Machine Learning

Lecture 5 - Learning from large-scale data

Mattias Villani

Department of Statistics
Stockholm University

Department of Computer and Information Science
Linköping University

mattiasvillani.com   @matvil   mattiasvillani

Lecture overview

- More on loss functions
- Optimization algorithms for large data

Loss minimization as a proxy for generalization

- Parametric model $y|\mathbf{x} \sim f_{\boldsymbol{\theta}}(y|\mathbf{x})$.
- Learn the parameters by minimizing a cost function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \underbrace{J(\boldsymbol{\theta}) \equiv \frac{1}{n}\sum_{i=1}^{n} \overbrace{L\left(\hat{y}(\mathbf{x}_i;\boldsymbol{\theta}), y_i\right)}^{\text{loss function}}}_{\text{cost function} J(\boldsymbol{\theta})}$$

- ML: **cost function** $J(\boldsymbol{\theta})$ can be considered as a **proxy for** the real objective of interest, the **expected new data error**:

$$E_{\text{new}} \equiv \mathbb{E}_{\star}\left[E\left(\hat{y}(\mathbf{x}_{\star};\mathcal{T}), y_{\star}\right)\right]$$

- **Early stopping** of iterative optimization:
  - when numerical accuracy is on par with statistical accuracy
  - to avoid overfitting (implicit regularization)

Loss functions for regression

- **Squared error loss**

$$L(y,\hat{y}) = (\hat{y}-y)^2$$

- **Absolute error loss**

$$L(y,\hat{y}) = |\hat{y}-y|$$

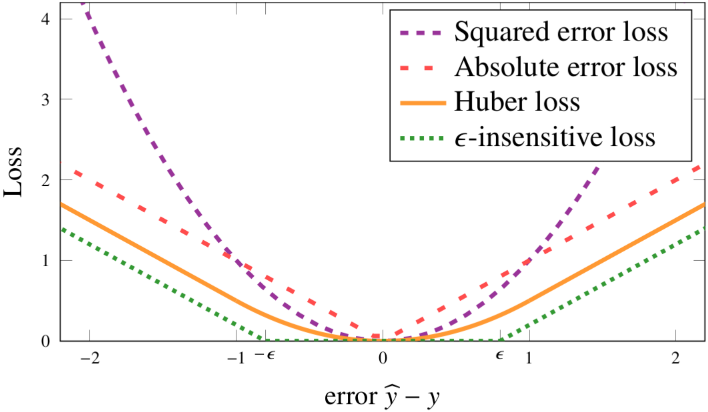
- **Huber loss**

$$L(y,\hat{y}) = \begin{cases} \frac{1}{2}(\hat{y}-y)^2 & \text{if } |\hat{y}-y|<1 \\ |\hat{y}-y|-\frac{1}{2} & \text{otherwise} \end{cases}$$

- **ϵ-insensitive loss**

$$L(y,\hat{y}) = \begin{cases} 0 & \text{if } |\hat{y}-y|<\epsilon \\ |\hat{y}-y|-\epsilon & \text{otherwise} \end{cases}$$

Loss functions for regression



Loss functions for classification

- **Misclassification loss**

$$L(y, \hat{y}) = \mathbb{I}\{\hat{y} \neq y\} = \begin{cases} 0 & \text{if } \hat{y} = y \\ 1 & \text{if } \hat{y} \neq y \end{cases}$$

  - doesn't care if a decision boundary is near a point or not.
  - Gradient zero everywhere.
- Better to use the probability in the loss: $\Pr(y = 1|\mathrm{x}) = g(\mathrm{x})$.
- **Cross-entropy loss**

$$L(y, g(\mathrm{x})) = \begin{cases} \log g(\mathrm{x}) & \text{if } y = 1 \\ \log(1 - g(\mathrm{x})) & \text{if } y = -1 \end{cases}$$

- Cross-entropy loss = Bernoulli distribution.

The margin for a classifier

- Class prediction by thresholding a function at zero:

$$\hat{y}(\mathrm{x}) = \mathrm{sign}(f(\mathrm{x}))$$

- Example: logistic regression with $f(\mathrm{x}) = \mathrm{x}^\top \beta$.
- The **margin for a classifier** at (x, y) is

$$\mathrm{margin} \equiv y \cdot f(\mathrm{x})$$

- Correct classification if and only if $f(\mathrm{x})$ and y have same sign.
- Misclassification loss in terms of margin

$$L(y \cdot f(\mathrm{x})) = \begin{cases} 1 & \text{if } y \cdot f(\mathrm{x}) < 0 \\ 0 & \text{if } y \cdot f(\mathrm{x}) > 0 \end{cases}$$

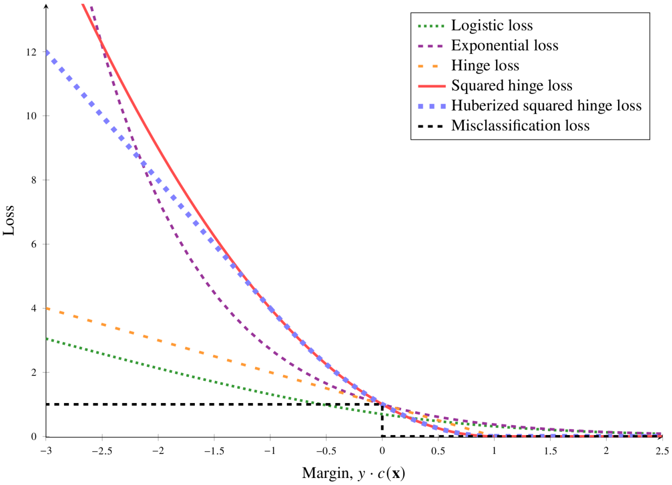
- The larger the margin, the more certain is the classfier.
- Log-likelihood for logistic regression (see eq. 3.34 in MLES):

$$L(y \cdot f(\mathrm{x})) = \log\left(1 + \exp(-y \cdot f(\mathrm{x}))\right)$$

- **Exponential loss**

$$L(y \cdot f(\mathrm{x})) = \exp(-y \cdot f(\mathrm{x})).$$

Loss functions for classification



Log-likelihood as cost function

- Negative **log-likelihood as cost function**

$$J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} \log p(y_i|\mathrm{x}_i;\boldsymbol{\theta})$$

- Classification: cross-entropy.
- Gaussian $\Longrightarrow$ squared loss. Laplace $\Longrightarrow$ absolute loss.
- **Advantages of the log-likelihood as loss function**:
  - ML estimator has good (asymptotic) properties.
  - loss function comes from thinking about properties of the data: skewness, heavy-tails, truncation etc. We can **assess model fit**.
  - natural extension to more complex problems:
    - censoring
    - missing data
    - dependent observations: $J(\boldsymbol{\theta}) = -\log p(y_{1:n}|\mathrm{x}_{1:n};\boldsymbol{\theta})$.
  - log prior for regularization in a Bayesian approach.
  - log-likelihood is a **strictly proper** loss function.

Regularization revisited

- Note: MLES the loglikelihood is multiplied by the factor $\frac{1}{n}$

$$J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} \log p(y_i|\mathrm{x}_i;\boldsymbol{\theta})$$

- This changes the interpretation of λ in L2-regularization and the Ridge estimator

$$\hat{\boldsymbol{\beta}} = (\mathrm{X}^\top \mathrm{X} + n\lambda I)^{-1}\mathrm{X}^\top \mathrm{y}.$$

- **General explicit regularization**

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta};\mathrm{X},\mathrm{y}) + \lambda \cdot R(\boldsymbol{\theta})$$

- Implicit regularization:
  - Early stopping in iterative optimization methods
  - Dropout in neural networks.

Parameter optimization

- Optimization problems in ML:
  - **Parameter learning**

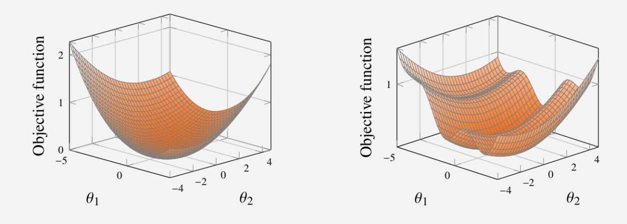
$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ J(\boldsymbol{\theta}; \mathrm{X}, \mathrm{y})$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ J(\boldsymbol{\theta}; \mathrm{X}, \mathrm{y}) + \lambda \cdot R(\boldsymbol{\theta})$$

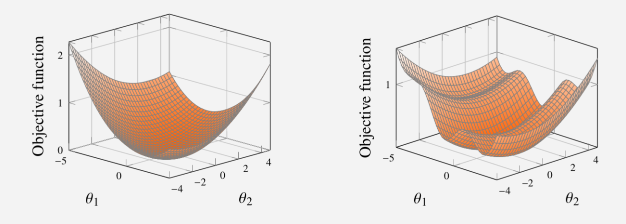
  - **Hyperparameter learning**

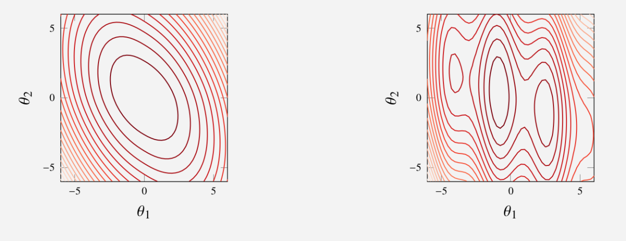
$$\hat{\boldsymbol{\lambda}} = \underset{\boldsymbol{\lambda}}{\operatorname{argmin}}\ E_{\text{hold-out}}(\boldsymbol{\lambda})$$

- Non-convexity and local minima.



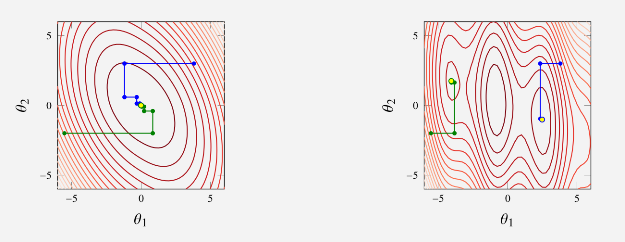
Parameter optimization





Coordinate descent

- Optimize $J(\theta_1, \theta_2, \ldots, \theta_p)$ one coordinate θ_j at the time.
- L1 regularization for squared error loss: coordinate descent guaranteed to reach global minimum.



Gradient descent

- Idea: move in the direction of steepest descent, i.e. opposite direction of the **gradient**

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left(\frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}), \ldots, \frac{\partial}{\partial \theta_p} J(\boldsymbol{\theta}) \right)^\top$$

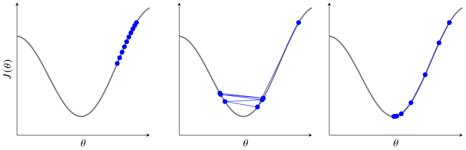
- Example logistic regression

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{1 + e^{y_i \mathbf{x}_i^\top \beta}} \right) y_i \mathbf{x}_i$$

Algorithm 5.1: Gradient descent

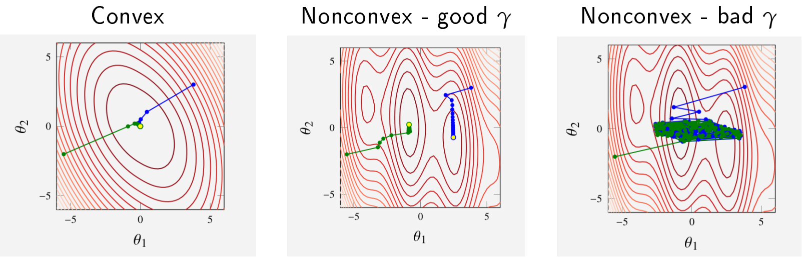
Input: Objective function $J(\boldsymbol{\theta})$, initial $\boldsymbol{\theta}^{(0)}$, learning rate γ
Result: $\widehat{\boldsymbol{\theta}}$

Set $t \leftarrow 0$
while $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t-1)}\|$ *not small enough* **do**
    Update $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$
    Update $t \leftarrow t + 1$
end
return $\widehat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(t-1)}$



(a) Low learning rate $\gamma = 0.05$ **(b)** High learning rate $\gamma = 1.2$ **(c)** Good learning rate $\gamma = 0.3$

Gradient descent



Newton's method

- Gradient descent from first-order Taylor approximation of $J(\boldsymbol{\theta})$.
- **Newton's method** from second-order Taylor approx of $J(\boldsymbol{\theta})$:

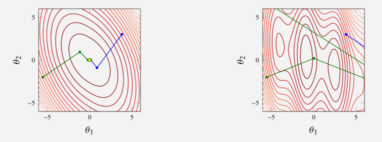
$$J(\boldsymbol{\theta}+\mathrm{v}) \approx \underbrace{J(\boldsymbol{\theta}) + \mathrm{v}^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \frac{1}{2}\mathrm{v}^\top \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta})\mathrm{v}}_{s(\boldsymbol{\theta},\mathrm{v})}$$

- The approx $s(\boldsymbol{\theta},\mathrm{v})$ has minimum (solve $\nabla_{\mathrm{v}} s(\boldsymbol{\theta},\mathrm{v}) = 0$ for v)

$$\mathrm{v} = -\left(\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta})\right)^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

suggesting the iterative update

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \left(\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})\right)^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$$



Newton's method with trust regions

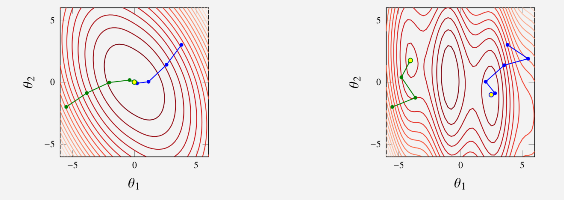
- Newton's method can be unstable. Hessian not always positive definite. Divergence.
- **Newton with trust regions**: don't allow large updates:

Algorithm 5.2: Trust-region Newton's method

Input: Objective function $J(\boldsymbol{\theta})$, initial $\boldsymbol{\theta}^{(0)}$, trust region radius D
Result: $\widehat{\boldsymbol{\theta}}$

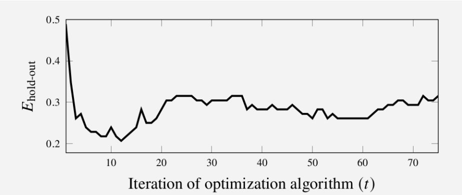
```
Set t ← 0
while ||θ^(t) − θ^(t−1)|| not small enough do
    Compute v ← [∇²_θ J(θ^(t))]^(−1) [∇_θ J(θ^(t))]
    Compute η ← D / max(||v||, D)
    Update θ^(t+1) ← θ^(t) − ηv
    Update t ← t + 1
end
return θ̂ ← θ^(t−1)
```

Set $t \leftarrow 0$
while $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t-1)}\|$ *not small enough* **do**
  Compute $\mathbf{v} \leftarrow [\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})]^{-1} [\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})]$
  Compute $\eta \leftarrow \frac{D}{\max(\|\mathbf{v}\|, D)}$
  Update $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \mathbf{v}$
  Update $t \leftarrow t + 1$
end
return $\widehat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(t-1)}$



Early stopping to avoid overfitting

- Song classification problem with multi-class logistic regression
- 20 degree polynomial with interactions. Overfitting!
- Monitor optimization progress on hold-out data.



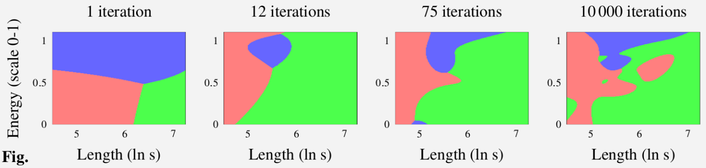


Fig. 5.13

Stochastic gradient descent

- **Large (independent) data**: gradient is a large sum. Costly!

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(\mathrm{x}_i, y_i; \boldsymbol{\theta})$$

- **Stochastic gradient descent**:
  - each iteration computes the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ on a random **mini-batch** of $n_b \ll n$ observations.
  - each mini-batch gradient is an **unbiased estimator** of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.
  - mini-batch gradient descent. Learning rate γ_t in iteration t.
- **Epoch**: a complete sweep through all data. n/n_b iterations.
- Robbins-Monro **conditions for convergence**:
  - $\sum_{t=0}^{\infty} \gamma_t = \infty$ [step length shouldn't vanish too fast]
  - $\sum_{t=0}^{\infty} \gamma_t^2 < \infty$ [... but also not too slow]
  - Fulfilled by $\gamma_t = \frac{1}{t^{\alpha}}$ for $\alpha \in (0.5, 1]$.
- However, often use lower cap: $\gamma_t \to \gamma_{\min} > 0$, e.g.

$$\gamma_t = \gamma_{\min} + (\gamma_{\max} - \gamma_{\min}) e^{-\frac{t}{\tau}}.$$

Stochastic gradient descent

Algorithm 5.3: Stochastic gradient descent

Input: Objective function $J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta})$, initial $\boldsymbol{\theta}^{(0)}$, learning rate $\gamma^{(t)}$

Result: $\widehat{\theta}$

Set $t \leftarrow 0$

while *Convergence criteria not met* **do**

    for $i = 1, 2, \ldots, E$ **do**

        Randomly shuffle the training data $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$

        for $j = 1, 2, \ldots, \frac{n}{n_b}$ **do**

            Approximate the gradient using the mini-batch $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=(j-1)n_b+1}^{jn_b}$, $\widehat{\mathbf{d}}^{(t)} = \frac{1}{n_b}\sum_{i=(j-1)n_b+1}^{jn_b} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}^{(t)})$.

            Update $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma^{(t)}\widehat{\mathbf{d}}^{(t)}$

            Update $t \leftarrow t + 1$

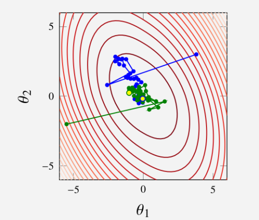
        end

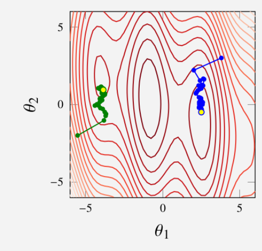
    end

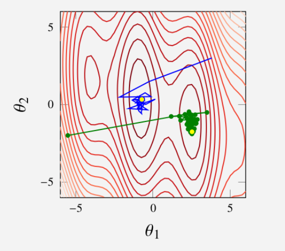
end

return $\widehat{\theta} \leftarrow \boldsymbol{\theta}^{(t-1)}$

Stochastic gradient descent







Stochastic second order gradient methods

- Stochastic **Quasi-Newton** methods. Use past gradients to approximate Hessian info:

$$\text{learning rate:} \qquad \gamma_t = \gamma(\nabla J_t, \nabla J_{t-1}, \ldots, \nabla J_0)$$

$$\text{search directions:} \qquad d_t = d(\nabla J_t, \nabla J_{t-1}, \ldots, \nabla J_0)$$

- **Adaptive Stochastic Gradient Descent**

$$\boldsymbol{\theta}^{(t+1)} = \theta^{(t)} - \gamma_t d_t$$

- The **ADAM** optimizer is based on exponential decay

$$d_t = (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\nabla J_i$$

$$\gamma_t = \frac{\eta}{\sqrt{t}}\left((1-\beta_2)\mathrm{diag}\left(\sum_{i=1}^{t}\beta_2^{t-i}\left\|\nabla J_i\right\|^2\right)\right)^{1/2}.$$

- β_1 and β_2 close to one is a common choice.

Hyperparameter learning

- Minimizing $E_{\text{hold-out}}(\lambda)$ by grid search.

Algorithm 5.4: Grid search for regularization parameter λ

Input: Training data $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$, validation data $\{\mathbf{x}_j, y_j\}_{j=1}^{n_v}$

Result: $\widehat{\lambda}$

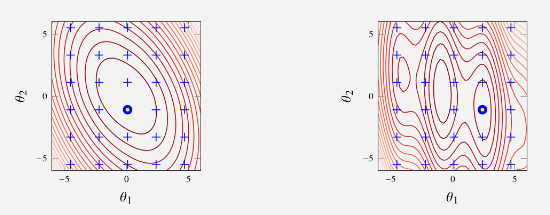
for $\lambda = 10^{-3}, 10^{-2}, \ldots, 10^{3}$ *(as an example)* **do**

  Learn $\widehat{\boldsymbol{\theta}}$ with regularization parameter λ from training data

  Compute error on validation data $E_{\text{val}}(\lambda) \leftarrow \frac{1}{n_v} \sum_{j=1}^{n_v} (\widehat{y}(\mathbf{x}_j; \widehat{\boldsymbol{\theta}}) - y_j)^2$

end

return $\widehat{\lambda}$ as $\arg\min_{\lambda} E_{\text{val}}(\lambda)$



- **Bayesian optimization** based on Gaussian processes (Lecture 8) when objective is costly and λ low-dim.