Today

- Continue Multilayer Neural Networks (MNN)
  - Training/testing/validation curves
  - Practical Tips for Implementation
  - Concluding Remarks on MNN
MNN Training

training time

Large training error: in the beginning random decision regions

Small training error: decision regions improve with time

Zero training error: decision regions separate training data perfectly, but we overfitted the network
**MNN Learning Curves**

- **Training data**: data on which learning (gradient descent for MNN) is performed
- **Test data**: used to assess network generalization capabilities
- Training error typically goes down, since with enough hidden units, can find discriminant function which classifies training patterns exactly
- Test error first goes down, but then goes up since at some point we start to **overfit** the network to the training data
Learning Curves

- this is a good time to stop training, since after this time we start to overfit
- However, stopping criterion is part of training phase, we **cannot use test data for anything that has to do with the learning phase**
Learning Curves

- Create a third separate data set called **validation data**:
  - validation data is used to determine “parameters”, in this case when learning should stop

- Stop training after the first local minimum on validation data
  - We are assuming performance on test data will be similar to performance on validation data
Data Sets

- **Training data**
  - data on which learning is performed

- **Validation data**
  - validation data is used to determine any free parameters of the classifier
    - $k$ in the knn neighbor classifier
    - $h$ for parzen windows
    - number of hidden layers in the MNN
    - etc

- **Test data**
  - used to assess network generalization capabilities
Practical Tips for BP: Momentum

- Gradient descent finds only a local minima
  - not a problem if $J(w)$ is small at a local minima. Indeed, we do not wish to find $w$ s.t. $J(w) = 0$ due to overfitting
  - problem if $J(w)$ is large at a local minimum $w$
Practical Tips for BP: Momentum

- Momentum: popular method to avoid local minima and also speeds up descent in plateau regions
  - weight update at time $t$ is $\Delta w^{(t)} = w^{(t)} - w^{(t-1)}$
  - add temporal average direction in which weights have been moving recently

$$w^{(t+1)} = w^{(t)} + (1 - \alpha) \left[ \eta \frac{\partial J}{\partial w} \right] + \alpha \Delta w^{(t-1)}$$

- at $\alpha = 0$, equivalent to gradient descent
- at $\alpha = 1$, gradient descent is ignored, weight update continues in the direction in which it was moving previously (momentum)
- usually, $\alpha$ is around 0.9
Practical Tips for BP: Activation Function

- Gradient descent will work with any continuous $f$ however some choices are better than others
- Desirable properties of $f$:
  - Continuous and differentiable Nonlinearity to express nonlinear decision boundaries
  - Saturation, that is $f$ has minimum and maximum values ($-a$ and $b$). Keeps and weights $w$, $v$ bounded, thus training time down
  - Monotonicity so that activation function itself does not introduce additional local minima
  - Linearity for a small values of net, so that network can produce linear model, if data supports it
  - Antisymmetric, that is $f(-1) = -f(1)$, leads to faster learning
**Practical Tips for BP: Activation Function**

- Sigmoid activation function $f$ satisfies all of the above properties

$$f(\text{net}) = \alpha \frac{e^{\beta \cdot \text{net}} - e^{-\beta \cdot \text{net}}}{e^{\beta \cdot \text{net}} + e^{-\beta \cdot \text{net}}}$$

- Convenient to set $\alpha = 1.716$, $\beta = 2/3$
- Asymptotic values $\pm 1.716$
- Linear range is roughly for $-1 < \text{net} < 1$
Practical Tips for BP: Target Values

- For sigmoid function, to represent class $c$, use
  \[
  t^{(c)} = \begin{bmatrix}
  -1 \\
  1 \\
  -1
  \end{bmatrix}
  \]
  $c$th row

- Always use values less than asymptotic values for target
  - For small error, need $t$ to be close to $z = f(\text{net})$
  - For any finite value of $\text{net}$, $f(\text{net})$ never reaches the asymptotic value
  - The error will always be too large, training will never stop, and weights $w, v$ will go to infinity
Practical Tips for BP: Normalization

- Each feature of input data should be normalized
- Suppose we measure fish length in meters and weight in grams
  - Typical sample [length = 0.5, weight = 3000]
  - Feature length will be basically ignored by the network
  - If length is in fact important, learning will be VERY slow
**Practical Tips for BP: Normalization**

- Normalize each feature $i$ to be of mean $0$ and variance $1$
  - First for each feature $i$, compute $\text{var}[x^{(i)}]$ and $\text{mean}[x^{(i)}]$
  - Then
    \[
    x_k^{(i)} \leftarrow \frac{x_k^{(i)} - \text{mean}(x^{(i)})}{\sqrt{\text{var}(x^{(i)})}}
    \]
  - Cannot do this for online version of the algorithm since data is not available all at once

- If there are a lot of highly correlated or redundant features, can reduce dimensionality with PCA
- Test samples should be subjected to the same transformations as the training samples
Practical Tips for BP: # of Hidden Units

- # of input units = number of features, # output units = # classes. How to choose $N_H$, the # of hidden units?
- $N_H$ determines the expressive power of the network
  - Too small $N_H$ may not be sufficient to learn complex decision boundaries
  - Too large $N_H$ may overfit the training data resulting in poor generalization
Practical Tips for BP: # of Hidden Units

- Choosing $N_H$ is not a solved problem
- Rule of thumb
  - if total number of training samples is $n$, choose $N_H$ so that the total number of weights is $n/10$
  - total number of weights = (# of $w$) + (# of $v$)
- Can choose $N_H$ which gives the best performance on the validation data
Practical Tips for BP: Initializing Weights

- Do not set either $w$ or $v$ to 0
- Rule of thumb for our sigmoid function
  - Choose random weights from the range
    
    $-\frac{1}{\sqrt{d}} < w_{ji} < \frac{1}{\sqrt{d}}$
    
    $-\frac{1}{\sqrt{N_H}} < v_{kj} < \frac{1}{\sqrt{N_H}}$
Practical Tips for BP: Learning Rate

- As any gradient descent algorithm, backpropagation depends on the learning rate $\eta$
- Rule of thumb $\eta = 0.1$
- However we can adjust $\eta$ at the training time
- The objective function $J$ should decrease during gradient descent
  - If it oscillates, $\eta$ is too large, decrease it
  - If it goes down but very slowly, $\eta$ is too small, increase it
Practical Tips for BP: Weight Decay

- To simplify the network and avoid overfitting, it is recommended to keep the weights small.
- Implement weight decay after each weight update:
  \[ w_{\text{new}} = w_{\text{old}} (1 - \varepsilon), \quad 0 < \varepsilon < 1 \]
- Additional benefit is that “unused” weights grow small and may be eliminated altogether.
  - A weight is “unused” if it is left almost unchanged by the backpropagation algorithm.
Practical Tips for BP: # Hidden Layers

- Network with 1 hidden layer has the same expressive power as with several hidden layers.
- For some applications, having more than 1 hidden layer may result in faster learning and less hidden units overall.
- However, networks with more than 1 hidden layer are more prone to the local minima problem.
MNN as Nonlinear Mapping

This module implements nonlinear input mapping $\varphi$

This module implements linear classifier (Perceptron)
Thus MNN can be thought as learning 2 things at the same time:

- the nonlinear mapping of the inputs
- linear classifier of the nonlinearly mapped inputs
MNN as Nonlinear Mapping

Original feature space \( \mathbf{x} \); patterns are not linearly separable

MNN finds nonlinear mapping \( \mathbf{y} = \phi(\mathbf{x}) \) to 2 dimensions (2 hidden units); patterns are almost linearly separable

MNN finds nonlinear mapping \( \mathbf{y} = \phi(\mathbf{x}) \) to 3 dimensions (3 hidden units) that; patterns are linearly separable
Concluding Remarks

- **Advantages**
  - MNN can learn complex mappings from inputs to outputs, based only on the training samples
  - Easy to use
  - Easy to incorporate a lot of heuristics

- **Disadvantages**
  - It is a “black box”, that is difficult to analyze and predict its behavior
  - May take a long time to train
  - May get trapped in a bad local minima
  - A lot of “tricks” to implement for the best performance