)	1	1

CRFs, Max Margin, or Perceptron?

- For supervised problems, we do not expect large differences.
- Perceptron is easiest to implement.
 - With cost-augmented inference, it should get better and begins to approach MIRA and M^3Ns .
- CRFs are best for probability fetishists.
 - Probably most appropriate if you are extending with latent variables; the jury is out.
- Not yet “plug and play.”

$R(\mathbf{w})$

- Regularization term – avoid overfitting
 - Usually means “avoid large magnitudes in \mathbf{w} ”
- (Log) Prior – respect background beliefs about the predictor $h_{\mathbf{w}}$

$R(\mathbf{w})$

- Usual starting point: squared L_2 norm
 - Computationally convenient (it's strongly convex, it is its own Fenchel conjugate, ...)
 - Probabilistic view: Gaussian prior on weights (Chen and Rosenfeld, 2000)
 - Geometric view: Euclidean distance (original regularization method in SVMs)
 - Only one hyperparameter

$$R(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 = \lambda \sum_j w_j^2$$

$R(\mathbf{w})$

- Another option: L_1 -norm
 - Computationally less convenient (not everywhere differentiable)
 - Probabilistic view: Laplacian prior on weights (originally proposed as “lasso” in regression)
 - Sparsity inducing (“free” feature selection)

$$R(\mathbf{w}) = \lambda \|\mathbf{w}\|_1 = \lambda \sum_j |w_j|$$

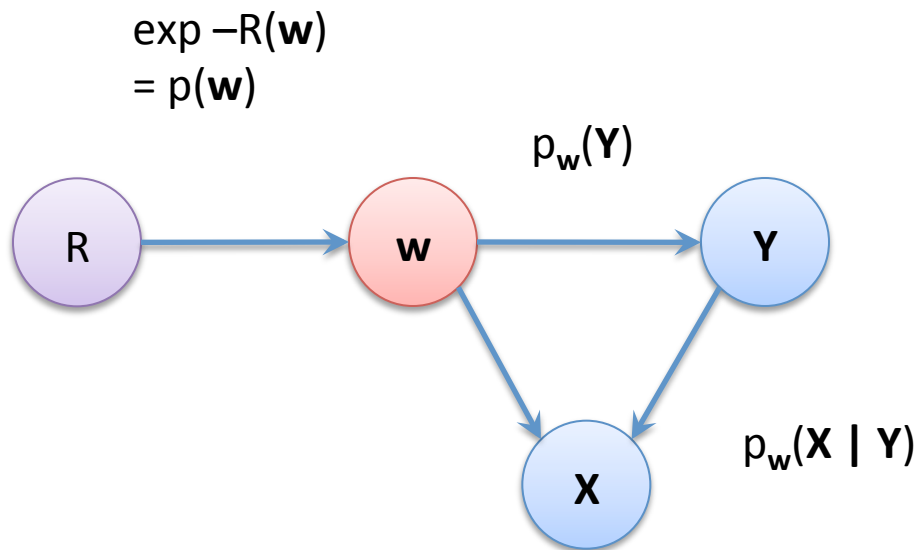
$R(\mathbf{w})$

- Lots of attention to this in machine learning.
- “Structured sparsity”
 - Want groups of features to go to zero, or group-internal sparsity, or ...
- Interpolation between L_1 and L_2 – “elastic net”
 - Sparsity but maybe better behaved
- This is not yet “plug and play.”
 - Optimization algorithm is heavily affected.

MAP Learning is Inference

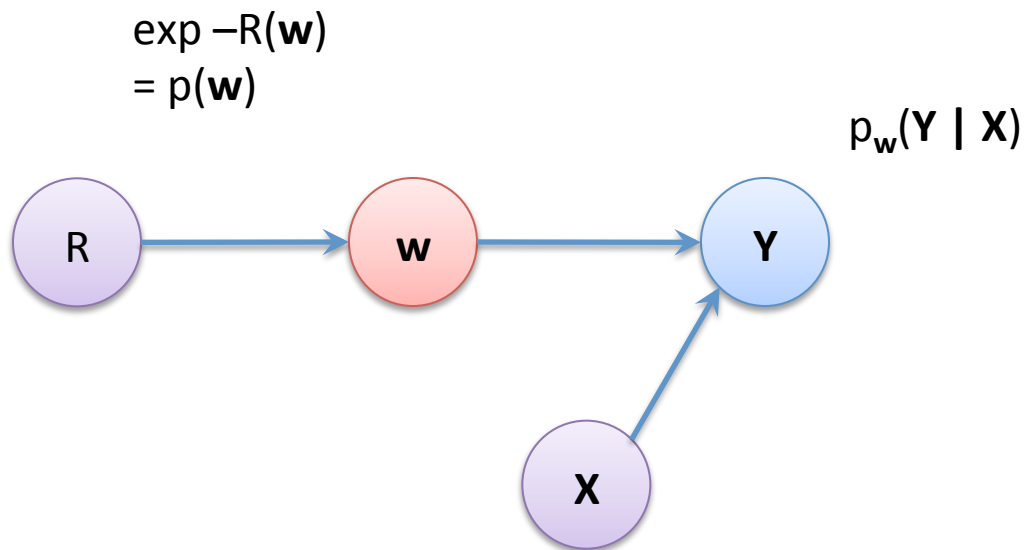
- Seeking “most probable explanation” of the data, in terms of \mathbf{w} .
 - Explain the data: $p(\mathbf{x}, \mathbf{y} \mid \mathbf{w})$
 - Not too surprising: $p(\mathbf{w})$
- If we think of “ \mathbf{W} ” as another random variable, MAP learning is MAP inference.
 - Looks very different from decoding!
 - But at a high level of abstraction, it is the same.

MAP Learning as a Graphical Model



- This is a view of learning a “noisy channel” model.

MAP Learning as a Graphical Model



- This is a view of learning in a CRF.

MAP Estimation for CRFs

$\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$, which is MAP inference

iterate to obtain gradient:

sufficient statistics from $p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$, obtained by
posterior inference

How To Think About Optimization

- Depending on your choice of loss and R , different approaches become available.
 - Learning algorithms can interact with inference/decoding algorithms, too.
- In NLP today, it is probably more important to focus on the features, error function, and prior knowledge.
 - Decide what you want, and then use the best available optimization technique.

Key Techniques

- Quasi-Newton – **batch** method for differentiable loss functions
 - LBFGS, OWLQN when using L_1 regularization
- Stochastic subgradient ascent – **online**
 - Generalizes perceptron, MIRA, stochastic gradient ascent
 - Sometimes sensitive to step size
 - Can often use “mini-batches” to speed up convergence
 - Recent trick: Adagrad, which adapts the step size
- For error minimization: randomization

Pitfalls

- Engineering learning procedures is tempting and *may* help you get better performance.
 - Without at least some analysis in terms of loss, error, and regularization, it's unlikely to be useful outside your problem/dataset.
- When randomization is involved, look at variance across runs (Clark et al., 2011)
- Always tune hyperparameters (e.g., regularization strength) on *development* data!

Major Topics in Current Work

- Coping with approximate inference
- Exploiting incomplete data
 - Semisupervised learning
 - Creating features from raw text
 - Latent variable models (discussed tomorrow)
- Feature management
 - Structured sparsity (R)