Recap: Generalized Linear Models

- Generalized linear model
  \[ y(x) = g(w^T x + w_0) \]
  - \( g(\cdot) \) is called an activation function and may be nonlinear.
  - The decision surfaces correspond to \( y(x) = \text{const.} \iff w^T x + w_0 = \text{const.} \)
  - If \( g \) is monotonous (which is typically the case), the resulting decision boundaries are still linear functions of \( x \).
- Advantages of the non-linearity
  - Can be used to bound the influence of outliers and “too correct” data points.
  - When using a sigmoid for \( g(\cdot) \), we can interpret \( y(x) \) as posterior probabilities.

Recap: Extension to Nonlinear Basis Fcts.

- Generalization
  - Transform vector \( x \) with \( M \) nonlinear basis functions \( \phi_j(x) \):
  \[ y_k(x) = \sum_{j=1}^{M} w_{kj} \phi_j(x) + w_{k0} \]
- Advantages
  - Transformation allows non-linear decision boundaries.
  - By choosing the right \( \phi_j \), every continuous function can (in principle) be approximated with arbitrary accuracy.
- Disadvantage
  - The error function can in general no longer be minimized in closed form.
  \( \Rightarrow \) Minimization with Gradient Descent

Recap: Basis Functions

- Generally, we consider models of the following form
  \[ y_k(x) = \sum_{j=0}^{M} w_{kj} \phi_j(x) = w^T \phi(x) \]
  - where \( \phi_j(x) \) are known as basis functions.
  - In the simplest case, we use linear basis functions: \( \phi_j(x) = x \).
- Other popular basis functions
  - Polynomial
  - Gaussian
  - Sigmoid

Recap: Iterative Methods for Estimation

- Gradient Descent (1st order)
  \[ w^{(r+1)} = w^{(r)} - \eta \nabla E(w) \big|_{w^{(r)}} \]
  - Simple and general
  - Relatively slow to converge, has problems with some functions
- Newton-Raphson (2nd order)
  \[ w^{(r+1)} = w^{(r)} - \eta H^{-1} \nabla E(w) \big|_{w^{(r)}} \]
  where \( H = \nabla^2 E(w) \) is the Hessian matrix, i.e. the matrix of second derivatives.
  - Local quadratic approximation to the target function
  - Faster convergence
Recap: Gradient Descent

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

Example: Quadratic error function 

- Sequential updating leads to delta rule (LMS rule) 
  \[ w_{kj}^{(r+1)} = w_{kj}^{(r)} - \eta (y_k(x_n; w) - t_{kn}) \phi_j(x_n) \]
  \[ = w_{kj}^{(r)} - \eta \delta_{kn} \phi_j(x_n) \]

where 
\[ \delta_{kn} = y_k(x_n; w) - t_{kn} \]

\[ \Rightarrow \text{Simply feed back the input data point, weighted by the classification error.} \]

Recap: Gradient Descent

- Cases with differentiable, non-linear activation function 
  \[ y_k(x) = g(ak) = g \left( \sum_{j=0}^{M} w_{kj} \phi_j(x_n) \right) \]
- Gradient descent (again with quadratic error function) 
  \[ \frac{\partial E_n(w)}{\partial w_{kj}} = \frac{\partial g(ak)}{\partial w_{kj}} (y_k(x_n; w) - t_{kn}) \phi_j(x_n) \]
  \[ w_{kj}^{(r+1)} = w_{kj}^{(r)} - \eta \delta_{kn} \phi_j(x_n) \]
  \[ \delta_{kn} = \frac{\partial g(ak)}{\partial w_{kj}} (y_k(x_n; w) - t_{kn}) \]

Recap: Probabilistic Discriminative Models

- Consider models of the form 
  \[ p(C_1|\phi) = \frac{1}{1 + \exp(-\beta \phi)} = 1 - p(C_2|\phi) \]

- 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 \( p(C_1|C_2) \) and \( p(C_2|C_1) \).

Recap: Logistic Regression

- Let’s consider a data set \( \{ \phi_n, t_n \} \) with \( n = 1, \ldots, N \), 
  where \( \phi_n = \phi(x_n) \) and \( t_n \in \{0, 1\} \), \( t = (t_1, \ldots, t_N)^T \).
- With \( y_n = p(C_1|\phi_n) \), we can write the likelihood as
  \[ p(t|w) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n} \]
- Define the error function as the negative log-likelihood
  \[ E(w) = -\ln p(t|w) = -\sum_{n=1}^{N} \left( t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right) \]
- This is the so-called cross-entropy error function.

Gradient of the Error Function

- Error function 
  \[ E(w) = -\sum_{n=1}^{N} \left( t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right) \]
- Gradient 
  \[ \nabla E(w) = -\sum_{n=1}^{N} \left( t_n \frac{\phi_n}{y_n} + (1 - t_n) \frac{\phi_n}{1 - y_n} \right) \]
  \[ = -\sum_{n=1}^{N} \left( t_n \frac{\phi_n}{y_n} \right) \]
  \[ = -\sum_{n=1}^{N} \left( (t_n - 1) y_n + (1 - t_n) \phi_n \right) \]
  \[ = \sum_{n=1}^{N} \left( y_n - t_n \right) \phi_n \]
Gradient of the Error Function

- Gradient for logistic regression
  \[ \nabla E(w) = \sum_{n=1}^{N} (y_n - t_n) \phi_n \]
- Does this look familiar to you?
- This is the same result as for the Delta (=LMS) rule
  \[ w_{kj}^{(r+1)} = w_{kj}^{(r)} - \eta (y_k(X_n; w) - t_{kn}) \phi_j(X_n) \]
- We can use this to derive a sequential estimation algorithm.
  - However, this will be quite slow...

Newton-Raphson for Logistic Regression

- Now, let's try Newton-Raphson on the cross-entropy error function:
  \[ E(w) = -\sum_{n=1}^{N} \left( t_n \ln y_n + (1 - t_n) \ln (1 - y_n) \right) \]
  \[ \nabla E(w) = \sum_{n=1}^{N} \left( y_n - t_n \right) \phi_n = \Phi^T (y - t) \]
  \[ H = \nabla \nabla E(w) = \sum_{n=1}^{N} \left( y_n (1 - y_n) \right) \phi_n^T = \Phi^T R \Phi \]
  where \( R \) is an \( N \times N \) diagonal matrix with \( R_{nn} = y_n(1 - y_n) \).
  \( \Rightarrow \) The Hessian is no longer constant, but depends on \( w \) through the weighting matrix \( R \).

Iteratively Reweighted Least Squares

- Update equations
  \[ w^{(r+1)} = w^{(r)} - \left( \Phi^T R \Phi \right)^{-1} \left( \Phi^T R w^{(r)} - \Phi^T (y - t) \right) \]
  \[ = \left( \Phi^T R \Phi \right)^{-1} \Phi^T R z \]
  with \( z = \Phi w^{(r)} - R^{-1} (y - t) \)
- Again very similar form (normal equations)
  - But now with non-constant weighting matrix \( R \) (depends on \( w \)).
  - Need to apply normal equations iteratively.
  \( \Rightarrow \) Iteratively Reweighted Least-Squares (IRLS)

Summary: Logistic Regression

- Properties
  - Directly represent posterior distribution \( p(\theta | C_n) \)
  - Requires fewer parameters than modeling the likelihood + prior.
  - Very often used in statistics.
  - It can be shown that the cross-entropy error function is concave
    - Optimization leads to unique minimum
    - But no closed form solution exists
    - Iterative optimization (IRLS)
  - Both online and batch optimizations exist
- Caveat
  - Logistic regression tends to systematically overestimate odds ratios when the sample size is less than ~500.

Topics of This Lecture

- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
- Linear Support Vector Machines
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion
### 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.

$$\begin{align*}
y(x; w) &= \begin{pmatrix} P(y = 1|x, w) \\ P(y = 2|x, w) \\ \vdots \\ P(y = K|x, w) \end{pmatrix} = \frac{1}{\sum_{k=1}^{K} \exp(w_k^T x)} \begin{pmatrix} \exp(w_1^T x) \\ \exp(w_2^T x) \\ \vdots \\ \exp(w_K^T x) \end{pmatrix}
\end{align*}$$

- This uses the **softmax function**
  $$\exp(a_k) = \sum_j \exp(a_j)$$

- Note: the resulting distribution is normalized.

---

### Softmax Regression Cost Function

- **Logistic regression**
  - Alternative way of writing the cost function with indicator function $I()$

$$E(w) = - \sum_{n=1}^{N} \{t_n \ln(y_n) + (1 - t_n) \ln(1 - y_n)\}$$

---

### 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 $w_c$.
  - We can now plug this into a standard optimization package.

---

### Note on Error Functions

- **Ideal misclassification error function** (black)
  - This is what we want to approximate (error = #misclassifications)
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.
  - We cannot minimize it by gradient descent.

---

### Topics of This Lecture

- **Softmax Regression**
  - Multi-class generalization
  - Gradient descent solution

- **Note on Error Functions**
  - Ideal error function
  - Quadratic error
  - Cross-entropy error

- **Linear Support Vector Machines**
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

---

### 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.
Comparing Error Functions (Loss Functions)

- Cross-Entropy Error
  - Minimizer of this error is given by posterior class probabilities.
  - Concave error function, unique minimum exists.
  - Robust to outliers, error increases only roughly linearly
  - But no closed-form solution, requires iterative estimation.

Overview: Error Functions

- Ideal Misclassification Error
  - This is what we would like to optimize.
  - But cannot compute gradients here.
- Quadratic Error
  - Easy to optimize, closed-form solutions exist.
  - But not robust to outliers.
- Cross-Entropy Error
  - Minimizer of this error is given by posterior class probabilities.
  - Concave error function, unique minimum exists.
  - But no closed-form solution, requires iterative estimation.

Looking at the error function this way gives us an analysis tool to compare the properties of classification approaches.

Let’s Put This To Practice…

- Squared error on sigmoid/tanh output function
  - Avoids penalizing “too correct” data points.
  - But: zero gradient for confidently incorrect classifications!
  - Do not use L2 loss with sigmoid outputs (instead: cross-entropy)!

Topics of This Lecture

- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-Entropy error
- Linear Support Vector Machines
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

Example: Linearly Separable Data

- Overfitting is often a problem with linearly separable data
  - Which of the many possible decision boundaries is correct?
  - All of them have zero error on the training set…
  - However, they will most likely result in different predictions on novel test data.
  - Different generalization performance
- How to select the classifier with the best generalization performance?

Generalization and Overfitting

- Goal: predict class labels of new observations
  - Train classification model on limited training set.
  - The further we optimize the model parameters, the more the training error will decrease.
  - However, at some point the test error will go up again.
  - Overfitting to the training set!
Revisiting Our Previous Example...

- How to select the classifier with the best generalization performance?
  - Intuitively, we would like to select the classifier which leaves maximal "safety room" for future data points.
  - This can be obtained by maximizing the margin between positive and negative data points.
  - It can be shown that the larger the margin, the lower the corresponding classifier’s VC dimension (capacity for overfitting).
- The SVM takes up this idea
  - It searches for the classifier with maximum margin.
  - Formulation as a convex optimization problem → Possible to find the globally optimal solution!

Support Vector Machine (SVM)

- Let's first consider linearly separable data
  - \( X \) training data points \( \{(x_i, y_i)\}_{i=1}^N \) \( x_i \in \mathbb{R}^d \)
  - Target values \( t_i \in \{-1, 1\} \)
  - Hyperplane separating the data

\[ w^T x + b = 0 \]

- Margin of the hyperplane:
  - \( d_+ \): distance to nearest pos. training example
  - \( d_- \): distance to nearest neg. training example
- We can always choose \( w, b \) such that \( d_+ = d_- = \frac{1}{\|w\|} \)

Support Vector Machine (SVM)

- Since the data is linearly separable, there exists a hyperplane with
  - \( w^T x_i + b \geq +1 \) for \( t_i = +1 \)
  - \( w^T x_i + b \leq -1 \) for \( t_i = -1 \)

- Combined in one equation, this can be written as
  \[ t_i(w^T x_i + b) \geq 1 \quad \forall n \]

\[ t_n(w^T x_n + b) = 1 \]

- By definition, there will always be at least one such point.

Support Vector Machine (SVM)

- We can choose \( w \) such that
  - \( w^T x_i + b = +1 \) for one \( t_i = +1 \)
  - \( w^T x_i + b = -1 \) for one \( t_i = -1 \)

- The distance between those two hyperplanes is then the margin
  \[ d_- = d_+ = \frac{1}{\|w\|} \]
  \[ d_- + d_+ = \frac{2}{\|w\|} \]

\( \Rightarrow \) We can find the hyperplane with maximal margin by minimizing \( \|w\|^2 \)

Support Vector Machine (SVM)

- Optimization problem
  - Find the hyperplane satisfying
    \[ \arg \min_{w,b} \frac{1}{2}\|w\|^2 \]
    under the constraints
    \[ t_n(w^T x_n + b) \geq 1 \quad \forall n \]

  - Quadratic programming problem with linear constraints.
  - Can be formulated using Lagrange multipliers.

- Who is already familiar with Lagrange multipliers?
  - Let's look at a real-life example…
**Recap: Lagrange Multipliers**

**Problem**
- We want to maximize $K(x)$ subject to constraints $f(x) = 0$.
- Example: we want to get as close as possible, but there is a fence.
- How should we move?
  - $f(x) = 0$ (subject to constraints)
  - $f(x) > 0$ (subject to constraints)
- We want to maximize $\nabla f$.
- But we can only move parallel to the fence, i.e., along $\nabla K = \nabla f + \lambda \nabla f$ with $\lambda = 0$.

**Solution lies inside $+\$**

**Solution lies on boundary $k$**

**Example:** There might be a hill from $1$.

**Introduction positive Lagrange multipliers:** $K(x)$

**In both cases**

**Example:** we want to get as close as $b = 0$.

**Lagrangian Formulation**

**Two cases**

- Solution lies on boundary
  - $f(x) = 0$ for some $\lambda > 0$
  - Solution lies inside $f(x) > 0$
  - Constraint inactive: $\lambda = 0$
  - In both cases
    - $\lambda f(x) = 0$

**SVM – Lagrangian Formulation**

**Find hyperplane minimizing $\|w\|^2$ under the constraints**

$t_n (w^T x_n + b) - 1 \geq 0 \quad \forall n$

**Lagrangian formulation**

- Introduce positive Lagrange multipliers: $\alpha_n \geq 0 \quad \forall n$
- Minimize Lagrangian ("primal form")

$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} \alpha_n \{ t_n (w^T x_n + b) - 1 \}$

- i.e., find $w$, $b$, and $\alpha$ such that

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n t_n = 0$$

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n t_n x_n$$

**Optimize**

$$\max_{x,\lambda} L(x, \lambda) = K(x) + \lambda f(x)$$

$$\frac{\partial L}{\partial x} = \nabla K + \lambda \nabla f = 0$$

**Karush-Kuhn-Tucker (KKT) conditions:**

$$\lambda \geq 0$$

$$f(x) \geq 0$$

$$\lambda f(x) = 0$$

**The solution of $L_p$ needs to fulfill the KKT conditions**

- Necessary and sufficient conditions

$$\alpha_n \geq 0$$

$$t_n y(x_n) - 1 \geq 0$$

$$f(x) \geq 0$$

$$\lambda f(x) = 0$$
**SVM – Solution (Part 1)**

- **Solution for the hyperplane**
  - Computed as a linear combination of the training examples
    \[
    w = \sum_{n=1}^{N} \alpha_n t_n x_n
    \]
  - Because of the KKT conditions, the following must also hold
    \[
    \alpha_n \left( t_n (w^T x_n + b) - 1 \right) = 0 \quad \text{(KKT)}
    \]
  - This implies that \( \alpha_n > 0 \) only for training data points for which
    \[
    (t_n (w^T x_n + b) - 1) = 0
    \]
  - \( \Rightarrow \) Only some of the data points actually influence the decision boundary!

**SVM – Support Vectors**

- The training points for which \( \alpha_n > 0 \) are called “support vectors”.
- Graphical interpretation:
  - The support vectors are the points on the margin.
  - They define the margin and thus the hyperplane.
  - \( \Rightarrow \) Robustness to “too correct” points!

**SVM – Solution (Part 2)**

- **Solution for the hyperplane**
  - To define the decision boundary, we still need to know \( b \).
  - Observation: any support vector \( x_n \) satisfies
    \[
    t_n f(x_n) = t_n \left( \sum_{m \in S} \alpha_m t_m x_n^T x_m + b \right) = 1
    \]
  - Using \( t_n^2 = 1 \) we can derive:
    \[
    b = t_n - \sum_{m \in S} \alpha_m t_m x_n^T x_m
    \]
  - In practice, it is more robust to average over all support vectors:
    \[
    b = \frac{1}{NS} \sum_{n \in S} \left( t_n - \sum_{m \in S} \alpha_m t_m x_n^T x_m \right)
    \]

**SVM – Discussion (Part 1)**

- **Linear SVM**
  - Linear classifier
  - SVMs have a “guaranteed” generalization capability.
  - Formulation as convex optimization problem.
  - \( \Rightarrow \) Globally optimal solution!
- **Primal form formulation**
  - Solution to quadratic prog. problem in \( M \) variables is in \( \mathcal{O}(M^3) \).
  - Here: \( D \) variables \( \Rightarrow \mathcal{O}(D^3) \)
  - Problem: scaling with high-dim. data (“curse of dimensionality”)

**SVM – Dual Formulation**

- Improving the scaling behavior: rewrite \( L_p \) in a dual form
  \[
  L_p = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} \alpha_n \left\{ t_n (w^T x_n + b) - 1 \right\}
  \]
  \[
  = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} \alpha_n t_n w^T x_n - b \sum_{n=1}^{N} \alpha_n t_n + \sum_{n=1}^{N} \alpha_n
  \]
  - Using the constraint \( \sum_{n=1}^{N} \alpha_n t_n = 0 \) we obtain
    \[
    \frac{\partial L_p}{\partial b} = 0
    \]
  - \( L_p = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} \alpha_n t_n w^T x_n + \sum_{n=1}^{N} \alpha_n \)

**SVM – Dual Formulation**

- Using the constraint \( w = \sum_{n=1}^{N} \alpha_n t_n x_n \), we obtain
  \[
  \frac{\partial L_p}{\partial w} = 0
  \]
  \[
  L_p = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m x_n^T x_m + \sum_{n=1}^{N} \alpha_n
  \]
  \[
  = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m (x_n^T x_m) + \sum_{n=1}^{N} \alpha_n
SVM – Dual Formulation

\[ L = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m) + \sum_{n=1}^{N} a_n \]

> Applying \( \frac{1}{2} \|w\|^2 = \frac{1}{2} w^T w \) and again using \( w = \sum_{n=1}^{N} a_n t_n x_n \)

\[ \frac{1}{2} w^T w = \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m) \]

> Inserting this, we get the Wolfe dual

\[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m) \]

SVM – Discussion (Part 2)

- Dual form formulation
  - In going to the dual, we now have a problem in \( N \) variables \((a_n)\).
  - Isn’t this worse?? We penalize large training sets!

- However...
  1. SVMs have sparse solutions: \( a_n \neq 0 \) only for support vectors!
  2. We have avoided the dependency on the dimensionality.

References and Further Reading

- More information on SVMs can be found in Chapter 7.1 of Bishop’s book.