Intro to Neural Networks

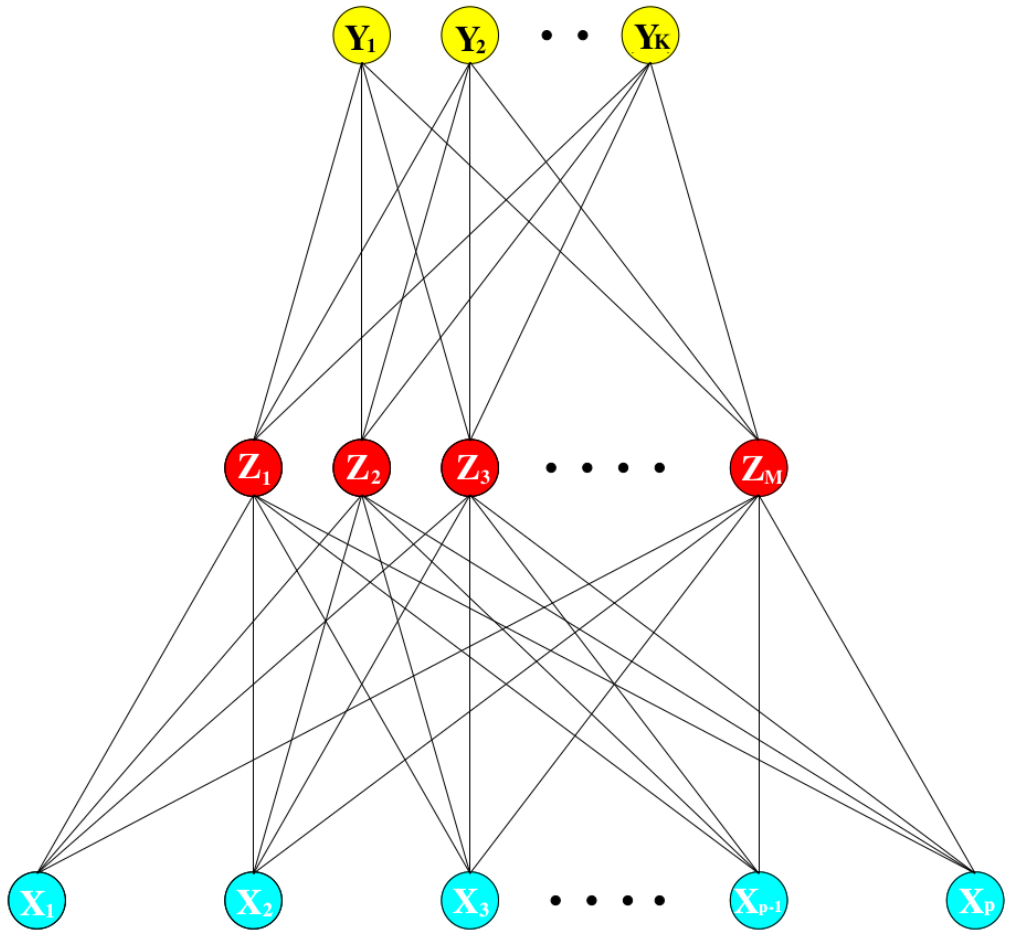
Matthew S. Shotwell, Ph.D.

Department of Biostatistics
Vanderbilt University School of Medicine
Nashville, TN, USA

April 8, 2020

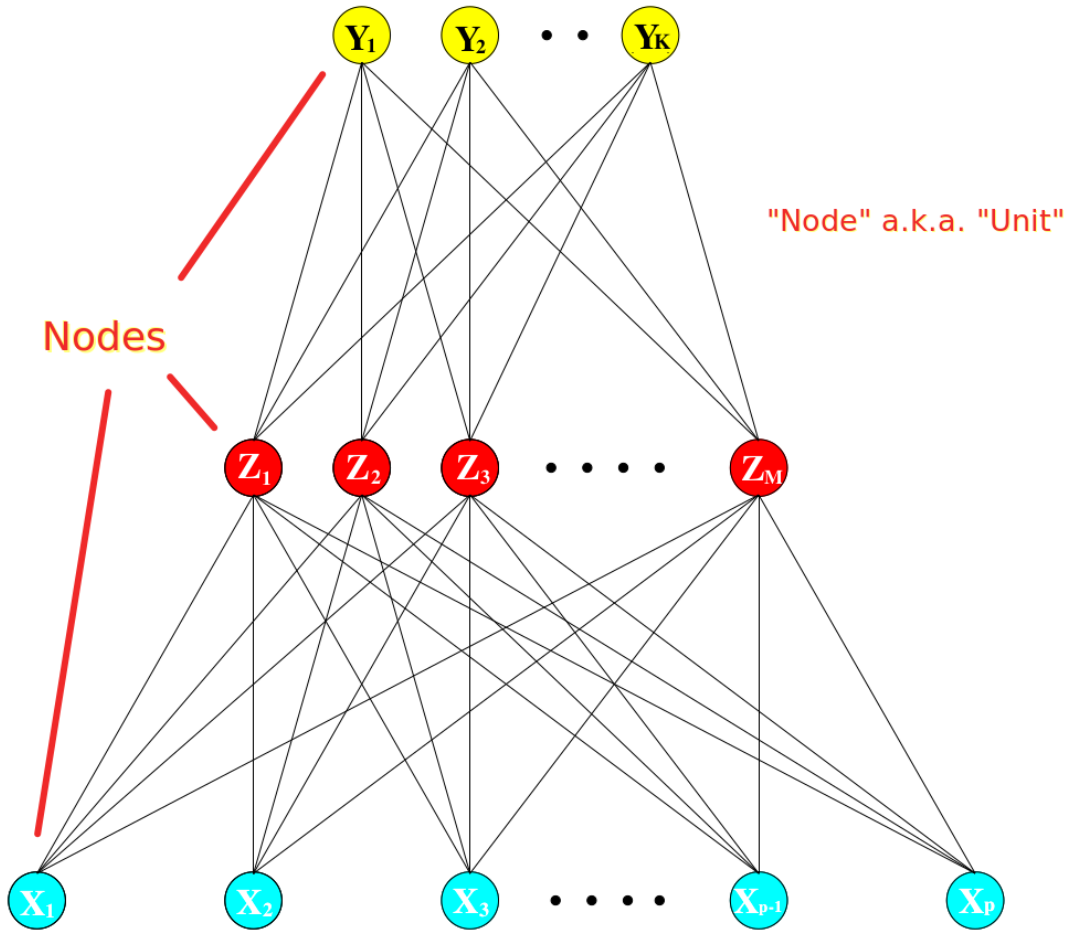
Neural Networks

NN is a nonlinear model, often represented by network diagram:



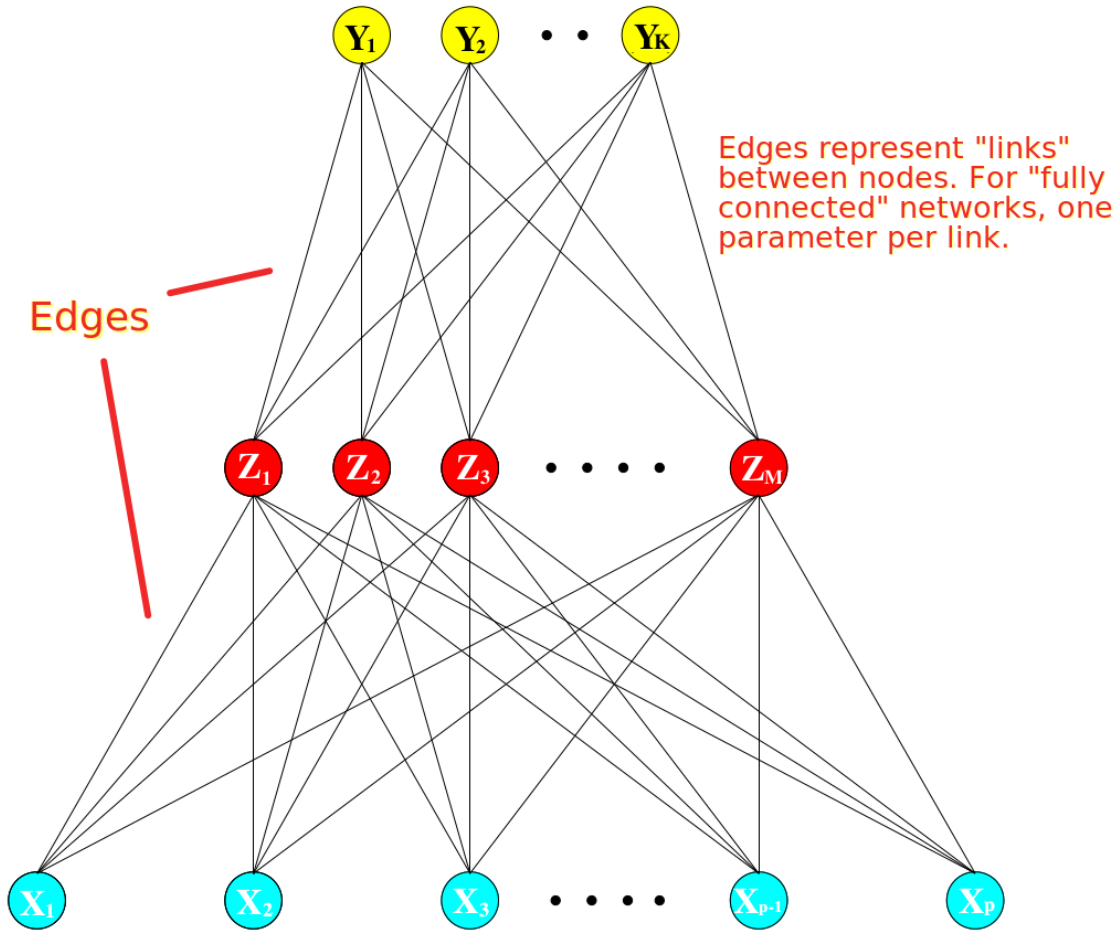
Neural Networks

NN is a nonlinear model, often represented by network diagram:



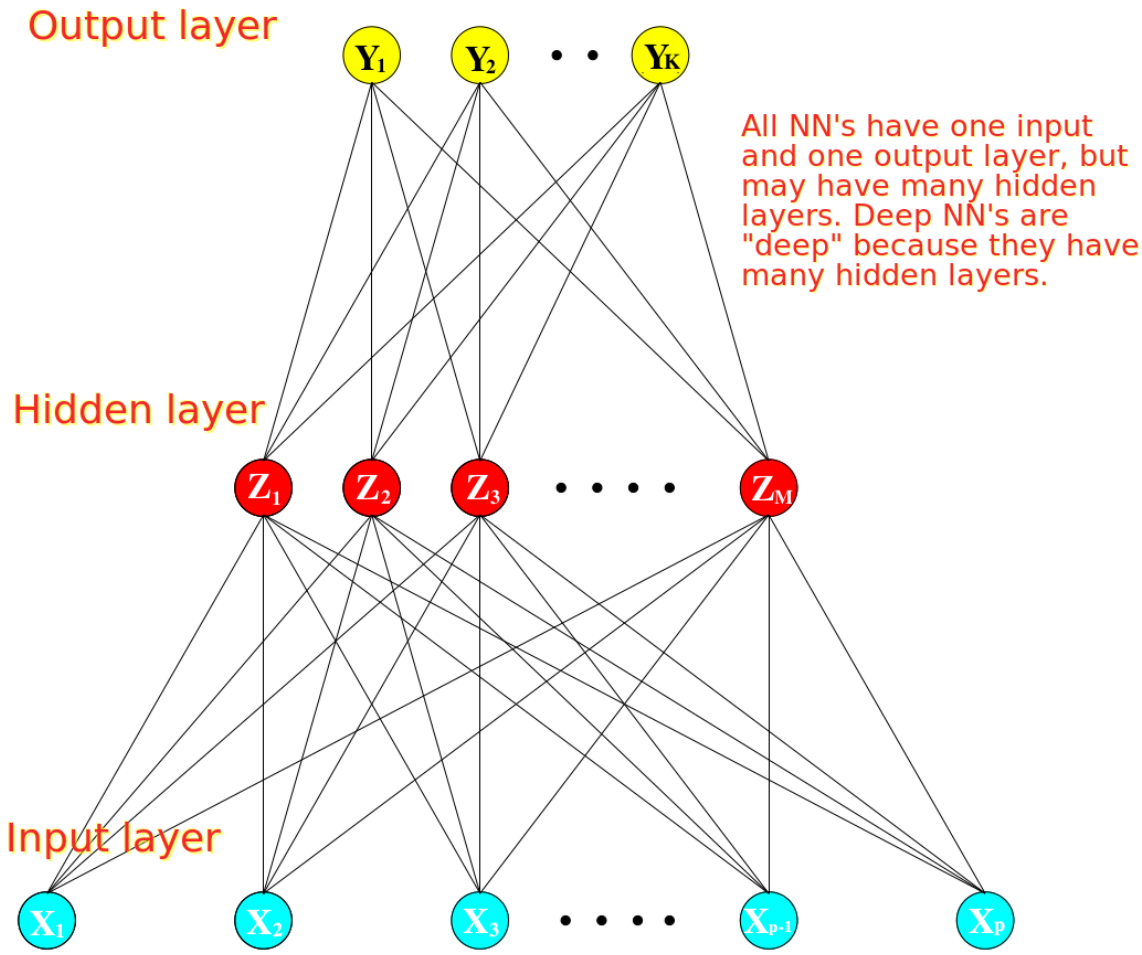
Neural Networks

NN is a nonlinear model, often represented by network diagram:



Neural Networks

NN is a nonlinear model, often represented by network diagram:



Neural Networks

- NN in previous diagram is for a K class problem
- output layer has K nodes, one for each class
- NN for regression would have just one output node

Neural Networks

Model formula for NN in previous figure; K class problem:

- input layer: P features, $x_1 \cdots, x_P$
- hidden layer: for each hidden node $m = 1, \ldots, M$

$$z_m = \sigma(\alpha_{0m} + \alpha_{1m} x_1 + \cdots + \alpha_{Pm} x_P)$$

- output layer: for each output node $k = 1, \ldots, K$

$$t_k = s(\beta_{0k} + \beta_{1k} z_1 + \cdots + \beta_{Mk} z_M)$$

- σ is an "activation" function
- s is a "link" function, e.g., logit, or softmax function

Neural Networks

Model formula for NN in previous figure; K class problem:

- input layer: P features, $x_1 \cdots, x_P$
- hidden layer: for each hidden node $m = 1, \ldots, M$

$$z_m = \sigma(\alpha_{0m} + \alpha_{1m} x_1 + \cdots + \alpha_{Pm} x_P)$$

- output layer: for each output node $k = 1, \ldots, K$

$$t_k = s(\beta_{0k} + \beta_{1k} z_1 + \cdots + \beta_{Mk} z_M)$$

- layers are "fully connected" to lower layer
- all nodes in lower layer contribute to each node of layer above

Neural Networks

Model formula for NN in previous figure; K class problem:

- input layer: P features, $x_1 \cdots, x_P$
- hidden layer: for each hidden node $m = 1, \ldots, M$

$$z_m = \sigma(\alpha_{0m} + \alpha_{1m} x_1 + \cdots + \alpha_{Pm} x_P)$$

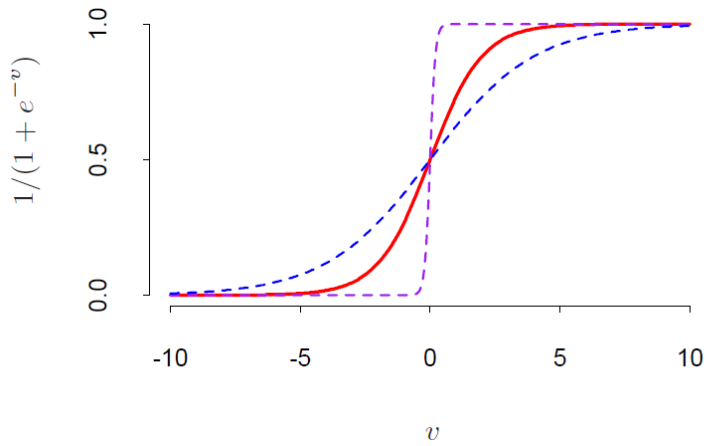
- output layer: for each output node $k = 1, \ldots, K$

$$t_k = s(\beta_{0k} + \beta_{1k} z_1 + \cdots + \beta_{Mk} z_M)$$

- α_{0m} and β_{0k} called "bias" parameters
- # params: hidden $M \times (P + 1)$; output $K \times (M + 1)$

σ Activation function

σ is an "activation function" designed to mimic the behavior of neurons in propagating signals in the (human) brain. The activation function makes the model nonlinear (in the parameters)



- sigmoid - $\sigma(x) = \frac{1}{1+e^{-x}}$
- ReLU - $\sigma(x) = \max(0, x)$
- ReLU - "rectified linear unit"
- ReLU - faster training vs sigmoid, may perform better

s Link function

The link function transforms the output nodes so that they make sense for the type of probelm: classification or regression.

- regression - identity link - $s(t) = t$
- classification - softmax link -

$$s(t_k) = \frac{e^{t_k}}{\sum_{l=1}^{K} e^{t_l}}$$

- $s(t_k)$ is a value between 0 and 1
- $s(t_k)$ is $P(y_k = 1|x)$ where y_k is target code for class k

Training (fitting) neural networks

- NN's often have large number of parameters: θ
- fit NN's by minimizing average loss in training data!
- Regression: $\overline{\mathrm{err}} = \sum_{i=1}^{N}(y_i - f(x_i, \theta))^2$
- Classificatin: $\overline{\mathrm{err}} = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i, \theta)$, where y_{ik} is the indicator for class k, and f_k gives the probability for class k. This is called the "cross-entropy" or "deviance"
- other loss functions can be used
- $\overline{\mathrm{err}}$ minimized w.r.t. θ using a gradient descent algorithm called "back-propagation" or "backprop"

Training (fitting) neural networks

- backprop iterates two steps:
- forward step: fix θ and compute values of hidden and output nodes z_m and t_k, and apply link function to get predictions $f(x_i)$ (probabilities for classification, or numerical prediction for regression)
- backward step: fix z_m, t_k, $f(x_i)$ and update θ using a gradient descent step

Training (fitting) neural networks

- usually don't want global minimum of $\overline{\text{err}}$ due to overfitting
- # of iters of backprop, learning rate (of gradient descent algorithm), and shrinkage penalties (weight decay, dropout) are tuning parameters

Training (fitting) neural networks

Shrinkage:

- weight decay: modified objective

$$\overline{\mathrm{err}}(\theta) + \lambda J(\theta)$$

where

$$J(\theta) = \sum_k \theta_k^2$$

like a ridge penalty, λ is tuning parameter

- dropout: at each round of training, some of the hidden or input nodes are Z_m or X_m are ignored (assigned a value zero); ignored inputs are selected at random at each round of backprop; number of ignored features is tuning parameter

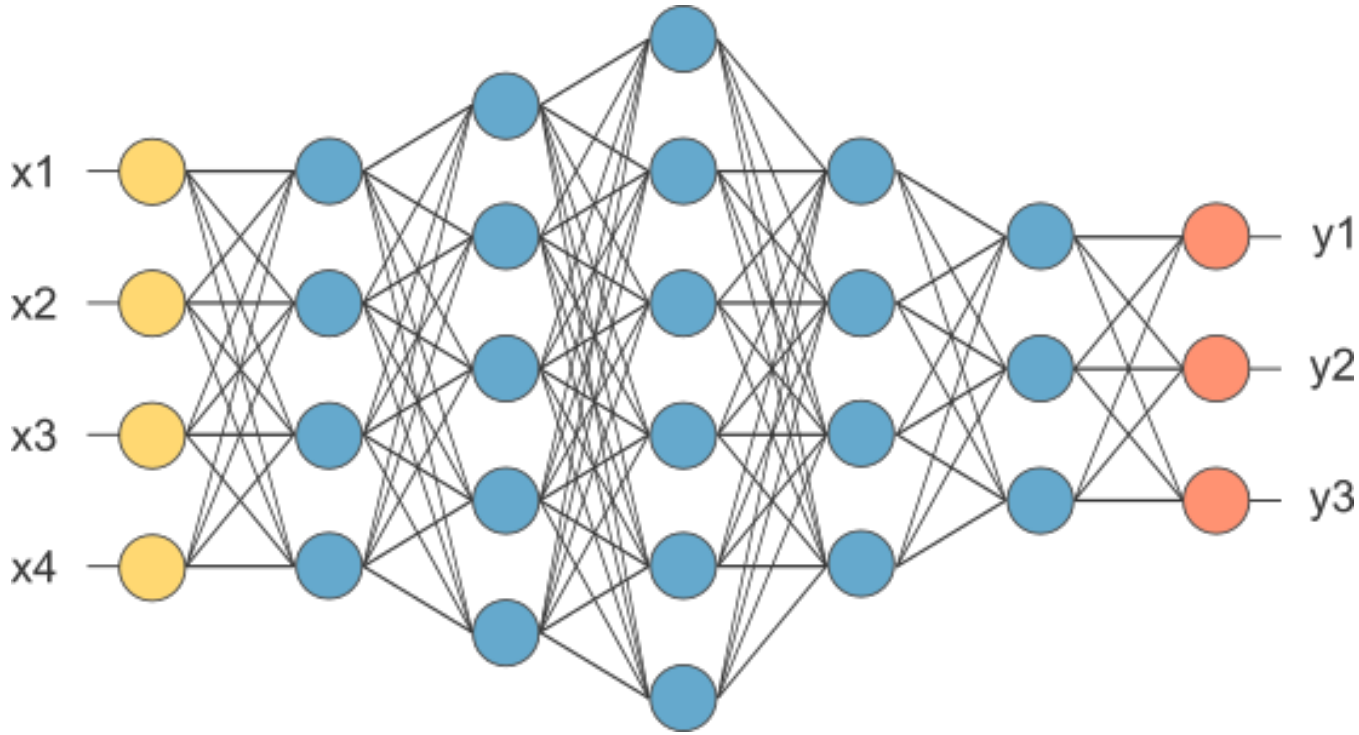
Simple NN in R: nnet.R

Extending NN's

- The real power of NN's comes through various extensions:
- Additional hidden layers
- Modifying connectivity between layers
- Processing between layers

Additional layers

More than one hidden layer:



Modified connectivity

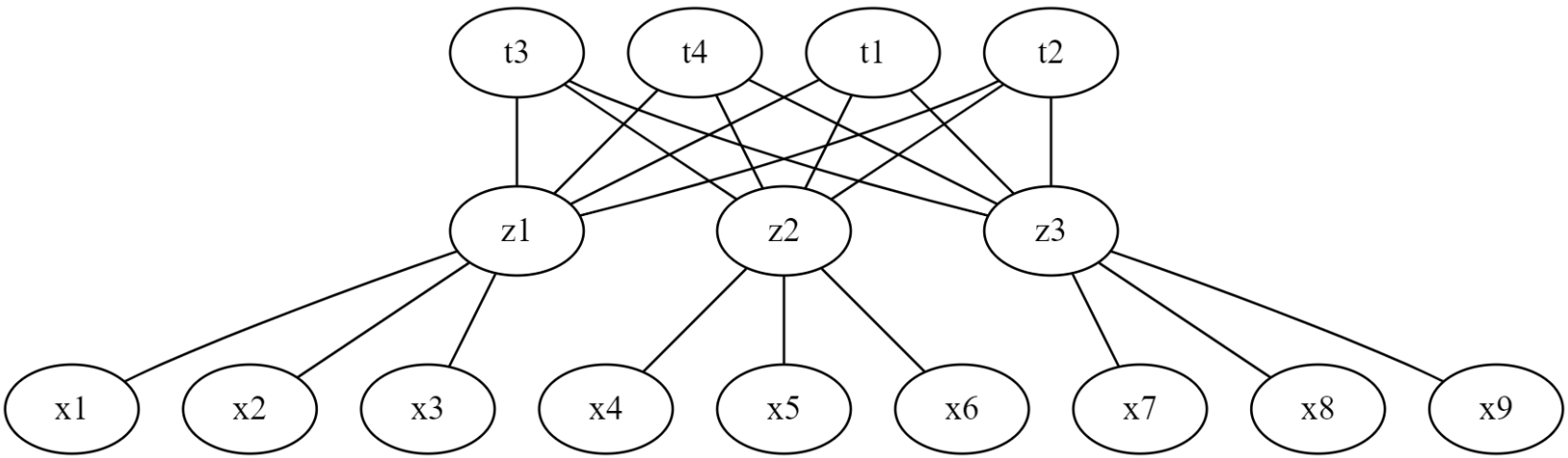
- local connectivity: hidden units only receive input from a subset of "local" units in the layer below; not "fully" connected
- say there are 3 hidden nodes and 9 features

$$z_1 = \sigma(\alpha_{01} + \alpha_{11}x_1 + \alpha_{21}x_2 + \alpha_{31}x_3)$$
$$z_2 = \sigma(\alpha_{02} + \alpha_{12}x_4 + \alpha_{22}x_5 + \alpha_{32}x_6)$$
$$z_3 = \sigma(\alpha_{03} + \alpha_{13}x_7 + \alpha_{23}x_8 + \alpha_{33}x_9)$$

- each hidden node linked to just 3 of 9 features
- hidden layer just $3 \times 4 = 12$ parameters instead of 30
- output layer typically always fully connected

Modified connectivity

Local connectivity



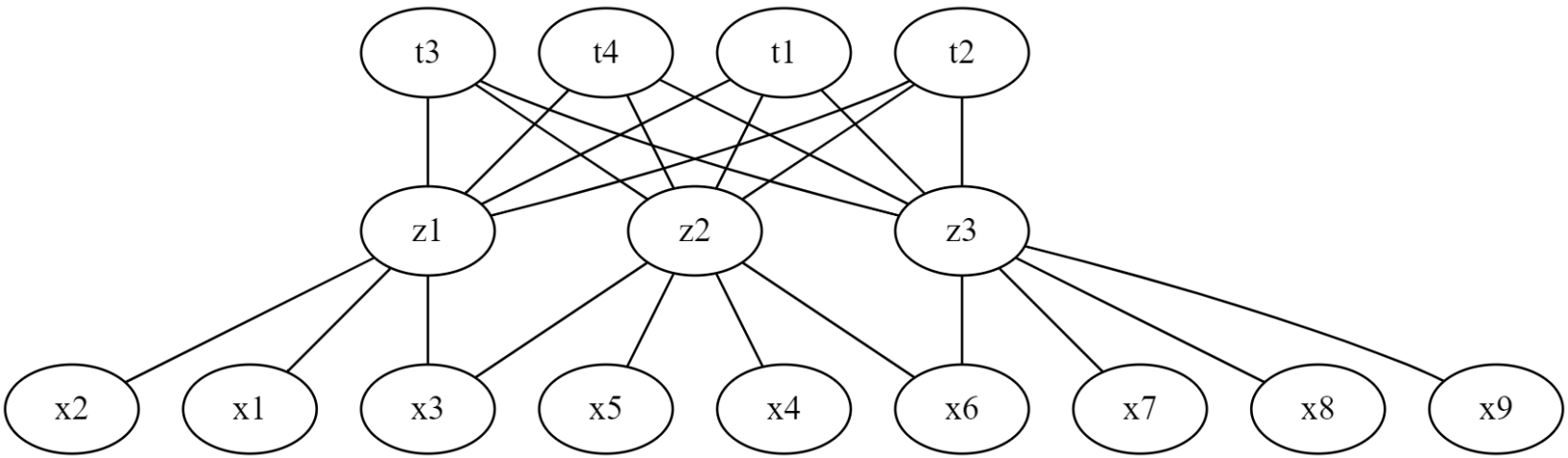
Modified connectivity

- local connectivity: there can be some "overlap": some hidden nodes can take input from some of the same featuresw
- say there are 3 hidden nodes and 9 features

$$z_1 = \sigma(\alpha_{01} + \alpha_{11}x_1 + \alpha_{21}x_2 + \alpha_{31}x_3)$$
$$z_2 = \sigma(\alpha_{02} + \alpha_{12}x_3 + \alpha_{22}x_4 + \alpha_{32}x_5 + \alpha_{42}x_6)$$
$$z_3 = \sigma(\alpha_{03} + \alpha_{13}x_6 + \alpha_{23}x_7 + \alpha_{33}x_8 + \alpha_{43}x_9)$$

Modified connectivity

Local connectivity



Modified connectivity

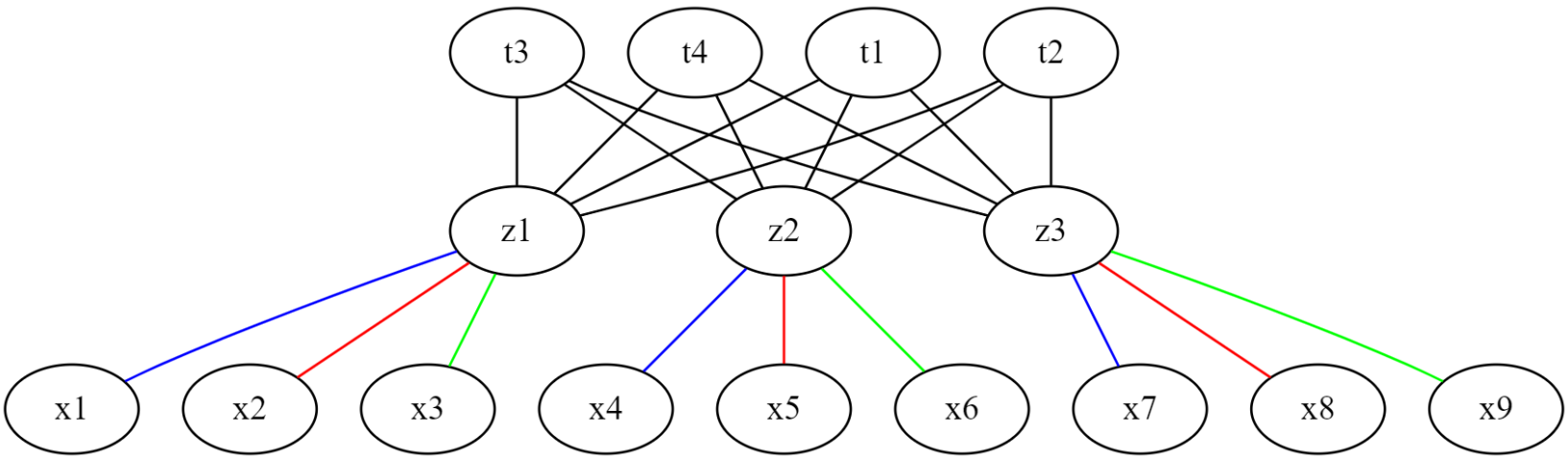
- weight sharing: some hidden units share weights
- only makes sense in combination with local connectivity
- say there are 3 hidden nodes and 9 features

$$z_1 = \sigma(\alpha_{01} + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3)$$
$$z_2 = \sigma(\alpha_{02} + \alpha_1 x_4 + \alpha_2 x_5 + \alpha_3 x_6)$$
$$z_3 = \sigma(\alpha_{03} + \alpha_1 x_7 + \alpha_2 x_8 + \alpha_3 x_9)$$

- typically each hidden node retains a distinct bias (α_{0m})
- hidden layer has just $3 + 3 = 6$ parameters
- number of links $\neq$ number of parameters (more links)

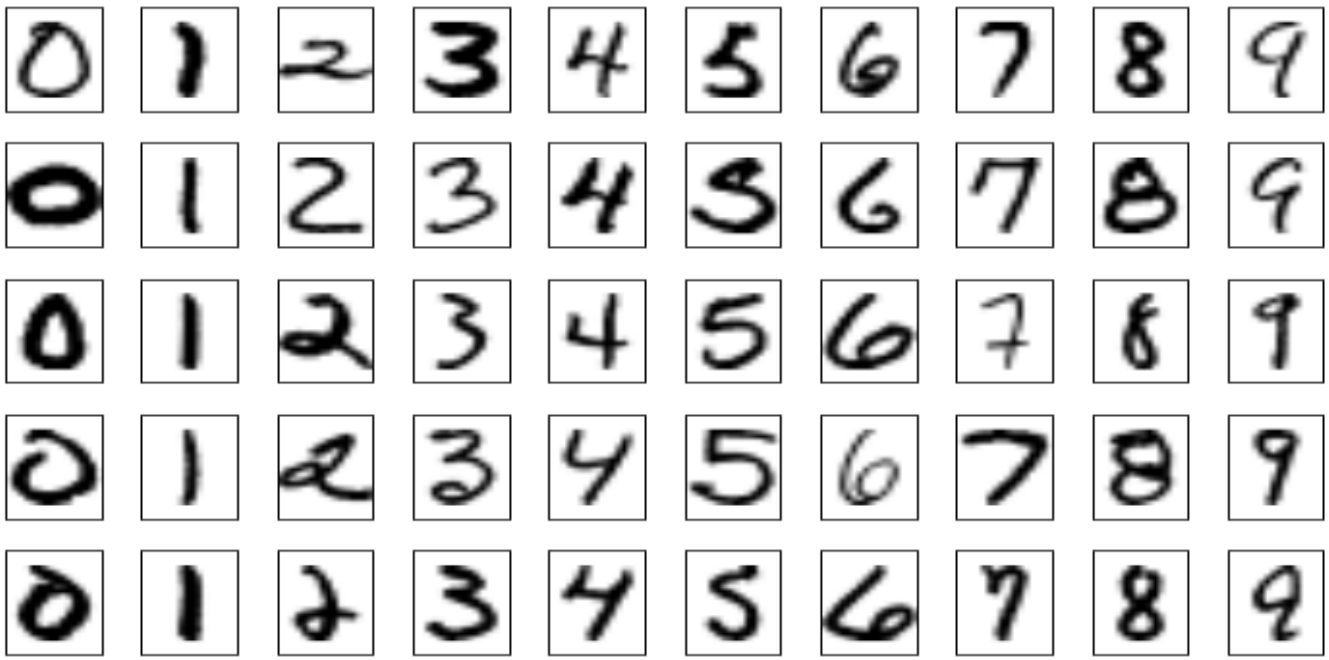
Modified connectivity

Local connectivity + weight sharing



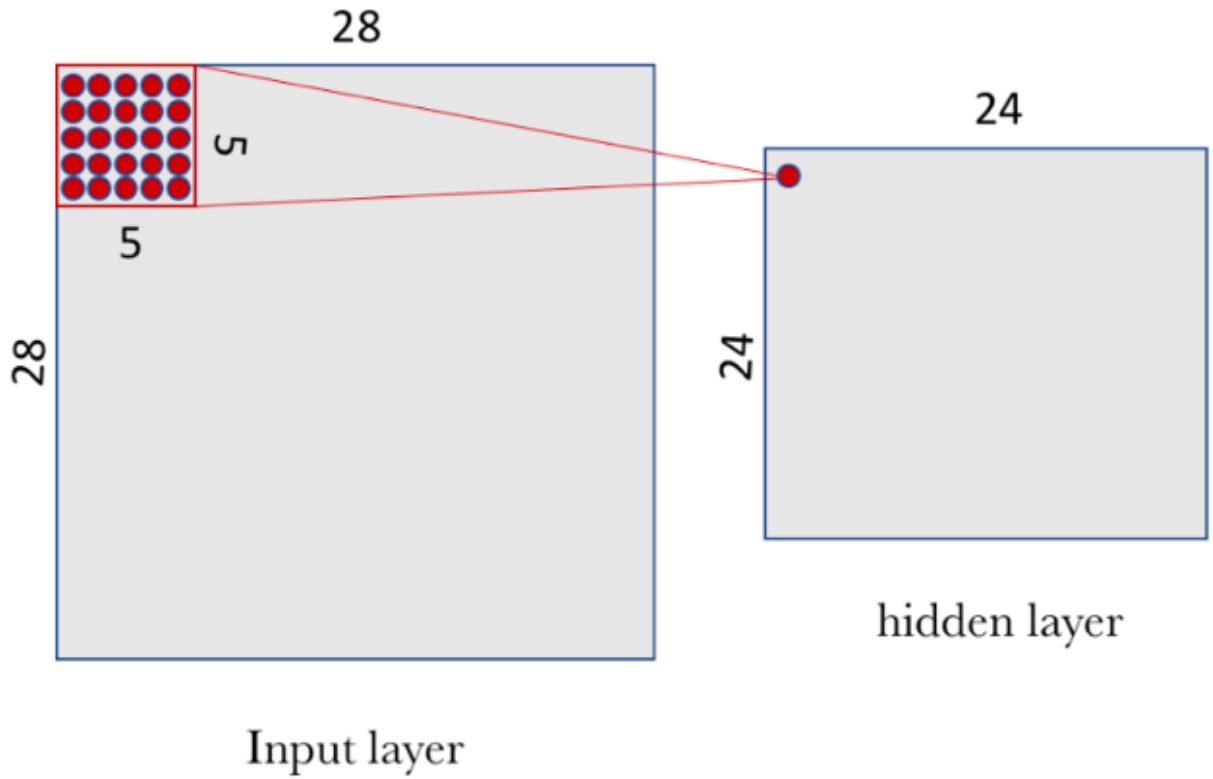
Example: zipcode data

- hand-written integers
- output: 10-class classification
- input: 16x16 B&W image



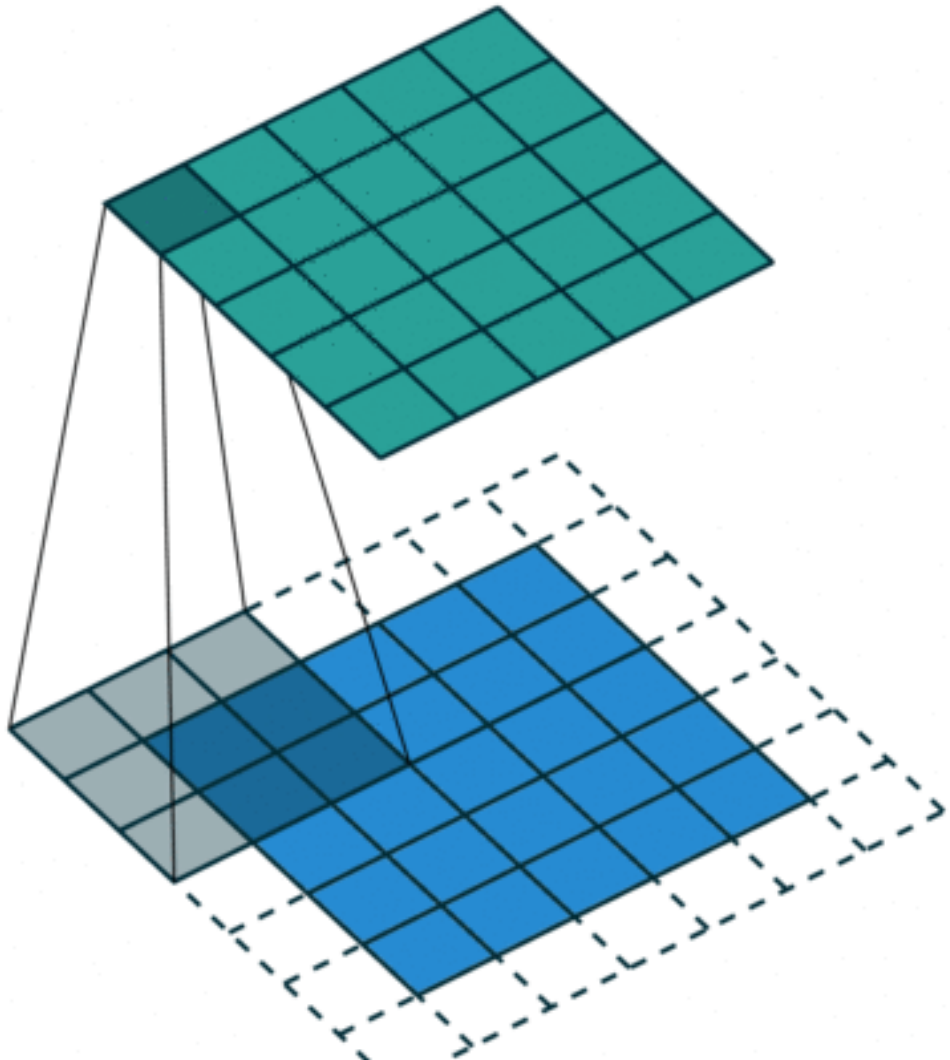
Example: zipcode data

- input: 16x16 B&W image
- when input is an array (2D in this case), typically the hidden units in a hidden layer are represented as an array too
- figure below shows local connectivity with 5×5 "kernel"



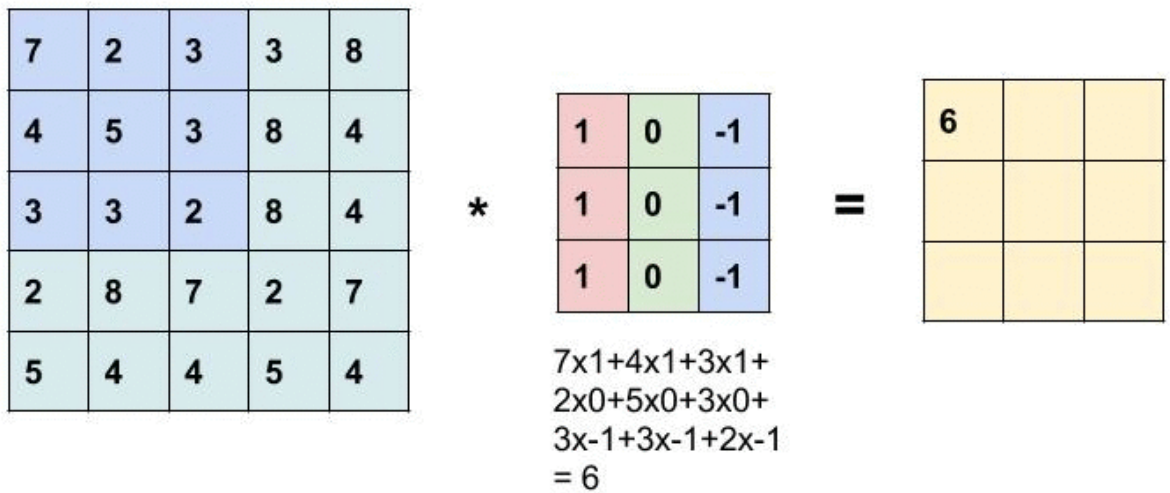
Local connectivity

- 3×3 kernel
- stride 1
- edge handling



Local conn. w/weight sharing (convolution)

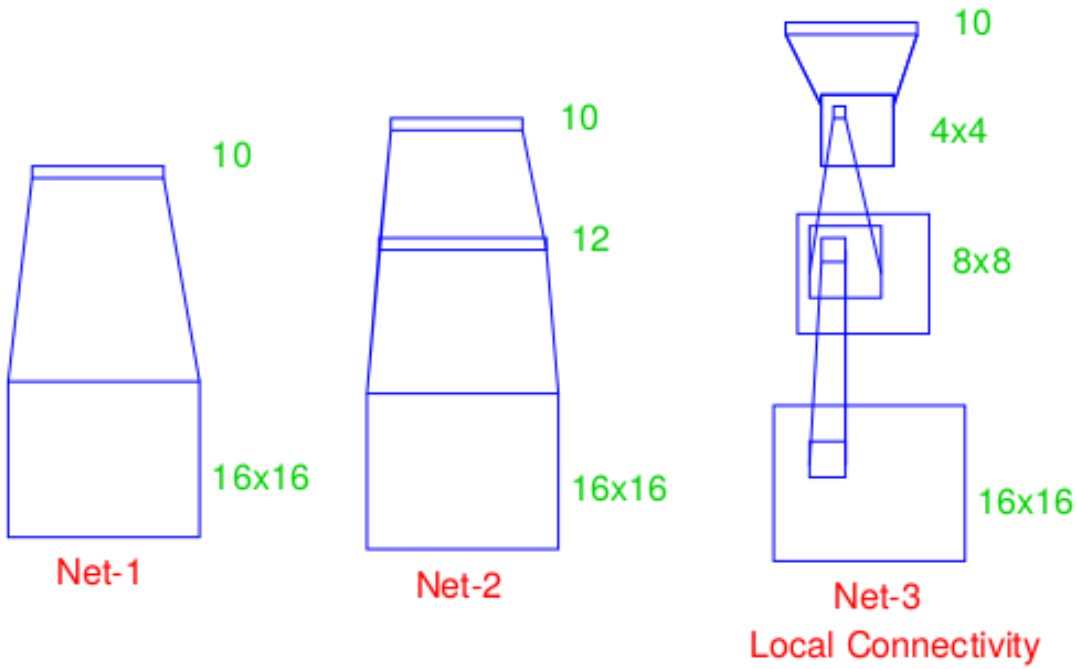
Convolution is type of shape detector or "feature map":



Example: zipcode data

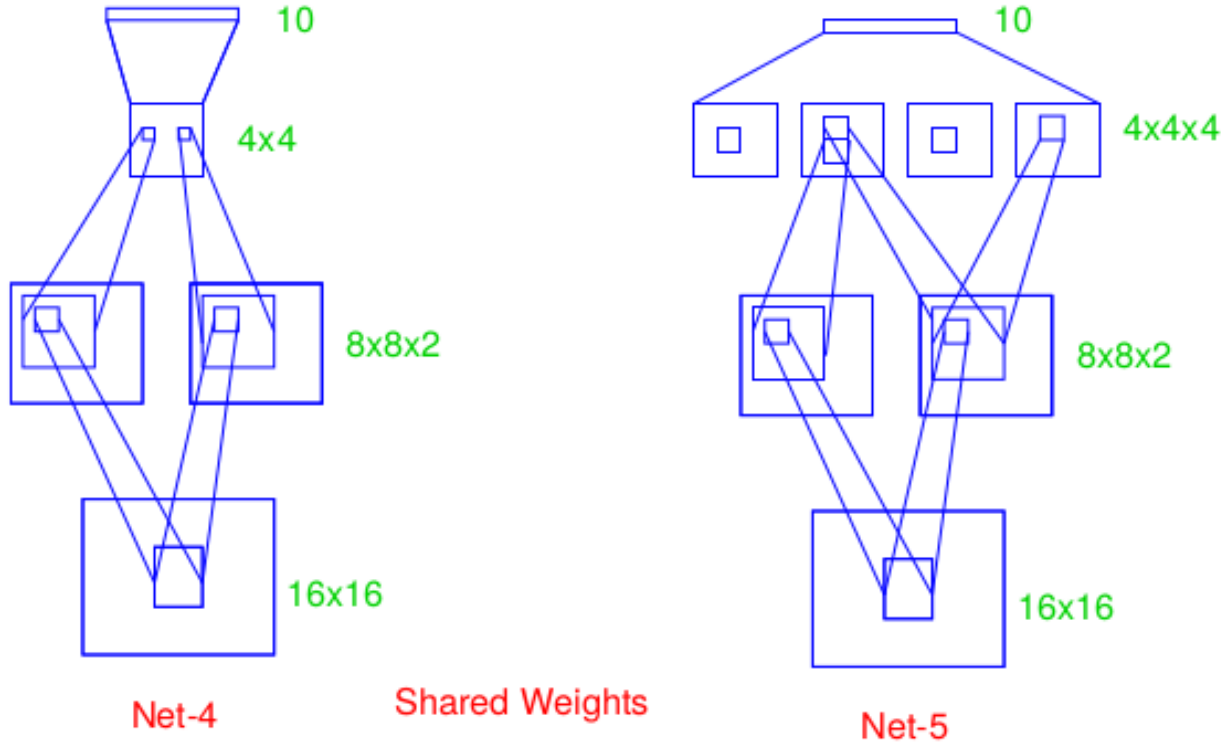
- Net 1: no hidden layer, same as multinomial logistic regression, $(256+1)10 = 2570$ parameters
- Net 2: one hidden layer, 12 units, fully connected, $(256+1)12 + (12+1)10 = 3214$ parameters
- Net 3: two hidden layers, locally connected, 1226 parameters
- Net 4: two hidden layers, locally connected, weight sharing, 1132 parameters, 2266 links

Local connectivity



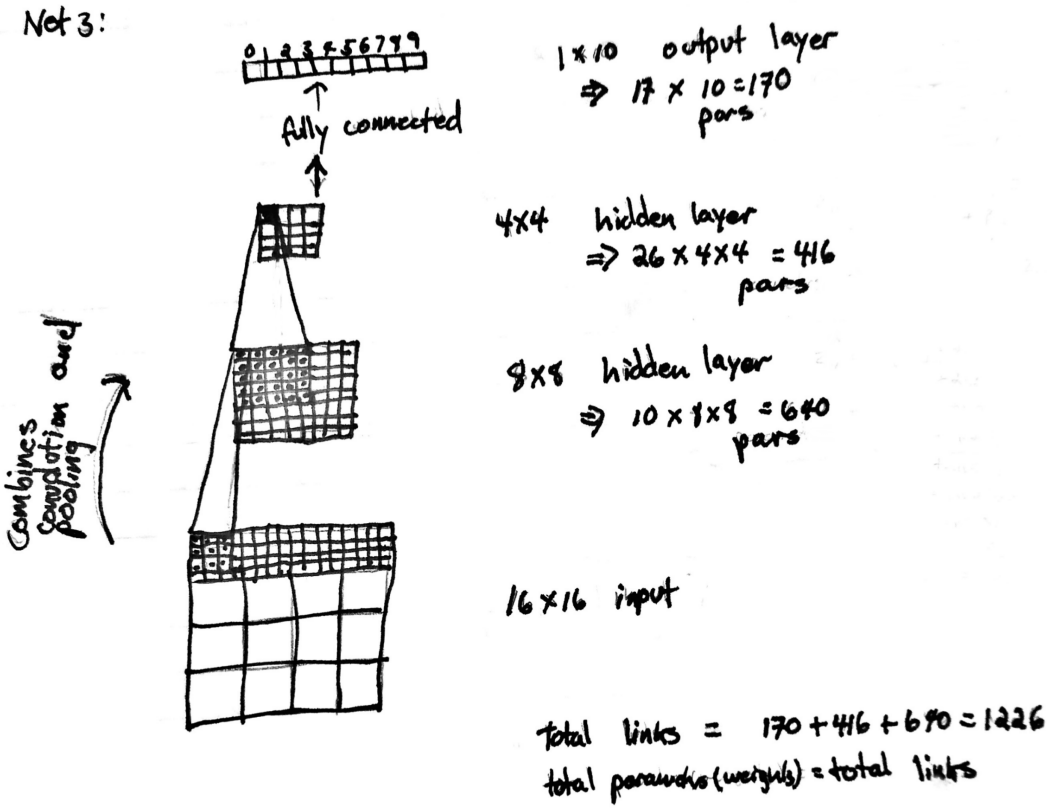
Local connectivity and shared weights

AKA: convolutional neural networks

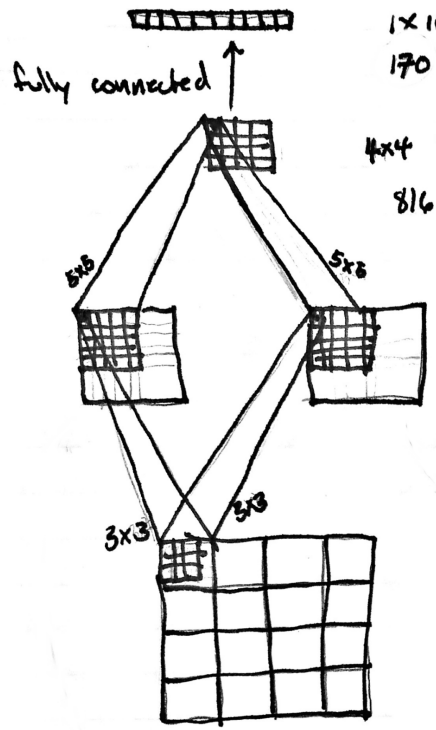


- groups of hidden units form "shape detectors" or "feature maps"
- more complex shape detectors near output layer

Net 3


Net 3:
0 1 2 3 4 5 6 7 8 9
fully connected
combines convolution and pooling
1×10 output layer
⇒ 17 × 10 = 170 pars
4×4 hidden layer
⇒ 26 × 4×4 = 416 pars
8×8 hidden layer
⇒ 10 × 8×8 = 640 pars
16×16 input
total links = 170 + 416 + 640 = 1226
total parameters (weights) = total links

Net 4



1x10 output layer
170 = 17x10 pars + links

4x4 hidden layer w/convolution
816 = 4x4x51 pars + links

8x8x2 hidden layer w/convolution + weight sharing
146 = (64+9)x2 pars
1280 = 64x10x2 links

16x16

170 + 816 + 146 = 1132 pars
170 + 816 + 1280 = 2266 links

Performance on zipcode data

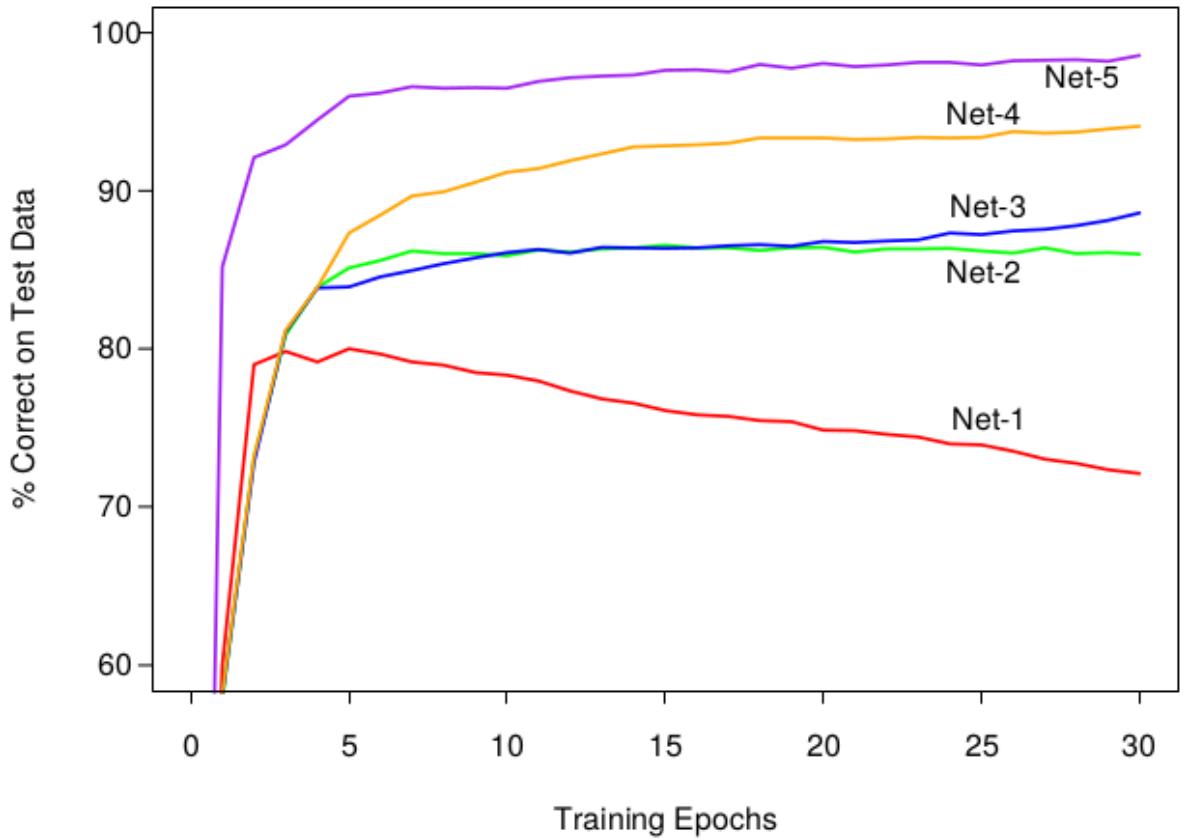
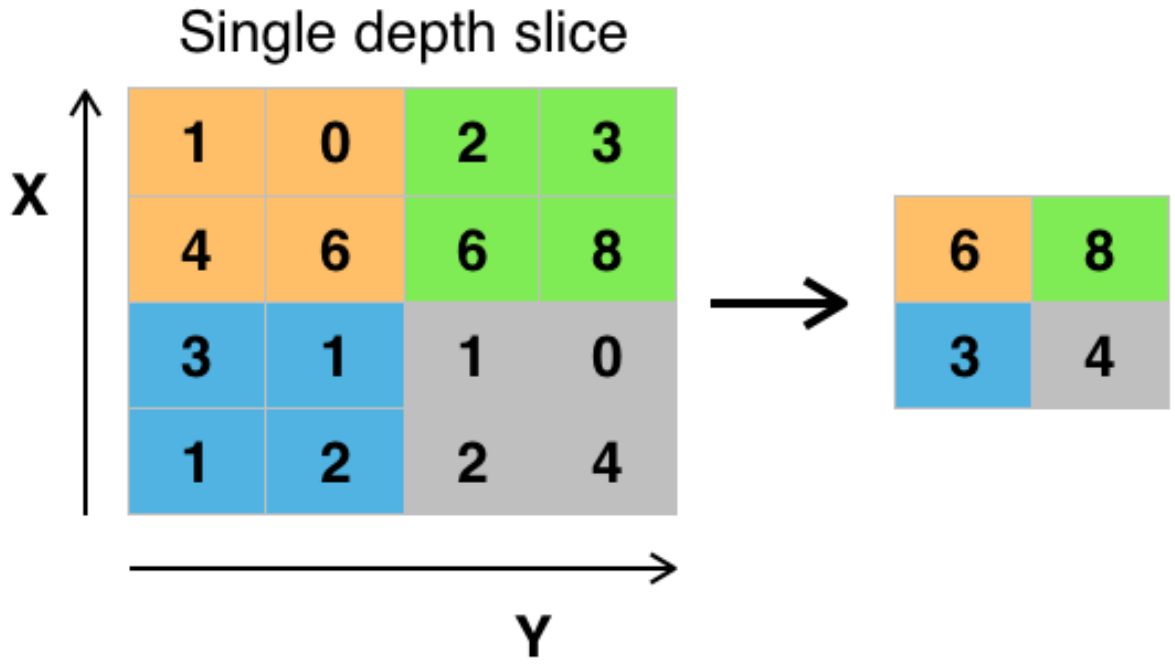


FIGURE 11.11. *Test performance curves, as a function of the number of training epochs, for the five networks of Table 11.1 applied to the ZIP code data. (Le Cun, 1989)*

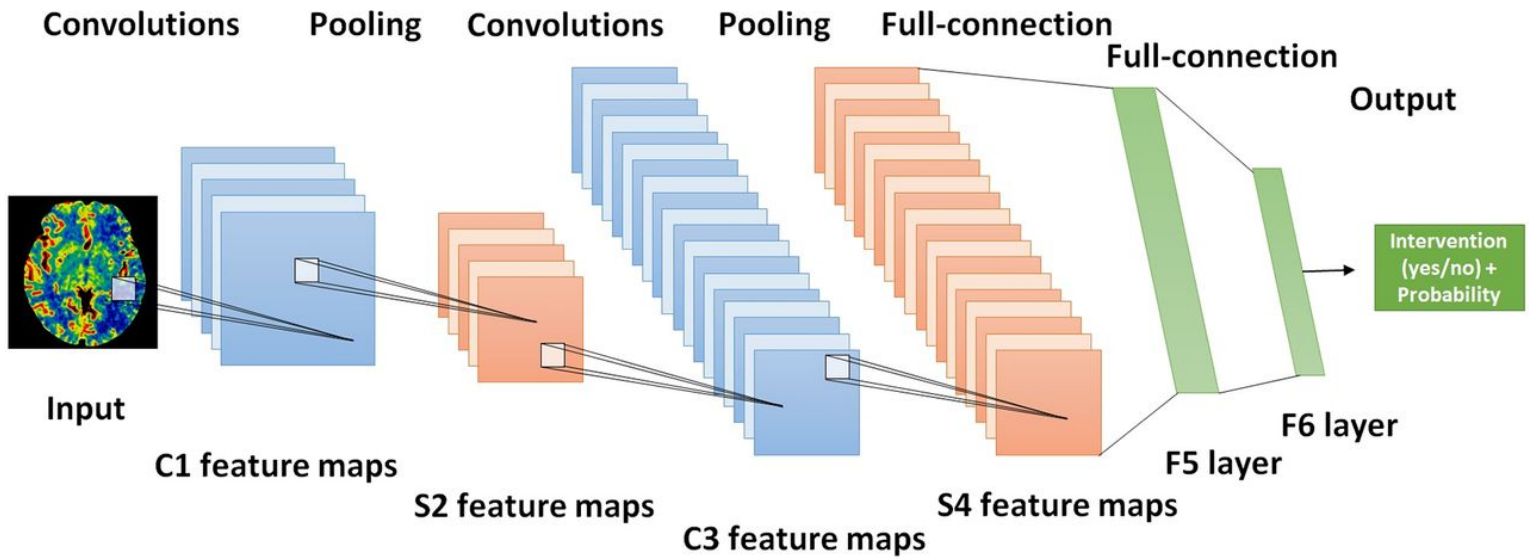
Processing between layers

- Pooling/subsampling: down-sample data from a layer by summarizing of a group of units
- Max-pooling: summarize using maximum:



Deep learning and deep neural networks

- Deep learning uses deep NNs
- Deep NNs are simply NNs with many layers, complex connectivity, and processing steps between layers:



Complex NNs in R

- No (good) native R libraries for complex NNs
- R can interface to good libraries, e.g., Keras, TensorFlow
- See https://keras.rstudio.com/