

Neural Networks

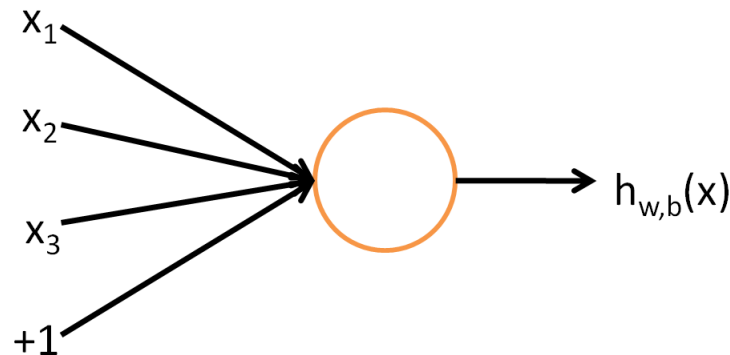
Statistical Learning

Master in Big Data. University of Santiago de Compostela

Manuel Mucientes

Introduction

- Basic idea:
 - Extract linear combinations of the inputs as derived features
 - Model the target as a nonlinear function of these features
- An example of a single neuron:



- $$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^3 W_i x_i + b)$$
 - f is the activation function
 - b is the bias term

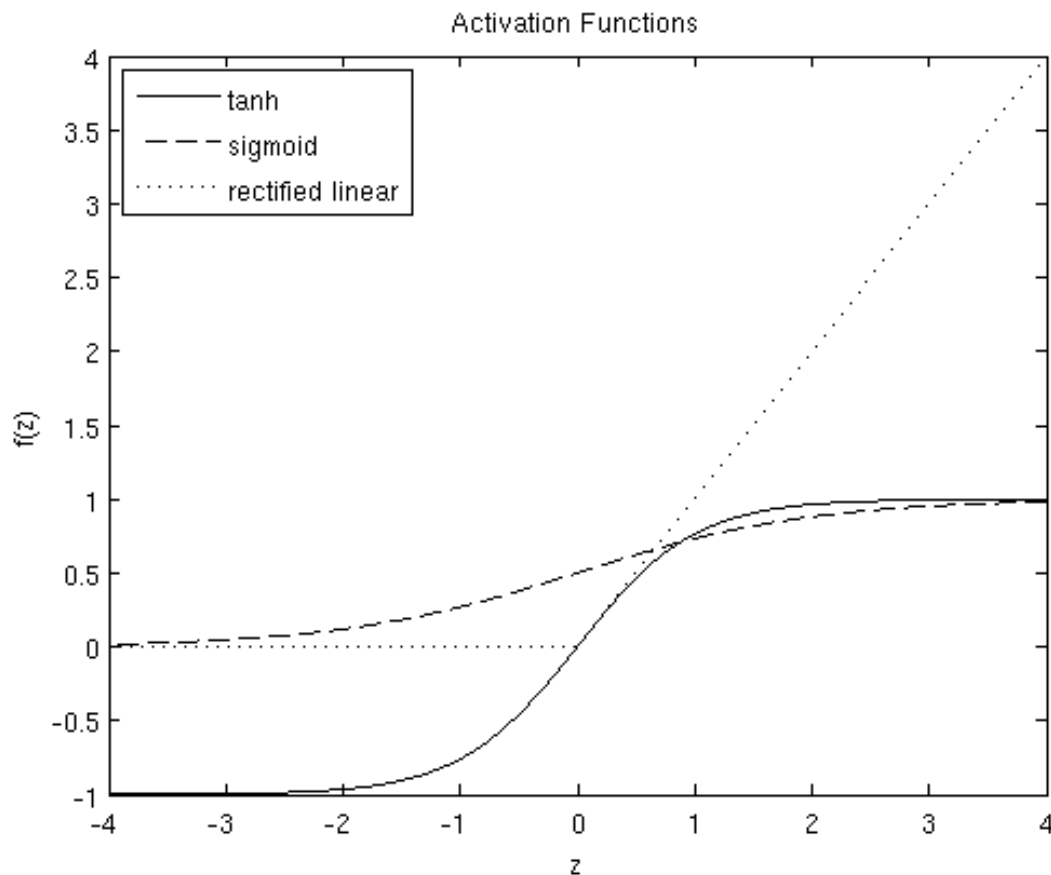
Activation functions

- Sigmoid: $[0, 1]$

$$f(z) = \frac{1}{1 + \exp(-z)}$$

- Derivative:

- $f'(z) = f(z)(1 - f(z))$



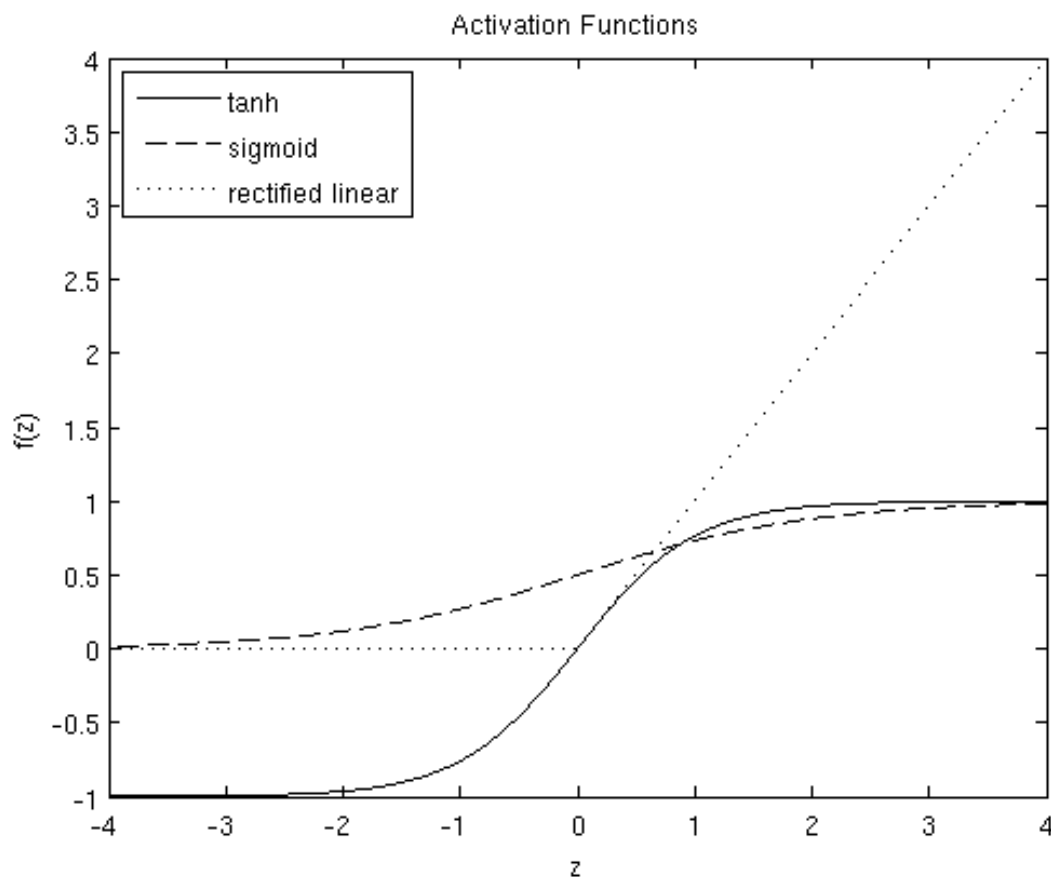
Activation functions (ii)

- Hyperbolic tangent: $[-1, 1]$

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Derivative:

- $f'(z) = 1 - (f(z))^2$



Activation functions (iii)

■ Rectified linear unit (ReLU):

$$f(z) = \max(0, z)$$

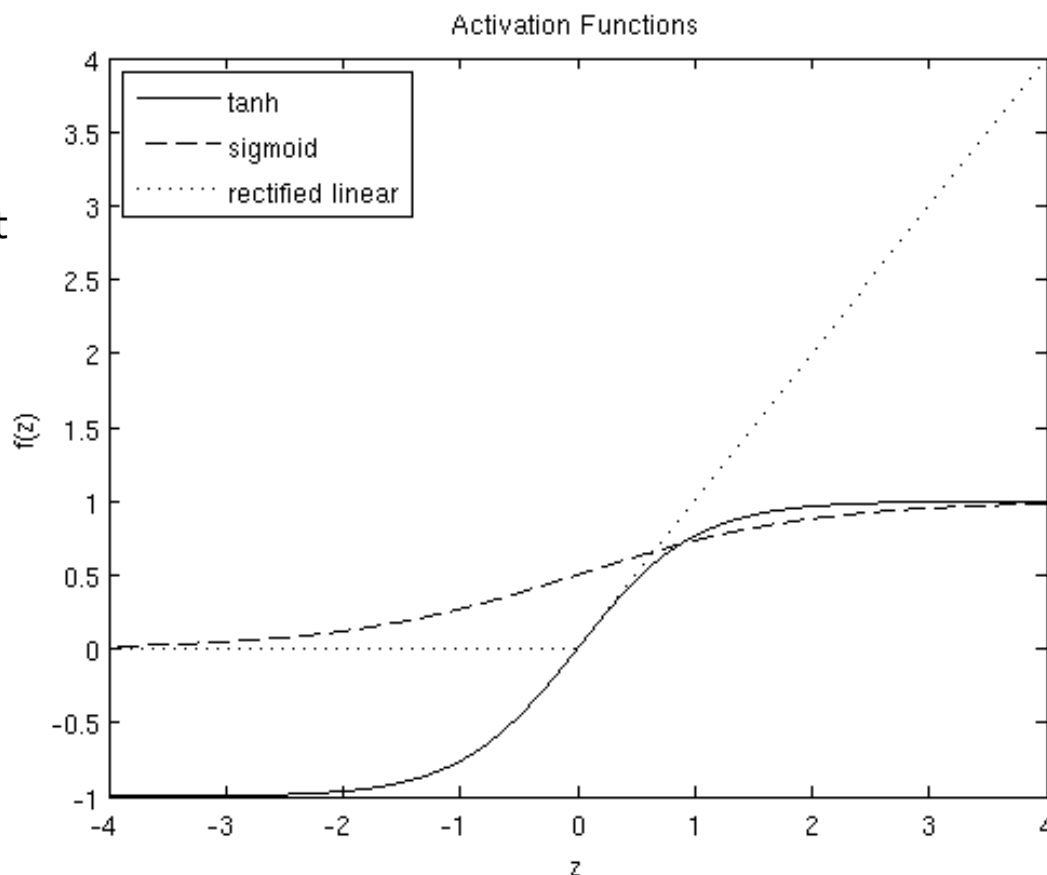
■ Derivative:

- 0 if $z < 0$, 1 otherwise
- Undefined at $z = 0$
 - Average the gradient over many training examples during optimization

■ Gaussian radial basis functions

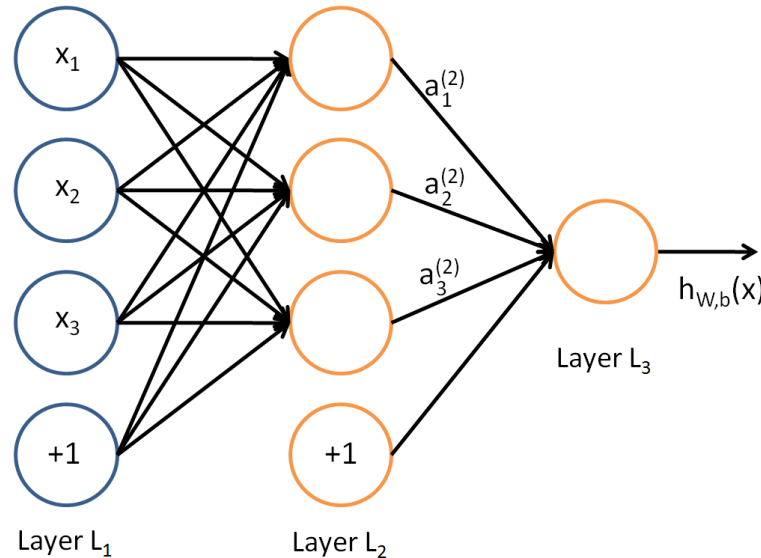
■ RBF networks

$$f(z) = \exp(-(1/2)(z-c)^T \Sigma^{-1}(z-c))$$



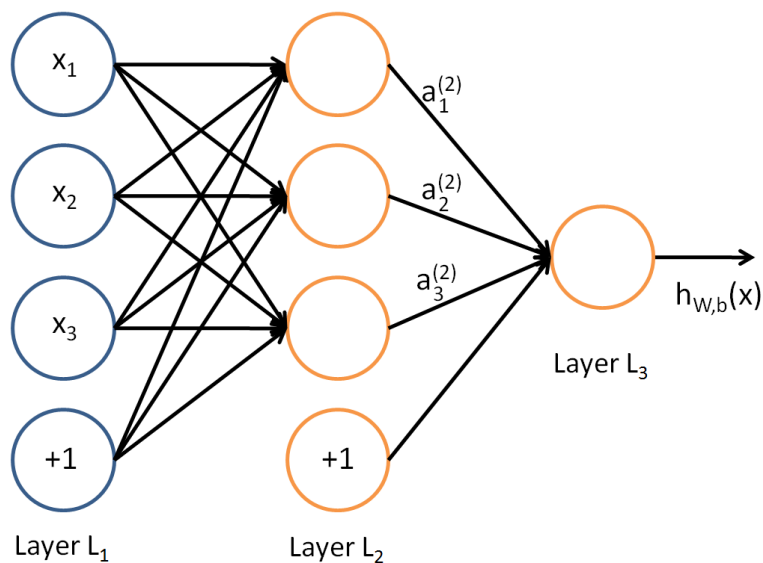
Neural Network Model

- A single hidden layer feed-forward neural network:



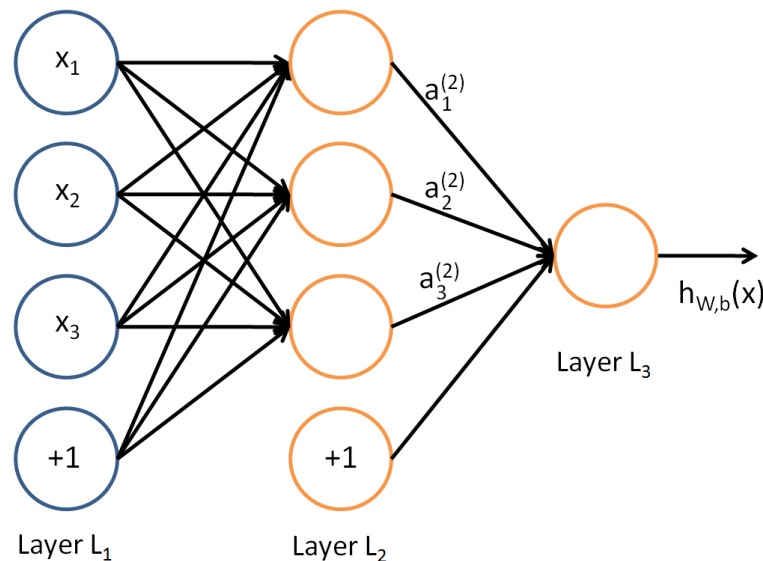
- Input layer, hidden layer, output layer
- Parameters: $(W,b)=(W^{(1)},b^{(1)},W^{(2)},b^{(2)})$
 - $W_{ij}^{(l)}$: weight associated with the connection between unit j in layer l and unit i in layer $l+1$
 - $W^{(1)} \in \mathbb{R}^{3 \times 3}$, and $W^{(2)} \in \mathbb{R}^{1 \times 3}$
 - $b_i^{(l)}$ is the bias associated with unit i in layer $l+1$
 - Bias units do not have inputs or connections going into them

Neural Network Model (ii)



- $a_i^{(l)}$: activation (output value) of unit i in layer l
 - $a_i^{(1)} = x_i$
- s_l : number of nodes in layer l (not counting the bias unit)

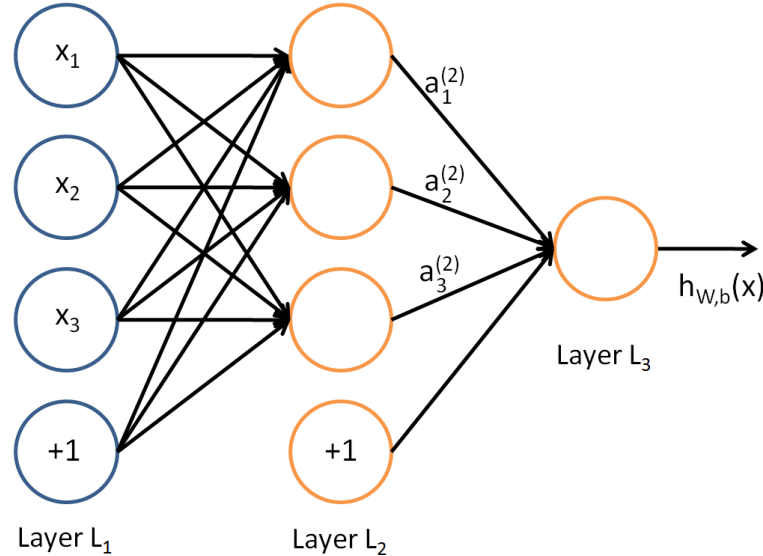
Neural Network Model (iii)



■ Computation:

$$\begin{aligned}a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})\end{aligned}$$

Neural Network Model (iv)

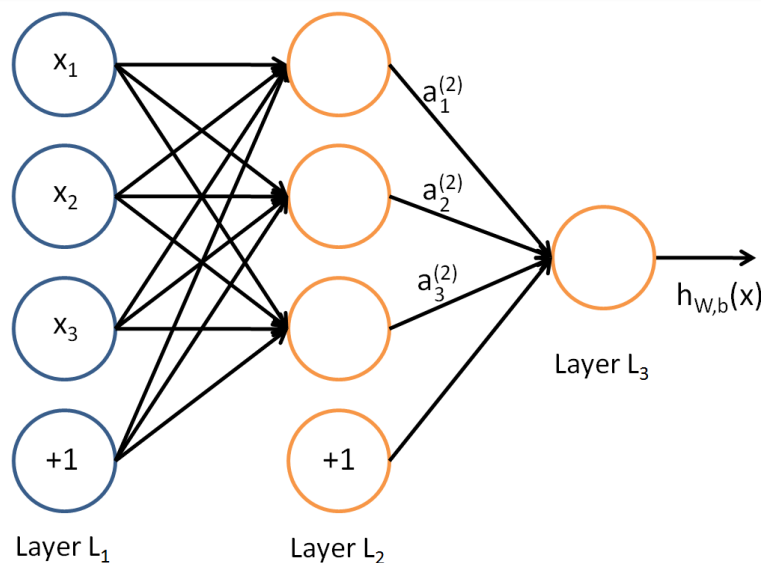


- If $z_i^{(l+1)}$ is the total weighted sum of inputs to unit i in layer $l+1$:

$$z_i^{(l+1)} = \sum_{j=1}^{s_l} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)}$$

- $a_i^{(l)} = f(z_i^{(l)})$

Neural Network Model (v)



- Computation: forward propagation

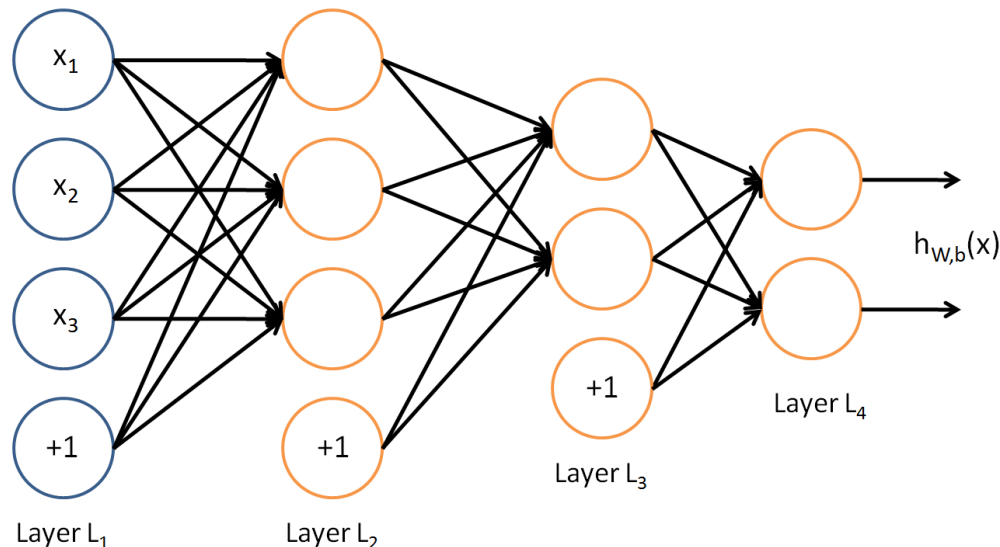
$$\begin{aligned}z^{(2)} &= W^{(1)}x + b^{(1)} \\a^{(2)} &= f(z^{(2)}) \\z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\h_{W,b}(x) &= a^{(3)} = f(z^{(3)})\end{aligned}$$

- In general (matrix-vector operations):

$$\begin{aligned}z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\a^{(l+1)} &= f(z^{(l+1)})\end{aligned}$$

Architectures

- Patterns of connectivity between neurons
 - Multiple hidden layers
 - Fully (densely) vs. locally connected
 - Weight sharing (locally connected)
 - feed-forward (no loops)
- Most common choice: multilayer feed-forward neural network (multilayer perceptron network)
 - Forward propagation step to calculate outputs of each layer



Architectures (ii)

- Regression:

- Typically, one output unit, except when there are several outputs
- The output function is typically the identity function

- Classification:

- For K-class classification, K units in the output layer

E.g. $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$
pedestrian car motorcycle truck

- Output function: *softmax*

$$f(z_i^{(l)}) = \frac{e^{z_i^{(l)}}}{\sum_{j=1}^K e^{z_j^{(l)}}}$$

Architectures (iii)

- An example of the output of *softmax*
 - Three categories (K=3): bike, car, truck

$$f(z_i^{(l)}) = \frac{e^{z_i^{(l)}}}{\sum_{j=1}^K e^{z_j^{(l)}}}$$



	z_i	$\exp(z_i)$	$f(z_i)$	Correct probs.
Bike	-0.2	0.8	0.02 (2%)	0.00
Car	3.6	36.6	0.75 (75%)	1.00
Truck	2.4	11.0	0.23 (23%)	0.00

Fitting Neural Networks

■ Backpropagation algorithm

■ Training examples: $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

■ For a single example:

- Regression and classification:

$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2$$

- For classification (mutually exclusive classes), cross -entropy is also valid:

$$J(W, b, x, y) = - \sum_{k=1}^K y_k \log (h_{W,b}(x))_k$$

■ For classification:

■ Sigmoid, *softmax*: $y=0, 1$; output in $[0, 1]$

■ tanh: $y=-1, 1$; output in $[-1, 1]$

■ For regression: scale the outputs (range depends on the activation function)

Fitting Neural Networks (ii)

- Overall cost function:

$$\begin{aligned} J(W, b) &= \left[\frac{1}{m} \sum_{i=1}^m J(W, b; x^{(i)}, y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^m \left(\frac{1}{2} \|h_{W,b}(x^{(i)}) - y^{(i)}\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \end{aligned}$$

- Regularization term (weight decay): prevent overfitting
 - Not applied to the bias terms
- Minimize J through backpropagation
 - J is non-convex: local minima
 - Gradient descent: usually works fairly well

Fitting Neural Networks (iii)

- One iteration of gradient descent updates W, b : Widrow-Hoff rule

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$
$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

- α : learning rate
- Compute the partial derivatives: backpropagation algorithm

$$\frac{\partial J(W, b)}{\partial W_{ij}^{(l)}} = \left[\frac{1}{m} \sum_{e=1}^m \frac{\partial J(W, b; x^{(e)}, y^{(e)})}{\partial W_{ij}^{(l)}} \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial J(W, b)}{\partial b_i^{(l)}} = \frac{1}{m} \sum_{e=1}^m \frac{\partial J(W, b; x^{(e)}, y^{(e)})}{\partial b_i^{(l)}}$$

Backpropagation algorithm

1. Perform a feedforward pass, computing the activations for layers L_2 , L_3 , and so on up to the output layer L_{n_l} .
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \dots, 2$

For each node i in layer l , set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

Intuition:

$\delta_i^{(l)}$ = error of node i in layer l

4. Compute the desired partial derivatives, which are given as:

$$\begin{aligned} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) &= a_j^{(l)} \delta_i^{(l+1)} \\ \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) &= \delta_i^{(l+1)}. \end{aligned}$$

How much error changes with modifications in $W_{ij}^{(l)}$:
[activation of node j in layer l] times
[error of node i in layer $l+1$]

Backpropagation algorithm (ii)

■ Matrix-vectorial notation

1. Perform a feedforward pass, computing the activations for layers L_2 , L_3 , up to the output layer L_{n_l} , using the equations defining the forward propagation steps

2. For the output layer (layer n_l), set

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \dots, 2$, set

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \bullet f'(z^{(l)})$$

4. Compute the desired partial derivatives:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

Full gradient descent algorithm

1. Set $\Delta W^{(l)} := 0$, $\Delta b^{(l)} := 0$ (matrix/vector of zeros) for all l .
2. For $i = 1$ to m ,
 1. Use backpropagation to compute $\nabla_{W^{(l)}} J(W, b; x, y)$ and $\nabla_{b^{(l)}} J(W, b; x, y)$.
 2. Set $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$.
 3. Set $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$.
3. Update the parameters:

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$
$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

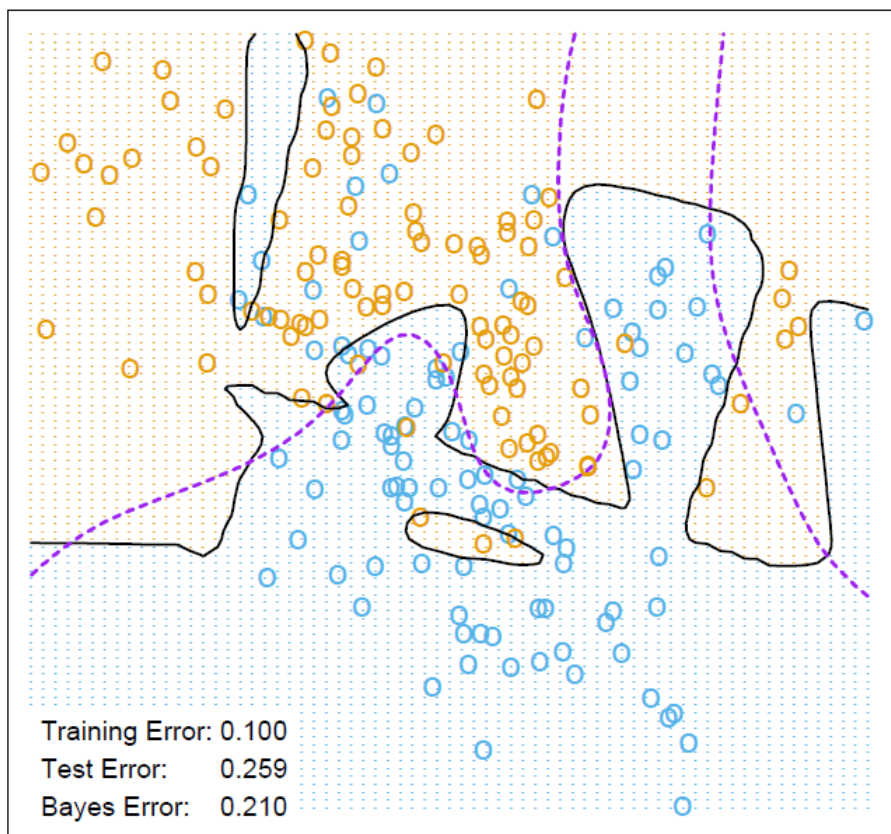
Training Neural Networks

- Over-parametrized model, non-convex and unstable optimization problem
- Starting values:
 - Random values near 0 for W , biases to 0
 - With standardized inputs, random uniform weights in $[-0.7, 0.7]$
 - Ok for small networks
 - Xavier initialization: $\text{random()} * \sqrt{1/n}$
 - random() : mean 0, variance 1
 - n : number of inputs
 - Model starts out nearly linear, and becomes nonlinear as weights increase
 - 0 weights give perfect symmetry, large weights lead poor solutions

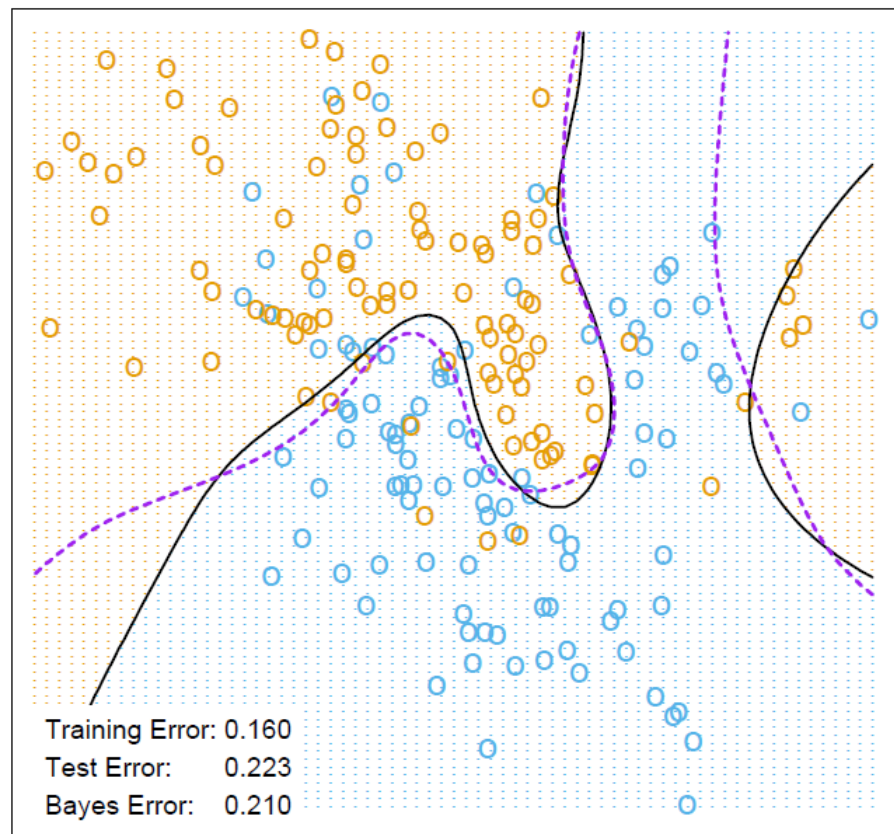
Training Neural Networks (ii)

- Overfitting: regularization term (weight decay)
 - Cross-validation to estimate the regularization parameter

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02



Training Neural Networks (iii)

■ Scaling of the inputs:

- Can have a large effect in the quality of the final solution: scaling of the weights
- Standardize inputs (and outputs) to have mean zero and standard deviation one:
 - Treat all inputs equally: regularization
 - Choose a meaningful range for the starting weights

■ Number of hidden units:

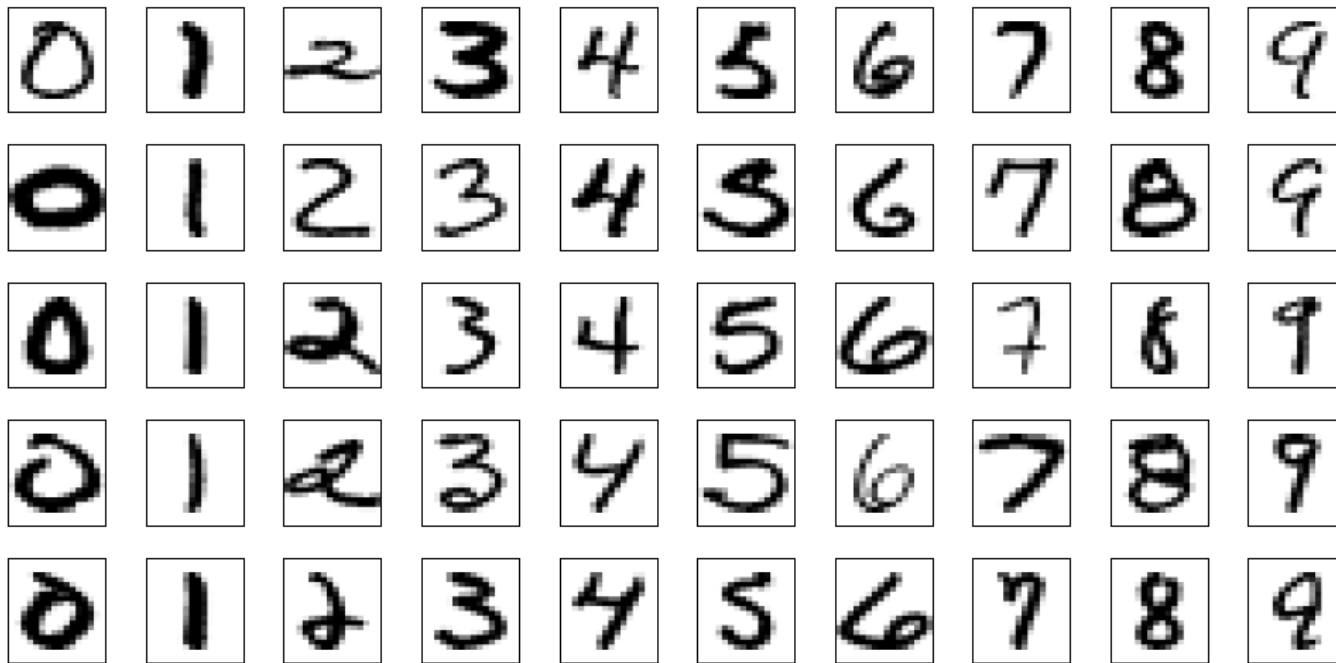
- Better to have too many hidden units than too few
 - Too few: not enough flexibility to capture nonlinearities
 - Too many: extra weights can be shrunk toward zero with regularization
- Typical number of hidden units: 5 to 100 (per layer)
 - Number increasing with number of inputs and number of training cases
 - Same number in every layer (reasonable default)

Training Neural Networks (iv)

- Number of hidden layers:
 - background knowledge and experimentation
 - Allow the construction of hierarchical features at different levels of resolution
- Multiple minima: non-convex error function
 - Try a number of random starting configurations: choose the best solution
 - Better approach: average the predictions over a collection of networks
 - Another approach: bagging
- Batch learning: all the examples
 - Stochastic Gradient Descent (SGD): 1 example
- Advanced optimization methods:
 - Conjugate gradient
 - Quasi-Newton: Broyden–Fletcher–Goldfard–Shanno (BFGS) and L-BFGS

Example: ZIP Code Data

- Le Cun, 1989
- 16x16 grayscale images
- Training with 320 digits, test with 160
- Five different networks: sigmoidal output units, sum-of-squares error function



Example: ZIP Code Data (ii)

- Net-1: no hidden layer, equivalent to multinomial logistic regression
- Net-2: one hidden layer, 12 hidden units fully connected
- Net-3: two hidden layers locally connected

- First hidden layer:

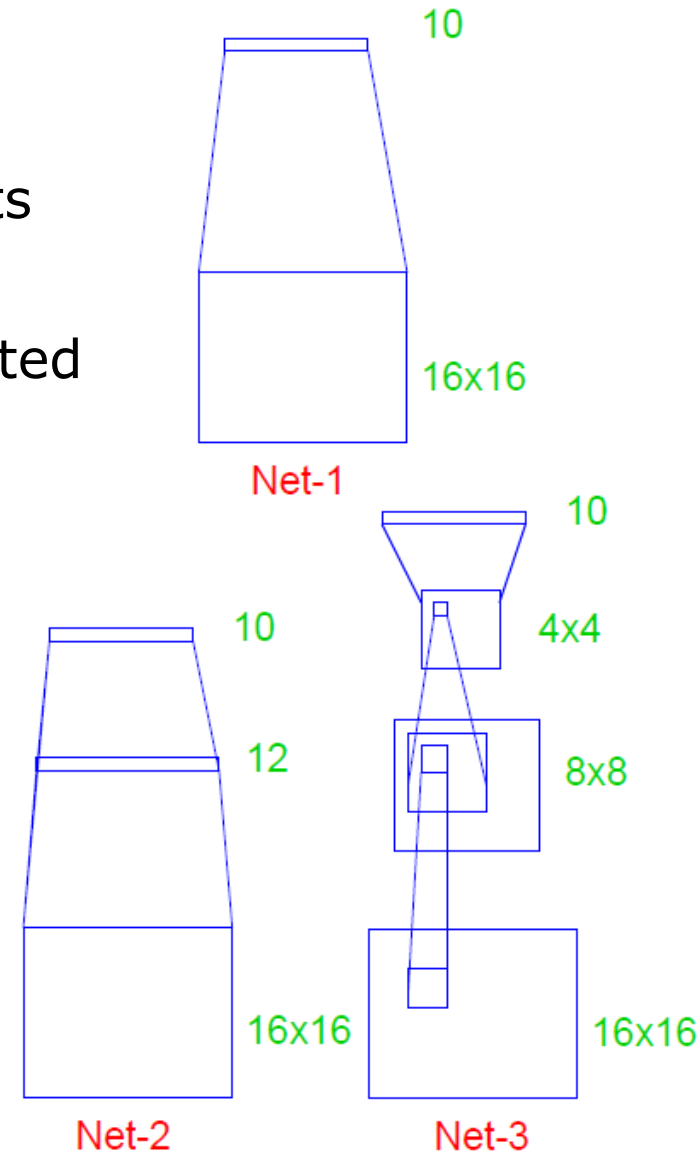
- Inputs from a 3x3 patch
- Units 1 unit apart are 2 pixels apart

- Second hidden layer:

- Inputs from a 5x5 patch
- Units 1 unit apart are 2 pixels apart

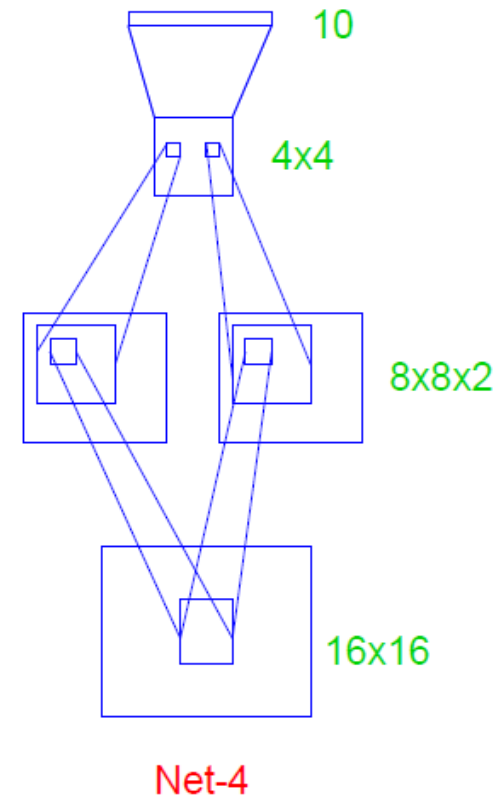
- Local connectivity:

- Each unit extracts local features from the previous layer
- Lower number of weights



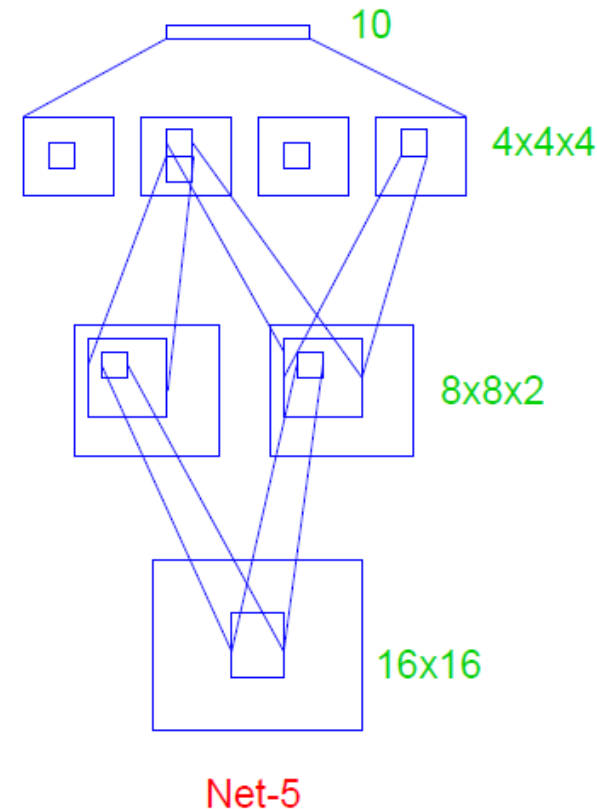
Example: ZIP Code Data (iii)

- Net-4: two hidden layers, locally connected, weight sharing
 - First hidden layer: two 8x8 feature maps
 - Input from 3x3 patch
 - Units in the same 8x8 feature map share the same set of 9 weights (bias not shared)
 - Extracted features in different parts of the image are computed with the same linear functional:
convolutional networks
 - Second hidden layer: no weight sharing
- The gradient of the error for a shared weight is the sum of the gradients to each connection controlled by the weight



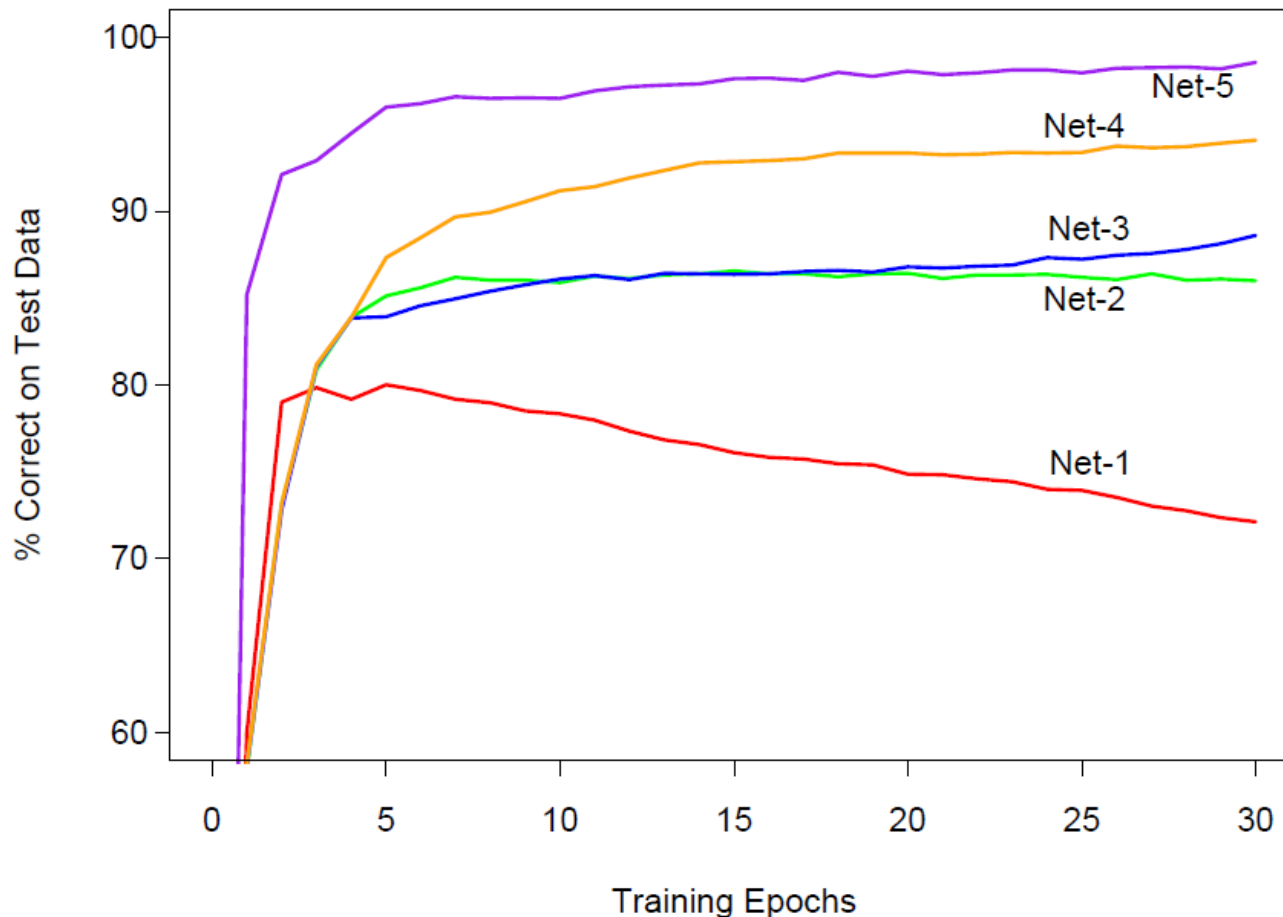
Example: ZIP Code Data (iv)

- Net-5: two hidden layers, locally connected, two levels of weight sharing
 - First hidden layer: same as Net-4
 - Input from a 5x5 patch
 - Weights shared in each of the feature maps
 - Second hidden layer: four 4x4 feature maps
 - Input from a 5x5 patch
 - Weights shared in each of the feature maps
- Features of handwritten style should appear in more than one part of a digit
- Subject matter knowledge should be used to improve performance



Example: ZIP Code Data (v)

- Training error was 0% in all the cases
 - More parameters than training observations



Example: ZIP Code Data (vi)

	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%

- Best results on a large database: Le Cun et al., 1998
 - 60,000 training and 10,000 test examples
 - LeNet-5: a more complex convolutional network
 - 99.2% correct
 - Boosted LeNet-4: boosting with a predecessor of LeNet-5
 - 99.3% correct

Bibliography

- C. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995.
 - Chapter 4
- T. Hastie, R. Tibshirani, y J. Friedman, The elements of statistical learning. Springer, 2009.
 - Chapter 11
- Unsupervised Feature Learning and Deep Learning Tutorial
 - <http://ufldl.stanford.edu/tutorial/>