Lecture 29
Machine Learning: Introduction to Artificial Neural Networks

MP574: Applications

Sean B. Fain, PhD (sfain@wisc.edu)

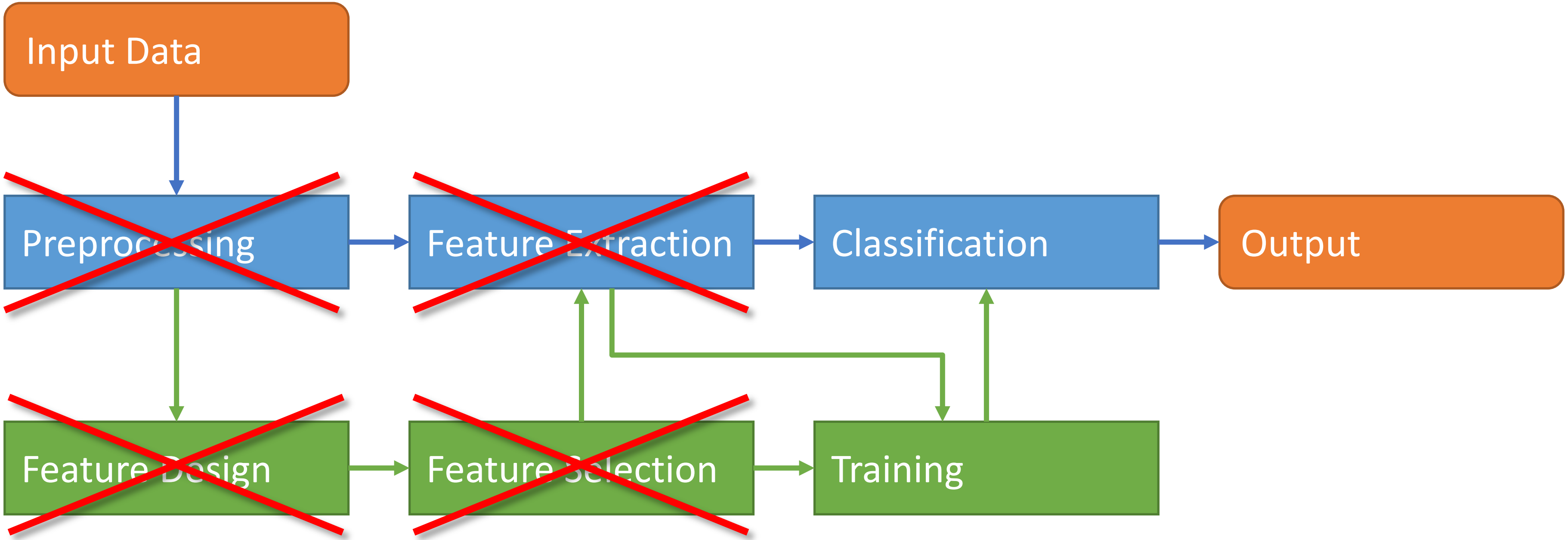
Diego Hernando, PhD (dhernando@wisc.edu)

ITK/VTK Applications: Andrew Hahn, PhD (adhahn@wisc.edu)

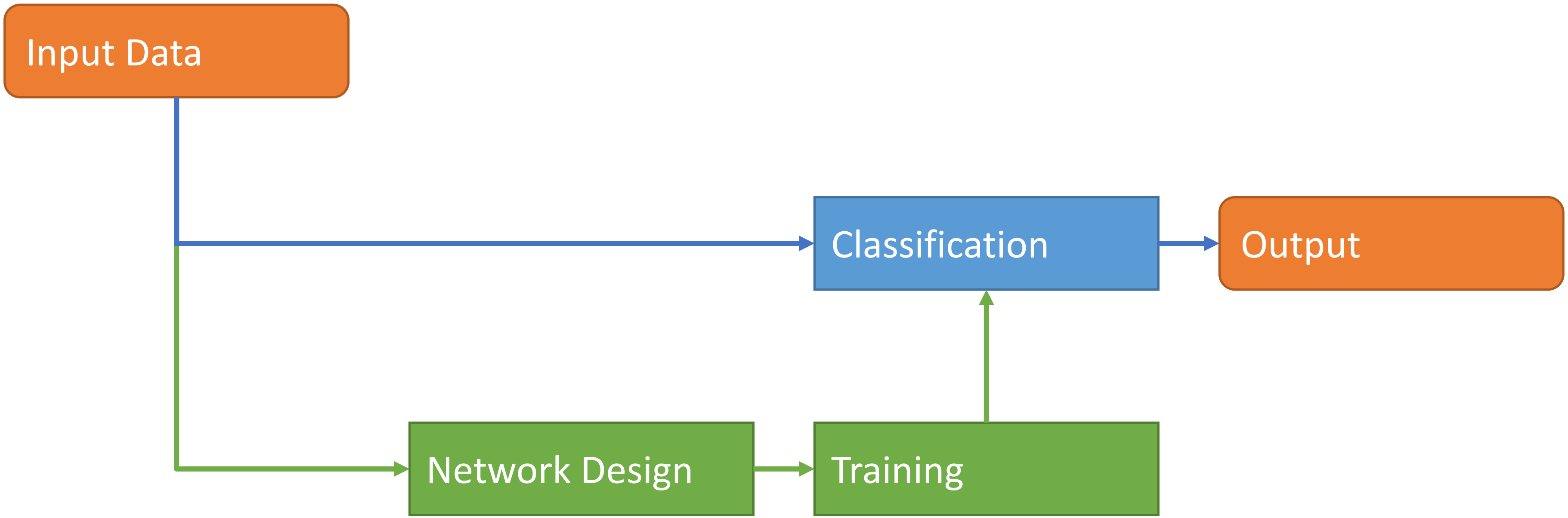
Learning Objectives

- Introduce deep learning vs. machine learning concept
- Introduce artificial neural networks
  - Layers and activation functions
  - Architecture and Training
  - Gradient based back-propagation strategy
- CNN architecture and libraries

Classic Machine Learning Pipeline



Deep Learning Pipeline



Deep Learning vs Classical Machine Learning

Classical Approach:

- Developer designs preprocessing steps based on visual appearance or trial and error
- Developer designs features based on experts' strategies
- Feature selection step to determine relevant set of features
- Large number of classifiers with different degrees of complexity
- **Most important step**: Feature design

Deep Learning:

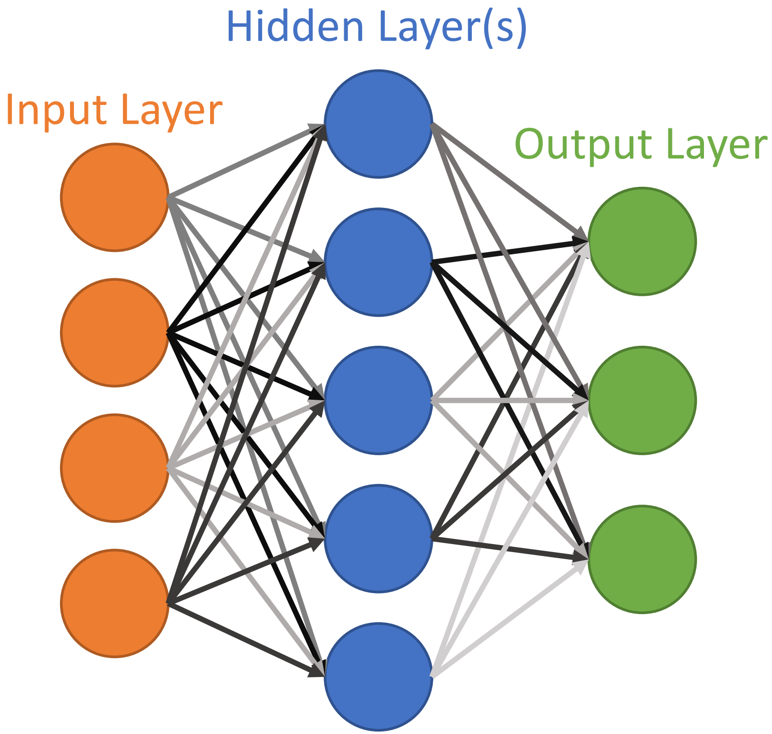
- Developer selects classification network based on experience (Multilayer Perceptrons)
- Classifier is trained to automatically learn
    - Preprocessing
    - Feature extraction and selection
    - Classification parameters
- **Most important step:** Training with **large number** of training data
- Training is computationally expensive

History

- 1943: McCulloch and Pitts created a computational model for neural networks (threshold logic)
- 1958: Rosenblatt develops the Perceptron, one of the first artificial neural networks
- 1965: First functional multilayer Perceptrons published by Ivakhnenko and Lapa
- Research on artificial neural networks stagnated afterwards due to simpler and computationally less expensive techniques as well as some inherent problems.
- Revived in the 1980s and popularized in the 2000s (Deep Learning)

Artificial Neural Network (ANN)

- Inspired by biological neural networks in animal and human brains
- Collection of nodes (artificial neurons)
- Nodes are usually organized in layers (1 input, 1 output, and N hidden layers) => multilayer Perceptron
- Connections represent synapses and connect two nodes from adjacent layers



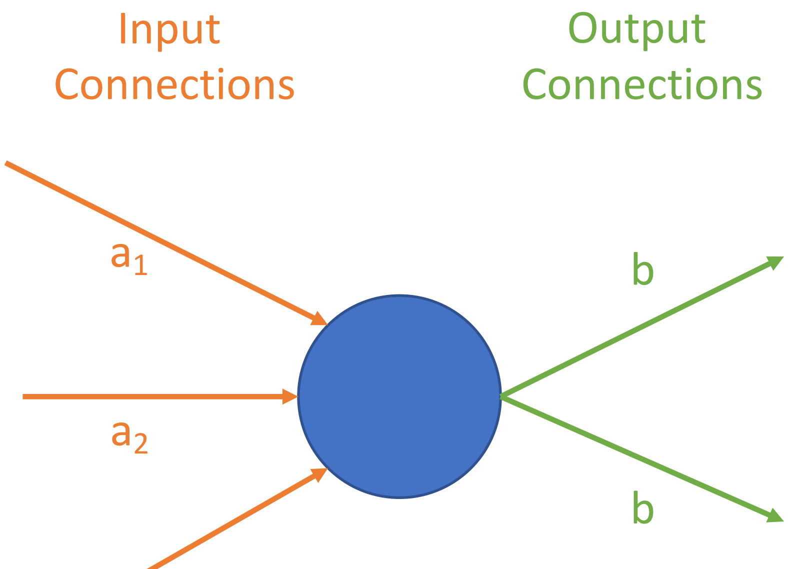
Types of ANN

- Feedforward neural network
  - Connections are only allowed between adjacent layers
  - Information moves only in one direction
  - Example: Multilayer Perceptron
- Recurrent neural network
  - Often used for cases where input is a time sequence
  - Output of previous inputs can be used as input for subsequent time points
  - Can use “internal state” (memory) to process data
- Other types: Modular, Dynamic, …

Components of ANN

Artificial Neuron

- Each node has
  - A set of inputs a_i
  - A set of weights for each input w_i
  - One output value b
  - An activation function, with threshold, θ.
- Inputs are determined by the outputs of the previous layer
- Training process modifies input weight w_i



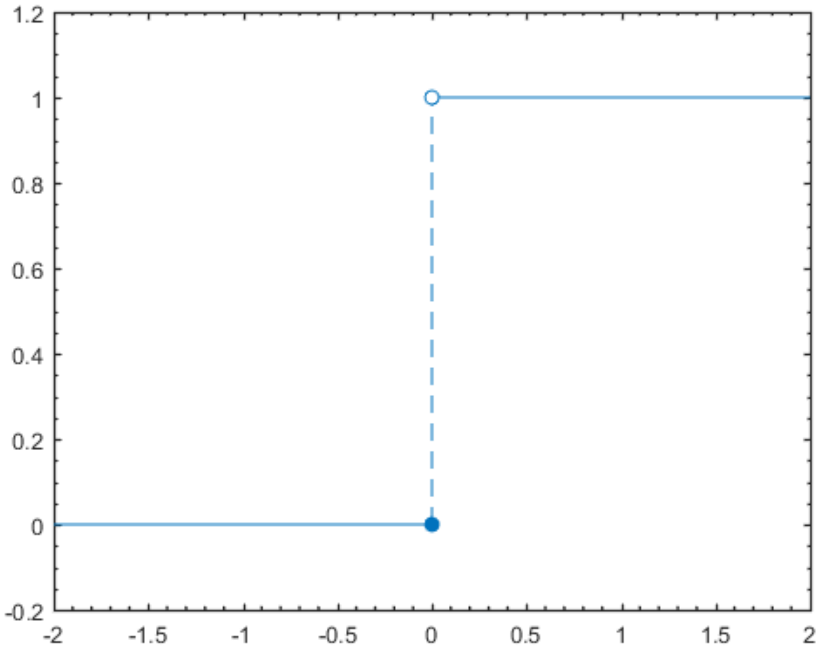
$$b = f_\theta\left(\sum_i w_i a_i\right)$$

Activation Function: Step Function

- Threshold Learning

$$b = \begin{cases} 1 & \text{if } \sum_{\forall i} w_i a_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

- Drawbacks:
  - Not suited for more than two classes (ambiguities if more than one output neuron is active)
  - Not continuous (not differentiable at x = 0)
  - Derivative for all other points is 0
  - ➢ Not well suited for gradient based training

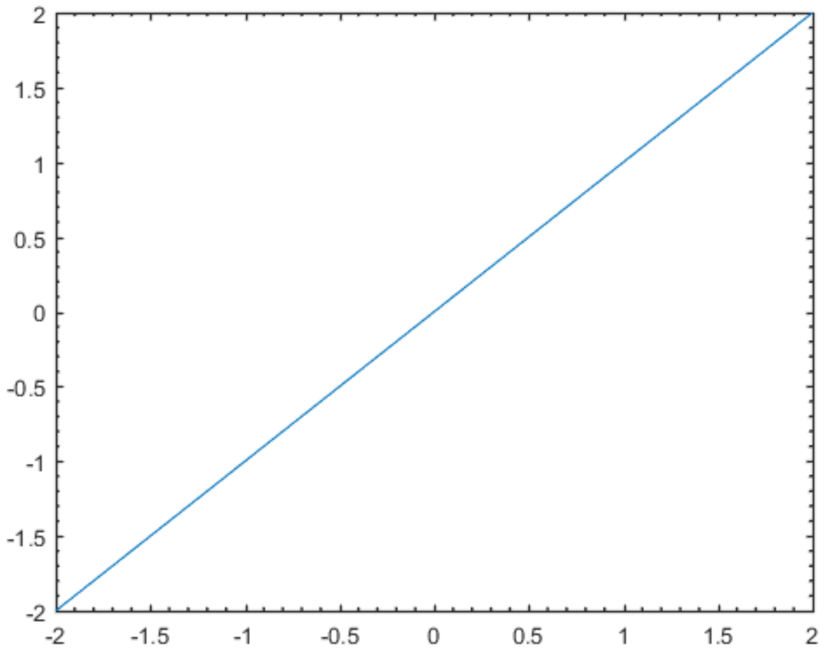


Activation Function: Linear (Identity)

- Output is proportional to the weighted some of the input (e.g. identity function)

$$b = \sum_{\forall i} w_i a_i$$

- Drawbacks
  - Derivative is constant
  - Not suitable for gradient based learning since back-propagated error is not dependent on changes in input

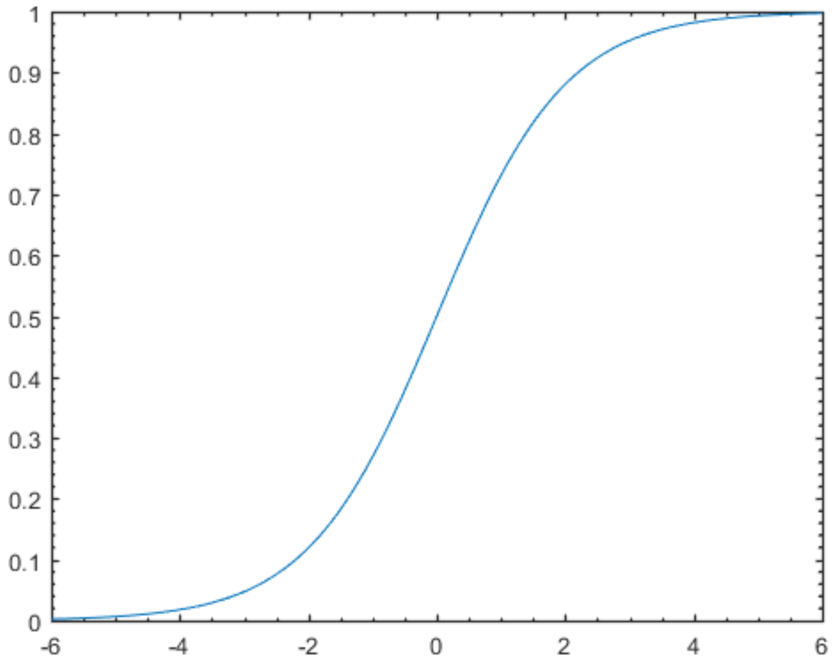


Activation Function: Sigmoid Function

- Smooth step-like function

$$b = \frac{1}{1 + e^{-\sum_{\forall i} w_i a_i}}$$

- Advantages
  - Continuous
  - Smooth gradient
- Drawbacks
  - Gradient very small when x is very different from 0
- One of the most widely used activation functions

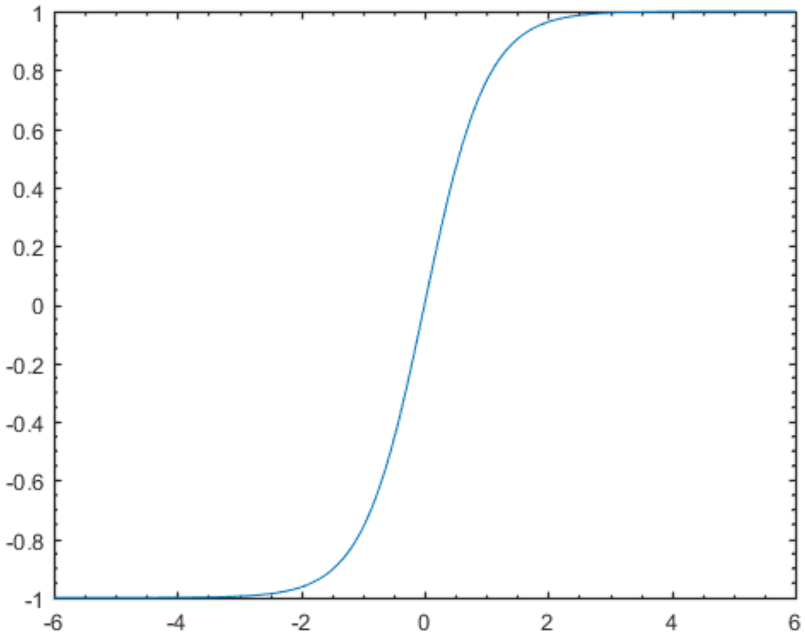


Activation Function: Hyperbolic Tangent Function

- Equivalent to scaled sigmoid function

$$b = \tanh\left(\sum_{\forall i} w_i a_i\right) = \frac{2}{1 + e^{-2\sum_{\forall i} w_i a_i}} - 1$$

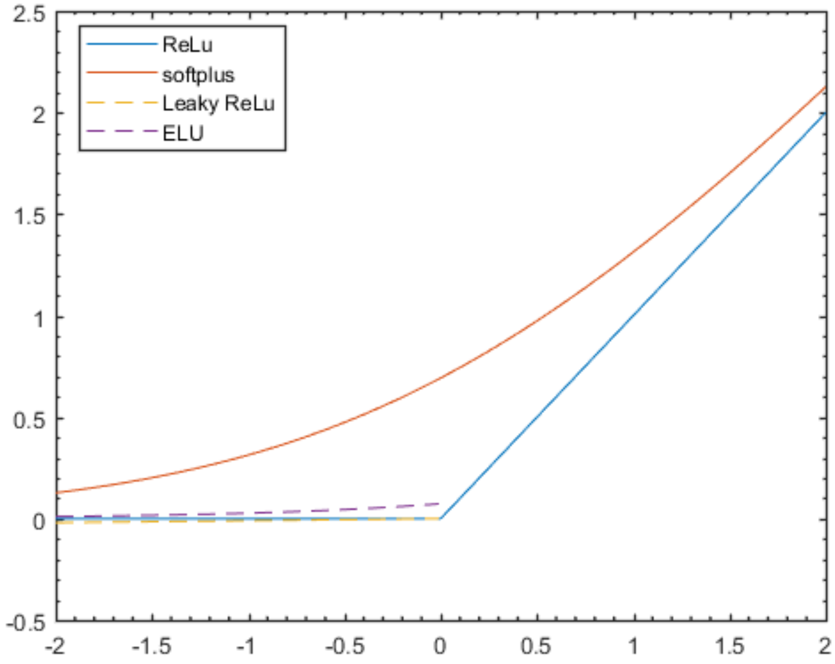
- Advantages and drawbacks similar to sigmoid function



Activation Function: Rectifier (ReLu)

$$b = \max\left(0, \sum_{\forall i} w_i a_i\right)$$

- Advantages:
  - Sparsifies set of active neurons (more efficient, less overfitting)
- Drawbacks:
  - Gradient is 0 for x < 0
    - Weights will not be adjusted during training if x < 0
    - Neurons may become passive and not respond anymore (dying ReLu problem)
- Variants: Softplus, Noisy ReLu, Leaky ReLu, ELU

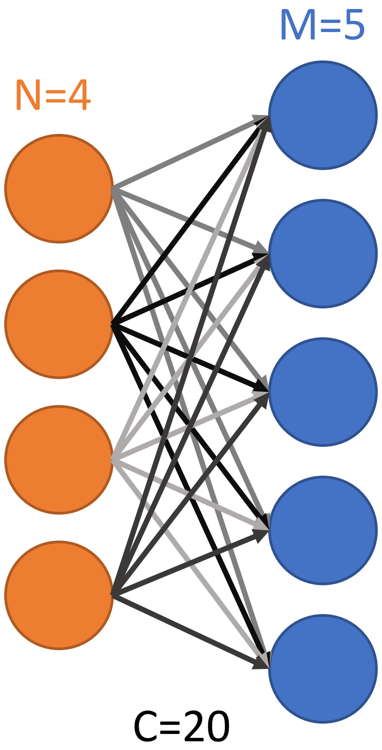


Layers

- Set of neurons (of the same type)
- Neurons are not connected to other neurons within the same layer
- Can be classified by
  - Connectedness e.g.
    - Fully connected: each neuron is connected to every neuron of the previous layer
    - Convolutional layer: each neuron is connected to a local block (usually 2D for image processing) of neurons from the previous layer
  - Type of neurons (function) e.g.
    - ReLu
    - Normalization
    - Pooling

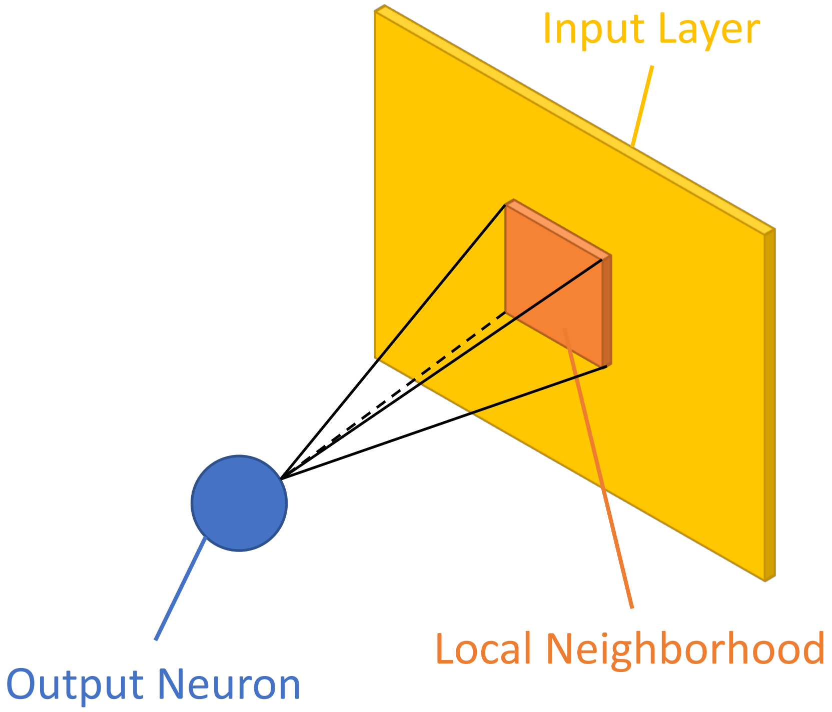
Fully Connected Layer

- Each neuron is connected to every neuron of the previous layer
- Each connection has a separate weight
  - Can represent very complex functions
  - Large number of weights (N*M)
  - Tends to overfitting
- Usage
  - Small number of nodes (e.g. simple problems or close to output layer)



Convolutional Layer

- Connectivity pattern resembles the neuron organization in the visual cortex
- Well suited to detect local features in images
- Input neurons commonly organized in 2D or 3D
- Each neuron is connected to the neurons in the local neighborhood in the previous layer



Convolutional Layer

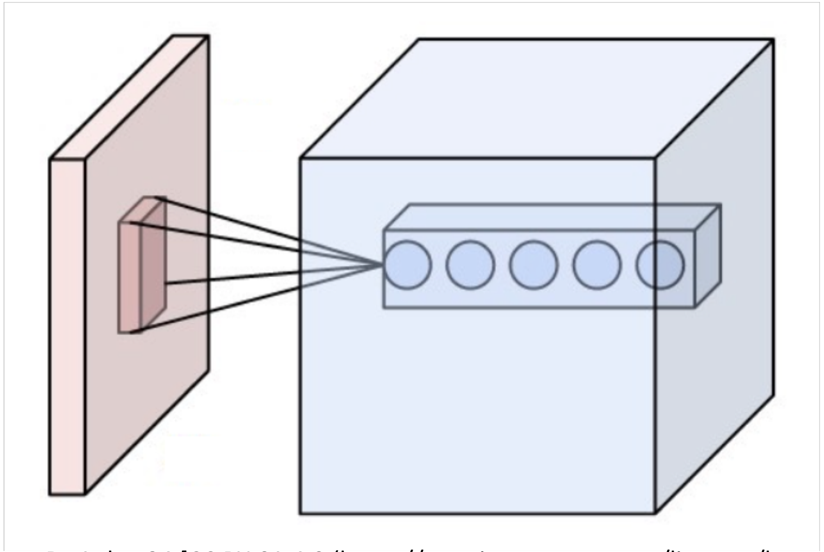
- Weights are shared for all output neurons
- Only one set of weights has to be stored for the local neighborhood
  - Considerable reduction of variables compared to fully connected e.g. 5x5 neighborhood requires 25 variables (fully connected layers for common images would require millions of variables)

$$b_1(x, y) = \sum_{\forall i} \sum_{\forall j} w_{ij} \, b_0(x + i, y + j)$$

- Corresponds to a convolution operation over the set of neurons from the previous layer

Stacked Convolutional Layer

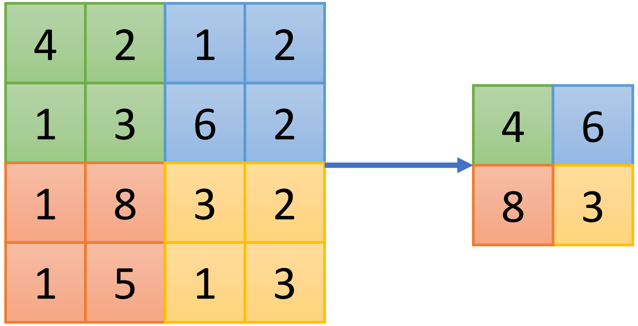
- Often multiple 2D convolutions are combined in a single layer
- Allows detection of different types of features
- Each convolution can have a different set of weights
- 3D output (Z-dimension corresponds to different sets of weights)



By Aphex34 [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], from Wikimedia Commons

Pooling Layer

- Commonly used to reduce number of neurons
- Allows detecting features on a larger scale
- Common variants:
  - Max pooling
  - Average pooling
- Parameters
  - Extend: number of neurons in each direction
  - Stride: step-size between pooling operations in each direction



Output Encoding

Binary decision (2 classes)

- Classes are encoded as -1 and 1
- Single output node e.g.
  - ≤ 0: class 1
  - > 0: class 2

Regression

- Real valued output (scalar or vector)

Mutliple Classes

- Array of zeros and ones
- True class is one all others 0
- One output node for each class
- Choose node with maximum value

Output Layer

- Purpose: scale output data to probability
  - Each output node is in the interval [0, 1]
  - Sum of all output node is 1
- Softmax

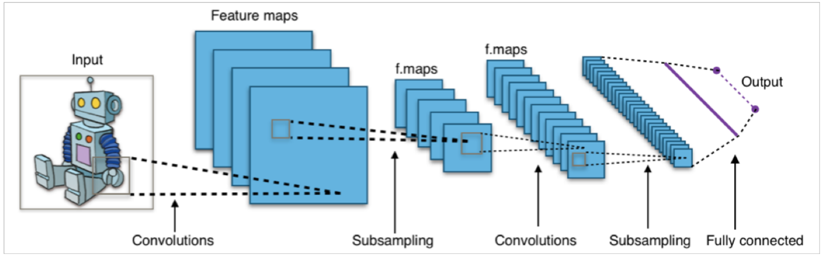
$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{\forall k} e^{y_k}}$$

- Binary classification: value between 0 and 1
  - Activation functions (e.g. sigmoid) can be used

Convolutional Network Architecture

- Number of input nodes given by image size
- Number of output nodes given by number of classes
- Hidden layers often build from multiple blocks of layers at different scale spaces
- Each block is a combination of convolutional ReLu layers followed by Max-pooling
- Width and height are reduced to 1, depth becomes the number of output nodes

- Final layer often fully connected



Convolutional Network Architecture

- No clear rule on the number and size of the hidden layers
- Networks are often designed empirically
- Networks can be trained against each other to choose best design
- Architectures are often reused for different purposes

- Popular designs:
  - LeNet-5 (1998): 7 levels
  - AlexNet (2012): stacked convolutional layers
  - GoogleNet (2014): 22 layers but less parameters than AlexNet
  - ResNet (2015): 152 layer architecture with skip-connections and batch normalization

Training Artificial Neural Networks

Training (Supervised Learning)

- Requirements
  - Artificial neural network (ANN)
  - Set of input images
  - Label for each input image representing ground truth
- Loop through all input images
  1. Apply ANN to input image
  2. Calculate loss function based on network output
- Adjust network weights based on combined loss
- Repeat training procedure until accuracy is not improving

Training: Loss Functions

- Compares the output of the ANN to the ground truth
- Combines the error of multiple samples (images)
- Combines the error of output nodes

Output Layer	Ground Truth	Error
0.2	0	0.2
0.1	0	0.1
0.7	1	-0.3

Loss Functions: Mean Squared Error (MSE)

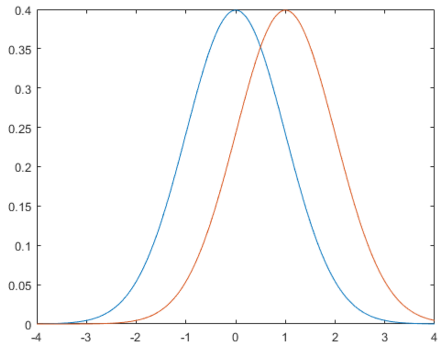
- Commonly used for regression problems
- Can be problematic for multiclass classification problems
  - Small partial derivatives when weights are close to 0

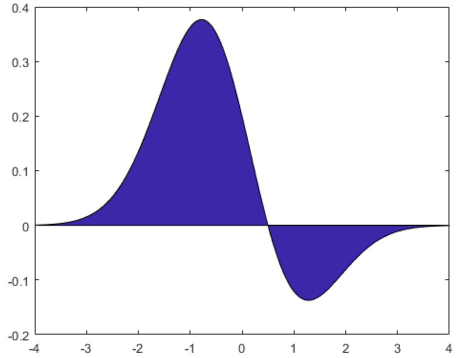
$$E = \frac{1}{2N} \sum_{\forall i} \|y_i - \hat{y}_i\|^2$$

Loss Function: Cross-Entropy for Two Classes

- Minimizes the Kullback-Leibler divergence
  - Measures the distance between two probability distributions
- Information theory interpretation
  - Average number of bits required to encode the measured events
- Magnitude of the partial derivatives is only dependent on the error
  - Faster learning rates when weights are close to 0

$$E = -\sum_{\forall i} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$





Loss Function: Cross-Entropy for Multiple Classes

- Sum over entropies for all classes
- Second term is usually omitted
  - For each sample one of the classes must be true

$$E = -\sum_{\forall i} \sum_{\forall j} y_{ij} \log(\hat{y}_{ij})$$

Training Neural Networks

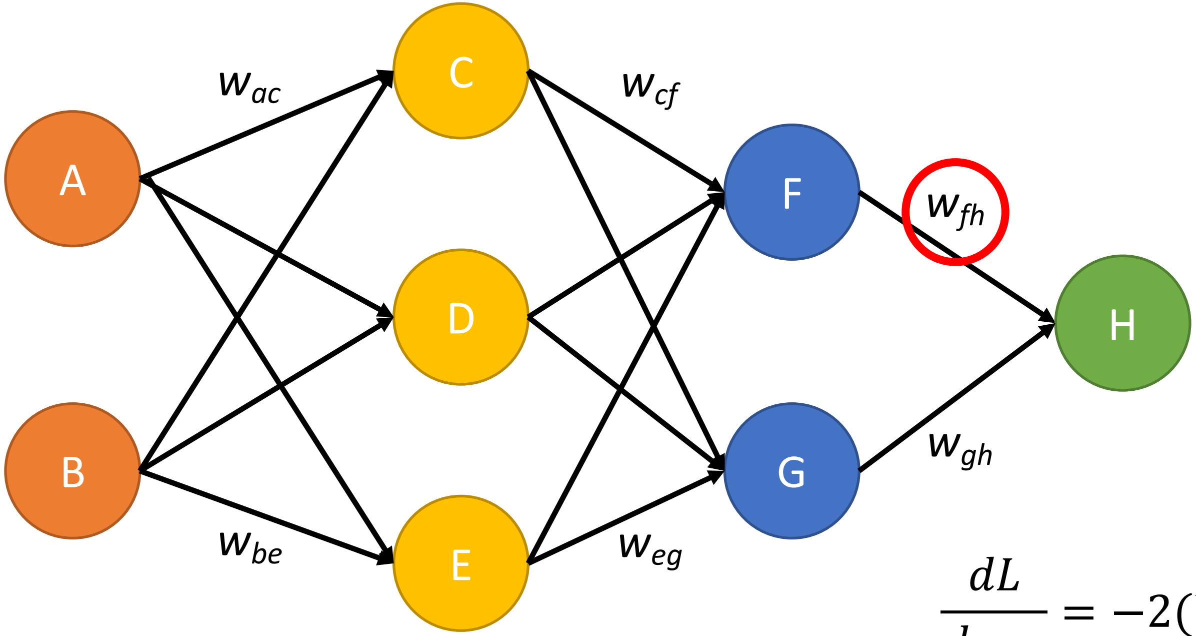
- Common optimization tasks:
  - Gradient based approaches can be used to minimize a cost function e.g.
    - Gradient descent
    - Newton-Raphson
    - Levenberg-Marquardt
- Problem for neural networks:
  - Performance measure usually only affected indirectly
  - Ground truth is only known for the output layer

Training NN: Perturbing Weights

- Simple algorithm:
  1. Calculate combined loss function
     a) Calculate output of the NN for all input images
     b) Compare output to known ground truth
  2. Randomly change weights
  3. Calculate combined loss function
  4. Keep changes if loss is reduced otherwise undo the changes

- Not very efficient!

Training NN: Gradient-Based Optimization



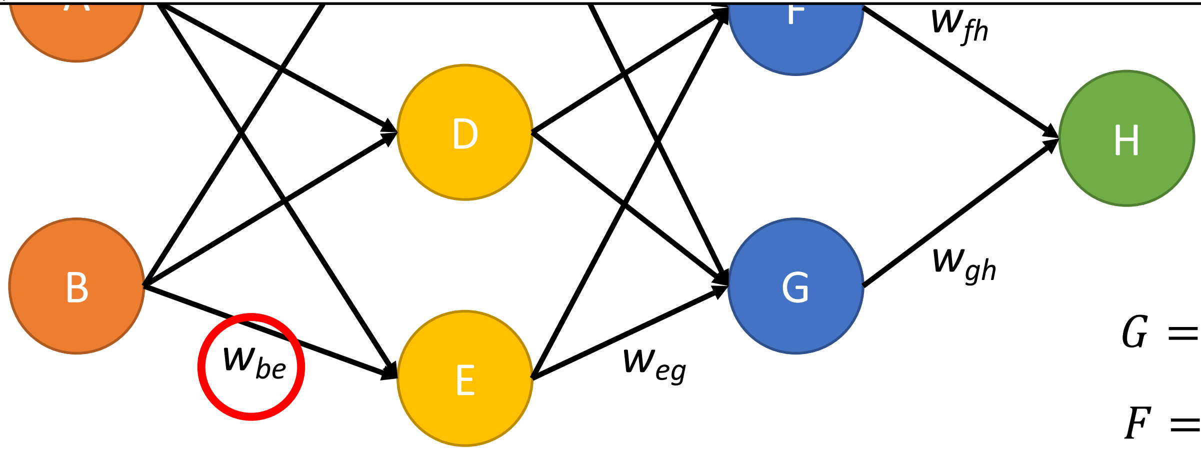
$$\text{loss} = L(Y - H)$$

$$H = \sigma\left(w_{fh}F + w_{gh}G\right)$$

$$\frac{dL}{dw_{fh}} = -2(Y - H)\sigma'\left(w_{fh}F + w_{gh}G\right)F$$

Training NN: Gradient-Based Optimization

$$\frac{dL}{dw_{be}} = -2(Y-H)B\sigma'(Aw_{ae}+Bw_{be})\sigma'\left(Fw_{fh}+Gw_{gh}\right)$$
$$\left(w_{eg}w_{gh}\sigma'\left(Cw_{cg}+Dw_{dg}+Ew_{eg}\right)+w_{ef}w_{fh}\sigma'\left(Cw_{cf}+Dw_{df}+Ew_{ef}\right)\right)$$



$$\text{loss} = L(Y-H)$$
$$H = \sigma\left(w_{fh}F + w_{gh}G\right)$$
$$G = \sigma\left(w_{cg}C + w_{dg}D + w_{eg}E\right)$$
$$F = \sigma\left(w_{cf}C + w_{df}D + w_{ef}E\right)$$

…

Training NN: Backpropagation

- Not an optimization technique but an efficient way to calculate gradients (partial derivatives)
- Uses chain rule recursively

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

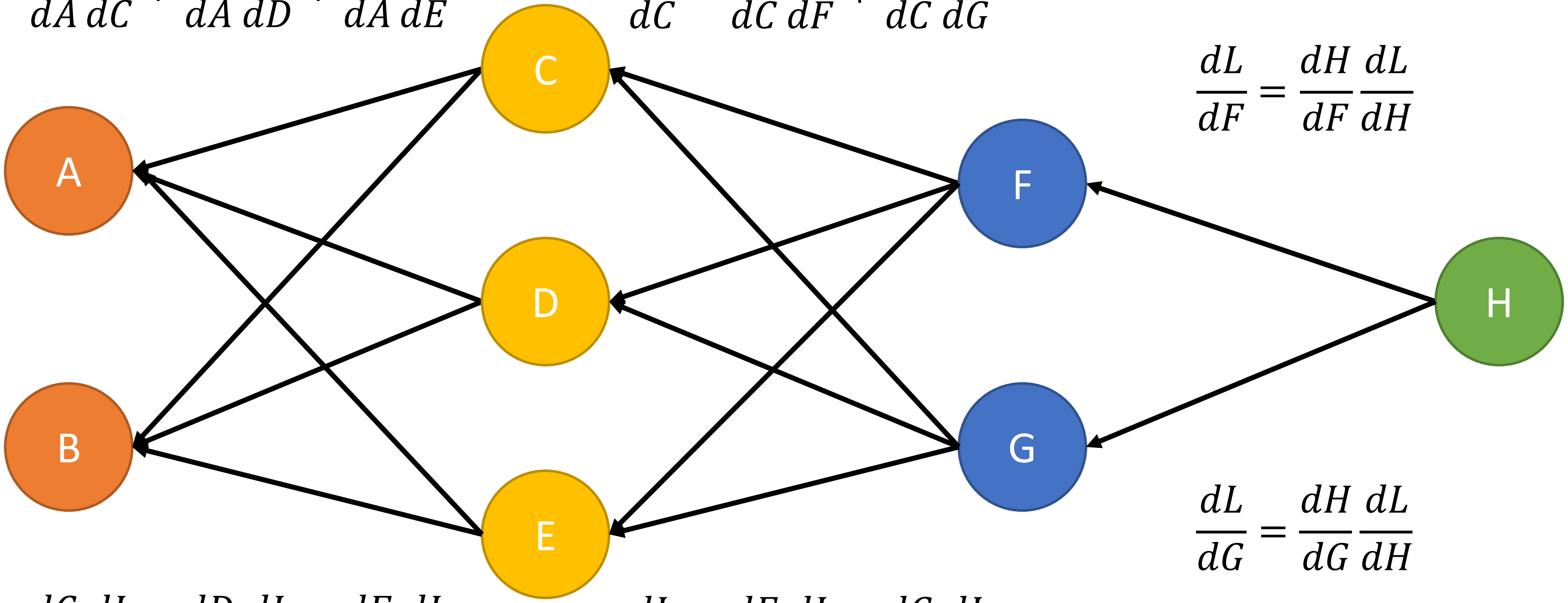
$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

Training NN: Backpropagation

$$\frac{dL}{dA} = \frac{dC}{dA}\frac{dL}{dC} + \frac{dD}{dA}\frac{dL}{dD} + \frac{dE}{dA}\frac{dL}{dE}$$

$$\frac{dL}{dC} = \frac{dF}{dC}\frac{dL}{dF} + \frac{dG}{dC}\frac{dL}{dG}$$

$$\frac{dL}{dF} = \frac{dH}{dF}\frac{dL}{dH}$$



$$\frac{dL}{dG} = \frac{dH}{dG}\frac{dL}{dH}$$

$$\frac{dL}{dB} = \frac{dC}{dB}\frac{dL}{dC} + \frac{dD}{dB}\frac{dL}{dD} + \frac{dE}{dB}\frac{dL}{dE}$$

$$\frac{dL}{dE} = \frac{dF}{dE}\frac{dL}{dF} + \frac{dG}{dE}\frac{dL}{dG}$$

Training NN: Backpropagation

- Vector notation: **Jacobi matrix** times gradient

$$\nabla_{\boldsymbol{x}_{n-1}} L = \left(\frac{\partial \boldsymbol{x}_n}{\partial \boldsymbol{x}_{n-1}}\right)^{\mathsf{T}} \nabla_{\boldsymbol{x}_n} L$$

- Activation function derivative (e.g. sigmoid)

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- Update rules e.g. Gradient descent

$$\breve{w}_i = w_i - \eta \frac{dL}{dw_i}$$

Deep Learning Libraries

- TensorFlow (Google):
  - Powerful library (high-level and low-level functions)
  - Python
  - Supports GPU
  - Under active development
- Matlab Neural Network Toolbox
  - Easy to use
  - Support to generate GPU code
- Caffee
  - Focus on convolutional neural networks
  - Large number of pre-trained networks
  - Python
  - GPU support
- Many more …