E9 205 Machine Learning for Signal Processing

Neural Networks

09-11-2016

Established 1911

INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

Why do we look beyond SVMs and other classifiers

- SVM and linear regression models

$$f(\mathbf{x}; \mathbf{w}, b) = sgn(\mathbf{w}^T \phi(\mathbf{x}) + b)$$

- Choice of representation ϕ is heuristic or chosen based on validation.
- Need a mechanism for learning the non-linear mapping function.
    - Compromise the good properties of convex optimization.

Neural Network Architecture with Hidden Layer

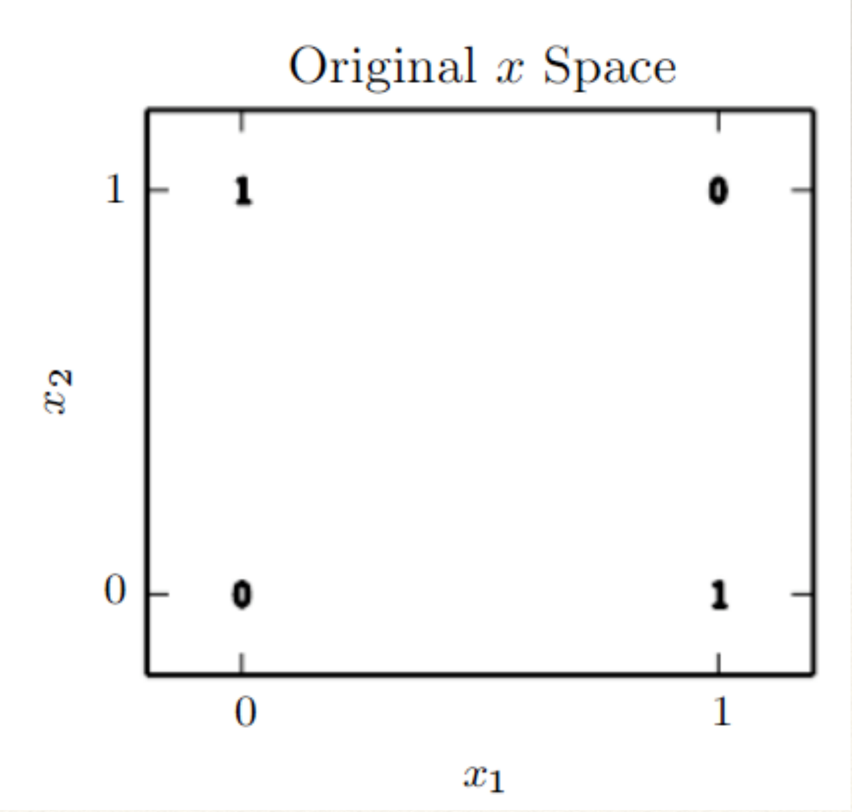
- A hidden layer representation followed by a output layer.

$$h = f^{(1)}(x; W, c) \text{ and } y = f^{(2)}(h; w, b)$$

- Output as composition of simpler function from layers

$$f(x; W, c, w, b) = f^{(2)}(f^{(1)}(x))$$

- XOR problem
  - Need a non-linear activation in the hidden layers



Neural Network Architecture with Hidden Layer

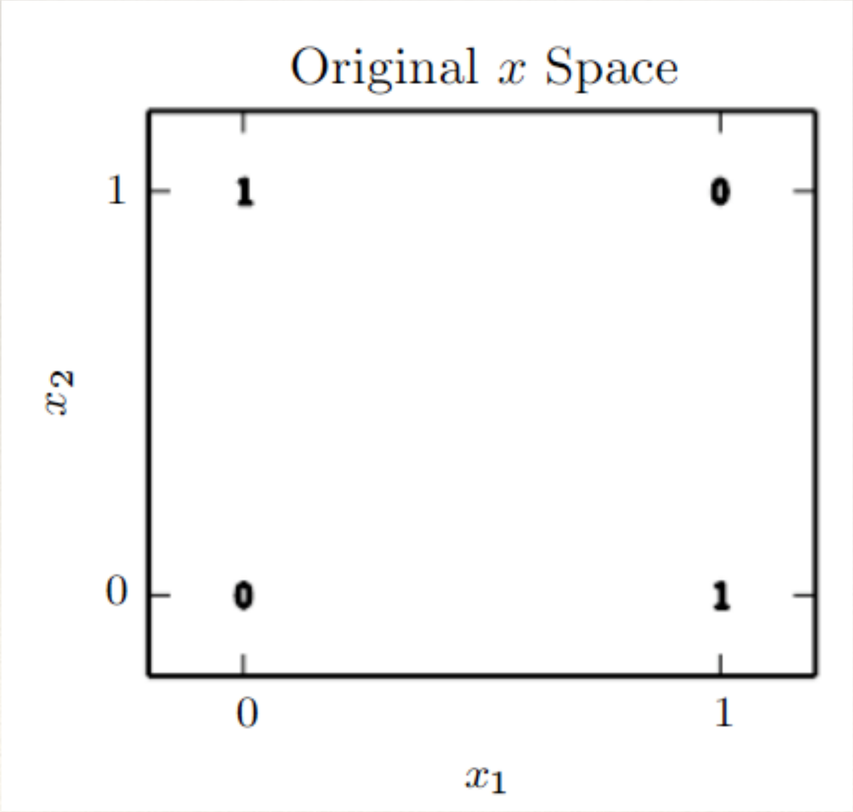
- A hidden layer representation followed by a output layer.

$$\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}) \text{ and } y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$$

- Output as composition of simpler function from layers

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$$

- XOR problem
  - Need a non-linear activation in the hidden layers



Non-linear activation functions

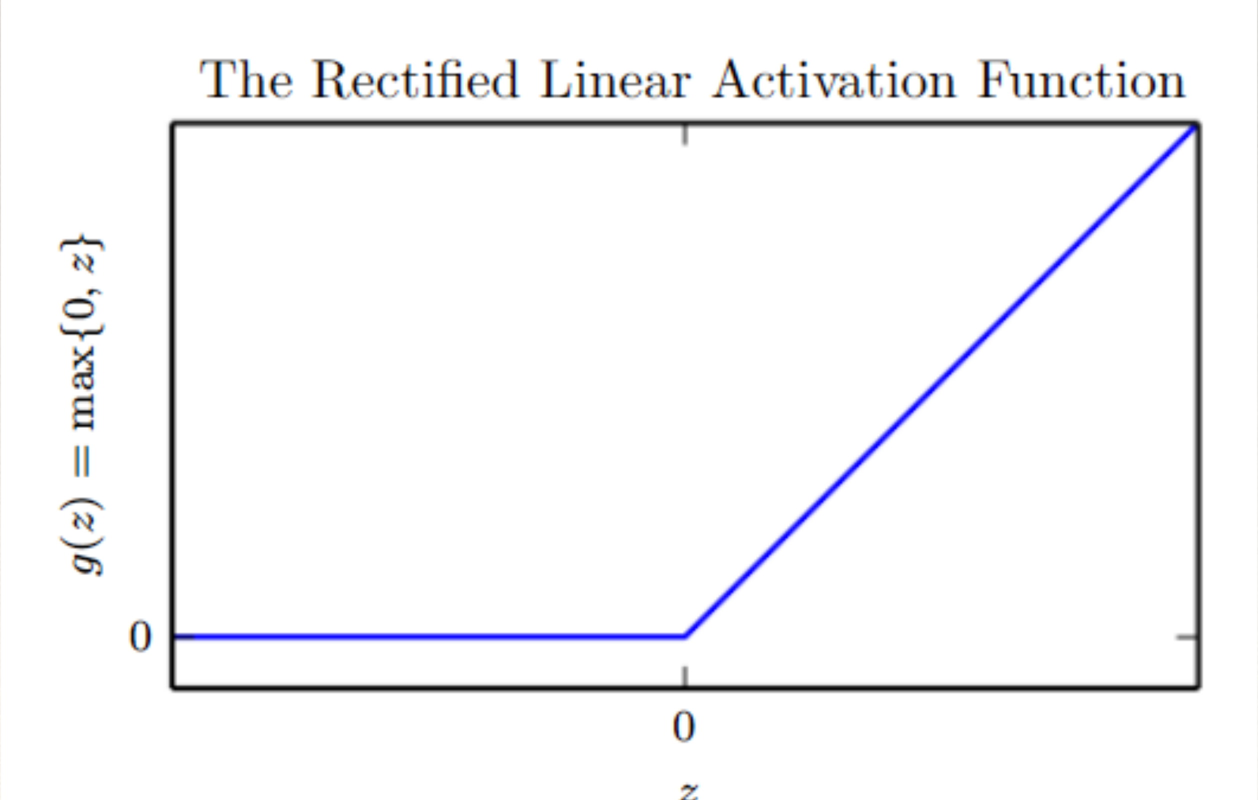
- Types of non-linear activations - element wise non-linear functions

$$h_i = g(\boldsymbol{x}^\top \boldsymbol{W}_{:,i} + c_i)$$

- Commonly used ReLU activation with variations

$$g(z) = \max\{0, z\}$$

  - Induces sparsity
  - Piece-wise linearity
  - Fast gradient computations

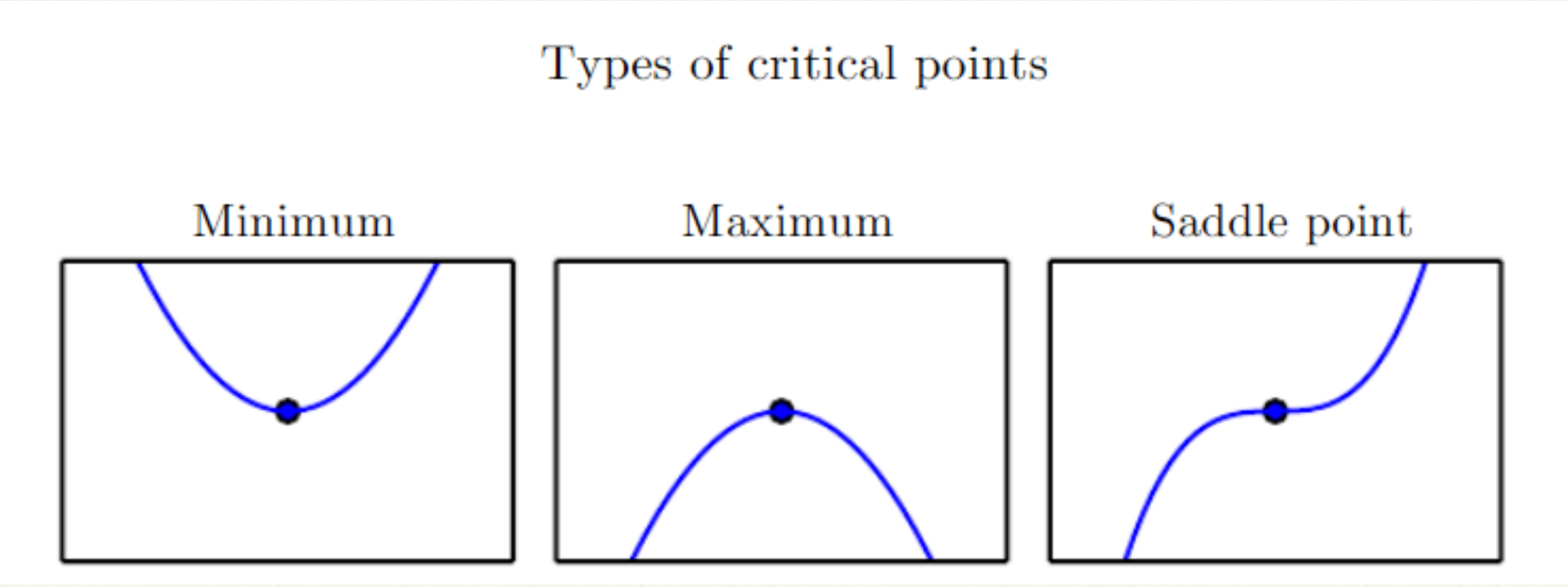


Parameter Learning

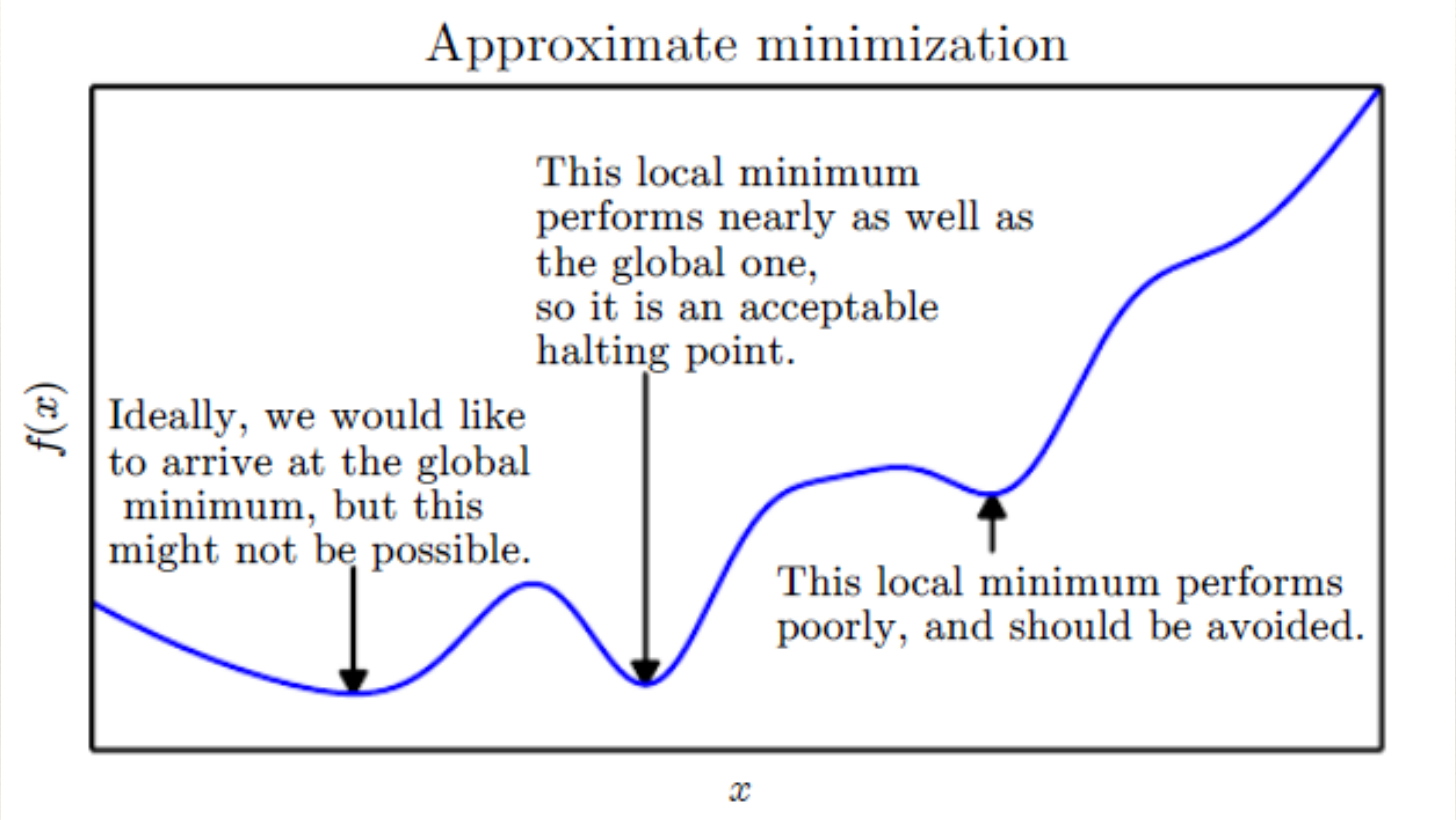
- First order Taylor series expansion (smooth continuous function $$f(x+\epsilon) \approx f(x) + \epsilon f'(x)$$
- Make changes to parameters in such a way that objective function is minimized (for example)

$$f(x - \epsilon \, \mathrm{sign}(f'(x)))$$

- Critical points



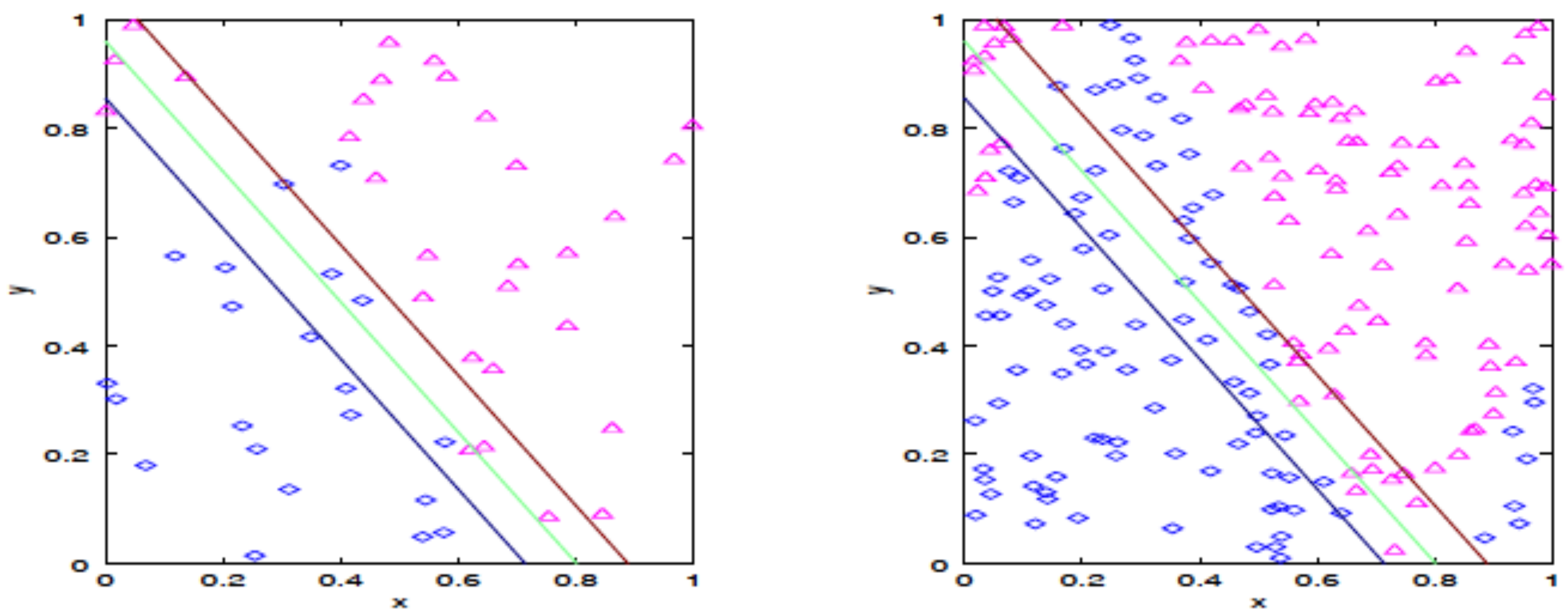
Parameter Learning



- Method of steepest descent (vector case)

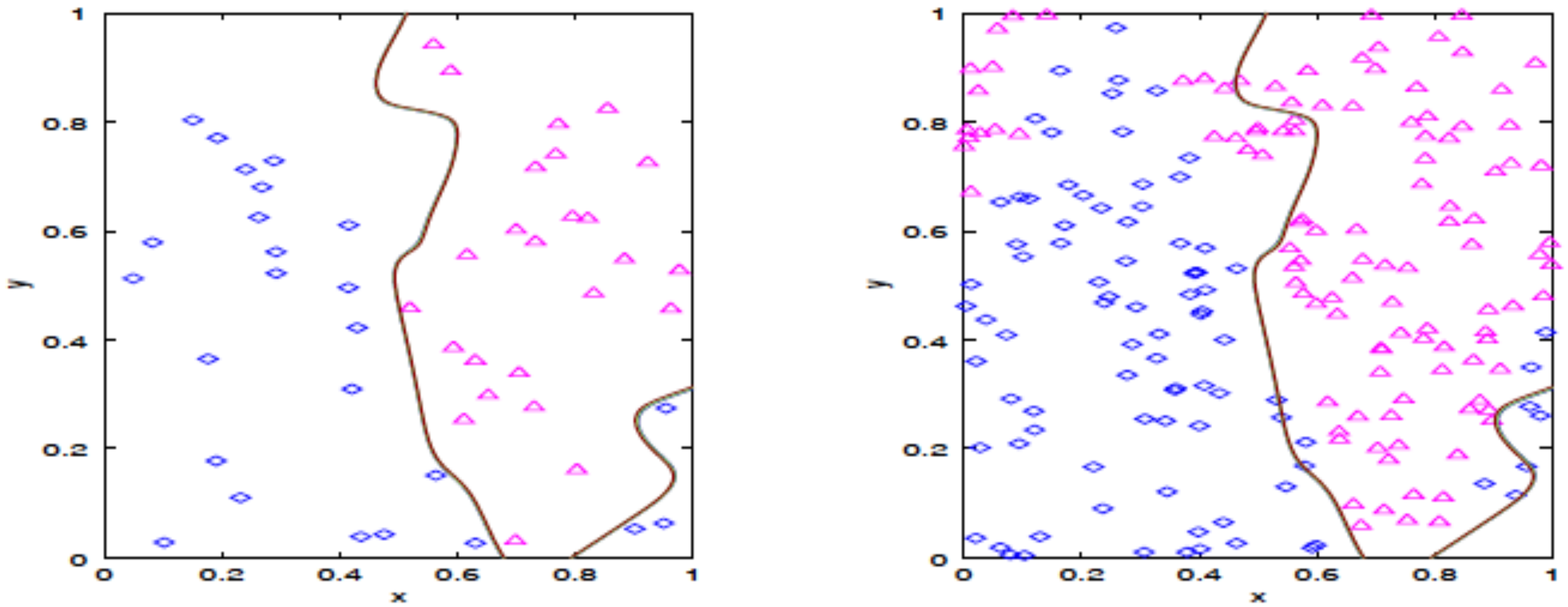
$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

Overfit versus Underfit



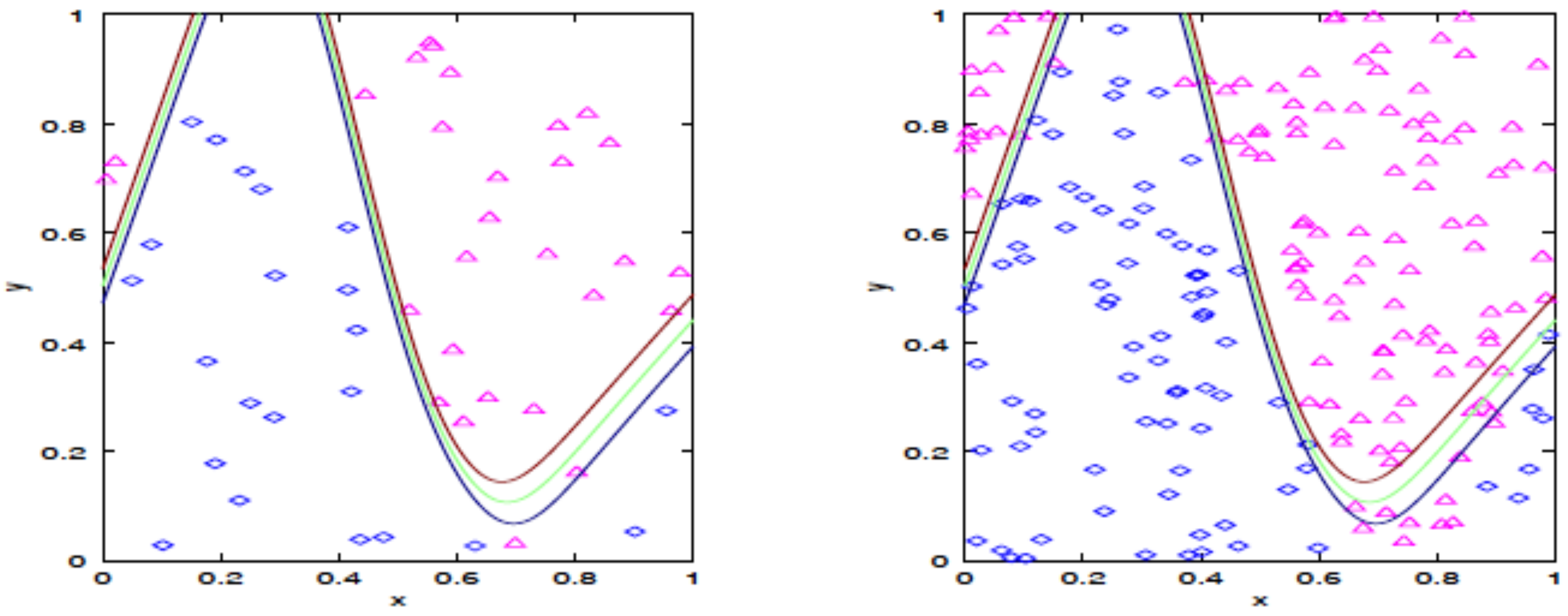
- The model is not able to capture the variability in the data (Linear Model)
- Both the training and testing error are high (15%,20%)
- Try to learn a more complex model – more features, more hidden neurons, decrease regularization
- More data would not help

Overfit versus Underfit



- The model is capturing data as well as accidental variations (100 hidden neurons)
- Training error is too low and testing error is too high (0%, and 16%)
- Try to learn a simpler model – less features, less hidden neurons, increase regularization
- More data would help

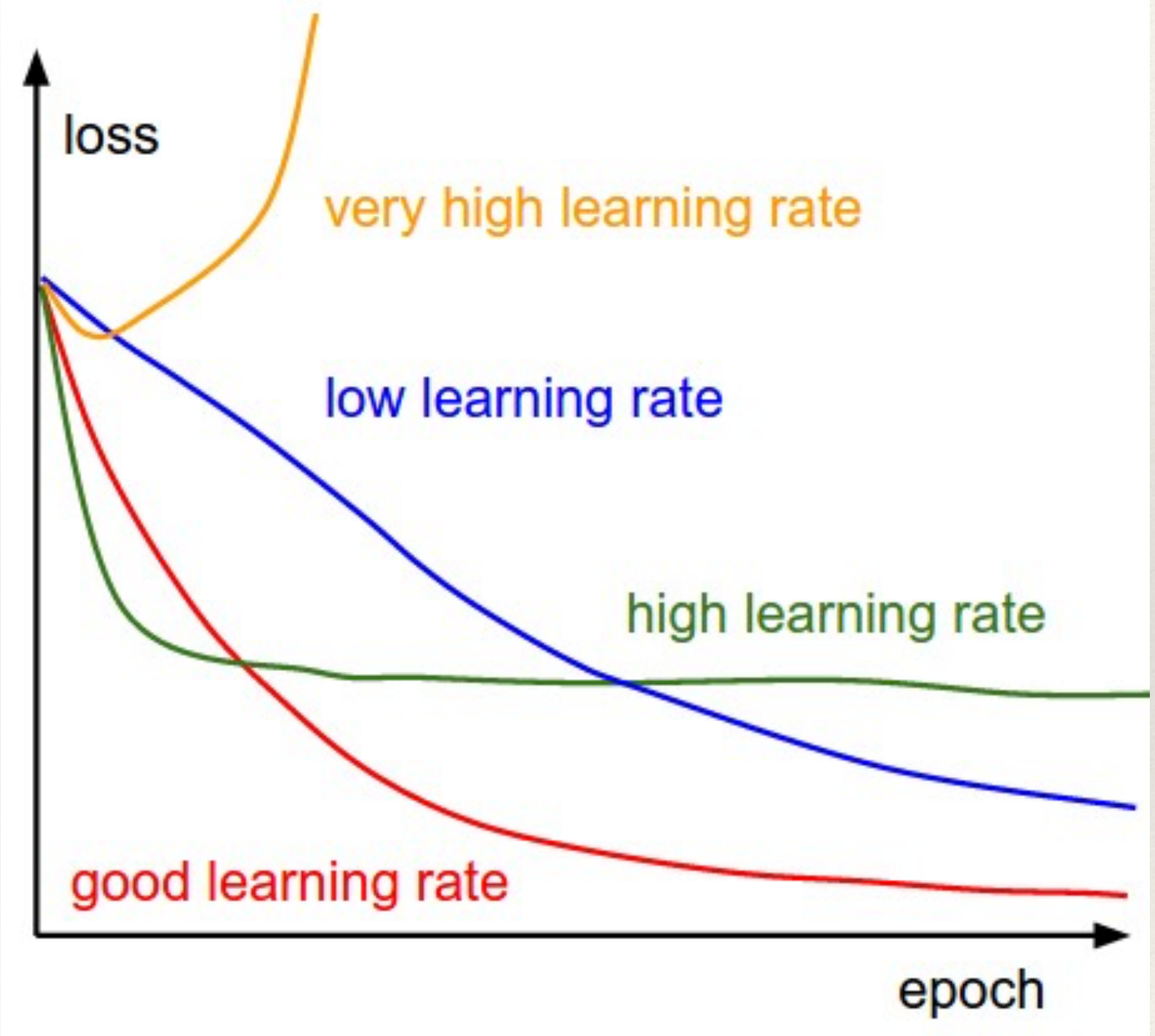
Overfit versus Underfit



- Reasonable training and test errors – (4%, 8%)
- Appropriate model – capturing only the global characteristics not details

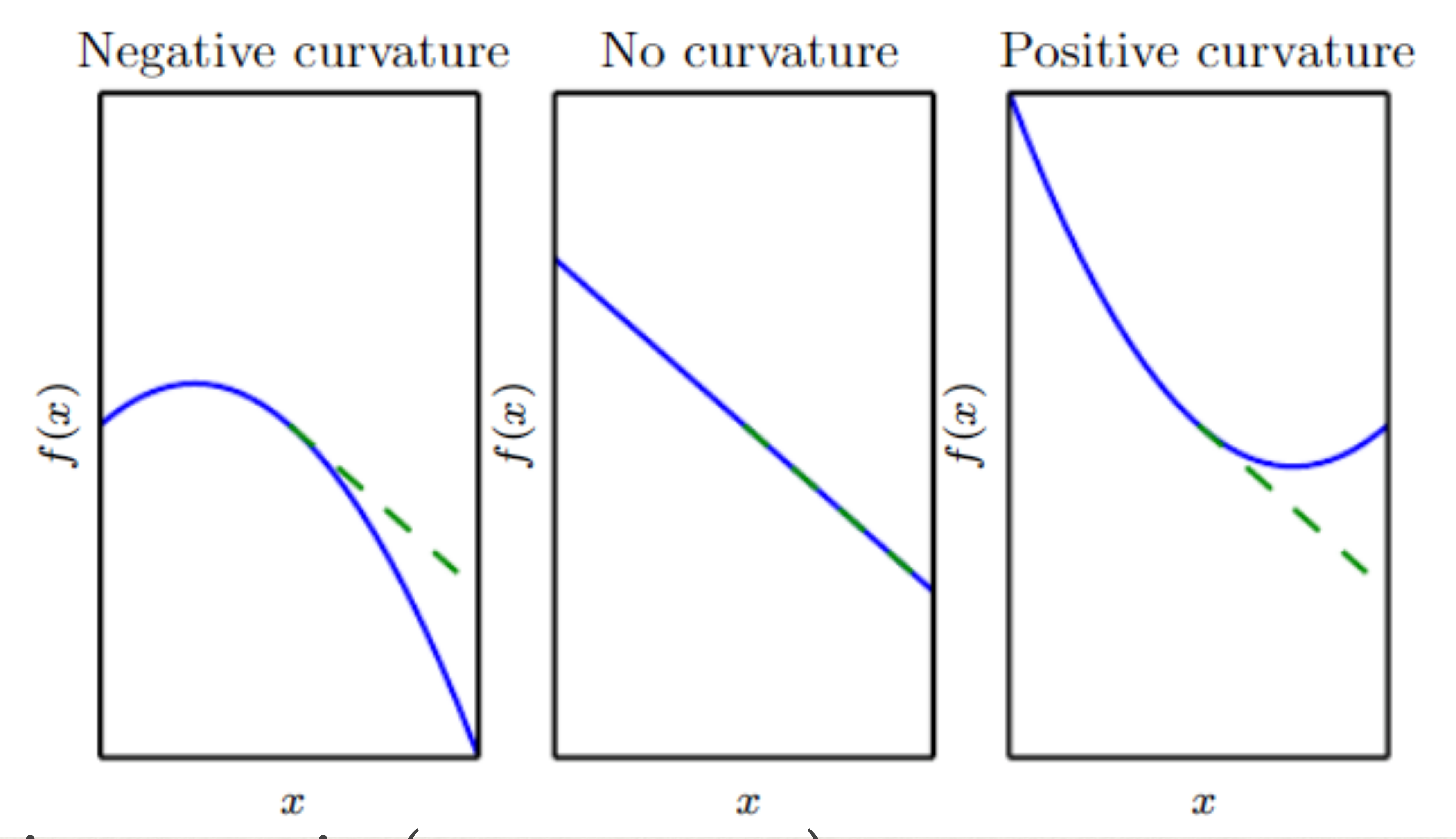
Parameter Learning Summary

- Solving a non-convex optimization.
- Iterative solution.
- Depends on the initialization.
- Convergence to a local optima.
- Judicious choice of learning rate



Beyond Gradients

- Second derivative is indicative of curvature

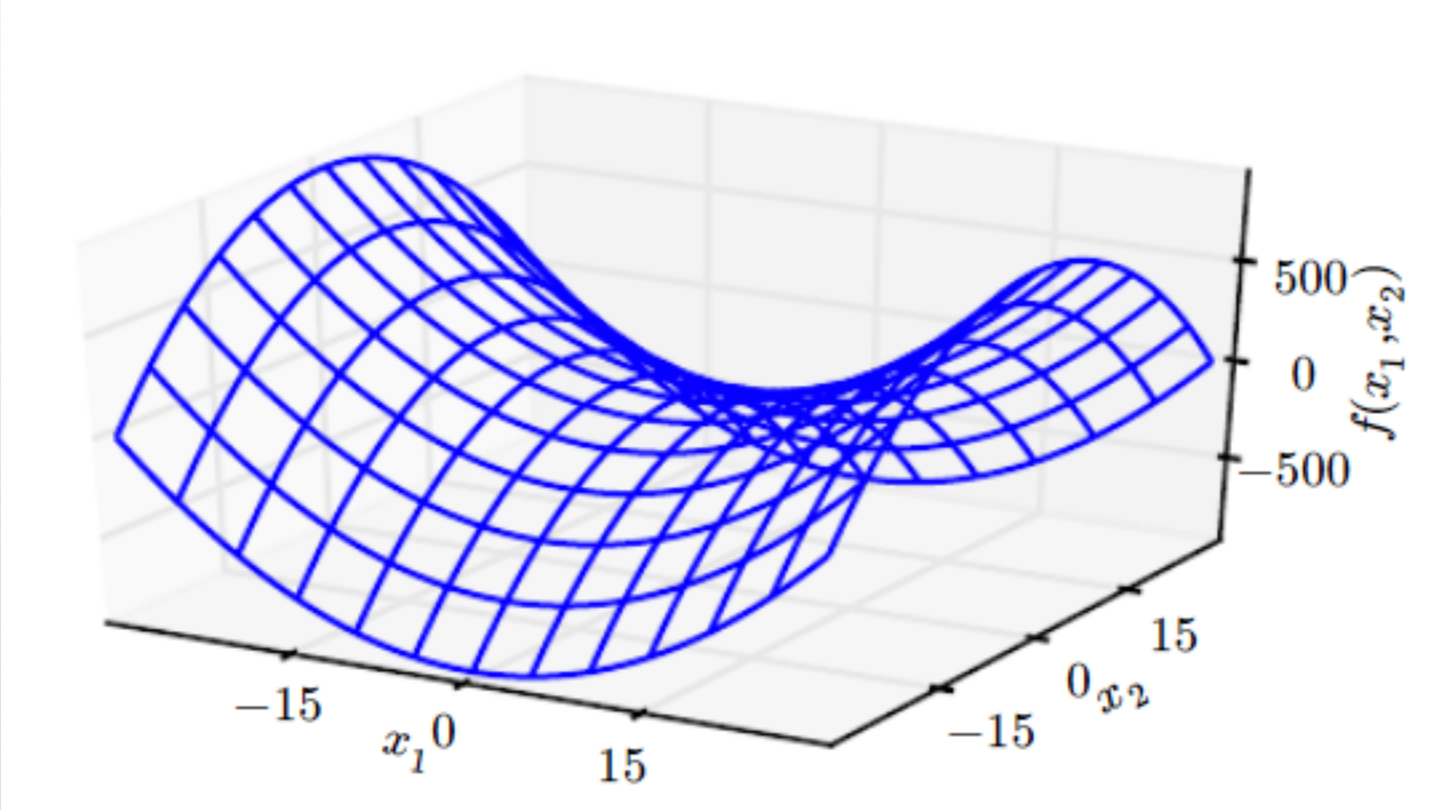


- Hessian matrix (vector case)

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$$

Beyond Gradients

- Nature of saddle points (two dimensions)



- Maximum value of learning rate

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}.$$

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}.$$

Newton's Method

- Problem with first order methods - choosing learning rate.
- Second order Taylor series

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

- Newton's method of parameter learning

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)})$$

  - Works well when function is quadratic or approximately quadratic with positive definite Hessian.

Stochastic Gradient Descent

- Standard gradient computation

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

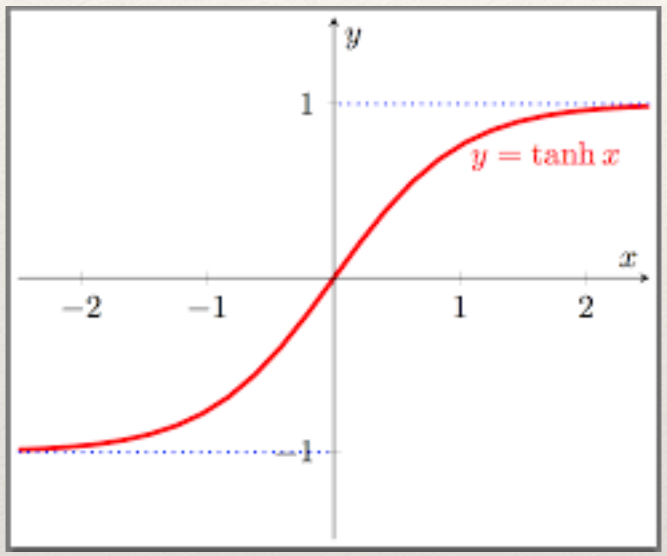
- Instead of using all the samples, using a random subset of the samples to compute the gradient

$$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

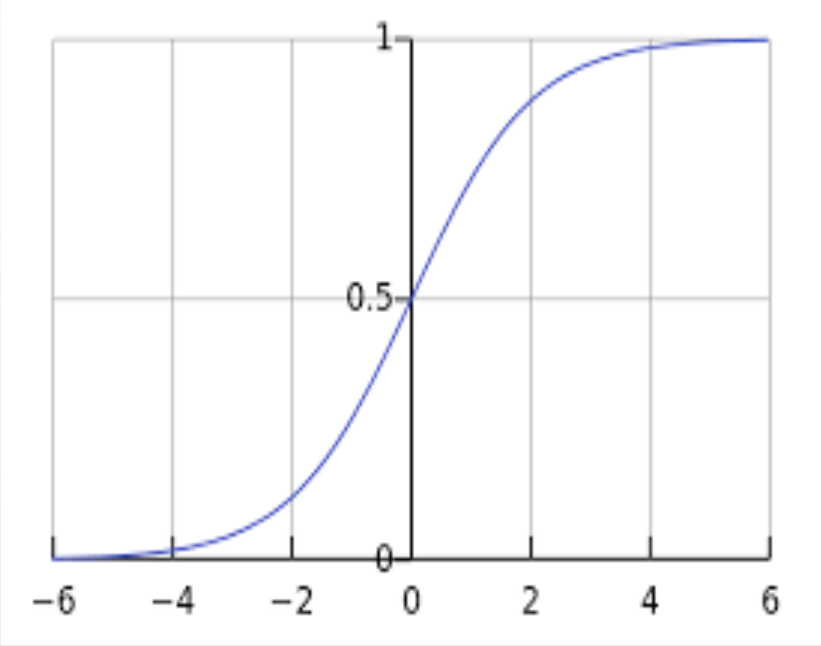
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$

Cost Function and Hidden Activations

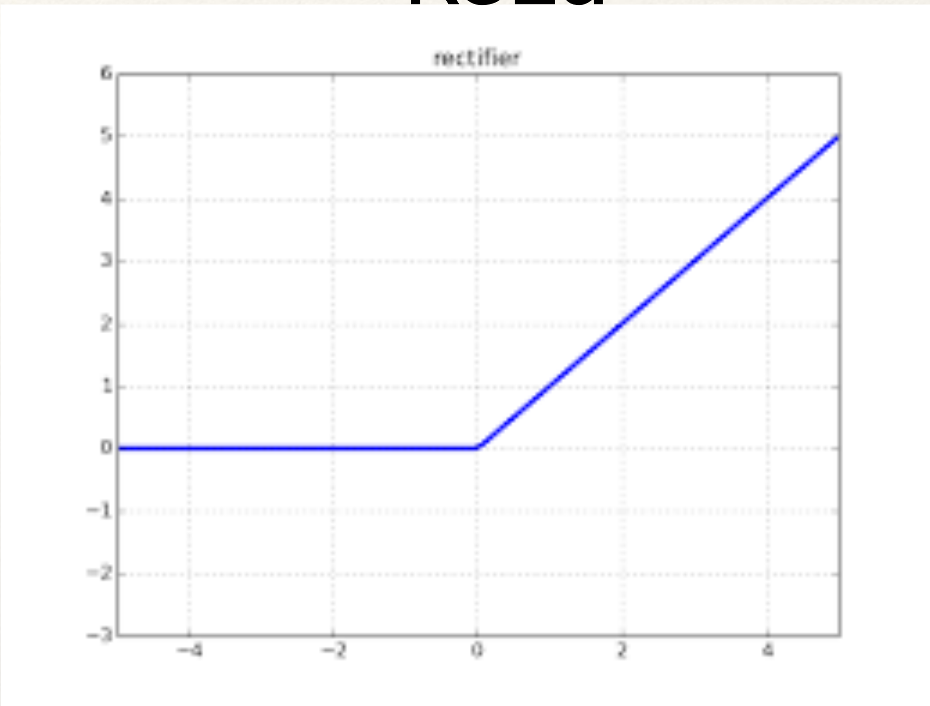
tanh



sigmoid



ReLu



Cost-Function

Mean Square Error

$$J_{MSE} = \sum_{i=1}^{M} ||\mathbf{v}_i - \mathbf{y}_i||^2$$

Cross Entropy

$$J_{CE} = -\sum_{i=1}^{M} \mathbf{y}_i^T \log(\mathbf{v}_i)$$

$\mathbf{y}_i$ are the desired outputs

Output Activation

Choice of output activation ψ

- Linear for regression
- Softmax function for classification

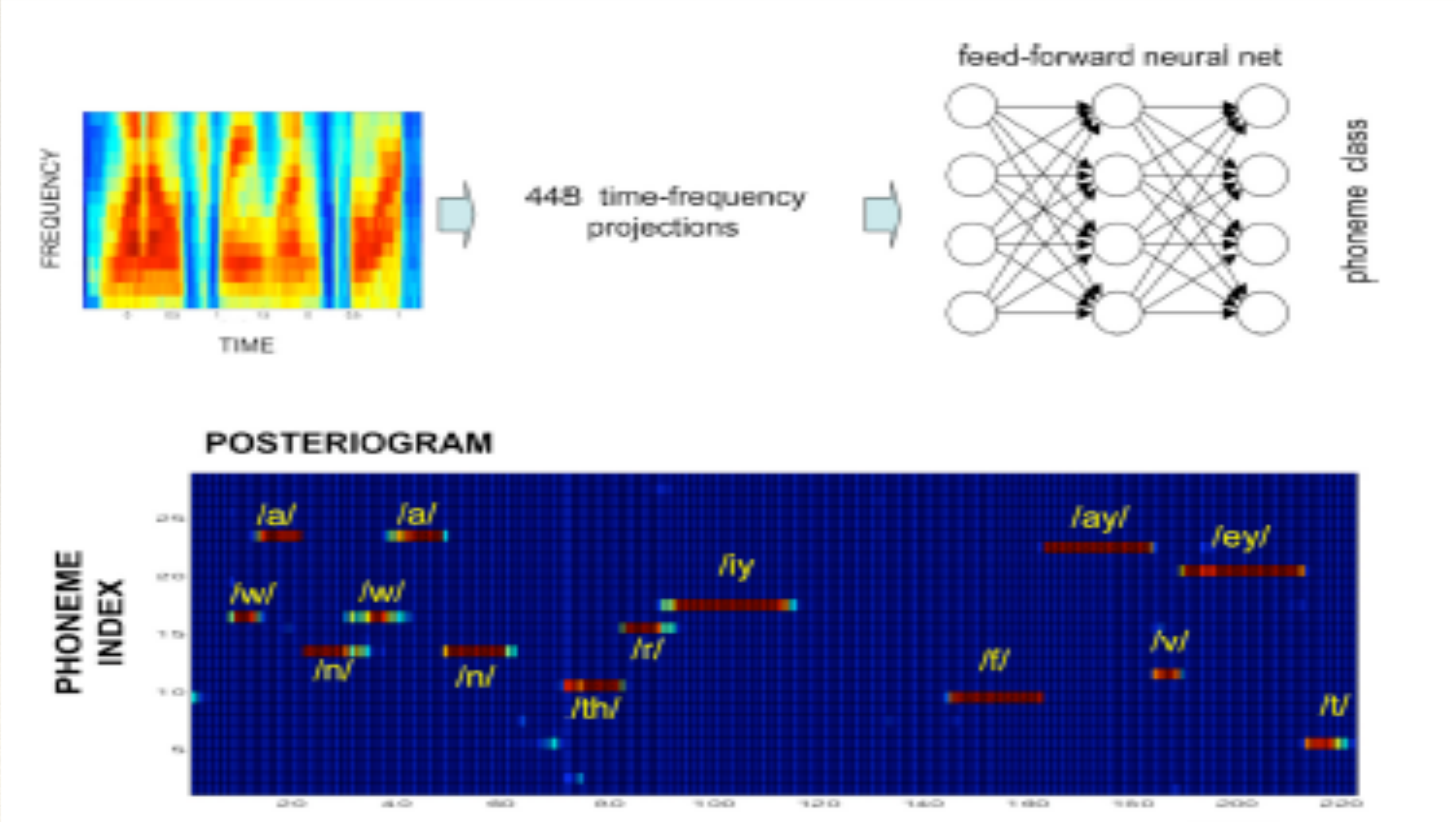
$$\psi(v_i) = \frac{e^{v_i}}{\sum_i e^{v_i}}$$

- Softmax produces positive values that sum to 1
- Allows the interpretation of outputs as posterior probabilities

Example - Train a NN to approximate speech classes using sigmoidal non-linearity and softmax target function

Properties of NN

- Neural networks estimate posterior probabilities of classes (one-of-K class outputs) [Lippmann 1991] - with mean square error and cross entropy error

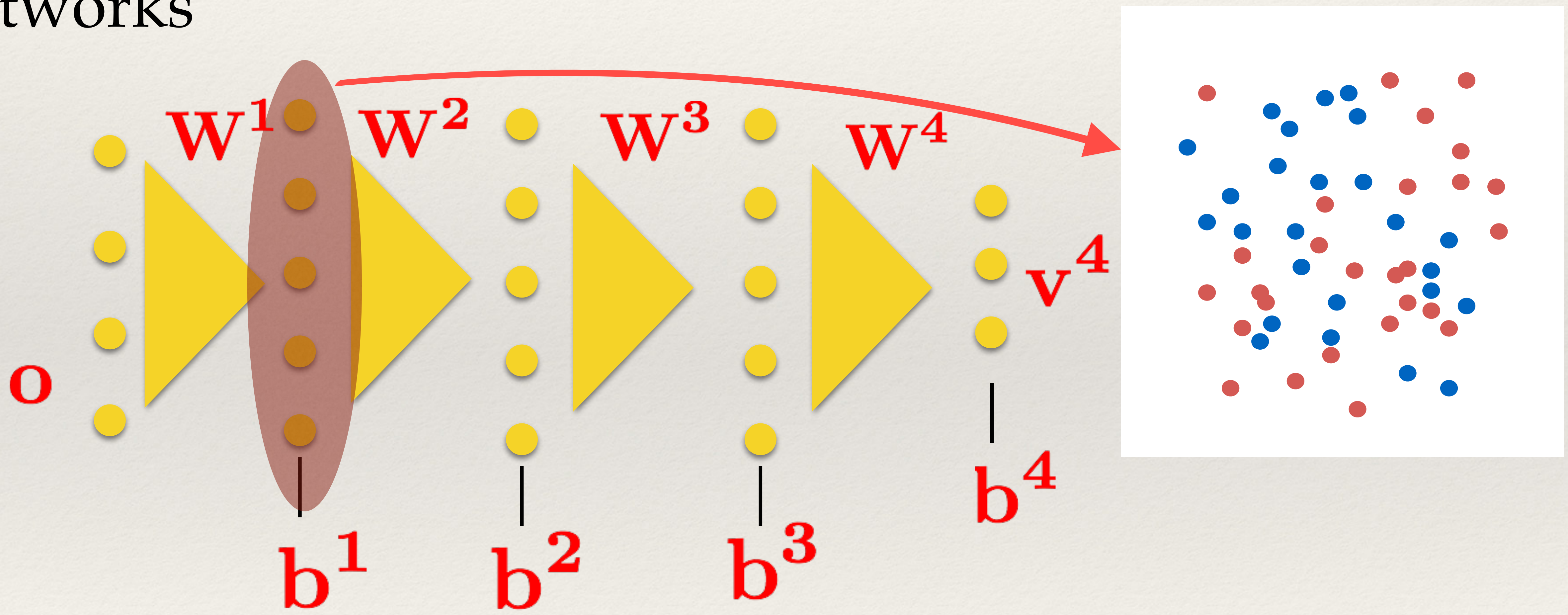


Architecture Design in NN

- Universal approximation properties of NN
  - Single layer with sufficient units may be able to approximate any continuous mapping function well.
    - Learning such a function could be difficult
    - Finding family of functions to search is cumbersome.
- Deep architectures
  - Representations which are build in a hierarchical fashion
  - Complex functions that defined from simpler functions - all being learned from the training data.

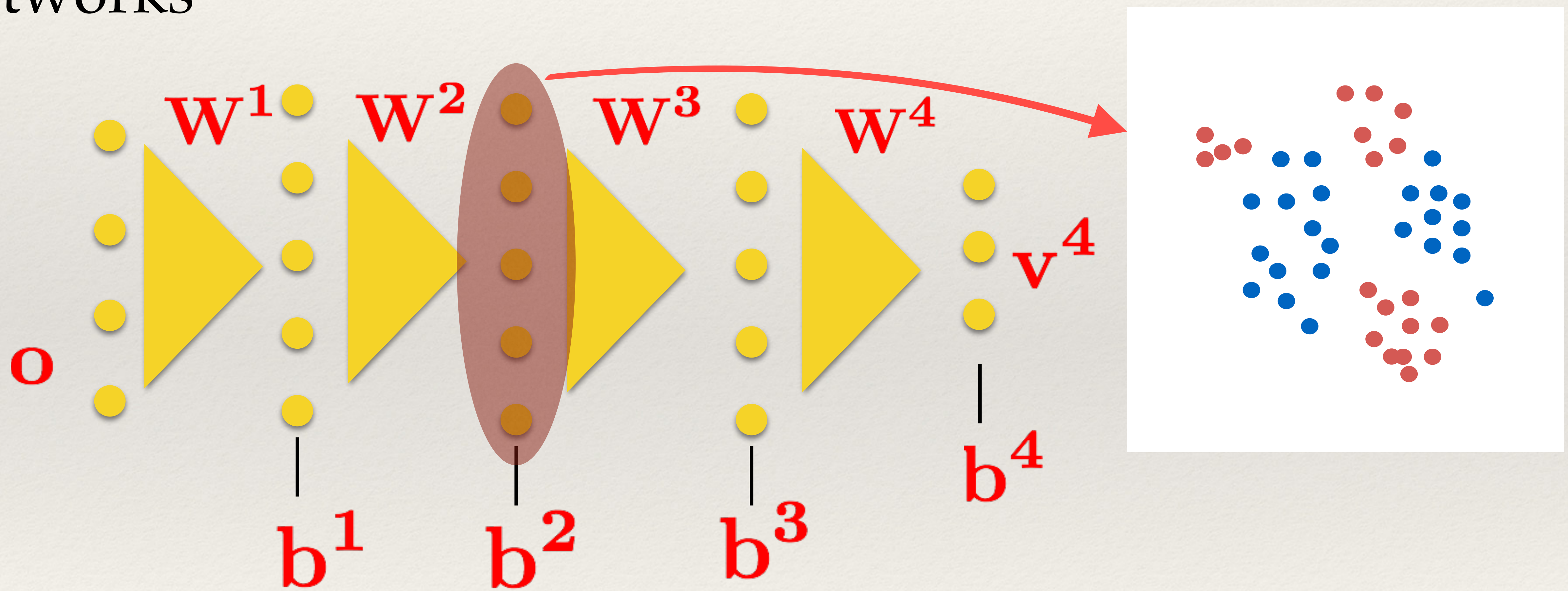
Architecture Design in NN

Neural networks with multiple hidden layers - Deep networks



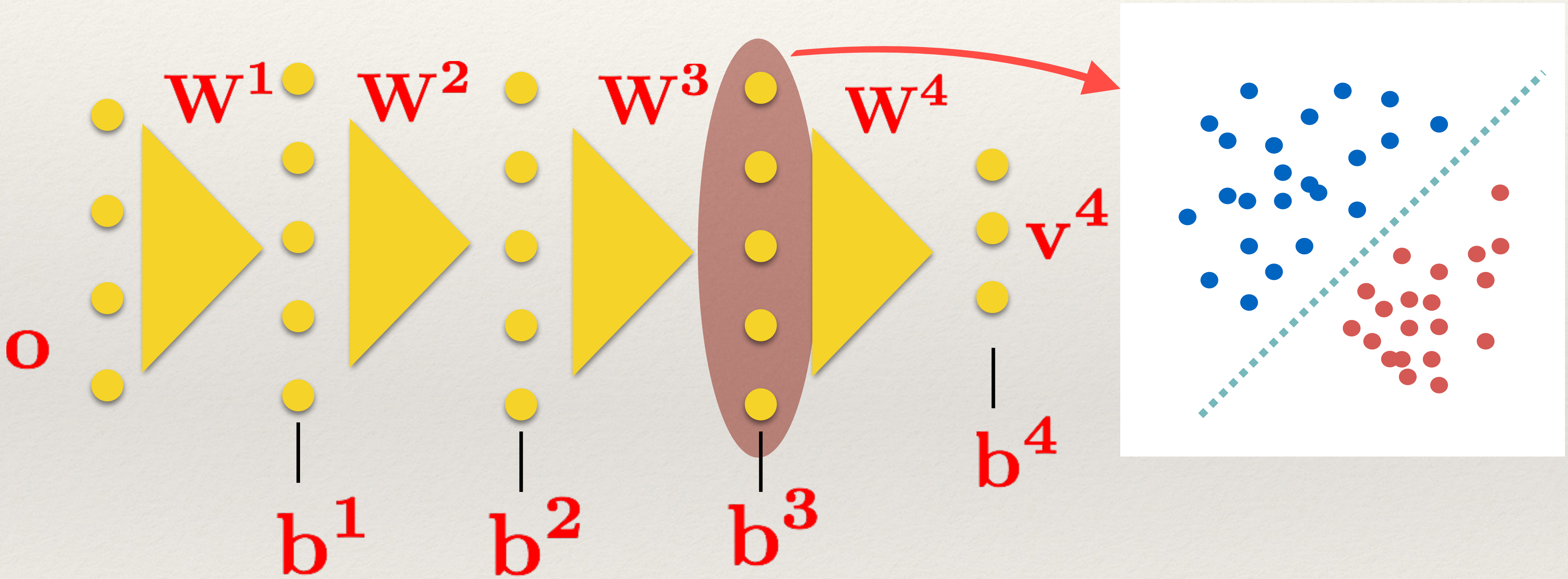
Architecture Design in NN

Neural networks with multiple hidden layers - Deep networks



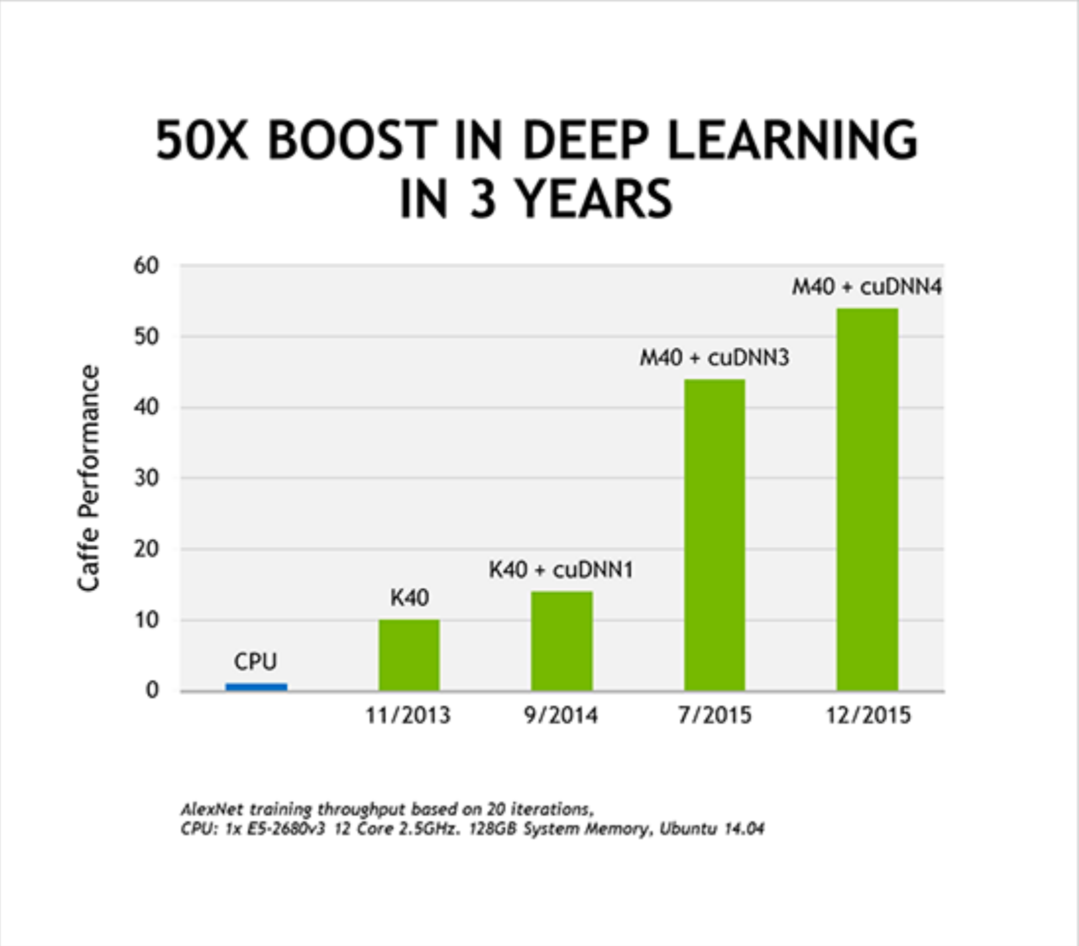
Architecture Design in NN

Neural networks with multiple hidden layers - Deep networks



Deep networks perform hierarchical data abstractions which enable the non-linear separation of complex data samples.

Feasibility of Deep Networks



- Are these networks trainable ?

- Advances in computation and processing
- Graphical processing units (GPUs) performing multiple parallel multiply accumulate operations.
- Large amounts of supervised data sets