Neural Networks

Course 22525

Koen Van Leemput
DTU HealthTech
Technical University of Denmark
Course structure

Fitting functions

Registration

Segmentation
Remember regression?

- \[ f(x) = \sum_{m=1}^{M} \beta_m \phi_m(x) \]

- Training set \( \{x_i, y_i\}_{i=1}^{N} \)
  
  Input vector: \( x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T \)
  
  Corresponding output: \( y_i \)

- Estimate parameters \( \theta = (\beta_1, \ldots, \beta_M)^T \)
  by minimizing the cost
  \[ \sum_{i=1}^{N} (y_i - f(x_i))^2 \]
Remember regression?

Example: $p=1$ and $M=5$ cosines

$$\phi_m(x)$$

$$f(x) = \sum_{m=1}^{M} \beta_m \phi_m(x)$$
Remember Gaussian mixture model?

Posterior using Bayes’ rule:

\[
p(l = k|d, \theta) = \frac{\mathcal{N}(d|\mu_k, \sigma_k^2)\pi_k}{\sum_{k'} \mathcal{N}(d|\mu_{k'}, \sigma_{k'}^2)\pi_{k'}}
\]
Remember Gaussian mixture model?

- "Training samples"
- "y=1" if l=1
- "y=0" if l=2

Posterior using Bayes’ rule: 
\[ p(l = k|d, \theta) = \frac{\mathcal{N}(d|\mu_k, \sigma^2_k)\pi_k}{\sum_{k'} \mathcal{N}(d|\mu_{k'}, \sigma^2_{k'})\pi_{k'}} \]
Remember Gaussian mixture model?

This lecture: can we get a “classifier” directly without a model?

Posterior using Bayes’ rule: \[ p(l = k|d, \theta) = \frac{\mathcal{N}(d|\mu_k, \sigma^2_k)\pi_k}{\sum_{k'} \mathcal{N}(d|\mu_{k'}, \sigma^2_{k'})\pi_{k'}} \]

New notation/terminology:
- “Training samples”
- “y=1” if \( l=1 \)
- “y=0” if \( l=2 \)
Logistic regression

- Logistic function as a “squashing” function

\[ \sigma(a) = \frac{1}{1 + \exp(-a)} \]
Logistic regression

\[ a = \sum_{m=1}^{M} \beta_m \phi_m(x) \]

\[ \sigma(a) = \frac{1}{1 + \exp(-a)} \]

- \( p(y = 1|x, \theta) = f(x) \) where \( f(x) = \sigma \left( \sum_{m=1}^{M} \beta_m \phi_m(x) \right) \)

- Of course: \( p(y = 0|x, \theta) = 1 - p(y = 1|x, \theta) = 1 - f(x) \)
Voxel-based classifier

- Training data \( \{x_i, y_i\}_{i=1}^{N} \) with \( x_i = d_i \) (i.e., \( p = 1 \)) and \( y_i \in \{0, 1\} \)
- Estimate parameters \( \theta = (\beta_1, \ldots, \beta_M)^T \)
  by maximizing the likelihood function

\[
\prod_{i=1}^{N} p(y_i | x_i, \theta)
\]
Voxel-based classifier

- Training data \( \{x_i, y_i\}_{i=1}^{N} \) with \( x_i = d_i \) (i.e., \( p = 1 \)) and \( y_i \in \{0, 1\} \)
- Estimate parameters \( \theta = (\beta_1, \ldots, \beta_M)^T \)
  by maximizing the likelihood function
  \[
  \prod_{i=1}^{N} p(y_i | x_i, \theta)
  \]
Voxel-based classifier

- Training data \( \{x_i, y_i\}_{i=1}^{N} \) with \( x_i = d_i \) (i.e., \( p = 1 \)) and \( y_i \in \{0, 1\} \)
- Estimate parameters \( \theta = (\beta_1, \ldots, \beta_M)^T \) by maximizing the likelihood function

\[
\prod_{i=1}^{N} p(y_i | x_i, \theta)
\]
Voxel-based classifier

- Training data \( \{x_i, y_i\}_{i=1}^{N} \)
  
  with \( x_i = d_i \) (i.e., \( p = 1 \)) and \( y_i \in \{0, 1\} \)

- Estimate parameters \( \theta = (\beta_1, \ldots, \beta_M)^T \)
  
  by maximizing the likelihood function

\[
\prod_{i=1}^{N} p(y_i | x_i, \theta)
\]
Voxel-based classifier

- Once trained keep the classifier
  \[ p(l = 1|d, \hat{\theta}) \]

- Simply apply it to new data
Optimization algorithm for training

- Maximizing the likelihood function \( \prod_{i=1}^{N} p(y_i | x_i, \theta) \) is equivalent to minimizing

\[
E_N(\theta) = -\log \prod_{i=1}^{N} p(y_i | x_i, \theta) = -\sum_{i=1}^{N} \{y_i \log f(x_i) + (1 - y_i) \log [1 - f(x_i)]\}
\]

- Gradient descent: 
  \[
  \theta^{(\tau+1)} = \theta^{(\tau)} - \nu \nabla E_N(\theta^{(\tau)}) \\
  \text{with gradient } \nabla E_N(\theta) = \frac{\partial E_N}{\partial \theta}
  \]

- Stochastic gradient descent: use only \( N' \ll N \) randomly sampled training points, and approximate:

\[
\nabla E_N(\theta) \approx \frac{N}{N'} \nabla E_{N'}(\theta)
\]
More fun: patch-based classifier

- Classify 3x3 image “patches”:
  intensity of the pixel to be classified +
  intensities of 8 neighboring pixels

- \( \mathbf{x} \) is now a 9-dimensional vector \( (p = 9) \),
  but otherwise everything is the same:
  \[
p(y = 1|\mathbf{x}, \hat{\theta}) = \sigma \left( \sum_{m=1}^{M} \hat{\beta}_m \phi_m(\mathbf{x}) \right)
  \]

- But how to choose basis functions \( \phi_m(\mathbf{x}) \)
  in a 9-dimensional space?
Basis functions in high dimensions?

- Idea: remember the tensor B-spline trick?

Example: take outer products of four 1D basis functions to “make” sixteen 2D basis functions

- Does this work in 9D?
Basis functions in high dimensions?

- Idea: remember the tensor B-spline trick?

Example: take outer products of four 1D basis functions to "make" sixteen 2D basis functions

Does this work in 9D?

No! $4^9 = 262144$ basis functions!
Adaptive basis functions

- Introduce extra parameters that alter the form of a limited set of basis functions

- Prototypical example:

  \[
  \phi_m(x) = \begin{cases} 
  1 & \text{if } m = 1, \\
  \sigma \left( \sum_{j=1}^{p} w_{m,j} x_j + w_{m,0} \right) & \text{otherwise},
  \end{cases}
  \]

- All parameters (\(\{w_{m,j}\}\) and \(\{\beta_m\}\)) optimized together during training (stochastic gradient descent)
Adaptive basis functions

- Introduce extra parameters that alter the form of a limited set of basis functions

- Prototypical example:

\[
\phi_m(x) = \begin{cases} 
  1 & \text{if } m = 1, \\
  \sigma \left( \sum_{j=1}^{p} w_{m,j} x_j + w_{m,0} \right) & \text{otherwise,}
\end{cases}
\]

- All parameters (\( \{w_{m,j}\} \) and \( \{\beta_m\} \)) optimized together during training (stochastic gradient descent)
Adaptive basis functions (p=1)

$\sigma(1x + 0)$
Adaptive basis functions (p=1)

\[ \sigma(0.3x + 0) \]
Adaptive basis functions ($p=1$)

$\sigma(1x + 10.5)$
Adaptive basis functions (p=2)

\[ \sigma(1x_1 + 0x_2 + 0) \]

\[ \sigma(0.707x_1 - 0.707x_2 + 0) \]

\[ \sigma(1x_1 + 0x_2 - 11.5) \]

\[ \sigma(6x_1 - 6x_2 + 0) \]
Feed-forward neural network

So the model is

$$p(y = 1 | \mathbf{x}, \theta) = \sigma \left( \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x}) \right)$$

with basis functions

$$\phi_m(\mathbf{x}) = \begin{cases} 1 & \text{if } m = 1, \\ \sigma \left( \sum_{j=1}^{p} w_{m,j} x_j + w_{m,0} \right) & \text{otherwise}, \end{cases}$$
Feed-forward neural network

Graphical representation of our 3x3 patch-based classifier:
(p=9 and M=4)

- Can insert more than one “hidden” layer (“deep learning”)

flow of information
Applying the trained classifier on new data:

\[ p(y = 1 | x, \hat{\theta}) \]
Applying the trained classifier on new data:

\[ p(y = 1 | x, \hat{\theta}) \]

\[ \{w_{2,1}, \ldots, w_{2,9}\} \]

\[ \phi_2(x) \]

\[ \{w_{3,1}, \ldots, w_{3,9}\} \]

\[ \phi_3(x) \]

\[ \{w_{4,1}, \ldots, w_{4,9}\} \]

\[ \phi_4(x) \]
Filtering operations can be implemented using convolutions

\[ p(y = 1 | x, \hat{\theta}) \]

Applying the trained classifier on new data:

\[ \{ w_{2,1}, \ldots, w_{2,9} \} \]

\[ \phi_2(x) \]

\[ \{ w_{3,1}, \ldots, w_{3,9} \} \]

\[ \phi_3(x) \]

\[ \{ w_{4,1}, \ldots, w_{4,9} \} \]

\[ \phi_4(x) \]
Neural networks = ultimate solution?

No model, only training data:

- No domain expertise needed
- Very easy to train and deploy
- Super fast (GPUs)

- Training data often very hard to get in medical imaging!
- Scanning hardware/software/protocol changes routinely!