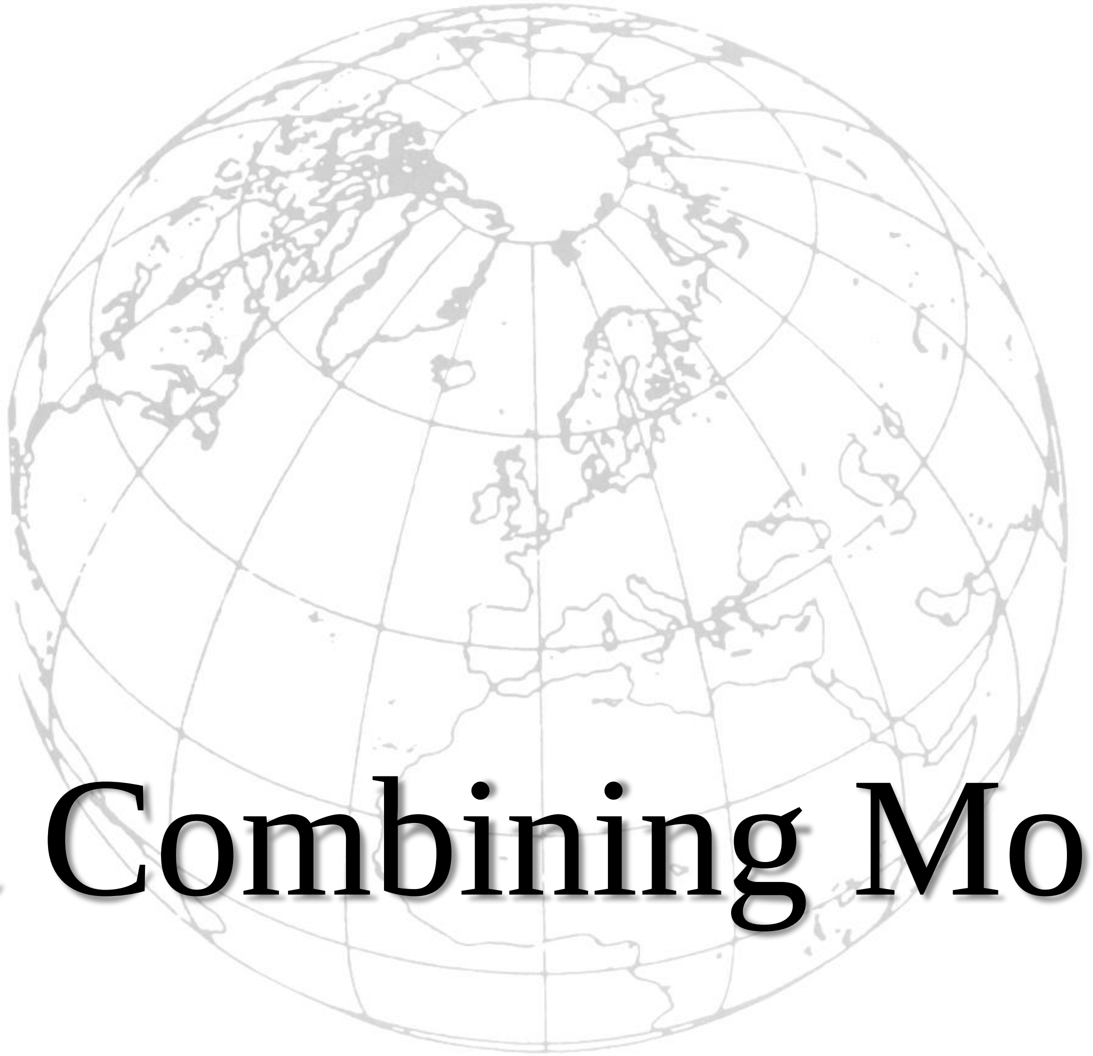


Data Combining Models

Dr. Cahit Karakuş

Motivation

- Many models can be trained on the same data
- Typically none is strictly better than others
  - Recall “no free lunch theorem”
- Can we “combine” predictions from multiple models?

- Yes, typically with significant reduction of error!

Motivation

- Combined prediction using Adaptive Basis Functions

$$f(x) = \sum_{i=1}^{M} w_m \phi_m(x; v_m)$$

- M basis functions with own parameters
- Weight / confidence of each basis function
- Parameters including M trained using data

- Another interpretation: automatically learning best representation of data for the task at hand

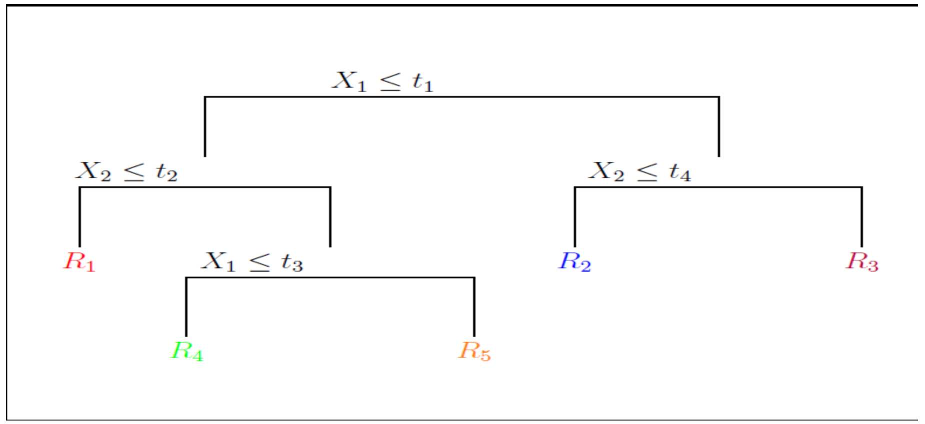
- Difference with mixture models?

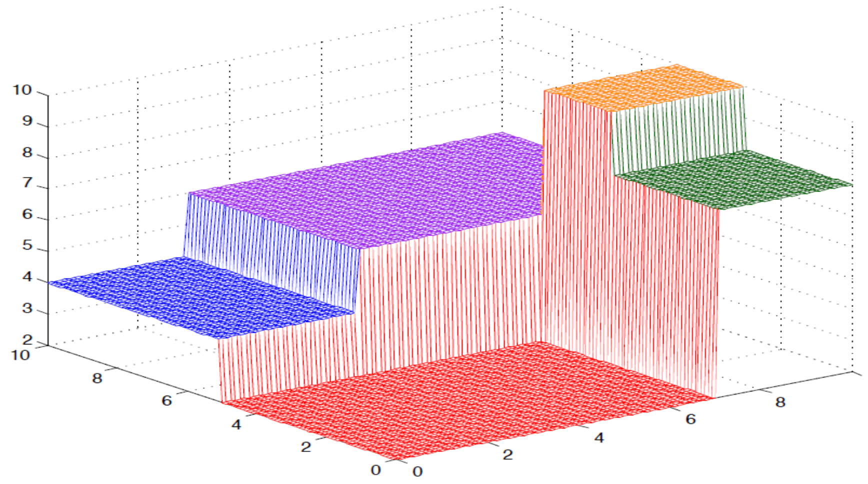
Examples of Model Combinations

- Also called Ensemble Learning
- Decision Trees
- Bagging
- Boosting
- Committee / Mixture of Experts
- Feed forward neural nets / Multi-layer perceptrons
- …

Decision Trees

- Partition input space into cuboid regions
- Simple model for each region
  - Classification: Single label; Regression: Constant real value
- Sequential process to choose model per instance
  – Decision tree





Learning Decision Trees

- Decision for each region
  - Regression: Average of training data for the region
  - Classification: Most likely label in the region
- Learning tree structure and splitting values
  - Learning optimal tree intractable
- Greedy algorithm
  - Find (node, dim., value) w/ largest reduction of “error”
    - Regression error: residual sum of squares
    - Classification: Misclassification error, entropy, …
  - Stopping condition
- Preventing overfitting: Pruning using cross validation

Pros and Cons of Decision Trees

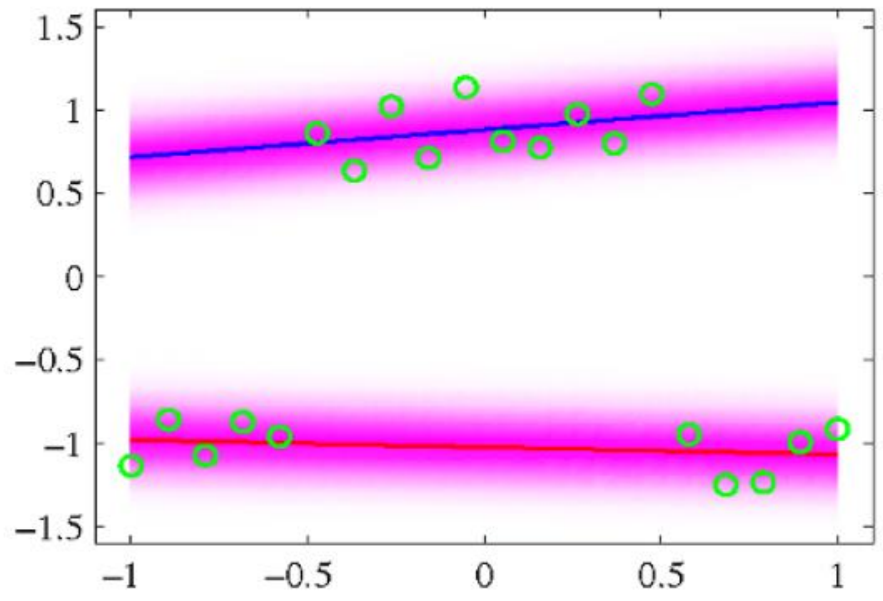
- Easily interpretable decision process
  - Widely used in practice, e.g. medical diagnosis

- Not very good performance
  - Restricted partition of space
  - Restricted to choose one model per instance
  - Unstable

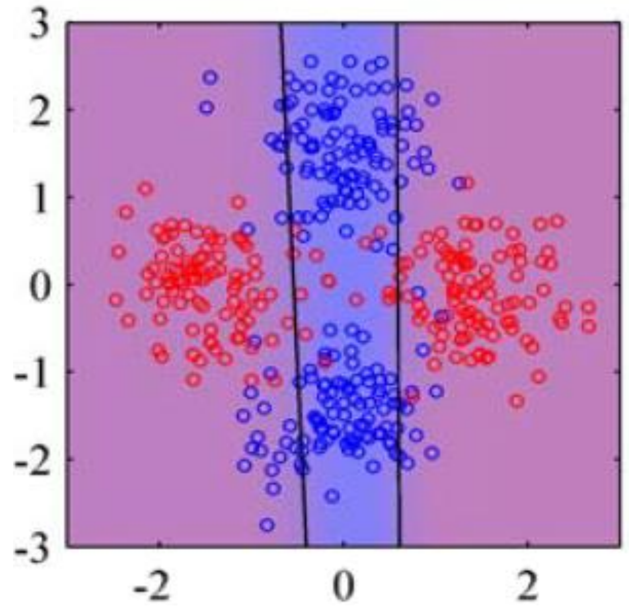
Mixture of Supervised Models

$$f(x) = \sum_i \pi_k \phi_k(x, w)$$

Mixture of linear regression models



Mixture of logistic regression models

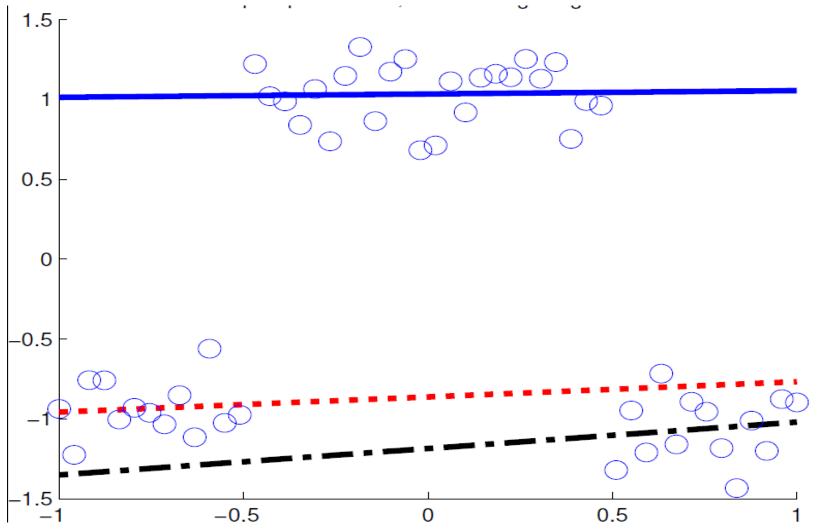


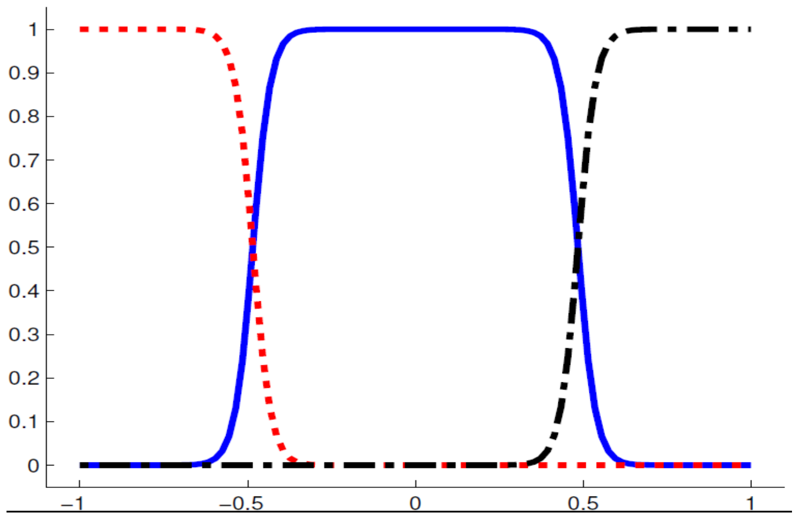
- Training using EM

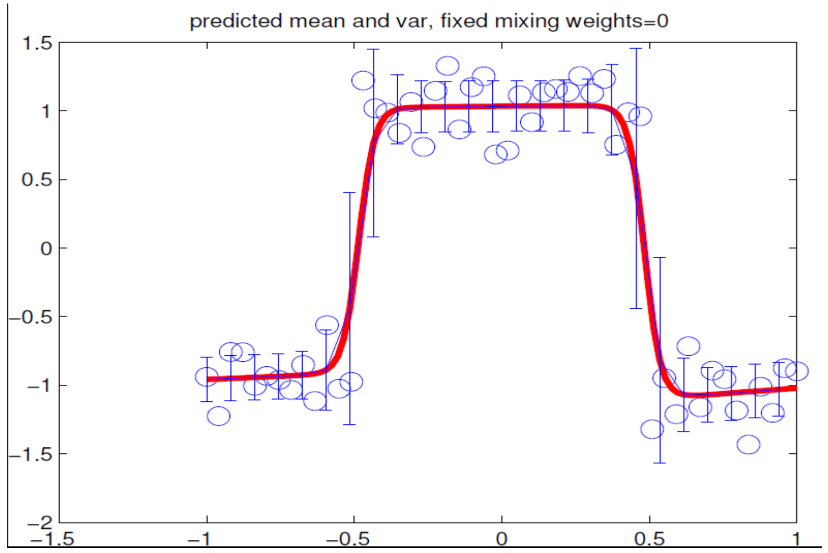
Conditional Mixture of Supervised Models

- Mixture of experts

$$f(x) = \sum_i \pi_k(x)\phi_k(x,w)$$







Bootstrap Aggregation / Bagging

- Individual models (e.g. decision trees) may have high variance along with low bias

- Construct M bootstrap datasets
- Train separate copy of predictive model on each
- Average prediction over copies

$$f(x) = \frac{1}{M} \sum_{i} f_m(x)$$

- If the errors are uncorrelated, then bagged error reduces linearly with M

Random Forests

- Training same algorithm on bootstraps creates correlated errors

- Randomly choose (a) subset of variables and (b) subset of training data

- Good predictive accuracy
- Loss in interpretability

Boosting

- Combining weak learners, ϵ-better than random
  - E.g. Decision stumps

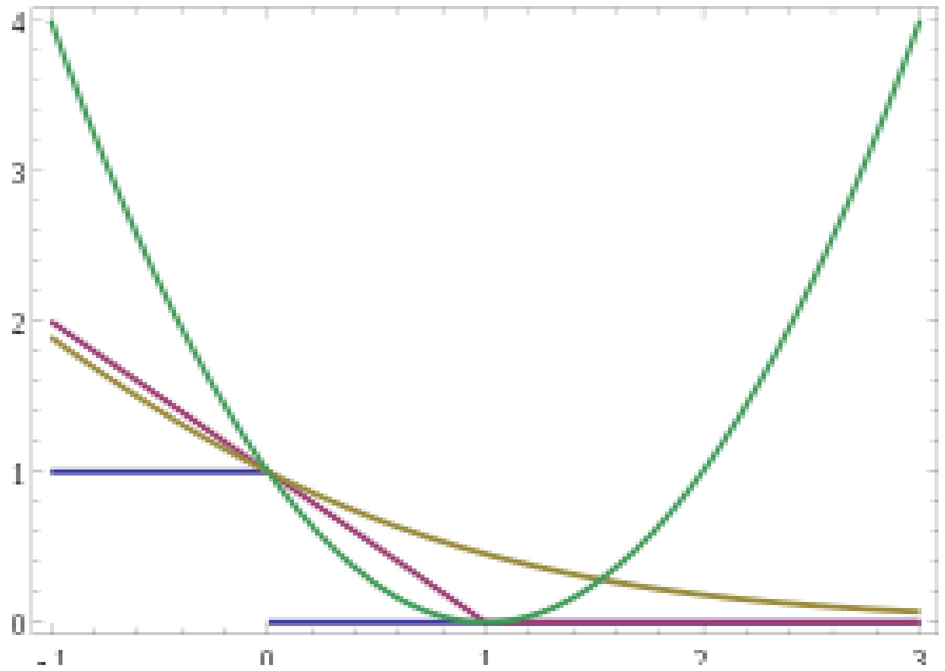
- Sequence of weighted datasets
- Weight of data point in each iteration proportional to no of misclassifications in earlier iterations

- Specific weighting scheme depends on loss function
- Theoretical bounds on error

Example loss functions and algorithms

- Squared error $(y_i - f(x_i))^2$
- Absolute error $|y_i - f(x_i)|$

- Squared loss $(1 - \tilde{y}_i f(x_i))^2$
- 0-1 loss $I(\tilde{y}_i \neq f(x_i))$
- Exponential loss $\exp(-\tilde{y}_i f(x_i))$
- Logloss $\frac{1}{\log 2}\log(1 + e^{-\tilde{y}_i f(x_i)})$
- Hinge loss $|1 - \tilde{y}_i f(x_i)|_+$



Example: AdaBoost

- Binary classification problem + Exponential loss

1. Initialize $w_n^{(1)} = \frac{1}{N}$
2. Train classifier $y_m(x)$ minimizing $\sum_n w_n^{(m)} I(y_m(x_n) \neq y_n)$
3. Evaluate $\epsilon_m = \frac{\sum_n w_n^{(m)} I(y_m(x_n) \neq y_n)}{\sum_n w_n^{(m)}}$ and $\alpha_m = \log \frac{1-\epsilon_m}{\epsilon_m}$
4. Update wts $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(x_n) \neq y_n)\}$
5. Predict $f_M(x) = sgn(\sum_{i=1}^{M} \alpha_m y_m(x))$

Neural networks: Multilayer Perceptrons

- Multiple layers of logistic regression models
- Parameters of each optimized by training

- Motivated by models of the brain
- Powerful learning model regardless

LR and R remembered …

- Linear models wi tions

$$y(x,w) = f(\sum_i w_i \phi_i(x))$$

- Fixed basis functions
- Non-linear transformation
- ϕ_i linear $\hat{y}(x; w, v) = h(\sum_{j=1\ to\ M} w_{kj}\ g(\sum_{i=1\ to\ D} v_{ji} x_i))$ tion

Feed-forward network functions

- M linear combinations of input variables

$$a_j = \sum_{i=1 \ to \ D} v_{ji} x_i$$

- Apply non-linear activation function

$$z_j = g(a_j)$$

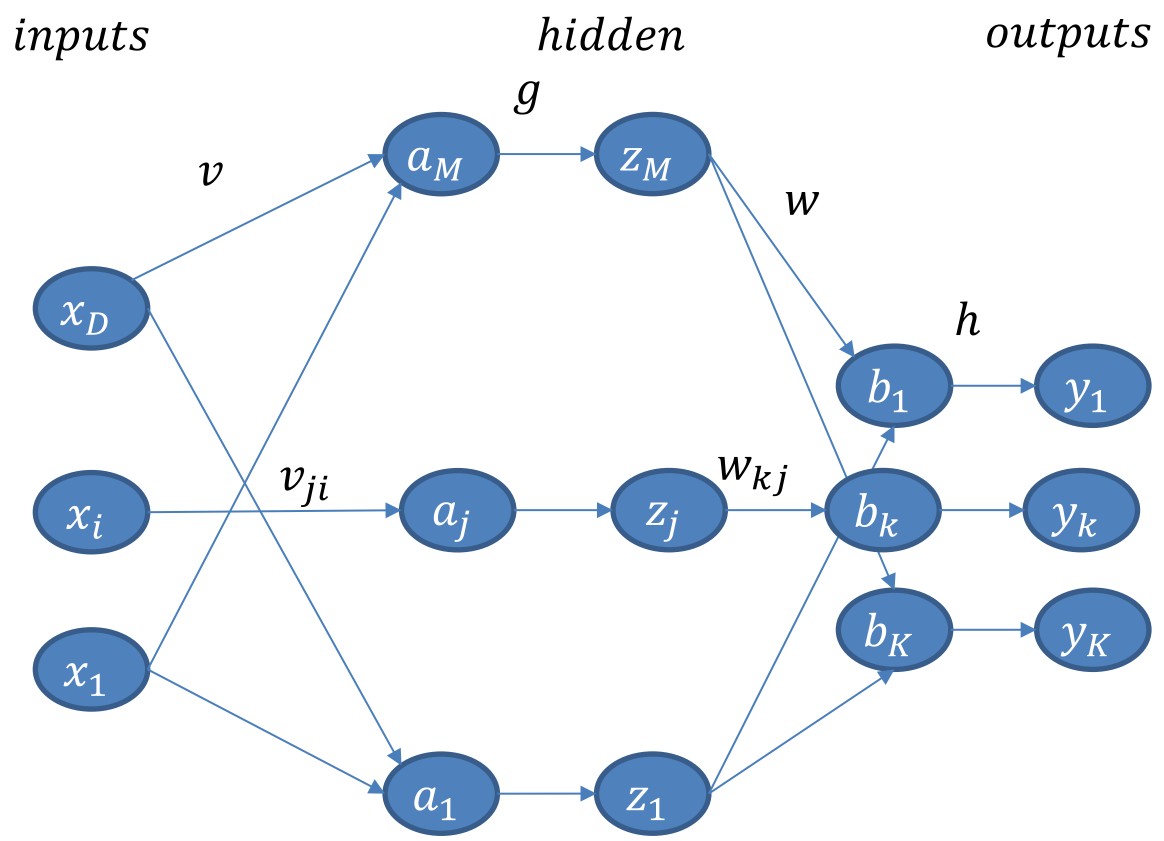
- Linear combinations to get output activations

$$b_k = \sum_{j=1 \ to \ M} w_{kj} z_j$$

- Apply output activation function to get outputs

$$y_k = h(b_k)$$

Network Representation



Easy to generalize to multiple layers

Power of feed-forward networks

- Universal approximators

  A 2 layer network with linear outputs can uniformly approximate any smooth continuous function with arbitrary accuracy given sufficient number of nodes in hidden layer

- Why are >2 layers needed?

Training

- Formulate error function in terms of weights

$$E(w, v) = \sum_{i=1\ to\ N} \left|\left|\hat{y}(x_n; w, v) - y_n\right|\right|^2$$

- Optimize weights using gradient descent

$$(w, v)^{(t+1)} = (w, v)^{(t)} - \eta \nabla E((w, v)^{(t)})$$

- Deriving the gradient looks complicated because of feed-forward …

Error Backpropagation

- Full gradient: sequence of local computations and propagations over the network

Output layer

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial b_{nk}} \frac{\partial b_{nk}}{\partial w_{kj}} = \delta^w_{nk} z_{nj}$$

$$\delta^w_{nk} = \hat{y}_{nk} - y_{nk}$$

Hidden layer

$$\frac{\partial E_n}{\partial v_{ji}} = \frac{\partial E_n}{\partial a_{nj}} \frac{\partial a_{nj}}{\partial v_{ji}} = \delta^v_{nj} x_{ni}$$

$$\delta^v_{nj} = \sum_k \frac{\partial E_n}{\partial b_{nk}} \frac{\partial b_{nk}}{\partial a_{nj}} = \sum_k \delta^w_{nk} w_{kj} g'(a_{nj})$$

$forward\ function$

$$\begin{tikzcd}
x_i \arrow[r, "v_{ji}"] & a_j \arrow[r, "g"] & z_j \arrow[r, "w_{kj}"] & b_k \arrow[r, "h"] & \hat{y}_k & \\
 & & \delta^v_{nj} & \delta^w_{nk} & & y_k
\end{tikzcd}$$

$backward\ error$

$$\frac{\partial E}{\partial w} = \sum_n \frac{\partial E_n}{\partial w}$$

Backpropagation Algorithm

1. Apply input vector x_n and compute derived variables $a_j, z_j, b_k, \hat{y}_k$
2. Compute δ_{nk}^{w} at all output nodes
3. Back propagate δ_{nk}^{w} to compute δ_{nj}^{v} at all hidden nodes
4. Compute derivatives $\frac{\partial E_n}{\delta w_{kj}}$ and $\frac{\partial E_n}{\delta v_{ji}}$
5. Batch: Sum derivatives over all input vectors

- Vanishing gradient problem

Neural Network Regularization

- Given such a large number of parameters, preventing overfitting is vitally important

- Choosing the number of layers + no of hidden nodes
- Controlling the weights
  - Weight decay
- Early stopping
- Weight sharing
- Structural regularization
  - Convolutional neural networks for invariances in image data

So… Which classifier is the best in practice?

- Low dimensions (9-200)

1. Boosted decision trees
2. Random forests
3. Bagged decision trees
4. SVM
5. Neural nets
6. K nearest neighbors
7. Boosted stumps
8. Decision tree
9. Logistic regression
10. Naïve Bayes

- High dimensions (500-100K)

1. HMC MLP
2. Boosted MLP
3. Bagged MLP
4. Boosted trees
5. Random forests

Usage Notes

- A lot of slides are adopted from the presentations and documents published on internet by experts who know the subject very well.
- I would like to thank who prepared slides and documents.
- Also, these slides are made publicly available on the web for anyone to use
- If you choose to use them, I ask that you alert me of any mistakes which were made and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides.

Sincerely,
Dr. Cahit Karakuş
cahitkarakus@gmail.com