CEng 783 – Deep Learning

Week 2 – Machine Learning
Background and Basics

Fall 2017

Emre Akbaş
Reminders

- **Hw #1 is due this Monday, Oct 16, 23:55**
  - Submit via ODTUClass
- **Project proposals are due in ~3 weeks (Oct 30).**
  - Will be done in pairs.
  - Use the forum on ODTUClass to find a partner or discuss ideas.
  - Email me with ideas.
- **Hw #2 will be announced Oct 27.**
Overview of today's lecture

Supervised Learning

Types:
- linear
- non-linear

Model

Loss function

Training (or learning, fitting)

Numerical Optimization

Regularization

Hyper-parameters and cross-validation

Capacity

0-1

L2

Hinge

Log-loss

Softmax

Maximum likelihood

Maximum a-posteriori

Underfitting, overfitting, generalization

Convexity

Gradient descent & variants
Example: Linear SVM

Supervised Learning

Model
- Types: linear, non-linear
- Low capacity

Loss function
- Numerical Optimization
- Convex

Training (or learning, fitting)
- Regularization: maximum margin
- Underfitting, overfitting, generalization

Loss function
- 0-1
- L2
- Hinge
- Log-loss
- Softmax

Hyper-parameters and cross-validation

Numerical Optimization
- Gradient descent & custom optimization algorithms
Example: ConvNet for image classification

Supervised Learning

Model
- Types: linear, non-linear

Loss function
- Very high capacity

Training (or learning, fitting)

Loss function
- Numerical Optimization
- Softmax + cross-entropy

Regularization: Dropout
- Non-convex
- Stochastic gradient descent

Hyperparameters and cross-validation

0-1, L2, Hinge, Log-loss
Underfitting, overfitting, generalization
Supervised Learning

Model
- Types: linear, non-linear
- Capacity

Loss function
- Numerical Optimization
- 0-1, L2, Hinge, Log-loss, Softmax

Training (or learning, fitting)
- Regularization
- Convexity
- Gradient descent & variants

Hyper-parameters and cross-validation
- Underfitting, overfitting, generalization

Hyper-parameters and cross-validation
- Underfitting, overfitting, generalization
Supervised Learning

Given a dataset $S = \{(x_i, y_i)\}_{i=1}^{N}$ where

$x_i \in \mathbb{R}^D$ and $y_i \in \{-1, +1\} \quad \forall i = 1 \ldots N$

The task is to “learn” a function $f(\cdot)$ that predicts (probably correctly) the label of a previously unseen example:

$\hat{y} = f(x)$
Supervised Learning

The purpose of learning $f()$ is to apply it to \textit{unseen} data, usually called the “test set.”

$S$ is called the training set.

The lower the error rate on the test set $\rightarrow$ the more successful a learning algorithm is.
Supervised Learning

Model
- Types: linear, non-linear
- Capacity

Loss function

Training (or learning, fitting)

Numerical Optimization
- Convexity
- Gradient descent & variants

Regularization

Hyper-parameters and cross-validation

Underfitting, overfitting, generalization

Loss functions:
- 0-1
- L2
- Hinge
- Log-loss
- Softmax

Regularization

Numerical Optimization

Convexity

Hyper-parameters and cross-validation

Underfitting, overfitting, generalization
The model

Machine learning models are usually described in terms of how they **process** a given example. This process is called the **model**.

For example:

\[ f(x; w) = w^T x \]

is a linear model.

Linear SVM's classification model:

\[ f(x; w) = \text{sign}(w^T x) \]

- **Linear or non-linear?**
- **Strictly speaking, it is a non-linear function because of the sign().**
The model

How about \( f(x; w) = w_1 x + w_2 x^2 + w_3 x^3 \) ?

- Linear w.r.t. model parameters, \( w \)
- But, implements a non-linear decision boundary, i.e. non-linear w.r.t. to data

What about the nearest neighbor method, i.e. \( f(x) = y_{i^*} \)
where \( i^* = \arg \max_j \| x_j - x \|^2 \) where \( x_j \in S \)

No parameters (non-parametric). Implements non-linear decision boundary.
Model capacity

Informally, capacity is the ability of the model to learn complicated functions.

- e.g. in supervised classification, higher the capacity → more complicated decision boundaries

Statistical learning theory provides quantitative measure for capacity:

- The best known: Vapnik-Chervonenkis (VC) dimension
VC dimension

The VC dimension of a model $M$ is the largest number $n$ such that the model can correctly label arbitrary configurations of $n$ points.

[Figure from https://en.wikipedia.org/wiki/VC_dimension]

For binary classification.
Model capacity

True or false?:
If more capacity means that the model can learn more complicated functions, then we should always look for models with higher capacity in order to perform better (e.g. well in supervised learning).

False because what matters for success is the generalization error.
Typical relationship between capacity and error

[Figure 5.3 from Goodfellow et al. 2016]
Capacity vs Error: linear regression example

[Figure 5.2 from Goodfellow et al. 2016]
Supervised Learning

Model
- Types: linear, non-linear
- Capacity

Loss function
- 0-1
- L2
- Hinge
- Log-loss
- Softmax

Training (or learning, fitting)

Numerical Optimization

Regularization

Hyperparameters and cross-validation

Capacity
- Underfitting, overfitting, generalization

Convexity
- Gradient descent & variants
The loss function

- Measures how well our prediction is:

  \[ \text{Loss}(y, \hat{y}) \]

  The lower the better.

Relation to cost (or objective) function?
Let’s consider an example.
Suppose our classification model is

\[ f(x; w) = \text{sign}(w^T x) \]

We want \( f() \) to perform “well” on both the train and test sets.

That is, we want to minimize:

\[ L(w) = \sum_{i=1}^{N} I_{f(x_i; w) \neq y_i} \] where \( I \) is the indicator func.

This is the 0-1 loss function.
The 0-1 loss

\[ L(w) = \sum_{i=1}^{N} I_{f(x_i; w) \neq y_i} \]

can also be written as

\[ L(w) = \sum_{i=1}^{N} I_{z_i \leq 0} \]

where \[ z_i = y_i f(x_i; w) \].
To perform well on S, we should search for a $w$ that minimizes this loss on the training set.

$$L_{01}(w) = \sum_{i=1}^{N} I_{y_i \text{sign}(w^T x_i) \leq 0}$$

However, we do not use this loss function in practice.

**WHY?**
L₂ loss function

\[ L_2(w) = \sum_{i=1}^{N} (1-z_i)^2 \quad \text{where} \quad z_i = y_i f(x_i; w). \]

More commonly used in regression as follows

\[ L_2(w) = \sum_{i=1}^{N} (y_i - f(x_i; w))^2 \]

where, for linear regression, \( y_i \in \mathbb{R} \) and \( f(x_i; w) = w^T x_i \).
Hinge loss

\[ L_{\text{Hinge}}(w) = \sum_{i=1}^{N} \max(0, 1 - z_i) \quad \text{where} \quad z_i = y_i f(x_i; w). \]

Support vector machine and many maximum margin based models use the hinge loss.

Also, rectified linear units (ReLU) use a form the hinge function.

When \( z_i \) is linear in model parameters, then Hinge loss is \textbf{convex} in model parameters.
Logistic loss

\[ L_{\text{log}}(w) = \sum_{i=1}^{N} \ln\left(1 + e^{-z_i}\right) \quad \text{where} \quad z_i = y_i f(x_i; w). \]

Similar to the hinge loss but it assigns a non-zero penalty to all examples.

Also convex.
Softmax

- Softmax is not a loss function!
- But a normalization function used in multiclass classification.
- **Cross-entropy** is the loss function that is usually used with softmax.
- For binary classification, cross-entropy is the same thing with logistic loss.
Softmax & cross-entropy example

Consider a three class classification problem.

A training example is given as such a pair:

\[(x, y)\] where \(x \in \mathbb{R}^D, y \in \mathbb{R}^3\) and \(y_c \in [0,1]\) for \(c = 1, 2, 3\)

Let each class have a one-vs-rest linear model:

\[f_c(x) = w_c^T x \text{ for } c = 1, 2, 3\]
Softmax & cross-entropy example

Consider a three class classification problem.

A training example is given as such a pair:

\[(x, y) \text{ where } x \in \mathbb{R}^D, y \in \mathbb{R}^3 \text{ and } y_c \in [0,1] \text{ for } c=1,2,3\]

Let each class have a one-vs-

\[f_c(x) = w_c^T x \text{ for } c=1,2,3\]

- Be careful about the labels
- No longer -1, +1
- Now probabilities in [0,1]
- One hot vector
Softmax & cross-entropy example

We convert these classification scores to probabilities using the softmax normalization:

\[ q_c(x) = \frac{e^{f_c(x)}}{\sum_{c=1}^{3} e^{f_c(x)}} \]

Now, \( q_c(x) \in (0,1] \)

Then, cross-entropy loss for example \((x,y)\) is

\[ H((x,y)) = -\sum_{c=1}^{3} y_c \log q_c(x) \]
Cross-entropy for two classes?

Convert classification scores to probability using the sigmoid function:

$$\sigma(x; w) = \frac{1}{1 + e^{-w^T x}}$$

$$q_1 = \sigma(x; w), q_2 = 1 - q_1$$

$$H((x, y)) = -\sum_{c=1}^{2} y_c \log q_c(x) = -y_1 \log q_1 - (1 - y_1) \log (1 - q_1)$$

For $$y = [1, 0]^T$$, $$H((x, y)) = -\log q_1 = \log (1 + e^{-w^T x})$$
Loss functions in one picture

Try to name the functions
Types: linear, non-linear

Model

Loss function

Training (or learning, fitting)

Supervised Learning

Numerical Optimization

Capacity

Regularization

Numerical Optimization

Hyperparameters and cross-validation

Convexity

Gradient descent & variants

0-1
L2
Hinge
Log-loss
Softmax

Underfitting, overfitting, generalization
Training

Write down the cost (or objective or loss) function on the training set and then minimize.

\[ \theta^* = \arg \min_{\theta} L(\theta, S) \]

Is the cost function convex?
Why convex cost functions are good?

- We know how to minimize them reliably and efficiently.
- No need to provide a “good” initial point (initialization).
- When you minimize a convex cost function, you are sure that you did your best. (i.e. a local minima is the global minima)
  - Compare this to a non-convex optimization problem.
Convex function:

Convexity
How to check convexity?

• Often difficult

• Analytically:
  – Verify the definition
  – A practical test for twice-differentiable functions:
    • If the second-derivative (or the Hessian) is positive (positive semidefinite), then convex
    – Show that the function is obtained from convex functions and convexity preserving operations

• Numerically
Examples on $\mathbb{R}$

convex:

- affine: $ax + b$ on $\mathbb{R}$, for any $a, b \in \mathbb{R}$
- exponential: $e^{ax}$, for any $a \in \mathbb{R}$
- powers: $x^\alpha$ on $\mathbb{R}_{++}$, for $\alpha \geq 1$ or $\alpha \leq 0$
- powers of absolute value: $|x|^p$ on $\mathbb{R}$, for $p \geq 1$
- negative entropy: $x \log x$ on $\mathbb{R}_{++}$

concave:

- affine: $ax + b$ on $\mathbb{R}$, for any $a, b \in \mathbb{R}$
- powers: $x^\alpha$ on $\mathbb{R}_{++}$, for $0 \leq \alpha \leq 1$
- logarithm: $\log x$ on $\mathbb{R}_{++}$

Examples on $\mathbb{R}^n$ and $\mathbb{R}^{m \times n}$

affine functions are convex and concave; all norms are convex

examples on $\mathbb{R}^n$

- affine function $f(x) = a^T x + b$
- norms: $\|x\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$ for $p \geq 1$; $\|x\|_\infty = \max_k |x_k|$

examples on $\mathbb{R}^{m \times n}$ ($m \times n$ matrices)

- affine function

$$f(X) = \text{tr}(A^T X) + b = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} X_{ij} + b$$

- spectral (maximum singular value) norm

$$f(X) = \|X\|_2 = \sigma_{\max}(X) = (\lambda_{\max}(X^T X))^{1/2}$$

[From Lieven Vandenberghe’s lecture notes: http://www.seas.ucla.edu/~vandenbe/ee236b/lectures/functions.pdf]
Operations that preserve convexity

- nonnegative weighted sum
- composition with affine function
- pointwise maximum and supremum
- composition
- minimization
- perspective

Exercise

Prove log loss function is convex.
Training: numerical optimization

We will only look at gradient descent and its variants.

Huge literature on numerical optimization.
Mathematical Optimization

Nonlinear Optimization

Convex Optimization

Least-squares

LP

[Slide by Rong Jin from http://www.cse.msu.edu/~rongjin/adv_ml/slides/cvxchap1.ppt]
• Analytical solution
• Good algorithms and software
• High accuracy and high reliability
• Time complexity:

\[ C n^2 k \text{ where } A \in \mathbb{R}^{k \times n} \]

A mature technology!

\[ \text{minimize } \|Ax - b\|^2_2 \]
• No analytical solution
• Algorithms and software
• Reliable and efficient
• Time complexity: $C \cdot n^2 m$

minimize $c^T x$
subject to $a_i^T x \leq b_i$
$i = 1, \ldots, m$

Also a mature technology!
Mathematical Optimization

Convex Optimization

Nonlinear Optimization

Least-squares

LP

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

- 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!

[Slide by Rong Jin from http://www.cse.msu.edu/~rongjin/adv_ml/slides/cvxchap1.ppt]
Mathematical Optimization

Nonlinear Optimization

Convex Optimization

Least-squares

LP

- 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

Far from a technology! (something to avoid)
Gradient descent

• The goal is to minimize the cost function $L$.
• We compute the gradient of $L$ with respect to model parameters $w$.
• The gradient points in the direction of the greatest rate of increase of the function.
• Then, take a step in the opposite direction:

$$w^{\text{new}} = w^{\text{old}} - \eta \nabla_w L(w)$$
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

[Slide from Course cs231n of Standard University]
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 \( y \):
  \[
  \frac{df(x,y)}{dx} = \lim_{h \to 0} \frac{f(x+h,y)-f(x,y)}{h} = 1
  \]
- 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:
  \[
  \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]
  \]

[Slide from Course cs231n of Standard University]
Chain rule

This is the basis for backpropagation.

Consider \( z \) to be a function of the variable \( y \), which is itself a function of \( x \) (\( y \) and \( z \) are therefore dependent variables), and so, \( z \) becomes a function of \( x \) as well:

\[
\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}
\]
An example

Let's minimize the following cost function:

\[ L(w) = \sum_{i=1}^{N} \max(0, 1 - y_i w^T x_i) \]

Question: suppose \( f \) and \( g \) are both differentiable. Is \( h = \max(f, g) \) differentiable?

No. If we choose \( f(x) = x \) and \( g(x) = -x \), then \( h \) becomes the absolute value function. And this is not differentiable.
Subgradient

Max() is not differentiable, so no gradient.

But, we can use its subgradient (or subderivative).

Subderivative generalizes the derivative to non-differentiable functions such as max().
Subgradient

Rigorously, a subderivative of a function $f:I \rightarrow \mathbb{R}$ at a point $x_0$ in the open interval $I$ is a real number $c$ such that

$$f(x) - f(x_0) \geq c(x - x_0)$$

for all $x$ in $I$.

A convex function (blue) and "subtangent lines" at $x_0$ (red). [Source: https://en.wikipedia.org/wiki/Subderivative]
Back to our example

\[
L(w) = \sum_{i=1}^{N} \max(0, 1 - y_i w^T x_i)
\]

Now let's write the sub-gradient of hinge function:

\[
\nabla_w \max(0, g(w)) = \begin{cases} 
  g'(w), & \text{if } g(w) > 0 \\
  0, & \text{otherwise}
\end{cases}
\]
Let $h_i(w) = \max(0, 1 - y_i w^T x_i)$ then $L(w) = \sum_{i=1}^{N} h(x_i; w)$

$$\nabla_w L(w) = \sum_{i=1}^{N} \nabla_w h(x_i; w)$$

$$\nabla_w h(x_i; w) = \begin{cases} -y_i x_i, & \text{if } 1 - y_i w^T x_i > 0 \\ 0, & \text{otherwise} \end{cases}$$
Gradient descent, learning rate

\[ w^{new} = w^{old} - \eta \nabla_w L(w) \]

How to set the learning rate?

Numerical optimization literature provides

- Many answers and algorithms for this problem. e.g. Newton's method, BFGS, L-BFGS, line search, Wolfe conditions, etc.

- Theoretical results on convergence rates and guarantees subject to certain conditions.
Gradient descent, learning rate

In this class, about setting the learning rate, we will study the most basic methods and some adaptive ones, as well.

The most basic method: choose a small, constant learning rate.

Would it converge?
Stochastic gradient descent

Stochastic estimation of the gradient.

Instead of using the whole dataset to compute the gradient, use just a single example or a small number of examples.

Choose those examples randomly.
Stochastic gradient descent

When the learning rates decrease with an appropriate rate, stochastic gradient descent converges almost surely to a global minimum when the objective function is convex or pseudoconvex, and otherwise converges almost surely to a local minimum. [Bottou, Léon (1998). "Online Algorithms and Stochastic Approximations". Online Learning and Neural Networks.]

Works very well in practice.
Stochastic gradient descent

\[ L(w) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i w^T x_i) = \frac{1}{N} \sum_{i=1}^{N} h(x_i; w) \]

\[ \nabla_w L(w) \approx \nabla_w h(x_i; w) \] where \( i \) is randomly chosen.

Or,

\[ \nabla_w L(w) \approx \frac{1}{m} \sum_{i \in Idx} \nabla_w h(x_i; w) \]

where \( Idx \) is a set of randomly chosen. \( m \) indices in \([1, N]\)
Stochastic gradient descent

A stochastic gradient descent version of linear SVM, known as PEGASOS, works very well.

Obtains on-par results with a full-blown linear SVM implementation with 10x speedup.
Supervised Learning

Model
- Types: linear, non-linear
- Capacity

Loss function
- 0-1
- L2
- Hinge
- Log-loss
- Softmax

Training (or learning, fitting)
- Convexity
- Gradient descent & variants

Numerical Optimization

Regularization

Hyper-parameters and cross-validation
- Underfitting, overfitting, generalization
Regularization

- A way of controlling the capacity of a learning model.
- Intuitively, favors simpler (i.e. lower capacity) models.
- Occam's razor principle (circa 1300):
  - Among competing hypotheses, the one with the fewest assumptions should be selected.
  - Or, if two different model explains the data equally well, choose the simpler model.
Regularization

Intuitive motivation

For linearly separable data, there are infinitely many equally correct decision boundaries.

Choose the “best” one!
Regularization: Linear SVM example

\[ L(w) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \max(0, 1 - y_i w^T x_i) \]

Empirical loss (or training error)

Regularization

[Figure from https://en.wikipedia.org/wiki/Support_vector_machine]
Regularization

Most commonly used regularizers:

L2 norm and L1 norm

L1 norm enforces sparsity

We'll study dropout and data augmentation.
Types: linear, non-linear

Model

Loss function

Training (or learning, fitting)

Numerical Optimization

Regularization

Hyper-parameters and cross-validation

Capacity

0-1, L2, Hinge, Log-loss, Softmax

Underfitting, overfitting, generalization

Convexity

Gradient descent & variants
This is in your Hw #1, so we are skipping.

But a question: suppose you are asked to train a linear SVM on a given dataset. What exactly do you do?
- Two datasets (breast_cancer and madelon) from the UCI machine learning repo
- 5-fold CV, Linear SVM (libsvm)
Overview of today's lecture

Supervised Learning

- Model
- Loss function
- Training (or learning, fitting)
- Regularization
- Hyper-parameters and cross-validation

Types: linear, non-linear
- Capacity

0-1
- Hinge
- Log-loss
- Softmax

Numerical Optimization
- Convexity
- Gradient descent & variants

Loss function
- Maximum likelihood
- Maximum a-posteriori

Capacity
- Underfitting, overfitting, generalization

Maximum likelihood

Maximum a-posteriori
Maximum likelihood estimation

Given a dataset \( S = \{ x_1, x_2, \ldots, x_N \} \) and a model with parameter vector \( \theta \):

\[
\theta_{\text{ML}} = \arg \max_{\theta} p(S; \theta) \quad \text{where} \quad p(S; \theta) = \prod_{i=1}^{N} p(x_i; \theta)
\]

(using the i.i.d. assumption of the data generating distribution.)

\[
\theta_{\text{ML}} = \arg \max_{\theta} \sum_{i=1}^{N} \log p(x_i; \theta) = \arg \min_{\theta} \sum_{i=1}^{N} -\log p(x_i; \theta)
\]

Maximum likelihood = minimizing negative log-likelihood
= minimizing cross-entropy
Maximum a-posteriori estimation

Given a dataset $S = \{x_1, x_2, \ldots, x_N\}$ and a model with parameter vector $\theta$:

$$\theta_{\text{MAP}} = \arg \max_{\theta} p(\theta | S) = \arg \max_{\theta} \log p(S | \theta) + \log p(\theta)$$
The difference

\[ \theta_{\text{ML}} = \arg \max_\theta \sum_{i=1}^{N} \log p(x_i; \theta) \]

\[ \theta_{\text{MAP}} = \arg \max_\theta \sum_{i=1}^{N} \log p(x_i; \theta) + \log p(\theta) \]
References