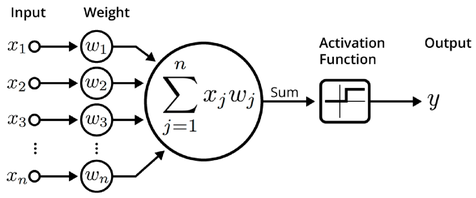
Neural Networks

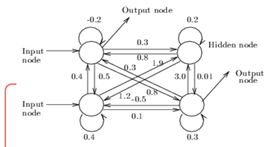


Created by
Omar Alkhateeb

For more MindMaps
Check our Blitz Website
https://blitz.deveopix.com/

Introduction

- **what is artificial neural network?** — it consists of
  - model
  - non-linear — output is non linear combination of inputs
  - input is numeric
  - output is numeric
  - pre & post processing are completed separate from the model
- **Training of NNs**
  - initialize weights for all neurons
  - present input layer
  - calculate outputs
  - compare outputs
  - update weights to attempt a match
  - repeat until all examples presented
- **components of NN**
  - nodes
    - Each node has one or more inputs from other nodes
    - and one output to other nodes
    - Input/output values can be
      - Binary {0, 1}
      - Bipolar {-1, 1}
      - Continuous
  - All inputs to one node come in at the same time and remain activated until the output is produced
  - Weights associated with links
  - node function — examples
    - identity function
    - constant function
    - step function
    - ramp function
    - sigmoid function
    - gaussian function
- **Network Architecture**



Fully Connected Layer

- every node is connected to every other node
- connection may be positive or negative or irrelevant
- most general
- in symmetric fully connected nets — weights are symmetric
- its nodes are
  - **input** nodes — receive input from the environment
  - **output** nodes — send signals to the environment
  - **hidden** notes — no direct interaction to the environment

- **Layered Networks** — Nodes are partitioned into subsets — called **Layers**
- **Feedforward Networks**
  - A connection is allowed from a node in layer i — only to nodes in layer i + 1
  - Most widely used architecture
- **Acyclic Networks**
  - Connections do not form directed cycles
  - Multi-layered feedforward nets are acyclic
- **Recurrent Networks**
  - Nets with directed cycles
  - Much harder to analyze than acyclic nets
- **Modular nets**
  - Consists of several modules — each of which is itself a neural net — for a particular sub-problem
  - Sparse connections between modules

Perceptron Single Layer Feedforward Networks

- **Simple Perceptron** — Structure
  - Single output node with threshold function
  - n input nodes with weights wi — i = 1 – n
- **Linear models**
  - linear model in n-dimensional space
    - is define by n+1 weights
    - In two dimensions — $w1\,x1 + w2\,x2 + b = 0$ — (where b = -a)
    - in 3 dimensions — $w1\,x1 + w2\,x2 + w3\,x3 + b = 0$
    - in n dimensions — $b + \sum_{i=1}^{n} w_i x_i = 0$
  - **Learning** — The network must learn decision surfaces from a set of **training patterns** — so that these training patterns are classified correctly
  - **Generalization** — After training — the network must also be able to **generalize** — correctly classify test patterns it has never seen before
- Multilayer Perceptron — consists of 3 layers at least

Backpropagation

- **Training & Update Process**
  - **initial steps**
    - initial weights prediction training — training before backpropagation
    - backpropagation goal
      - updating NN weights
      - used when predictions are incorrect
  - **forward pass**
    - prepare activation function input (SOP) — sum of products (input * weights)
    - activation function output
      - uses SOP as input
      - sigmoid function — example
    - measure prediction error
      - squared error function
      - 1/2 * (desired - predicted)^2
  - **backward pass**
    - goal
      - minimize goal to acceptable value
    - parameter to change
      - weights
    - weight update equation — perceptron rule — W(new) = W(old) + n(d - Y)X
      - learning rate — n
      - desired output — d
      - predicted output — Y
      - current input — X
    - iterative process
      - updating weights
      - retraining network
      - calculating prediction error
      - continue until the error os acceptable
- **backpropagation process**
  - **purpose**
    - understand the effect of each weight on preediction error
    - know how changing weights changes error
  - **forward vs backward passes**
    - prediction calculation — forward
    - error contribution calculation — backward
  - decomposing the calculation of the partial derivatives of the error on each neuron — the error function is composed of several nested functions

$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial out_{out}} \times \frac{\partial out_{out}}{\partial out_{in}} \times \frac{\partial out_{in}}{\partial h1_{out}} \times \frac{\partial h1_{out}}{\partial h1_{in}} \times \frac{\partial h1_{in}}{\partial W_1}$$

we can chain through hidden layers if any exists

$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial Predicted} \times \frac{\partial Predicted}{\partial s} \times \frac{\partial s}{\partial W_1}$$

chain rule

  - **interpreting derivitives**
    - derivative sign
      - decrease weight to decrease error — positive
      - increase weight to decrease error — negative
    - deriative magnitude — Magnitude * P = Error change
    - update equation based on gradient decent — Wi(new) = Wi(old) - n*(dE/dWi)
- **NN with hidden layers**
  - initial setup
    - training data
      - X1
      - X2
      - output
    - initial weights
      - W1-Wn
      - b1-bn
  - forward pass calculations
    - Hidden Layer Neuron1 (h1)
      - input (h1 in)
      - using sigmoid — output (h1 out)
    - Hidden Layer Neuron2 (h2)
      - input (h2 in)
      - using sigmoid — output (h2 out)
    - Output Layer Neuron (out))
      - input from previous outputs
      - predicted output out
    - Prediction Error E (desired vs out)
  - calculate error on multiple neurons — using **chain rule** — backward pass
  - using **gradient descent** — updating weights

NN WorkFlow

- **Prediction Error** — 1/2 * (desired - predicted)^2 — we get prediction error — after the training process
- **Performance Measures** — we have certain metrics to consider — in classification problems
  - accuracy
  - sensitivity
  - specificity
- **Splitting Dataset**
  - **Training** Set
    - used to train the model
    - large portion of data
    - basis for model parameter estimation
    - must include all possible inputs
  - **Testing** Set
    - separate subset withheld during training
    - unbiased benchmark for evaluation
    - simulates real-world data
    - test ability to generalize
  - **Validation** Set
    - intermediate between training & test
    - fine tune hyperparameters — primary purpose
    - used iteratively during development
    - identifies overfitting/underfitting
- **hyperparameters**
  - which are parameters in the model that affects performance & training speed
  - examples in CNN
    - size of kernals
    - number of kernals
    - length of strides
    - pooling size
- **Overfitting & Underfitting**
  - **overfitting**
    - model too tuned to training set
    - learns training data too well
    - low accuracy — poor predictions for unseen data
    - high variance — model is too complex
  - **underfitting**
    - model not well-tuned to training set
    - doesn't capture input/output relation too well
    - even for training data — inaccurate prediction
    - high bias — model not too complex enough
  - **detection**
    - detecting overfitting
    - detecting underfitting
  - **goal**
    - overcome bias-variance tradeoff
  - **Cures**
    - overfitting
      - reduce model complexity
      - manual selection or algorithm — reduce number of input features
      - apply regularization
      - dropout layer
      - limit neural network weights
      - early stopping
      - using more training examples
    - underfitting
      - ensure model complexity
      - obtain more training data
      - data augmentation
      - increase epochs on training data
      - increase number of model parameters
      - introduce engineered features
      - reduce regularization

Presented with xmind