CENG 783

On Machine Learning and Optimization

Sinan Kalkan
Optimization Methods for Large-Scale Machine Learning

Léon Bottou*  Frank E. Curtis†  Jorge Nocedal‡

June 16, 2016

Abstract

This paper provides a review and commentary on the past, present, and future of numerical optimization algorithms in the context of machine learning applications. Through case studies on text classification and the training of deep neural networks, we discuss how optimization problems arise in machine learning and what makes them challenging. A major theme of our study is that large-scale machine learning represents a distinctive setting in which the stochastic gradient (SG) method has traditionally played a central role while conventional gradient-based nonlinear optimization techniques typically falter. Based on this viewpoint, we present a comprehensive theory of a straightforward, yet versatile SG algorithm, discuss its practical behavior, and highlight opportunities for designing algorithms with improved performance. This leads to a discussion about the next generation of optimization methods for large-scale machine learning, including an investigation of two main streams of research on techniques that diminish noise in the stochastic directions and methods that make use of second-order derivative approximations.
Now

- **Introduction to ML**
  - Problem definition
  - Classes of approaches
  - K-NN
  - Support Vector Machines
  - Softmax classification / logistic regression
  - Parzen Windows

- **Optimization**
  - Gradient Descent approaches
  - A flavor of other approaches
Introduction to Machine learning

Uses many figures and material from the following website:
Overview

Test input

Test input

Extract Features

Learned Models or Classifiers

Predict

Human or Animal or ...

Training

Data with label (label: human, animal etc.)

Extract Features

“Learn”

- Size
- Texture
- Color
- Histogram of oriented gradients
- SIFT
- Etc.
Content

• Problem definition
• General approaches
• Popular methods
  – kNN
  – Linear classification
  – Support Vector Machines
  – Parzen windows
Problem Definition

• Given
  – Data: a set of instances \( x_1, x_2, \ldots, x_n \) sampled from a space \( X \in \mathbb{R}^d \).
  – Labels: the corresponding labels \( y_i \) for each \( x_i \). \( y_i \in Y \in \mathbb{R} \).

• Goal:
  – Learn a mapping from the space of “data” to the space of labels, i.e.,
    • \( M : X \rightarrow Y \).
    • \( y = f(x) \).
Issues in Machine Learning

• Hypothesis space
• Loss / cost / objective function
• Optimization
• Bias vs. variance
• Test / evaluate
• Overfitting, underfitting
General Approaches

Discriminative

Find separating line (in general: hyperplane)

Generative

Learn a model for each class.

Fit a function to data (regression).
Generative Approaches (cont’d)

**Generative**
- Regression
- Markov Random Fields
- Bayesian Networks/Learning
- Clustering via Gaussian Mixture Models, Parzen Windows etc.
- ...

**Discriminative**
- Support Vector Machines
- Artificial Neural Networks
- Conditional Random Fields
- K-NN

![Discriminative Approach Example](image1)

![Generative Approach Example](image2)
### General Approaches (cont’d)

#### Supervised

<table>
<thead>
<tr>
<th>Instance</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="elma.png" alt="" /></td>
<td>elma</td>
</tr>
<tr>
<td><img src="armut.png" alt="" /></td>
<td>armut</td>
</tr>
<tr>
<td><img src="elma.png" alt="" /></td>
<td>elma</td>
</tr>
<tr>
<td><img src="armut.png" alt="" /></td>
<td>armut</td>
</tr>
</tbody>
</table>

1. **Extract Features**
2. **Learn a model**

- e.g. SVM

#### Unsupervised

1. **Extract Features**
2. **Learn a model**

- e.g. k-means clustering
# General Approaches (cont’d)

<table>
<thead>
<tr>
<th></th>
<th>Generative</th>
<th>Discriminative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Supervised</strong></td>
<td>Regression, Markov Random Fields, Bayesian Networks</td>
<td>Support Vector Machines, Neural Networks, Conditional Random Fields, Decision Tree Learning</td>
</tr>
<tr>
<td><strong>Unsupervised</strong></td>
<td>Gaussian Mixture Models, Parzen Windows</td>
<td>K-means, Self-Organizing Maps</td>
</tr>
</tbody>
</table>
Now

<table>
<thead>
<tr>
<th></th>
<th>Generative</th>
<th>Discriminative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Supervised</strong></td>
<td>Softmax/Logistic Regression</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td><strong>Unsupervised</strong></td>
<td>Parzen Windows</td>
<td></td>
</tr>
</tbody>
</table>
Feature Space

(a) Good features vs. Bad features

(b) Linear separability, Non-linear separability, Multi-modal, Highly correlated
General pitfalls/problems

- Overfitting
- Underfitting
- Occam’s razor
K-NEAREST NEIGHBOR
A very simple algorithm

- **Advantages:**
  - No training
  - Simple

- **Disadvantages:**
  - Slow testing time (more efficient versions exist)
  - Needs a lot of memory

http://cs231n.github.io/classification/
A non-parametric method

PARZEN WINDOWS
Probability and density

• Probability of a continuous probability function, \( p(x) \), satisfies the following:
  – Probability to be between two values:
    \[
P(a < x < b) = \int_{a}^{b} p(x)dx
    \]
  – \( p(x) > 0 \) for all real \( x \).
  – And
    \[
    \int_{-\infty}^{\infty} p(x)dx = 1
    \]

• In 2-D:
  – Probability for \( x \) to be inside region \( R \):
    \[
P = \int_{R} p(x)dx
    \]
  – \( p(x) > 0 \) for all real \( x \).
  – And
    \[
    \int_{-\infty}^{\infty} p(x)dx = 1
    \]
Density Estimation

• Basic idea:

\[ P = \int_R p(x) \, dx \]

• If \( R \) is small enough, so that \( p(x) \) is almost constant in \( R \):
  
  \(-\quad P = \int_R p(x) \, dx \approx p(x) \int_R \, dx = p(x) V\)

  \(-\quad V: \text{volume of region} \ R.\)

• If \( k \) out of \( n \) samples fall into \( R \), then

\[ P = k/n \]

• From which we can write:

\[ \frac{k}{n} = p(x)V \Rightarrow p(x) = \frac{k/n}{V} \]
Parzen Windows

• Assume that:
  – $R$ is a hypercube centered at $\mathbf{x}$.
  – $h$ is the length of an edge of the hypercube.
  – Then, $V = h^2$ in 2D and $V = h^3$ in 3D etc.

• Let us define the following window function:

$$w \left( \frac{\mathbf{x}_i - \mathbf{x}_k}{h} \right) = \begin{cases} 
1, & \frac{|\mathbf{x}_i - \mathbf{x}_k|}{h} < 1/2 \\
0, & \text{otherwise}
\end{cases}$$

• Then, we can write the number of samples falling into $R$ as follows:

$$k = \sum_{i=1}^{n} w \left( \frac{\mathbf{x}_i - \mathbf{x}_k}{h} \right)$$
Parzen Windows (cont’d)

• Remember: $p(x) = \frac{k/n}{V}$.

• Using this definition of $k$, we can rewrite $p(x)$ (in 2D):

$$p(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^2} \cdot w \left( \frac{x_i - x}{h} \right)$$

• Interpretation: Probability is the contribution of window functions fitted at each sample!
Parzen Windows (cont’d)

• The type of the window function can add different flavors.

• If we use Gaussian for the window function:

\[
p(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\sigma \sqrt{2\pi}} \exp \left( - \frac{(x_i - x)^2}{2\sigma^2} \right)
\]

• Interpretation: Fit a Gaussian to each sample.
LINEAR CLASSIFICATION
Linear classification

- Linear classification relies on the following score function:

\[ f(x_i; W, b) = W x_i + b \]

Or

\[ f(x_i; W, b) = W x_i \]

- where the bias is implicitly represented in \( W \) and \( x_i \)

One row per class

http://cs231n.github.io/linear-classify/
Linear classification

• We can rewrite the score function as:

\[ f(x_i; W, b) = W x_i \]

• where the bias is implicitly represented in \( W \) and \( x_i \)

http://cs231n.github.io/linear-classify/
Linear classification: One interpretation

- Since an image can be thought as a vector, we can consider them as points in high-dimensional space.

One interpretation of:

\[ f(x_i; W, b) = W x_i + b \]

- Each row describes a line for a class, and “b”
Linear classification: Another interpretation

- Each row in $W$ can be interpreted as a template of that class.
  - $f(x_i; W, b) = Wx_i + b$ calculates the inner product to find which template best fits $x_i$.
  - Effectively, we are doing Nearest Neighbor with the “prototype” images of each class.

http://cs231n.github.io/linear-classify/
Loss function

• A function which measures how good our weights are.
  – Other names: cost function, objective function
• Let \( s_j = f(x_i; W)_j \)
• An example loss function:

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)
\]

Or equivalently:

\[
L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)
\]

• This directs the distances to other classes to be more than \( \Delta \) (the margin)

http://cs231n.github.io/linear-classify/
Example

• Consider our scores for $x_i$ to be $s = [13, -7, 11]$ and assume $\Delta$ as 10.

• Then,

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

http://cs231n.github.io/linear-classify/
Regularization

• In practice, there are many possible solutions leading to the same loss value.
  – Based on the requirements of the problem, we might want to penalize certain solutions.
• E.g.,

\[ R(W) = \sum_i \sum_j W_{i,j}^2 \]

  – which penalizes large weights.
  • Why do we want to do that?

http://cs231n.github.io/linear-classify/
Combined Loss Function

• The loss function becomes:

\[
L = \frac{1}{N} \sum_{i} L_{i} + \lambda R(W)
\]

\[
\text{data loss} \quad \text{regularization loss}
\]

• If you expand it:

\[
L = \frac{1}{N} \sum_{i} \sum_{j \neq y_i} \left[ \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \right] + \lambda \sum_{i} \sum_{j} W_{i,j}^2
\]

Hyper parameters (estimated using validation set)

http://cs231n.github.io/linear-classify/
Hinge Loss, or Max-Margin Loss

\[ L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \right] + \lambda \sum_i \sum_j W_{i,j}^2 \]

http://cs231n.github.io/linear-classify/
Interactive Demo

http://vision.stanford.edu-teaching/cs231n-linear-classify-demo/
An alternative formulation of

SUPPORT VECTOR MACHINES

Barrowed mostly from the slides of:
- Machine Learning Group, University of Texas at Austin.
- Mingyue Tan, The University of British Columbia
Linear Separators

- Binary classification can be viewed as the task of separating classes in feature space:
  \[ w^T x + b = 0 \]
  \[ w^T x + b > 0 \]
  \[ w^T x + b < 0 \]

\[ f(x) = \text{sign}(w^T x + b) \]
Linear Separators

• Which of the linear separators is optimal?
Classification Margin

- Distance from example \( x_i \) to the separator is \( r = \frac{w^T x_i + b}{\|w\|} \).
- Examples closest to the hyperplane are support vectors.
- **Margin** \( \rho \) of the separator is the distance between support vectors.

\[
\rho = r
\]
Maximum Margin Classification

• Maximizing the margin is good according to intuition.
• Implies that only support vectors matter; other training examples are ignorable.
Linear SVM Mathematically

What we know:

• $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
• $\mathbf{w} \cdot \mathbf{x}^- + b = -1$
• $\mathbf{w} \cdot (\mathbf{x}^+-\mathbf{x}^-) = 2$

\[
\rho = \frac{(\mathbf{x}^+ - \mathbf{x}^-) \cdot \mathbf{w}}{|\mathbf{w}|} = \frac{2}{|\mathbf{w}|}
\]

\(\rho=\text{Margin Width}\)
Linear SVMs Mathematically (cont.)

- Then we can formulate the *optimization problem*:

  Find $w$ and $b$ such that
  
  $\rho = \frac{2}{\|w\|}$ is maximized

  and for all $(x_i, y_i)$, $i=1..n$: $y_i(w^T x_i + b) \geq 1$

  Which can be reformulated as:

  Find $w$ and $b$ such that

  $\Phi(w) = \|w\|^2 = w^T w$ is minimized

  and for all $(x_i, y_i)$, $i=1..n$: $y_i (w^T x_i + b) \geq 1$
Lagrange Multipliers

• Given the following optimization problem:
  \[
  \text{minimize } f(x, y) \text{ subject to } g(x, y) = c
  \]

• We can formulate it as:
  \[
  L(x, y, \lambda) = f(x, y) + \lambda(g(x, y) - c)
  \]

• and set the derivative to zero:
  \[
  \nabla_{x, y, \lambda} L(x, y, \lambda) = 0
  \]
Lagrange Multipliers

• Main intuition:
  - The gradients of $f$ and $g$ are parallel at the maximum

\[ \nabla f = \lambda \nabla g \]

Fig: http://mathworld.wolfram.com/LagrangeMultiplier.html
Lagrange Multipliers

• See the following for proof

• A clear example:

• More intuitive explanation:
• In the SVM problem the Lagrangian is

\[ L_P \equiv \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} \alpha_i \]

\[ \alpha_i \geq 0, \forall i \]

• From the derivatives = 0 we get

\[ w = \sum_{i=1}^{l} \alpha_i y_i x_i, \sum_{i=1}^{l} \alpha_i y_i = 0 \]
Non-linear SVMs

• Datasets that are linearly separable with some noise work out great:

• But what are we going to do if the dataset is just too hard?

• How about... mapping data to a higher-dimensional space:
SOFTMAX CLASSIFIER
OR LOGISTIC REGRESSION
Softmax classifier – cross-entropy loss

• Assumes that the score functions are unnormalized log probabilities.

• Uses cross-entropy loss:

\[ L_i = -\log \left( \frac{e^{f_{yi}}}{\sum_j e^{f_j}} \right) = -f_{yi} + \log \sum_j e^{f_j} \]

Softmax function: \( f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \) (maps arbitrary values to probabilities)

http://cs231n.github.io/
Why take logarithm of probabilities?

• Maps probability space to logarithmic space
• Multiplication becomes addition
  – Multiplication is a very frequent operation with probabilities
• Speed:
  – Addition is more efficient
• Accuracy:
  – Considering loss in representing real numbers, addition is friendlier
• Since log-probability is negative, to work with positive numbers, we usually negate the log-probability
Softmax classifier: One interpretation

- Information theory
  - Cross-entropy between a true distribution and an estimated one:
    \[ H(p, q) = - \sum_x p(x) \log q(x). \]
  - In our case, \( p = [0, \ldots, 1, 0, \ldots 0] \), containing only one 1, at the correct label.
  - Since \( H(p, q) = H(p) + D_{KL}(p||q) \), we are minimizing the Kullback-Leibler divergence.

\[
D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}.
\]

http://cs231n.github.io/
Softmax classifier: Another interpretation

• Probabilistic view

\[ P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}. \]

• In our case, we are minimizing the negative log likelihood.

• Therefore, this corresponds to Maximum Likelihood Estimation (MLE).

http://cs231n.github.io/
Numerical Stability

• Exponentials may become very large. A trick:

\[
\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C \sum_j e^{f_j}} = \frac{e^{f_{y_i} + \log C}}{\sum_j e^{f_j + \log C}}
\]

• Set \( \log C = -\max_j f_j \).
SVM loss vs. cross-entropy loss

• SVM is happy when the classification satisfies the margin
  – Ex: if score values = [10, 9, 9] or [10, -10, -10]
    • SVM loss is happy if the margin is 1
  – SVM is local

• cross-entropy always wants better
  – cross-entropy is global
0-1 Loss

- Minimize the # of cases where the prediction is wrong:

\[ L = \sum_{i} 1(f(x_i; W, b)_{y_i} \neq \hat{y}_i) \]

Or equivalently,

\[ L = \sum_{i} 1(\hat{y}_i f(x_i; W, b)_{y_i} < 0) \]
Absolute Value Loss, Squared Error Loss

\[ L_i = \sum_j |s_j - y_j|^q \]

- \( q = 1 \): absolute value loss
- \( q = 2 \): square error loss.
MORE ON LOSS FUNCTIONS
Visualizing Loss Functions

• If you look at one of the example loss functions:

\[ L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + 1) \]

• Since \( W \) has too many dimensions, this is difficult to plot.

• We can visualize this for one weight direction though, which can give us some intuition about the shape of the function.
  
  – E.g., start from an arbitrary \( W_0 \), choose a direction \( W_1 \) and plot \( L(W_0 + \alpha W_1) \) for different values of \( \alpha \).
Visualizing Loss Functions

• Example:

\[
L_0 = \max(0, w_1^T x_0 - w_0^T x_0 + 1) + \max(0, w_2^T x_0 - w_0^T x_0 + 1)
\]
\[
L_1 = \max(0, w_0^T x_1 - w_1^T x_1 + 1) + \max(0, w_2^T x_1 - w_1^T x_1 + 1)
\]
\[
L_2 = \max(0, w_0^T x_2 - w_2^T x_2 + 1) + \max(0, w_1^T x_2 - w_2^T x_2 + 1)
\]
\[
L = (L_0 + L_1 + L_2)/3
\]

• If we consider just \(w_0\), we have many linear functions in terms of \(w_0\) and loss is a linear combination of them.

http://cs231n.github.io/
Visualizing Loss Functions

- You see that this is a convex function.
  - Nice and easy for optimization
- When you combine many of them in a neural network, it becomes non-convex.

http://cs231n.github.io/
Another approach for visualizing loss functions

- **0-1 loss:**
  \[ L = 1(f(x) \neq y) \]
  or equivalently as:
  \[ L = 1(yf(x) > 0) \]
- **Square loss:**
  \[ L = (f(x) - y)^2 \]
  in binary case:
  \[ L = (1 - yf(x))^2 \]
- **Hinge-loss**
  \[ L = \max(1 - yf(x), 0) \]
- **Logistic loss:**
  \[ L = (\ln 2)^{-1} \ln(1 + e^{-yf(x)}) \]

Various loss functions used in classification. Here \( t = yf(x) \).

Rosacco et al., 2003
SUMMARY OF LOSS FUNCTIONS
Linear Classifier: SVM

Input: $x \in \mathbb{R}^D$

Binary label: $y \in \{-1, +1\}$

Parameters: $w \in \mathbb{R}^D$

Output prediction: $w^T x$

Loss: $L = \frac{1}{2} \| w \|^2 + \lambda \max \left[ 0, 1 - w^T x y \right]$
Linear Classifier: Logistic Regression

Input: \( \mathbf{x} \in \mathbb{R}^D \)

Binary label: \( y \in \{-1, +1\} \)

Parameters: \( \mathbf{w} \in \mathbb{R}^D \)

Output prediction: \( p(y = 1 | \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \)

Loss: \( L = \frac{1}{2} \| \mathbf{w} \|^2 - \lambda \log(p(y | \mathbf{x})) \)
Side Note: Different Losses

Logistic regression:
\[ \sum_{i=1}^{m} \ln(1 + \exp(-y_if(x_i))) \]

Boosting:
\[ \frac{1}{m} \sum_{i} \exp(-y_if(x_i)) = \prod_{t} Z_t \]

0-1 Loss:
\[ \delta(H(x_i) \neq y_i) \]

SVM:
\[ \text{minimize}_{w,b} \quad w.w + C \sum_{j} \xi_j \]
\[ (w.x_j + b) y_j \geq 1 - \xi_j, \quad \forall j \]
\[ \xi_j \geq 0, \quad \forall j \]

Hinge loss:
\[ \xi_j = (1 - f(x_i)y_i)^+ \]

All our new losses approximate 0/1 loss!
Sum up

• 0-1 loss is not differentiable/helpful at training
  – It is used in testing
• Other losses try to cover the “weakness” of 0-1 loss
• Hinge-loss imposes weaker constraint compared to cross-entropy
• For classification: use hinge-loss or cross-entropy loss
• For regression: use squared-error loss, or absolute difference loss
SO, WHAT DO WE DO WITH A LOSS FUNCTION?
Optimization strategies

• We want to find $W$ that minimizes the loss function.
  – Remember that $W$ has lots of dimensions.

• Naïve idea: random search
  – For a number of iterations:
    • Select a $W$ randomly
    • If it leads to better loss than the previous ones, select it.
  – This yields **15.5%** accuracy on CIFAR after 1000 iterations (chance: 10%)

http://cs231n.github.io/
Optimization strategies

• Second idea: random local search
  – Start at an arbitrary position (weight $W$)
  – Select an arbitrary direction $W + \delta W$ and see if it leads to a better loss
    • If yes, move along
  – This leads to 21.4% accuracy after 1000 iterations
  – This is actually a variation of simulated annealing

• A better idea: follow the gradient

http://cs231n.github.io/
A quick reminder on gradients / partial derivatives

– In one dimension: \( \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h} \)

– In practice:
  
  • \( \frac{df[n]}{dn} = \frac{f[n+h]-f[n]}{h} \)
  
  • \( \frac{df[n]}{dn} = \frac{f[n+h]-f[n-h]}{2h} \) (centered difference – works better)

– In many dimensions:
  
  1. Compute gradient numerically with finite differences
     
        – Slow
        
        – Easy
        
        – Approximate

  2. Compute the gradient analytically
     
        – Fast
        
        – Exact
        
        – Error-prone to implement

      • In practice: implement (2) and check against (1) before testing

http://cs231n.github.io/
A quick reminder on gradients / partial derivatives

• If you have a many-variable function, e.g., \( f(x, y) = x + y \), you can take its derivative wrt either \( x \) or \( y \):

\[
\frac{df(x,y)}{dx} = \lim_{h \to 0} \frac{f(x+h,y)-f(x,y)}{h} = 1
\]

\[
\frac{df(x,y)}{dy} = 1
\]

– Similarly, \( \frac{df(x,y)}{dy} = 1 \)

– In fact, we should denote them as follows since they are “partial derivatives” or “gradients on \( x \) or \( y \)”:

• \( \frac{\partial f}{\partial x} \) and \( \frac{\partial f}{\partial y} \)

• Partial derivative tells you the rate of change along a single dimension at a point.

  – E.g., if \( \frac{\partial f}{\partial x} = 1 \), it means that a change of \( x_0 \) in \( x \) leads to the same amount of change in the value of the function.

• Gradient is a vector of partial derivatives:

\[
\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]
\]
A quick reminder on gradients / partial derivatives

- A simple example:
  - \( f(x, y) = \max(x, y) \)
  - \( \nabla f = [1(x \geq y), 1(y \geq x)] \)

- Chaining:
  - What if we have a composition of functions?
  - E.g., \( f(x, y, z) = q(x, y)z \) and \( q(x, y) = x + y \)
    - \( \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} = z \)
    - \( \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = z \)
    - \( \frac{\partial f}{\partial z} = q(x, y) = x + y \)

- Back propagation
  - Local processing to improve the system globally
  - Each gate locally determines to increase or decrease the inputs

http://cs231n.github.io/
Partial derivatives and backprop

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

• Which has many gates:

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

http://cs231n.github.io/
In fact, that was the sigmoid function

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

\[
\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x)
\]

• We will combine all these gates into a single gate and call it a neuron
Intuitive effects

• Commonly used operations
  – Add gate
  – Max gate
  – Multiply gate

• Due to the effect of the multiply gate, when one of the inputs is large, the small input gets the large gradient
  – This has a negative effect in NNs
  – When one of the weights is unusually large, it effects the other weights.
  – Therefore, it is better to normalize the input / weights

http://cs231n.github.io/
Optimization strategies: gradient

• Move along the negative gradient (since we wish to go down)
  – $W = s \frac{\partial L(W)}{\partial W}$
  – $s$: step size

• Gradient tells us the direction
  – Choosing how much to move along that direction is difficult to determine
  – This is also called the learning rate
  – If it is small: too slow to converge
  – If it is big: you may overshoot and skip the minimum

http://cs231n.github.io/
Optimization strategies: gradient

• Take our loss function:

\[ L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + 1) \]

• Its gradient wrt \( w_j \) is:

\[
\frac{\partial L_i}{\partial w_j} = 1(w_j^T x_i - w_{y_i}^T x_i + 1 > 0)x_i
\]

• For the “winning” weight, this is:

\[
\nabla_{w_{y_i}} L_i = -\left( \sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right)x_i
\]

http://cs231n.github.io/
Gradient Descent

• Update the weight:

\[ W_{new} \leftarrow W - s \frac{\partial L(W)}{\partial W} \]

• This computes the gradient after seeing all examples to update the weight.
  – Examples can be on the order of millions or billions

• Alternative:
  – Mini-batch gradient descent: Update the weights after, e.g., 256 examples
  – Stochastic (or online) gradient descent: Update the weights after each example
  – People usually use batches and call it stochastic.
  – Performing an update after one example for 100 examples is more expensive than performing an update at once for 100 examples due to matrix/vector operations

http://cs231n.github.io/
A GENERAL LOOK AT OPTIMIZATION
Mathematical Optimization

Nonlinear Optimization

Convex Optimization

Least-squares

LP
Mathematical Optimization

(mathematical) optimization problem

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq b_i, \quad i = 1, \ldots, m
\end{align*}
\]

- \( x = (x_1, \ldots, x_n) \): optimization variables
- \( f_0 : \mathbb{R}^n \rightarrow \mathbb{R} \): objective function
- \( f_i : \mathbb{R}^n \rightarrow \mathbb{R}, \ i = 1, \ldots, m \): constraint functions

optimal solution \( x^* \) has smallest value of \( f_0 \) among all vectors that satisfy the constraints
Convex Optimization

minimize \( f_0(x) \)
subject to \( f_i(x) \leq b_i, \quad i = 1, \ldots, m \)

- objective and constraint functions are convex:

\[
f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)
\]

if \( \alpha + \beta = 1, \ \alpha \geq 0, \ \beta \geq 0 \)

- includes least-squares problems and linear programs as special cases
Interpretation

- Function’s value is below the line connecting two points
Another interpretation

A differentiable function $f$ is convex if for all $x$ and $y$ we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x),$$

- The function is globally above the tangent at $x$. 
Example convex functions

Some simple convex functions:

- $f(x) = c$
- $f(x) = a^T x$
- $f(x) = x^T A x$ (for $A \succeq 0$)
- $f(x) = \exp(ax)$
- $f(x) = x \log x$ (for $x > 0$)
- $f(x) = \|x\|^2$
- $f(x) = \|x\|_p$
- $f(x) = \max_i \{x_i\}$

Some other notable examples:

- $f(x, y) = \log(e^x + e^y)$
- $f(X) = \log \det X$ (for $X$ positive-definite).
- $f(x, Y) = x^T Y^{-1} x$ (for $Y$ positive-definite)
Operations that conserve convexity

1. Non-negative weighted sum:
   \[ f(x) = \theta_1 f_1(x) + \theta_2 f_2(x). \]

2. Composition with affine mapping:
   \[ g(x) = f(Ax + b). \]

3. Pointwise maximum:
   \[ f(x) = \max_i \{ f_i(x) \}. \]
Show that least-residual problems are convex for any $\ell_p$-norm:

$$f(x) = \|Ax - b\|_p$$

We know that $\| \cdot \|_p$ is a norm, so it follows from (2).

$$\|x + y\|_p \leq \|x\|_p + \|y\|_p$$

Show that SVMs are convex:

$$f(x) = \frac{1}{2}\|x\|^2 + C \sum_{i=1}^{n} \max\{0, 1 - b_i a_i^T x\}.$$  

Know first term is convex, for the other terms use (3) on the two (convex) arguments, then use (1) to put it all together.
Why convex optimization?

• Can’t solve most OPs
  • E.g. NP Hard, even high polynomial time too slow

• Convex OPs
  • (Generally) No analytic solution
  • Efficient algorithms to find (global) solution
  • Interior point methods (basically Iterated Newton) can be used:
    - $\sim[10-100]*\max\{p^3, p^2m, F\}$; F cost eval. obj. and constr. f
  • At worst solve with general IP methods (CVX), faster specialized
Convex Function

- Easy to see why convexity allows for efficient solution
- Just “slide” down the objective function as far as possible and will reach a minimum
Convex vs. Non-convex Ex.

- Convex, min. easy to find

Affine – border case of convexity
Convex vs. Non-convex Ex.

- Non-convex, easy to get stuck in a local min.
- Can’t rely on only local search techniques
Non-convex

- Some non-convex problems highly multi-modal, or NP hard
- Could be forced to search all solutions, or hope stochastic search is successful
- Cannot guarantee best solution, inefficient
- Harder to make performance guarantees with approximate solutions
Analytical solution
• Good algorithms and software
• High accuracy and high reliability
• Time complexity: $C \cdot n^2 k$

A mature technology!

minimize $\|Ax - b\|_2^2$
minimize $c^T x$
subject to $a_i^T x \leq b_i$
$i = 1, \ldots, m$

- No analytical solution
- Algorithms and software
- Reliable and efficient
- Time complexity: $C \cdot n^2 m$

Also a mature technology!
Mathematical Optimization

Nonlinear Optimization

Convex Optimization

Least-squares

LP

- No analytical solution
- Algorithms and software
- Reliable and efficient
- Time complexity (roughly)

\[
\propto \max\{ n^3, n^2m, F \}
\]

\( F \) is cost of evaluating \( f_i \)'s and their first and second derivatives

Almost a mature technology!
Far from a technology! (something to avoid)

- Sadly, no effective methods to solve
- Only approaches with some compromise
- Local optimization: "more art than technology"
- Global optimization: greatly compromised efficiency
- Help from convex optimization
  1) Initialization 2) Heuristics 3) Bounds
Why Study Convex Optimization

With only a bit of exaggeration, we can say that, if you formulate a practical problem as a convex optimization problem, then you have solved the original problem. If not, there is little chance you can solve it.

-- Section 1.3.2, p8, Convex Optimization
Recognizing Convex Optimization Problems

- often difficult to recognize
- many tricks for transforming problems into convex form
- surprisingly many problems can be solved via convex optimization
A least-squares problem is an optimization problem with no constraints (i.e., $m = 0$) and an objective which is a sum of squares of terms of the form $a_i^T x - b_i$:

$$
\text{minimize} \quad f_0(x) = \|Ax - b\|_2^2 = \sum_{i=1}^{k} (a_i^T x - b_i)^2. \quad (1.4)
$$

Here $A \in \mathbb{R}^{k \times n}$ (with $k \geq n$), $a_i^T$ are the rows of $A$, and the vector $x \in \mathbb{R}^n$ is the optimization variable.
Analytical Solution of Least-squares

\[ \text{minimize } f_0(x) = \| Ax - b \|_2^2 = \sum_{i=1}^{k} (a_i^T x - b_i)^2. \]

• Set the derivative to zero:
  
  \[ -\frac{df_0(x)}{dx} = 0 \]
  
  \[ - (A^T A) 2x - 2Ab = 0 \]
  
  \[ - (A^T A)x = Ab \]

• Solve this system of linear equations
Linear Programming (LP)

Another important class of optimization problems is *linear programming*, in which the objective and all constraint functions are linear:

\[
\begin{align*}
& \text{minimize} \quad c^T x \\
& \text{subject to} \quad a_i^T x \leq b_i, \quad i = 1, \ldots, m.
\end{align*}
\]  

Here the vectors \( c, a_1, \ldots, a_m \in \mathbb{R}^n \) and scalars \( b_1, \ldots, b_m \in \mathbb{R} \) are problem parameters that specify the objective and constraint functions.

- no analytical formula for solution
- reliable and efficient algorithms and software
To sum up

• We introduced some important concepts in machine learning and optimization
• We introduced popular machine learning methods
• We talked about loss functions and how we can optimize them using gradient descent