INFSCI 2915: Machine Learning

Neural Networks

Mai Abdelhakim

School of Computing and Information

610 IS Building

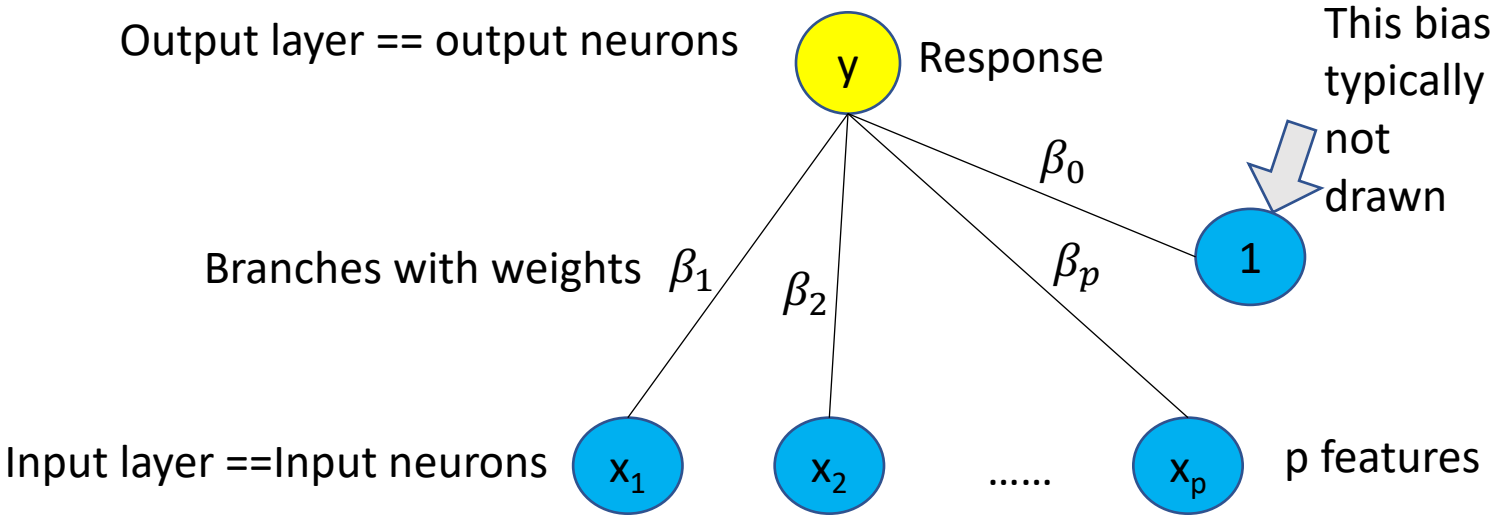
Spring 2018

Neural Networks

- First developed as models for human brain, then used as a powerful tool for statistical learning
- Is a generalization of linear models for **classification** and **regression**
- Has the ability to **learn complex non linear functions**
- Perform multiple stages of processing to predict a response

Review Linear Regression

- Recall linear regression Y= $\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta_pX_p$
- One can view linear regression model graphically as:
  - Input layer: composed of input units, representing features
  - Output layer: composed of the target (response unit)
  - Branches with coefficients representing weights of connection between output and each input units
- These units can be referred to as neurons

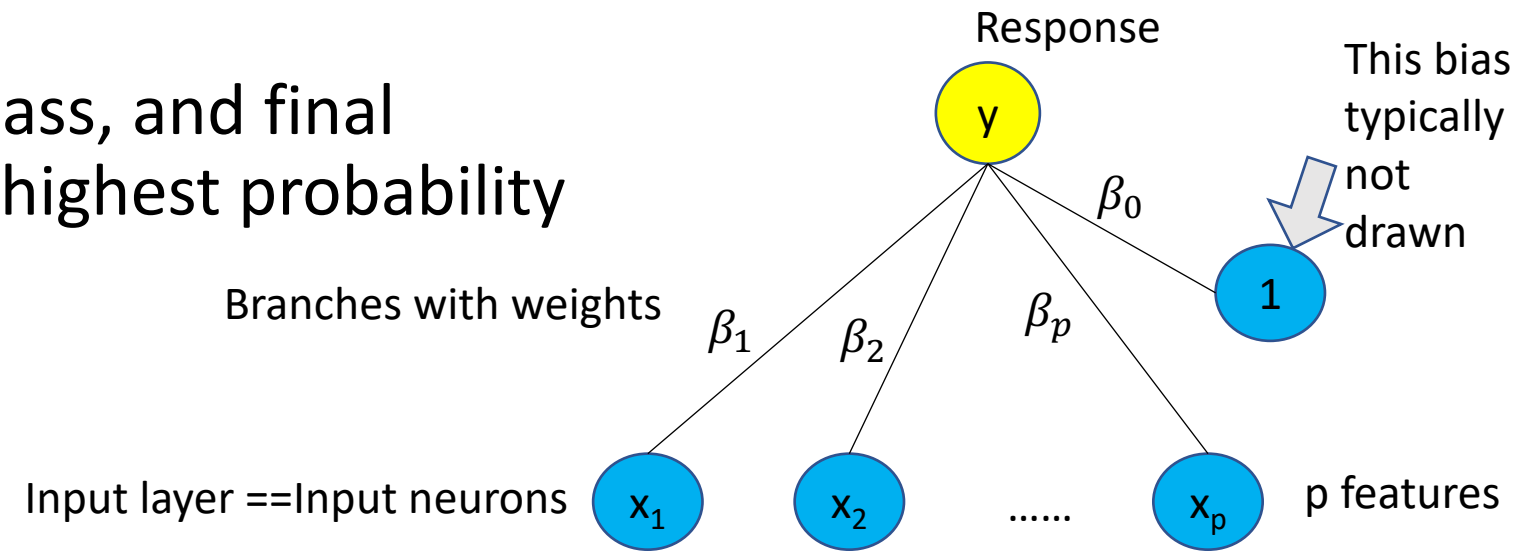


Review Logistic Regression

- For classification the output unit computes a function (sigmoid function)

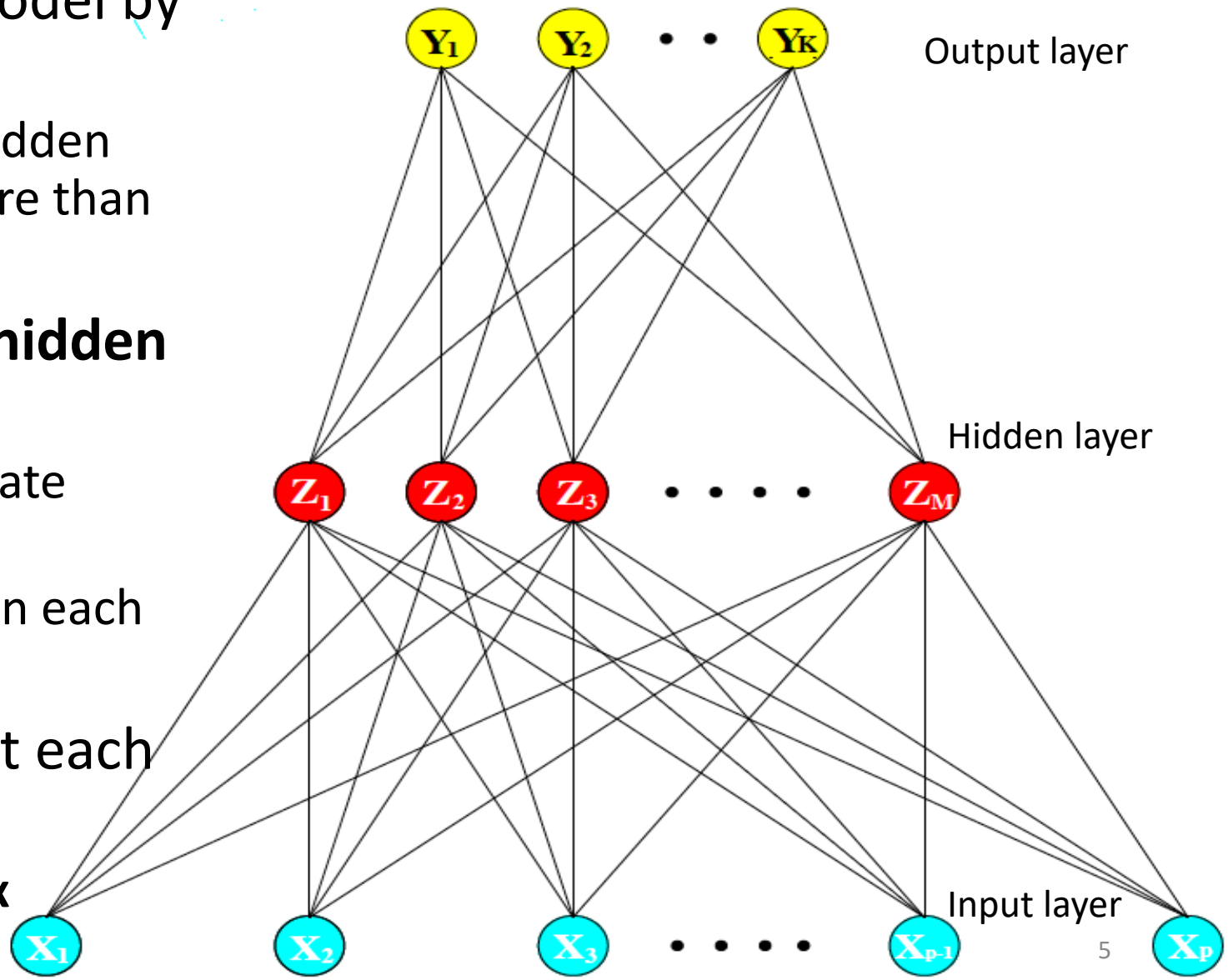
$$\frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

- Output is probability of a class, and final prediction is the class with highest probability

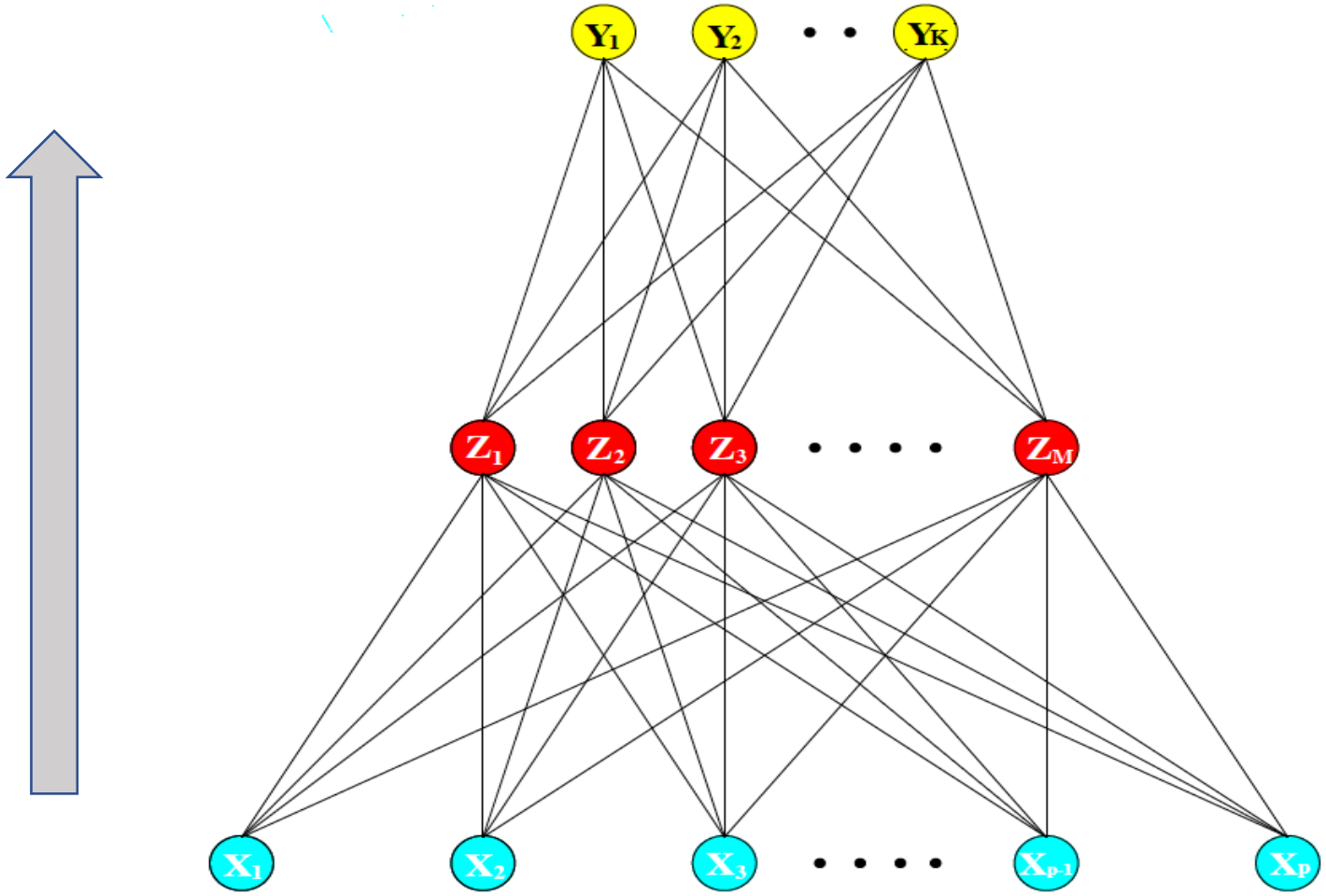


Neural Networks

- Neural networks extends this model by **adding hidden layers**
  - Figure shows a model with one hidden layer. In general, there can be more than one
- Each hidden layer has **multiple hidden units** (hidden neurons)
  - Hidden units represent intermediate processing steps
  - There can be thousands of them in each layer
- **Non-linear function** is applied at each hidden unit
  - This gives ability to **learn complex boundaries**



- **Feedforward network (Multilayer Perceptron)**: The output of each layer feeds into the input of next layer until we get the final output



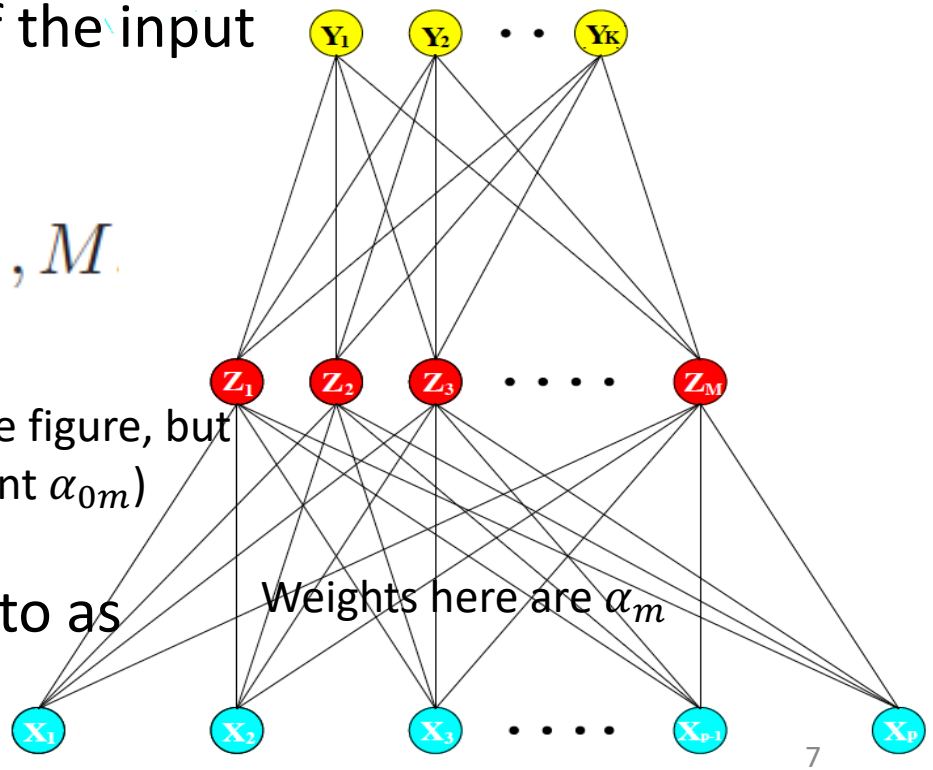
Neural Network with One-Hidden Layer

- Let the number of neurons in output layer be K
  - For regression, K=1 and there will be only Y_1 at the top
- As if the **hidden layer derives new features** Z_m(M features) from the original input X
  - The new features (Z_m) are **non-linear combination** of the input feature vector (X)

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X) \qquad m = 1, \ldots, M$$

Activation function

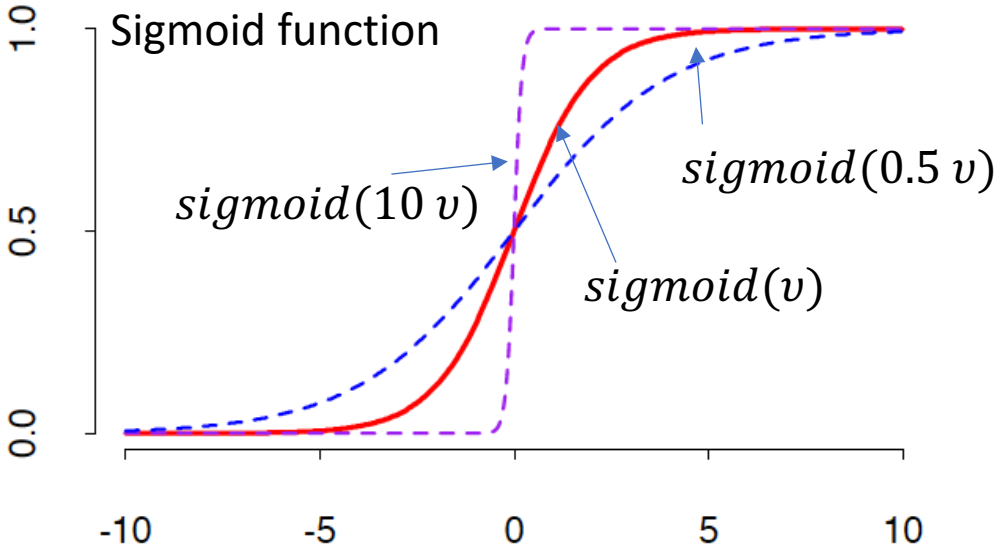
Bias term is not shown in the figure, but it is in calculations (coefficient α_{0m})

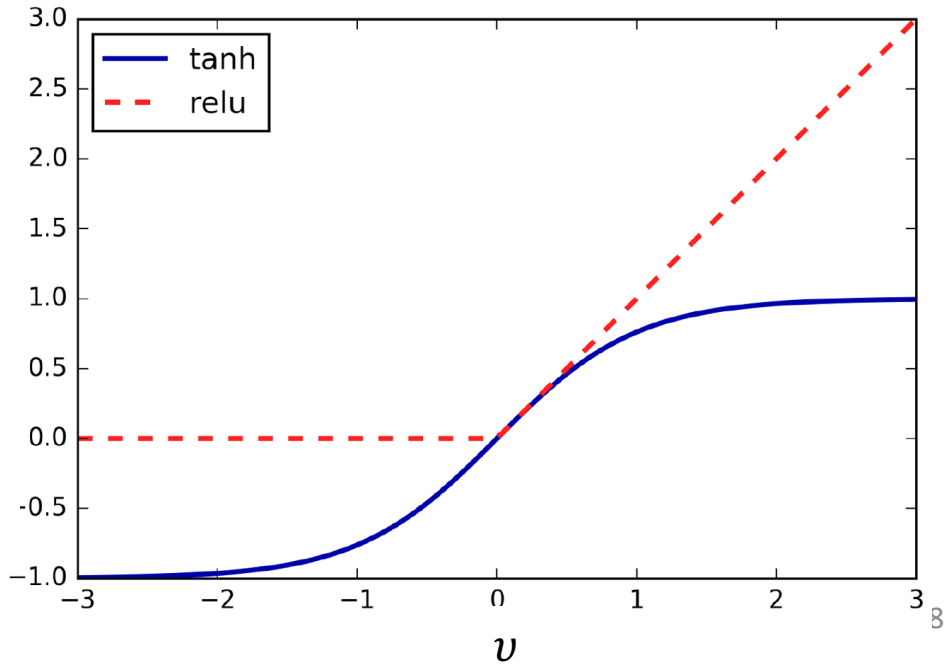


  - Function computed at hidden units, $\sigma(v)$, is referred to as activation function

Activation Functions at Hidden Layer

- Activation function can be a Sigmoid function
  - $\sigma(v)$= $sigmoid(v)$=1/(1+exp(- v))
  - Note sign function can be $sigmoid(10\,v)$
- Other popular activation function for hidden layer are
  - The **hyperbolic tan function**, 'tanh'
    - $\tanh(v)= \frac{2}{1+\exp(-2v)} - 1$
  - The **rectified linear unit function**, 'relu',
    - relu(v)=max(0, v)



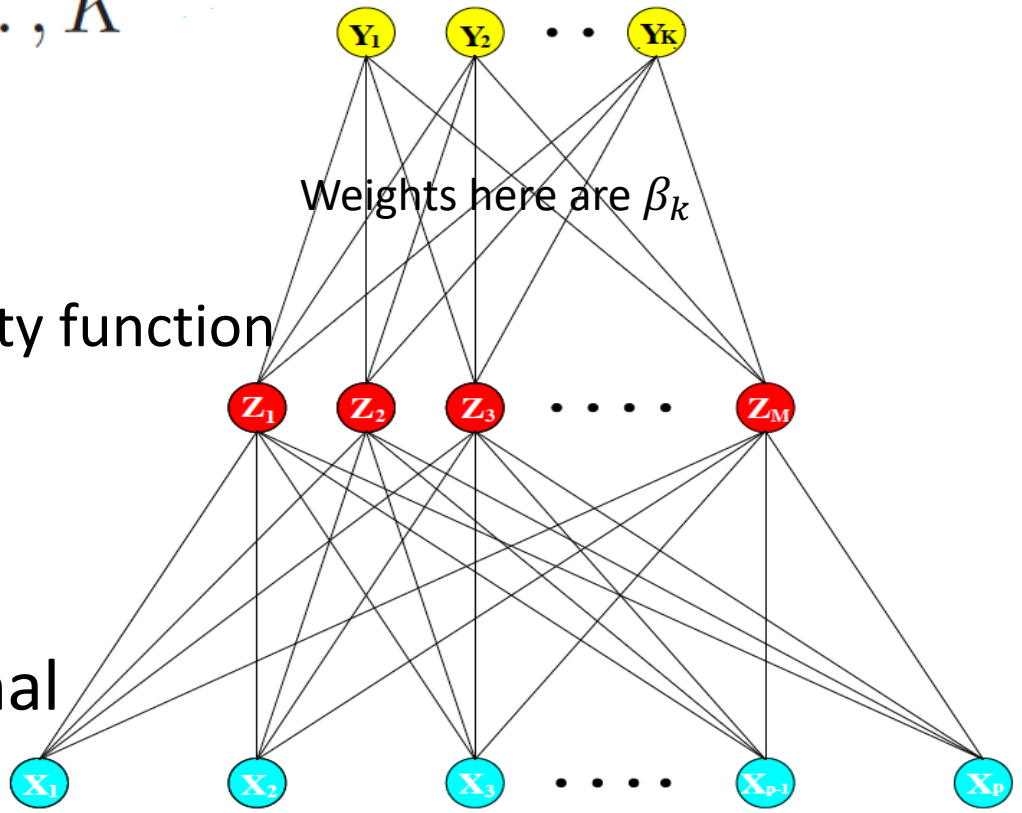


Output Layer

- Output layer computes: linear combination of the new derived features (Z_m)

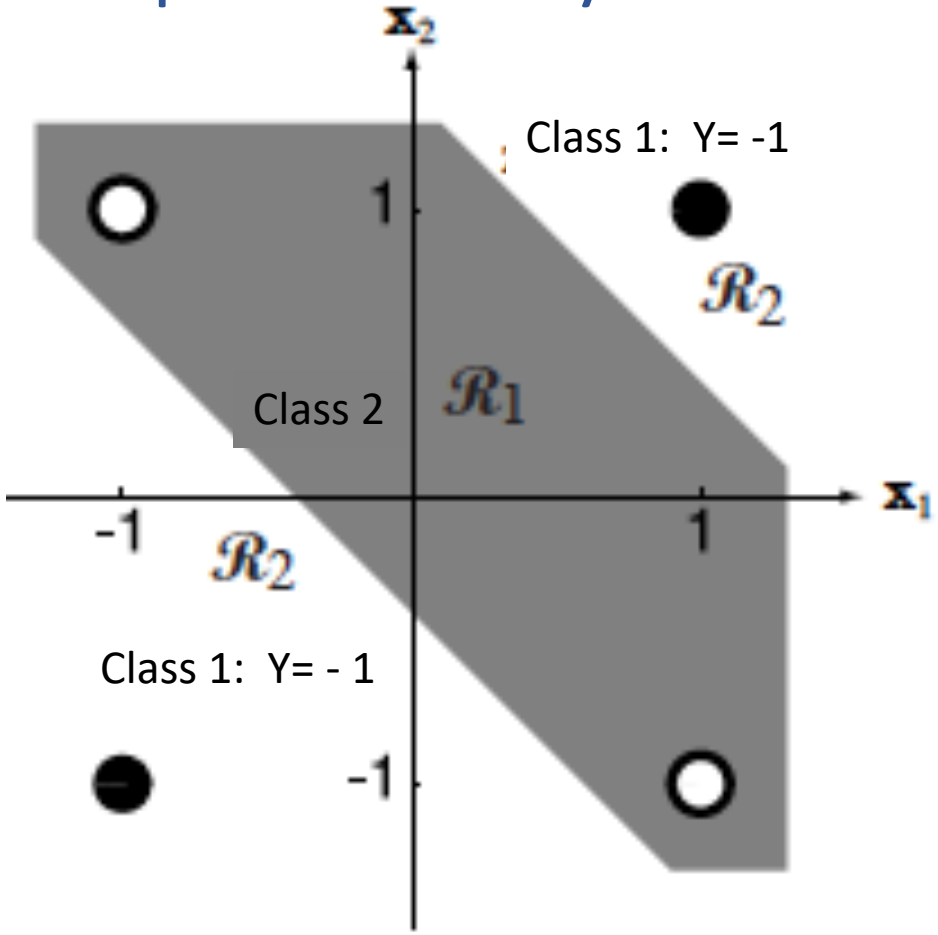
$$T_k = \beta_{0k} + \beta_k^T Z \qquad k = 1, \ldots, K$$

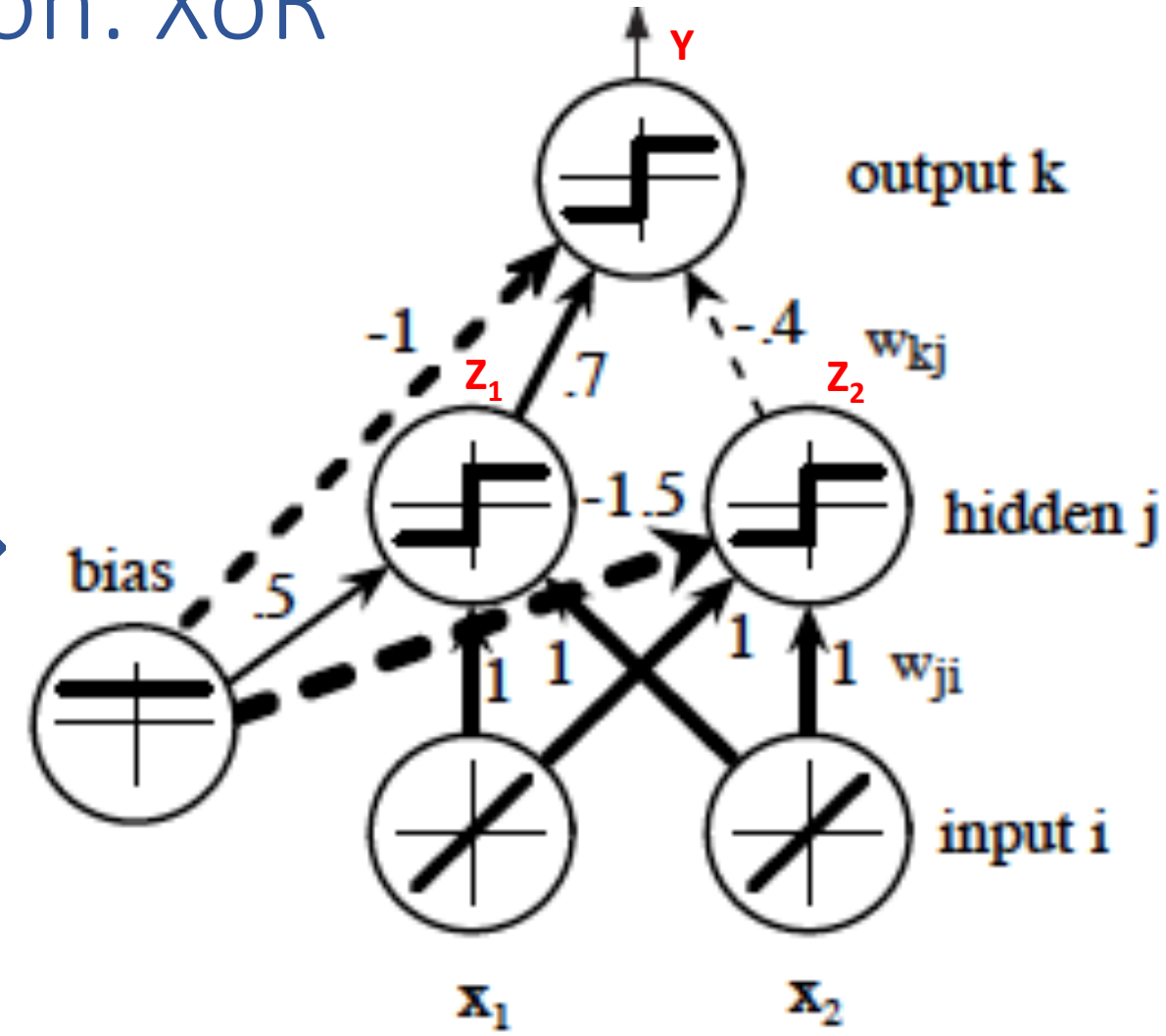
- In **regression** K=1, and $\hat{Y} = f_1(X) = T_1$
  - This means that output activation function is identity function
  - The output is linear of the derived hidden features Z



- For classification: final transformation to get final output
  - Output layer computes function $g_k(T)$ for each class K

Example Binary Classification: XoR





Two classes with nonlinear decision boundary:
Class 1 Y=-1, Class 2 Y=+1

- First hidden unit computes: Z_1=sign(0.5+x_1+x_2)
  - Equals 1 if 0.5+x_1+x_2 is +ve, and -1 otherwise
- Second hidden unit computes: Z_2=sign(-1..5+x_1+x_2)
- Output layer computes: Y=sign(-1+0.7 Z_1+-0.4 Z_2)

Reference et al: Duda, Pattern Classification, Chapter 6.

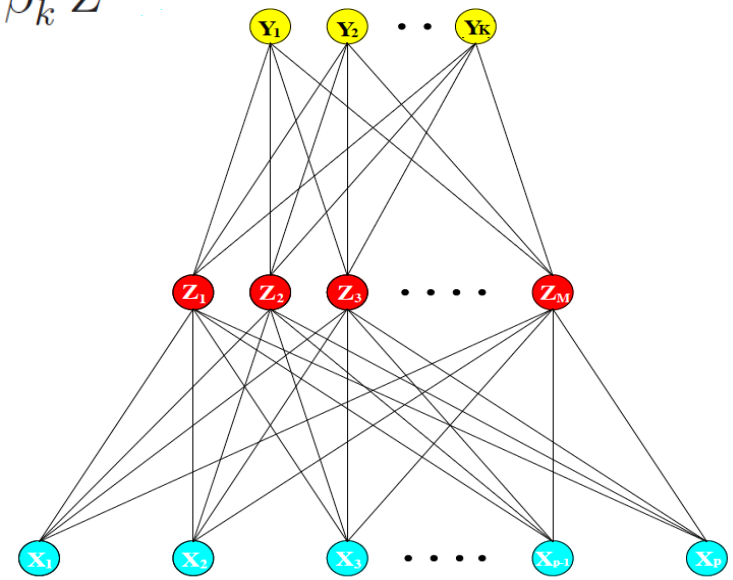
Output Layer for Classification with K classes

- **For classification, the output layer has a neuron for each class (K classes)**
  - The output function of each neuron is the probability of the corresponding class
- Function applied at the output layer is called known as the ***softmax*** function and is given by

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}$$

Recall: $T_k = \beta_{0k} + \beta_k^T Z$

  - P(Y=Class k|x) = $g_k(T)$
  - Sum of $g_k(T)$ over all K classes will give 1
- The final prediction is the class with **highest probability**
  - Y_k=1 when $g_k(T)$ is large, and zero otherwise



Ex: Iris data, we have 3 classes (3 species), output layer should have 3 units

Complexity with One Hidden Layer

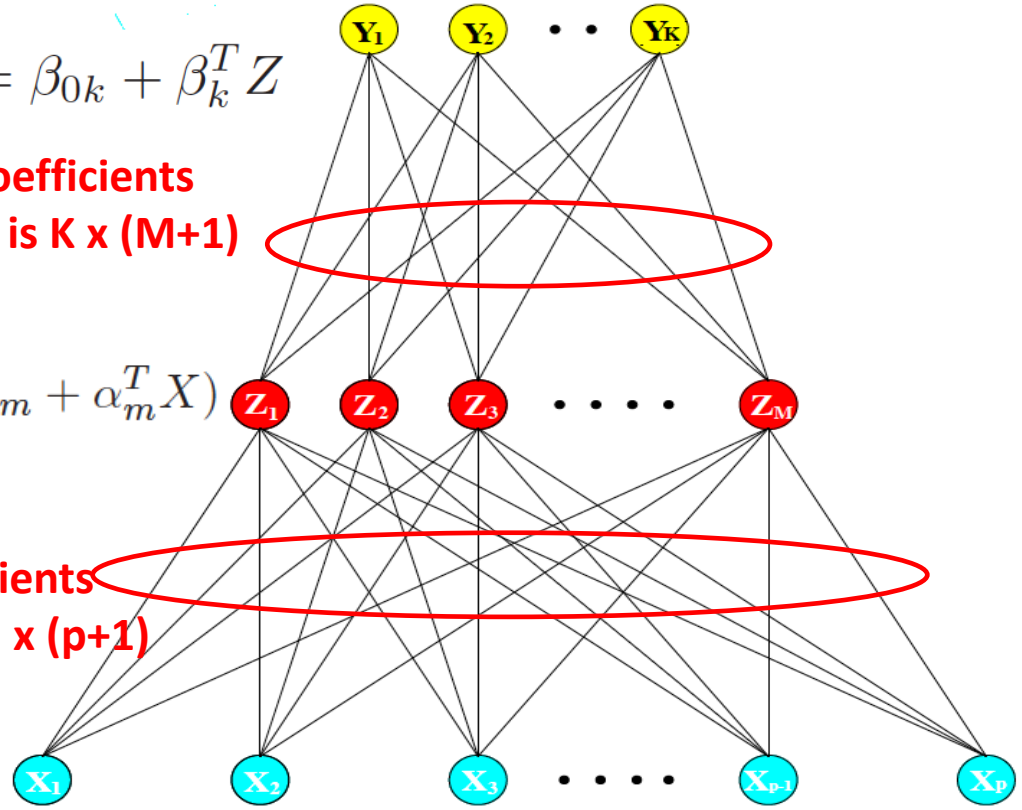
- When input features are **p**, number of hidden neurons in the hidden layer is **M**
- Number of coefficients from input layer to hidden layer is **M x (p+1)**
  - p coefficients + bias term need to be estimated at each neuron
- Similar, number of coefficients from hidden layer to output is K x (M+1)

$$T_k = \beta_{0k} + \beta_k^T Z$$

Number of coefficients (weights β_k) is K x (M+1)

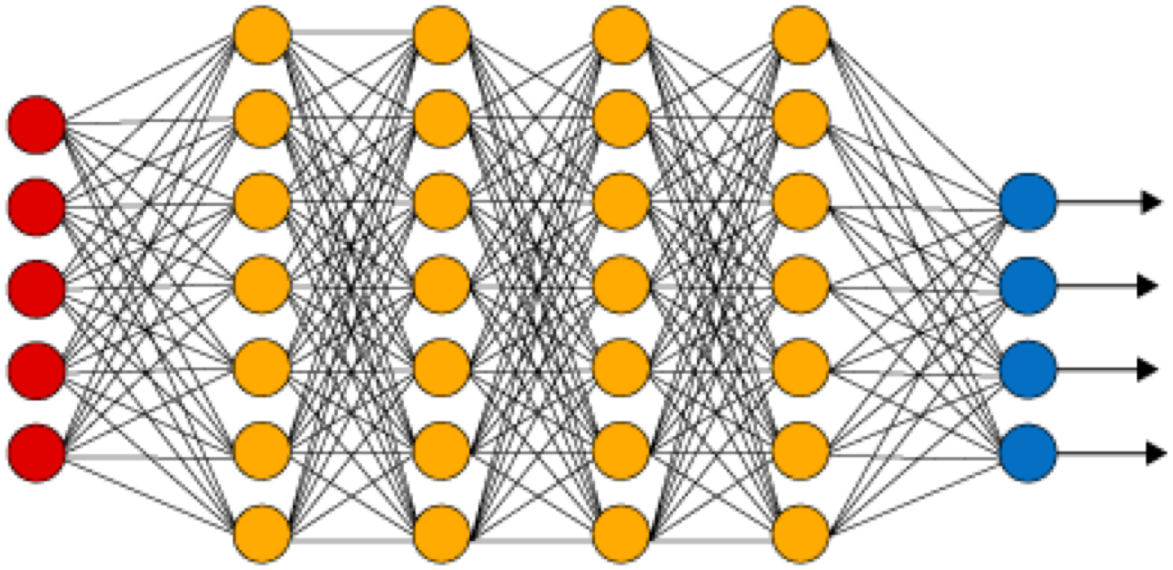
$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X)$$

Number of coefficients (weights α_m) is M x (p+1)



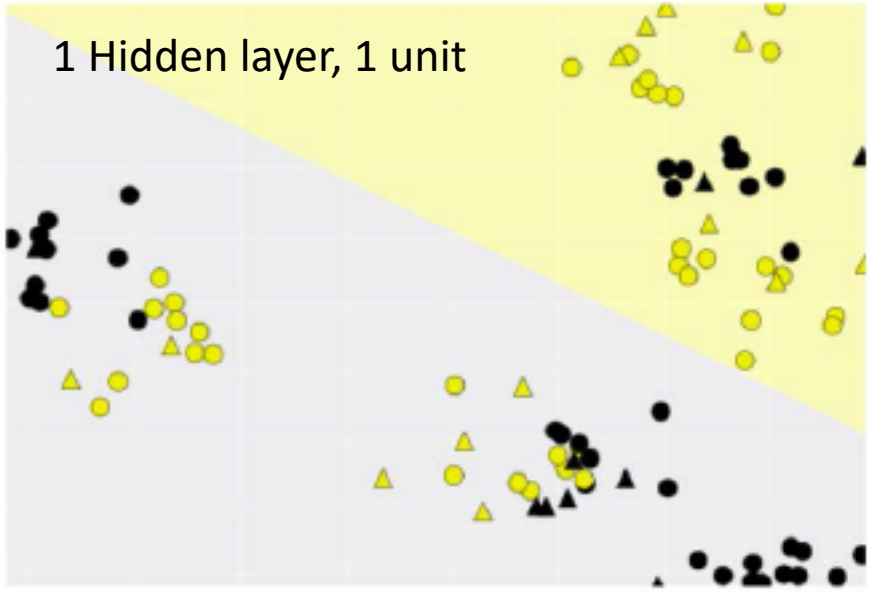
More than One Hidden Layer

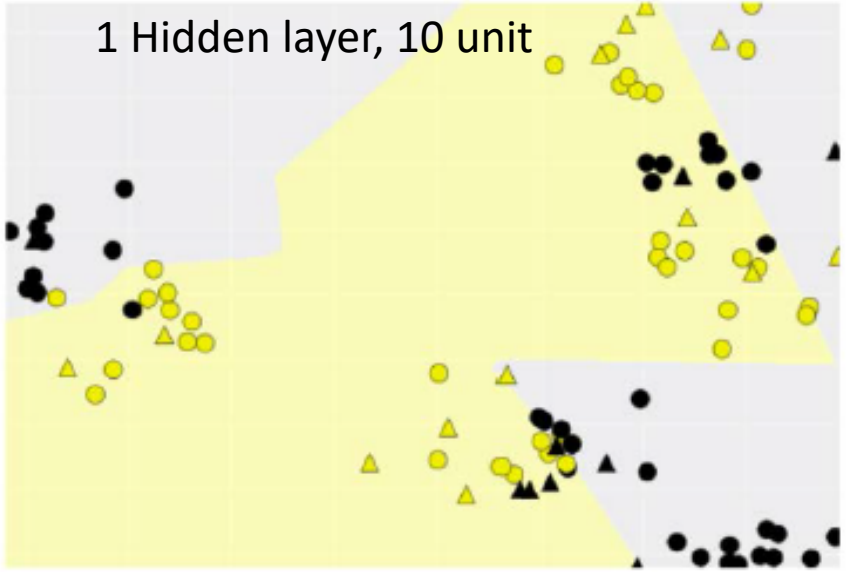
- In general we can have multiple hidden layers
- From layer with N_j neurons to next layer with N_{j+1} neurons we need to estimate $N_{j+1}x(N_j+1)$
- We can tune the network to find the best:
  - Number of hidden layer
  - Number of units per layer

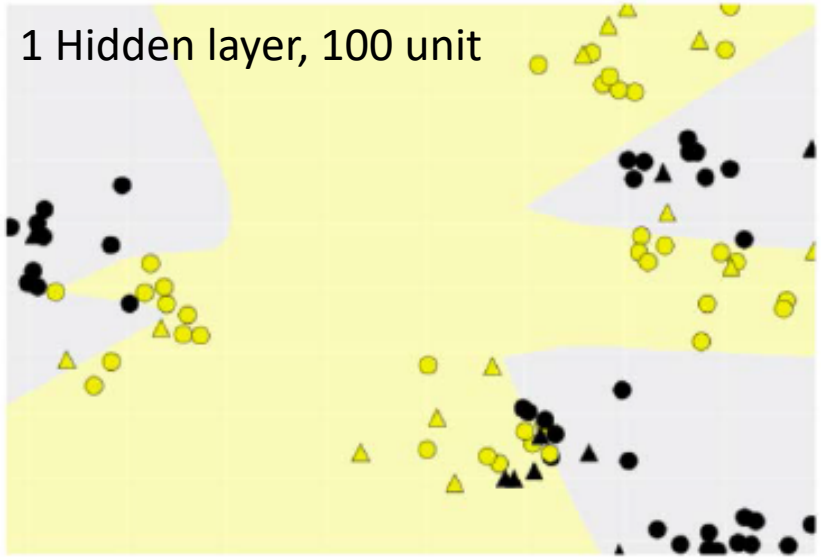


Example of Decision Boundaries – Neural Network Classifier with One Hidden Layer

2 classes (yellow and black), train data (circle) and test (triangle)
tanh activation function is used at hidden layers







Neural networks can learn complex decision boundaries

Ref: Kevyn Collins-Thompson

Fitting the Network

- Fitting the network: estimate all coefficients
- Let complete set of coefficient be denoted as θ (includes β_k , β_{0k} , α_m, α_{0m} …)
- For **regression**, objective is to **minimize the squared error**

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

- For **classification**, we can use **maximum likelihood** (similar to logistic regression)
- Parameters can be estimated using **gradient descent**
  - Approach for finding optimal weights is called **back propagation**
- Other optimization techniques

Issues

- Overfitting: too many weights **need large training** sample to estimate.
  - Otherwise, model will overfit
    - Use Regularization, such as Ridge to shrink the coefficients
    - We can use combining methods

- Sensitivity to feature scale

- The objective function is **nonconvex**, and final weights could be at local minima
  - Final model depends on the choice of initial weights
  - One may try different initialization values and choose solution with lowest errors

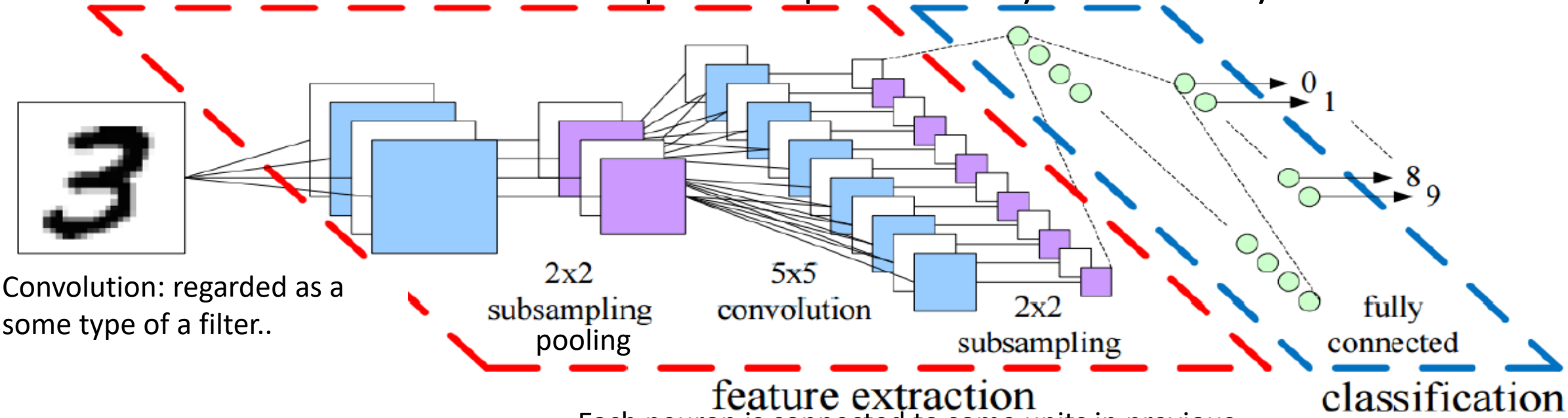
Features: Advantages and Disadvantages

- Advantages: Neural networks can **provide higher accuracy** compared to other approaches as they can **learn complex non-linear models**
- Limitations:
  - More complex and requires **more computational power**
  - Needs **huge training data**
  - They often take a **long time to train**
  - Require careful preprocessing of data
    - Feature scaling is important
  - **Works best when features have similar meaning**
    - For example, all features are pixel values
    - For very different types of features, other models may work better

Deep Learning Concept

Figure: Peemen et al., “Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform”, International Conference on Advanced Concepts for Intelligent Vision Systems, 2011

- Based on neural networks with two main phases:
  - Feature extraction (feature learning) phase with several layers
  - Output prediction phase
- Example: image recognition
  - automatic feature extraction step made up of hierarchy of feature layers.



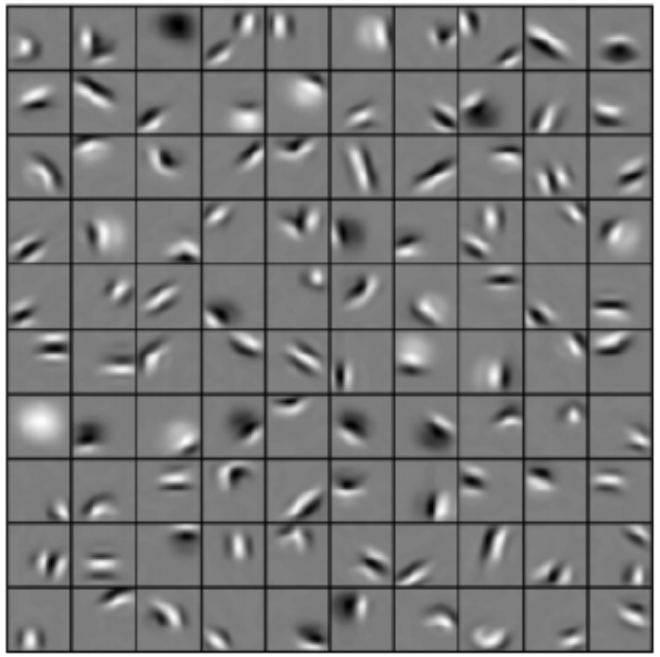
Convolution: regarded as a some type of a filter..

Each neuron is connected to some units in previous layer (not fully connected)

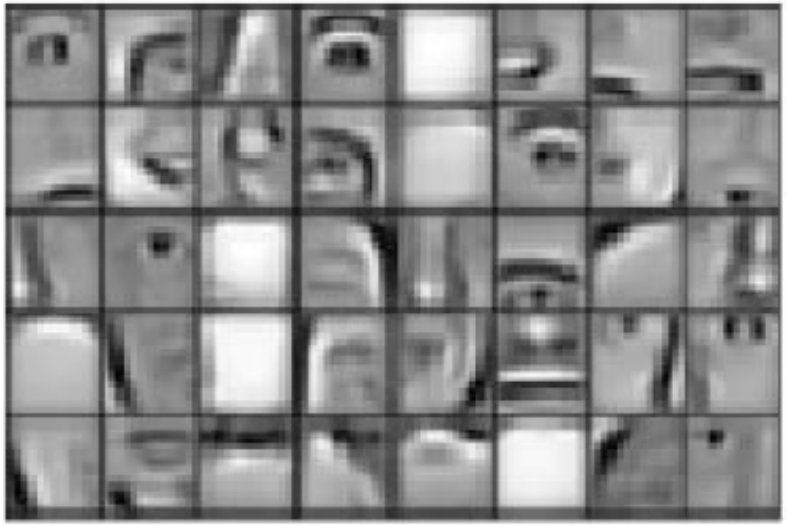
Example of Features in Three Hidden Layers for Face Recognition

Ref: Honglak Lee et al. “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks”. Communications of the ACM, 2011

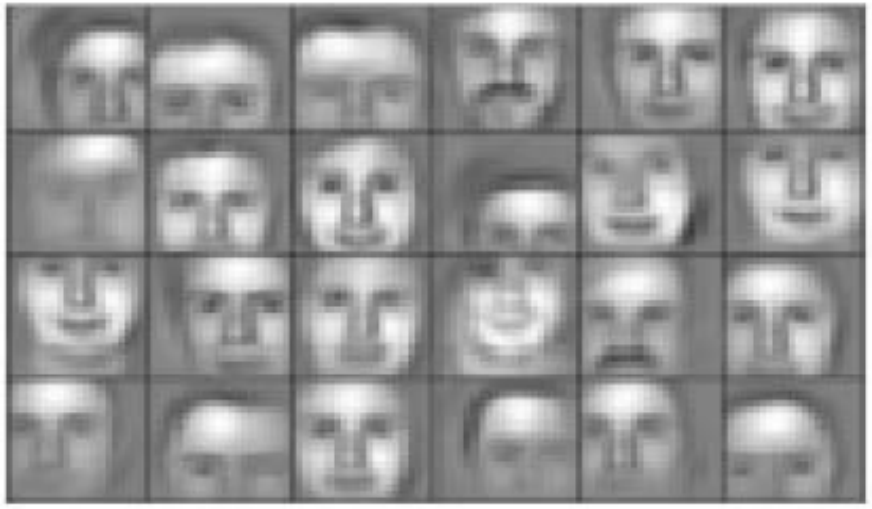
Patterns captured by neurons in first layer



Patterns captured by neurons in second layer



Patterns captured by neurons in third layer



- In feature extraction phase, each neuron focuses on certain part of an image and detects a particular pattern
- First level of feature extraction obtains some low-level features (like edges), then layers that follow capture higher level features (like facial expression)
- Each layer uses a combination of features of previous layers

Deep Learning

- Used in commercial products (e.g. Siri)
- Typically **tailored to specific application**
- **Does automatic feature extraction**
  - **R**educes the need for guessing which features would work well
- **Complex** models, **not easily interpretable**
- Requires **huge computing power** and training data
- Several popular deep leaning models, including:
  - CNN: Convolutional Neural Network (application example: video processing and image recognition)
  - RNN: Recurring Neural Network (for sequential data, application examples: text and speech recognition)
    - Output of unit can be feedback to input
- Reading: https://www.nature.com/articles/nature14539

Python

- Neural networks implementation in Sklearn is made through Multilayer perceptron functions (feedforward neural network): **MLPClassifier**, **MLPRegressor** (check links for details)
- Example:

```
from sklearn.neural_network import MLPClassifier
MLPmodel=MLPClassifier(solver='lbfgs', activation='relu', random_state=0
hidden_layer_sizes=[100,10], alpha=0.5)
```

Weight initialization

Activation function e.g.: 'logistic', 'relu', 'tanh'
logistic (sigmoid)

Alpha for L2 regularization

Hidden_layer_sizes is array, with ith element being the number of neurons in the ith hidden layer

Solver for the optimization problem:
'sgd' (stochastic gradient descent): if selected you may need to increase **max_iter** parameter to converge)

'lbfgs' is another numerical optimization technique (based on quasi-Newton method) which works well for small data sets

Python

- There are other more powerful python-based libraries for deep learning with neural networks (beyond Sklearn), such as:
  - **TensorFlow**
    - https://www.tensorflow.org/
  - **Keras**:
    - https://keras.io/
  - **Lasagne**
    - https://lasagne.readthedocs.io/en/latest/