Neural networks

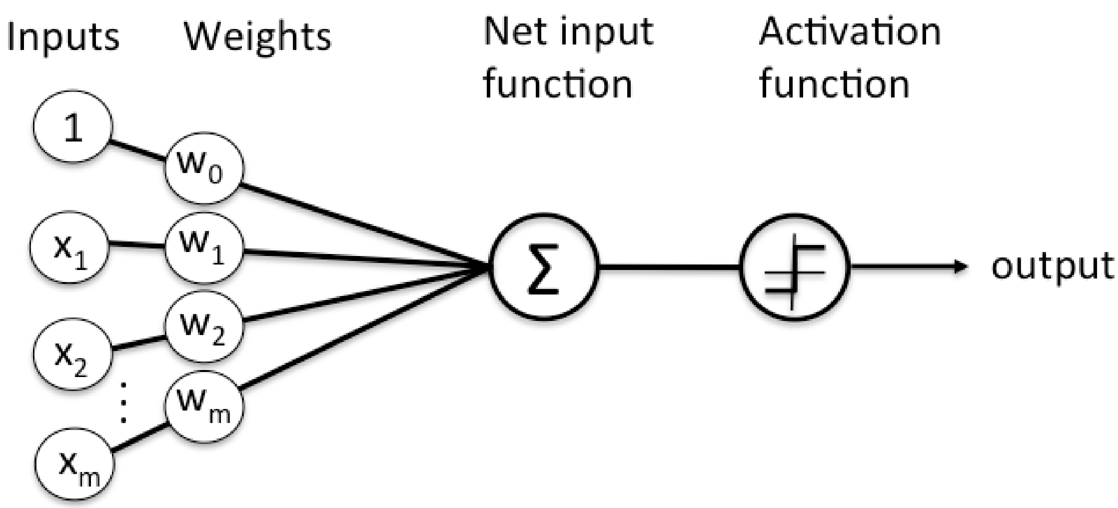
Hastie, Tibshirani, Friedman Ch 11

Kevin Murphy Ch. 8.3 and 16.5

CS 6140
Machine Learning
Professor Olga Vitek

April 11, 2017

Simple example: single-layer network



Two-class prediction, $Y = \{0, 1\}$:

$$f(X) = f(w_0 \cdot 1 + w_1 X_1 + \ldots + w_m X_m) \geq t$$

Perceptron:

- Two layers of nodes (input and output)
- Activation: step function (a neuron "fires"):

$$f(X) = \begin{cases} 1, & \text{if } w_0 \cdot 1 + w_1 X_1 + \ldots + w_m X_m \geq t \\ 0, & \text{if } w_0 \cdot 1 + w_1 X_1 + \ldots + w_m X_m < t \end{cases}$$

$\Rightarrow$ linear decision boundary: $w_0 \cdot 1 + w_1 X_1 + \ldots + w_m X_m = t$

`https://deeplearning4j.org/neuralnet-overview`

Recall: logistic regression

- 2 classes: a sigmoidal (activation) function

$$\log\left(\frac{\pi(X)}{1-\pi(X)}\right) = \beta_0 + \beta' X$$
$$\pi(X) = \frac{e^{\beta_0+\beta' X}}{1+e^{\beta_0+\beta' X}} = \frac{1}{1+e^{-(\beta_0+\beta' X)}}$$

  - A monotonic increasing/decreasing function
  - Perceptron: sigmoid activation function

- Extension to $K > 2$ classes
  - Baseline category logistic regression

$$\log\frac{\pi_k(X)}{\pi_1(X)} = \alpha_k + \beta_k' X,\ k = 2, \ldots, K$$

  - Since $\pi_k(X) = \pi_1(X) e^{\alpha_k+\beta_k' X}$ and $\sum_{k=1}^{K} \pi_k(X) = 1$

$$\begin{cases} \pi_1(X) = \frac{1}{1+\sum_{k=2}^{K} \exp(\alpha_k+\beta_k' X)},\ k = 1 \\ \pi_j(X) = \frac{\exp(\alpha_k+\beta_k' X)}{1+\sum_{k=2}^{K} \exp(\alpha_k+\beta_k' X)},\ k = 2, 3, \ldots, K \end{cases}$$

Details on $K > 2$ classes: conditional multinomial distributions

- $Y|X = x \sim Multinomial(n_{i+}, \pi_1(x), \ldots, \pi_K(x))$
- X - predictor;
  Y - multinomial response with K categories

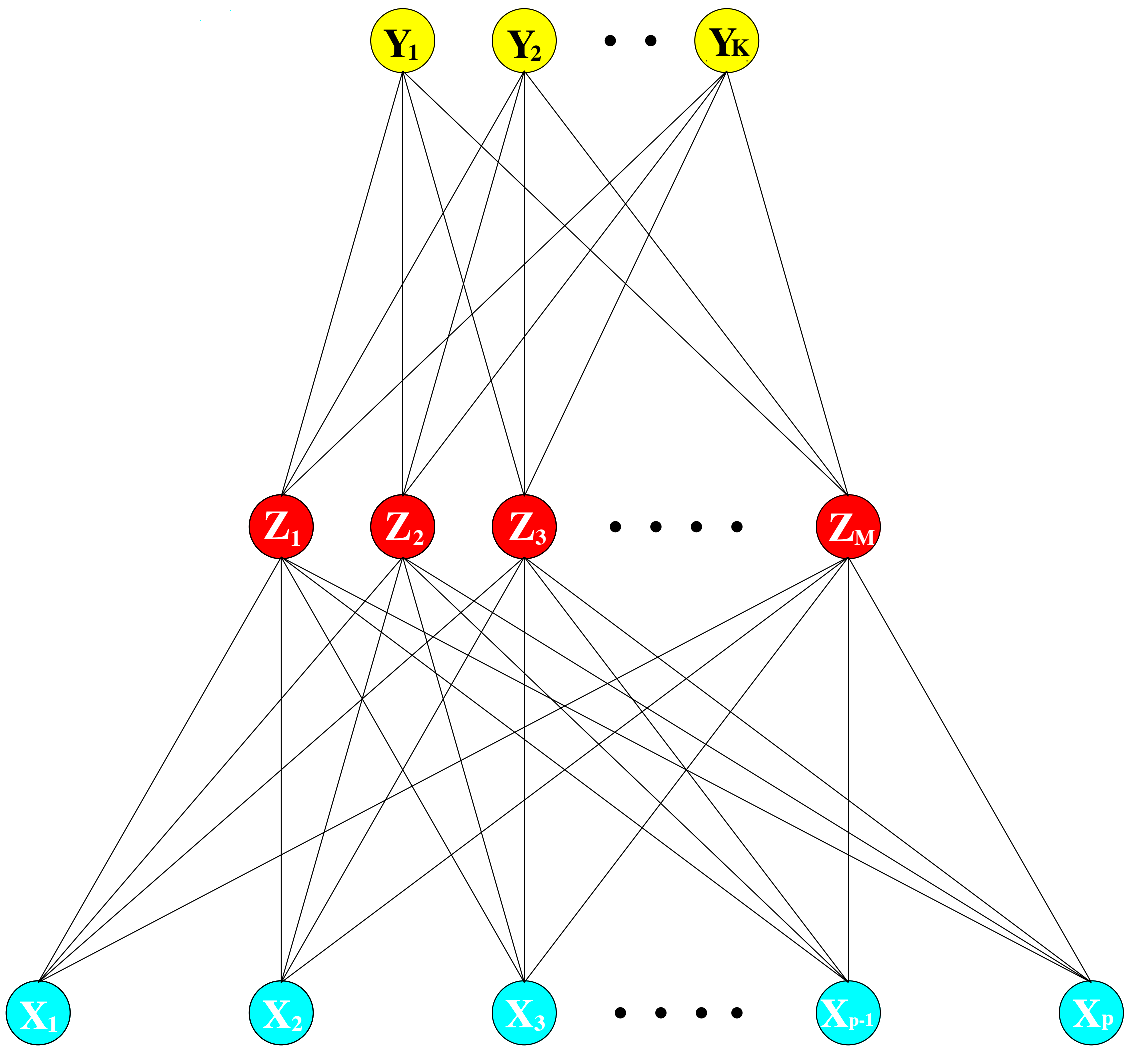
Row	Column			Total
	1	$\cdots$	K	
1	π_{11} ($\pi_{1\|1}$)	$\cdots$	π_{1K} ($\pi_{K\|1}$)	π_{1+}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
I	π_{I1} ($\pi_{1\|I}$)	$\cdots$	π_{IK} ($\pi_{K\|I}$)	π_{I+}
Total	π_{+1}	$\cdots$	π_{+J}	π_{++}

- Consider the new notation:
  $\pi_k(X) = P(Y = k|X = x)$, $\sum_{k=1}^{J} \pi_k = 1$
  - If Y is ordered, we can also be interested in cumulative probabilities:
    $P_j(X) = P(Y \leq k|X = x)$

Alternative logistic regression with $K > 2$ classes

- Log-linear model, multinomial distribution
  - Model $\log \pi_k(X)$ as linear of X + constraints
    $$\log \pi_k(X) = \alpha_k + \beta_k' X - \log Z,$$
    where Z is a normalizing constraint.
  - Since $\pi_k(X) = \frac{1}{Z} \cdot e^{\alpha_k + \beta_k' X}$, and $\sum_{k=1}^{K} \pi_k(X) = 1$,
    $$\Rightarrow Z = \sum_{k=1}^{K} e^{\alpha_k + \beta_k' X}$$
    $$\Rightarrow \pi_k(X) = \frac{e^{\alpha_k + \beta_k' X}}{\sum_{k=1}^{K} e^{\alpha_k + \beta_k' X}},$$ a softmax function
  - The exp exaggerates differences between X
  - Close to 1 when $\alpha_k + \beta_k' X \approx$ max of all values, close to 0 otherwise
  - Only $K - 1$ free parameters
  - Set $\alpha_K = 0$, $\beta_K = 0$ to have the previous case
- Use as activation function in perceptron

Neural networks

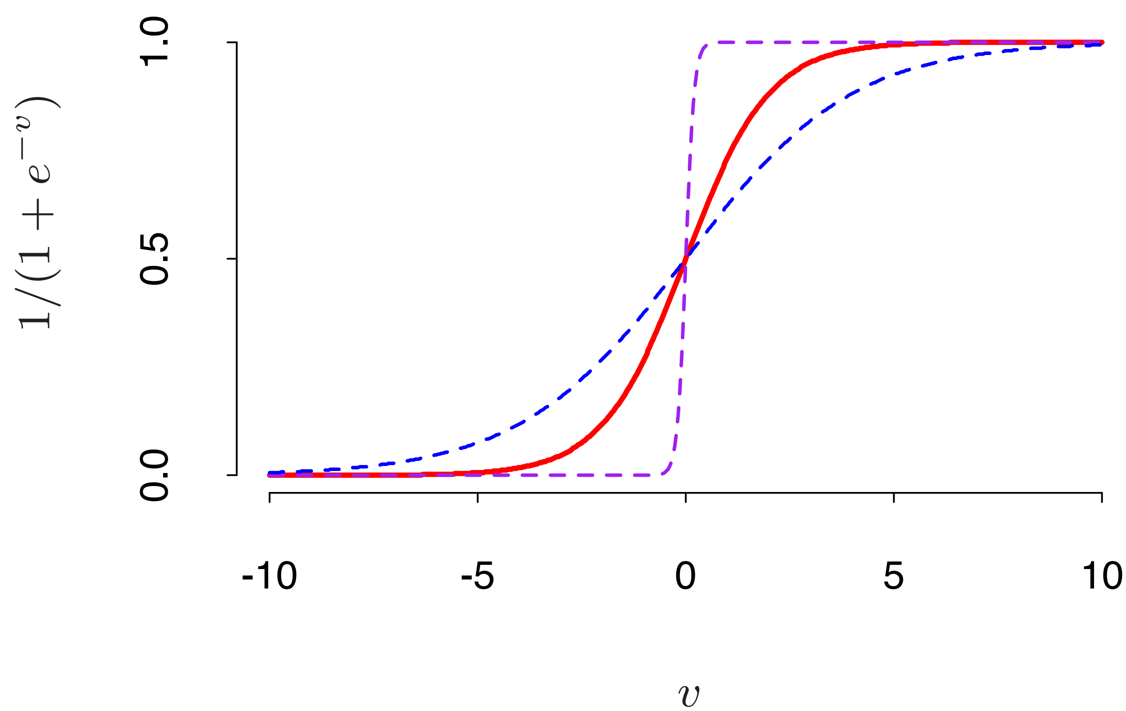


Single hidden layer neural network

PUseful when input features are not very informative

Fig. 11.2. Hastie, Tibshirani, Friedman
The Elements of Statistical Learning 2008

Activation function



Solid red: $\sigma(v) = 1/(1 + exp(-v))$

Dashed blue: $\sigma(\frac{1}{2}v)$

Dashed purple: $\sigma(10v)$

Large multiplier $\rightarrow$ hard activation at $v = 0$.

$\sigma(s(v - v_0))$ shifts the activation threshold from 0 to v_0.

In original neural networks: $\sigma(v)$ step activation
(neuron "fires" above a threshold)

Fig. 11.3. Hastie, Tibshirani, Friedman
The Elements of Statistical Learning 2008

Single hidden layer neural network

- K-class classification (regression: 1 class)
  - $Y_i \in \{0, 1\}$ for each class $\{1, \dots, K\}$
  - p input features X (including constant 1)

- Hidden features:
  created from linear combinations of inputs

$$
\begin{aligned}
Z_m &= \sigma(\alpha_{0m} + \alpha_m' X), \; m = 1, \dots, M \\
T_k &= \beta_{0k} + \beta_z' Z, \; k = 1, \dots, K \\
f_k(X) &= g_k(T), \; k = 1, \dots, K
\end{aligned}
$$

  Z_m: basis expansions of original inputs
  Need to learn all the parameters from data!

- Activation function:

$$\text{Example: sigmoid } \sigma(v) = \frac{1}{1 + e^{-v}}$$

  Linear $\sigma \to$ linear model

- Output function

$$\text{Example: softmax } g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_k}}$$

  (can use identity for regression)

Example

- Set-up:
  - K classes; $M = 3$ (3 hidden nodes or layers) $P = 2$ (two predictors)
- Prediction: transformation of outputs

  $T = \{T_1, T_2\}$: $f_k(X) = g_k(T)$

$$
\begin{aligned}
T_k &= \beta_{0k} \\
&+ \beta_{1k} \cdot \frac{1}{1 + \exp\{-(\alpha_{01} + \alpha_{11} X_1 + +\alpha_{21} X_2)\}} \\
&+ \beta_{2k} \cdot \frac{1}{1 + \exp\{-(\alpha_{02} + \alpha_{12} X_1 + +\alpha_{22} X_2)\}} \\
&+ \beta_{3k} \cdot \frac{1}{1 + \exp\{-(\alpha_{03} + \alpha_{13} X_1 + +\alpha_{23} X_2)\}}
\end{aligned}
$$

  - Class-specific linear combinations of node-specific input summaries
- Notes:
  - β_{0k} and α_{0m} - "bias"-nodes
  - Basis expansion, where parameters of the bases α_{lm} are learned from the data
  - $\sigma()$ is linear $\rightarrow$ linear model in inputs
  - $f_k()$ softwmax $\rightarrow$ logistic regression, linear in Z_m
  - $M = 1 \rightarrow$ perceptron

2-class logistic regression: likelihood

- $Y_i \stackrel{ind}{\sim}$ Bernoulli(π_i) where

$$\pi_i = \frac{\exp(\beta_0 + \beta_1 X_i)}{1 + \exp(\beta_0 + \beta_1 X_i)}$$

- Log likelihood (=cross-entropy): logL =

$$\begin{aligned}
&= \log\left\{\prod_{i=1}^{n} \pi_i^{Y_i}(1-\pi_i)^{1-Y_i}\right\} \\
&= \sum_{i=1}^{n} Y_i \log(\pi_i) + \sum_{i=1}^{n} (1-Y_i)\log(1-\pi_i) \\
&= \sum_{i=1}^{n} Y_i \log\left(\frac{\pi_i}{1-\pi_i}\right) + \sum_{i=1}^{n} \log(1-\pi_i) \\
&= \sum_{i=1}^{n} Y_i(\beta_0 + \beta_1 X_i) - \sum_{i=1}^{n} \log(1 + \exp(\beta_0 + \beta_1 X_i))
\end{aligned}$$

  - Recall that equivalent specification for Binomial distribution exists
  - Maximum likelihood estimates do not have closed form

Optimizing the likelihood: gradient descent

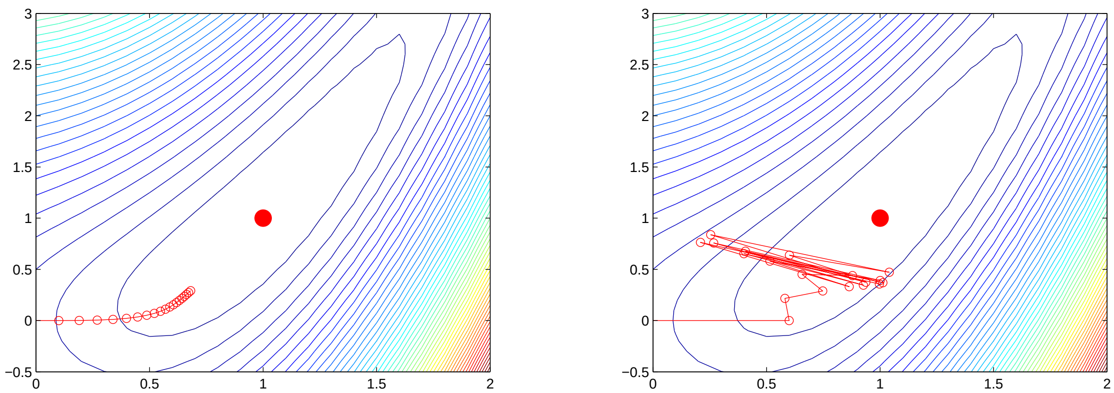
- Example: minimize function with two parameters $J(\theta_0, \theta_1)$
  - Start with some θ_0, θ_1
  - Simultaneously change θ_0, θ_1 in direction of steepest descent

$$\theta_j^{r+1} = \theta_j^r - \eta_r \frac{\partial}{\partial \theta_j} J(\theta_0^r, \theta_1^r)$$

  - Stop when reaching (local) minimum of $J(\theta_0, \theta_1)$

- Comments:
  - η_r: step size or learning rate. Typically a fixed parameter. Advanced algorithms choose automatically.
  - Results reach local minimum
  - Results sensitive to starting values

Example: gradient descent



Minimize $J(\theta_0, \theta_1) = 0.5(\theta_0^2 - \theta_1)^2 + 0.5(\theta_0 - 1)^2$,

Start at (0,0), 20 steps, fixed learning rate η.

Left: $\eta = 0.1$. Moves slowly but finds the valley.

Right: $\eta = 0.6$. Oscillates and never converges.

KM Fig 8.2

Logistic regression: parameter estimation

- Minimizing -log likelihood $J(\beta) = -\text{logL}$

$$J(\beta) \;\;=\;\; -\left[\sum_{i=1}^{n} Y_i \log(\pi_i) + \sum_{i=1}^{n} (1 - Y_i) \log(1 - \pi_i)\right]$$

  - π_i are non-linear functions of β
  - Partial derivatives:

$$\frac{\partial}{\partial \beta_j} J(\beta) = \sum_{i=1}^{n} (\pi_i - y_i) x_j$$

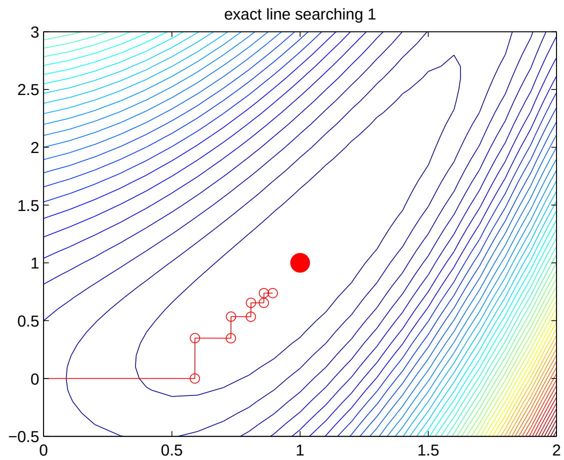
  - Update to each parameter:

$$\beta_j^{r+1} = \beta_j^r - \eta_r \sum_{i=1}^{n} (\pi_i^r - y_i) x_j$$

- Note
  - Same for linear regression, replace π_i^k with $\widehat{y}_i^k$

- Improvements
  - Line search algorithm: automatically picks η_k
  - Newton's method: takes into account the curvature of the space (second derivatives)

Example: line search



- Set $r = 0$ and initial guess θ_0^r and θ_1^r
  - Compute a descent direction $\mathbf{d}_r$
  - Choose η_r to "loosely" minimize $\phi(\alpha) = J(\boldsymbol{\theta}_r + \eta \mathbf{d}_k)$
  - Update $\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}_r + \eta \mathbf{d}_k$
- Until tolerance reached

KM Fig 8.3

Neural network: learning (parameter estimation)

- Fitting to the data
  - The full collection of parameters

    $\theta = \{\alpha_{0m}, \alpha_m,\ m = 1, \ldots, M\}$ $M(P+1)$ weights
    and $\{\beta_{0k}, \beta_k,\ k = 1, \ldots, K\}$ $K(M+1)$ weights

    Minimize
  - Regression: RSS $R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_i - f_k(x_i))^2$
  - Classification: cross-entropy $R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_i \, log \, f_k(x_i)$
  - $g_k(T_k)$ softmax + cross-entropy $R(\theta)$
    $\rightarrow$ linear logistic regression in the hidden units
  - Maximum likelihood parameter estimation

- Global optimum $R(\theta)$ likely overfits
  - Heavily overparametrized
  - Start with predictor weights ≈ 0 (approximately linear in X), gradually increase

Back propagation: K classes, squared loss

- Objective function

$$R(\theta) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ij} - f_k(x_i))^2$$

  – Here $f_k(x_i) = g_k(\beta_{0k} + \beta_k' Z)$, $k = 1, \ldots, K$ and $Z = \{\sigma(\alpha_{0m} + \alpha_m' x_i)\}$, $m = 1, \ldots, M$

- Derivatives

$$\begin{aligned}
\frac{\partial R_i}{\partial \beta_{km}} &= -2\left(y_{ik} - f_k(x_i)\right) g_k'\left(\beta_k' z_i\right) z_{mi} \\
\frac{\partial R_i}{\partial \alpha_{ml}} &= -\sum_{k=1}^{K} 2\left(y_{ik} - f_k(x_i)\right) g_k'\left(\beta_k' z_i\right) \beta_{km} \sigma'\left(\alpha_m \prime x_i\right) x_{il}
\end{aligned}$$

- Gradient descent at iteration $r+1$

$$\begin{aligned}
\beta_{km}^{r+1} &= \beta_{km}^{r} - \eta_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{r}} \\
\alpha_{ml}^{r+1} &= \alpha_{ml}^{r} - \eta_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{r}}
\end{aligned}$$

Back-propagation equations

- Can write the derivatives above as

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi} \text{ and } \frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}$$

  - δ_{ki} and s_{mi} are "errors" of the current model at the output and hidden layer levels

  - Satisfy back-propagation equations

    $s_{mi} = \sigma'(\alpha' x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$

- Update $r \to r+1$ in two passes:

  - Forward pass: fix current parameters, fix weights δ_{ki} and s_{mi}, compute predicted values $\hat{f}_k(x_i)$

  - Backward pass: Compute δ_{ki}, then back-propagate to s_{mi}, and calculate gradients of the update

  - The info is passed to/from connected units

- Comments

  - Step size can be optimized by line search

  - Newton method is not attractive, because the second derivative can be large

  - See HTF Sec 11.4 for more details

Neural network: parameter estimation with weight decay

- Stopping rule
  - Start with $\theta \approx 0$, $\uparrow$, stop before global optimal
  - Amounts to shrinkage of global optimum weights

- Explicit regularization: weight decay

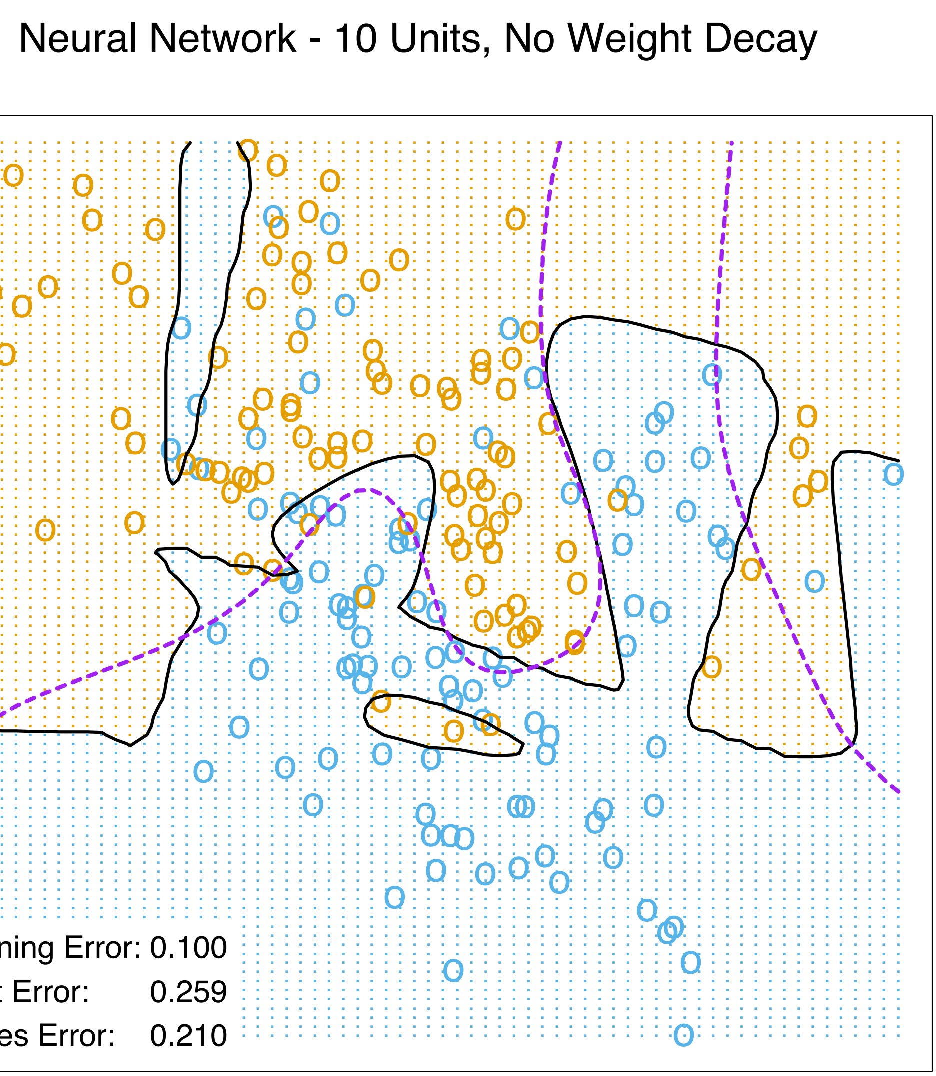
$$\text{Minimize } R(\theta) + \lambda \left(\sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2 \right)$$

  - $\lambda \geq 0$ is a tuning parameter
  - Large λ shrink weights more towards linearity
  - Estimate λ by cross-validation
  - Scale the predictors, so that they are equally affected by regularization
  - More hidden nodes can be compensated for with more regularization

- Back-propagation algorithm
  - Forward pass: fix weights, calculate $\widehat{f}(x_i)$
  - Backward pass: fix predictions, calculate errors, and weights to minimize errors

  - Each unit passes and receives info only from its connection (easy to parallelize)

Prone to overfitting

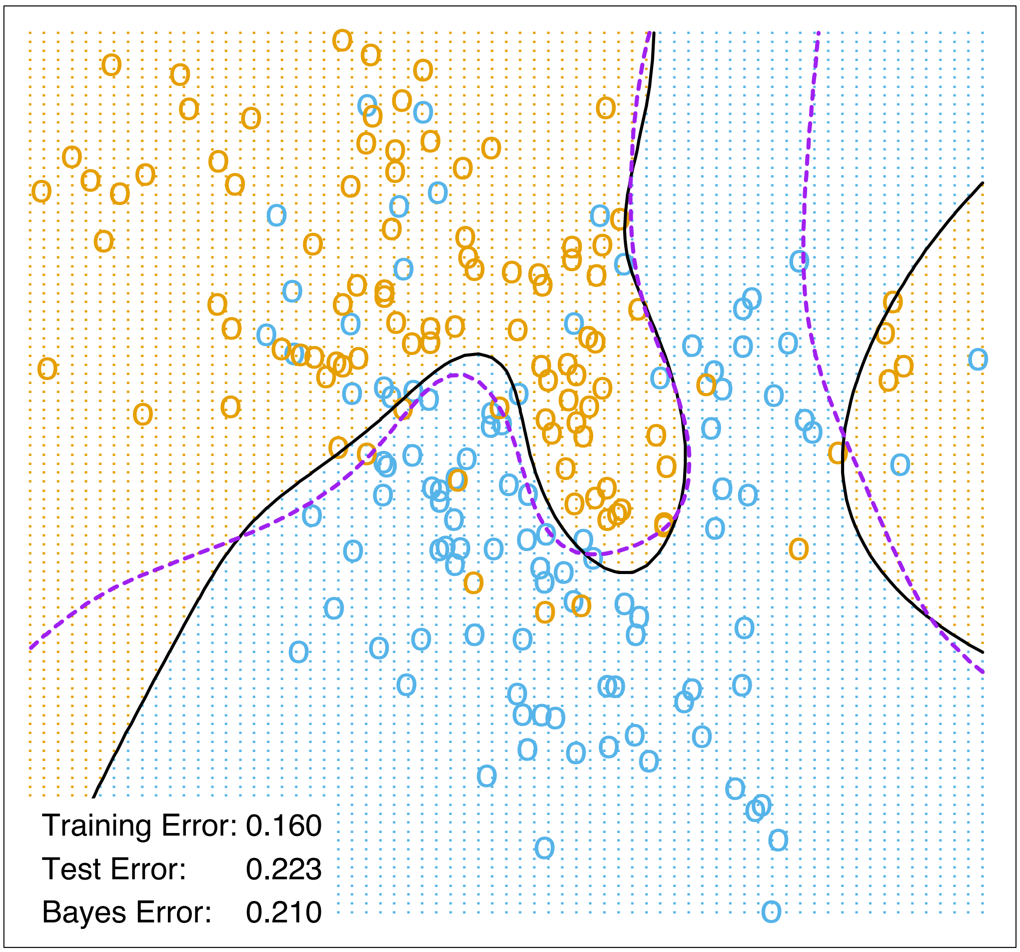


Soft max activation and cross-entropy error. No weights decay, overfits the data

HTF Fig 11.4

Prone to overfitting

Neural Network - 10 Units, Weight Decay=0.02



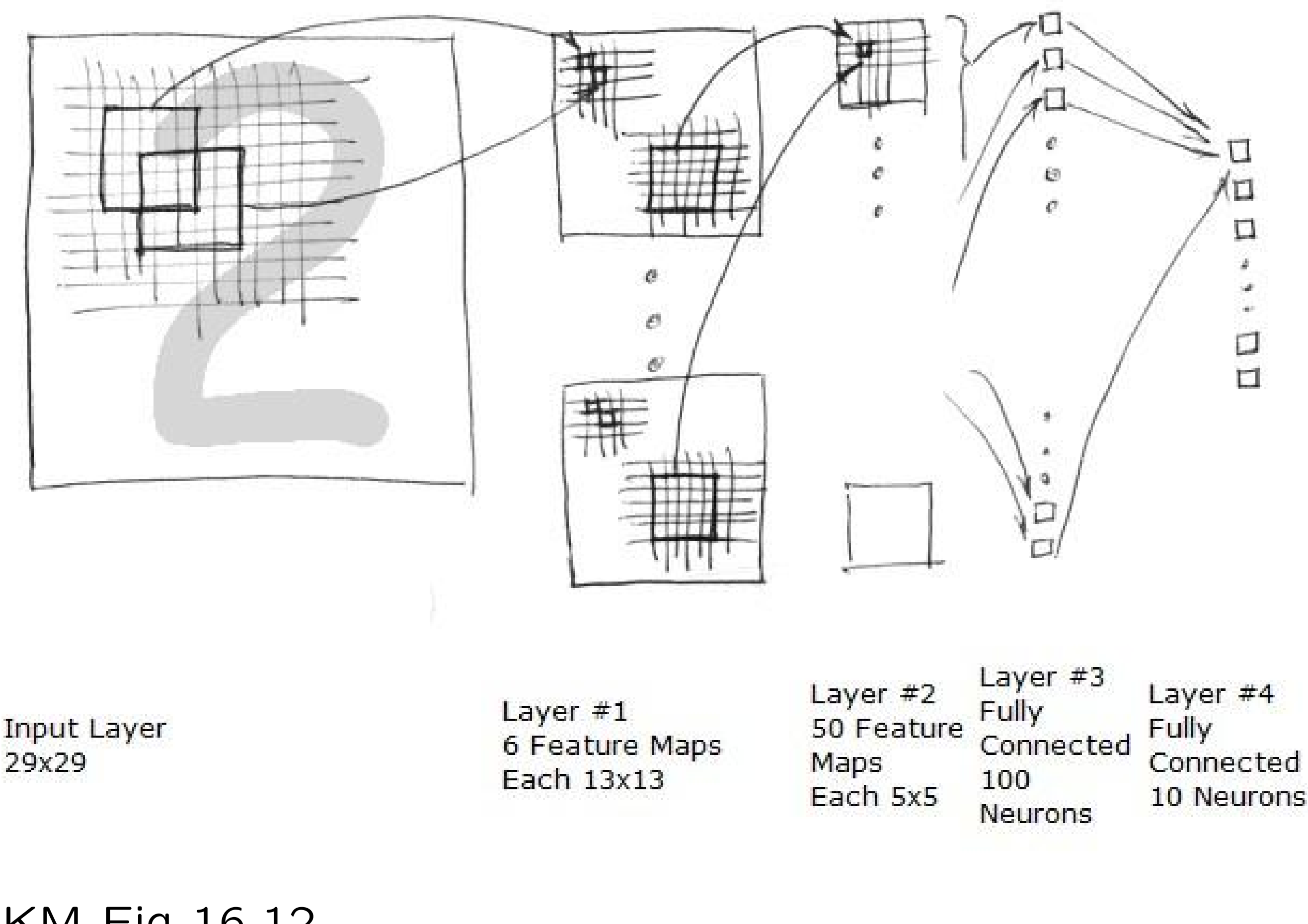
Soft max activation and cross-entropy error. Weights decay.

HTF Fig 11.4

Generalizations:

- Deep learning:
  - Neural networks with > 1 hidden layer

- Convolutional neural networks
  - Train neural networks on local fields



KM Fig 16.12