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

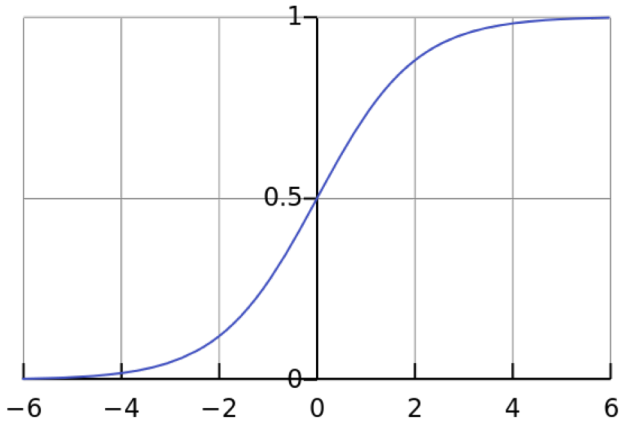
- Linear regression (linear in model parameters): $\hat{y}_i = \sum_{j=0}^{p} A_{ij}\beta_j$
  - Least squares fitting: $\text{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \rightarrow min$: $\boldsymbol{\beta} = (A^T A)^{-1} A^T \boldsymbol{y}$
  - Should normalize the data
- Regularization:
  - Ridge: $J(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{d} |\beta_j|^2$ (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

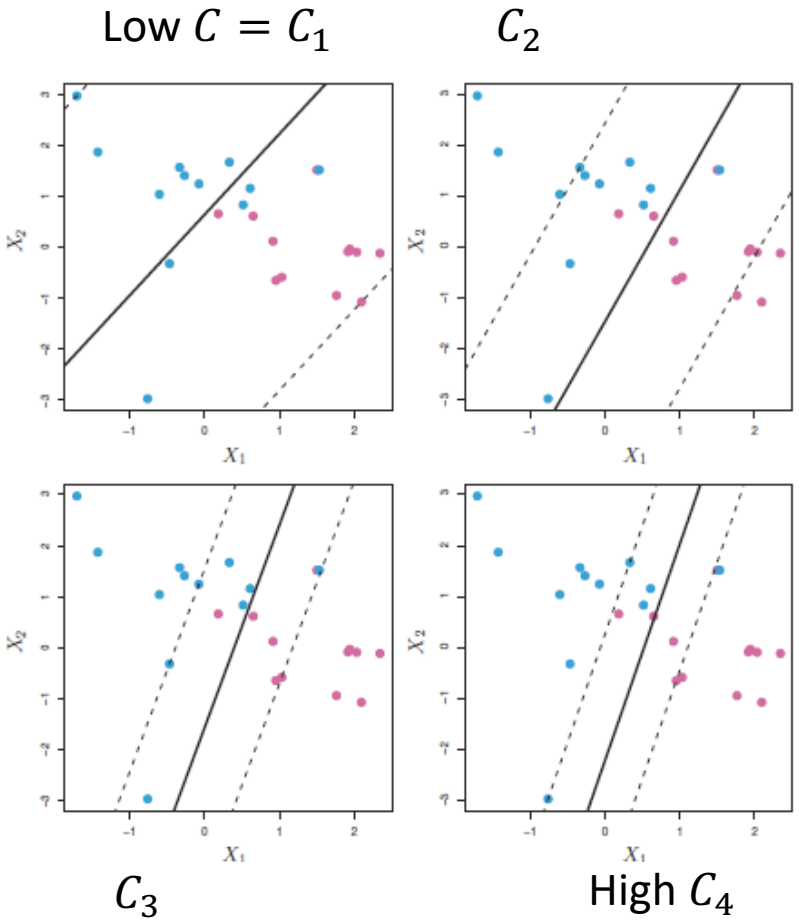
- Binary classification)
  - Linear discriminant function $z = w_0 + \sum_{j=1}^{k} w_k x_k$
  - Mapping z to probability using sigmoidal: $f(z) = 1/(1 + e^{-z})$
  - Minimize log likelihood = binary cross entropy
  - A linear classifier as it divides the feature space using hyperplane
  - Good only 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

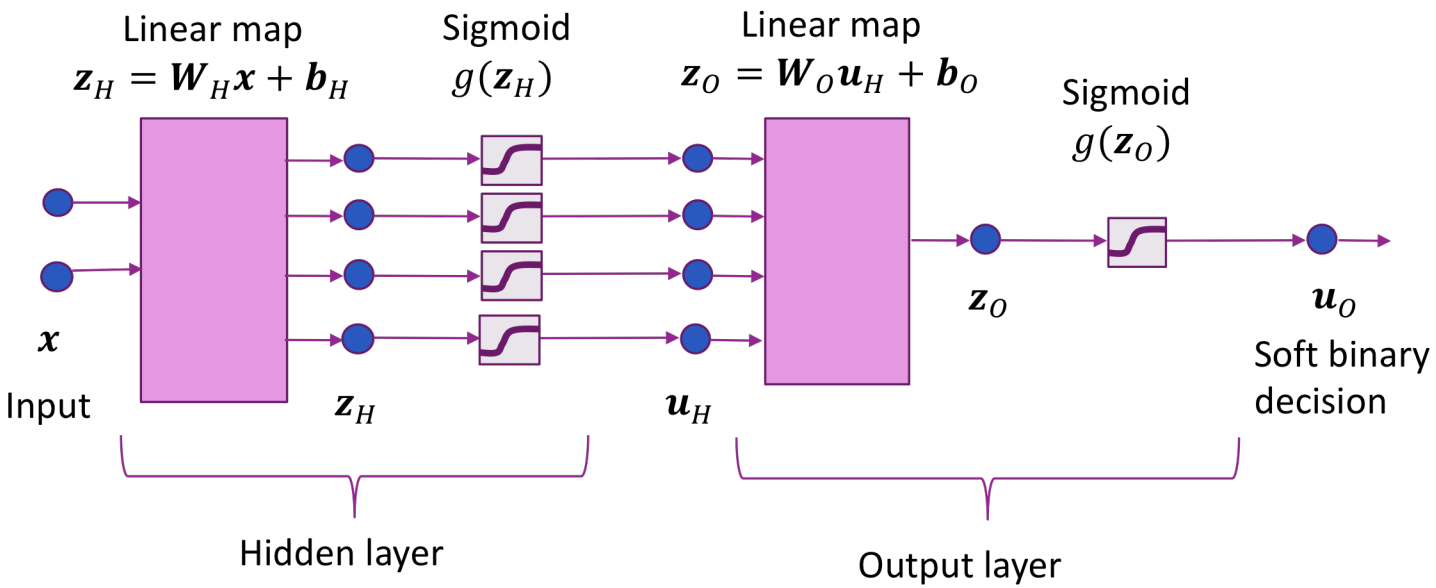
Support vector machine

- 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 ☹

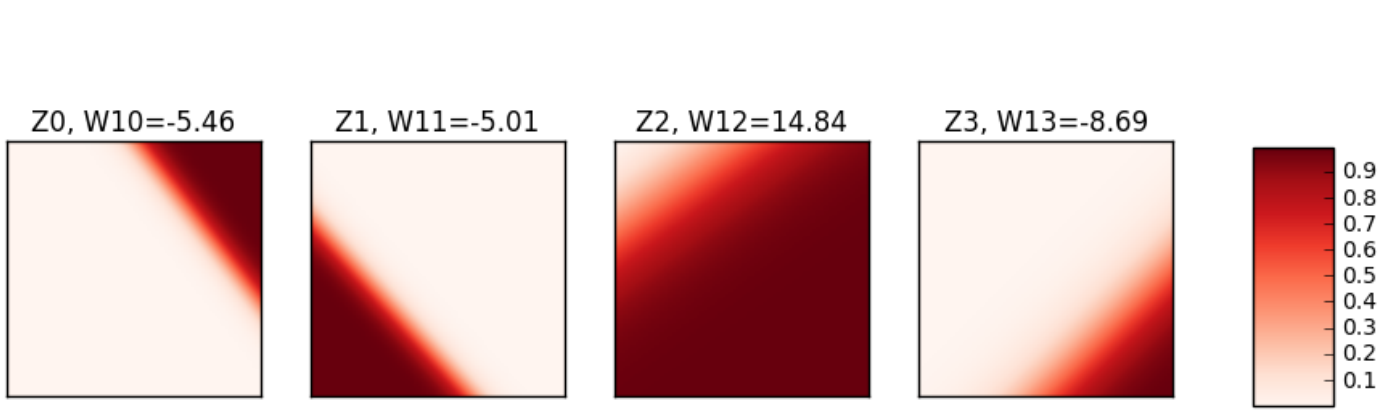


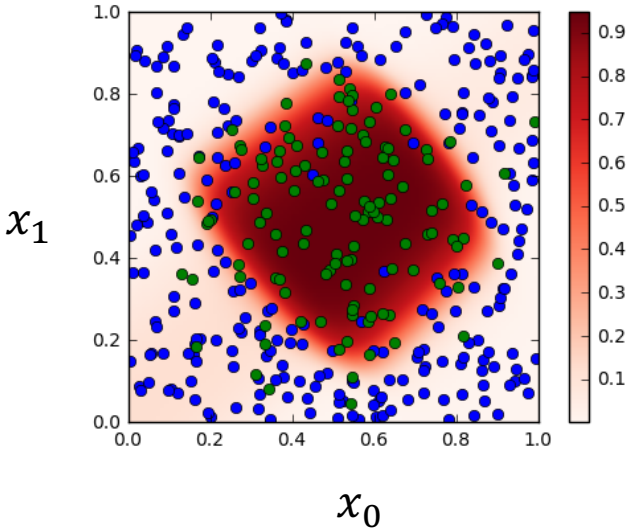
Neural networks

- 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

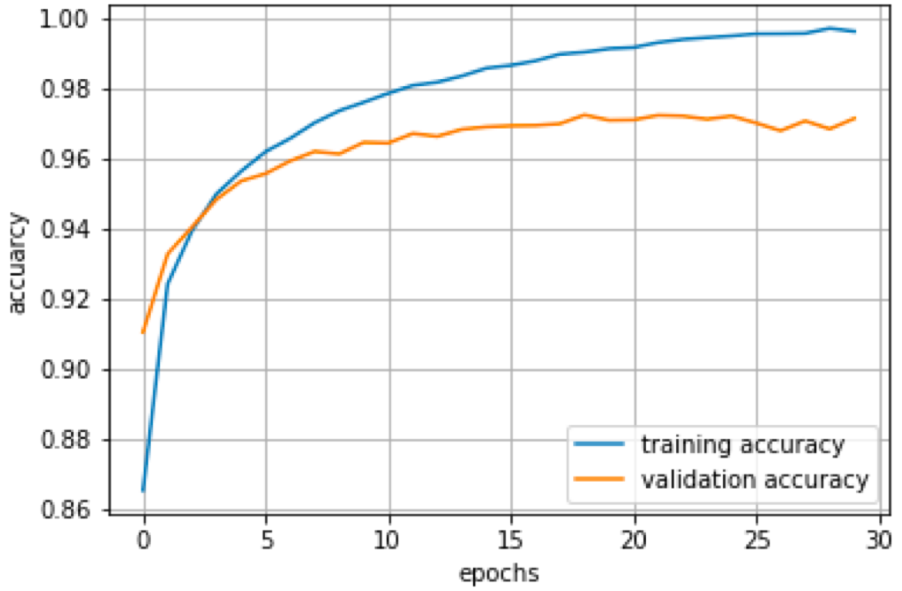




- Each output from step 1 is from a linear classifier with soft decision (Logistic regression)
- Final output is a weighted average of step 1 outputs using the weights indicated on top of the figures

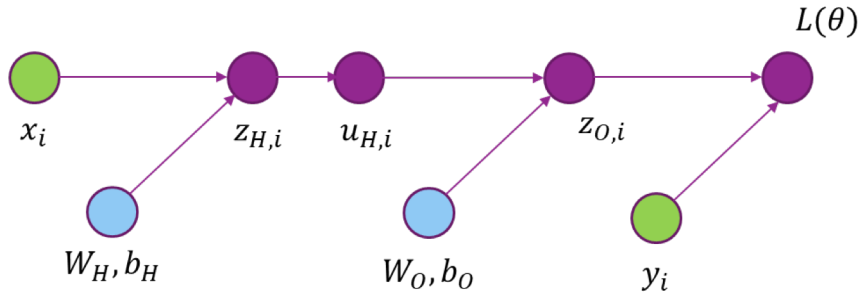
Training a neural net

- 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