Neural networks and optimization

Nicolas Le Roux

13/11/12
1 Introduction

2 Linear classifier

3 Convolutional neural networks

4 Stochastic gradient descent
I’m here for you, I already know that stuff

It’s better to look silly than to stay so

Ask questions if you don’t understand!
Goal: classification and regression

- Medical imaging: cancer or not? **Classification**
- Autonomous driving: optimal wheel position **Regression**
- Kinect: where are the limbs? **Regression**
- OCR: what are the characters? **Classification**
Goal: classification and regression

- Medical imaging: cancer or not? **Classification**
- Autonomous driving: optimal wheel position **Regression**
- Kinect: where are the limbs? **Regression**
- OCR: what are the characters? **Classification**

Regression and classification are similar problems.
Goal: real-time object recognition
Linear classifier

- Dataset: \((X^{(i)}, Y^{(i)})\) pairs, \(i = 1, \ldots, N\).
- \(X^{(i)} \in \mathbb{R}^n\), \(Y^{(i)} \in \{-1, 1\}\).
- Goal: Find \(w\) and \(b\) such that \(\text{sign}(w^\top X^{(i)} + b) = Y^{(i)}\).
Linear classifier

- Dataset: \((X^{(i)}, Y^{(i)})\) pairs, \(i = 1, \ldots, N\).
- \(X^{(i)} \in \mathbb{R}^n\), \(Y^{(i)} \in \{-1, 1\}\).
- Goal: Find \(w\) and \(b\) such that \(\text{sign}(w^\top X^{(i)} + b) = Y^{(i)}\).
Perceptron algorithm (Rosenblatt, 57)

- $w_0 = 0, \ b_0 = 0$
- $\hat{Y}^{(i)} = \text{sign}(w^{\top}X^{(i)} + b)$
- $w_{t+1} \leftarrow w_t + \sum_i (Y^{(i)} - \hat{Y}^{(i)})X^{(i)}$
- $b_{t+1} \leftarrow b_t + \sum_i (Y^{(i)} - \hat{Y}^{(i)})$

Movie \textit{linearly\_separable\_perceptron.avi}
Some data are not separable

The Perceptron algorithm is **NOT** convergent for non linearly separable data.
Non convergence of the perceptron algorithm

- Movie non_linearly_separable_perceptron.avi

- We need an algorithm which works both on separable and non separable data.
Cost function

- Classification error is not smooth.
Cost function

- Classification error is not smooth.
- Sigmoid is smooth but not convex.

Convexity guarantees the same solution every time.

In practice, it is not always crucial.
Convex cost functions

- Classification error is not smooth.
- Sigmoid is smooth but not convex.
- Logistic loss is a convex upper bound.
Convex cost functions

- Classification error is not smooth.
- Sigmoid is smooth but not convex.
- Logistic loss is a convex upper bound.
- Hinge loss (SVMs) is very much like logistic.
Solving separable AND non-separable problems

Movie non_linearly_separable_logistic.avi Movie linearly_separable_logistic.avi
Non-linear classification

Movie non_linearly_separable_poly_kernel.avi
Non-linear classification

- Features: $X_1, X_2 \rightarrow$ linear classifier
- Features: $X_1, X_2, X_1 X_2, X_1^2, \ldots \rightarrow$ non-linear classifier
Choosing the features

To make it work, I created lots of extra features:

$$(X_1, X_2, X_1 X_2, X_1^2 X_2, X_1 X_2^2)^{(1,2,3,\ldots,10)}$$
Choosing the features

To make it work, I created lots of extra features:

\((X_1, X_2, X_1X_2, X_1^2X_2, X_1X_2^2)^{(1,2,3,...,10)}\)

Would it work with fewer features?

Test with \((X_1, X_2, X_1X_2, X_1^2X_2, X_1X_2^2)^{(1,2)}\)

Movie non_linearly_separable_poly_2.avi
A graphical view of the classifiers

\[ f(X) = w_1 X_1 + w_2 X_2 + b \]
A graphical view of the classifiers

\[
f(X) = w_1 X_1 + w_2 X_2 + w_3 X_1^2 + w_4 X_2^2 + w_5 X_1 X_2 + \ldots
\]
Non-linear features

- A linear classifier on a non-linear transformation is non-linear.
- A non-linear classifier relies on non-linear features.
- Which ones do we choose?
Non-linear features

- A linear classifier on a non-linear transformation is non-linear.
- A non-linear classifier relies on non-linear features.
- Which ones do we choose?

Example: \( H_j = X_1^{p_j} X_2^{q_j} \)
Non-linear features

- A linear classifier on a non-linear transformation is non-linear.
- A non-linear classifier relies on non-linear features.
- Which ones do we choose?
- Example: $H_j = X_1^{p_j} X_2^{q_j}$
- SVM: $H_j = K(X, X^{(j)})$ with $K$ some kernel function
Non-linear features

- A linear classifier on a non-linear transformation is non-linear.

- A non-linear classifier relies on non-linear features.

- Which ones do we choose?

- Example: $H_j = X_1^{p_j} X_2^{q_j}$

- SVM: $H_j = K(X, X^{(j)})$ with $K$ some kernel function

- Do they have to be predefined?
A neural network will learn the $H_j$'s
A neural network will learn the $H_j$'s

Usually, we use

$$H_j = g(v_j^\top X)$$
A neural network will learn the $H_j$'s

Usually, we use

$$H_j = g(v_j^\top X)$$

$H_j$ : Hidden unit
$v_j$ : Input weight
$g$ : Transfer function
Transfer function

\[ f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^\top X) + b \]

- \( g \) is the transfer function.
- Usually, \( g \) is the sigmoid or the tanh.
Neural networks

\[ f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^T X) + b \]
Example on the non-separable problem

Movie *non_linearly_separable_mlp_3.avi*
Training a neural network

- Dataset: \((X^{(i)}, Y^{(i)})\) pairs, \(i = 1, \ldots, N\).

- Goal: Find \(w\) and \(b\) such that

\[
\text{sign} \left( w^\top X^{(i)} + b \right) = Y^{(i)}
\]
Training a neural network

Dataset: \((X^{(i)}, Y^{(i)})\) pairs, \(i = 1, \ldots, N\).

Goal: Find \(w\) and \(b\) to minimize

\[
\sum_{i} \log (1 + \exp (-Y^{(i)} (w^T X^{(i)} + b)))
\]
Training a neural network

- Dataset: \((X^{(i)}, Y^{(i)})\) pairs, \(i = 1, \ldots, N\).

- Goal: Find \(v_1, \ldots, v_k, w\) and \(b\) to minimize

\[
\sum_i \log \left( 1 + \exp \left( -Y^{(i)} \left[ \sum_j w_j g(v_j^\top X^{(i)}) \right] \right) \right)
\]
Neural network - 8 hidden units

**Movie** non_linearly_separable_mlp_8.avi
Neural network - 5 hidden units

Movie non_linearly_separable_mlp_5.avi
Neural network - 3 hidden units

Movie non_linearly_separable_mlp_3.avi
Neural network - 2 hidden units

Movie non_linearly_separable_mlp_2.avi
Non-linear classification
Non-linear classification
Non-linear classification
Cost function

\[ s = \text{cost function (logistic loss, hinge loss, ...)} \]

\[
\ell(v, w, b, X(i), Y(i)) = s \left( \hat{Y}(i), Y(i) \right) \\
= s \left( \sum_j w_j H_j(X(i)), Y(i) \right) \\
= s \left( \sum_j w_j g(v_j^\top X(i)), Y(i) \right)
\]
Backpropagation - Output weights

\( s = \text{cost function (logistic loss, hinge loss, ...)} \)

\[
\hat{Y}^{(i)} = \sum_j w_j H_j(X^{(i)})
\]

\[
\frac{\partial \ell(v, w, b, X^{(i)}, Y^{(i)})}{\partial w_j} = \frac{\partial \ell(v, w, b, X^{(i)}, Y^{(i)})}{\partial \hat{Y}^{(i)}} \frac{\partial \hat{Y}^{(i)}}{\partial w_j} = \frac{\partial \ell(v, w, b, X^{(i)}, Y^{(i)})}{\partial \hat{Y}^{(i)}} H_j(X^{(i)})
\]
Backpropagation - Input weights

\[ s = \text{cost function (logistic loss, hinge loss, ...)} \]

\[ \hat{Y}^{(i)} = \sum_j w_j H_j(X^{(i)}) \]

\[ H_j(X^{(i)}) = g(v_j^\top X^{(i)}) \]

\[ \frac{\partial \ell(v, w, b, X^{(i)}, Y^{(i)})}{\partial v_j} = \frac{\partial \ell(v, w, b, X^{(i)}, Y^{(i)})}{\partial H_j(X^{(i)})} \frac{\partial H_j(X^{(i)})}{\partial v_j} \]

\[ = \frac{\partial \ell(v, w, b, X^{(i)}, Y^{(i)})}{\partial \hat{Y}^{(i)}} \frac{\partial \hat{Y}^{(i)}}{\partial H_j(X^{(i)})} \frac{\partial H_j(X^{(i)})}{\partial v_j} \]

\[ = \frac{\partial \ell(v, w, b, X^{(i)}, Y^{(i)})}{\partial \hat{Y}^{(i)}} w_j X^{(i)} g'(v_j^\top X^{(i)}) \]
Training neural networks - Summary

- For each datapoint, compute the gradient of the cost with respect to the weights.
- Done using the backpropagation of the gradient.
- Convex with respect to the output weights (linear classifier).
- NOT convex with respect to the input weights: POTENTIAL PROBLEMS!
A linear classifier in a feature space can model non-linear boundaries.

Finding a good feature space is essential.

One can design the feature map by hand.

One can learn the feature map, using fewer features than if it done by hand.

Learning the feature map is potentially HARD (non-convexity).
Neural networks - Not summary

- Linear combination of the output of soft classifiers.
- This is a non-linear classifier.
- One can take a linear combination of these.
Neural networks - Not summary

- Linear combination of the output of soft classifiers.
- This is a non-linear classifier.
- One can take a linear combination of these.
- This becomes a neural network with two hidden layers.
Advantages of neural networks

- They can learn anything.
- Extremely fast at test time (computing the answer for a new datapoint) because fewer features.
- Complete control over the power of the network (by controlling the hidden layers sizes).
Problems of neural networks

- Highly non-convex $\rightarrow$ many local minima
- Can learn anything but have more parameters $\rightarrow$ need tons of examples to be good.
Take-home messages

- Neural networks are potentially extremely efficient.
- But it is HARD to train them!
- If you wish to use them, be smart (or ask someone who knows)!
- If you have a huge dataset, they CAN be awesome!
$v_j$’s for images

\[ f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^\top X) + b \]

- If $X$ is an image, $v_j$ is an image too.
- $v_j$ acts as a filter (presence or absence of a pattern).
- What does $v_j$ look like?
$\nu_j$'s for images - Examples
Filters are mostly local
Basic idea of convolutional neural networks

- Filters are mostly local.
- Instead of using image-wide filters, use small ones over patches.
- Repeat for every patch to get a response image.
- Subsample the response image to get local invariance.
Filtering - Filter 1

Original image  Filter  Output image
Filtering - Filter 2

Original image  Filter  Output image
Filtering - Filter 3

Original image  Filter  Output image

Nicolas Le Roux (Criteo) Neural networks and optimization 13/11/12 42 / 80
Pooling - Filter 1

Original image  |  Output image  |  Subsampled image

How to do 2x subsampling-pooling:

- Output image = $O$, subsampled image = $S$.
- $S_{ij} = \max_k$ over window around $(2i, 2j)$ $O_k$. 

Nicolas Le Roux (Criteo)  |  Neural networks and optimization  |  13/11/12  43 / 80
Pooling - Filter 2

Original image  | Output image  | Subsampled image

How to do 2x subsampling-pooling:

- Output image = $O$, subsampled image = $S$.

- $S_{ij} = \max_k$ over window around $(2i,2j)$ $O_k$. 
Pooling - Filter 3

How to do 2x subsampling-pooling:

- Output image $= O$, subsampled image $= S$.
- $S_{ij} = \max_k$ over window around $(2i, 2j) \ O_k$. 

Nicolas Le Roux (Criteo)
A convolutional layer

“Simple cells”

“Complex cells”

Convolutions

pooling
subsampling

Retinotopic Feature Maps
Transforming the data with a layer

Original datapoint

New datapoint
A convolutional network
Face detection
Face detection
NORB dataset

- 50 toys belonging to 5 categories
  - animal, human figure, airplane, truck, car
- 10 instance per category
  - 5 instances used for training, 5 instances for testing
- Raw dataset
  - 972 stereo pairs of each toy. 48,600 image pairs total.
NORB dataset - 2

For each instance:

- 18 azimuths
- 0 to 350 degrees every 20 degrees
- 9 elevations
- 30 to 70 degrees from horizontal every 5 degrees
- 6 illuminations
- on/off combinations of 4 lights
- 2 cameras (stereo), 7.5 cm apart
- 40 cm from the object
NORB dataset - 3

Training instances

Test instances
Textured and cluttered versions
- 90,857 free parameters, 3,901,162 connections.
- The entire network is trained end-to-end (all the layers are trained simultaneously).
## Normalized-Uniform set

<table>
<thead>
<tr>
<th>Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Classifier on raw stereo images</td>
<td>30.2%</td>
</tr>
<tr>
<td>K-Nearest-Neighbors on raw stereo images</td>
<td>18.4%</td>
</tr>
<tr>
<td>K-Nearest-Neighbors on PCA-95</td>
<td>16.6%</td>
</tr>
<tr>
<td>Pairwise SVM on 96x96 stereo images</td>
<td>11.6%</td>
</tr>
<tr>
<td>Pairwise SVM on 95 Principal Components</td>
<td>13.3%</td>
</tr>
<tr>
<td>Convolutional Net on 96x96 stereo images</td>
<td>5.8%</td>
</tr>
</tbody>
</table>
Jittered-Cluttered Dataset

- 291,600 stereo pairs for training, 58,320 for testing
- Objects are jittered
  - position, scale, in-plane rotation, contrast, brightness, backgrounds, distractor objects,...
- Input dimension : 98x98x2 (approx 18,000)
### Jittered-Cluttered Dataset - Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM with Gaussian kernel</td>
<td>43.3%</td>
</tr>
<tr>
<td>Convolutional Net with binocular input</td>
<td>7.8%</td>
</tr>
<tr>
<td>Convolutional Net + SVM on top</td>
<td>5.9%</td>
</tr>
<tr>
<td>Convolutional Net with monocular input</td>
<td>20.8%</td>
</tr>
<tr>
<td>Smaller mono net (DEMO)</td>
<td>26.0%</td>
</tr>
</tbody>
</table>

Dataset available from [http://www.cs.nyu.edu/~yann](http://www.cs.nyu.edu/~yann)
NORB recognition - 1
NORB recognition - 2
NORB recognition - 3
NORB recognition - 4
NORB recognition - 5

Zoom= 1.0, Threshold= -1.2, filter on

car [-0.3]

plane [2.5] animal [3.0]
NORB recognition - 6

Zoom = 0.7, Threshold = -1.8, filter on

plane [-1.1]
plane [-0.8]
animal [-0.6]
truck [-0.8]
With complex problems, it is hard to design features by hand.

Neural networks circumvent this problem.

They can be hard to train (again...).

Convolutional neural networks use knowledge about locality in images.

They are much easier than standard networks.

And they are FAST (again...).
What has not been covered

- In some cases, we have lots of data, but without the labels.
- *Unsupervised* learning.
- There are techniques to use these data to get better performance.
- E.g. : *Task-Driven Dictionary Learning*, Mairal et al.
The need for fast learning

- Neural networks may need many examples (several millions or more).
- We need to be able to use them quickly.
Batch methods

\[ L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)}) \]

\[ \theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta} \]

- To compute one update of the parameters, we need to go through all the data.
- This can be very expensive.
- What if we have an infinite amount of data?
Potential solutions

1. Discard data.
   - Seems stupid
   - Yet many people do it
Potential solutions

1. Discard data.
   - Seems stupid
   - Yet many people do it

2. Use approximate methods.
   - Update = average of the updates for all datapoints.
   - Are these update really different?
   - If not, how can we learn faster?
Stochastic gradient descent

\[ L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)}) \]

\[ \theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta} \]
Stochastic gradient descent

\[ L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)}) \]

\[ \theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta} \]
Stochastic gradient descent

\[ L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)}) \]

\[ \theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta} \]

What do we lose when updating the parameters to satisfy just one example?
Disagreement

- $||\mu||^2/\sigma^2$ during optimization (log scale)
- As optimization progresses, disagreement increases
- Early on, one can pick one example at a time
- What about later?
Training vs test

Standard learning paradigm:

- We want to solve a task on new datapoints.
- We have a training set.
- We hope that the performance on the training set is informative of the performance on new datapoints.

Can we know when we start overfitting?
Overfitting

- When all gradients disagree, stochastic error stalls.
- When all gradients disagree, training and test error part.

**IT DOES NOT MATTER IF ONE DOES NOT REACH THE MINIMUM OF THE TRAINING ERROR!**
Decomposition of the error

\[ E(\tilde{f}_n) - E(f^*) = E(f^*_F) - E(f^*) \] Approximation error
\[ + E(f_n) - E(f^*_F) \] Estimation error
\[ + E(\tilde{f}_n) - E(f_n) \] Optimization error

Questions:

1. Do we optimize the training error to decrease \( E(\tilde{f}_n) - E(f_n) \)?
2. Do we increase \( n \) to decrease \( E(f_n) - E(f^*_F) \)?
Gradient Descent (GD)

Iterate
\[ w_{t+1} \leftarrow w_t - \eta \frac{\partial E_n(f_w)}{\partial w} \]

Best speed achieved with fixed learning rate \( \eta = \frac{1}{\lambda_{\text{max}}} \).
(e.g., Dennis & Schnabel, 1983)

<table>
<thead>
<tr>
<th>Cost per iteration</th>
<th>Iterations to reach ( \rho )</th>
<th>Time to reach accuracy ( \rho )</th>
<th>Time to reach ( E(\tilde{f}<em>n) - E(f^*</em>\mathcal{F}) &lt; \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{O}(nd) )</td>
<td>( \mathcal{O}(\kappa \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}(nd \kappa \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}\left(\frac{d^2 \kappa}{\varepsilon^2 \log^2 \frac{1}{\varepsilon}}\right) )</td>
</tr>
</tbody>
</table>

- In the last column, \( n \) and \( \rho \) are chosen to reach \( \varepsilon \) as fast as possible.
- Solve for \( \varepsilon \) to find the best error rate achievable in a given time.
- Remark: abuses of the \( \mathcal{O}() \) notation.
Second Order Gradient Descent (2GD)

\[ w_{t+1} = w_t - H^{-1} \frac{\partial E_n(f_{w_t})}{\partial w} \]

We assume \( H^{-1} \) is known in advance.
Superlinear optimization speed (e.g., Dennis & Schnabel, 1983)

<table>
<thead>
<tr>
<th></th>
<th>Cost per iteration</th>
<th>Iterations to reach ( \rho )</th>
<th>Time to reach accuracy ( \rho )</th>
<th>Time to reach ( E(\tilde{f}_n) - E(f^*_F) &lt; \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2GD</td>
<td>( \mathcal{O}(d(d+n)) )</td>
<td>( \mathcal{O}(\log \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}(d(d+n) \log \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}\left(\frac{d^2}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon}\right) )</td>
</tr>
</tbody>
</table>

- Optimization speed is much faster.
- Learning speed only saves the condition number \( \kappa \).
**Stochastic Gradient Descent (SGD)**

Iterate

- Draw random example \((x_t, y_t)\).
- \(w_{t+1} \leftarrow w_t - \frac{\eta}{t} \frac{\partial l(f_w(x_t), y_t)}{\partial w}\)

Best decreasing gain schedule with \(\eta = \frac{1}{\lambda_{\text{min}}}\).
(see Murata, 1998; Bottou & LeCun, 2004)

<table>
<thead>
<tr>
<th>Cost per iteration</th>
<th>Iterations to reach (\rho)</th>
<th>Time to reach accuracy (\rho)</th>
<th>Time to reach (E(\tilde{f}<em>n) - E(f</em>\star) &lt; \varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>(\mathcal{O}(d))</td>
<td>(\frac{\nu k}{\rho} + o\left(\frac{1}{\rho}\right))</td>
<td>(\mathcal{O}\left(\frac{d \nu k}{\rho}\right))</td>
</tr>
<tr>
<td></td>
<td>With (1 \leq k \leq \kappa^2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Optimization speed is *catastrophic*.
- Learning speed does not depend on the statistical estimation rate \(\alpha\).
- Learning speed depends on condition number \(\kappa\) but *scales very well*. 

Slide from Léon Bottou
Second order Stochastic Descent (2SGD)

Iterate
- Draw random example \((x_t, y_t)\).
- \(w_{t+1} \leftarrow w_t - \frac{1}{t} H^{-1} \frac{\partial \ell(f_{w_t}(x_t), y_t)}{\partial w}\)

Replace scalar gain \(\frac{\eta}{t}\) by matrix \(\frac{1}{t} H^{-1}\).

<table>
<thead>
<tr>
<th></th>
<th>Cost per iteration</th>
<th>Iterations to reach (\rho)</th>
<th>Time to reach accuracy (\rho)</th>
<th>Time to reach (E(\tilde{f}<em>n) - E(f^*</em>\calF) &lt; \varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2SGD</td>
<td>(O(d^2))</td>
<td>(\frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right))</td>
<td>(O\left(\frac{d^2 \nu}{\rho}\right))</td>
<td>(O\left(\frac{d^2 \nu}{\varepsilon}\right))</td>
</tr>
</tbody>
</table>

- Each iteration is \(d\) times more expensive.
- The number of iterations is reduced by \(\kappa^2\) (or less.)
- Second order only changes the constant factors.
Summary

- Stochastic methods update the parameters much more often than batch ones.
Summary

- Stochastic methods update the parameters much more often than batch ones.
- They are terrible to find the minimum of the training error.
Summary

- Stochastic methods update the parameters much more often than batch ones.
- They are terrible to find the minimum of the training error.
- It may not matter as the training error is not the test error.
Summary

- Stochastic methods update the parameters much more often than batch ones.
- They are terrible to find the minimum of the training error.
- It may not matter as the training error is not the test error.

In practice:

- You will ALMOST ALWAYS have enough data.
Summary

- Stochastic methods update the parameters much more often than batch ones.
- They are terrible to find the minimum of the training error.
- It may not matter as the training error is not the test error.

In practice:
- You will ALMOST ALWAYS have enough data.
- You will ALMOST ALWAYS lack time.
Summary

- Stochastic methods update the parameters much more often than batch ones.
- They are terrible to find the minimum of the training error.
- It may not matter as the training error is not the test error.

In practice:

- You will ALMOST ALWAYS have enough data.
- You will ALMOST ALWAYS lack time.
- You must ALMOST ALWAYS use stochastic methods.
Summary

- Stochastic methods update the parameters much more often than batch ones.
- They are terrible to find the minimum of the training error.
- It may not matter as the training error is not the test error.

In practice:

- You will ALMOST ALWAYS have enough data.
- You will ALMOST ALWAYS lack time.
- You must ALMOST ALWAYS use stochastic methods.
- How to use accelerated techniques remains to be seen.