Ensamble methods.
Mixtures of experts

Mixture of experts model

- Ensemble methods:
  - Use a combination of simpler learners to improve predictions
- Mixture of expert model:
  - Covers different input regions with different learners
  - A “soft” switching between learners

· Mixture of experts
  Expert = learner
Mixture of experts model

• **Gating network**: decides what expert to use
  
  \[ g_1, g_2, \ldots, g_k \] - gating functions

---

Learning mixture of experts

• **Learning consists of two tasks:**
  – Learn the parameters of individual expert networks
  – Learn the parameters of the gating network
    • Decides where to make a split
  • **Assume**: gating functions give probabilities
    
    \[ 0 \leq g_1(x), g_2(x), \ldots, g_k(x) \leq 1 \]
    \[ \sum_{u=1}^{k} g_u(x) = 1 \]

  • Based on the probability we partition the space
    – partitions belongs to different experts
  • How to model the gating network?
    – **A multiway classifier model**:
      • softmax model
      • a generative classifier model
Learning mixture of experts

- Assume we have a **set of linear experts**
  \[ \mu_i = \theta_i^T x \]  
  (Note: bias terms are hidden in x)
- Assume a **softmax gating network**
  \[ g_i(x) = \frac{\exp(\eta_i^T x)}{\sum_{u=1}^{k} \exp(\eta_u^T x)} \approx p(\omega_i \mid x, \eta) \]
- Likelihood of y (assumed that errors for different experts are normally distributed with the same variance)
  \[
P(y \mid x, \Theta, \eta) = \sum_{i=1}^{k} P(\omega_i \mid x, \eta) p(y \mid x, \omega_i, \Theta) = \sum_{i=1}^{k} \left[ \frac{\exp(\eta_i^T x)}{\sum_{j=1}^{k} \exp(\eta_j^T x)} \right] \left[ \frac{1}{\sqrt{2\pi}\sigma} \exp\left( -\frac{\|y - \mu_i\|^2}{2\sigma^2} \right) \right]
\]

---

Learning mixture of experts

**Gradient learning.**

**On-line update rule for parameters** \( \Theta_i \) of expert \( i \)
- If we know the expert that is responsible for \( x \)
  \[ \theta_{y} \leftarrow \theta_{y} + \alpha_{y} (y - \mu_{i}) x_j \]
- If we do not know the expert
  \[ \theta_{y} \leftarrow \theta_{y} + \alpha_{y} h_i (y - \mu_{i}) x_j \]

\( h_i \) - **responsibility of the \( i \)th expert** = a kind of posterior

\[
h_i(x, y) = \frac{g_i(x) p(y \mid x, \omega_i, \Theta)}{\sum_{u=1}^{k} g_u(x) p(y \mid x, \omega_u, \Theta)} = \frac{g_i(x) \exp\left( -\frac{1}{2\sigma^2} \|y - \mu_i\|^2 \right)}{\sum_{u=1}^{k} g_u(x) \exp\left( -\frac{1}{2\sigma^2} \|y - \mu_u\|^2 \right)}
\]

\( g_i(x) \) - a prior \( \exp(...) \) - a likelihood
Learning mixtures of experts

Gradient methods

- **On-line learning of gating network parameters** $\eta_j$

  $$\eta_j \leftarrow \eta_j + \beta_j (h_j(x,y) - g_j(x)) x_j$$

- The learning with conditioned mixtures can be extended to learning of parameters of an arbitrary expert network – e.g. logistic regression, multilayer neural network

  $$\theta_j \leftarrow \theta_j + \beta_j \frac{\partial l}{\partial \theta_j}$$

  $$\frac{\partial l}{\partial \theta_j} = \frac{\partial l}{\partial \mu_j} \frac{\partial \mu_j}{\partial \theta_j} = h_j \frac{\partial \mu_j}{\partial \theta_j}$$

Learning mixture of experts

**EM algorithm** offers an alternative way to learn the mixture

**Algorithm:**

1. **Expectation step**

   $$Q(\Theta | \Theta') = E_{H \mid X,Y,\Theta} \log P(H, Y \mid X, \Theta, \zeta)$$

2. **Maximization step**

   $$\Theta = \arg \max_{\Theta} Q(\Theta | \Theta')$$

   until no or small improvement in $Q(\Theta | \Theta')$

   - Hidden variables are identities of expert networks responsible for (x,y) data points
Learning mixture of experts with EM

- Assume we have a **set of linear experts**
  \[ \mu_i = \theta_i^T x \]
- Assume a **softmax gating network**
  \[ g_i(x) = P(\omega_i \mid x, \eta) \]
- **Q function to optimize**
  \[ Q(\Theta \mid \Theta') = E_{H \mid X,Y,\Theta'} \log P(H, Y \mid X, \Theta, \xi) \]
- **Assume:**
  - \( l \) indexes different data points
  - \( \delta_i^l \) an indicator variable for the data point \( l \) to be covered by an expert \( i \)
  \[ Q(\Theta \mid \Theta') = \sum_l \sum_i E(\delta_i^l \mid x', y', \Theta', \eta') \log(P(y', \omega_i \mid x', \Theta, \eta)) \]

---

Learning mixture of experts with EM

- **Assume:**
  - \( l \) indexes different data points
  - \( \delta_i^l \) an indicator variable for data point \( l \) and expert \( i \)
  \[ Q(\Theta \mid \Theta') = \sum_l \sum_i E(\delta_i^l \mid x', y', \Theta', \eta') \log(P(y', \omega_i \mid x', \Theta, \eta)) \]
  \[ E(\delta_i^l \mid x', y', \Theta', \eta') = h_i^l(x', y') = \frac{g_i(x') p(y \mid x', \omega_i, \theta')}{\sum_{u=1}^k g_u(x') p(y \mid x', \omega_u, \theta')} \]
  Responsibility of the expert \( i \) for \( (x,y) \)
  \[ Q(\Theta \mid \Theta') = \sum_l \sum_i h_i^l(x', y') \log(P(y', \omega_i \mid x', \Theta, \eta)) \]
Learning mixture of experts with EM

- The maximization step boils down to the problem that is equivalent to the problem of finding the ML estimates of the parameters of the expert and gating networks.

\[
Q(\Theta \mid \Theta') = \sum_i \sum_l h_i^l(x^l, y^l) \log(P(y^l, \omega \mid x^l, \Theta, \eta))
\]

\[
\log(P(y^l, \omega \mid x^l, \Theta, \eta)) = \log P(y^l \mid \omega, x^l, \Theta) + \log P(\omega \mid x^l, \eta)
\]

- Expert network \(i\) (Linear regression)
- Gating network (Softmax)

- Note that any optimization technique can be applied in this step.

Hierarchical mixture of experts

- **Mixture of experts**: define a probabilistic split
- The idea can be extended to a **hierarchy of experts** (a kind of a probabilistic decision tree)
Hierarchical mixture model

An output is conditioned (gated) on multiple mixture levels

\[ P(y \mid x, \Theta) = \sum_u P(\omega_u \mid x, \eta) \sum_v \prod_l P(\omega_{uv} \mid x, \omega_u, \xi) \times \prod_l P(\omega_{uv, l} \mid x, \omega_u, \omega_v, \ldots) \times \prod_l P(y \mid x, \omega_u, \omega_v, \ldots, \Theta) \]

- Define \( \Omega_{uv \ldots} = \{\omega_u, \omega_{uv}, \ldots, \omega_{uv \ldots} \} \)

\[ P(\Omega_{uv \ldots} \mid x, \Theta) = P(\omega_u \mid x)P(\omega_{uv} \mid x, \omega_u) \cdots P(\omega_{uv \ldots} \mid x, \omega_u, \omega_{uv}, \ldots) \]

- Then

\[ P(y \mid x, \Theta) = \sum_u \sum_v \cdots \sum_l P(\Omega_{uv \ldots} \mid x, \Theta)P(y \mid x, \Omega_{uv \ldots}, \Theta) \]

Hierarchical mixture of experts

- Multiple levels of probabilistic gating functions

\[ g_u(x) = P(\omega_u \mid x, \Theta) \quad \quad g_{uv}(x) = P(\omega_{uv} \mid x, \omega_u, \Theta) \]

- Multiple levels of responsibilities

\[ h_u(x, y) = P(\omega_u \mid x, y, \Theta) \quad \quad h_{uv}(x, y) = P(\omega_{uv} \mid x, y, \omega_u, \Theta) \]

- How they are related?

\[ P(\omega_{uv} \mid x, y, \omega_u, \Theta) = \frac{\sum_y P(y \mid x, \omega_u, \omega_{uv}, \Theta)P(\omega_{uv} \mid x, \omega_u, \Theta)}{\sum_y P(y \mid x, \omega_u, \omega_{uv}, \Theta)P(\omega_{uv} \mid x, \omega_u, \Theta)} \]

\[ \sum_y P(y, \omega_{uv} \mid x, \omega_u, \Theta) = P(y \mid x, \omega_u, \Theta) \]
Hierarchical mixture of experts

- Responsibility for the top layer

\[ h_u(x, y) = P(\omega_u | x, y, \Theta) = \frac{P(y | x, \omega_u, \Theta) P(\omega_y | x, \Theta)}{\sum_u P(y | x, \omega_u, \Theta) P(\omega_y | x, \Theta)} \]

- But \( P(y | x, \omega_u, \Theta) \) is computed while computing

\[ h_{vju}(x, y) = P(\omega_{uv} | x, y, \omega_u, \Theta) \]

- General algorithm:
  - Downward sweep; calculate

\[ g_{vju}(x) = P(\omega_{uv} | x, \omega_u, \Theta) \]

  - Upward sweep; calculate

\[ h_u(x, y) = P(\omega_u | x, y, \Theta) \]

On-line learning

- Assume linear experts \( \mu_{uv} = \theta_{uv}^T x \)

- Gradients (vector form):

\[ \frac{\partial l}{\partial \theta_{uv}} = h_u h_{vju}(y - \mu_{uv})x \]

\[ \frac{\partial l}{\partial \eta} = (h_u - g_u)x \quad \text{Top level (root node)} \]

\[ \frac{\partial l}{\partial \xi} = h_u (h_{vju} - g_{vju})x \quad \text{Second level node} \]

- Again: can it can be extended to different expert networks
Ensemble methods:
Bagging.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Ensemble methods

• Mixture of experts
  – Different ‘base’ models (classifiers, regressors) cover different parts of the input space
• Alternative idea:
  – Train several ‘base’ models on the complete input space, but on slightly different train sets
  – Combine their decision to produce the final result
    • Sometimes called Committee machines
• Goal: Improve the accuracy of the ‘base’ model
• Methods:
  – Bagging
  – Boosting
  – Stacking (not covered)
Bagging (Bootstrap Aggregating)

- **Given:**
  - Training set of $N$ examples
  - A class of learning models (e.g. decision trees, neural networks, …)
- **Goal:**
  - Improve the accuracy of one model by using multiple copies of it
- **Motivation:**
  - **Recall:** Average of misclassification errors on different data splits gives a better estimate of the predictive ability of a learning method
  - Train multiple models on different samples and average their predictions

---

Bagging algorithm

- **Training**
  - In each iteration $t, t=1,…,T$
    - Randomly sample with replacement $N$ samples from the training set
    - Train a chosen “base model” (e.g. neural network, decision tree) on the samples
- **Test**
  - For each test example
    - Start all trained base models
    - Predict by combining results of all $T$ trained models:
      - **Regression:** averaging
      - **Classification:** a majority vote
When Bagging Works

- **Expected error** = **Bias + Variance**
  - *Expected error* is the expected discrepancy between the estimated and true function
    \[ E \left[ \left( \hat{f}(X) - E[f(X)] \right)^2 \right] \]
  - *Bias* is squared discrepancy between averaged estimated and true function
    \[ \left( E[\hat{f}(X)] - E[f(X)] \right)^2 \]
  - *Variance* is expected divergence of the estimated function vs. its average value
    \[ E\left[ \left( \hat{f}(X) - E[\hat{f}(X)] \right)^2 \right] \]
When Bagging works?
Under-fitting and over-fitting

- **Under-fitting:**
  - High bias (models are not accurate)
  - Small variance (smaller influence of examples in the training set)

- **Over-fitting:**
  - Small bias (models flexible enough to fit well to training data)
  - Large variance (models depend very much on the training set)

Averaging decreases variance

- **Example**
  - Assume we measure a random variable \( x \) with a \( N(\mu, \sigma^2) \) distribution
  - If only one measurement \( x_1 \) is done,
    - The expected mean of the measurement is \( \mu \)
    - Variance is \( \text{Var}(x_1) = \sigma^2 \)
  - If random variable \( x \) is measured \( K \) times \( (x_1, x_2, \ldots x_K) \) and the value is estimated as: \( (x_1 + x_2 + \ldots + x_K) / K \),
    - Mean of the estimate is still \( \mu \)
    - But, variance is smaller:
      - \( \frac{\text{Var}(x_1) + \ldots + \text{Var}(x_K)}{K^2} = K \sigma^2 / K^2 = \sigma^2 / K \)
  - Observe: **Bagging is a kind of averaging!**
When Bagging works

• Main property of Bagging (proof omitted)
  – Bagging decreases variance of the base model without changing the bias!!!
  – Why? averaging!
• Bagging typically helps
  – When applied with an over-fitted base model
    • High dependency on actual training data
• It does not help much
  – High bias. When the base model is robust to the changes in the training data (due to sampling)