Neural Networks, Chapter 11 in ESL II

STK-IN4300
Statistical Learning Methods in Data Science
Odd Kolbjørnsen, oddkol@math.uio.no
Learning today Neural nets

• Projection pursuit
  – What is it?
  – How to solve it: Stagewise

• Neural nets
  – What is it?
  – Graphical display
  – Connection to Projection pursuit
  – How to solve it: Backpropagation
  – Stochastic Gradient decent
  – Deep and wide
  – CNN

• Example
Neural network

- Used for prediction
- Universal approximation
  - with enough data and the correct algorithm you will get it right eventually…
- Used for both «regression type» and «classification» type problems
- Many versions and forms, currently deep learning is a hot topic
- Often portrayed as fully automatic, but tailoring might help
- Perform highly advanced analysis
- Can create utterly complex models which are hard to decipher and hard to use for knowledge transfer.
- The network provide good prediction, but is it for the right reasons?

Constructed example from:
Ribeiro et.al (2016)“Why Should I Trust You?” Explaining the Predictions of Any Classifier

25. September 2019
STK-IN 4300 Lecture 4- Neural nets
In neural nets training is based on minimization of a loss function over the training set

General form

\[ L \left( Y, \hat{f}(X) \right) = \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \]

- Target might be multi dimensional
  \[ y_i = [y_{i1}, ..., y_{iK}]^T \]
- Continuous response («regression type»)
  Squared error (common)
  \[ L \left( Y, \hat{f}(X) \right) = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - \hat{f}_k(x_i))^2 \]
- Discrete (K –classes)
  Squared error
  \[ L \left( Y, \hat{f}(X) \right) = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - \hat{f}_k(x_i))^2 \]
  Cross-entropy or deviance
  \[ L \left( Y, \hat{f}(X) \right) = \sum_{i=1}^{N} \sum_{k=1}^{K} -\log \hat{f}_k(x_i) \cdot y_{ik} \]

\[ \hat{f}_k(x_i) \approx \text{Prob}(y_{ik} = 1) \]

\[ y_{ik} = \begin{cases} 0 & \text{if } y_i \neq k \\ 1 & \text{if } y_i = k \end{cases} \]
Projection pursuit Regression

\[
f(X) = \sum_{m=1}^{M} g_m(w_m^T X)
\]

Unknown functions \((\mathbb{R} \rightarrow \mathbb{R})\)

Derived feature (number \(m\)), \(V_m = w_m^T X\), (size \(1 \times 1\))

Unknown Weight (size \(p \times 1\))

Unit vector

Features = Explanatory Variables (size \(p \times 1\))

Ridge functions are constant along directions orthogonal to the directional unit vector \(w\)

Friedman and Tukey (1974)
Friedman and Stuetzle (1981)
Fitting Projection pursuit: M=1

- M = 1 model, known as the single index model in econometrics:
  \[ f(X) = g(w^T X) = g(V), \quad V = w^T X \]

- If \( w \) is known fitting \( \hat{g}(v) \) is just a 1D smoothing problem
  - Smoothing spline, Local linear (or polynomial) regression, Kernel smoothing, K-nearest…

- If \( g(\cdot) \) is known fitting \( \hat{w} \) is obtained by quasi-Newton search
  \[ g(w^T x_i) \approx g(w_{\text{old}}^T x_i) + g'(w_{\text{old}}^T x_i)(w - w_{\text{old}})^T x_i \]

  - Minimize the objective function (with approximation inserted)
    \[
    \sum_{i=1}^{N} (y_i - g(w^T x_i))^2 \approx \sum_{i=1}^{N} (y_i - g(w_{\text{old}}^T x_i) - g'(w_{\text{old}}^T x_i)(w - w_{\text{old}})^T x_i)^2 \\
    = \sum_{i=1}^{N} g'(w_{\text{old}}^T x_i)^2 \left( \frac{y_i - g(w_{\text{old}}^T x_i)}{g'(w_{\text{old}}^T x_i)} + w_{\text{old}}^T x_i - w^T x_i \right)^2
    
    \]

Solve for \( w \) using weighted regression:
weight = \( g'(w_{\text{old}}^T x_i)^2 \)
Fitting Projection pursuit, $M > 1$

- Stage-wise (greedy)
  - Set $y_{i,1} = y_i$
  - For $m = 1, \ldots, M$
    - Assume there is just one function to match (as previous page)
    - Minimize Loss with respect to $y_{i,m}$ to obtain $g_m(\cdot)$ and $w_m$

\[
[\hat{g}_m(\cdot), \hat{w}_m] = \underset{g_m(\cdot),w_m}{\text{argmin}} \sum_{i=1}^{N} (y_{i,m} - g_m(w_m^T x_i))^2
\]

- Store $\hat{g}_m(\cdot)$ and $\hat{w}_m$
- Subtract estimate from data $y_{i,m+1} = y_{i,m} - \hat{g}_m(\hat{w}_m^T x_i)$

- Final prediction:

\[
\hat{f}(X) = \sum_{m=1}^{M} \hat{g}_m(\hat{w}_m^T X)
\]
Implementation details

1. Need a smoothing method with efficient evaluation of $g(v)$ and $g'(v)$
   - Local regression or smoothing splines

2. $g_m(v)$ from previous steps can be readjusted using a backfitting procedure (Chapter 9), but it is unclear if this improves the performance
   1. Set $r_i = y_i - \hat{f}(x_i) + \hat{g}_m(\hat{w}_m x_i)$
   2. Re-estimate $g_m(\cdot)$ from $r_i$. (and center the results)
   3. Do this repeatedly for $m = 1, ... M, 1 ... M, ...$

3. It is not common to readjust $\hat{w}_m$, as this is computationally demanding

4. Stopping criterion for number of terms to include.
   1. When the model does not improve appreciably
   2. Use cross validation to determine M
Example

\[ Y = \sigma(a_1^T X) + (a_2^T X)^2 + 0.30 \cdot Z, \]

\[ a_1 = (3, 3), \quad a_2 = (3, -3); \]

- **Train data:** 1000
- **Two terms:**

Call:
```
ppr(formula = y ~ x1 + x2, data = trainData, nterms = 2)
```

Goodness of fit:
- 2 terms
- 2049.644

Projection direction vectors:
- term 1 term 2
  - x1 -0.7060166 0.7347320
  - x2 0.7081953 0.6783575

Coefficients of ridge terms:
- term 1 term 2
  - 26.9347577 0.4455549
Neural network

- Simplified model of a nerve system

Perceptron:

\[ v = \sum_{i=0}^{p} \alpha_i x_i \]

\[ \sigma(v) \]

\[ \sigma \left( \sum_{i=0}^{p} \alpha_i x_i \right) \]
Activation functions

never • Initially: The binary step function used

• Next: Sigmoid = Logistic = Soft step

• Now: there is a «rag bag» of alternatives some more suited than others for specific tasks

#2 – ArcTan

#1 – Rectified linear ReLu
(and variants)

– Gaussian (NB not monotone gives different behavior)

Illustrations from: https://en.wikipedia.org/wiki/Activation_function
Single layer feed-forward Neural nets

\[ f(X) = \sum_{m=1}^{M} \beta_m \sigma(\alpha_m^T X + \alpha_0) \]

- Activation function \( (\mathbb{R} \rightarrow \mathbb{R}) \)
- Derived feature (number \( m \)), \( Z_m = \alpha_m^T X + \alpha_0 \), (size \( 1 \times 1 \) )
- «Bias» or «Shift»
- Unknown Weight (size \( p \times 1 \))
- Not unit vector
- Feature = Explanatory variables (size \( p \times 1 \))

**Sigmoid**

\[ \sigma(s \cdot v) \]

- \( s=1 \)
- \( s=0.5 \)
- \( s=10 \)

**PP – Feature**

\[ \sigma(\alpha^T X + \alpha_0) = \sigma(s_m \cdot \omega_m^T X + \alpha_0) = \sigma(s_m \cdot \mathbf{Y}_m + \alpha_0) \]

\[ \omega_m = \frac{\alpha_m}{s_m}, \quad s_m = \| \alpha_m \| \]
Graphical display of single hidden layer feed forward neural network

\[ f_k(X) = \sum_{m=1}^{M} \beta_{k,m} \sigma(\alpha_m^T X + \alpha_0) \]

We will however traverse the graph in the opposite direction as well ....

**Note!**
With respect to model definition

Feed forward means:
- Connections in the graph are directional
- The direction goes from input to output

**FIGURE 11.2.** Schematic of a single hidden layer, feed-forward neural network.
Output layer is often «different»

Hidden layer:

\[ Z_m = \sigma(\alpha_{0,m} + \alpha_m^T X), \quad m = 1, \ldots, M \]

Output layer:

\[ T_k = \beta_{0,k} + \beta_k^T Z, \quad k = 1, \ldots, K \]

Some alternatives for \( f_k() \):

<table>
<thead>
<tr>
<th>Transform</th>
<th>( \sigma(T_k) )</th>
<th>Same as «hidden» layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>( T_k )</td>
<td>Common in regression setting</td>
</tr>
<tr>
<td>Joint transform</td>
<td>( g_k(T) )</td>
<td>Common for classification, e.g. softmax</td>
</tr>
</tbody>
</table>

Identity

\[
\begin{align*}
    f_k(X) &= T_k \\
    &= \beta_{0,k} + \sum_{m=1}^{M} \beta_{k,m} \sigma(\alpha_m^T X + \alpha_{0,m})
\end{align*}
\]

Softmax

\[
\begin{align*}
    f_k(X) &= \frac{\exp(T_k)}{\sum_{j=1}^{K} \exp(T_j)} \\
    &= \frac{\exp(\beta_{0,k} + \beta_k^T Z)}{\sum_{j=1}^{K} \exp(\beta_{0,j} + \beta_j^T Z)}
\end{align*}
\]
Comparision Projection pursuit (PP) and Neural nets (NN)

\[
f(X) = \sum_{m=1}^{M_{PP}} g_m(w_m^T X) \quad \text{vs} \quad f(X) = \sum_{m=1}^{M_{NN}} \beta_m \sigma(\alpha_m^T X + \alpha_0)
\]

\[
g_m(w_m^T X) \quad \text{vs} \quad \beta_m \sigma(s_m \cdot w_m^T X + \alpha_0)
\]

\[s_m = ||\alpha||\]

- The flexibility of \( g_m \) is much larger than what is obtained with \( s_m \) and \( \alpha_0 \) which are the additional parameters of neural nets
- There are usually less terms in PP than NN, i.e. \( M_{PP} \ll M_{NN} \)
- Both methods are powerful for regression and classification
- Effective in problems with high signal to noise ratio
- Suited for prediction without interpretation
- Identifiability of weights an open question and creates problems in interpretations
- The fitting procedures are different
Fitting neural networks

\[ \theta: \text{Statistical slang for all parameters} \]

Here:

\[ \{ \alpha_{0,m}, \alpha_m \}, \# \text{parameters: (} p + 1 \text{)} M \]

\[ \{ \beta_{0,k}, \beta_k \}, \# \text{parameters: (} M + 1 \text{)} K \]

\[ R(\theta) = L(Y, \hat{f}(X)) \]

Quadratic loss

\[ K \text{ output varaibles} = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - \hat{f}_k(x_i))^2 \]

\[ = \sum_{i=1}^{N} R_i(\theta) \]

Contribution of the i'th data record

\[ R_i(\theta) = \sum_{k=1}^{K} (y_{ik} - \hat{f}_k(x_i))^2 \]

The “standard” approach:

- Minimize the loss
- Use steepest decent to solve this minimization problem

- The key to success is the efficient way of computing the gradient
Steepest decent

- Minimize $R(\theta)$ wrt $\theta$,
  - Initialize: $\theta^{(0)}$
  - Iterate:

$$
\theta_j^{(r+1)} = \theta_j^{(r)} - \gamma_r \left. \frac{\partial R(\theta)}{\partial \theta_j} \right|_{\theta=\theta^{(r)}}
$$

Learning rate

$$
\frac{\partial R(\theta)}{\partial \theta_j} = \sum_{i=1}^{N} \frac{\partial R_i(\theta)}{\partial \theta_j}
$$

we compute term per data record

(easily aggregated from parallel computation)
Squared error loss

Output layer:

\[
\frac{\partial R_{i}(\theta)}{\partial \beta_{k,m}} = -2 \left(y_{i,k} - f_{k}(x_{i})\right) g'_{k} (\beta_{k} z_{i}) z_{m,i} \\
= \delta_{k,i} \cdot z_{m,i}
\]

Hidden layer:

\[
\frac{\partial R_{i}(\theta)}{\partial \alpha_{m,l}} = - \sum_{k=1}^{K} 2(y_{ik} - f_{k}(x_{i})) g'_{k} (\beta_{k} z_{i}) \beta_{km} \sigma'(\alpha_{m} x_{i}) x_{i,l} \\
= s_{m,i} \cdot \sigma'(\alpha_{m} x_{i}) x_{i,l}
\]

Back propagation equation

\[
s_{m,i} = \sigma'(\alpha_{m} x_{i}) \sum_{k=1}^{K} \beta_{km} \delta_{k,i}
\]
Backpropagation (delta rule)

• At top level. compute:
  \[ \delta_{k,i} = -2 \left(y_{i,k} - f_k(x_i)\right) g'_k \left(\beta_k^T z_i\right), \quad \forall (i, k) \]

• At hidden level, compute:
  \[ s_{m,i} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{k,m} \delta_{k,i}, \quad \forall (i, m) \]

• Evaluate:
  \[ \frac{\partial R_i(\theta)}{\partial \beta_{k,m}} = \delta_{k,i} z_{m,i} \quad \& \quad \frac{\partial R_i(\theta)}{\partial \alpha_m^l} = s_{m,i} x_{i,l} \]

• Update: \( \gamma_r \) is fixed
  \[ \beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{k,m}} \bigg|_{\theta=\theta(r)} \]
  \[ \alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_m^l} \bigg|_{\theta=\theta(r)} \]
Stochastic gradient decent

\[ \beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{k,m}} \bigg|_{\theta=\theta^{(r)}} \quad \alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m,l}} \bigg|_{\theta=\theta^{(r)}} \]

- Equations above updates with all data at the same time
- The form invites to update estimate using fractions of data
  - Perform a random partition of training data in to batches: \( \{B_j\}_{j=1}^{\text{#Batches}} \)
  - For all batches cycle over the data in this batch to update data
    \[ \beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i \in B_j} \frac{\partial R_i}{\partial \beta_{k,m}} \bigg|_{\theta=\theta^{(r)}} \quad \alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i \in B_j} \frac{\partial R_i}{\partial \alpha_{m,l}} \bigg|_{\theta=\theta^{(r)}} \]
  - Repeat

- One **iteration** is one update of the parameter (using one batch)
- One **Epoch** is one scan through all data (using all batches in the partition)
Online learning  (Extreme case Batch size =1)

- Learning based on one data point at the time

\[
\beta_{k,m}^{(r)} = \beta_{k,m}^{(r-1)} - \gamma_r \frac{\partial R_i}{\partial \beta_{k,m}} \bigg|_{\theta=\theta^{(r-1)}}
\]

\[
\alpha_{m,l}^{(r)} = \alpha_{m,l}^{(r-1)} - \gamma_r \frac{\partial R_i}{\partial \alpha_{m,l}} \bigg|_{\theta=\theta^{(r-1)}}
\]

- You might re-iterate (for several epochs) when completed or if you have an abundance of data just take on new data as they come along (hence the name)

- For convergence: \( \gamma_r \to 0 \), as \( \sum \gamma_r \to \infty \) and \( \sum \gamma_r^2 < \infty \),
e.g. \( \gamma_r = \frac{1}{r} \)
Other methods can be used

• Still use the Backpropagation to get the derivative
• Conjugate Gradient
  – Method for minimizing a quadratic form
  – Need «restart» for nonlinear problems
• Variable metric methods
  – E.g. Quasi newton methods
Neural networks

• Deep neural nets are currently «hot-topic»
• Deep means many hidden layers
• Multilayer feed forward
  – Characteristics
    • Network is arranged in layers,
      – first layer taking input
      – last layer outputs
      – Intermediate layers are hidden layers (no connection to the world outside)
    • A node in one layer is connected to every node in the next layer (Fully connected)
    • There are no connections among nodes in the same layer

• Other types:
  – Self organizing map (SOM), output is not defined (unsupervised)
  – Recurrent neural network (RNN), many forms
  – Hopfield networks (RNN with symmetric connections)
  – Boltzmann machine networks (Markov random fields)
  – Convolutional neural nets (locally connected)
Graphical display of feed forward neural network

Depth of NN = Q hidden layers

Width of NN may vary from layer to layer

Input

Hidden layer 1

Hidden layer 2

Hidden layer 3

Hidden layer Q

Output

\[ \begin{align*}
X_1 & \quad Z_1^1 \quad Z_1^2 \quad Z_1^3 \quad \ldots \quad Z_1^Q \quad Y_1 \\
X_2 & \quad Z_2^1 \quad Z_2^2 \quad Z_2^3 \quad \ldots \quad Z_2^Q \quad Y_2 \\
X_3 & \quad Z_3^1 \quad Z_3^2 \quad Z_3^3 \quad \ldots \quad Z_3^Q \quad Y_K \\
\vdots & \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
X_p & \quad Z_{q_1}^1 \quad Z_{q_2}^2 \quad Z_{q_3}^3 \quad \ldots \quad Z_{q_Q}^Q \quad \beta
\end{align*} \]

\[ \begin{align*}
\alpha^1 & \quad \sigma^1(\cdot) \quad \alpha^2 \quad \sigma^2(\cdot) \quad \alpha^3 \quad \sigma^3(\cdot) \quad \ldots \quad \alpha^Q \quad \sigma^Q(\cdot) \quad \beta
\end{align*} \]
Graphical display of feed forward neural network

Depth of NN = Q hidden layers

Input \( x \) \( \rightarrow \) Hidden layer 1 \( z^1 \) \( \rightarrow \) Hidden layer 2 \( z^2 \) \( \rightarrow \) Hidden layer 3 \( z^3 \) \( \cdots \) \( \rightarrow \) Hidden layer Q \( z^Q \) \( \rightarrow \) Output \( y \)

Matrix notation:
Size of \( W^1: ((p + 1) \times q_1) \)
Size of \( W^r: ((q_{r-1} + 1) \times q_r) \)
Bias

\[
\begin{align*}
z^r &= W^r z^{r-1}
\end{align*}
\]
Use computational graphs

(Do not try to write this in a closed form… )

\[ f_k(X) = \beta_{0,k} + \sum_{m=1}^{q_Q} \beta_{m,k} Z^{[Q]} \]

\[ Z^{[Q]} = \sigma^Q \left( \alpha_i^{[Q]} \cdot Z^{[Q-1]} + \alpha_0^{[Q]} \right) \]

size: \((1 \times q_Q-1)\)

\[ Z_i^{[Q-1]} = \sigma^{Q-1} \left( \alpha_i^{[Q-1]} \cdot Z^{[Q-2]} + \alpha_0^{[Q-1]} \right) \]

\[ Z_i^{[1]} = \sigma^1 \left( \alpha_i^1 \cdot X + \alpha_0^1 \right) \]

Number of outputs

\( k = 1, \ldots, K \)

Width of layer \( Q \)

\( i = 1, \ldots, q_Q \)

Width of layer \( 1 \)

\( i = 1, \ldots, q_1 \)

Number of inputs

Size: \((1 \times q_1)\)

Size: \((1 \times p)\)

Number of input

Size: \((1 \times q_Q-1)\)
Training Neural networks

- Backpropagation can still be used

Traversing the graph backwards,
(record number \((i)\) suppressed)

\[
\delta^{[\text{Top}]}_k = -2(y_k - f_k(x))g'_k(\beta^T_k z^{[Q]})
\]

\[
\delta^{[Q]}_m = \sigma^{[Q]}'(\alpha^{[Q]}_m z^{[Q-1]}) \sum_{k=1}^{K} \beta_{k,m}\delta^{\text{Top}}_k , \quad m = 1, \ldots, q_Q
\]

\[
\delta^{[Q-1]}_l = \sigma^{[Q-1]}'(\alpha^{[Q-1]}_m z^{[Q-2]}) \sum_{m=1}^{q_Q} \alpha^{[Q]}_{m,l}\delta^{[Q]}_m , \quad l = 1, \ldots, q_{Q-1}
\]
Scaling of input & Starting values

• Standardize input variables to avoid numerical scaling issues
  • Mean 0
  • Standard deviation 1

• Choose weights
  – Close to zero (model almost linear)
  – Do not choose zero (then it does not get started)
  – Too large values generally gives bad results
  – Common to use several random starting point

• Weights rule of thumb: (With standardized input )
  – Book suggest: weights from a uniform distribution uniform in \([-0.7 \; 0.7]\]
  – Also common (ReLu): weights \( N \left( 0, \sqrt{\frac{2}{\text{#input features}}} \right) \)
Avoiding overfitting

• Early stopping
  – Since starting regime is close to linear we will usually end up with something close to linear if we stop early
  – Use a validation set to select when to stop

• Regularization by weight decay
  – Minimize: \( R(\theta) + \lambda J(\theta) \)
  
  e.g. \( J(\theta) = \sum \alpha^2_{m,l} + \sum \beta^2_{k,m} \)
  
  – Use cross validation to select \( \lambda \)

• Dropouts
  – During training set \( x_{m,l} = 0 \) with probability \( p \)
  – During evaluation weight is set to \( \alpha_{m,l} \cdot p \)

Srivastava, et al. (2014)
Dropout: A Simple Way to Prevent Neural Networks from Overfitting
Effect of weight decay

Neural Network - 10 Units, No Weight Decay

No weight decay

Training Error: 0.100
Test Error: 0.259
Bayes Error: 0.210

Neural Network - 10 Units, Weight Decay=0.02

Weight decay

Training Error: 0.100
Test Error: 0.223
Bayes Error: 0.210
Multiple Minima

- $R(\theta)$ is not convex and have many local minima
- Try many starting configurations
  - choose solution with lowest penalize error
  - Use the average prediction of the collection of networks
    - NB do not average weights as these are not well ordered
- Use bagging, i.e. average predictions of networks trained from random perturbations of training data
Number of hidden Units and layers

• Number of units
  – From 5-100 units is common
  – Increase with number of input and training data
  – Better to have too many than to few
  – Start with a large number and use weight decay (regularization)

• Number of hidden layers
  – Guided by background knowledge
  – Models hierarchical features at different levels of resolution
  – Trial and error
  – Trend in CNN smaller filters more layers
What is Deep learning

• There is no universally agreed upon threshold of depth dividing shallow learning from deep learning, but most researchers in the field agree that deep learning has multiple nonlinear layers (CAP > 2), 3 layers and more.

• «Deep learning» Hinton et al 2006 (3 layers)
• «Very Deep learning» Simonyan et al. 2014 (16+ layers)
• «Extremely Deep» He et al 2016 (50 -> 1000)
• Schmidhuber 2015 considers more than 10 layers to be very deep learning
Example simulated data

Sum of sigmoids: \[ Y = \sigma(a_1^T X) + \sigma(a_2^T X) + \varepsilon_1; \]

- \( \frac{\text{Var}(f(X))}{\text{Var}(\varepsilon_1)} \approx 4 \)
- Training data size: 100 samples
- Test data size: 10,000 samples
Weight decay vs number of hidden units

Both the approach to select a high number of functions and optimize the weight decay and the approach to fix the weight decay and optimize the number of hidden units gives good result.
Examples of simulated data

\[
\text{Radial: } Y = \prod_{m=1}^{10} \phi(X_m) + \varepsilon_2. \\
\phi(t) = \left(\frac{1}{2\pi}\right)^{1/2} \exp\left(-\frac{t^2}{2}\right)
\]

- \( \frac{\text{Var}(f(X))}{\text{Var}(\varepsilon)} \approx 4 \)
- Training data size: 100 samples
- Test data size : 10 000 samples

NN does not always work:
All cases are worse than the mean
And it gets even worse as the number of units increase
“Hand crafting» NN might help
By reducing the number of parameters to be estimated.

- Setting weights to zero (localize)
- Setting weights equal (Convolutional NN)
Neural net reduction in number of parameters
(time series example)

Layer i

\[ x \]

Layer i+1

\[ z = Wx \]

Fully connected

\[ W \]

(100 × 100)

# 10100

(100+1) × 100

Localized (±4)

1000

(9+1) × 100

Convolutional (±4)

10

(9+1) × 1

CNN use multiple features
10 features => # = 100 param

Bias

25. September 2019

STK-IN 4300 Lecture 4- Neural nets
**Fully Connected Layer (FC layer)**
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

**# filters?**

**Size of filter?**

**Subsample?**

**Padding?**

**Activation function?**

**POOLING layer?**

---

Series:
- https://www.youtube.com/watch?v=vT1JzLTH4G4&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv

# Layers

Per Layer (CONV)

# filters?

Subsample?

Padding?

Activation function?
Learning today Neural nets

• Projection pursuit
  – What is it?
  – How to solve it: Stagewise

• Neural nets
  – What is it?
  – Graphical display
  – Connection to Projection pursuit
  – How to solve it: Backpropagation
  – Stochastic Gradient search
  – Deep – wide – convolutional