Summary and Review

EE-UY 4563/EL-GY 9123: INTRODUCTION TO MACHINE LEARNING

PROF. YAO WANG

Topics covered

- Supervised learning:
  - Regression
  - Classification
  - Various methods
- Unsupervised learning
  - Principle component analysis
  - Clustering

General concepts in supervised learning

- Training vs. testing
- Loss function for training
  - Regression: RSS or MSE for regression
  - Classification: Log Likelihood or Log posterior probability, cross entropy
  - Regularization: constrain the model parameters based on some prior knowledge
- Performance metric (for test samples)
  - Regression: RSS
  - Classification: Accuracy, confusion matrix, sensitivity, specificity, precision, recall, ROC curve, AUC
- Cross validation to estimate the test performance
- Cross validation for model selection or hyper parameter optimization
- K>2 folds are needed when we have limited data
- Bagging (model averaging)

Regression methods

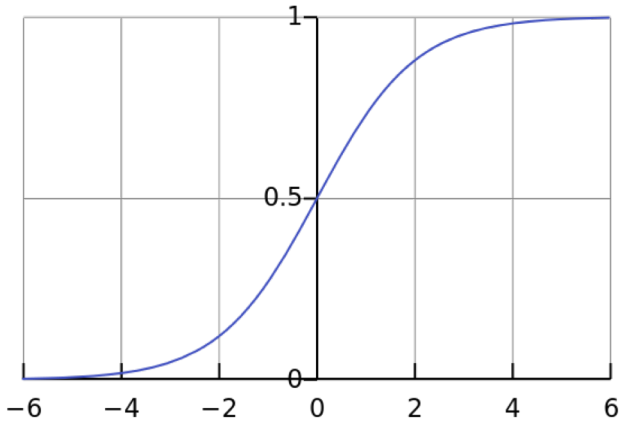
- Linear regression (linear in model parameters): $\hat{y}_i = \sum_{j=0}^{p} A_{ij}\beta_j$
  - Least squares fitting: $\text{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \rightarrow min$: $\boldsymbol{\beta} = (A^T A)^{-1} A^T \boldsymbol{y}$
  - Should normalize the data
- Regularization:
  - Ridge: $J(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{d} |\beta_j|^2$ (favor small coefficients)
  - LASSO: $J(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{d} |\beta_j|$ (favor sparse set of coefficients, many are zero)
    - Can be used for feature selection
    - Determine α through cross validation
- Support vector regression
  - Did not discuss in class, but very powerful especially with nonlinear kernel
- Neural net regression
- Decision tree and random forest for regression

Classification methods

- Logistic regression
- Support vector classifier
- Neural net: fully connected
- Neural net: convolution layers + fully connected
- Decision trees and random forest

Logistic regression

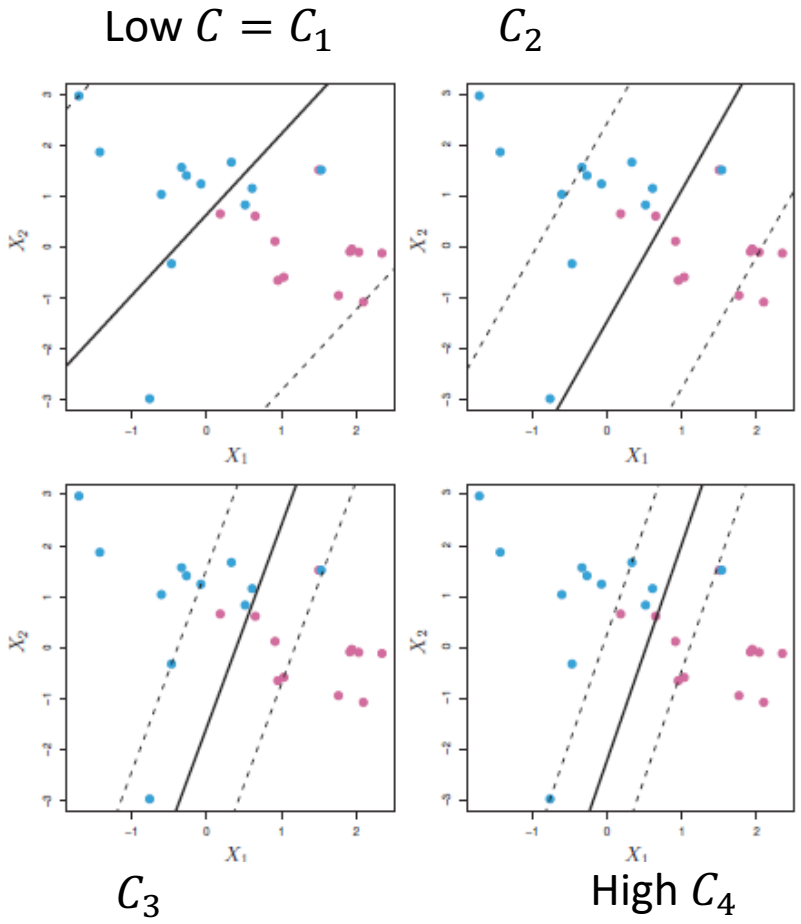
- Binary classification)
  - Linear discriminant function $z = w_0 + \sum_{j=1}^{k} w_k x_k$
  - Mapping z to probability using sigmoidal: $f(z) = 1/(1 + e^{-z})$
  - Minimize log likelihood = binary cross entropy
  - A linear classifier as it divides the feature space using hyperplane
  - Good only if the data are approximately linearly separable ☹
  - Need to transform original features if not linearly separable



- Multiclass
  - $z_k = \boldsymbol{w}_k^T \boldsymbol{x} + w_{0k}$
  - $g_k(\boldsymbol{z}) = \frac{e^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$ (softmax)
  - Train using multi-class cross entropy

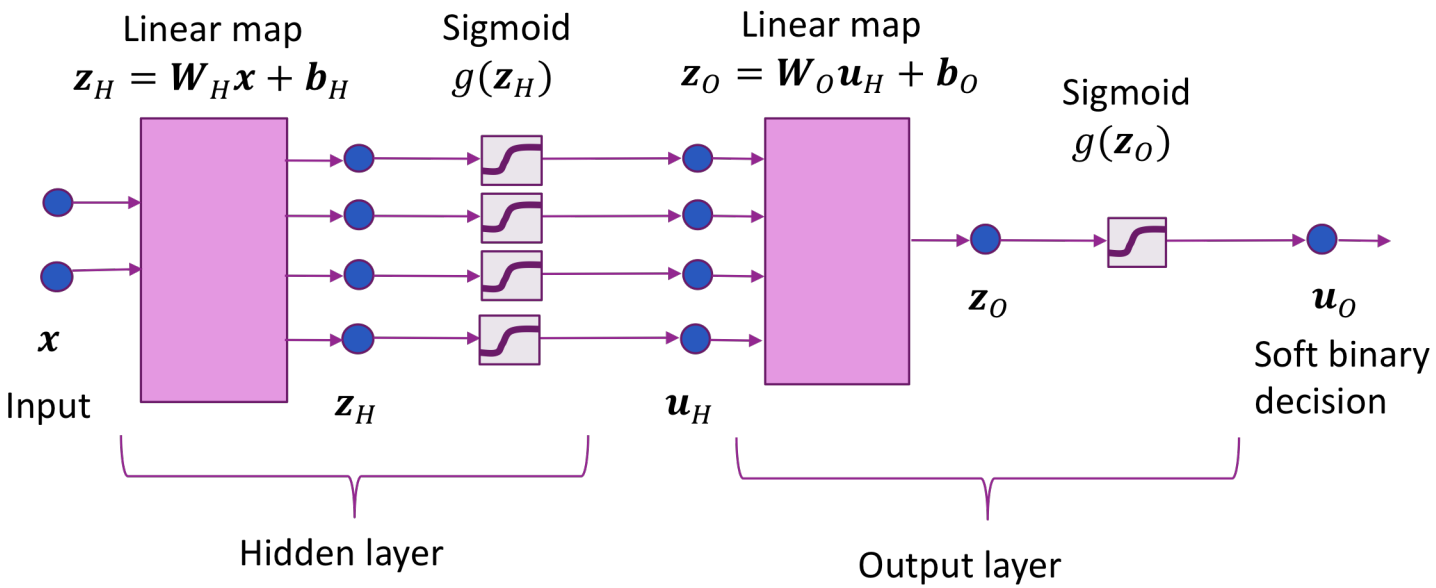
Support vector machine

- Also use linear discriminant (if using linear kernel):
  $z_i = b + \boldsymbol{w}^T x_i$
- Minimize a weighted average of the hinge loss (No loss if within the margin) and the margin. C chosen by cross validation
  $$C \sum_{i=1}^{N} \max(0, 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)) + \frac{1}{2}\|\boldsymbol{w}\|^2$$
- Support vectors: samples within the margin or on the wrong side of the line
  - $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$ ($\boldsymbol{x}_i$ are support vectors)
  - $\hat{z}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b = \sum \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + b$ (weighted average of y_i with weights proportional to "correlation" $\boldsymbol{x}_i^T \boldsymbol{x}$.
- Can be extended to nonlinear partition by using non-linear kernels
  $$z = b + \boldsymbol{w}^T \phi(\boldsymbol{x}) = b + \sum_{i=1}^{N} \alpha_i y_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})$$
- Need to save many support vectors ☹

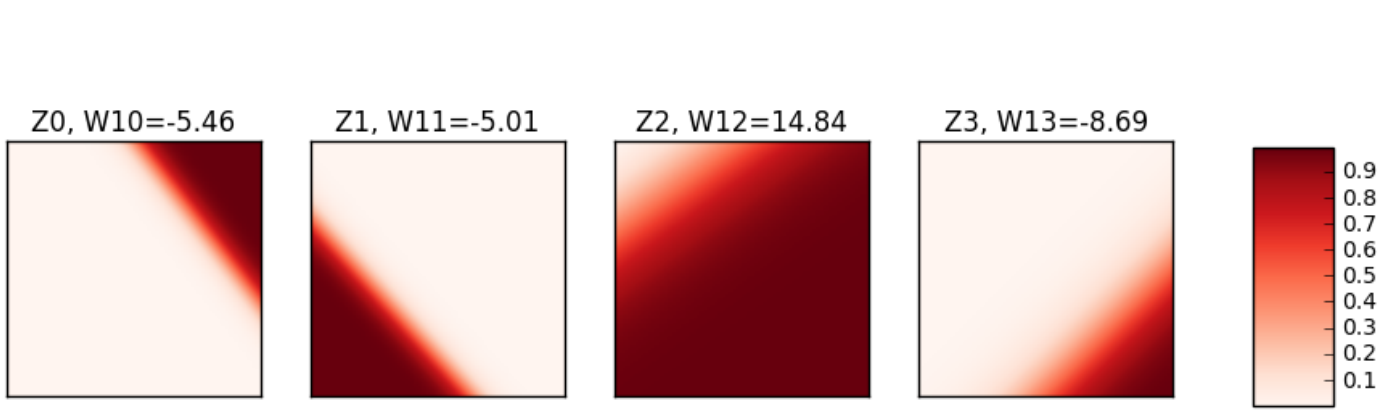


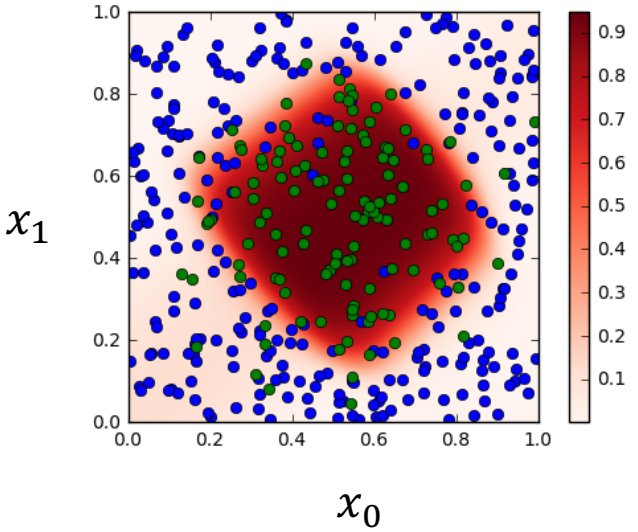
Neural networks

- Stacks of logistic regression layers
- Nonlinear mapping after each linear combination is important!
- In principle, two layers with sufficient number of hidden nodes can realize any function



Step 1 Outputs and Step 2 Outputs

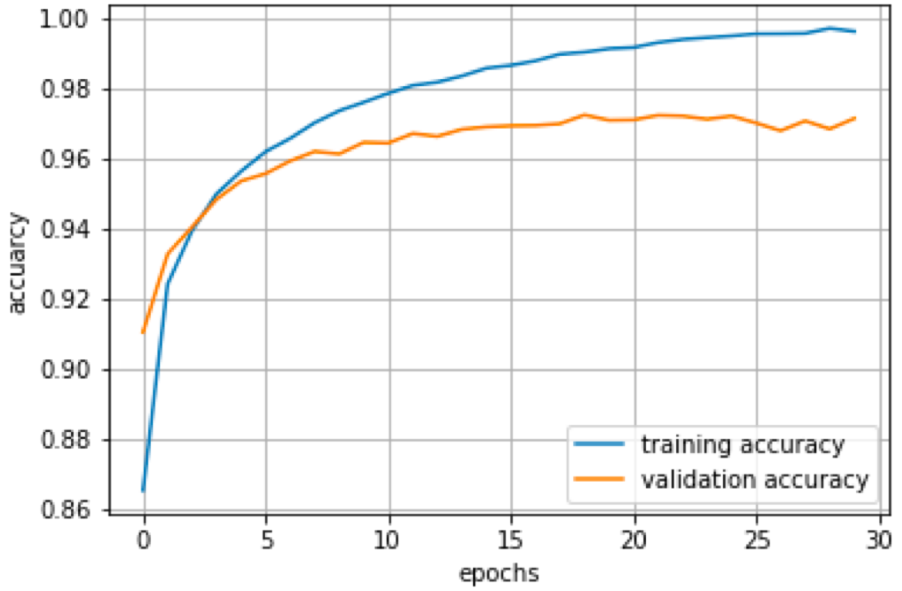




- Each output from step 1 is from a linear classifier with soft decision (Logistic regression)
- Final output is a weighted average of step 1 outputs using the weights indicated on top of the figures

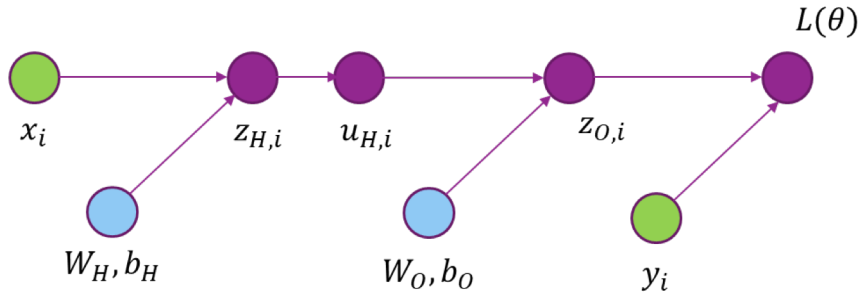
Training a neural net

- Minimize either RSS (for regression) or cross entropy (for classification) plus some regularization term
- Optimize parameters using gradient descent: Chain rule -> error backpropagation
- Stochastic gradient descent
  - Batches, epochs
  - Looking at the loss curves (training and validation) to determine when to stop
- Using Keras
  - Step 1. Describe model architecture
    - Number of hidden units, output units, activations, …
  - Step 2. Select an optimizer
  - Step 3. Select a loss function and compile the model
  - Step 4. Fit the model
  - Step 5. Test / use the model
  - Need to know how to organize your data and labels in tensors



Gradients on a Computation Graph

- Backpropagation: Compute gradients backwards
  - Use tensor dot products and chain rule
- First compute all derivatives of all the variables
  - $\partial L/\partial z_O$
  - $\partial L/\partial u_H = \langle \partial L/\partial z_O \, , \partial z_O/\partial u_H \rangle$
  - $\partial L/\partial z_H = \langle \partial L/\partial u_H \, , \partial u_H/\partial z_H \rangle$

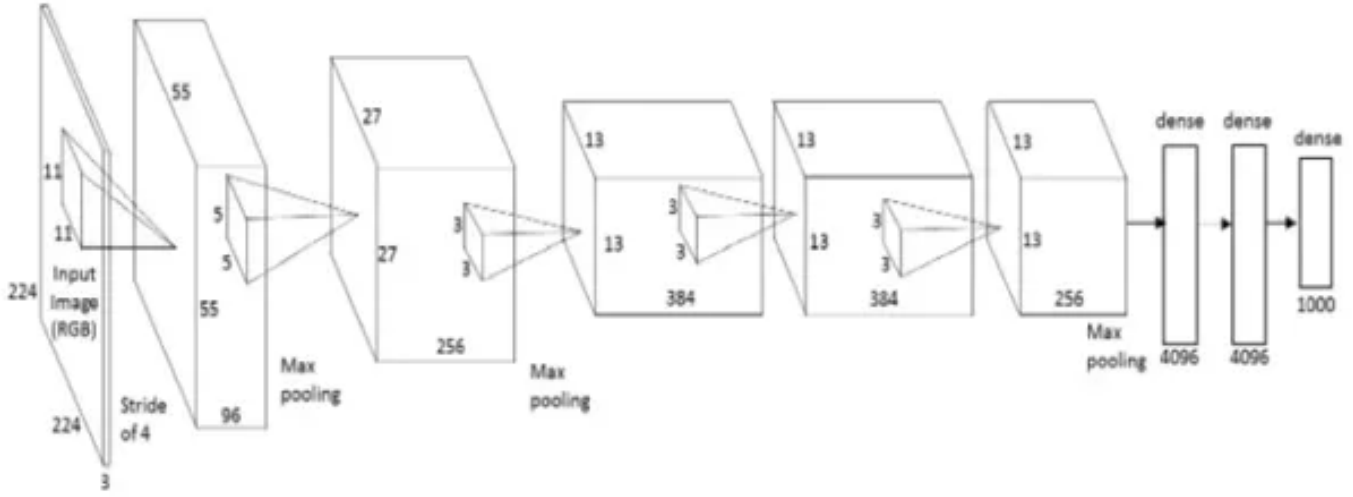


- Then compute gradient of parameters:
  - $\partial L/\partial W_O = \langle \partial L/\partial z_O \, , \partial z_O/\partial W_O \rangle$
  - $\partial L/\partial b_O = \langle \partial L/\partial z_O \, , \partial z_O/\partial b_O \rangle$
  - $\partial L/\partial W_H = \langle \partial L/\partial z_H \, , \partial z_H/\partial W_H \rangle$
  - $\partial L/\partial b_H = \langle \partial L/\partial z_H \, , \partial z_H/\partial b_H \rangle$

- You should know how to do this for a 2 layer network

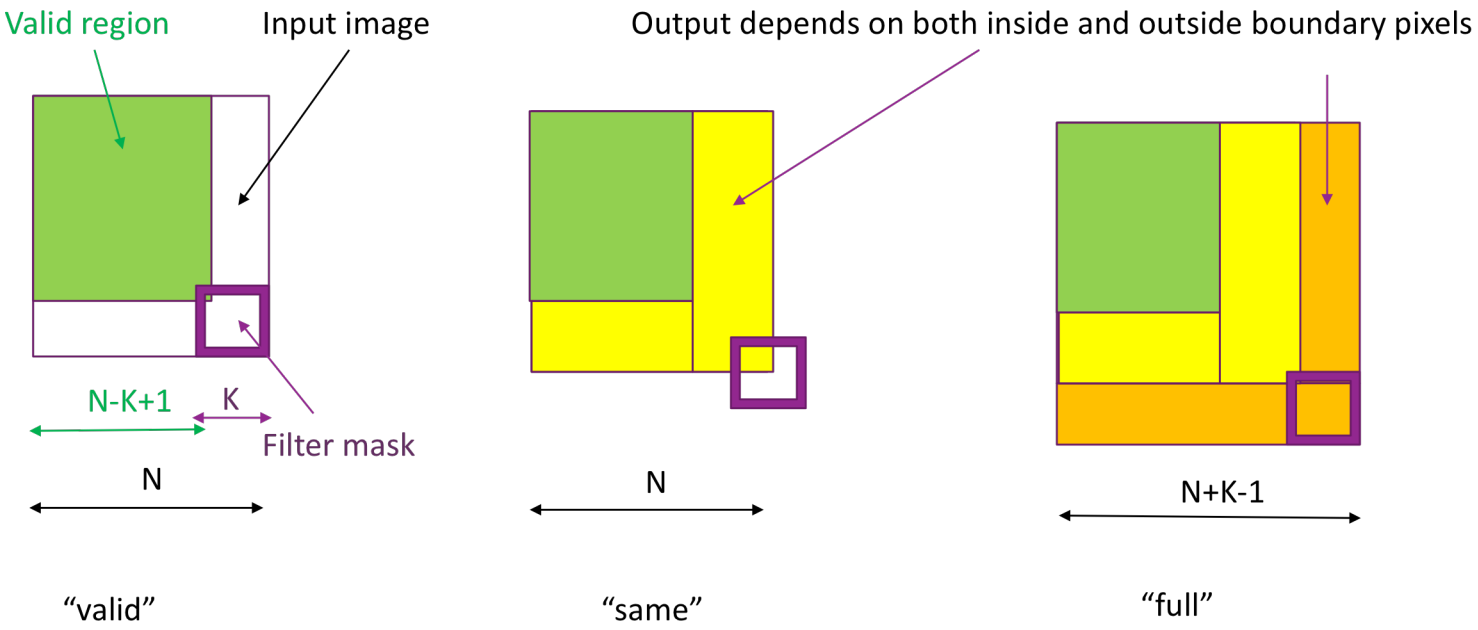
Convolutional neural networks

- Restrict the linear weighting to be local kernels sliding over all signal positions (convolution filters)
- Appropriate only for input signals for which “neighborhood” is meaningful (spatial or temporal data)
- Multiple signals over the same spatial/temporal extent are treated as “channels” and use fully connected weighting across channels – Multichannel convolution
  - Ex. color signals with 3 channels as input, multiple feature maps in the intermediate layers
- Need fully connected layer at the end for a regression or classification task



Convolution without reversal

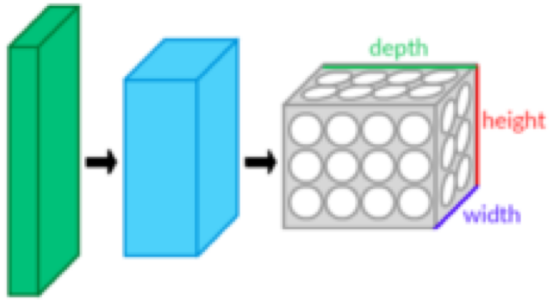
- $z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 + k_1, n_2 + k_2]$
- Boundary conditions: which pixels depend on the pixels outside the input?



- You should be able to compute convolution and count the size of valid and full region!

Convolutions with Multiple Channels

- Weight and bias:
  - W: Weight tensor, size $(K_1, K_2, N_{in}, N_{out})$
  - b: Bias vector, size N_{out}



- Convolutions performed over space and added over channels

$$Z[i_1, i_2, m] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} \sum_{n=0}^{N_{in}-1} W[k_1, k_2, n, m] X[i_1 + k_1, i_2 + k_2, n] + b[m]$$

- For each output channel m, input channel n
  - Computes 2D convolution with $W[:, :, n, m]$ (2D filters of size $K_1 \times K_2$)
  - Sums results over n
  - Different 2D filter for each input channel and output channel pair

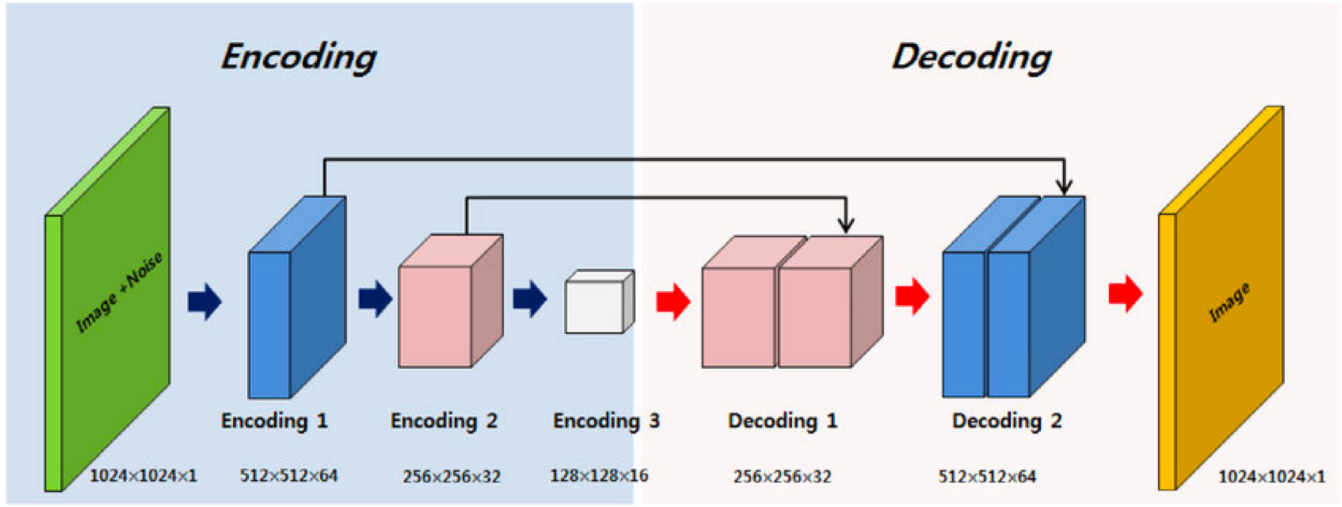
- You should be able to compute multichannel convolution for toy examples

Deep networks

- Many layers: conv + fully connected, 100K+ parameters
  - You should be able to determine the number of parameters based on a given network structure
- Need large training dataset to train
  - Data augmentation to increase data size
- When the input is raw signal, the first few layers learn the feature representation
- Regularization is important
  - Batch normalization
  - Drop out
  - L1 norm on weights
  - L2 or L1 norm on activations (output at intermediate layers)
- Transfer learning
  - Reuse front end layers of a learnt network for a different task

Autoencoders

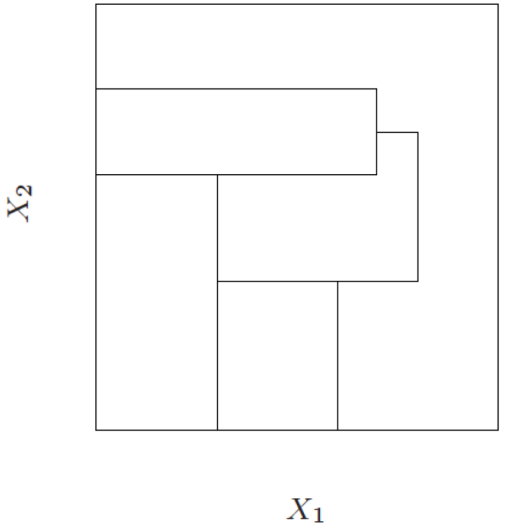
- CNN is not limited to classification/regression
- Can be used to map an image to image, speech to speech, or even image to speech
  - Denoising
  - Segmentation
  - Language translation
  - Image to caption
- Encoded features for signal reconstruction can be used to learn features without labels.

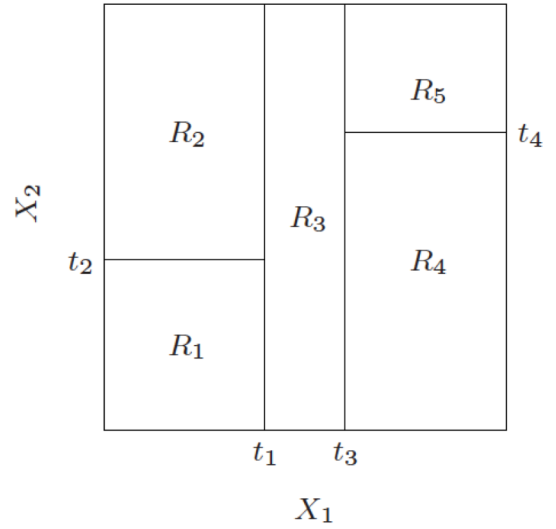


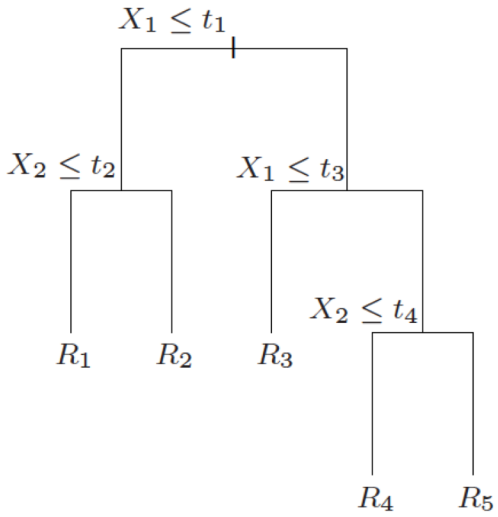
Symbol	Description	Parameters
➡	Convolution and ReLu activation	Kernel size = 3 × 3 Stride = 2 × 2
➡	Deconvolution and ReLu activation	Kernel size = 3 × 3 Stride = 2 × 2
→	Concatenate	Concatenate Encoding 1, Decoding 2 Concatenate Encoding 2, Decoding 1

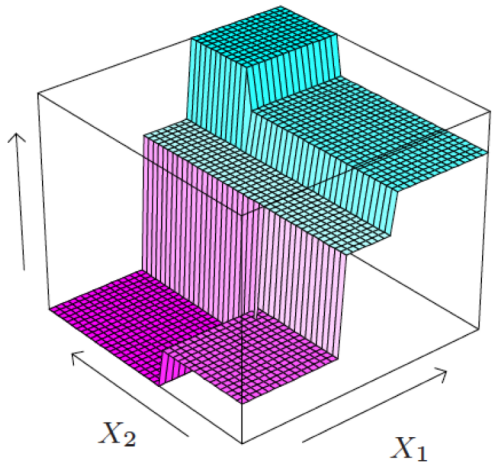
Decision trees

- Can reproduce any function (space partitioning) represented by the training data if we do not constrain the leaf node size and tree depth
- At each node, choose which feature as the splitting variable and the threshold to maximize loss reduction for the current node
- Overcome overfit by
  - pruning.
  - using multiple trees! (Bagging)
- Random forest: generating multiple independent trees
- Can generate feature ranking based on the loss reduction by these features ☺
- A single tree or a few trees is easy to interprete! and important for adoption!









Unsupervised learning --- Clustering

- Very important for understanding data without manual labels
  - Do all patients with a certain disease fall into some “unknown” sub-categories (so that a targeted treatments can be developed for each)
- Underlying assumption:
  - Samples are similar within the same cluster, and different among different clusters
  - Similarity is captured by a chosen distance metric over a chosen feature space
  - Clustering performance is as good (or bad) as the features used!
- K-means and EM-GMM can lead to meaningful clusters only when the clusters are separable in the feature space chosen and the number of clusters is known
- Unsupervised feature learning (e.g. autoencoders, spectral embedding) can be used to learn the features

K-means and GMM clustering

- K-means: Represent each cluster by its mean (centroid), assign a sample to the nearest centroid
- GMM: Represent each cluster by its mean and covariance matrix and prior probability, assign a sample by computing the posterior probability (soft assignment)
- Clusters are determined iteratively (EM algorithm)
  - E-step: Determine cluster assignment (nearest neighbor in K-means)
  - M-step: Determine cluster parameters (centroid update in K-means)
  - Greedy algorithm: Sensitive to initial solution

What features to use?

- Given many hand-crafted features, how to select the useful features?
  - Ideally, try all combination of subsets of features (may not be feasible)
  - Applying L1 norm on weights as a regularization term (applicable to linear regression, logistic regression, Neural net)
  - Using decision tree / random forest to generate feature ranking
  - Forward/backward feature selection
- Feature dimension reduction by PCA
  - Using PCA on original features, use PCA coefficients with higher variances
  - Mathematically sound, but does not “select” among all original features
  - Useful for both supervised and unsupervised problems!
- Feature learning using deep networks
  - Applicable when you have raw data and very large datasets
  - Can train the network end-to-end for a classification/regression task
  - Can use auto-encoder structure just for feature learning
- Other feature learning methods
  - Spectral embedding

Principle component analysis

- Decompose a raw signal (vector) as a weighted sum of some principle components (basis vectors)
- Determine the principle components to maximize the variances captured
  - Eigenvectors of the covariance matrix of the signals
  - Unsupervised: does not need to know “labels” for the given sample vectors
- Properties of PCs
  - Orthonormal to each other
  - Energy preservation
  - Energy compaction
- PCA is powerful for feature dimension reduction because it decorrelates original features and concentrate the total energy to a few coefficients
- Often used as the first step to convert raw features to reduced/normalized features for classical regressors/classifiers/clustering

So, which method should I use?

- Sadly, no simple answer
- Given a problem, rule out the unsuitable methods
  - Regression? Classification? Clustering?
  - Available amount of data with ground truth label, known or unknown features, is feature selection necessary?
  - Visualize the data! (Are data linearly separable or at least form some clusters?)
- When you have limited data and plausible features: Ideally try out all plausible methods, each one with optimized parameters and feature subset through cross validation
- When you have lots of data:
  - Convnets with many conv layers if your input signal has spatial/temporal structures
  - Neuralnets with a few dense layers otherwise
  - Simply divide the data into a training and testing set
- Data preprocessing (Very important!):
  - Data imputation (filling in missing entries), hot encoding for categorical features, data normalization and whitening
- Big open challenge – How to do machine learning with limited data!