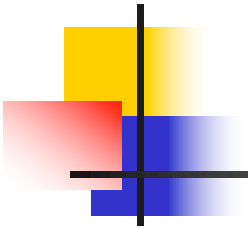
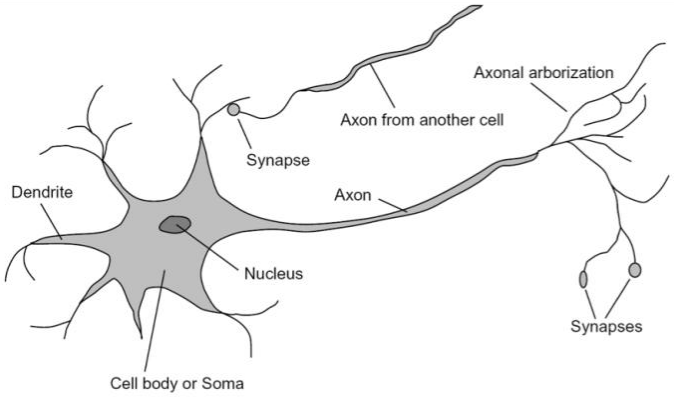


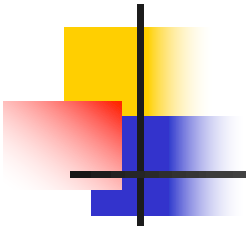
Machine Learning

Artificial Neural Networks

Artificial Neural Networks



- Artificial Neural Networks are inspired by neural networks in the brain
  - Large number of simple, parallel processing units
  - Signal transmission between neurons is adjustable
  - Complex functions formed by networks of units



Artificial Neural Networks

- Artificial Neural Networks form a family of representations that can be used for a range of learning tasks
  - A unit in an artificial neural network usually computes a simple function of the input x

$$h_\theta(x) = g(\theta^T x + b)$$

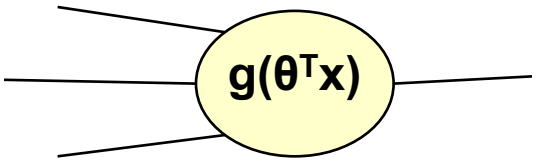
    - Using a bias weight and an additional input that is always

$$h_\theta(x) = g(\theta^T x)$$

Perceptrons

- A perceptron is one of the simplest ANNs
  - A perceptron consists of a single unit with a threshold function at its output

$$g(a) = \begin{cases} 1 & if\ a > 0 \\ -1 & otherwise \end{cases}$$

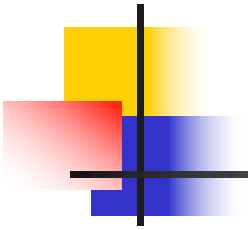


  - Perceptrons are classifiers
  - How can we learn the parameters of a perceptron ?

$$h_\theta(x) = g(\theta^T x)$$

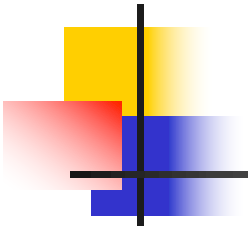
  - How to get the output h closer to the desired output y ?

$$\theta_j := \theta_j + \alpha(y - h_\theta(x))x_i$$



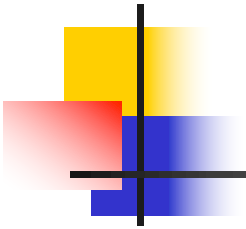
Perceptrons

- This looks like logistic regression
  - What are the differences ?
  - Since outputs are strictly 1 or 0, algorithm does not reduce step size near the minimum
    - It does not minimize the squared error
    - Solution can be brittle (barely separating)
  - If data is not linearly separable weights might thrash
    - No guaranteed convergence in this case



Perceptrons

- Perceptrons can only represent linearly separable classes
  - The same applies to logistic regression
- This can be addressed in two ways
  - Use nonlinear separation function
  - Use nonlinear features of the input
    - If we can learn these we can form more general function
- Building networks of neurons can be seen as doing either of these



Multi-Layer Networks

- Training networks of perceptrons is not systematically possible
  - No way to apply perceptron learning rule to hidden units
    - No available target value for hidden units
- Optimizing nonlinear function in the network requires a differentiable performance metric
  - E.g.: Least Squares Error
  - Differentiable output function *g(a)*

Multi-Layer Networks

- Many nonlinear, differentiable output functions can be used
  - The most common are sigmoid-type functions
    - Sigmoid: $g(a) = \frac{1}{1+e^{-a}}$, $\frac{\partial}{\partial a} g(a) = g(a)(1-g(a))$
    - Hyperbolic tangent: $g(a) = \tanh(a)$, $\frac{\partial}{\partial a} g(a) = 1 - g(a)^2$
  - A single unit with a sigmoid and least squares is equivalent to logistic regression on the original input features

Multi-Layer Networks

- The output of a multilayer sigmoid function network is a nonlinear function of the input

$$h_{j,\theta}(x) = g\left(\sum_i \theta_{i,j} h_{i,\theta}(x)\right)$$

  - Performance metric is often Least Squares defined over the network output $h_{o,\theta}(x)$

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_{o,\theta}(x^{(i)}) - y^{(i)}\right)^2$$

  - Minimizing this error results in a nonlinear regression solution
    - Multilayer ANNs are a representation for nonlinear regression

Error Backpropagation

- Using the error we can update weights of the output units using stochastic gradient descent

$$\frac{\partial}{\partial \theta_{i,o}} E[\theta] = \frac{\partial}{\partial \theta_{i,o}} \sum_{k \in O} \frac{1}{2} (y_k - h_{k,\theta}(x))^2 = (y_o - h_{o,\theta}(x)) \frac{\partial}{\partial \theta_{i,o}} (y_o - h_{o,\theta}(x))$$

$$= -(y_o - h_{o,\theta}(x)) \frac{\partial}{\partial \theta_{i,o}} h_{o,\theta}(x)$$

$$= -(y_o - h_{o,\theta}(x)) \frac{\partial}{\partial \theta_{i,o}} g\left(\sum_k \theta_{k,o} h_{k,\theta}(x)\right)$$

$$= -(y_o - h_{o,\theta}(x)) g\left(\sum_k \theta_{k,o} h_{k,\theta}(x)\right)\left(1 - g\left(\sum_k \theta_{k,o} h_{k,\theta}(x)\right)\right) \frac{\partial}{\partial \theta_{i,o}} \sum_k \theta_{k,o} h_{k,\theta}(x)$$

$$= -(y_o - h_{o,\theta}(x)) h_{o,\theta}(x)\left(1 - h_{o,\theta}(x)\right) h_{i,\theta}(x)$$

- How about the weights of the other units ?

Error Backpropagation

- We can form the derivative with respect to other weights
  - To update weights of the other units we need to compute an error estimate for them
    - We can break them into an "output error gradient" and a "unit parameter gradient"

$$\frac{\partial}{\partial \theta_{i,j}} E[\theta] = \frac{\partial E[\theta]}{\partial h_{j,\theta}(x)} \frac{\partial h_{j,\theta}(x)}{\partial \theta_{i,j}} = \Delta_j \frac{\partial h_{j,\theta}(x)}{\partial \theta_{i,j}}$$

    - "Unit parameter gradient":

$$\frac{\partial}{\partial \theta_{i,j}} h_{j,\theta}(x) = h_{j,\theta}(x)\left(1 - h_{j,\theta}(x)\right) h_{i,\theta}(x)$$

Error Backpropagation

- "Output error gradient":

$$\Delta_j = \frac{\partial E[\theta]}{\partial h_{j,\theta}(x)} = \frac{\partial}{\partial h_{j,\theta}(x)} \frac{1}{2} \sum_{o \in O} (y_o - h_{o,\theta}(x))^2$$

$$= \frac{1}{2} \sum_{o \in O} \frac{\partial}{\partial h_{j,\theta}(x)} (y_o - h_{o,\theta}(x))^2$$

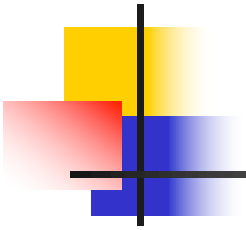
$$= \sum_{o \in O} \frac{\partial \frac{1}{2} (y_o - h_{o,\theta}(x))^2}{\partial h_{o,\theta}(x)} \frac{\partial h_{o,\theta}(x)}{\partial h_{j,\theta}(x)} = \sum_{o \in O} \Delta_o \frac{\partial h_{o,\theta}(x)}{\partial h_{j,\theta}(x)}$$

- This becomes recursive

$$\Delta_o = -(y_o - h_{o,\theta}(x))$$

$$\Delta_{j \in pred(O)} = \sum_{o \in O} \Delta_o \frac{\partial}{\partial h_{j,\theta}(x)} h_{o,\theta}(x) = \sum_{o \in O} \Delta_o h_{o,\theta}(x)(1 - h_{o,\theta}(x))\theta_{j,o}$$

$$\Delta_j = \sum_{k \in desc(j)} \Delta_k \frac{\partial}{\partial h_{j,\theta}(x)} h_{k,\theta}(x) = \sum_{k \in desc(j)} \Delta_k h_{k,\theta}(x)(1 - h_{k,\theta}(x))\theta_{j,k}$$



Error Backpropagation

- Error backpropagation provides an effective way to train multi-layer networks
  - Converges to a local minimum
  - Can represent very complex functions
    - Can arbitrarily precisely approximate any boolean function using one hidden layer
    - Can arbitrarily precisely approximate any bounded continuous function with one hidden layer and any function with two hidden layers
  - Backpropagation computes the network Jacobian

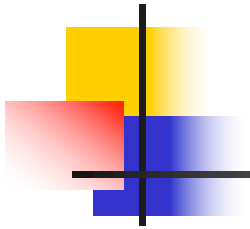
Weight Regularization

- Overfitting in multi-layer networks arises due to too many hidden units and parameters
  - Needs a lot of data or regularization
  - Regularization limits the weights in the network
    - L2 regularization:

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{n}\sum_{o\in O}\left(h_{o,\theta}(x^{(i)}) - y^{(i)}\right)^2 + \gamma\theta^T\theta$$

  - Alternatives are weight sharing (e.g. Convolutional Networks) or using early stopping
    - Regularization is more systematic than early stopping

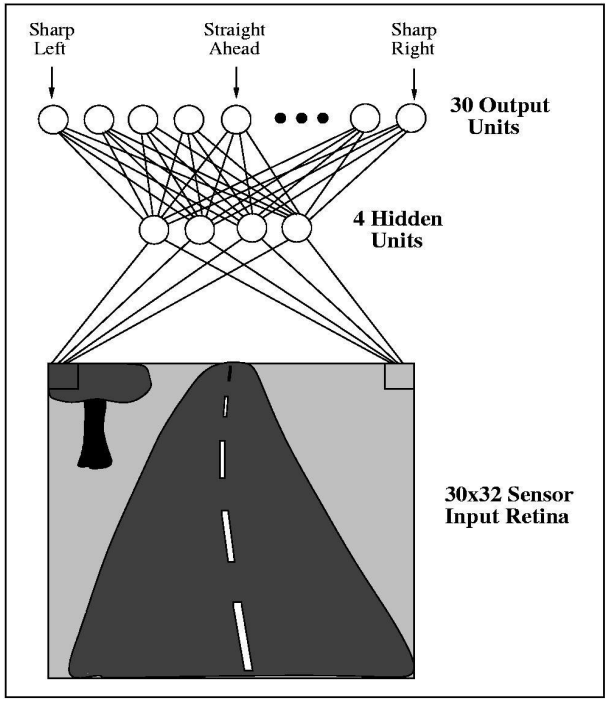
Feed-Forward Sigmoid Networks

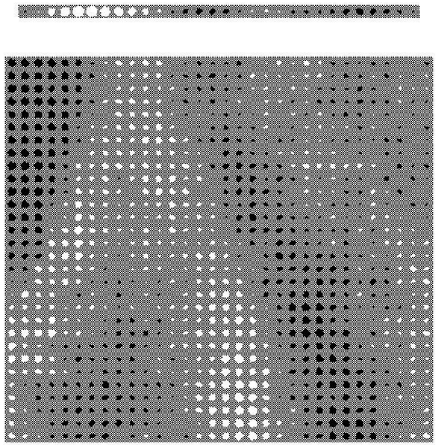


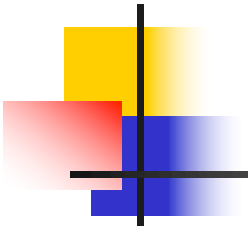
- Backpropagation networks have been used for a wide range of classification and regression problems
  - Choice of hidden layer and output representations is very important for learning performance
    - Choice of one unit to represent multiple classes through multiple values is very inefficient
    - Using a single continuous output for a very non-linear continuous output can be problematic
    - Use of "population coding" can be more efficient
      - Multiple output units are combined into the desired output

Feed-Forward Sigmoid Networks

- ALVINN example (1989)





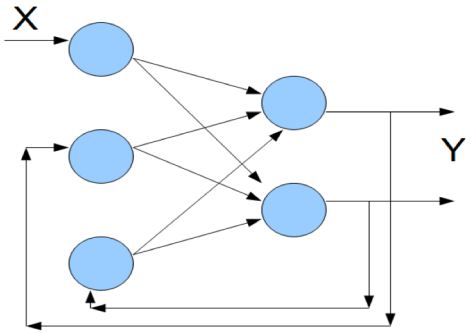


Neural Networks

- Sigmoid function feed-forward networks are not the only type of neural network
  - They are the most frequently used
- Recurrent neural networks
  - Provide temporal characteristics/memory
- Radial Basis function networks
  - Use radial basis functions as nonlinear output
    - Provide local features/properties

Recurrent Neural Networks

- Recurrent neural networks contain loops
  - Output of a unit influences its own input



  - Hidden units become temporal memory
  - Backpropagation runs into cycles
    - No analytic gradient/Jacobian through error backpropagation

Backpropagation Through Time (BPTT)

- To train recurrent neural networks we have to deal with the loops
  - The same input can "traverse" the same weight multiple times, adding to the overall gradient
  - BPTT "unrolls the network a fixed number of times and then performs Backpropagation
    - Full recurrence is not doable so a fixed number of recurrences are chosen
    - Weights are updated based on all the gradients computed for the weight

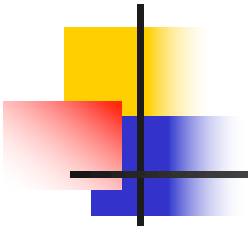
Radial Basis Function (RBF)Networks

- Besides sigmoid units, other nonlinear output functions can be used
  - Radial basis functions provide local "features" as opposed to the global sigmoid ones
    - RBF units compute and output as a local function of the distance to a unit-specific mean

$$h_\theta(x) = \phi\left(\left|x - \mu^{(j)}\right|\right)$$

    - Typical RBF functions:
      - Gaussian $\phi(d) = e^{-\frac{d^2}{2\sigma^2}}$
      - Multi-Quadratic $\phi(d) = \sqrt{d^2 + \sigma^2}$

Radial Basis Function (RBF)Networks



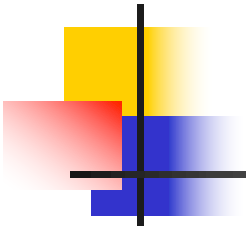
- RBF networks usually consist of an RBF layer followed by a linear or sigmoid layer
  - RBF layer represents local features
  - Linear or sigmoid layer is for regression/classification
- RBF units have parameters in terms of mean and standard deviation of the local function
  - Units are not linear in terms of the parameters

Radial Basis Function (RBF)Networks

- RBF networks can be trained in different ways
  - Introduce one RBF unit for each data point
    - Non-parametric representation
  - Introduce a fixed number of RBF units and initialize them using unsupervised learning
    - E.g. K-Means clustering
  - Learn all the parameters using gradient descent

$$h_{o,\theta,\mu,\sigma}(x) = \sum_j \theta_{j,o} \phi_j \left(\left| x - \mu^{(j)} \right| \right)$$

    - Large number of parameters lead to longer learning times and higher data requirements

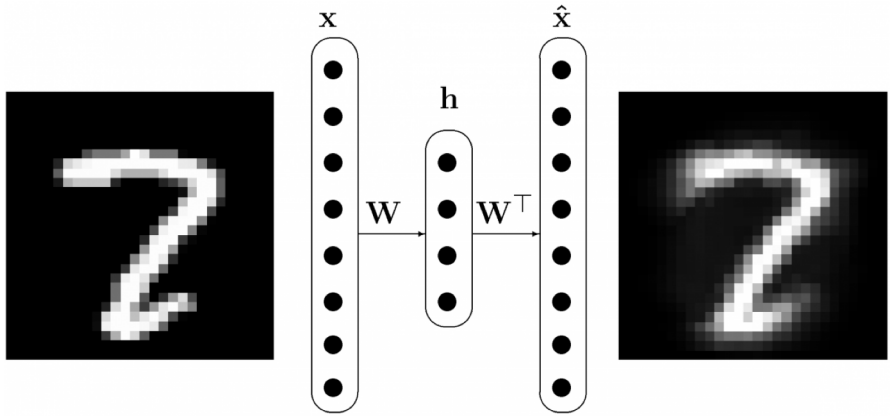


Neural Networks

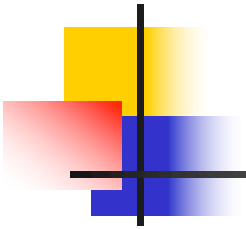
- Neural networks can not only be used for supervised learning
- Unsupervised learning networks
  - Autoencoder networks
  - Self-organizing Maps
- Associative memory
  - Hopfield networks

Autoencoder Networks

- Autoencoder networks are intended to learn efficient representations for data
  - Training tries to produce output that is equal to the input



(a) Input (b) Neural Encoding (c) Reconstruction

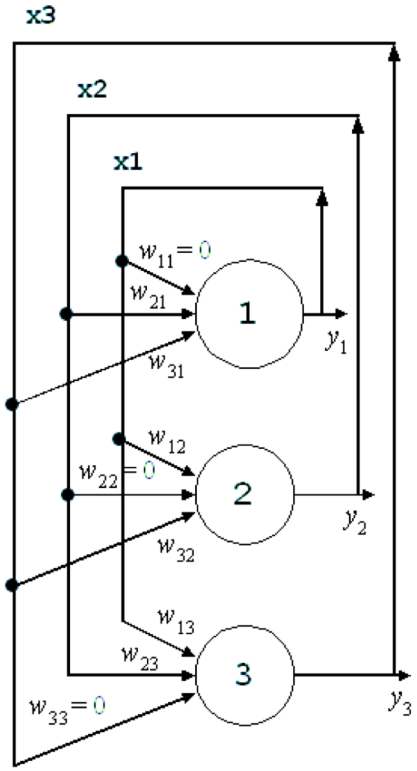


Autoencoder Networks

- Autoencoder learning can use two different types of "decoder" weights
  - Decoder weights are "reflected", coupled versions of the "encoder weights
  - Decoder weights are learned as separate, independent weights
- In both cases the main learning result is the "encoder" part of the network
  - Learns a new feature representation for the data that looses minimal information

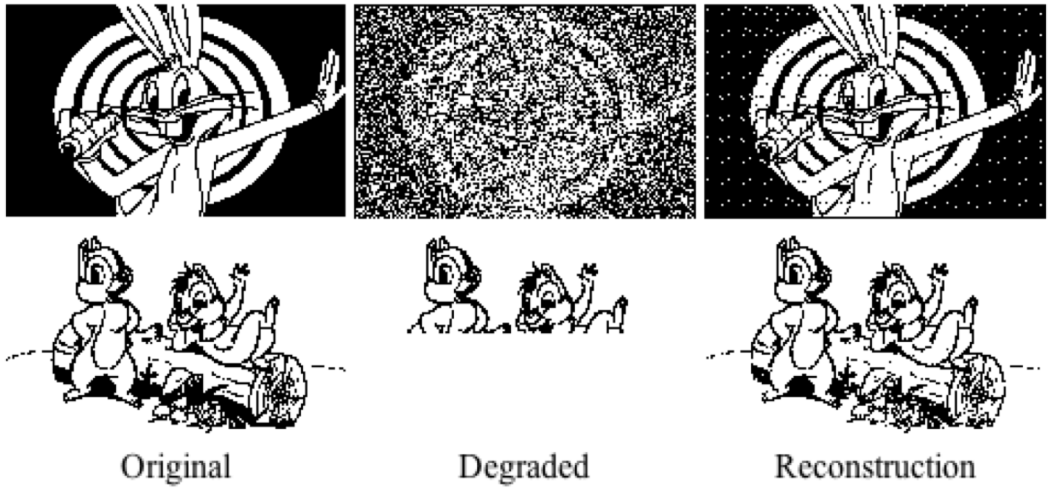
Associative Memory Networks

- Associative memory is intended to retrieve memorized items from incomplete queries
  - Automatic completion/retrieval
  - Used also to filter noise
- Hopfield networks are a very simple version of associative memory networks
  - Fully connected recursive with self-loop weights of 0



Hopfield Networks

- Hopfield networks can be thought of as networks that attempt to minimize an energy
  - Minimum energy corresponds to a perfect reconstruction

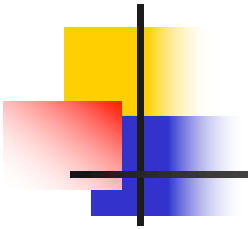


Hopfield Networks

- Hopfield networks have limited capacity
  - If patterns are not orthogonal it can result in imperfect reconstruction or reconstruction of mixtures of patterns
- Training can be done in different ways
  - Hebbian Learning

$$\theta_{i,j} = \frac{1}{n} \sum_k x_i^{(k)} x_j^{(k)}$$

  - Incremental Storkey learning
    - Results in higher capacity but longer learning times



Neural Networks

- Neural networks are mainly a structure to encode complex functions systematically
  - Feed forward networks are the most common and can be used for non-linear regression
    - Hidden units learn feature representation for the final regression units
  - Recurrent neural networks allow temporal functions to be encoded
  - Neural networks can also be used as a tool for other learning (e.g. unsupervised)