INTELLIGENT SYSTEMS LABORATORY

Lesson 10
Artificial Neural Networks

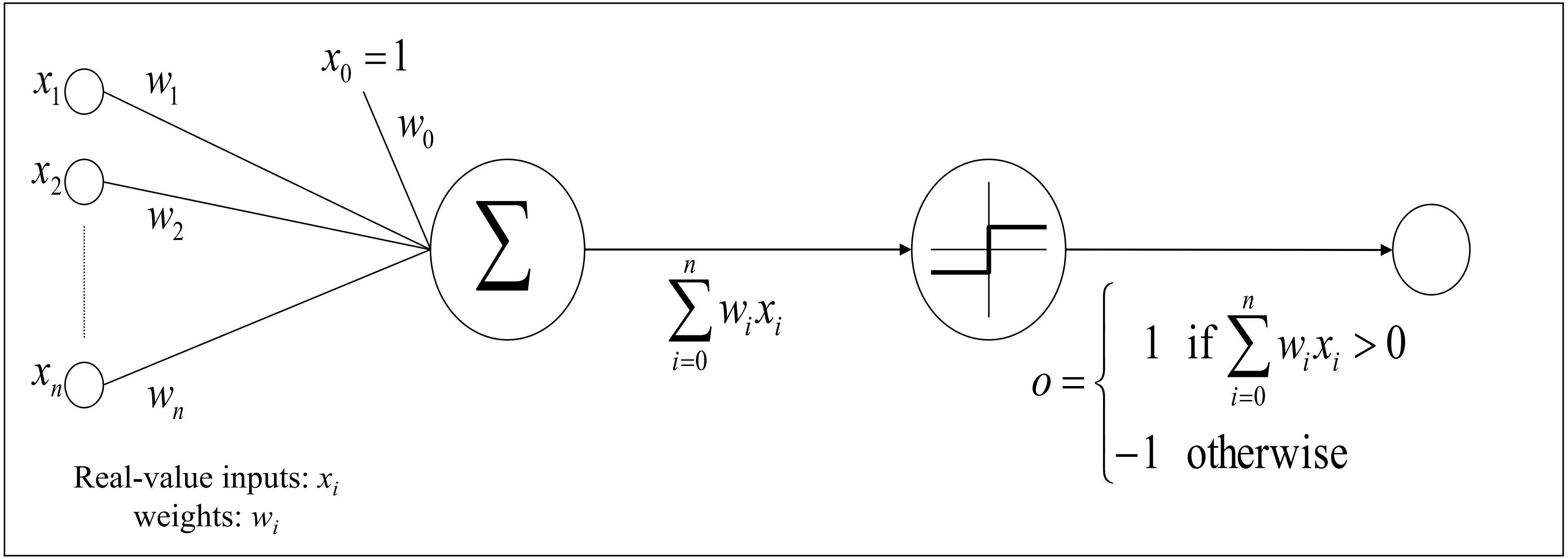
Mathematics and Statistics for Data Science

Tapanan Yeophantong
Vincent Mary School of Science and Technology
Assumption University

Content

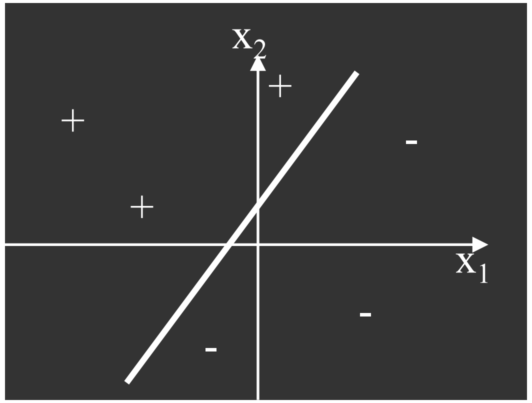
- Perceptrons
- Stochastic gradient descent algorithm
- Backpropagation & multi-layered networks
- Deep learning architectures & applications

Perceptrons



Representational Power

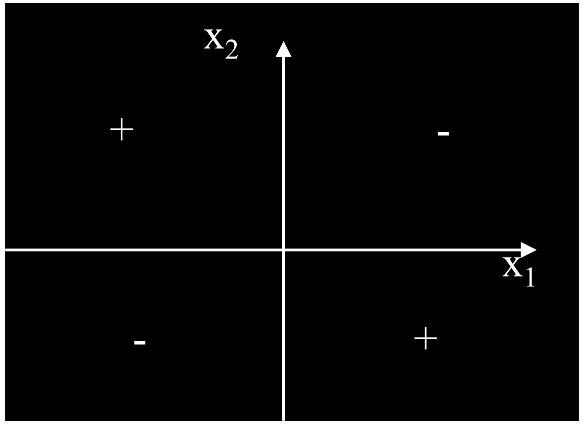
- Decision surfaces represented by a two-input perceptron



- Perceptron outputs a 1 for instances on one side of hyperplane and outputs a –1 for instances on other side.

Non-separability Problem

- Some sets of positive and negative examples cannot be separated by a perceptron.



- Those that can be separated by a hyperplane are linearly separable examples.

Multilayer Perceptrons

- Multilayer perceptron can compute all boolean functions.
- In fact, every Boolean function can be represented by a network of interconnected units.
- Only two-level depth is sufficient.

Training a Perceptron

- Learning the weights of a single perceptron.
- Produce correct +1 output given each training example
- Two algorithms:
  - Perceptron Rule
  - Delta Rule
- Guaranteed to converge to different hypotheses.

Perceptron Training Rule

- Begin with random weights.
- Iteratively apply the perceptron to each training example.
- Modify the weights whenever it misclassifies a sample.
- Iterate as many times as needed until all training samples are correctly classified.

Perceptron Training Rule

- Training involves changing weight wi associated with xi
  - Weight is revised by

$$w_i \leftarrow w_i + \Delta w_i$$

  - where $\Delta w_i = \eta(t - o)x_i$
  - Learning rate η is a positive constant that moderates the degree to which weights are changed at each step. It is set to some small value and sometimes decays as the number of iterations increases.
  - t is the target output.
  - o is the output generated by the perceptron.

Perceptron Rule Convergence

- Suppose training example is correctly classified
  - $(t - o) = 0$ so no weights updated
- But if perceptron outputs –1 when target output is +1
  - weights must be altered to increase value of w.x
  - for all i, if $x_i > 0$ (all attributes are positive), increasing w_i will bring perceptron closer to correctly classifying the example.
  - training rule increases w_i because $(t - o)$, η and x_i are all positive.

Perceptron Convergence

- Perceptron learning procedure is guarantee to converge provided that:
  - The training samples are linearly separable, and
  - A sufficiently small η is used.

Exercise 1

- Goal: Try a perceptron on scikit-learn.
- Instructions:
  - Create datasets corresponding to ADD, OR and XOR.
  - Train a perceptron with each dataset.
  - Check your results.

Content

- Perceptrons
- Stochastic gradient descent algorithm
- Backpropagation & multi-layered networks
- Deep learning architectures & applications

Delta Rule

- Perceptron rule fails to converge if examples are not linearly separable.
- Delta rule is designed to overcome this difficulty.
- If samples are not linearly separable, Delta rule converges toward a best-fit approximation of target concept.
- The key idea is to use gradient descent to search hypothesis space of possible weight vectors.

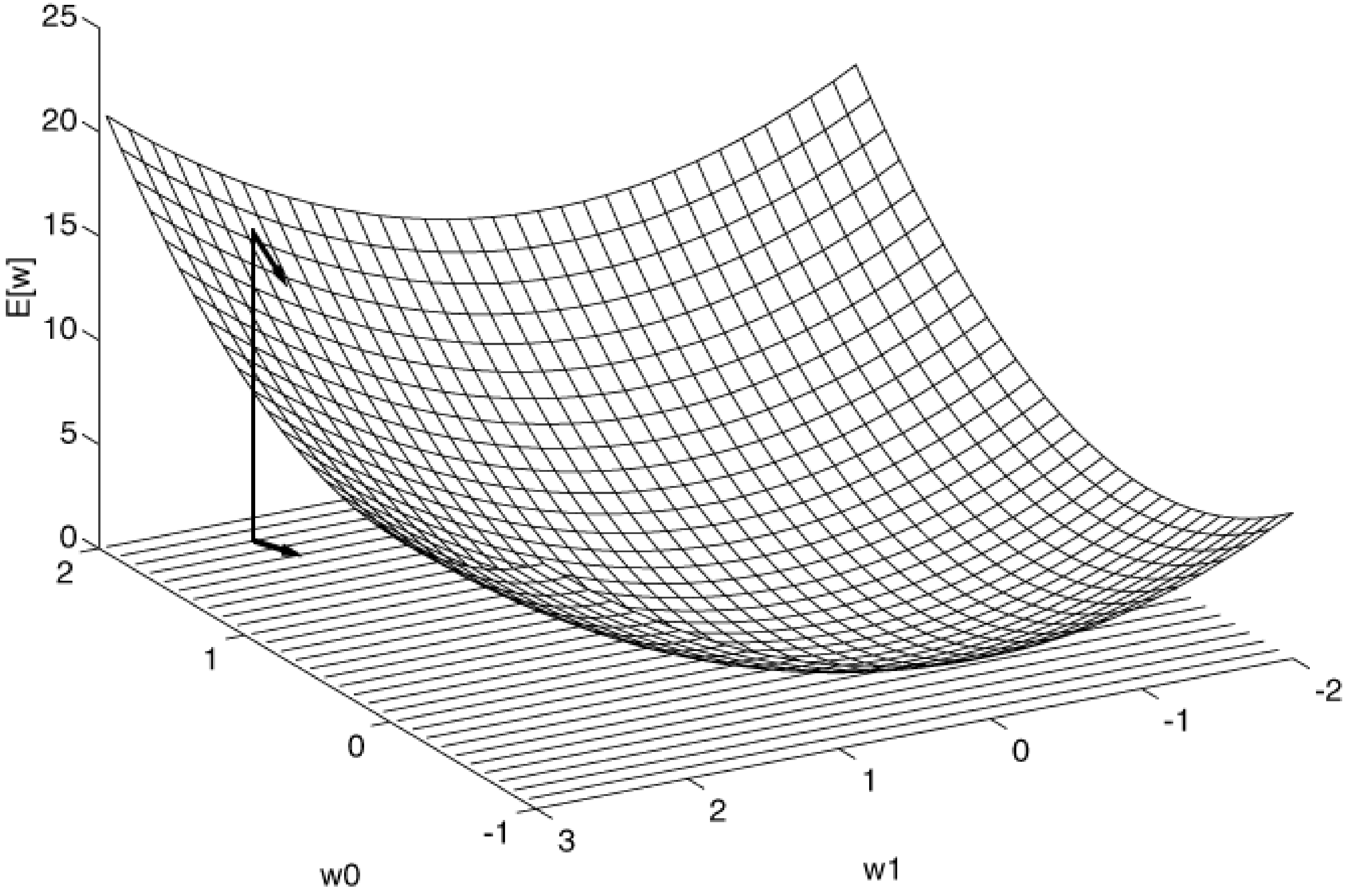
Overview of the Delta Rule

- Unthresholded perceptron: o(x) = w . x
- Training error of a hypothesis weight vectors:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

  - where D is the set of training samples
  - td is target output for training sample d
  - od is output for training sample d

Visualising Hypothesis Space



Gradient Descent

- Gradient descent determines a weight vector that minimises E
  - Starting with an arbitrary initial weight vector.
  - Repeatedly modifying it in small steps.
  - At each step, weight vector is modified in the direction that produces the steepest descent along the error surface.

Weight Update Rule

- Pick an initial random weight vector.
- Apply the linear unit to all training examples, then compute for each weight according to

$$\Delta w_i = \eta \sum_d (t_d - o_d) x_{id}$$

- Update each w_i by adding Δw_i then repeat process.

Gradient-Descent Algorithm

- Initialize each w_i to some small random value.
- Until the termination condition is met, do
  - Initialize each Δw_i to zero.
  - For each $\langle x, t \rangle$ in training_examples, do
    - Input the instance x to the unit and compute the output o
    - For each linear unit weight w_i, do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - For each linear unit weight w_i, do

$$w_i \leftarrow w_i + \Delta w_i$$

Learning Rate

- Converges to a weight vector with minimum error
  - Since error surface contains a single global minimum
  - Training examples are linearly separable
  - Given a sufficiently small learning rate η
- Large η runs the risk of overstepping the minimum in the error surface (i.e. not converging).
- We can also gradually reduce the value of η as the number of gradient descent steps grows.

Stochastic Gradient Descent

- Stochastic gradient descent updates w_i for each training sample, normally randomly selected.
- Update weight as each sample is processed

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

- Called Delta rule, LMS rule, Adaline rule, Widrow-Hoff rule.

SGD Algorithm

- Initialize each w_i to some small random value.
- Until the termination condition is met, do
  - Initialize each Δw_i to zero.
  - For each $\langle x, t \rangle$ in training examples, do
    - Input the instance x to the unit and compute the output o
    - For each linear unit weight w_i, do

$$w_i \leftarrow w_i + \eta(t - o)x_i$$

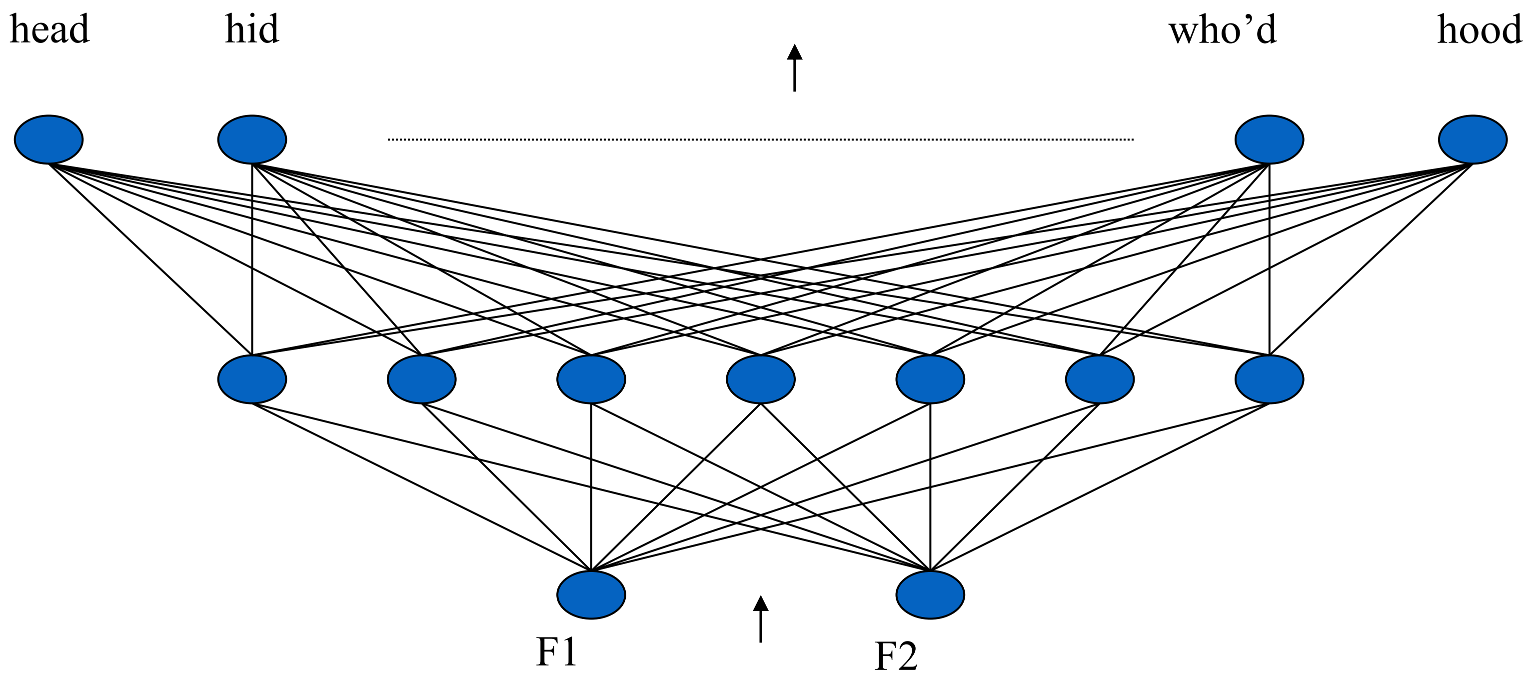
Exercise 2

- Goal: Try stochastic gradient on scikit-learn.
- Instructions:
  - Create datasets corresponding to ADD, OR and XOR.
  - Train linear_model.SGDClassifier with each dataset.
  - Check your results.
  - Extra: Try fitting a kernel before training (e.g. RBFSampler).

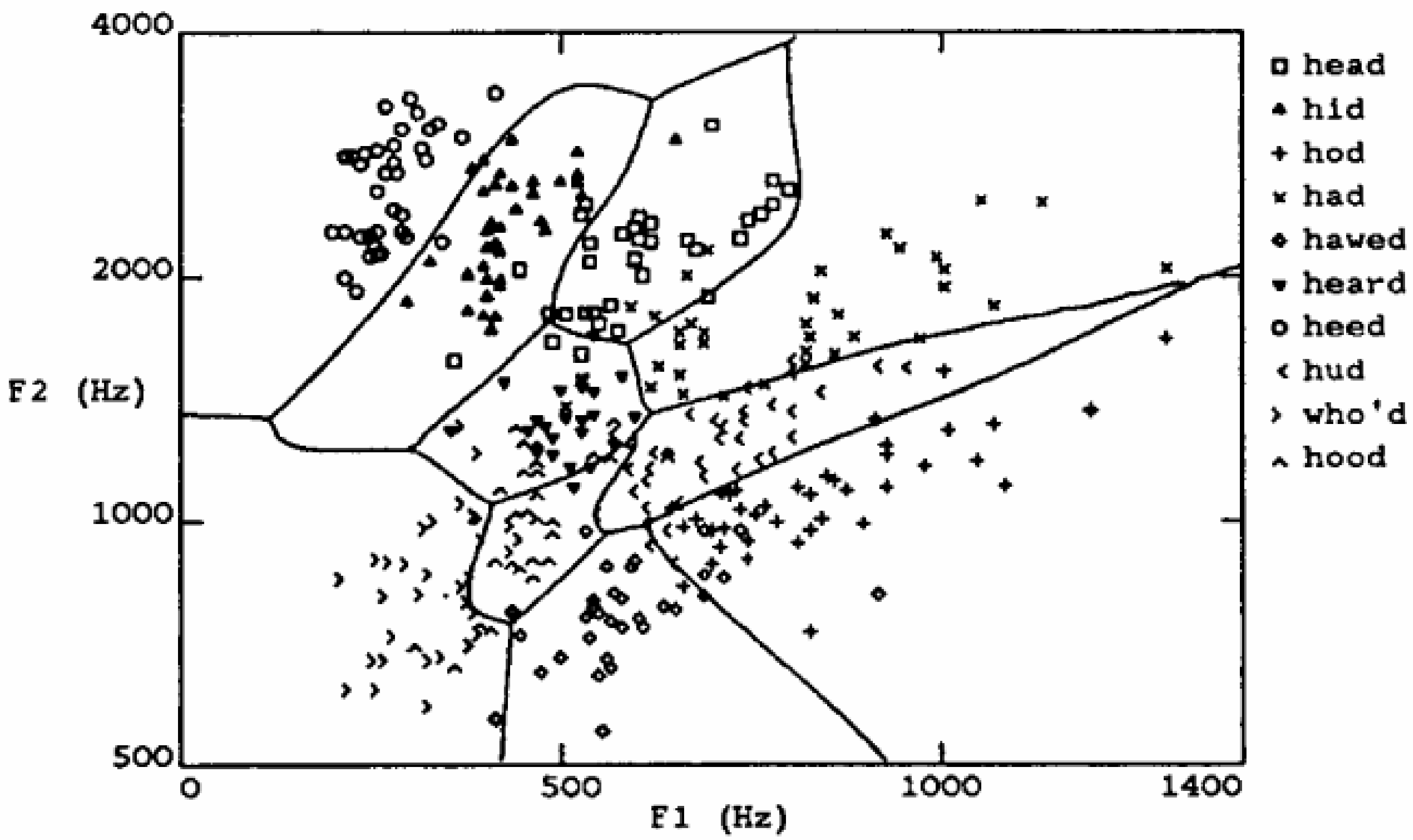
Content

- Perceptrons
- Stochastic gradient descent algorithm
- Backpropagation & multi-layered networks
- Deep learning architectures & applications

Multi-layered Neural Networks

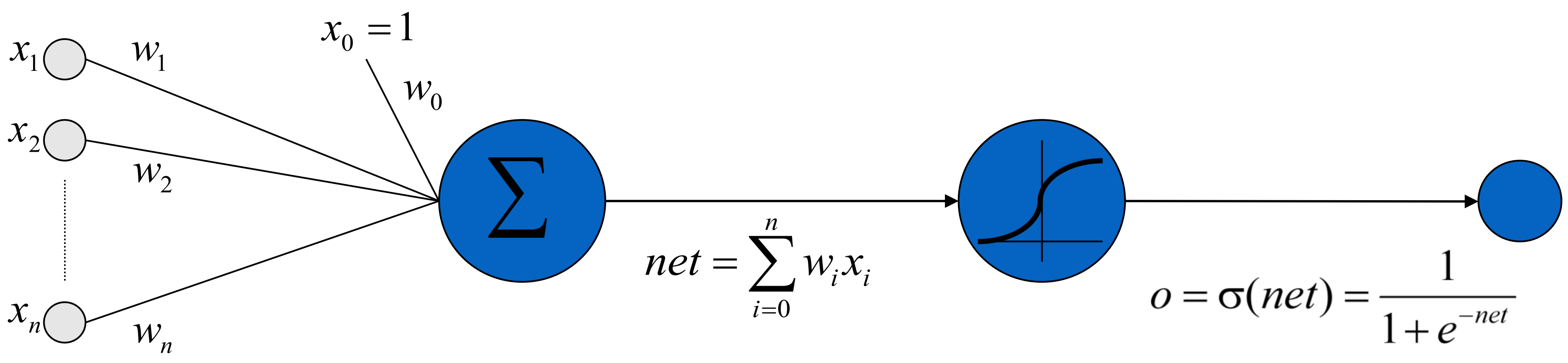


Decision Regions



Differentiable Threshold Unit

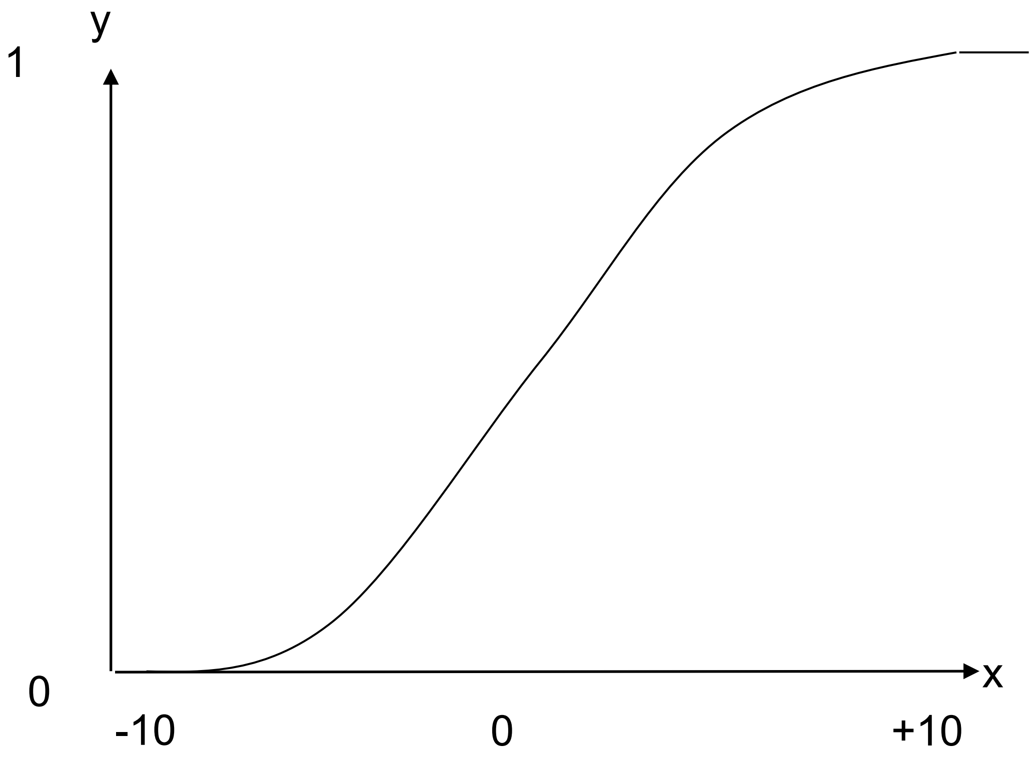
- Highly non-linear multiple layer functions require functions that output a non-linear function whose output is a differentiable function; for example, a sigmoid.



Sigmoid Function

- A logistic function:

  $y = 1/(1+e^{-x})$



Backpropagation Algorithm

- Until the termination condition is met, do
  - For each $\langle x, t \rangle$ in training_example, do
    - 1. Propagate the errors backwards through the network.
    - 2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

    - 3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k$$

    - 4. Update each network weight w_{ij} where

$$\Delta w_{ij} = \eta\delta_j x_{ji}$$

Momentum

- Most common variation of BACKPROPAGATION is one that adds "momentum" to Δw_{ji}.
- Weight update on n^{th} iteration depends upon update that occurred in $(n-1)^{th}$ iteration

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)$$

- where α is the momentum.

Multi-layered Networks

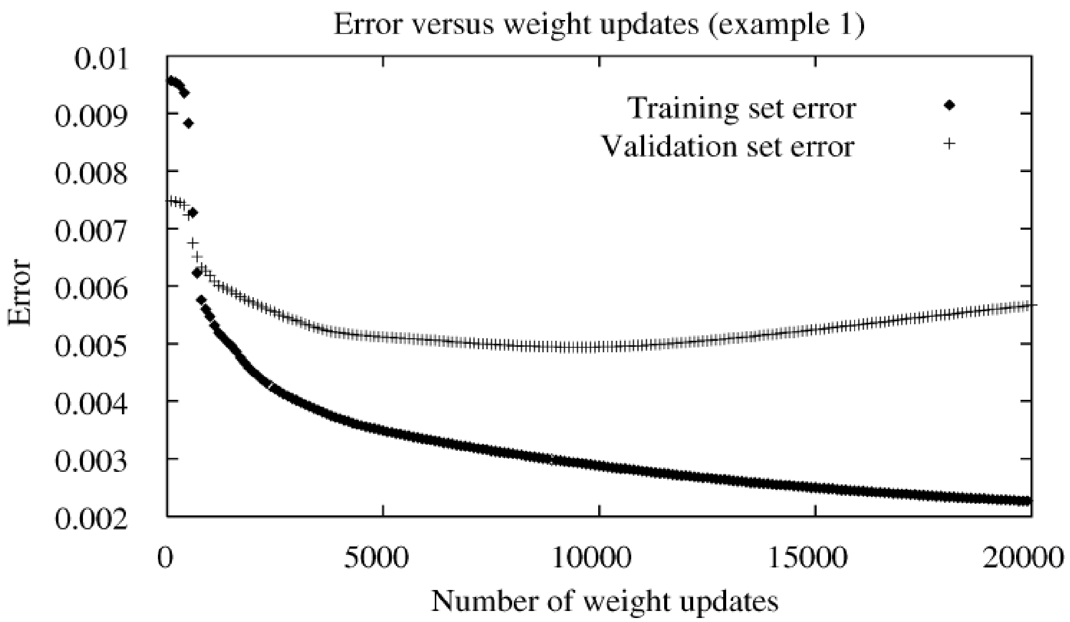
- Algorithm generalises to feedforward networks of arbitrary depth.
- For unit r in layer m, δ_r is computed from the values at next deeper layer m + 1

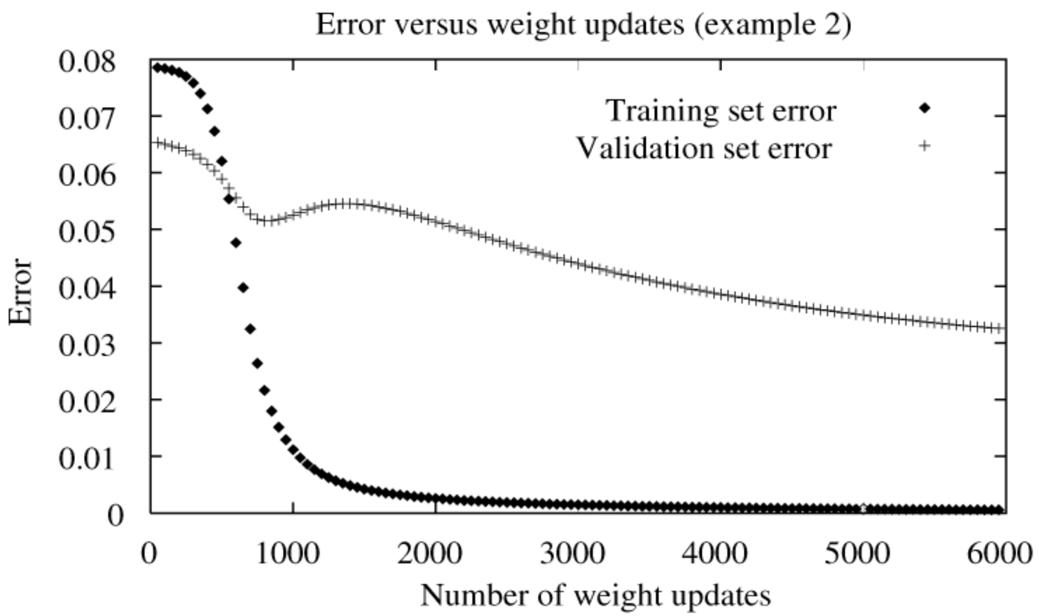
$$\delta_r \leftarrow o_r(1-o_r) \sum_{s \in layer\ m+1} w_{sr}\delta_s$$

When to Terminate?

- Termination condition for backpropagation is not specified.
- Possible choice:
  - Continue training until error E falls below a pre-determined threshold.
  - This is a poor strategy because it can overfit.

Generalisation vs Overfitting





Stopping Criteria

- Termination conditions:
  - Fixed number of iterations through loop.
  - Once error on training examples fall below a threshold.
  - Error on a separate validation set meets a criterion.
- Choice of termination criterion important:
  - Too few iterations can fail to reduce error sufficiently.
  - Too many can can lead to overfitting the training data.

Exercise 3

- Goal: Train an iris classifier using MLPClassifier.
- Instructions:
  - Load iris dataset.
  - Build an MLPClassifier for iris classification.
  - Evaluate the performance of your neural network.

Exercise 4

- Goal: Train digit classifier using MLPClassifier.
- Instructions:
  - Load digit dataset.
  - Build an MLPClassifier for handwritten digit recognition.
  - Evaluate the performance of your neural network.

Exercise 5

- Goal: Train a sentiment analyser using MLPClassifier.
- Instructions:
  - Read data from provided text files for sentiment analysis problem.
  - Build an MLPClassifier for sentiment classification.
  - Evaluate the performance of your neural network.

Content

- Perceptrons
- Stochastic gradient descent algorithm
- Backpropagation & multi-layered networks
- Deep learning architectures & applications

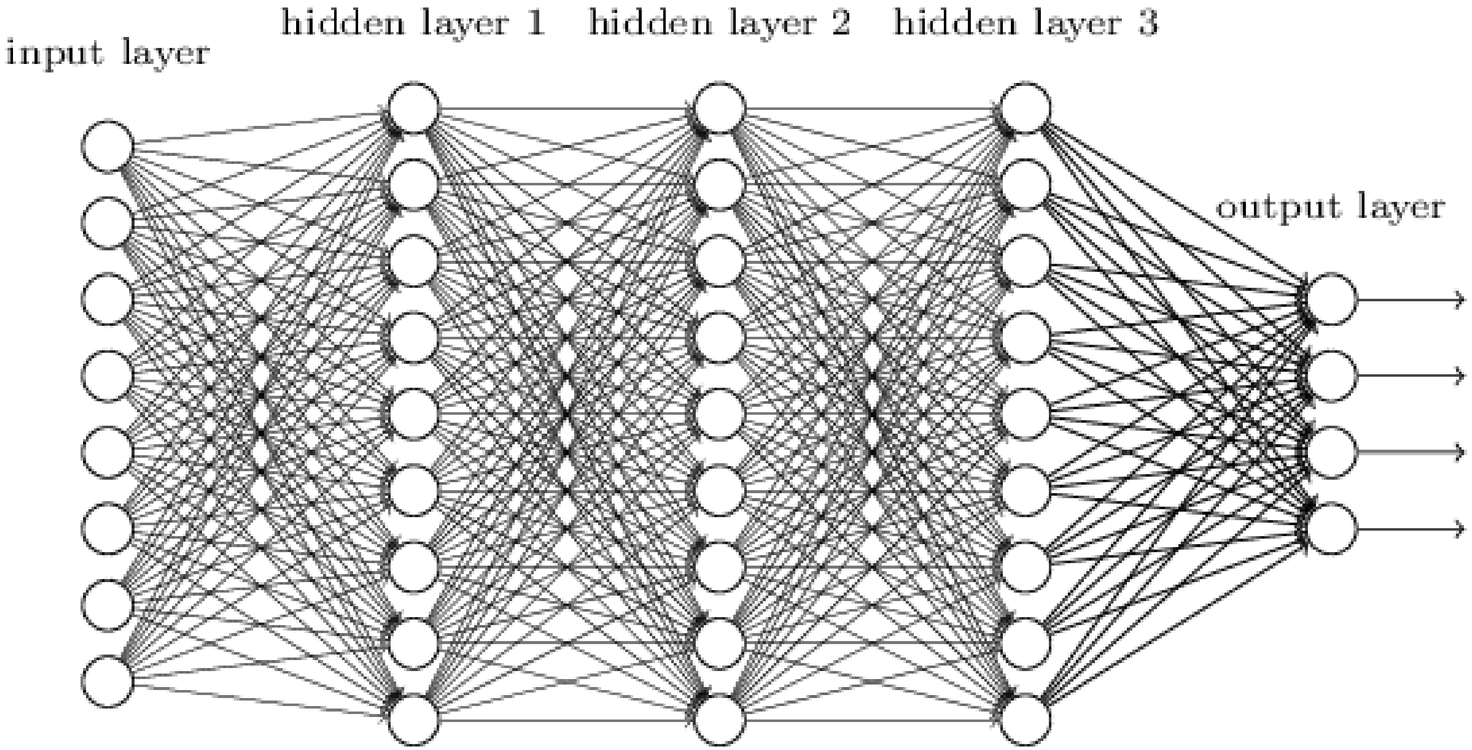
Overview of Deep Learning

- Related to the way the brain *processes* and *communicates* information and patterns, to define relationships between *stimuli* and *responses.*
- More *feature learning* than *task-specific learning.*
- Feature Learning
  - Also called representation learning.
  - Allows systems to automatically discover representations needed for feature detection or classification from raw data.

Common Architectures

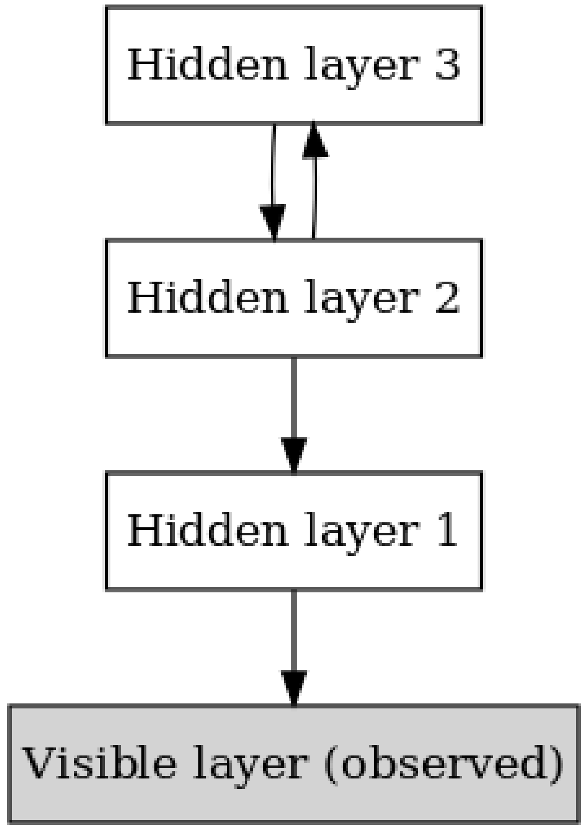
- Deep Neural Networks
  - Artificial neural networks with many hidden layers.
- Deep Belief Networks
  - Bayesian networks with many layers representing hidden variables.
- Deep Recurrent Networks
  - Allow capture temporal behaviour (i.e. time sequence).
- Convolutional Neural Networks
  - Have a feedforward, convolutional layer as the key building block.

Deep Neural Networks

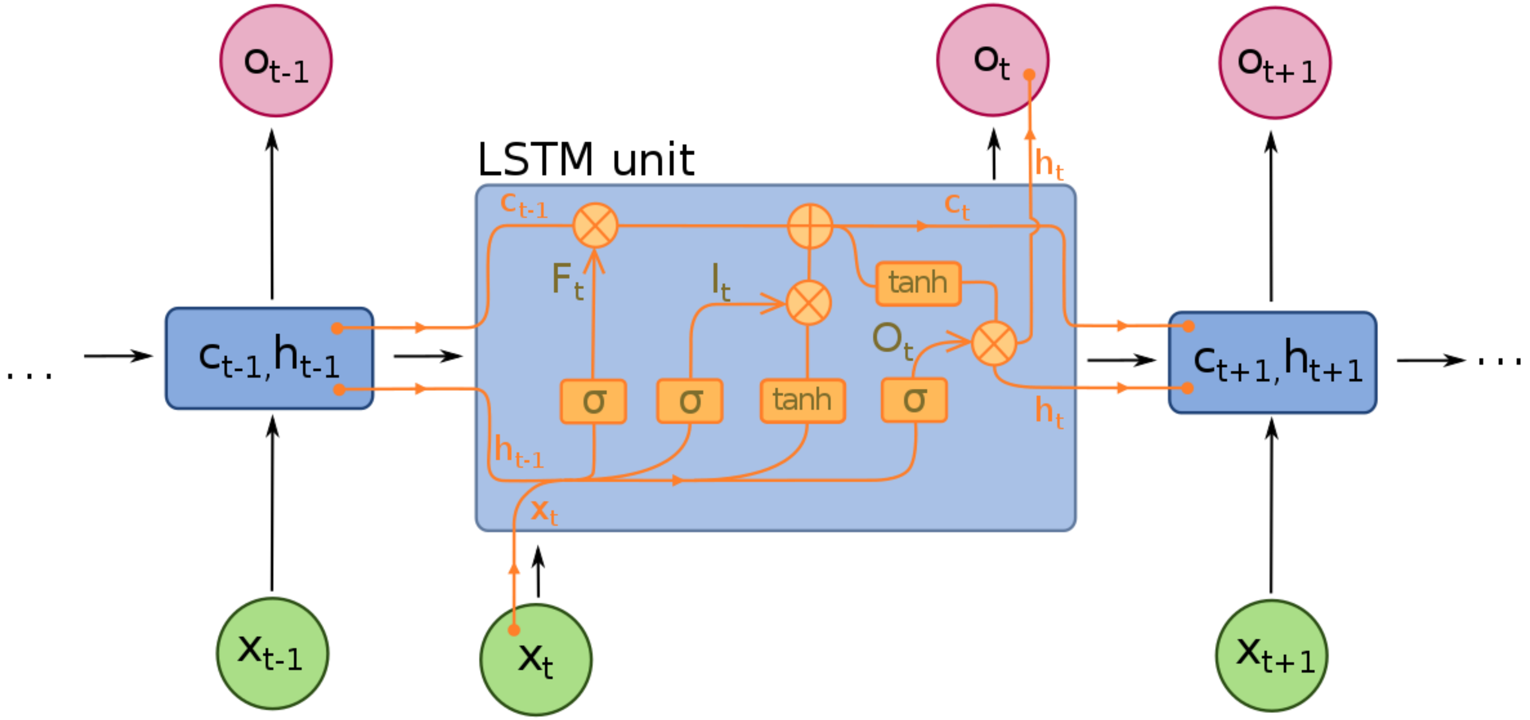


Deep Belief Networks

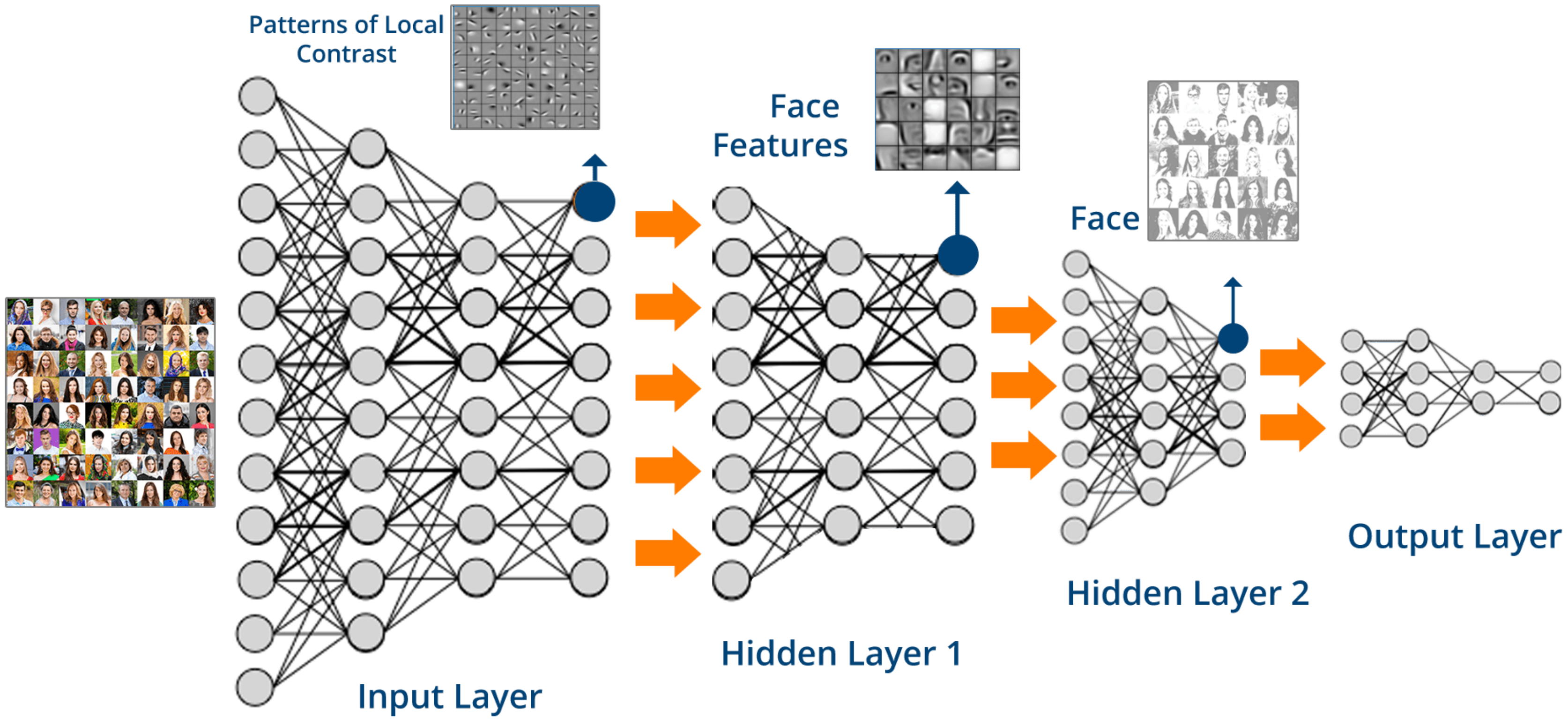
- Similar to deep neural networks, but connections are made between layers, NOT units within each layer.
- This allows a fast, layer-by-layer unsupervised training.
- Also can train a layer at a time.



Long Short-term Memory (LSTM)



Convolutional Neural Networks

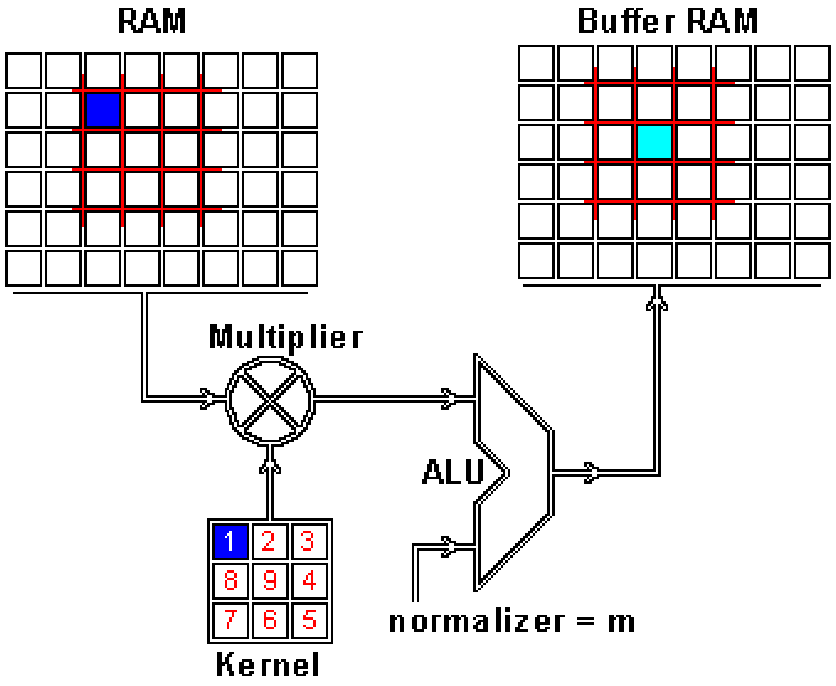


Components of a CNN

- Convolutional Layers
  - The core building block of CNN.
  - This layer is composed of a set of kernels (filters).
- Pooling Layers
  - A component for sampling to reduce the size of representation.
  - For example: MaxPooling.
- Activation Layers
  - A component for applying an activation function, like standard ANN.
  - For example: ReLU units.

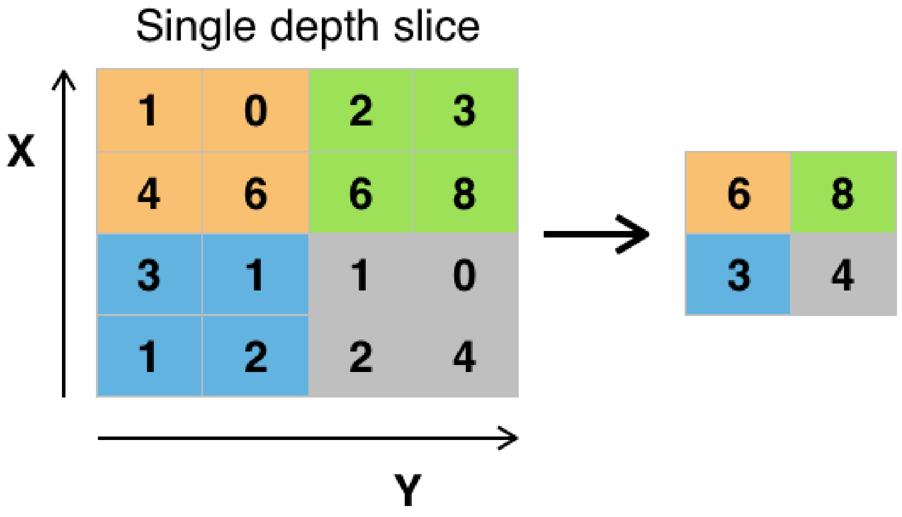
Convolutional Layer

- Performs a convolution on the input data (e.g. an image).
- This is just like any convolution done in image processing to apply filters.
- The challenge is the choice of *learnable filters*, or *kernels*.



Pooling Layers

- Performs down-sampling on the extracted features.
- We do not need the exact location of the feature, but only that *relative to other features*.
- Example: MaxPooling, where only the maximum is selected.



Activation Layer

- The activation layer is analogous to the activation function used in backpropagation neural networks.
- It increases the non-linear properties of decision function.
- Common activation functions:
  - ReLU (Rectified Linear Units): max(0, x)
  - Logistic function, e.g. sigmoid(x)
  - Other hyperbolic functions, e.g. tanh(x)