



UiO : **Matematisk institutt**

Det matematisk-naturvitenskapelige fakultet

Neural Networks, Chapter 11 in ESL II

STK-IN4300

Statistical Learning Methods in Data Science

Odd Kolbjørnsen, oddkol@math.uio.no



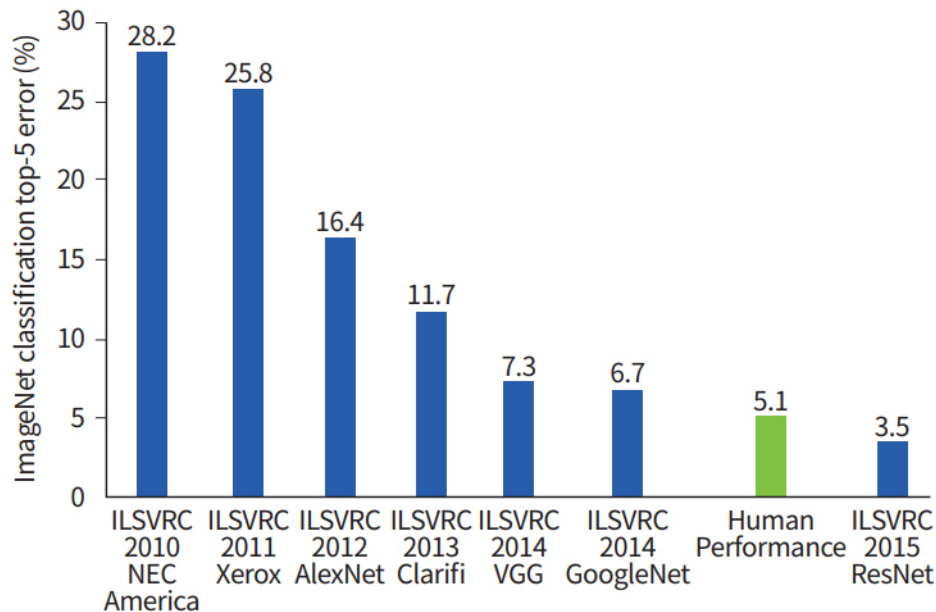
The fuzz of Neural nets (CNN)

ImageNet:

1000K images (with manual labels)

20K categories

Predict the correct among top 5



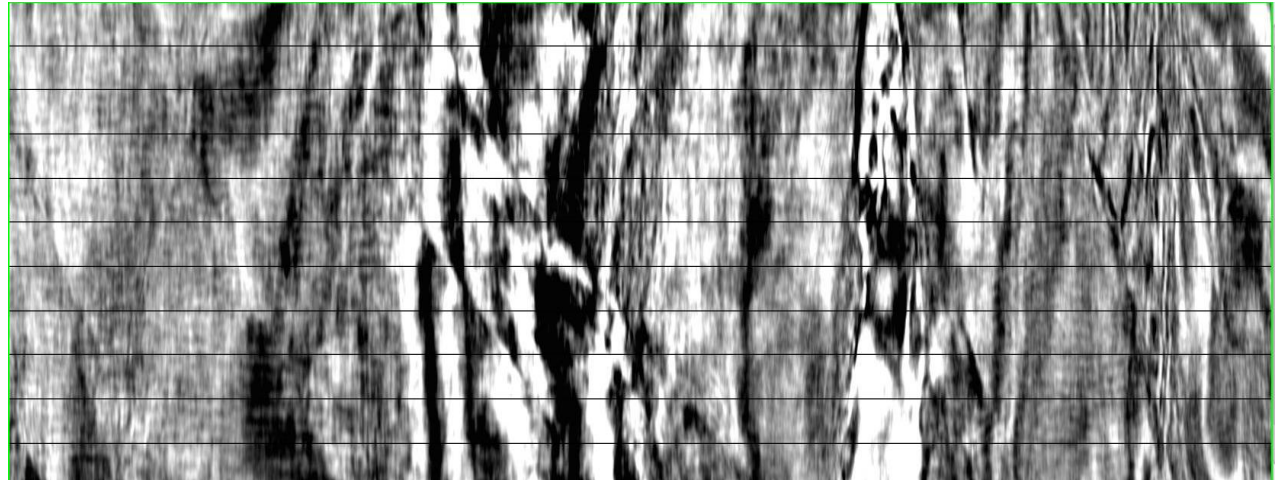
Questions:

- Which other problems does it solve?
- Why does it work?
- Can the solutions be trusted?
 - Stability?
 - Deep fool
- Modern day alchemy?

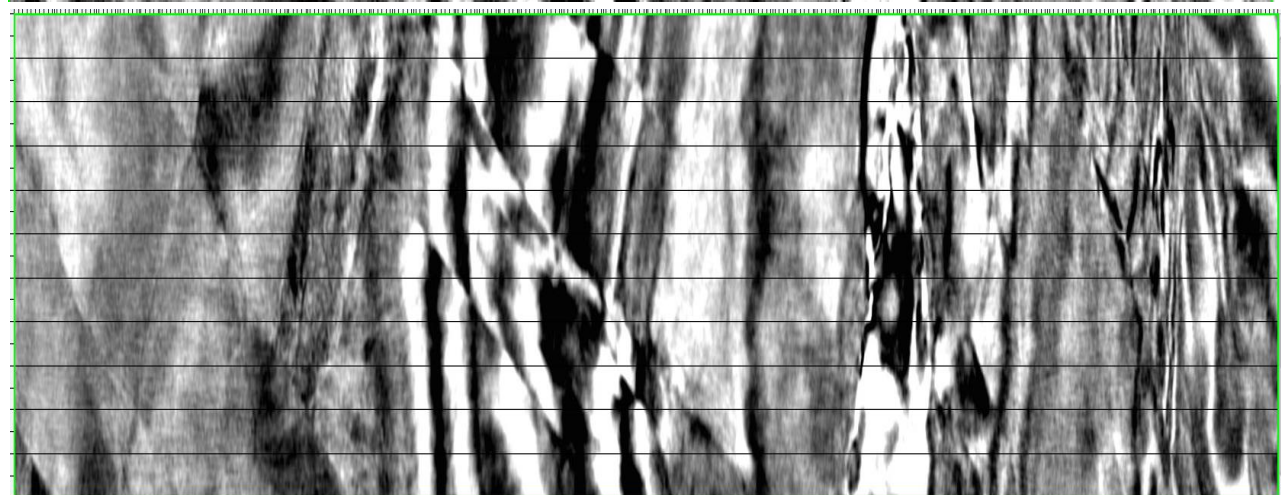
IMAGE from: <https://www.pfmjournal.org/journal/Figure.php?xn=pfm-2018-00030.xml&id=>

Example of seismic processing (Greiner et al 2021)

Industry
standard



CNN
Technology



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?

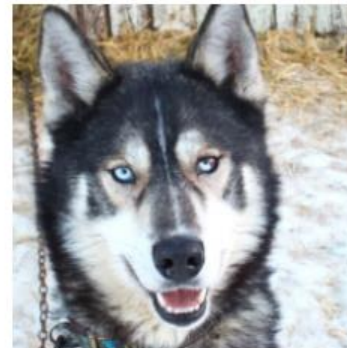
Constructed example from:
Ribeiro et.al (2016)“Why Should I Trust You?” Explaining
the Predictions of Any Classifier

Regression

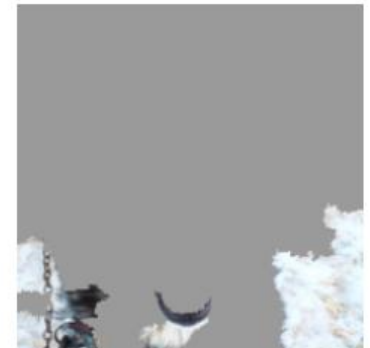
$$y_i = f(x_i) + \varepsilon_i$$

Classification

$$P(y_i = k|x_i) = f_k(x_i)$$



(a) Husky classified as wolf



(b) Explanation

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

Projection pursuit Regression

Friedman and Tukey (1974)
Friedman and Stuetzle (1981)

$$f(X) = \sum_{m=1}^M g_m(\underbrace{w_m^T X}_{\substack{\text{Derived feature (number } m), \\ V_m = w_m^T X, \text{ (size } 1 \times 1)}})$$

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

Unknown Weight (size $p \times 1$) Unit vector

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

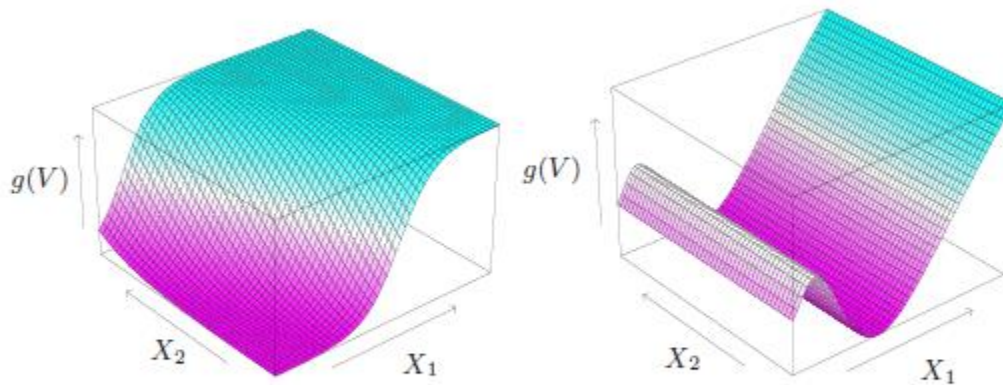


FIGURE 11.1. Perspective plots of two ridge functions.
(Left:) $g(V) = 1/[1 + \exp(-5(V - 0.5))]$, where $V = (X_1 + X_2)/\sqrt{2}$.
(Right:) $g(V) = (V + 0.1) \sin(1/(V/3 + 0.1))$, where $V = X_1$.

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

Fitting Projection pursuit: M=1

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

- Iterate
- 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 $g()$ is known fitting $\hat{w}$ is obtained by quasi-Newton method
 - $g(w^T x_i) \approx g(w_{\text{old}}^T x_i) + g'(w_{\text{old}}^T x_i)(w - w_{\text{old}}^T)^T x_i$

- Minimize the objective function (with approximation inserted)

$$\sum_{i=1}^N (y_i - g(w^T x_i))^2 \approx \sum_{i=1}^N \left(y_i - g(w_{\text{old}}^T x_i) - g'(w_{\text{old}}^T x_i)(w - w_{\text{old}}^T)^T x_i \right)^2$$

Only unknown

$$= \sum_{i=1}^N g'(w_{\text{old}}^T x_i)^2 \left(\frac{y_i - g(w_{\text{old}}^T x_i)}{g'(w_{\text{old}}^T x_i)} + w_{\text{old}}^T x_i - w^T x_i \right)^2$$

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 $m = 1, \dots, M$

- Assume there is just one function to match (as previous page)

- Minimize Loss with respect to $y_{i,m}$ to obtain $g_m()$ and w_m

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

- 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:

$$\hat{f}(X) = \sum_{m=1}^M \hat{g}_m(\hat{w}_m^T X)$$

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

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 $m = 1, \dots, M, 1 \dots M, \dots$
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

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

$$a_1 = (3, 3), \quad a_2 = (3, -3);$$

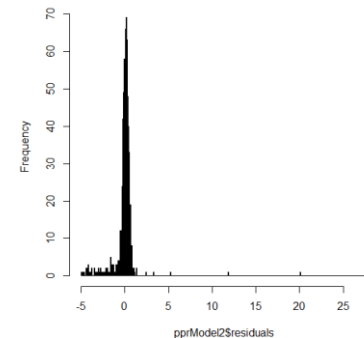
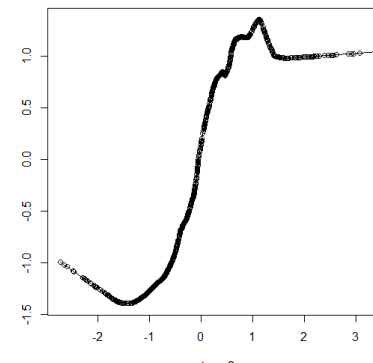
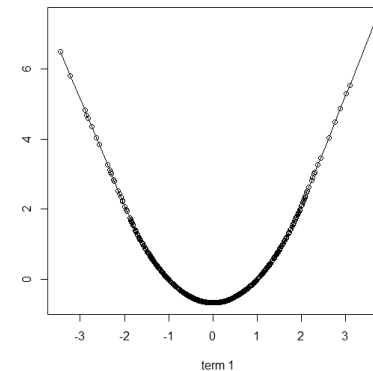
- Train data: 1000
- Two terms:

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

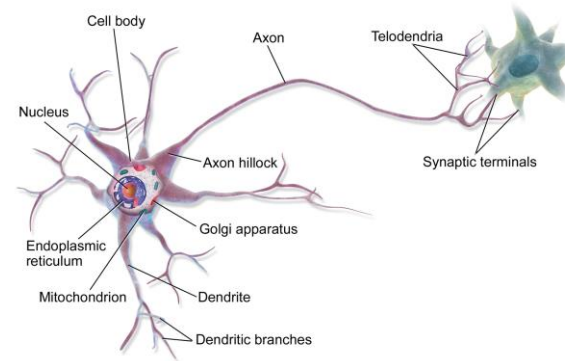
Goodness of fit:
 2 terms
2049.644

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
```

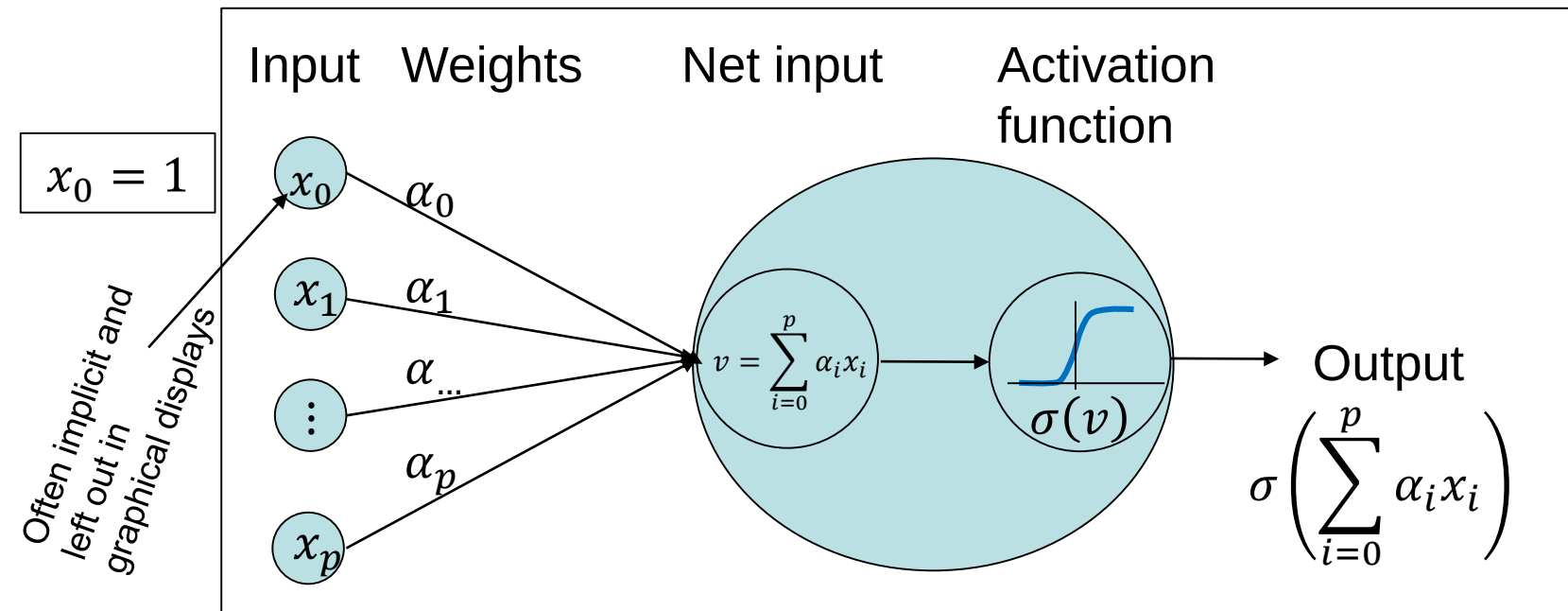


Neural network



- Simplified model of a nerve system

Perceptron:



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

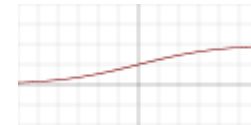
#2 – ArcTan

#1 – Rectified linear ReLu

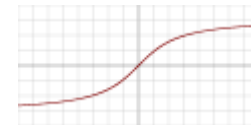
(and variants)

- Gaussian (NB not monotone gives different behavior)

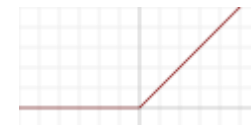
$$\sigma(v)$$



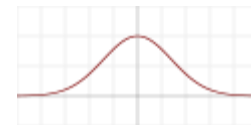
Smooth



Smooth



Continuous



Smooth

Illustrations from:

https://en.wikipedia.org/wiki/Activation_function

Single layer feed-forward Neural nets

$$f(X) = \sum_{m=1}^M \beta_m \sigma(\underbrace{\alpha_m^T X + \alpha_{0,m}}_{Z_m = \alpha_m^T X + \alpha_{0,m}, \text{ (size } 1 \times 1 \text{)}})$$

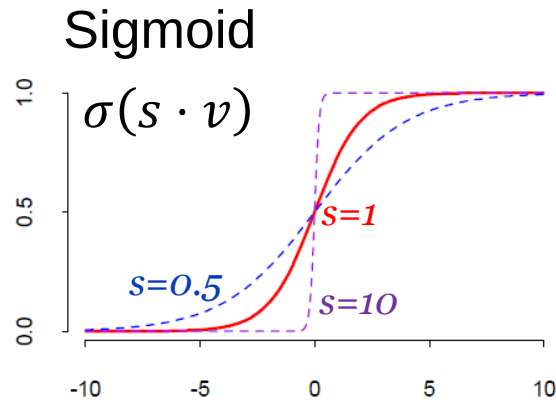
Activation function ($\mathbb{R} \rightarrow \mathbb{R}$)

Derived feature (number m),
 $Z_m = \alpha_m^T X + \alpha_{0,m}$, (size 1×1)

Unknown Weight (size $(p \times 1)$)
Not unit vector

Feature = Explanatory variables (size $p \times 1$)

«Bias» or «Shift»



$$\sigma(\alpha^T X + \alpha_{0,m}) = \sigma(\underbrace{s_m}_{\text{scale}} \cdot \underbrace{w_m^T X}_{\text{unit vector}} + \alpha_{0,m}) = \sigma(s_m \cdot V_m + \alpha_{0,m})$$

$$w_m = \frac{\alpha_m}{s_m}, \quad s_m = \|\alpha_m\|$$

«PP – Feature»

Graphical display of single hidden layer feed forward neural network

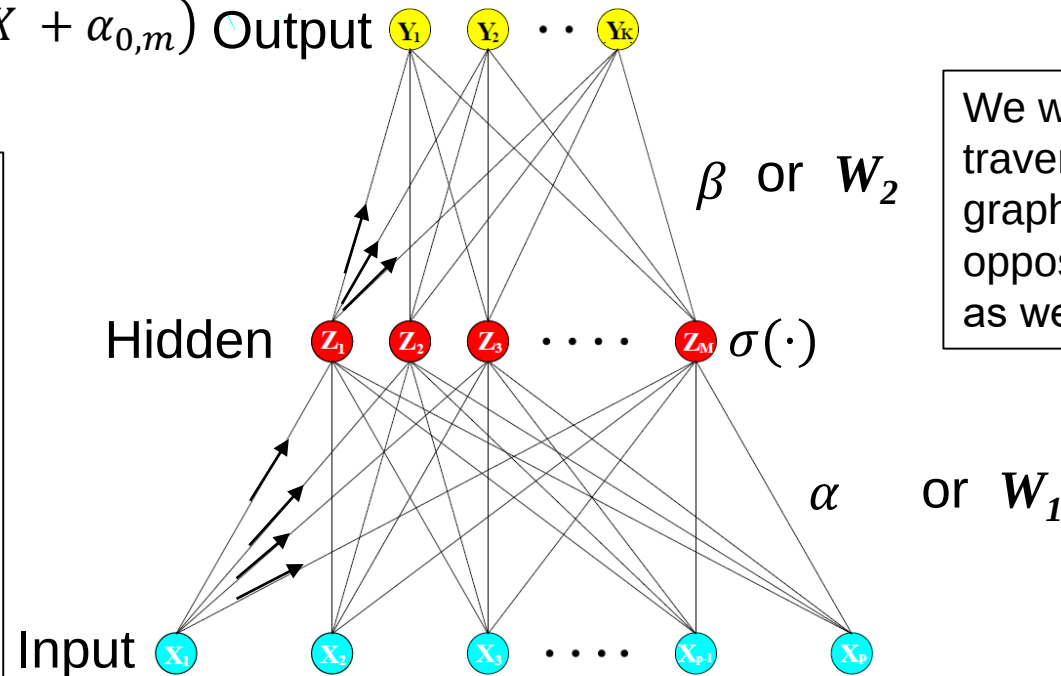
$$f_k(X) = \sum_{m=1}^M \beta_{k,m} \sigma(\alpha_m^T X + \alpha_{0,m})$$

11.3 Neural Networks 393

Note!
With respect to model definition

Feed forward means:

- Connections in the graph are directional
- The direction goes from input to output

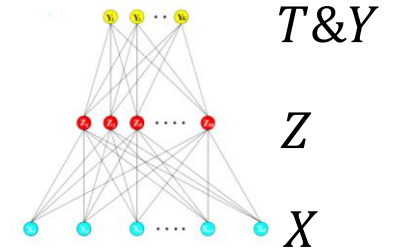


We will however traverse the graph in the opposite direction as well

FIGURE 11.2. Schematic of a single hidden layer, feed-forward neural network.

Output layer is often «different»

Hidden layer: $Z_m = \sigma(\alpha_{0,m} + \alpha_m^T X), \quad m = 1, \dots, M$
 Output layer: $T_k = \beta_{0,k} + \beta_k^T Z, \quad k = 1 \dots K$



Some alternatives for $f_k()$:

Transform	$\sigma(T_k)$	Same as «hidden» layers
Identity	T_k	Common in regression setting
Joint transform	$g_k(T)$	Common for classification, e.g. softmax

Identity

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

Softmax

$$\begin{aligned} f_k(X) &= \frac{\exp(T_k)}{\sum_{j=1}^K \exp(T_j)} \\ &= \frac{\exp(\beta_{0,k} + \beta_k^T Z)}{\sum_{j=1}^K \exp(\beta_{0,j} + \beta_j^T Z)} \end{aligned}$$

Comparison Projection pursuit (PP) and Neural nets (NN)

$$f(X) = \sum_{m=1}^{M_{PP}} g_m(w_m^T X)$$

$$f(X) = \sum_{m=1}^{M_{NN}} \beta_m \sigma(\alpha_m^T X + \alpha_0)$$

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

$$s_m = \|\alpha\|$$

- The flexibility of g_m is much larger than what is obtained with s_m and α_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

In neural nets training is based on minimization of a loss function over the training set

General form

$$L(Y, \hat{f}(X)) = \sum_{i=1}^N L(y_i, \hat{f}(x_i))$$

Neural nets are defined by a specific form of the model $f(X)$

- Target might be multi dimensional

$$y_i = [y_{i1}, \dots, y_{iK}]^T$$

- Continuous response («regression type»)

Squared error
(common)

$$L(Y, \hat{f}(X)) = \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - \hat{f}_k(x_i))^2$$

- Discrete (K –classes)

Squared error

$$L(Y, \hat{f}(X)) = \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - \hat{f}_k(x_i))^2$$

Cross-entropy
or deviance

$$L(Y, \hat{f}(X)) = \sum_{i=1}^N \sum_{k=1}^K -\log \hat{f}_k(x_i) \cdot y_{ik}$$

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

$$y_{ik} = \begin{cases} 0 & \text{if } y_i \neq k \\ 1 & \text{if } y_i = k \end{cases}$$

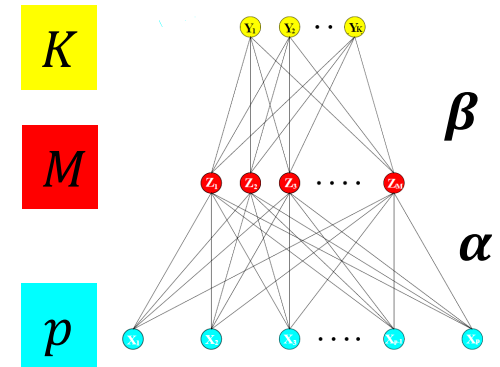
Fitting neural networks

θ : Statistical slang for all parameters

Here:

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

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



Quadratic loss
K output variables

$$\begin{aligned} R(\theta) &= L(Y, \hat{f}(X)) \\ &= \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - \hat{f}_k(x_i))^2 \\ &= \sum_{i=1}^N R_i(\theta) \end{aligned}$$

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 θ ,
 - Initialize: $\theta^{(0)}$
 - Iterate:

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

$$\theta_j^{(r+1)} = \theta_j^{(r)} - \underset{\substack{\uparrow \\ \text{Learning rate}}}{\gamma_r} \left. \frac{\partial R(\theta)}{\partial \theta_j} \right|_{\theta = \theta^{(r)}}$$

$$\frac{\partial R(\theta)}{\partial \theta_j} = \sum_{i=1}^N \frac{\partial R_i(\theta)}{\partial \theta_j} \quad \begin{array}{l} \text{we compute term per data record} \\ \text{(easily aggregated from} \\ \text{parallel computation)} \end{array} \quad \frac{\partial R_i(\theta)}{\partial \theta_j}$$

Squared error loss

$$R_i(\theta) = \sum_{k=1}^K (y_{ik} - \hat{f}_k(x_i))^2 \quad f_k(X) = g_k(T_k) \quad T_k = \sum_{m=1}^M \beta_{k,m} \overbrace{\sigma(\alpha_m^T X + \alpha_0)}^{z_m}$$

Output
layer:

$$\begin{aligned} \frac{\partial R_i(\theta)}{\partial \beta_{k,m}} &= \underbrace{-2(y_{i,k} - f_k(x_i)) g'_k(\beta_k^T z_i)}_{\delta_{k,i}} \cdot \underbrace{z_{m,i}}_{z_{m,i}} \\ &= \delta_{k,i} \cdot z_{m,i} \end{aligned}$$

Hidden
layer:

$$\begin{aligned} \frac{\partial R_i(\theta)}{\partial \alpha_{m,l}} &= - \sum_{k=1}^K 2(y_{ik} - f_k(x_i)) g'_k(\beta_k^T z_i) \beta_{km} \underbrace{\sigma'(\alpha_m^T x_i) x_{i,l}}_{x_{i,l}} \\ &= \underbrace{- \sum_{k=1}^K 2(y_{ik} - f_k(x_i)) g'_k(\beta_k^T z_i) \beta_{km}}_{s_{m,i}} \cdot x_{i,l} \end{aligned}$$

Back
propagation
equation

$$s_{m,i} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^K \beta_{km} \delta_{k,i}$$

Backpropagation (delta rule)

- At top level. compute:

$$\delta_{k,i} = -2 \left(y_{i,k} - f_k(x_i) \right) g'_k(\beta_k^T z_i), \quad \forall(i, k)$$

- 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 : γ_r is fixed

$$\beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \beta_{k,m}} \Big|_{\theta=\theta^{(r)}}$$

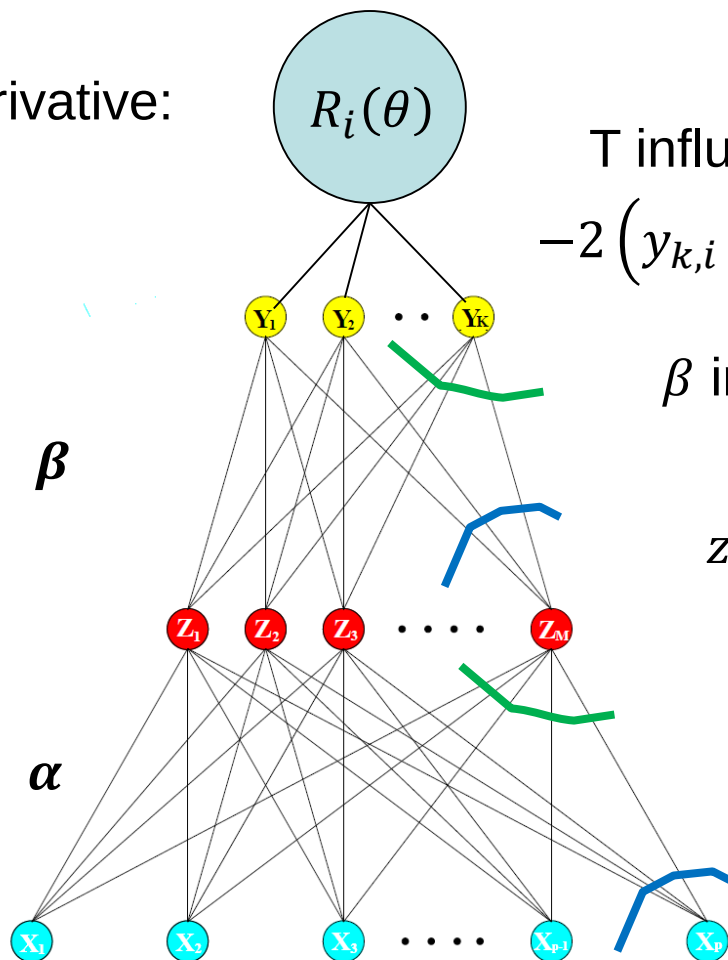
$$\alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \alpha_{m,l}} \Big|_{\theta=\theta^{(r)}}$$

Want the derivative:

K

M

p



T influence by its direct effect on R

$$-2 \left(y_{k,i} - g_k(T_{k,i}) \right) g'_k(T_{k,i}) = \delta_{k,i}$$

β influence through its effect on T

$$\delta_{k,i} z_{m,i}$$

z_m influence through
its sum of effects on T

$$s_{m,i} = \sigma'(z_m) \sum_{k=1}^K \beta_{k,m} \delta_{k,i}$$

α influence through its effect on Z

$$s_{m,i} x_{p,i}$$

x_p influence through
its sum of effects on Z

Stochastic gradient decent

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

- 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

$$\beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i \in B_j} \frac{\partial R_i}{\partial \beta_{k,m}} \Big|_{\theta=\theta^{(r)}} \quad \alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i \in B_j} \frac{\partial R_i}{\partial \alpha_{m,l}} \Big|_{\theta=\theta^{(r)}}$$

– 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

$$\beta_{k,m}^{(r)} = \beta_{k,m}^{(r-1)} - \gamma_r \left. \frac{\partial R_i}{\partial \beta_{k,m}} \right|_{\theta = \theta^{(r-1)}}$$
$$\alpha_{m,l}^{(r)} = \alpha_{m,l}^{(r-1)} - \gamma_r \left. \frac{\partial R_i}{\partial \alpha_{m,l}} \right|_{\theta = \theta^{(r-1)}}$$

- 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: $\gamma_r \rightarrow 0$, as $\sum \gamma_r \rightarrow \infty$ and $\sum \gamma_r^2 < \infty$,
e.g. $\gamma_r = \frac{1}{r}$

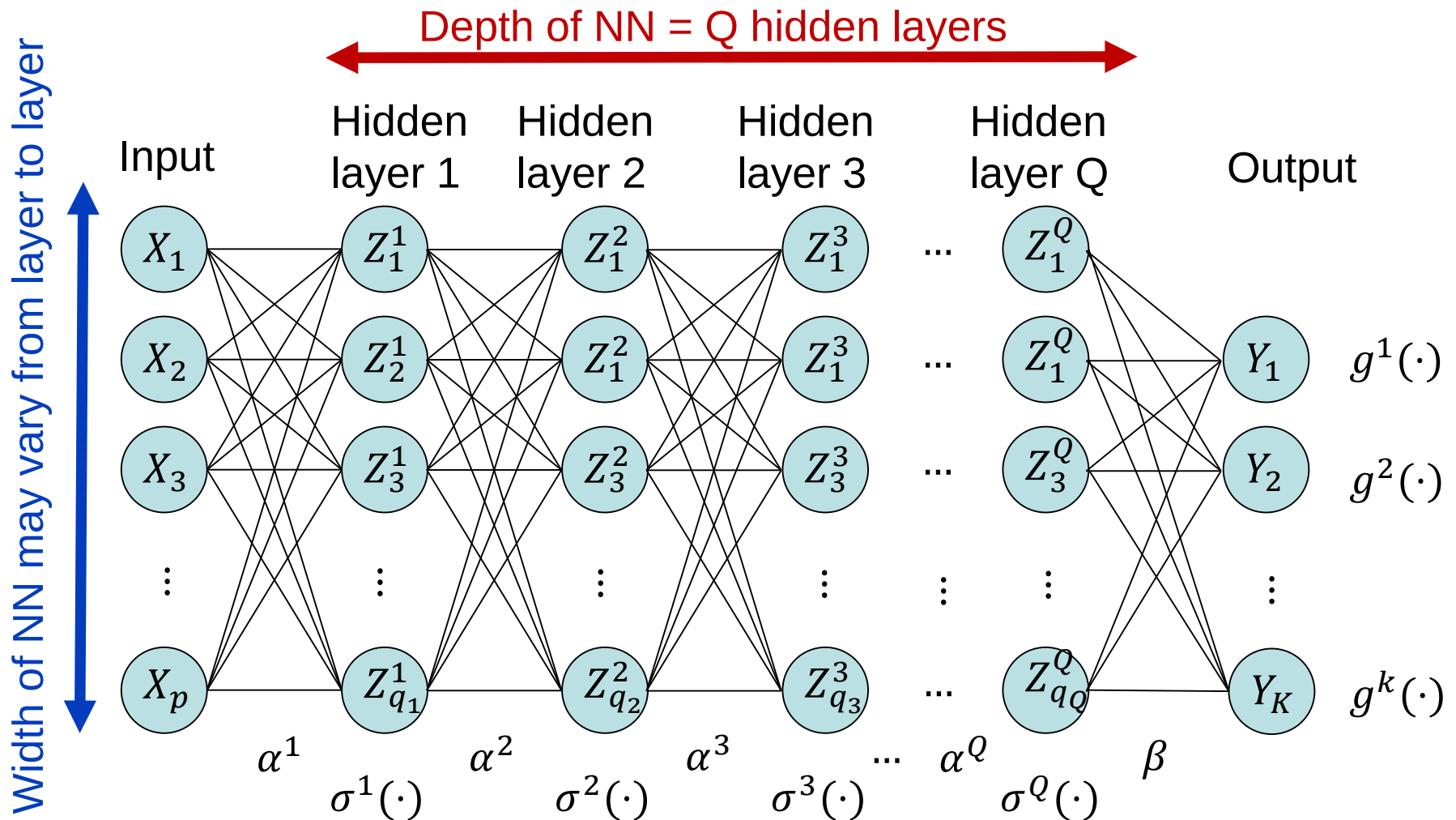
Other methods can be used

- 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

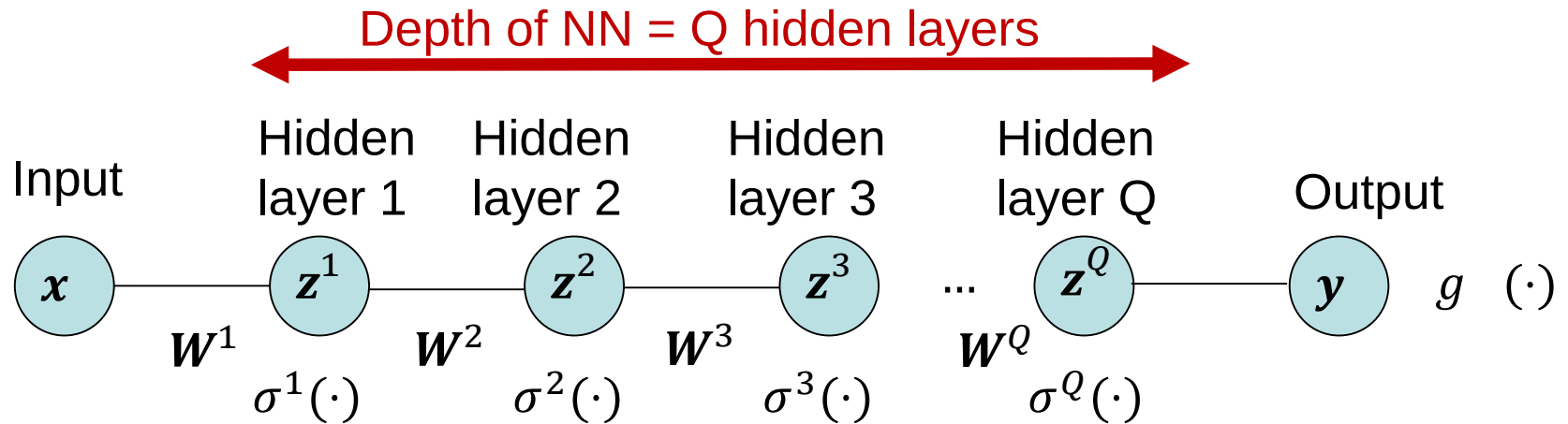
Neural networks

- 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



Graphical display of feed forward neural network



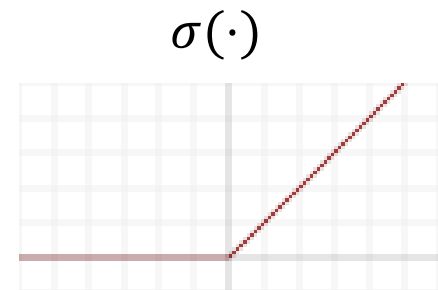
Matrix notation:

Size of W^1 : $((p + 1) \times q_1)$

Size of W^r : $((q_{r-1} + 1) \times q_r)$

Bias →

$$z^r = \sigma(a), a = W^r z^{r-1}$$



Nested definition Use computational graphs (Do not try to write this in a closed form...)

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

Number of outputs
 $k = 1, \dots, K$

$$Z_i^{[Q]} = \sigma^Q \left(\underset{\substack{\uparrow \\ \text{size: } (1 \times q_{Q-1})}}{\alpha_i^{[Q]T}} \cdot \underset{\substack{\uparrow \\ \text{size: } (1 \times 1)}}{Z^{[Q-1]}} + \alpha_{0,i}^{[Q]} \right)$$

Width of layer Q
 $i = 1, \dots, q_Q$

$$Z_i^{[Q-1]} = \sigma^{Q-1} \left(\alpha_i^{[Q-1]T} \cdot Z^{[Q-2]} + \alpha_{0,i}^{[Q-1]} \right) \quad i = 1, \dots, q_{Q-1}$$

$\vdots$

$$Z_i^{[1]} = \sigma^1 \left(\underset{\substack{\uparrow \\ \text{size: } (1 \times p)}}{\alpha_i^{1T}} \cdot \underset{\substack{\uparrow \\ \text{size: } (1 \times 1)}}{X} + \alpha_{0,i}^1 \right)$$

Width of layer 1
 $i = 1, \dots, q_1$

Number of input

Training Neural networks

- Backpropagation can still be used
Traversing the graph backwards,
(record number (i) suppressed)

$$\delta_k^{[\text{Top}]} = -2(y_k - f_k(x))g'_k(\beta_k^T z^{[Q]})$$

$$\delta_m^{[Q]} = \sigma^{[Q]'} \left(\alpha_m^{[Q]T} z^{[Q-1]} \right) \sum_{k=1}^K \beta_{k,m} \delta_k^{\text{Top}}, \quad m = 1, \dots, q_Q$$

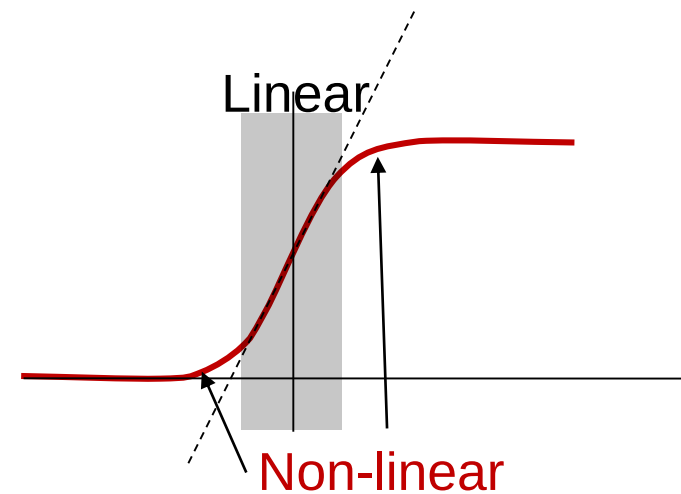
$$\delta_l^{[Q-1]} = \sigma^{[Q-1]'} \left(\alpha_m^{[Q-1]T} z^{[Q-2]} \right) \sum_{m=1}^{q_Q} \alpha_{m,l}^{[Q]} \delta_m^{[Q]}, \quad l = 1, \dots, q_{Q-1}$$

Scaling of input & Starting values

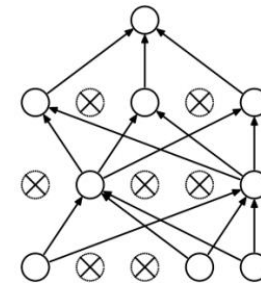
- Standardize input variables to avoid numerical scaling issues
 - Mean 0
 - Standard deviation 1
- Choose weights
 - Close to zero (model almost linear)
 - Do not choose zero (then it does not get started)
 - Too large values generally gives bad results
 - Common to use several random starting point
- Weights rule of thumb: (With standardized input)
 - Book suggest: weights from a uniform distribution uniform in $[-0.7 \ 0.7]$
 - Also common (ReLu): weights $N\left(0, \sqrt{\frac{2}{\text{\#input features}}}\right)$

[Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun](#) (2015)
Delving Deep into Rectifiers: Surpassing Human-Level
Performance on ImageNet Classification

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. $J(\theta) = \sum \alpha_{m,l}^2 + \sum \beta_{k,m}^2$ ← Penalty term
 - Use cross validation to select λ
- Dropouts
 - During training set $x_{m,l} = 0$ with probability p
 - During evaluation weight is set to $\alpha_{m,l} \cdot p$

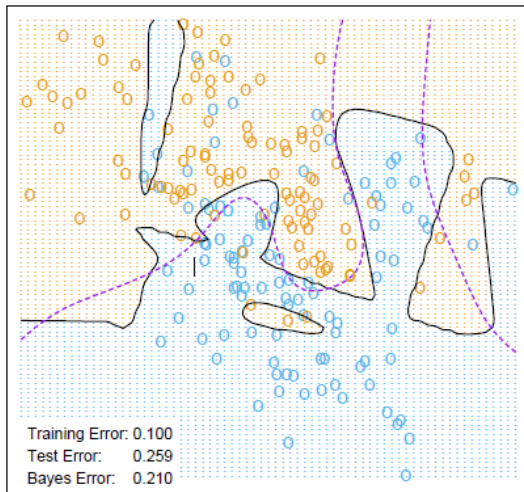


Srivastava, et al. (2014)
Dropout: A Simple Way to Prevent Neural Networks from Overfitting
Journal of Machine Learning Research 15 (2014) 1929-1958

Effect of weight decay

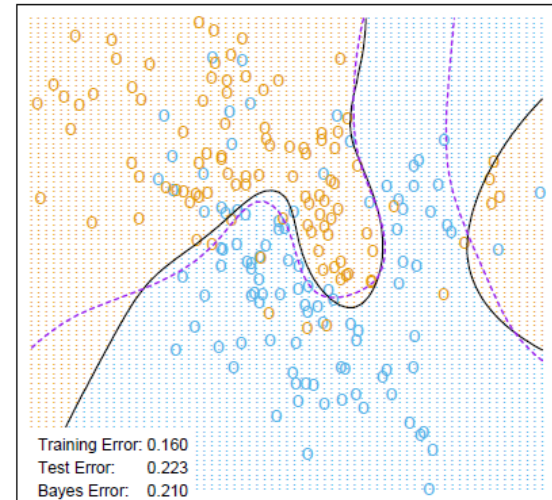
Prediction

Neural Network - 10 Units, No Weight Decay



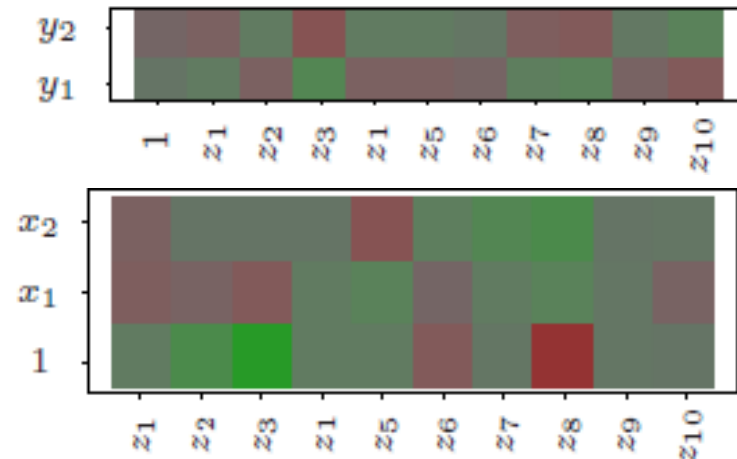
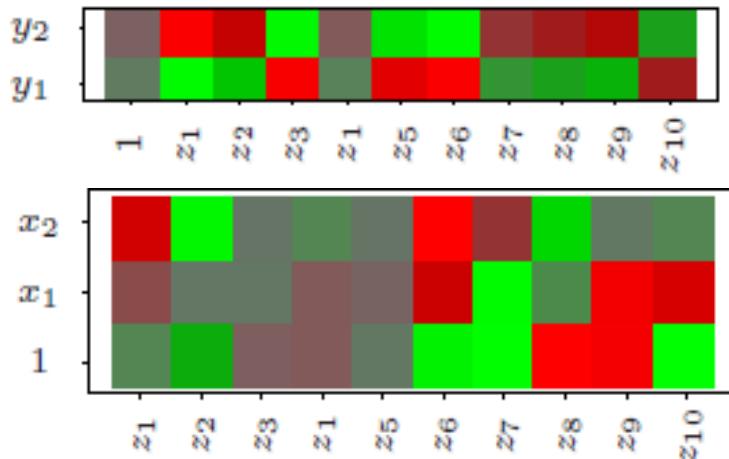
No weight decay

Neural Network - 10 Units, Weight Decay=0.02



Weight decay

Weights



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

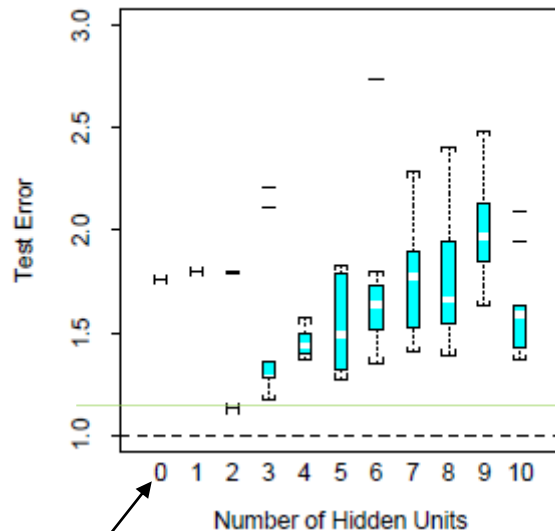
- There is no universally agreed upon threshold of depth dividing shallow learning from deep learning, but most researchers in the field agree that deep learning has multiple nonlinear layers ($CAP > 2$), 3 layers and more.
- «Deep learning» Hinton et al 2006 (3 layers)
- «Very Deep learning» Simonyan 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

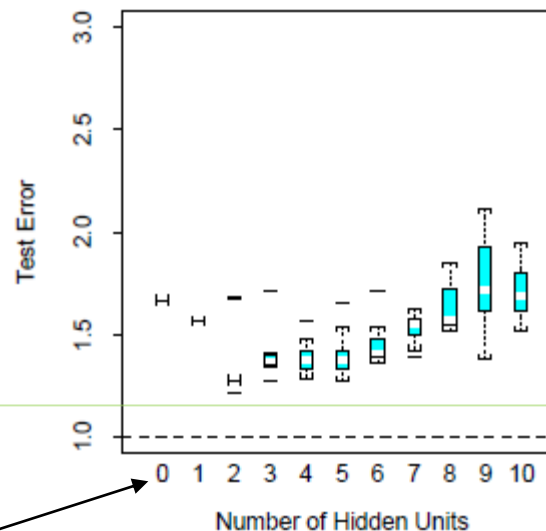
Sum of sigmoids: $Y = \sigma(a_1^T X) + \sigma(a_2^T X) + \varepsilon_1;$

- $\frac{\text{Var}(f(X))}{\text{Var}(\varepsilon_1)} \approx 4$ $a_1 = (3, 3), a_2 = (3, -3);$
- Training data size: 100 samples
- Test data size : 10 000 samples

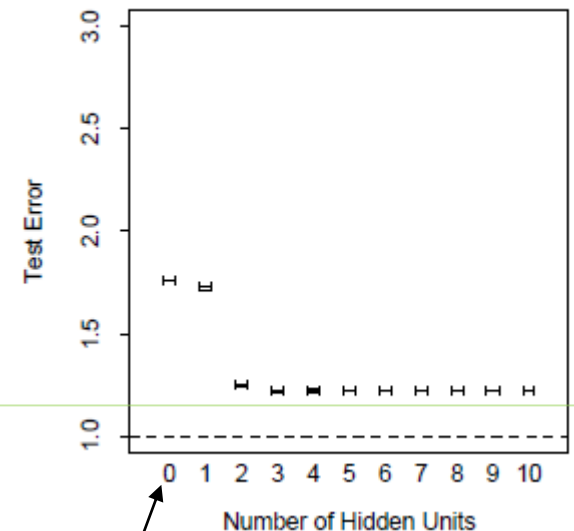
No Weight Decay



Weight decay = 0.0005



Weight Decay=0.1



October 6th 2021

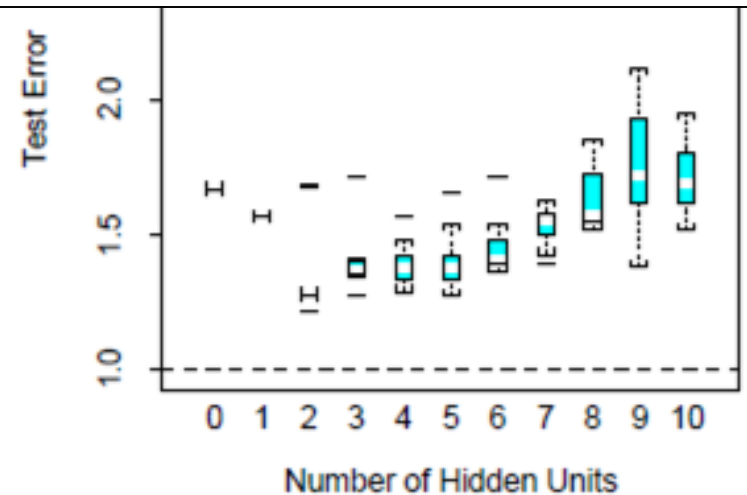
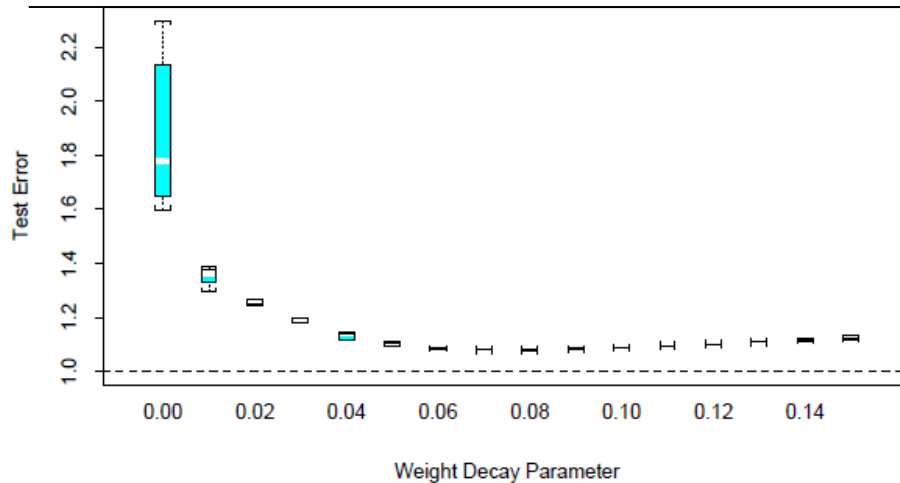
STK-IN 4300 Lecture - Neural nets

Linear regression

Weight decay vs number of hidden units

Sum of Sigmoids, 10 Hidden Unit Model

Weight decay = 0.0005



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

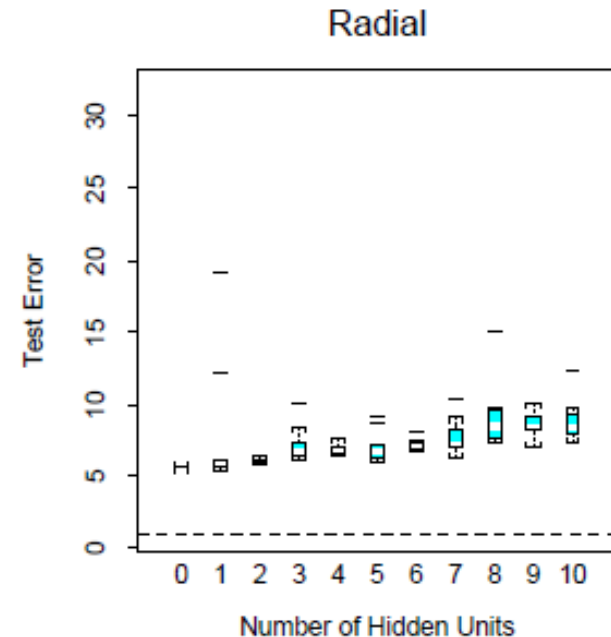
Examples of simulated data

Radial: $Y = \prod_{m=1}^{10} \phi(X_m) + \varepsilon_2.$

$$\phi(t) = (1/2\pi)^{1/2} \exp(-t^2/2)$$

- $\frac{\text{Var}(f(X))}{\text{Var}(\varepsilon)} \approx 4$
- 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



Alternative models for neural networks

11.7 Example: ZIP Code Data 405

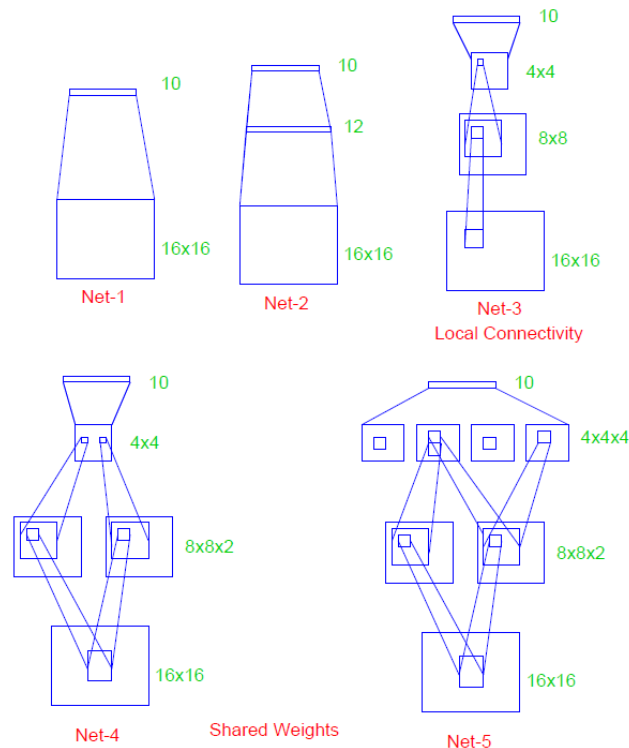


TABLE 11.1. Test set performance of five different neural networks on a hand-written digit classification example (Le Cun, 1989).

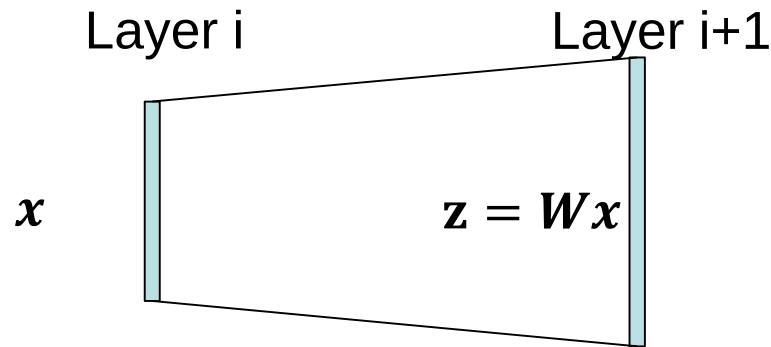
	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

«Hand crafting» NN might help
By reducing the number of
parameters to be estimated.

- Setting weights to zero (localize)
- Setting weights equal (Convolutional NN)

FIGURE 11.10. Architecture of the five networks used in the ZIP code example.

Neural net reduction in number of parameters (time series example)

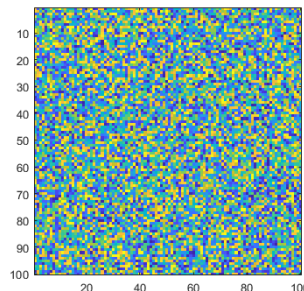


CNN use multiple features
 10 features => # = 100 param

Fully connected

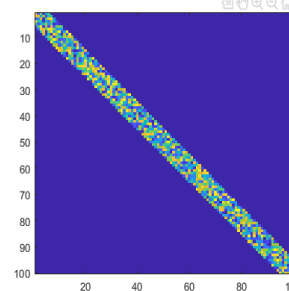
W
 (100×100)

#



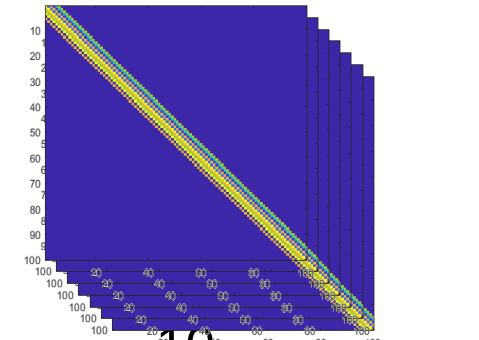
10100
 $(100+1) \times 100$

Localized (± 4)



1000
 $(9+1) \times 100$

Convolutional (± 4)



10 (per feature)
 $(9+1) \times 1$

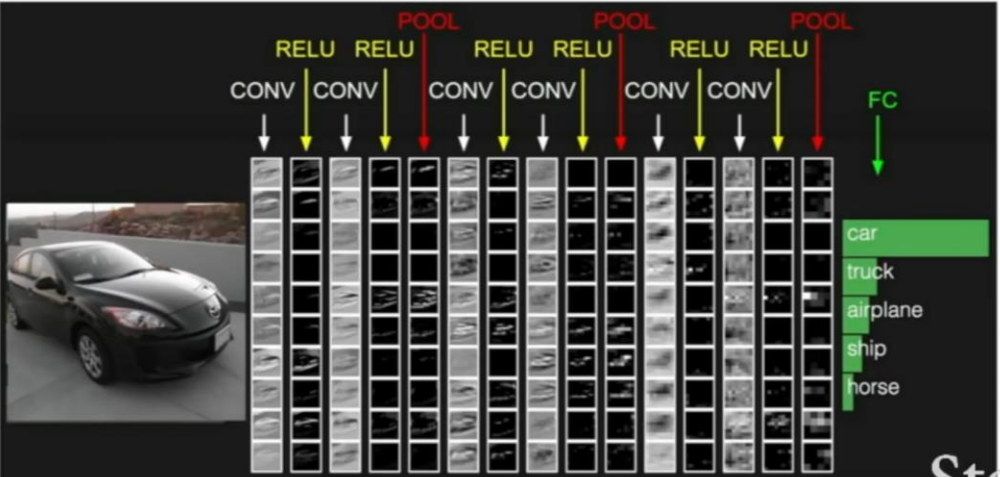
Bias

<https://www.youtube.com/watch?v=bNb2fEVKeEo>

Lecture 5 | Convolutional Neural Networks

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Stanford University

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 76

April 18, 2017

Series:

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

Layers

Per Layer
(CONV)

filters?

Size of filter?

Subsample?

Padding?

Activation
function?

POOLING
layer?

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 – wide – convolutional

