This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes
- Approximate Inference
  - Sampling Approaches
  - MCMC
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation
  - CNNs, RNNs, ResNets, etc.

Recap: Generalized Linear Discriminants

- Extension with non-linear basis functions
  - Transform vector $x$ with $M$ nonlinear basis functions $\phi_j(x)$:
    \[ y_k(x) = g \left( \sum_{j=1}^{M} w_k j \phi_j(x) + w_k 0 \right) \]
    - Basis functions $\phi_j(x)$ allow non-linear decision boundaries.
    - Activation function $g(\cdot)$ bounds the influence of outliers.
    - Disadvantage: minimization no longer in closed form.
- Notation
  \[ y_k(x) = g \left( \sum_{j=0}^{M} w_k j \phi_j(x) \right) \quad \text{with} \quad \phi_0(x) = 1 \]

Recap: Gradient Descent

- Iterative minimization
  - Start with an initial guess for the parameter values $w^{(0)}$.
  - Move towards a (local) minimum by following the gradient.
- Basic strategies
  - "Batch learning" $w^{(r+1)}_{kj} = w^{(r)}_{kj} - \eta \frac{\partial E(w)}{\partial w_{kj}} \bigg|_{w^{(r)}}$
  - "Sequential updating" $w^{(r+1)}_{kj} = w^{(r)}_{kj} - \eta \frac{\partial E_n(w)}{\partial w_{kj}} \bigg|_{w^{(r)}}$
  where $E(w) = \sum_{n=1}^{N} E_n(w)$. 

Talk Announcement

- Yann LeCun (NYU & FaceBook AI)
  28.11. 15:00-16:30h, SuperC 6th floor (Ford Saal)
  The rapid progress of AI in the last few years are largely the result of advances in deep learning and neural nets, combined with the availability of large datasets and fast GPUs. We now have systems that can recognize images with an accuracy that rivals that of humans. This will lead to revolutions in several domains such as autonomous transportation and medical image analysis. But all of these systems currently use supervised learning in which the machine is trained with inputs labeled by humans. The challenge of the next several years is to let machines learn from raw, unlabeled data, such as video or text. This is known as predictive (or unsupervised) learning. Intelligent systems today do not possess “common sense”, which humans and animals acquire by observing the world, by acting in it, and by understanding the physical constraints of it. I will argue that the ability of machines to learn predictive models of the world is a key component of that will enable significant progress in AI. The main technical difficulty is that the world is only partially predictable. A general formulation of unsupervised learning that deals with partial predictability will be presented. The formulation connects many well-known approaches to unsupervised learning, as well as new and exciting ones such as adversarial training.

- No lecture next Monday - go see the talk!
Consider models of the form
\[ p(C_i | \phi) = y(\phi) = \sigma(w^T \phi) \]
with
\[ p(C_2 | \phi) = 1 - p(C_1 | \phi) \]
This model is called logistic regression.

Properties
- Probabilistic interpretation
- But discriminative method; only focus on decision hyperplane
- Advantageous for high-dimensional spaces, requires less parameters than explicitly modeling \( p(\phi | C_i) \) and \( p(C_i) \).

Softmax Regression

- Multi-class generalization of logistic regression
  - In logistic regression, we assumed binary labels \( t_n \in \{0, 1\} \)
  - Softmax generalizes this to \( K \) values in 1-of-\( K \) notation.

\[
y(x; w) = \begin{bmatrix} P(y = 1 | x; w) \\ \vdots \\ P(y = K | x; w) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(w_j^T x)} \begin{bmatrix} \exp(w_1^T x) \\ \vdots \\ \exp(w_K^T x) \end{bmatrix}
\]
- This uses the softmax function
  \[ \frac{\exp(a_k)}{\sum_j \exp(a_j)} \]
- Note: the resulting distribution is normalized.

Softmax Regression Cost Function

- Logistic regression
  - Alternative way of writing the cost function

\[
E(w) = - \sum_{n=1}^{N} \left( t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right)
\]
- Softmax regression
  - Generalization to \( K \) classes using indicator functions.

\[
E(w) = - \sum_{n=1}^{N} \sum_{k=1}^{K} \left[ \{ t_n = k \} \ln \left( \frac{\exp(w_k^T x)}{\sum_{j=1}^{K} \exp(w_j^T x)} \right) \right]
\]

\[
\nabla_w E(w) = - \sum_{n=1}^{N} \left[ \{ t_n = k \} \ln P(y_n = k | x_n; w) \right]
\]

Optimization

- Again, no closed-form solution is available
  - Resort again to Gradient Descent
  - Gradient

\[
\nabla_w E(w) = - \sum_{n=1}^{N} \left[ \{ t_n = k \} \ln P(y_n = k | x_n; w) \right]
\]
- Note
  - \( \nabla_w E(w) \) is itself a vector of partial derivatives for the different components of \( w \).
  - We can now plug this into a standard optimization package.

A Note on Error Functions

\[ t_n \in \{-1, 1\} \]

\[
E(z_n) \quad \text{Ideal misclassification error}
\]

- Not differentiable!
- Ideal misclassification error function (black)
  - This is what we want to approximate,
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.
  - We cannot minimize it by gradient descent.
A Note on Error Functions

- Squared error used in Least-Squares Classification
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes "too correct" data points
    - Generally does not lead to good classifiers.

- Hinge loss enforces sparsity
  - Currently most efficient:
    \[ b \]
    "Hinge error"
  - Favors sparse solutions, not sensitive to outliers.
  - But no closed-form solution, requires iterative estimation.

- Squared error used in Least-Squares Classification
  - Linearly increasing error for misclassified points (\( z_i > 1 \)).
  - Unconstrained optimization, but non-differentiable function.
  - Solve, e.g. by subgradient descent
    - Currently most efficient: stochastic gradient descent

Side Note: Support Vector Machine (SVM)

- Basic idea
  - The SVM tries to find a classifier which maximizes the margin between pos. and neg. data points.
  - Up to now: consider linear classifiers
    \[ w^T x + b = 0 \]

- Formulation as a convex optimization problem
  - Find the hyperplane satisfying
    \[ \text{argmin} \quad \frac{1}{2} \| w \|^2 \]
    under the constraints
    \[ t_n (w^T x_n + b) \geq 1 \quad \forall n \]
  - based on training data points \( x_n \) and target values \( t_n \in \{-1, 1\} \).

SVM Error Function (Loss Function)

- Squared error used in SVMs
  - Zero error for points outside the margin (\( z_i > 1 \)).
  - Linearly increasing error for misclassified points (\( z_i < 1 \)).
  - Not differentiable around \( z_i = 1 \) → Cannot be optimized directly.

- "Hinge error" used in SVMs
  - Minimizer of this error is given by posterior class probabilities.
  - Concave error function, unique minimum exists.
  - Leads to sparse solutions, not sensitive to outliers.

SVM - Analysis

- Traditional soft-margin formulation
  \[ \min_{w \in \mathbb{R}^D, b, \xi_n \in \mathbb{R}^+} \frac{1}{2} \| w \|^2 + C \sum_{n=1}^{N} \xi_n \]
  subject to the constraints
  \[ t_n y(x_n) \geq 1 - \xi_n \]
  \( L_2 \) regularizer
  "Hinge loss"
  where \( [x]_+ := \max(0,x) \).

- Different way of looking at it
  - We can reformulate the constraints into the objective function.

SVM - Discussion

- SVM optimization function
  \[ \min_{w \in \mathbb{R}^D} \frac{1}{2} \| w \|^2 + C \sum_{n=1}^{N} [1 - t_n y(x_n)]_+ \]
  \( L_2 \) regularizer
  "Hinge loss"
  \[ + \]

- Hinge loss enforces sparsity
  - Only a subset of training data points actually influences the decision boundary.
  - This is different from sparsity obtained through the regularizer!
  - Unconstrained optimization, but non-differentiable function.
  - Solve, e.g. by subgradient descent
    - Currently most efficient: stochastic gradient descent
Topics of This Lecture

- A Brief History of Neural Networks
- Perceptrons
  - Definition
  - Loss functions
  - Regularization
  - Limits
- Multi-Layer Perceptrons
  - Definition
  - Learning

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron
- And a cool learning algorithm: “Perceptron Learning”
- Hardware implementation “Mark I Perceptron” for 20x20 pixel image analysis

The New York Times
“The embryo of an electronic computer that […] will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”

OMG! They work like the human brain!

Oh no! Killer robots will achieve world domination!

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron
1969 Minsky & Papert
1980s Resurgence of Neural Networks
- Some notable successes with multi-layer perceptrons.
- Backpropagation learning algorithm

1995+ Interest shifts to other learning methods
- Notably Support Vector Machines
- Machine Learning becomes a discipline of its own.

OMG! They work like the human brain!

Oh no! Killer robots will achieve world domination!

I can do science, me!
A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron
1969 Minsky & Papert
1980s Resurgence of Neural Networks
1995+ Interest shifts to other learning methods
   - Notably Support Vector Machines
   - Machine Learning becomes a discipline of its own.
   - The general public and the press still love Neural Networks.

So, you’re using Neural Networks?

I’m doing Machine Learning.

ACTUALLY...

2005+
   - Gradual progress

   2012 Breakthrough results
     - ImageNet Large Scale Visual Recognition Challenge
     - A ConvNet halves the error rate of dedicated vision approaches.
     - Deep Learning is widely adopted.

Are you using Neural Networks?

Come on. Get real!

Topics of This Lecture

- A Short History of Neural Networks
- Perceptrons
  - Definition
  - Loss functions
  - Regularization
  - Limits
- Multi-Layer Perceptrons
  - Definition
  - Learning

Extension: Multi-Class Networks

- One output node per class

Output layer
\[ y_1(x), y_2(x), \ldots, y_k(x) \]

Weights
\[ W_{i,j} \]

Input layer
\[ x_0, x_1, x_2, \ldots, x_d \]

Outputs
  - Linear outputs
  - Logistic outputs

\[ y_k(x) = \frac{1}{1 + e^{-W_{i,j} x_i}} \]

\[ y_k(x) = \sigma \left( \sum_{i=0}^{d} W_{i,j} x_i \right) \]

Can be used to do multidimensional linear regression or multiclass classification.

Perceptrons (Rosenblatt 1957)

- Standard Perceptron

Input layer
\[ x_0, x_1, x_2, \ldots, x_d \]

Weights
\[ w_0, w_1, w_2, \ldots, w_d \]

Output layer
\[ y(x) \]

- Input Layer
  - Hand-designed features based on common sense

- Outputs
  - Linear outputs
  - Logistic outputs

\[ y(x) = w^T x + w_0 \]

\[ y(x) = \sigma(w^T x + w_0) \]

- Learning = Determining the weights \( w \)
Extension: Non-Linear Basis Functions

- Straightforward generalization
  \[ y_0(x) = \sum_{i=0}^{d} W_k \phi(x_i) \]
  \[ y(x) = \sigma \left( \sum_{i=0}^{d} W_k \phi(x_i) \right) \]

- Outputs
  - Linear outputs
  - Logistic outputs

Perceptron Learning

- Very simple algorithm
- Process the training cases in some permutation
  - If the output unit is correct, leave the weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a one, subtract the input vector from the weight vector.
- This is guaranteed to converge to a correct solution if such a solution exists.

Loss Functions

- We can now also apply other loss functions
  - L2 loss
    \[ L(t, y(x)) = \sum_{n} (y(x_n) - t_n)^2 \]  \( \Rightarrow \) Least-squares regression
  - L1 loss:
    \[ L(t, y(x)) = \sum_{n} |y(x_n) - t_n| \]  \( \Rightarrow \) Median regression
  - Cross-entropy loss
    \[ L(t, y(x)) = -\sum_{n} \{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \} \]
  - Hinge loss
    \[ L(t, y(x)) = \sum_{n} [1 - t_n y(x_n)]_+ \]  \( \Rightarrow \) SVM classification
  - Softmax loss
    \[ L(t, y(x)) = -\sum_{n} \sum_k \{ t_n \ln p_k + (1 - t_n) \ln \frac{exp(p_k(x_n, k))}{\sum_k exp(p_k(x_n, k))} \} \]  \( \Rightarrow \) Multi-class probabilistic classification
Regularization

- In addition, we can apply regularizers
  - E.g., an L2 regularizer
    \[ E(w) = \sum L(t_n, y(x_n; w)) + \lambda \|w\|^2 \]
  - This is known as weight decay in Neural Networks.
- We can also apply other regularizers, e.g. L1 \(\Rightarrow\) sparsity
- Since Neural Networks often have many parameters, regularization becomes very important in practice.
- We will see more complex regularization techniques later on...

Limitations of Perceptrons

- What makes the task difficult?
  - Perceptrons with fixed, hand-coded input features can model any separable function perfectly...
  - ...given the right input features.
  - For some tasks this requires an exponential number of input features.
    - E.g., by enumerating all possible binary input vectors as separate feature units (similar to a look-up table).
    - But this approach won’t generalize to unseen test cases!
  - It is the feature design that solves the task!
  - Once the hand-coded features have been determined, there are very strong limitations on what a perceptron can learn.
    - Classic example: XOR function.

Wait...

- Didn’t we just say that...
  - Perceptrons correspond to generalized linear discriminants
  - And Perceptrons are very limited...
  - Doesn’t this mean that what we have been doing so far in this lecture has the same problems???
- Yes, this is the case.
  - A linear classifier cannot solve certain problems (e.g., XOR).
  - However, with a non-linear classifier based on the right kind of features, the problem becomes solvable.
  - So far, we have solved such problems by hand-designing good features \(\phi\) and kernels \(\phi^T \phi\).
  - Can we also learn such feature representations?

Multi-Layer Perceptrons

- Adding more layers

  \[
  y_k(x) = g^{(2)} \left( \sum_{i=1}^{h} W_{ki}^{(2)} g^{(1)} \left( \sum_{j=0}^{d} W_{ij}^{(1)} x_j \right) \right)
  \]

  - Activation functions \(g^{(k)}\)
    - For example: \(g^{(2)}(a) = \sigma(a), g^{(1)}(a) = a\)
  - The hidden layer can have an arbitrary number of nodes
    - There can also be multiple hidden layers.
  - Universal approximators
    - A 2-layer network (1 hidden layer) can approximate any continuous function of a compact domain arbitrarily well! (assuming sufficient hidden nodes)

Topics of This Lecture

- A Short History of Neural Networks
- Perceptrons
  - Definition
  - Loss functions
  - Regularization
  - Limits
- Multi-Layer Perceptrons
  - Definition
  - Learning
Learning with Hidden Units

- Networks without hidden units are very limited in what they can learn
  - More layers of linear units do not help ⇒ still linear
  - Fixed output non-linearities are not enough.

- We need multiple layers of adaptive non-linear hidden units. But how can we train such nets?
  - Need an efficient way of adapting all weights, not just the last layer.
  - Learning the weights to the hidden units = learning features
  - This is difficult, because nobody tells us what the hidden units should do.
  ⇒ Next lecture

References and Further Reading

- More information on Neural Networks can be found in Chapters 6 and 7 of the Goodfellow & Bengio book

Ian Goodfellow, Aaron Courville, Yoshua Bengio
Deep Learning
MIT Press, in preparation

https://gozlfe.github.io/dlbook/