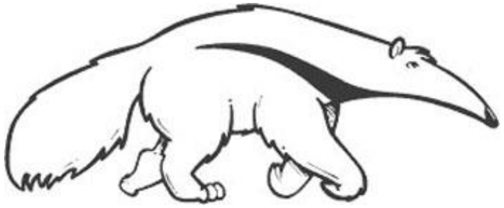
# Machine Learning and Data Mining

# Multi-layer Perceptrons & Neural Networks: Basics

Kalev Kask



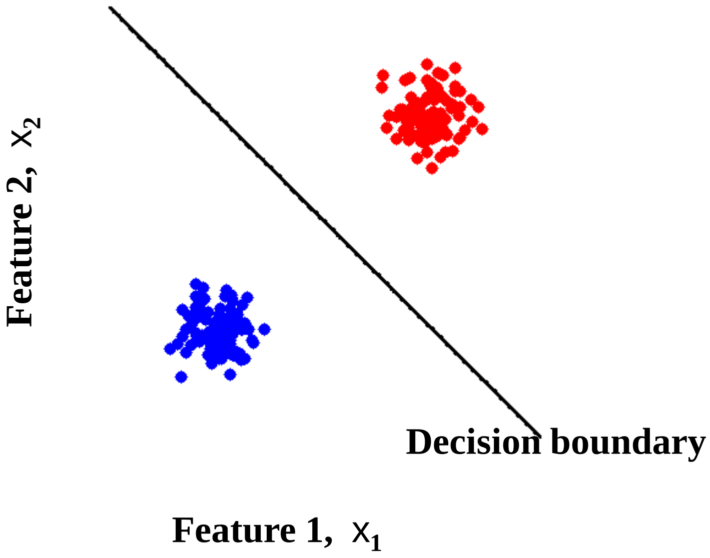
BREN:ICS
INFORMATION AND COMPUTER SCIENCES

UNIVERSITY of CALIFORNIA IRVINE

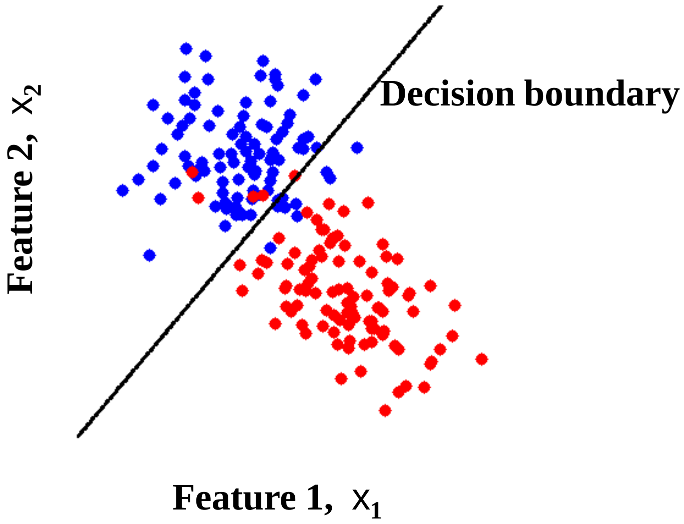
# Linear classifiers (perceptrons)

- Linear Classifiers
  - a linear classifier is a mapping which partitions feature space using a linear function (a straight line, or a hyperplane)
  - separates the two classes using a straight line in feature space
  - in 2 dimensions the decision boundary is a straight line

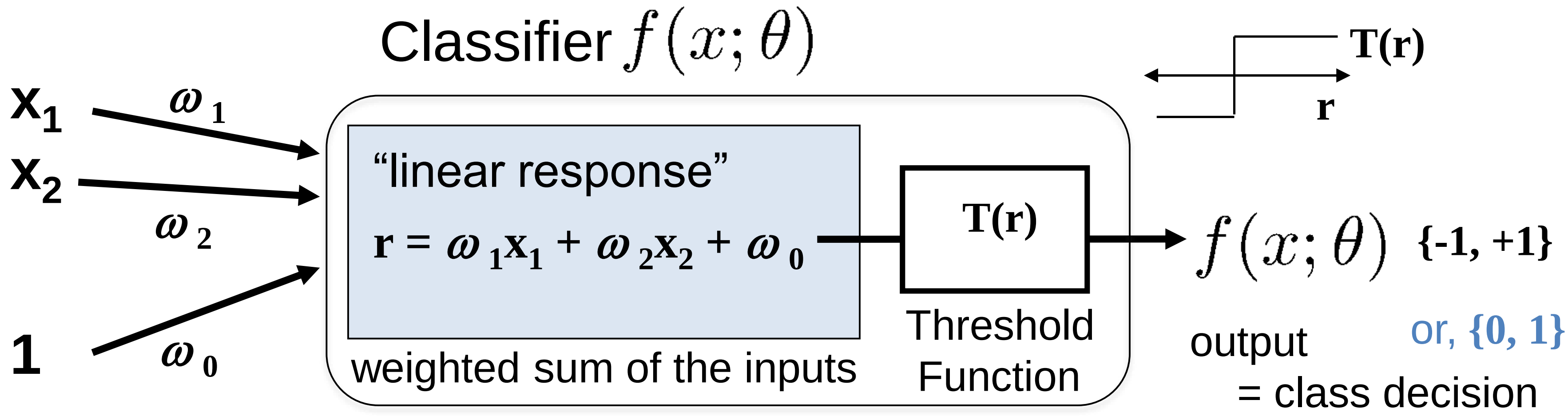
Linearly separable data




Linearly non-separable data



# Perceptron Classifier (2 features)

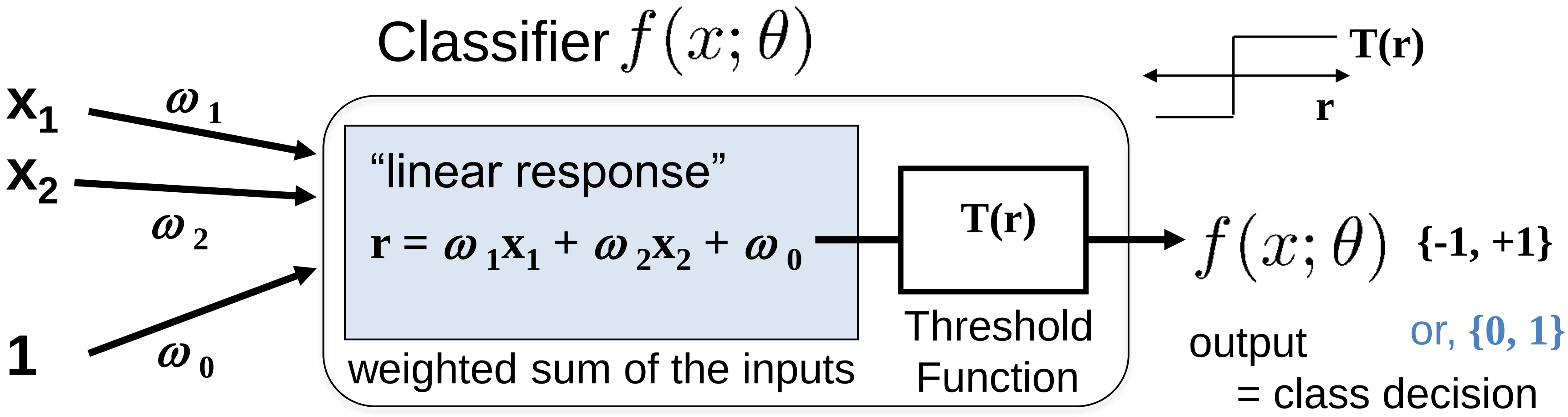



```
r = X.dot( theta.T );          # compute linear response
Yhat = 2*(r > 0)-1             # "sign": predict +1 / -1
```

Decision Boundary at $r(x) = 0$

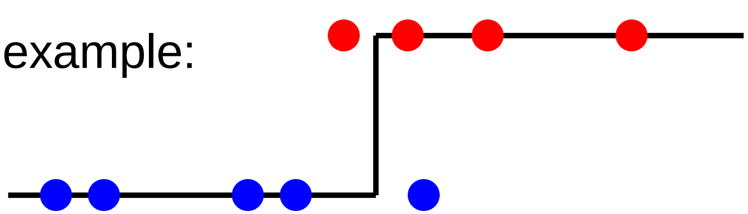
Solve: $X_2 = -w_1/w_2 \, X_1 - w_0/w_2$ (Line)

# Perceptron Classifier (2 features)




```
r = X.dot( theta.T );          # compute linear response
Yhat = 2*(r > 0)-1             # "sign": predict +1 / -1
```

1D example:



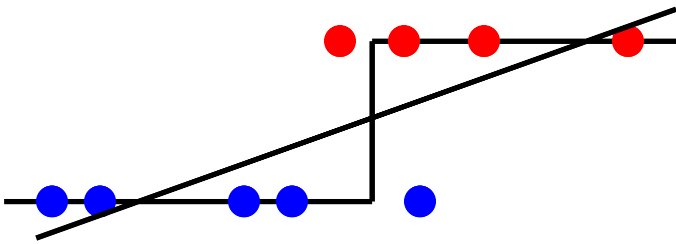
T(r) = -1 if r < 0

T(r) = +1 if r > 0

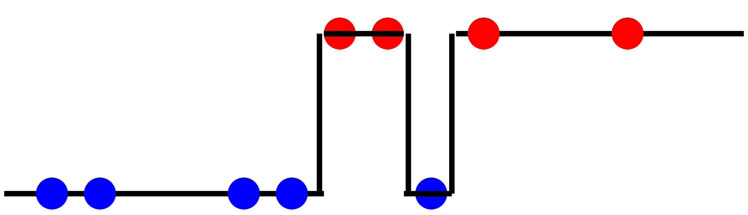
Decision boundary = "x such that T( $w_1$ x + $w_0$ ) transitions"

# Features and perceptrons

- Recall the role of features
  - We can create extra features that allow more complex decision boundaries
  - Linear classifiers
  - Features [1,x]
    - Decision rule: T(ax+b) = ax + b >/< 0
    - Boundary ax+b =0 => point
  - Features [1,x,$x^2$]
    - Decision rule T($ax^2$+bx+c)
    - Boundary $ax^2$+bx+c = 0 = ?



  - What features can produce this decision rule?

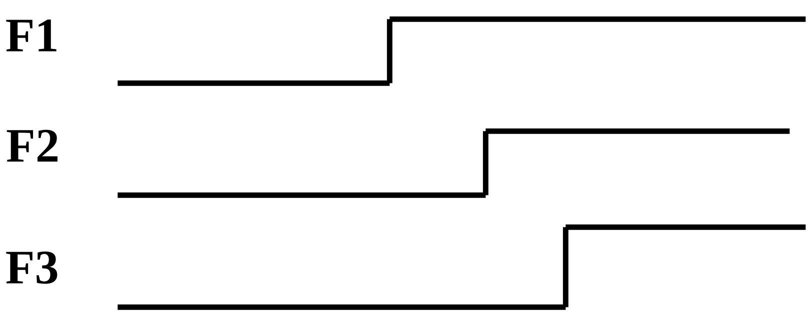


# Features and perceptrons

- Recall the role of features
  - We can create extra features that allow more complex decision boundaries
  - For example, polynomial features

    $\Phi(x) = [1 \;\; x \;\; x^2 \;\; x^3 \; \ldots]$

- What other kinds of features could we choose?
  - Step functions?

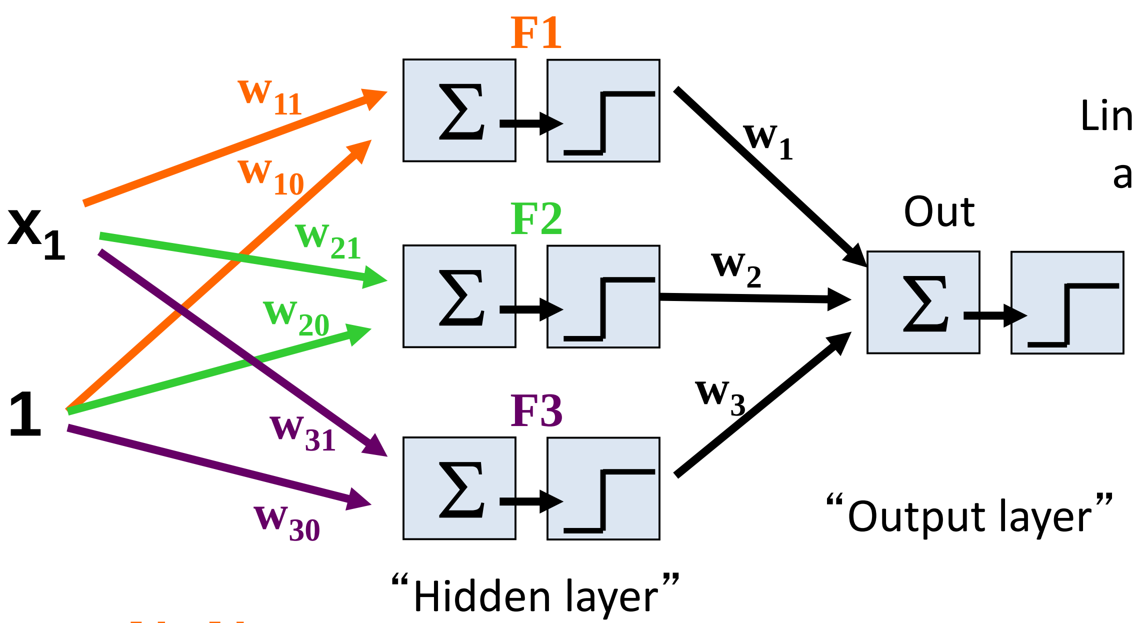



**Linear function of features**
**a F1 + b F2 + c F3 + d**

**Ex: F1 – F2 + F3**

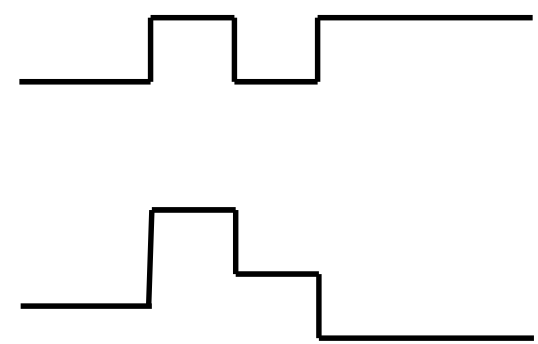
# Multi-layer perceptron model

- Step functions are just perceptrons!
  - "Features" are outputs of a perceptron
  - Combination of features output of another




Linear function of features:
a F1 + b F2 + c F3 + d

Ex: F1 – F2 + F3

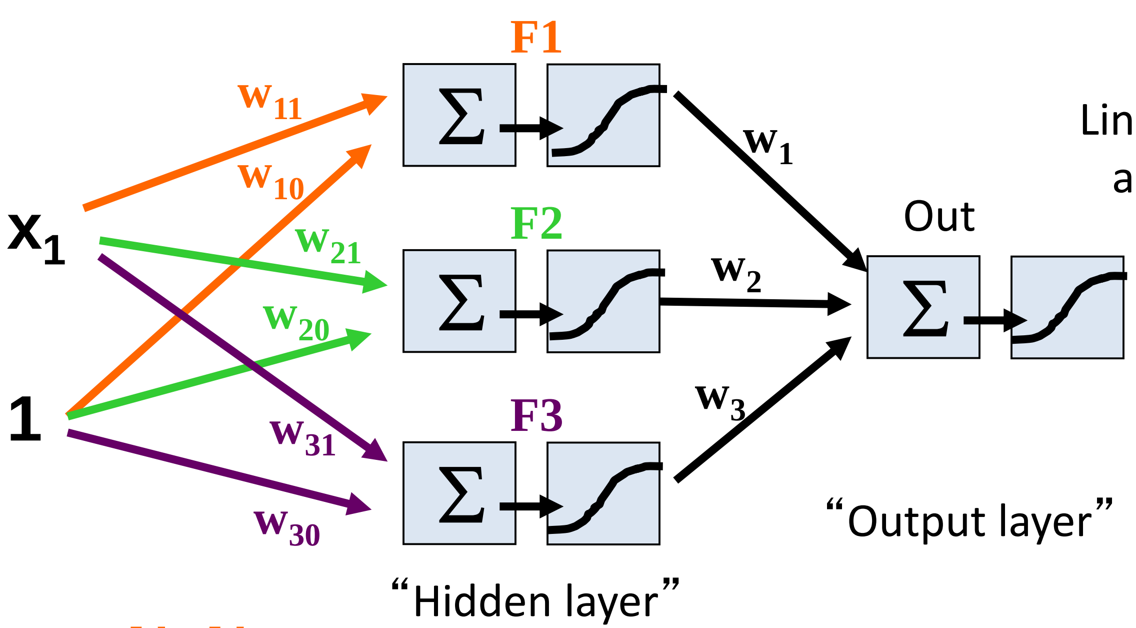


$$W^1 = \begin{matrix} w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{matrix} \qquad W^2 = w_1 \;\; w_2 \;\; w_3$$

# Multi-layer perceptron model

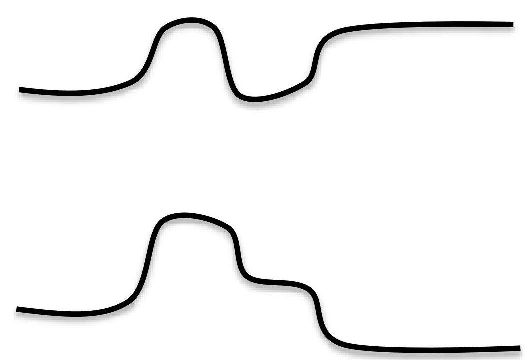
- Step functions are just perceptrons!
  - "Features" are outputs of a perceptron
  - Combination of features output of another

Regression version:
Remove activation
function from output




Linear function of features:
a F1 + b F2 + c F3 + d

Ex: F1 – F2 + F3

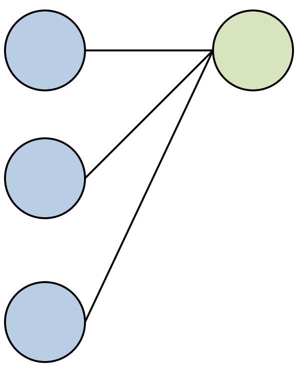


$$W^1 = \begin{bmatrix} w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{bmatrix} \qquad W^2 = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}$$

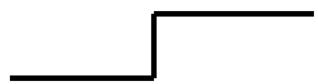
# Features of MLPs

- Simple building blocks
  - Each element is just a perceptron f'n

- Can build upwards



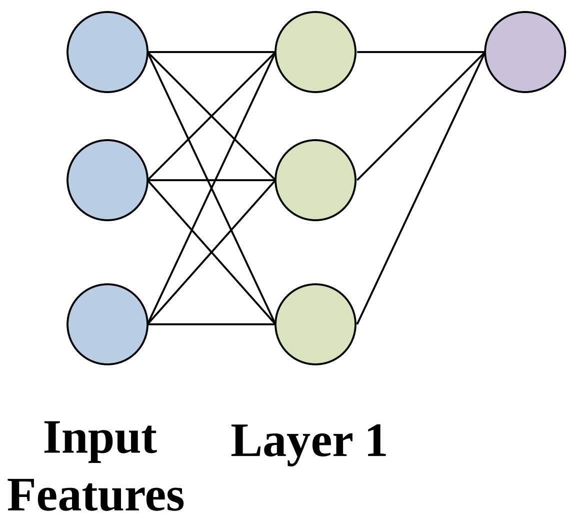
**Input Features**

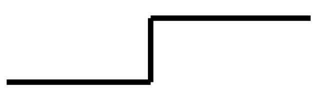


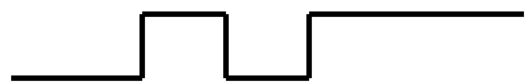
Perceptron:
  Step function /
  Linear partition

# Features of MLPs

- Simple building blocks
  - Each element is just a perceptron f’n
- Can build upwards



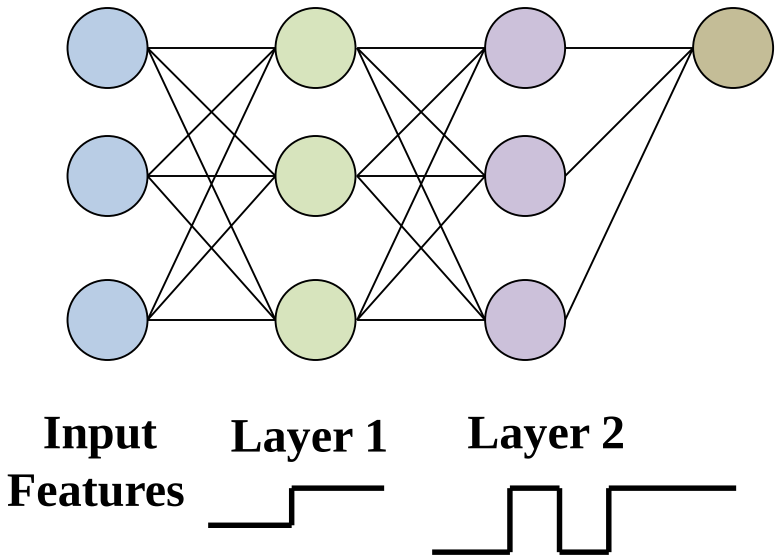





2-layer:
“Features” are now partitions
All linear combinations of those partitions

# Features of MLPs

- Simple building blocks
  - Each element is just a perceptron f'n
- Can build upwards

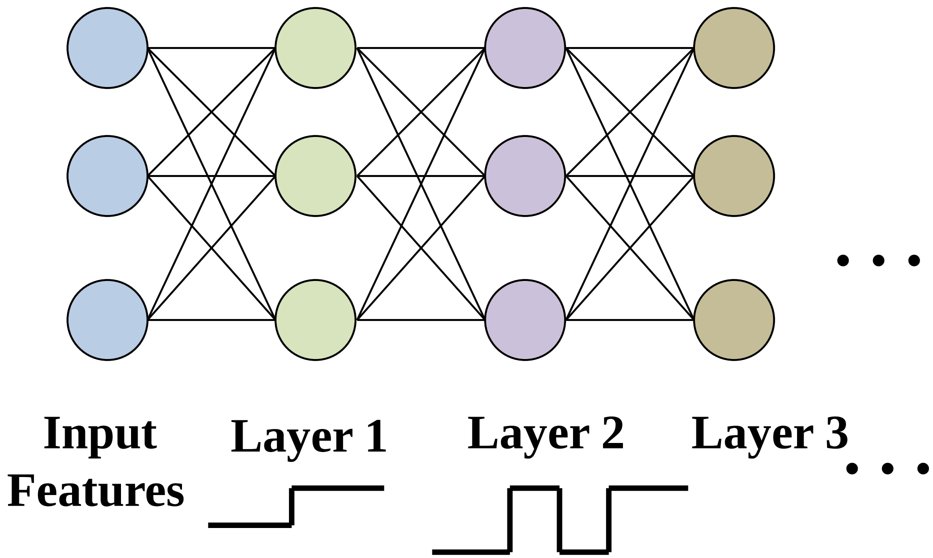



3-layer:
"Features" are now complex functions
Output any linear combination of those

# Features of MLPs

- Simple building blocks
  - Each element is just a perceptron f’n

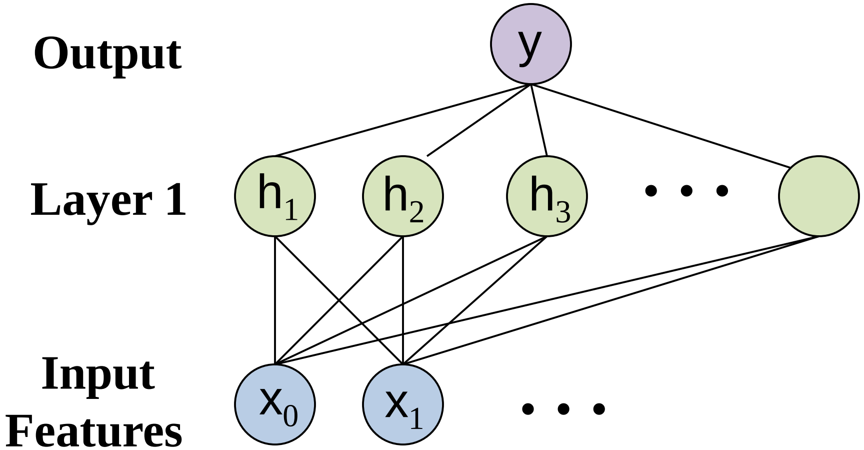
- Can build upwards

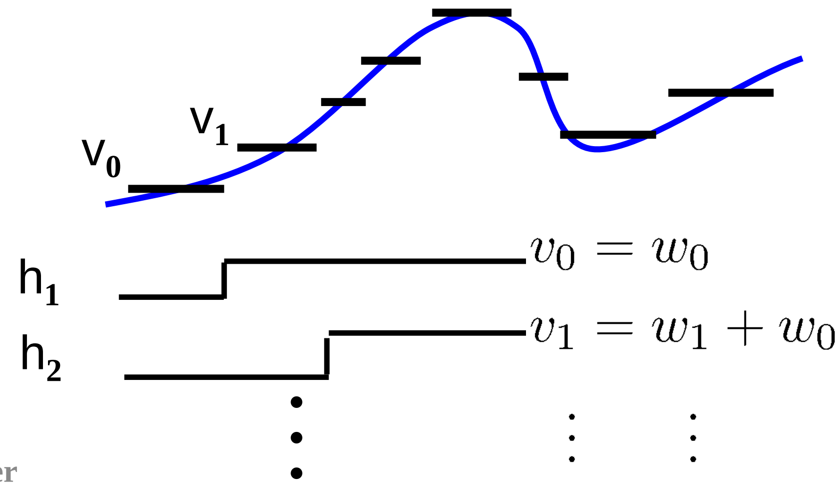



Current research:
“Deep” architectures
(many layers)

# Features of MLPs

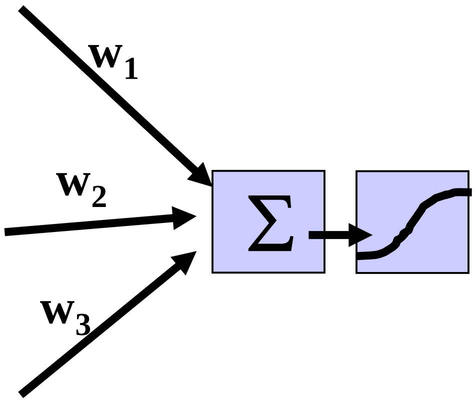
- Simple building blocks
  - Each element is just a perceptron function
- Can build upwards
- Flexible function approximation
  - Approximate arbitrary functions with enough hidden nodes

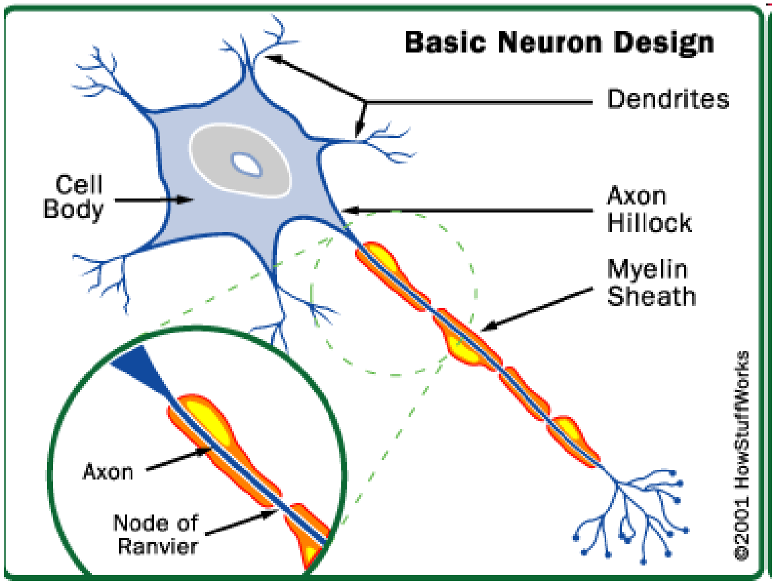


# Neural networks

- Another term for MLPs
- Biological motivation
- Neurons
  - "Simple" cells
  - Dendrites sense charge
  - Cell weighs inputs
  - "Fires" axon



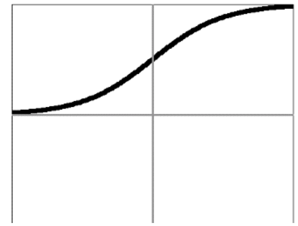




"How stuff works: the brain"

# Activation functions

Logistic

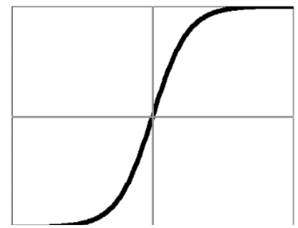
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



$$\frac{\partial \sigma}{\partial z}(z) = \sigma(z)(1 - \sigma(z))$$

Hyperbolic Tangent

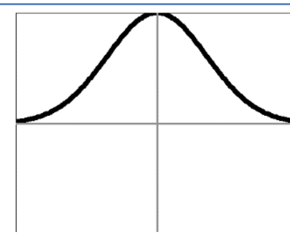
$$\sigma(z) = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}$$



$$\frac{\partial \sigma}{\partial z}(z) = 1 - (\sigma(z))^2$$

Gaussian

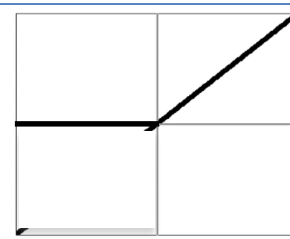
$$\sigma(z) = \exp(-z^2/2)$$



$$\frac{\partial \sigma}{\partial z}(z) = -z\sigma(z)$$

ReLU (rectified linear)

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



$$\frac{\partial \sigma}{\partial z}(z) = \mathbb{1}[z > 0]$$

Linear

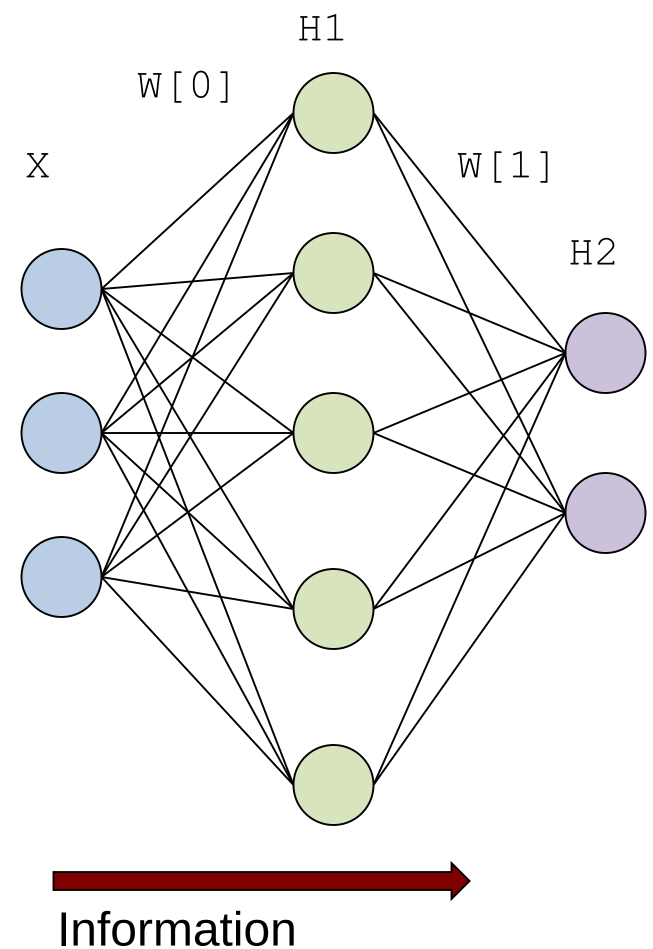
$$\sigma(z) = z$$

and many others…

# Feed-forward networks

- Information flows left-to-right
  - Input observed features
  - Compute hidden nodes (parallel)
  - Compute next layer…

```
R = X.dot(W[0])+B[0]; # linear response
H1= Sig( R );         # activation f'n

S = H1.dot(W[1])+B[1]; # linear response
H2 = Sig( S );         # activation f'n

% ...
```

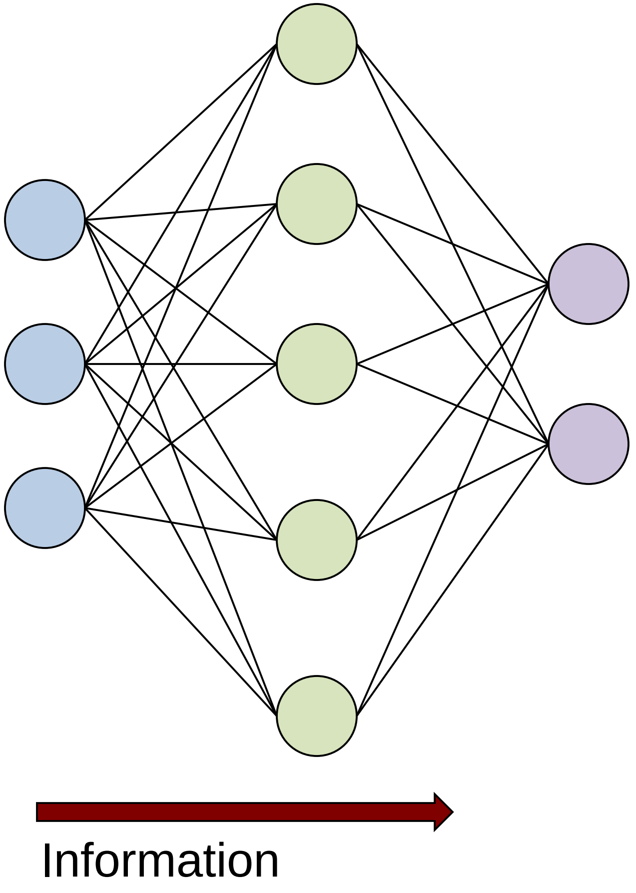



- Alternative: recurrent NNs…

# Feed-forward networks

A note on multiple outputs:

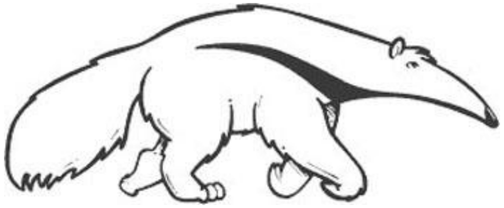
- Regression:
  - Predict multi-dimensional y
  - "Shared" representation = fewer parameters
- Classification
  - Predict binary vector
  - Multi-class classification y = 2 = [0 0 1 0 … ]
  - Multiple, joint binary predictions (image tagging, etc.)
  - Often trained as regression (MSE), with saturating activation



# Machine Learning and Data Mining

## Multi-layer Perceptrons & Neural Networks: Backpropagation

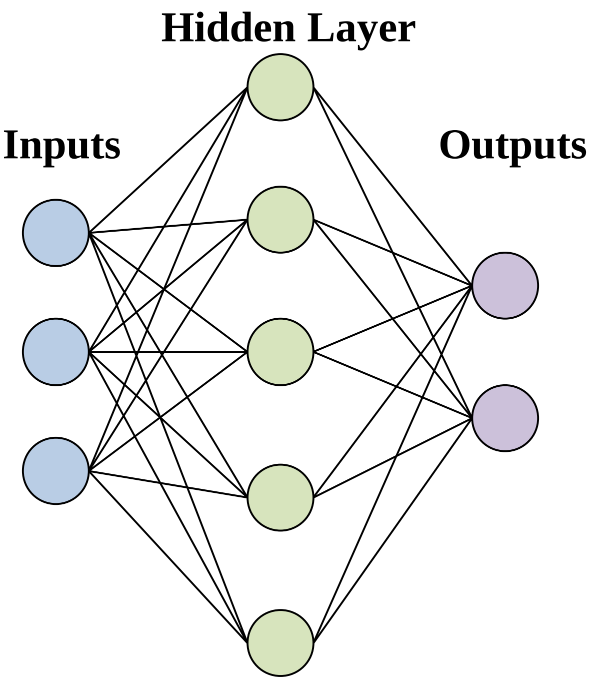
Kalev Kask



# Training MLPs

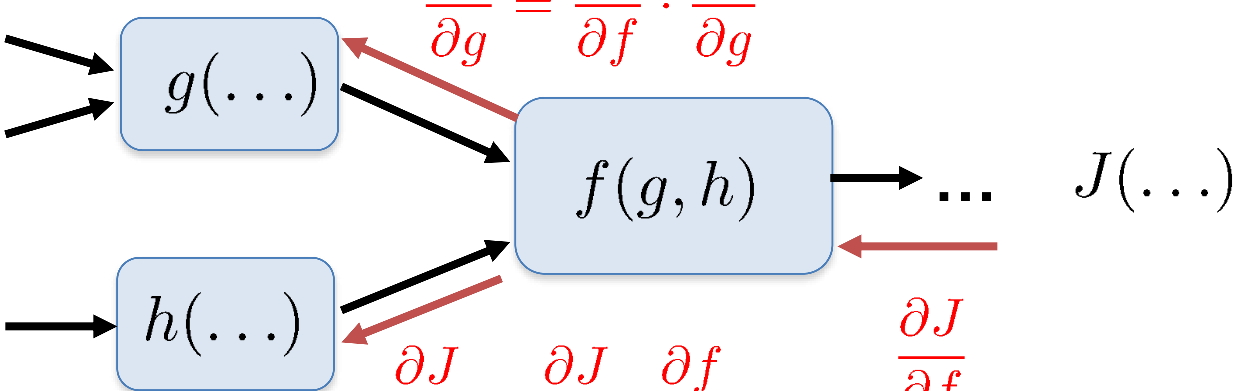
- Observe features "x" with target "y"
- Push "x" through NN = output is "ŷ"
- Error: $(y - \hat{y})^2$ (Can use different loss functions if desired…)
- How should we update the weights to improve?

- Single layer
  - Logistic sigmoid function
  - Smooth, differentiable
- Optimize using:
  - Batch gradient descent
  - Stochastic gradient descent



# Gradient calculations

- Think of NNs as "schematics" made of smaller functions
  - Building blocks: summations & nonlinearities
  - For derivatives, just apply the chain rule, etc!




$$\frac{\partial J}{\partial g} = \frac{\partial J}{\partial f} \cdot \frac{\partial f}{\partial g}$$

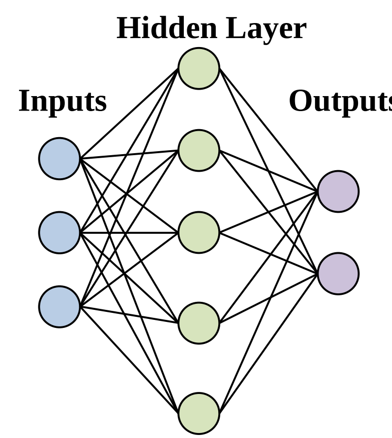
$$\frac{\partial J}{\partial f}$$

$$\frac{\partial J}{\partial h} = \frac{\partial J}{\partial f} \cdot \frac{\partial f}{\partial h}$$

Ex: f(g,h) = $g^2$ h

$$\frac{\partial J}{\partial g} = \frac{\partial J}{\partial f} \cdot 2\, g(\cdot)\, h(\cdot) \qquad \frac{\partial J}{\partial h} = \frac{\partial J}{\partial f} \cdot g^2(\cdot)$$

save & reuse info (g,h) from forward computation!



# Backpropagation

- Just gradient descent…
- Apply the chain rule to the MLP

$$\frac{\partial J}{\partial w_{kj}^2} = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'}) \; (\partial \hat{y}_{k'})$$

$$= -2(y_k - \hat{y}_k) \; \sigma'(s_k) \; h_j$$

(Identical to logistic mse regression with inputs "$h_j$")

**Forward pass**

Loss function

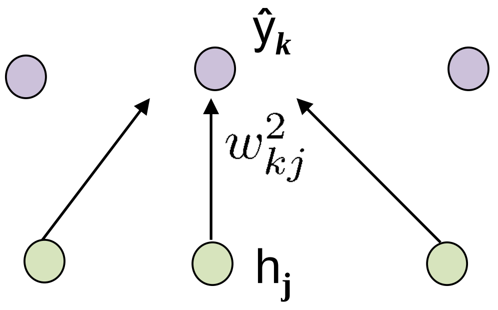
$$J_i(W) = \sum_k \; (y_k^{(i)} - \hat{y}_k^{(i)})^2$$

Output layer

$$\hat{y}_k = \sigma(s_k) = \sigma(\sum_j w_{kj}^2 h_j)$$

Hidden layer

$$h_j = \sigma(t_j) = \sigma(\sum_i w_{ji}^1 x_i)$$



# Backpropagation

- Just gradient descent…
- Apply the chain rule to the MLP

**Forward pass**

Loss function

$$J_i(W) = \sum_k \left(y_k^{(i)} - \hat{y}_k^{(i)}\right)^2$$

Output layer

$$\hat{y}_k = \sigma(s_k) = \sigma\left(\sum_j w^2_{kj} h_j\right)$$

Hidden layer

$$h_j = \sigma(t_j) = \sigma\left(\sum_i w^1_{ji} x_i\right)$$

$$\frac{\partial J}{\partial w^2_{kj}} = -2 \sum_{k'} (y_{k'} - \hat{y}_{k'})\ (\partial \hat{y}_{k'})$$

$$= \boxed{-2(y_k - \hat{y}_k)\ \sigma'(s_k)}\ h_j$$

(Identical to logistic mse regression with inputs "$h_j$")

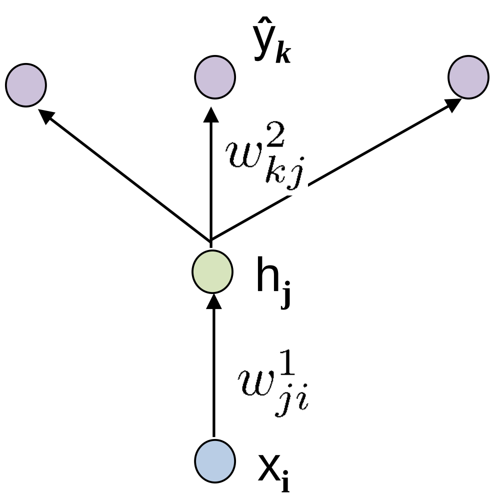
$$\beta^2_k$$

$$\frac{\partial J}{\partial w^1_{ji}} = \sum_k -2(y_k - \hat{y}_k)\ (\partial \hat{y}_k)$$

$$= \sum_k -2(y_k - \hat{y}_k)\ \sigma'(s_k)\ w^2_{kj}\ \partial h_j$$

$$= \sum_k \boxed{-2(y_k - \hat{y}_k)\ \sigma'(s_k)}\ w^2_{kj}\ \sigma'(t_j)\ x_i$$

$$\beta^2_k$$



# Backpropagation

- Just gradient descent…
- Apply the chain rule to the MLP

$$\frac{\partial J}{\partial w^2_{kj}} = \boxed{-2(y_k - \hat{y}_k)\ \sigma'(s_k)}\ h_j$$

$$\frac{\partial J}{\partial w^1_{ji}} = \sum_k \underbrace{\boxed{-2(y_k - \hat{y}_k)\ \sigma'(s_k)}}_{\beta^2_k}\ w^2_{kj}\ \sigma'(t_j)\ x_i$$

**Forward pass**

Loss function

$$J_i(W) = \sum_k \left(y_k^{(i)} - \hat{y}_k^{(i)}\right)^2$$

Output layer

$$\hat{y}_k = \sigma(s_k) = \sigma\left(\sum_j w^2_{kj} h_j\right)$$

Hidden layer

$$h_j = \sigma(t_j) = \sigma\left(\sum_i w^1_{ji} x_i\right)$$

```
% X  : (1xN1)
H  = Sig(X1.dot(W[0]))
% W1 : (N2 x N1+1)
% H  : (1xN2)
Yh = Sig(H1.dot(W[1]))
% W2 : (N3 x N2+1)
% Yh : (1xN3)
```

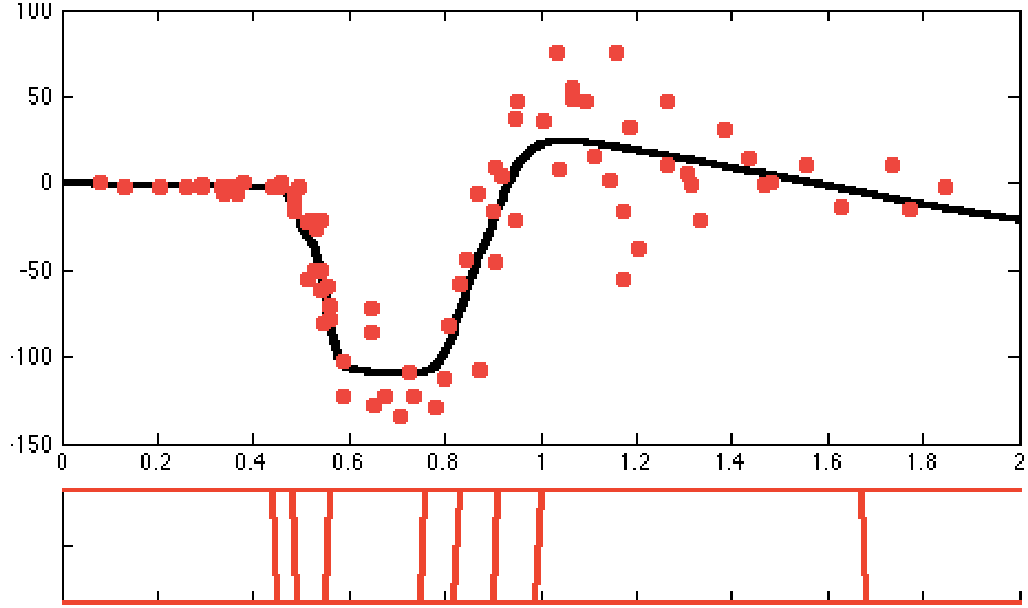
```
B2 = (Y-Yhat) * dSig(S)   #(1xN3)

G2 = B2.T.dot( H )        #(N3x1)*(1xN2)=(N3xN2)

B1 = B2.dot(W[1])*dSig(T)#(1xN3).(N3*N2)*(1xN2)

G1 = B1.T.dot( X )        #(N2 x N1+1)
```

# Example: Regression, MCycle data

- Train NN model, 2 layer
  - 1 input features => 1 input units
  - 10 hidden units
  - 1 target => 1 output units
  - Logistic sigmoid activation for hidden layer, linear for output layer

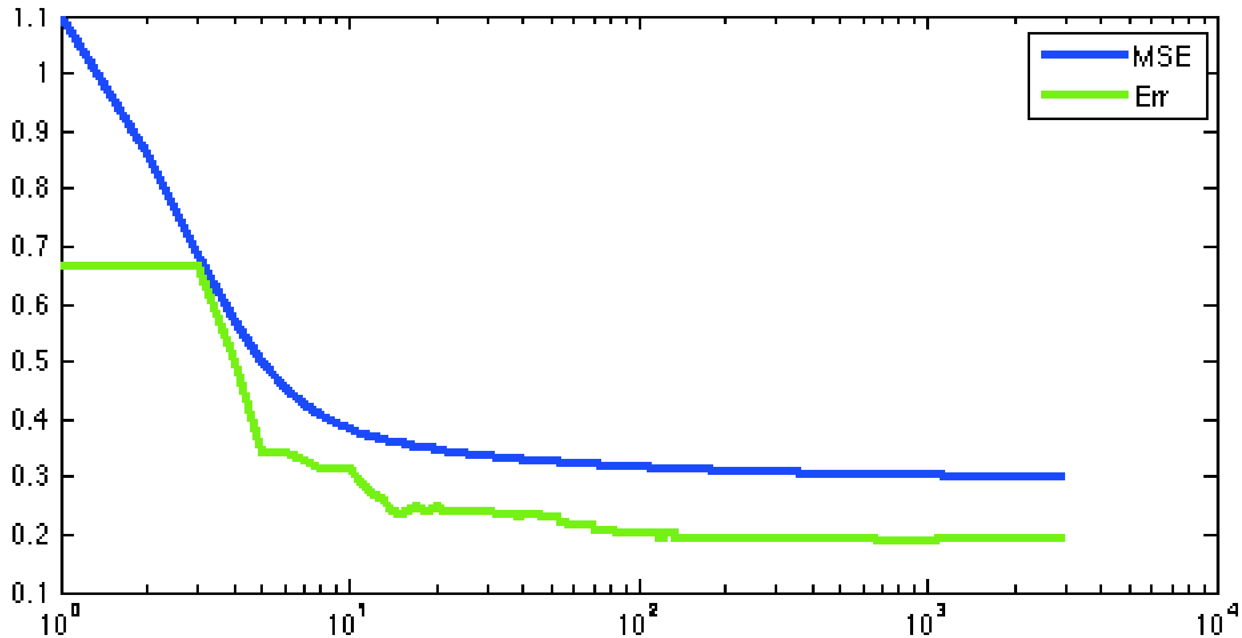
**Data:**
**+**
**learned prediction f'n:**

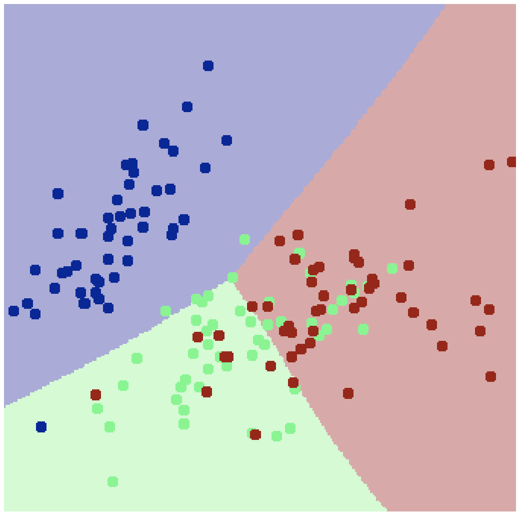
Responses of hidden nodes
(= features of linear regression):
select out useful regions of "x"



# Example: Classification, Iris data

- Train NN model, 2 layer
  - 2 input features => 2 input units
  - 10 hidden units
  - 3 classes => 3 output units (y = [0 0 1], etc.)
  - Logistic sigmoid activation functions
  - Optimize MSE of predictions using stochastic gradient

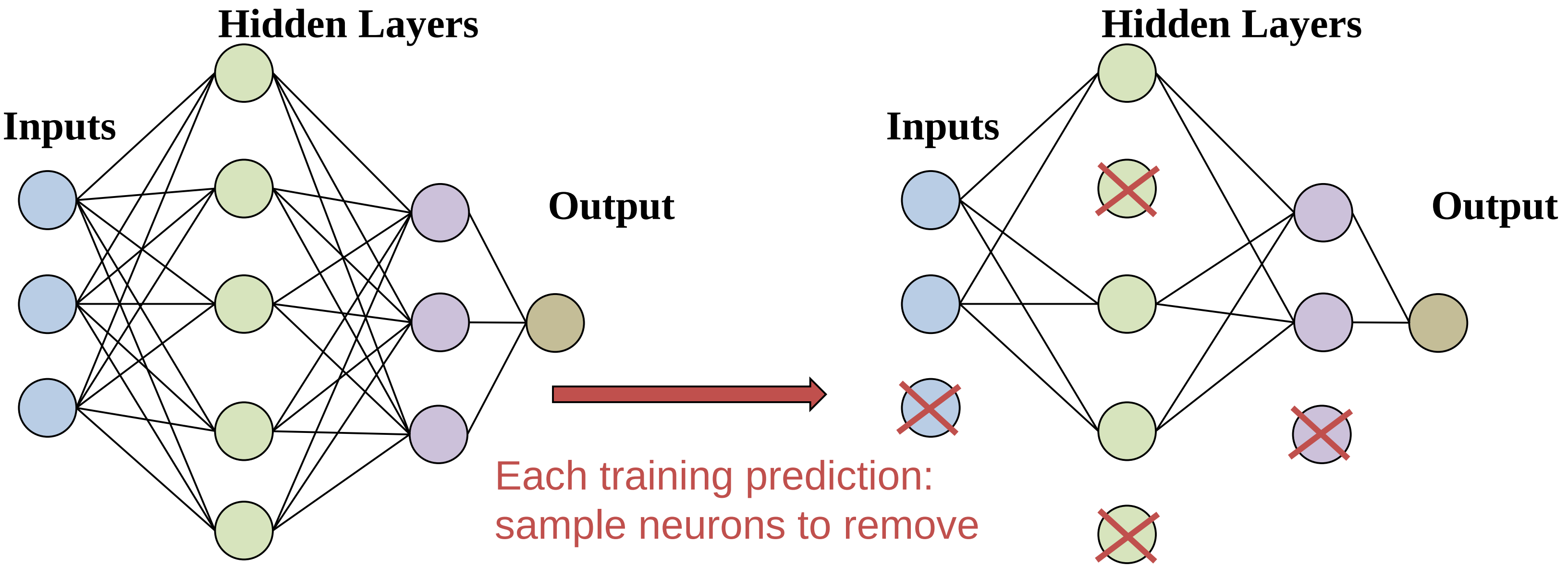


# Dropout

[Srivastava et al 2014]

- Another recent technique
  - Randomly “block” some neurons at each step
  - Trains model to have redundancy (predictions must be robust to blocking)



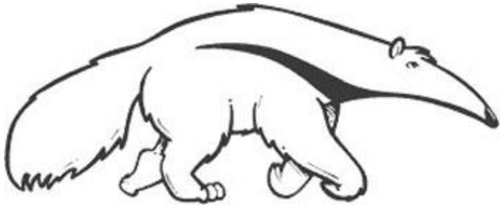

Each training prediction:
sample neurons to remove

```
% ... during training ...
R = X.dot(W[0])+B[0];       # linear response
H1= Sig( R );               # activation f'n
H1 *= np.random.rand(*H1.shape)<p; #drop out!
% ...
```

# Machine Learning and Data Mining

## Neural Networks in Practice

Kalev Kask



# CNNs vs RNNs

- CNN
  - Fixed length input/output
  - Feed forward
  - E.g. image recognition
- RNN
  - Variable length input
  - Feed back
  - Dynamic temporal behavior
  - E.g. speech/text processing

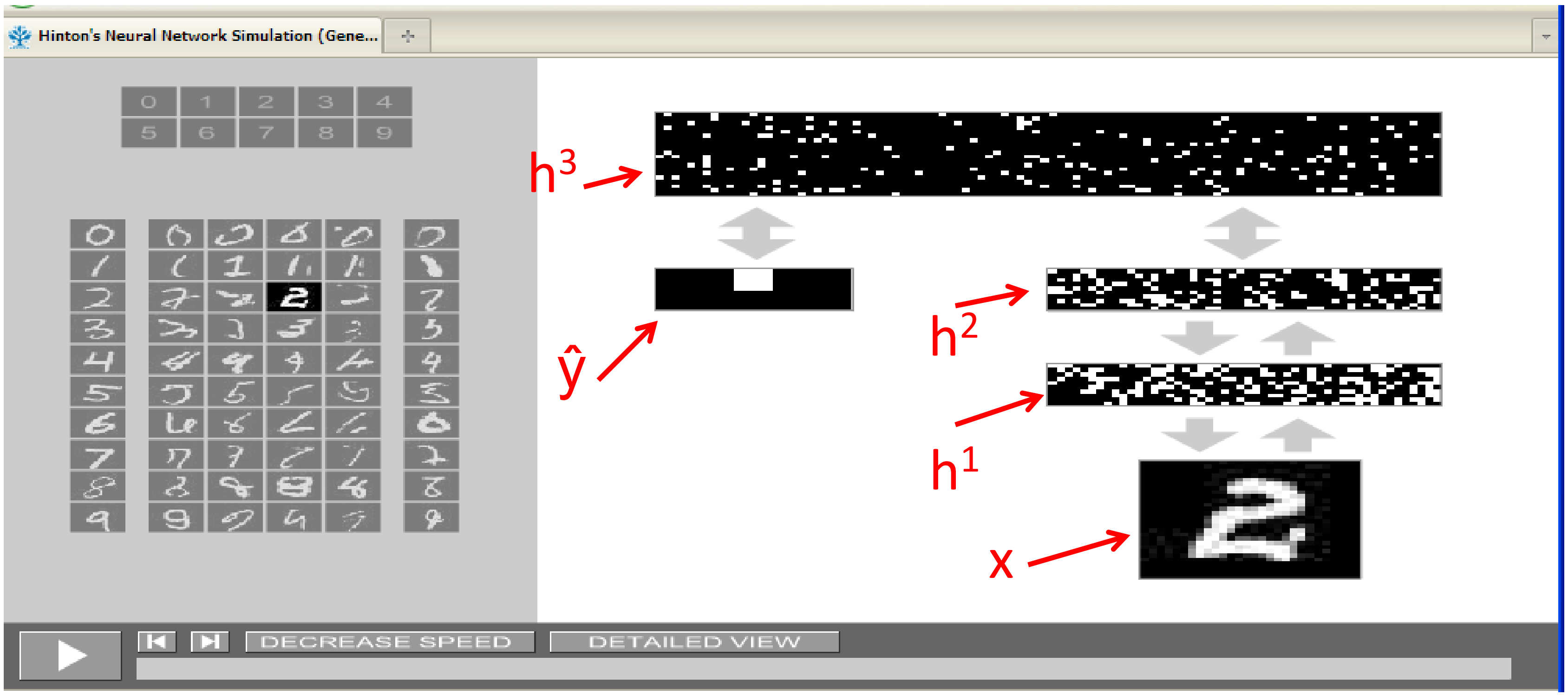
- http://playground.tensorflow.org

# MLPs in practice

[Hinton et al. 2007]

- Example: Deep belief nets
  - Handwriting recognition
  - Online demo
  - 784 pixels ⇔ 500 mid ⇔ 500 high ⇔ 2000 top ⇔ 10 labels

$x$ $h^1$ $h^2$ $h^3$ $\hat{y}$

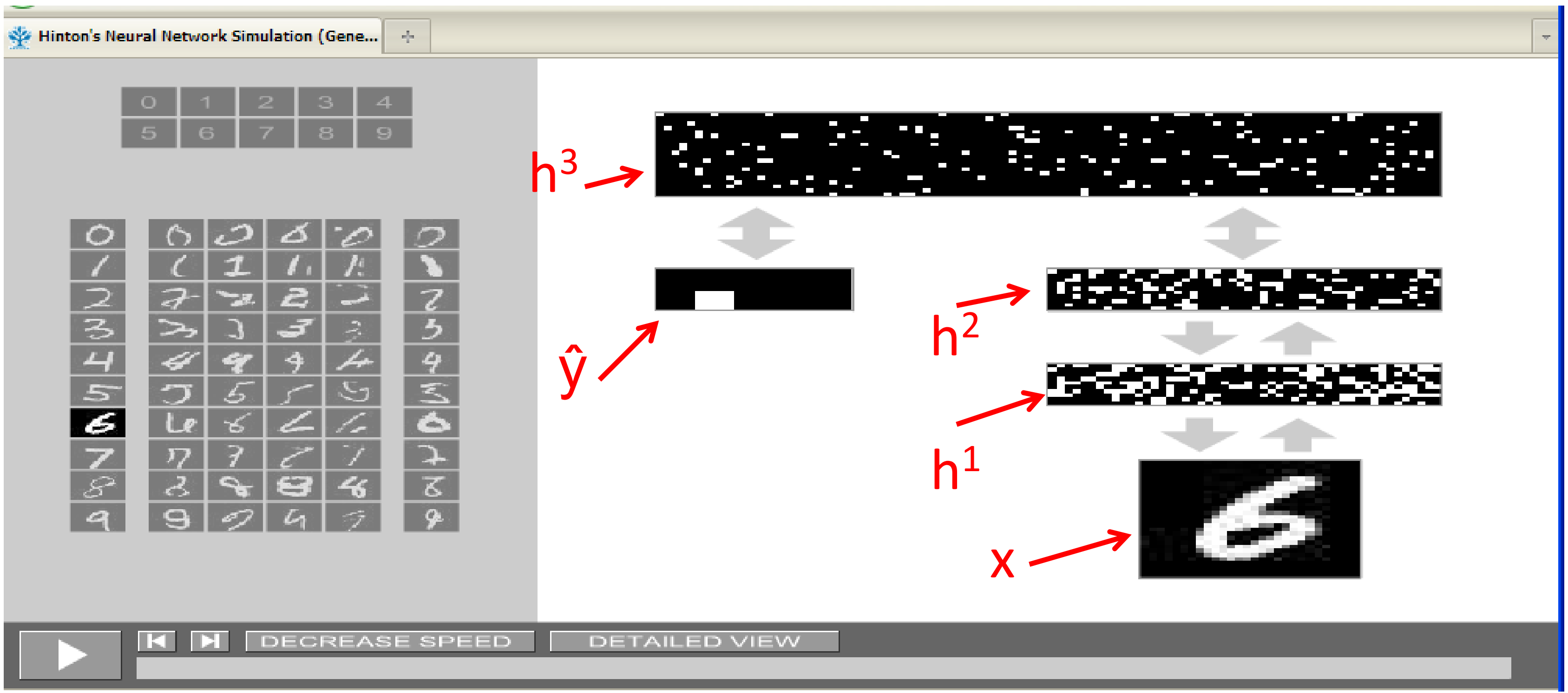


# MLPs in practice

[Hinton et al. 2007]

- Example: Deep belief nets
  - Handwriting recognition
  - Online demo
  - 784 pixels ⇔ 500 mid ⇔ 500 high ⇔ 2000 top ⇔ 10 labels

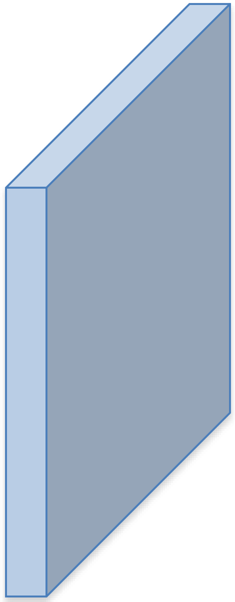
x $\quad h^1 \quad h^2 \quad h^3 \quad \hat{y}$



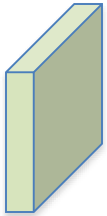
# Convolutional networks

- Organize & share the NN's weights   (vs "dense")
- Group weights into "filters"

Input: 28x28 image



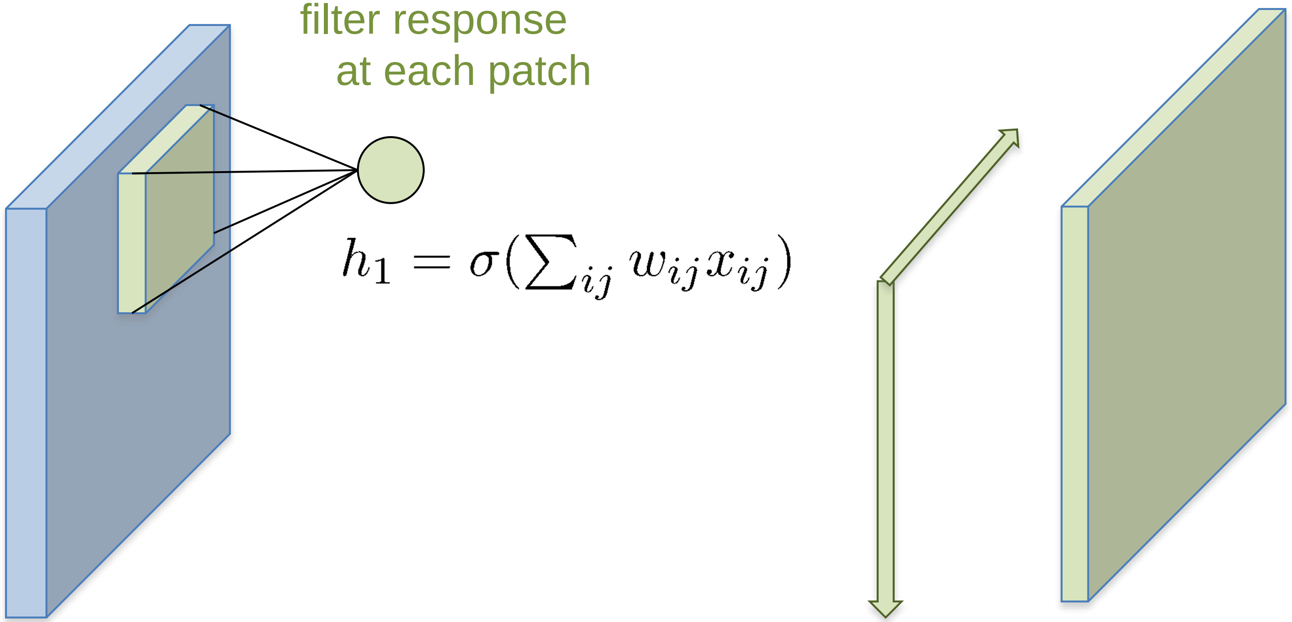
Weights: 5x5



# Convolutional networks

- Organize & share the NN's weights (vs "dense")
- Group weights into "filters" & convolve across input image

Input: 28x28 image　　Weights: 5x5　　24x24 image



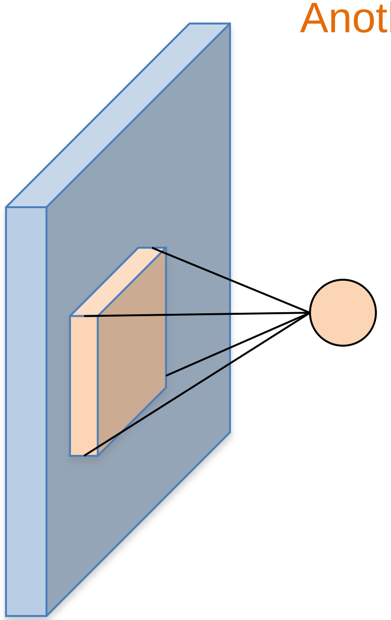

Run over all patches of input
) activation map

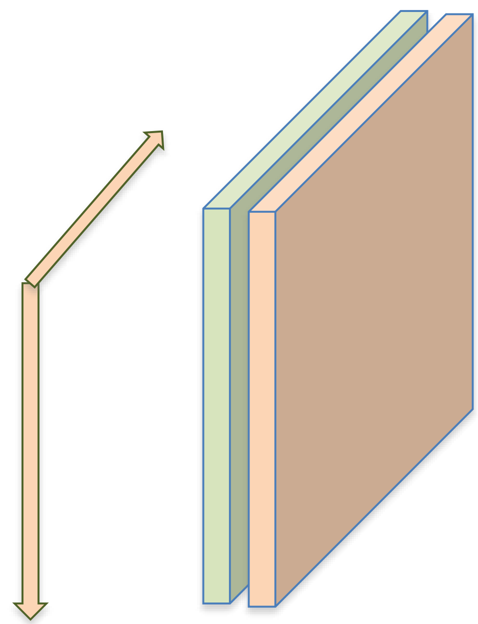
# Convolutional networks

- Organize & share the NN's weights   (vs "dense")
- Group weights into "filters" & convolve across input image

Input: 28x28 image       Weights: 5x5

Another filter

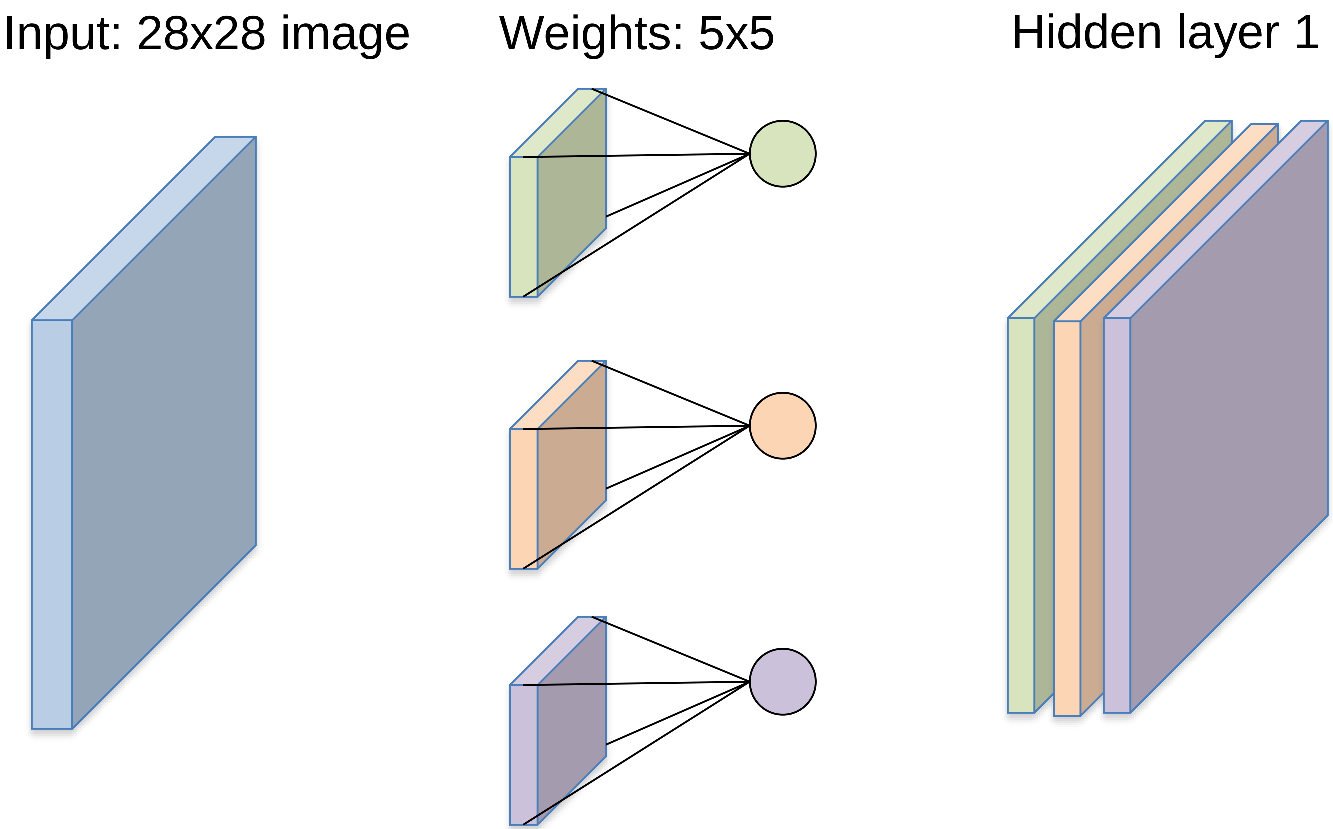




Run over all patches of input
) activation map

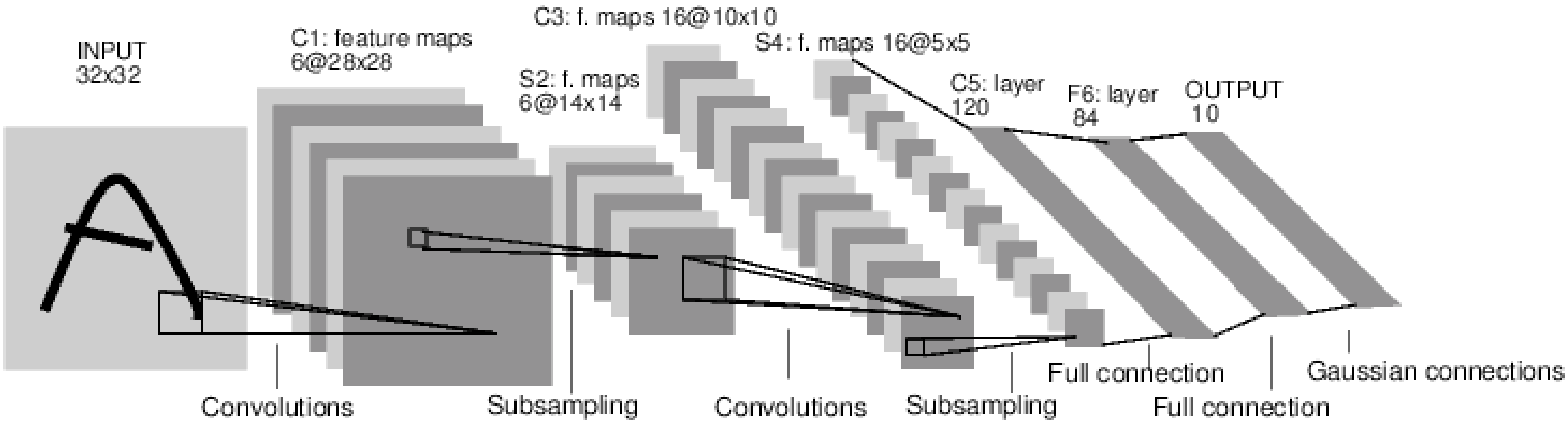
# Convolutional networks

- Organize & share the NN's weights (vs "dense")
- Group weights into "filters" & convolve across input image
- Many hidden nodes, but few parameters!



# Convolutional networks

- Again, can view components as building blocks
- Design overall, deep structure from parts
  - Convolutional layers
  - “Max-pooling” (sub-sampling) layers
  - Densely connected layers



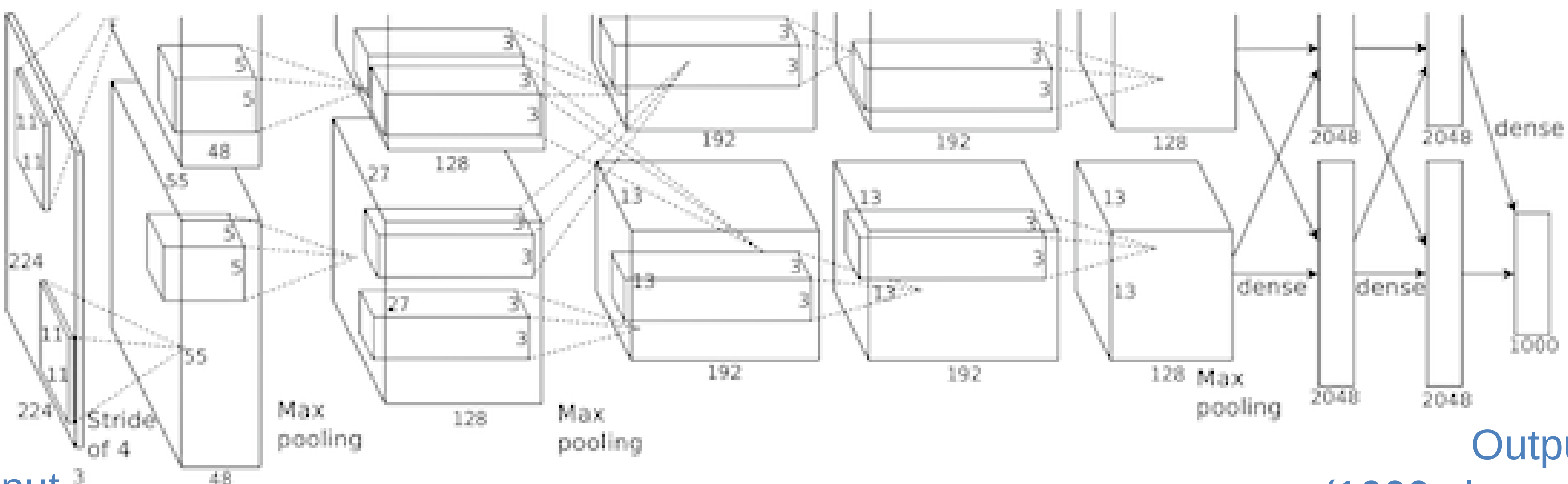

LeNet-5 [LeCun 1980]

# Ex: AlexNet

[Krizhevsky et al. 2012]

- Deep NN model for ImageNet classification
  - 650k units; 60m parameters
  - 1m data; 1 week training (GPUs)

Convolutional Layers (5) Dense Layers (3)

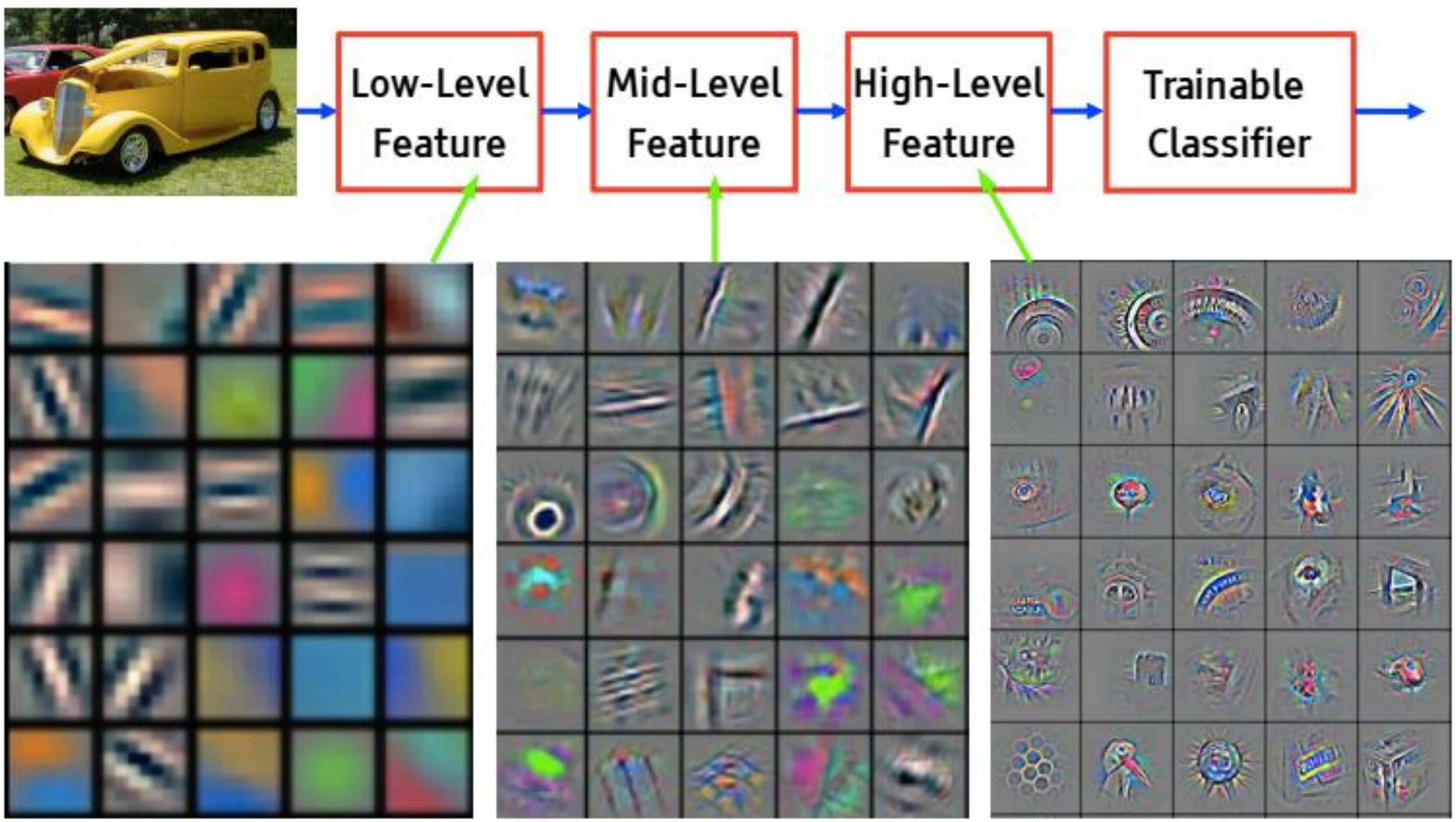



Input 224x224x3

Output (1000 classes)

# Hidden layers as "features"

- Visualizing a convolutional network's filters [Zeiler & Fergus 2013]




Slide image from Yann LeCun:
https://drive.google.com/open?id=0BxKBnD5y2M8NclFWSXNxa0JlZTg

# Neural networks & DBNs

- Want to try them out?
- Matlab “Deep Learning Toolbox”

  https://github.com/rasmusbergpalm/DeepLearnToolbox

  rasmusbergpalm / **DeepLearnToolbox**

  Matlab/Octave toolbox for deep learning. Includes Deep Belief Nets, Stacked Autoencoders, Convolutional Neural Nets, Convolutional Autoencoders and vanilla Neural Nets. Each method has examples to get you started.

- PyLearn2

  https://github.com/lisa-lab/pylearn2

- TensorFlow

# Summary

- Neural networks, multi-layer perceptrons
- Cascade of simple perceptrons
  - Each just a linear classifier
  - Hidden units used to create new features
- Together, general function approximators
  - Enough hidden units (features) = any function
  - Can create nonlinear classifiers
  - Also used for function approximation, regression, …
- Training via backprop
  - Gradient descent; logistic; apply chain rule. Building block view.
- Advanced: deep nets, conv nets, dropout, …