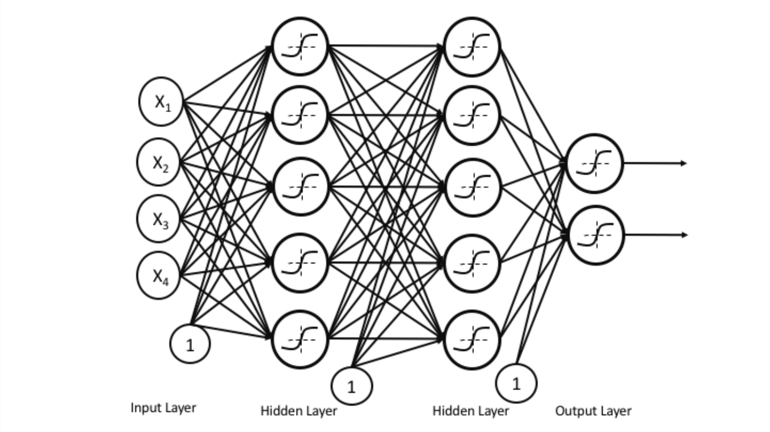
MULTI LAYER PERCEPTRON

Dr. Hilman F. Pardede

Research Center for Informatics
Indonesian Institute of Sciences

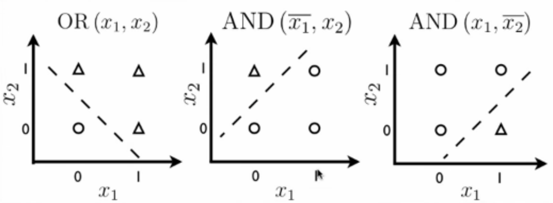
MULTI LAYER PERCEPTRON

Multiple layers of stacked neuron

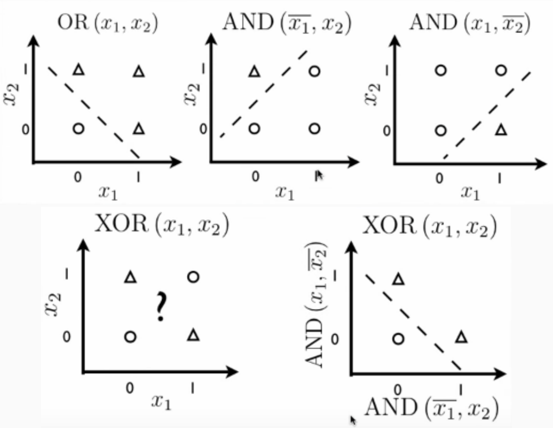


- Composed of several Perceptron-like units arranged in multiple layers
- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers compute a nonlinear transform of the inputs
- Also called a Feedforward Neural Network
- "Feedforward": no backward connections between layers (no loops)
- Note: All nodes between layers are assumed connected with each other

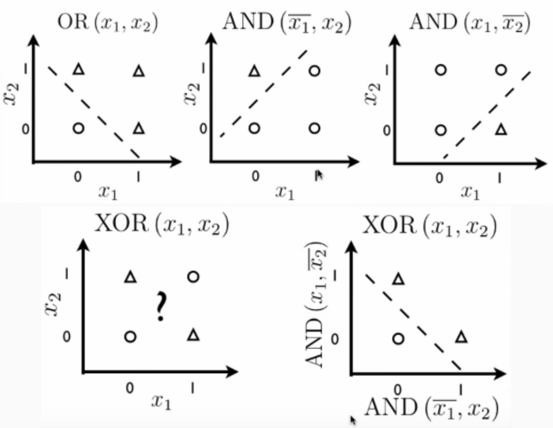
- Compositional features



- Compositional features

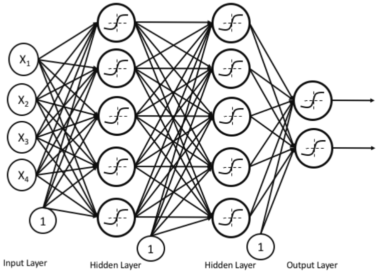


- Compositional features



- Universal Function Approximator (Hornik, 1991): A one hidden layer FFNN with sufficiently large number of hidden nodes can approximate any function

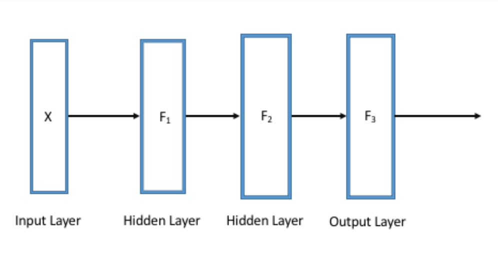
- Each layer creates a new representation of the input data:
- $h^{(0)} = f^{(0)}(\mathbf{x})$
- $h^{(1)} = f^{(1)}(\mathbf{h^{(0)}})$
- $y = f^{(2)}(\mathbf{h^{(1)}})$



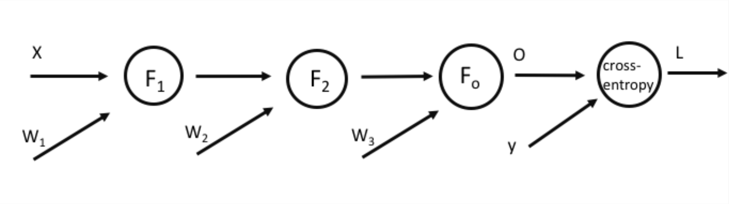
- Overall MLP is a function f
- $y = f(x, \theta)$
- Nested functions: $f^{(3)}(f^{(2)}(f^{(1)}(x)))$
  - First layer: $f^{(1)}$
  - Second layer: $f^{(2)}$
  - Third layer: $f^{(3)}$

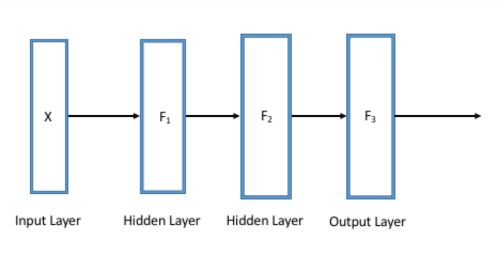
It is just gradient descent in rule chain

- Model:
  - $o_\theta = \phi_1(\mathbf{w_1}^\top \phi_2(\mathbf{w_2}^\top \phi(\mathbf{w_3}^\top x)))$
  - $\theta : \{\mathbf{W}\}$
- Loss function:
  - $L(\mathbf{x}, y; \mathbf{W}) = \frac{1}{2n} \sum_{i=0}^{n} (o_\theta - y)^2$
- Gradient of L wrt $\mathbf{W}$:
  - $\frac{\partial}{\partial W} L(.)$

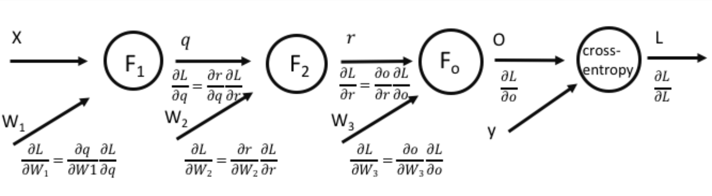


Graph representation





Graph representation



- For nonlinear function, features x are transformed by ϕ, a nonlinear function, before applying to linear model

$$y = \boldsymbol{\theta}^T \phi(x) + b \tag{1}$$

  - In the transformed domain, the transformed features could be linearly separated

- SVM applies kernel trick due to the fact that "very complex" feature transformation equals to using much more simple kernel functions

$$y = b + \sum_i \alpha_i k(x, x^i) \tag{2}$$

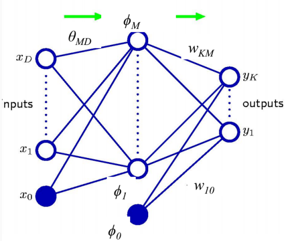
where $k(x, x^i) = \phi(x)\phi(x^i)$

APPROACHES TO FIND ϕ

- Generic functions
  - Ex. Radial Basis Function (RBF), Polynomial, Hyperbolic tangent, etc.
- Manually Engineered
  - Dominant approach especially for natural signals such as speech, video, image, etc
  - Requires decades of effort
  - Laborious, non-transferable between domains
- Learn $\phi \rightarrow$ Principle of deep learning

Approach 3. Feature Learning

- We have a model $y = f(\mathbf{x}; \boldsymbol{\theta}, \mathbf{w}) = \phi(\mathbf{x}, \boldsymbol{\theta})^T \mathbf{w}$
  - Parameters $\boldsymbol{\theta}$ to learn ϕ from a broad class of function
  - parameters $\mathbf{w}$ that map $\phi(\mathbf{x})$ to the desired output
  - ϕ defining the hidden layer
- The solution gives **convex optimization** but its benefits outweigh harms



- K outputs y_1,y_2,...,y_K for a given input x. Hidden layer consists of M units

$$y = f(\mathbf{x}; \boldsymbol{\theta}, \mathbf{w}) = \phi(\mathbf{x}, \boldsymbol{\theta})^T \mathbf{w}$$

$$y(\mathbf{x}; \boldsymbol{\theta}, \mathbf{w}) = \sum_{j=1}^{M} w_{kj}\phi_j \left(\sum_{i=1}^{D} \theta_{ji}x_i + \theta_{j0} \right) + w_{k0}$$

can be seen as a generalization of linear model

- Nonlinear function f_k with $M + 1$ parameters $w_k = (w_{k0}, ..., w_{kM})$
- M basis functions, ϕ_j $j = 1, ...M$ each with D parameters $\theta_j = (\theta_{j1}, ..., \theta_{jD})$
- Both w_k andθ_j are learnt from data

- Parameterize the basis function as $\phi(\mathbf{x};\boldsymbol{\theta})$
  - Use optimization to find θ that corresponds to a good representation
- Approach can capture benefit of first approach (fixed basis functions) by being highly generic
  - By using a broad family for $\phi(\mathbf{x};\boldsymbol{\theta})$
- Can also capture benefits of second approach
  - Need only find right function family rather than precise right function

Neural networks with many new layers are nothing new

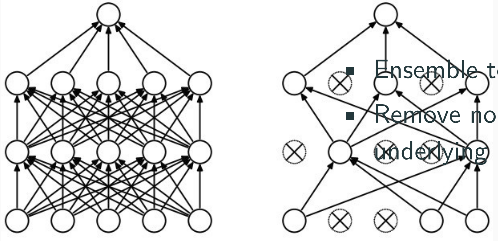
- NN with many hidden layers are difficult to train.
- too many parameters to be optimized.

- Capability of computing powers
- Techniques for training the network

- When training with large models, we often observe that the training error decreses steadly over time, but the validation set error begins to rise again.
- We can obtain a model with better validation error (hopefully this corresponds to better test error) by returning to set of parameters with lowest validation error.
- When the validation set error decreases, we store the parameters and when training stops, we return to parameters that produce the lowest validation errors rather than the last ones.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.

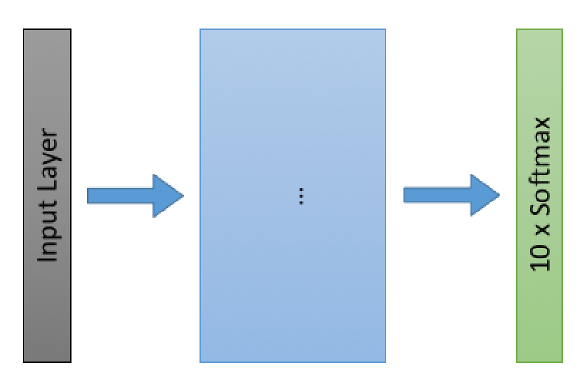
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*



- Ensemble techniques
- Remove non-output units from an underlying base network

Model Design

- Input format
- Output layer
- Loss function(s)
- Model Architecture
- Optimization parameters

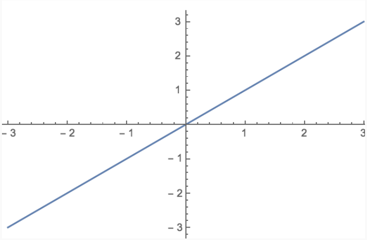


Linear Activation

$$g(z) = z \tag{3}$$

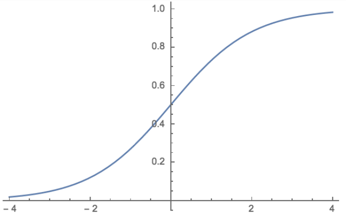
$$h = g(W^\top x + b) \tag{4}$$

- Usually used as a last layer activation for doing regression
- If all neurons are linear, the MLP is linear, which limits the generalization



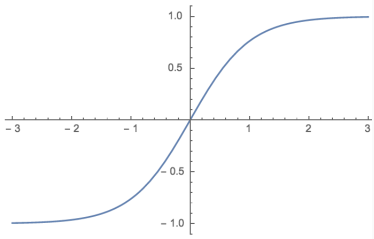
Logistic Sigmoid

- $\phi(z) = \frac{1}{1+e^{-z}}$
- $h = \phi(W^{\top}x + b)$



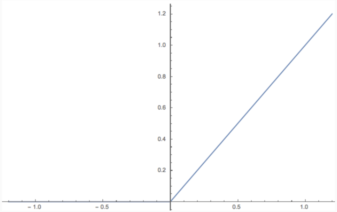
Hyperbolic Tangent

- $\phi(z) = \tanh(z)$
- $h = \phi(W^\top x + b)$



Hyperbolic Tangent

- $\phi(z) = max\{0, z\}$
- $h = \phi(W^\top x + b)$



- Learn in Batches
- Reduce learning rate when it plateaus
  - Learning rate adaptation: AdaGrad, RMSProp, Adam, etc

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
end while

Greedy layer-wise unsupervised learning

- Used as initialization for supervised learning
- Procedure
  - Train each layer of feature greedily unsupervised
  - add supervised classifier on the top
  - optimize entire network with back-propagation