Session 1
Machine learning and Neural network

General machine learning

Neurons

Limitation of traditional NN

Course resources

- Previous lecturer Pr. Fabien Moutarde:

https://github.com/fabienMoutarde/DLcourse

- Stanford class cs231n by Fei-Fei Li,A.Karpathy and J.Johnson:

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

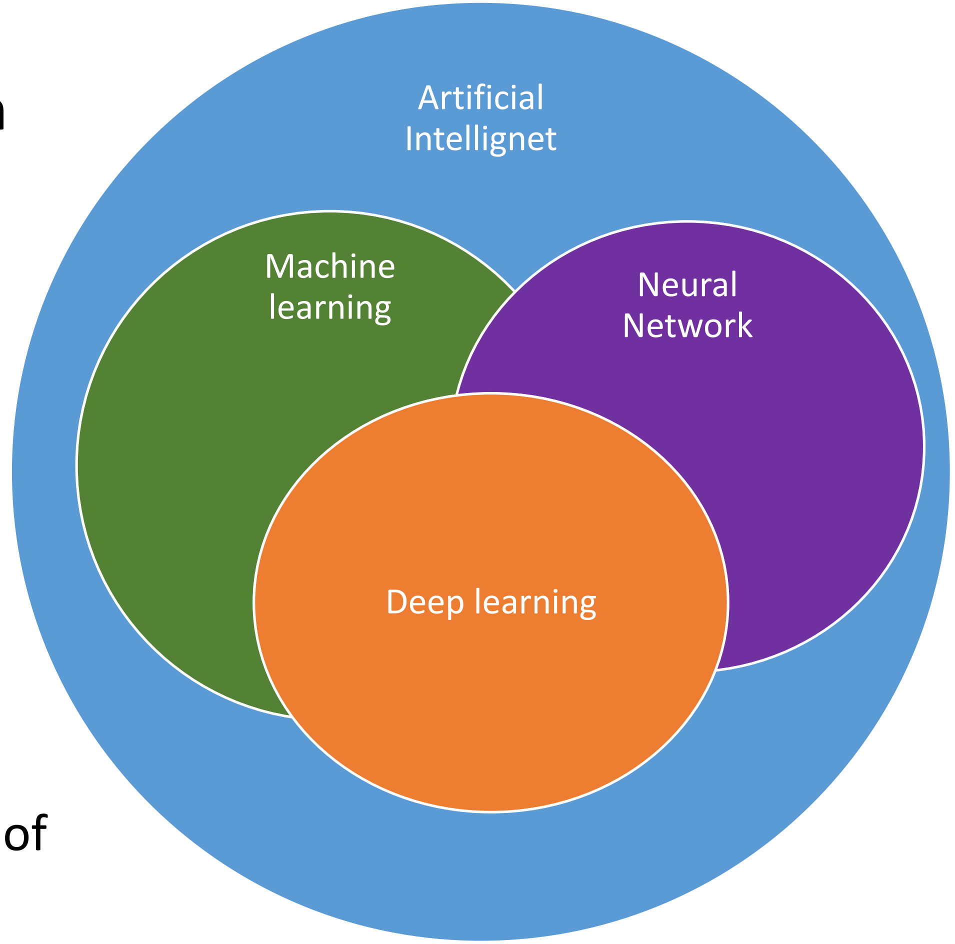
- Mitchell, Tom (1997). *Machine Learning*. New York: McGraw Hill. ISBN 0-07-042807-7. OCLC 36417892.

Outline

- Machine learning
- Review of artificial neural network
  - MLP
  - Training
  - Breakthrough of neural network
  - Deep learning
  - Application
- Convolution neural network
- Famous ConvNet architecture

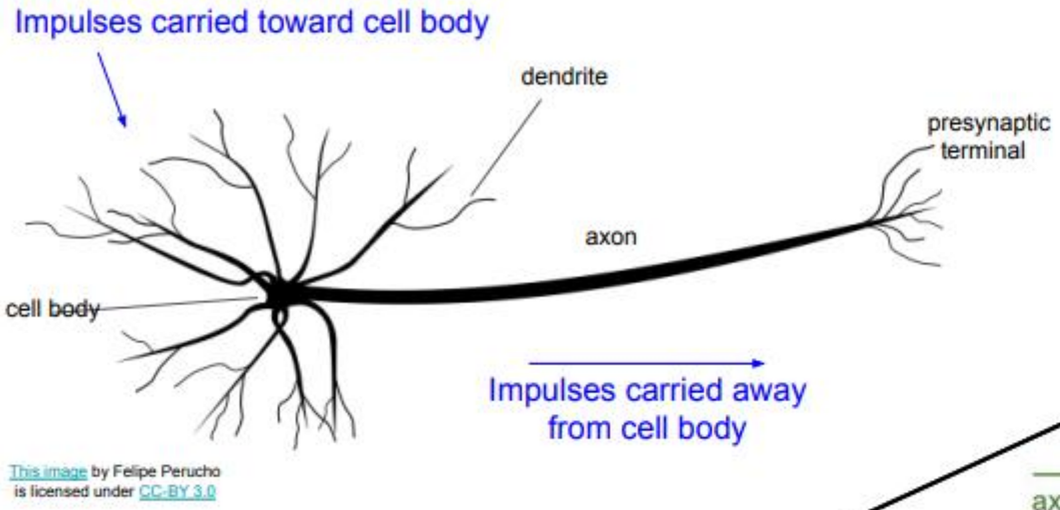
Overview of Artificial Intelligent

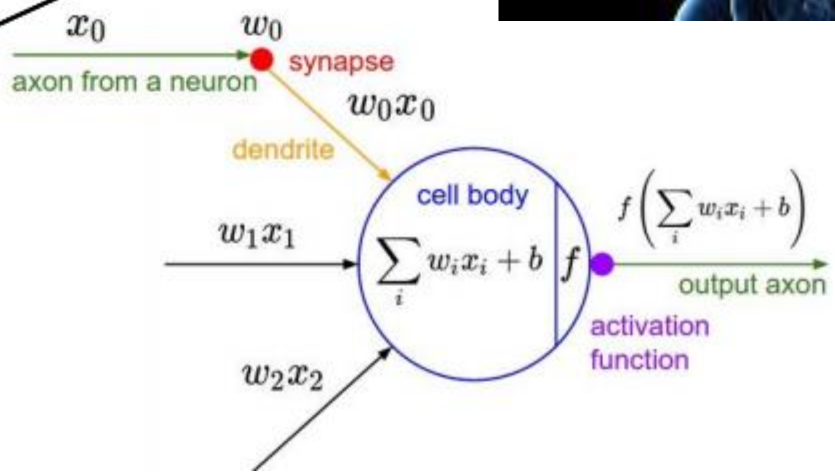
- **Machine learning (ML)** is the study of computer algorithms that can improve automatically through the experience and by the use of data [Mitchell, Tom, 1997]. Popular algorithms are:
  - Linear classifiers
  - K-nearest Neighbors
  - Boosted stumps
  - Support Vector machines (SVMs)
- **Artificial neural networks** (**ANNs**), usually simply called **neural networks** (**NNs**), are computing systems inspired by the biological neural networks that constitute animal brains.
  - Concerning on the design of structure, connection, flow of the signal



Neural Network

- With the understanding of how our human brains works (although not completely), the most success “imitate” human brain modules are:

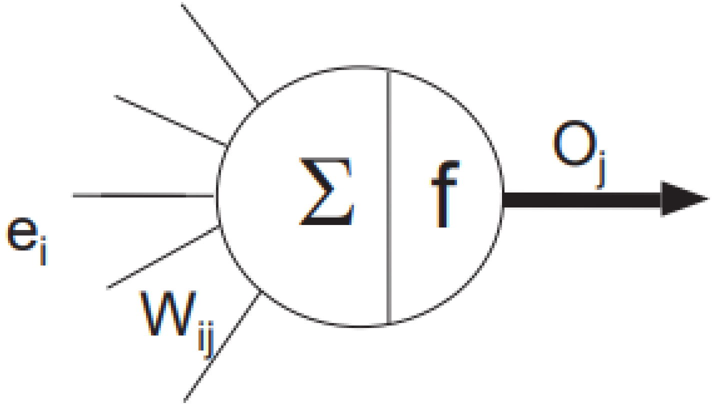




Activation (1)

- However, the possible type of neurons (activation) might not be that simple as well as the way synaptic weights connect to each other

PRINCIPLE

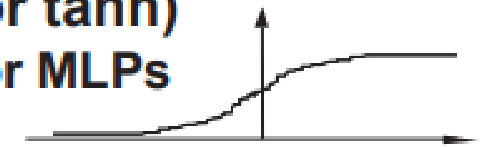


$$O_j = f\left(W_{0j} + \sum_{i=1}^{n_j} W_{ij} e_i\right)$$

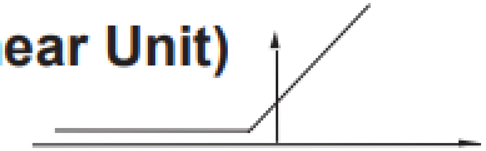
W_{0j} = "bias"

ACTIVATION FUNCTIONS

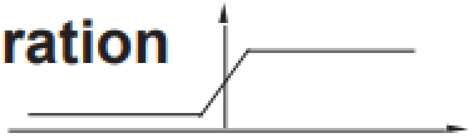
- **Threshold (Heaviside or sign)**
  → ***binary*** **neurons**
- **Sigmoïd (logistic or tanh)**
  → **most common for MLPs**



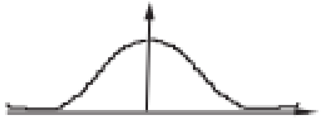
- **Identity** → ***linear*** **neurons**
- **ReLU (Rectified Linear Unit)**



- **Saturation**

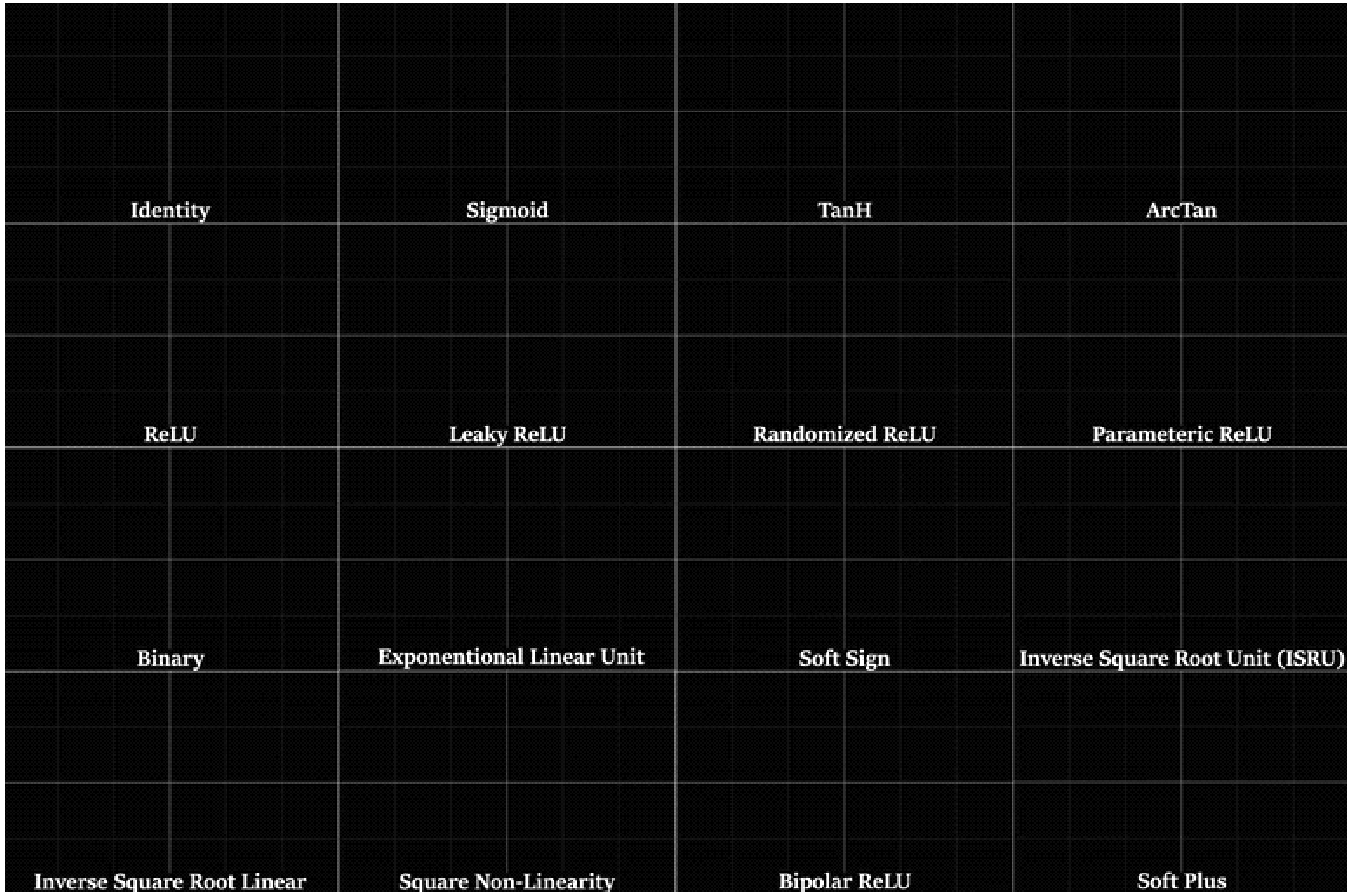


- **Gaussian**



Activation (2)
Modern approaches

- In the past decades, more researches have found the new design of activation functions that improve specific tasks



Activation (3)
Modern approaches (ReLU)

- Rectified Linear Unit (ReLU) is proposed by Hahnloser et al. in 2000

$$ReLU(\mathrm{x}) = \max(0, \mathrm{x})$$

$$ReLU'(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \leq 0 \end{cases}$$

- Pros:
  - Less time and space complexity, because of sparsity, and compared to the sigmoid, it does not evolve the exponential operation, which are more costly.
  - Avoids the vanishing gradient problem.
- Cons:
  - Introduces the *dead relu* problem, where components of the network are most likely never updated to a new value. This can sometimes also be a pro.
  - ReLUs does not avoid the exploding gradient problem

Activation (4)
Modern approaches (ELU)

- Exponential Linear Unit (ELU) is proposed by D.A. Clevert et al. 2016

$$ELU\ (x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha(e^x - 1) & \text{if} \quad x \leq 0 \end{cases}$$

$$ELU'(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ \alpha e^x & \text{if} \quad x \leq 0 \end{cases}$$

α is around 0.1 to 0.3

- Pros
  - Avoids the *dead relu* problem.
  - Produces negative outputs, which helps the network nudge weights and biases in the right directions.
  - Produce activations instead of letting them be zero, when calculating the gradient.
- Cons
  - Introduces longer computation time, because of the exponential operation included
  - Does not avoid the exploding gradient problem
  - The neural network does not learn the alpha value

Activation (5)
Modern approaches (Leaky ReLU)

- Leaky Rectified Linear Unit is proposed by AL Maas et al. 2013

$$LReLU\ (x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha x & \text{if} \quad x \leq 0 \end{cases}$$

$$LReLU'(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ \alpha & \text{if} \quad x \leq 0 \end{cases}$$

α is around 0.1 to 0.3

- Pros
  - Like the ELU, we avoid the *dead relu* problem, since we allow a small gradient, when computing the derivative.
  - Faster to compute then ELU, because no exponential operation is included
- Cons
  - Does not avoid the exploding gradient problem
  - The neural network does not learn the alpha value
  - Becomes a linear function, when it is differentiated, whereas ELU is partly linear and nonlinear.

Activation (6) Modern approaches (GELU)

- Gaussian Error Linear Unit. An activation function used in the most recent Transformers – Google's BERT and OpenAI's GPT-2. It is proposed by Hendrycks, Dan; Gimpel, Kevin (2016)

$$GELU\ (x) = 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

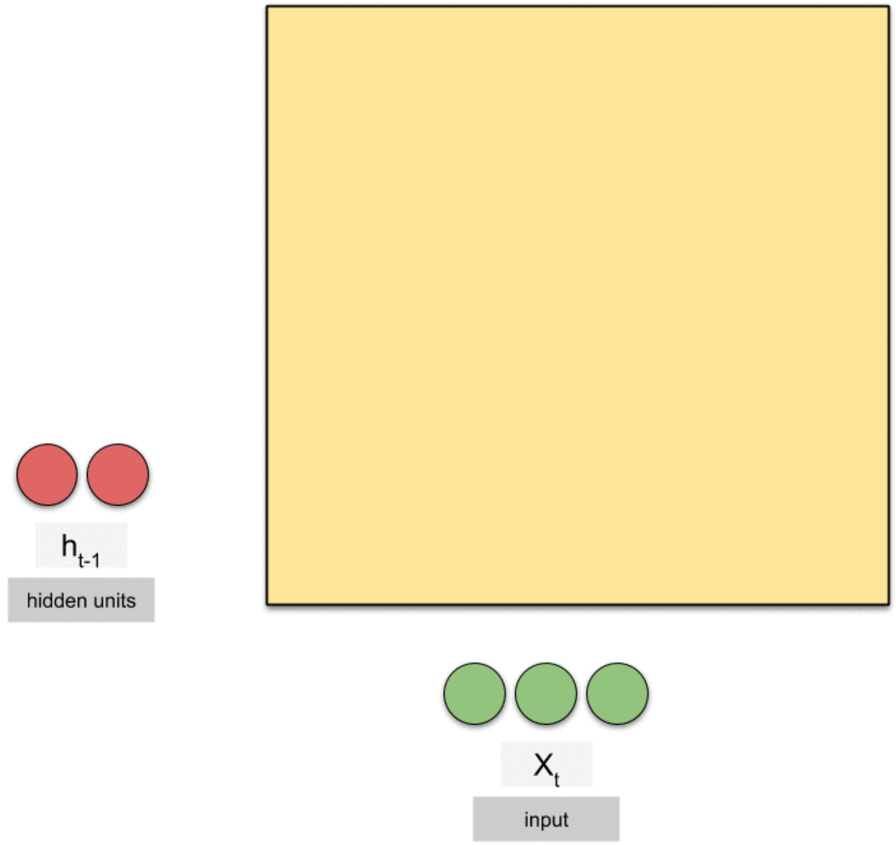
$$GELU'(x) \\ = 0.5\tanh(0.0356774x^3 + 0.797885x) \\ + (0.535161x^3 + 0.398942x)\,\mathrm{sech}^2(0.0356774x^3 + 0.797885x) + 0.5$$

- Pros
  - Seems to be state-of-the-art in NLP, specifically Transformer models – i.e. it performs best
  - Avoids vanishing gradients problem
- Cons
  - Fairly new in practical use, although introduced in 2016.

Good resource to understand how you choose/test activation functions: https://mlfromscratch.com/activation-functions-explained/#relu

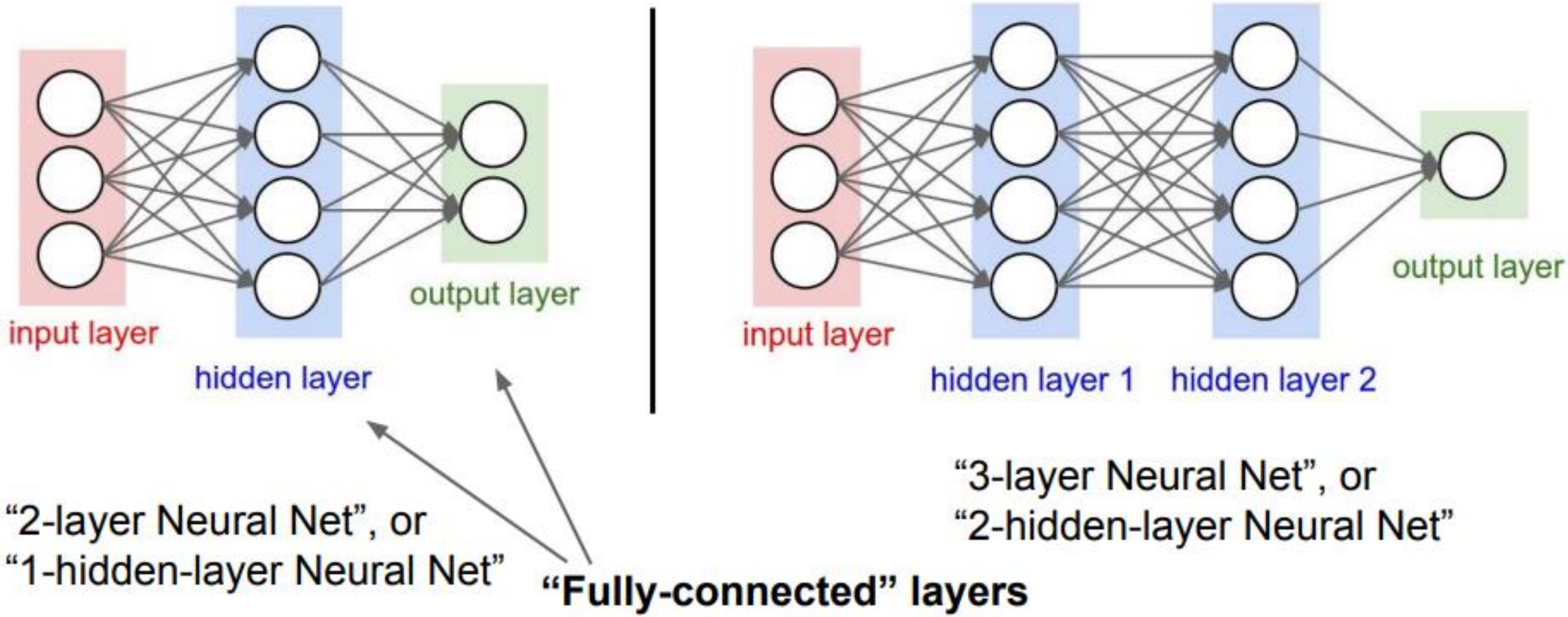
Network of formal neurons

- Feed-forward networks
  - No feedback connection
  - The output depends only on current input (No memory)
- Feedback or recurrent networks
  - Some internal feedback/backwards connection
  - All previous inputs can be the input (some memory system)
  - Tend to be used with temporal data or Natural Language Processing (NLP)



Vanilla RNN created by Raimi Karim

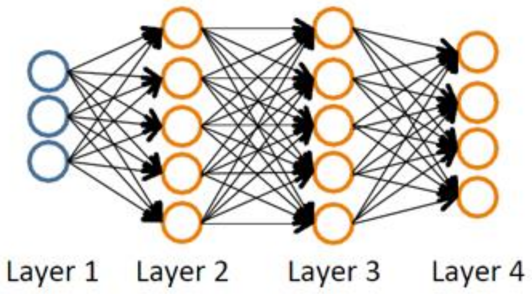
Networks architecture (1)



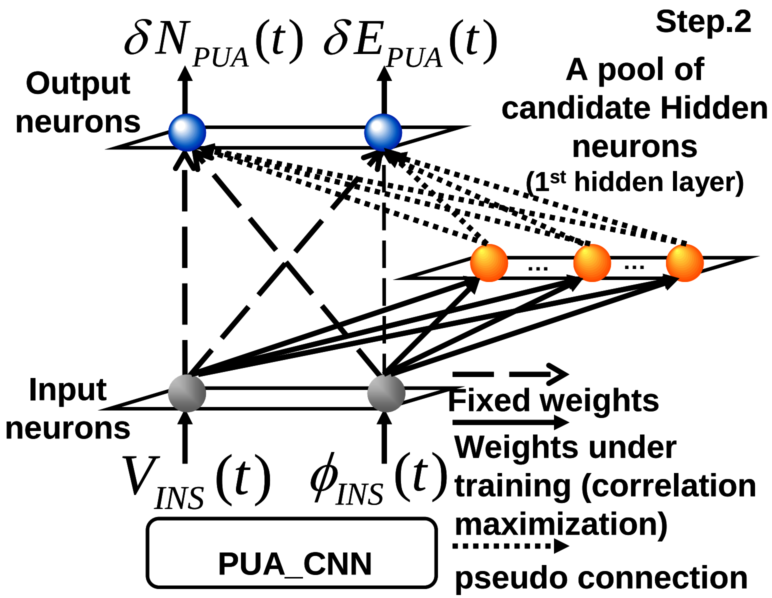
*Also called Multi-layer perceptron (MLP)

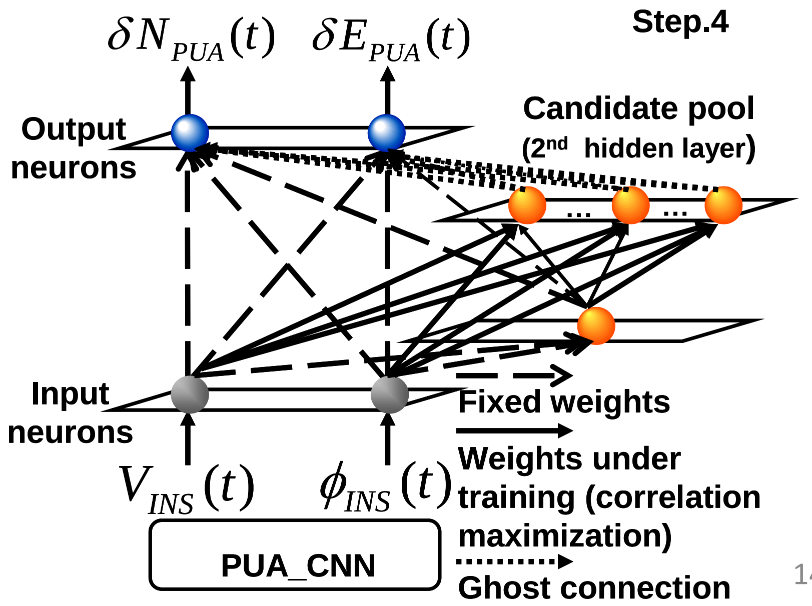
Networks architecture (2)

- Feed-forward NN



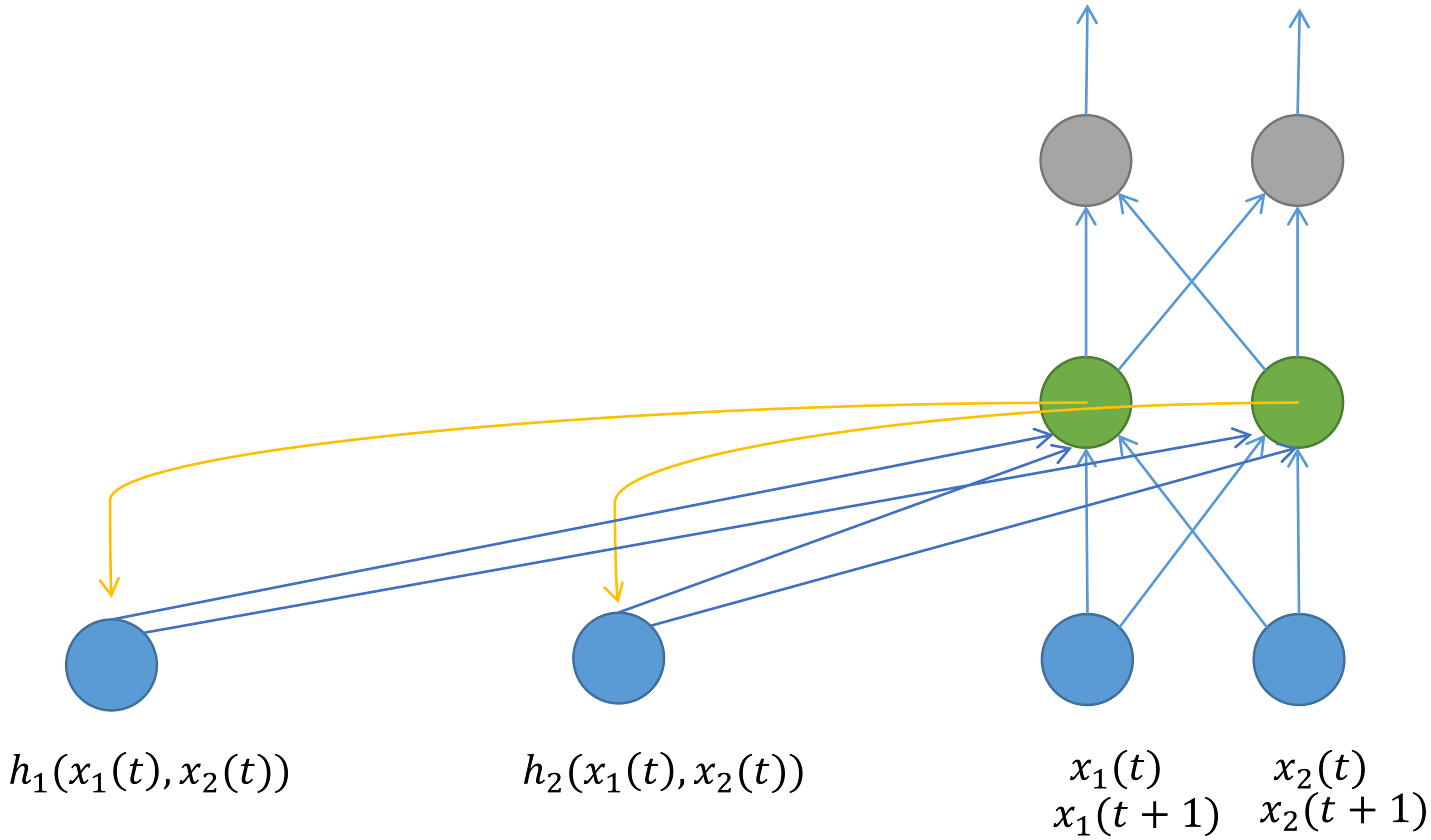
- Dynamic NN
  - Cascade correlation neural network





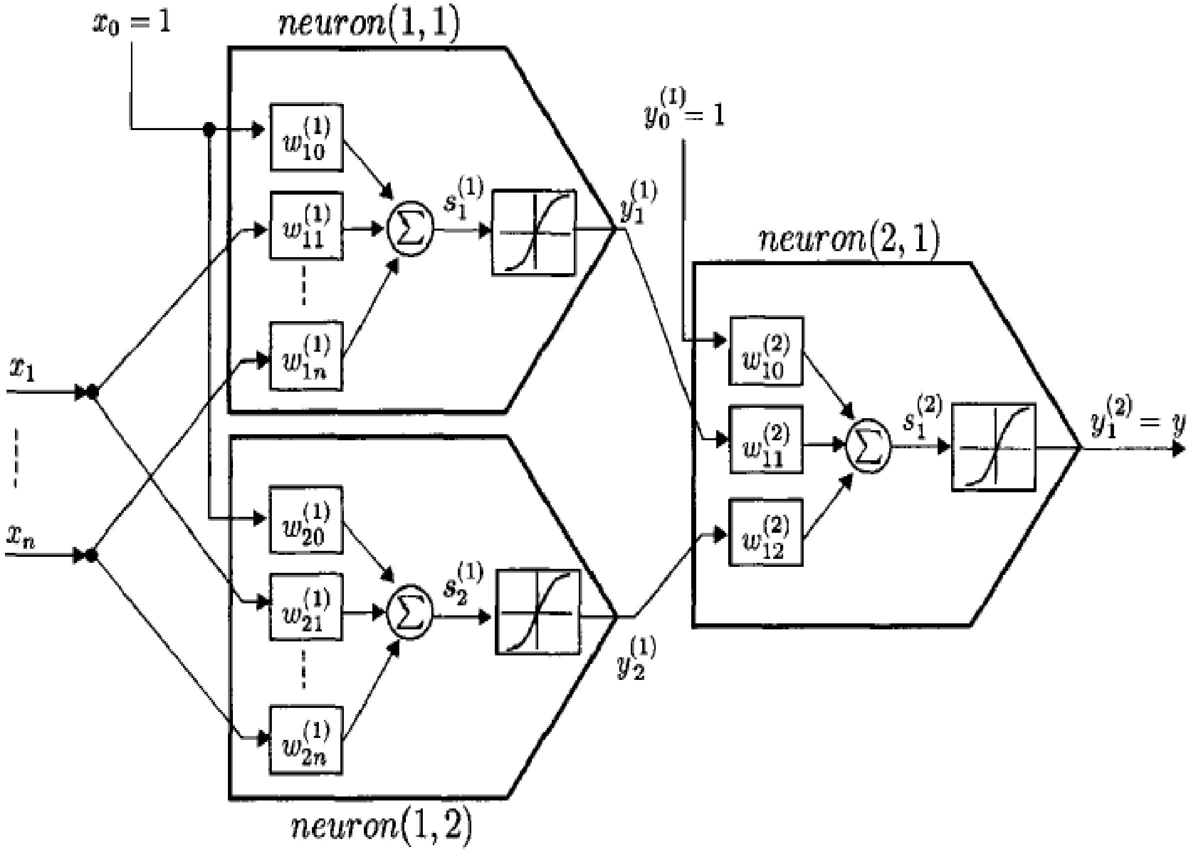
Networks architecture (3)

- Recurrent net architectures



Methodology for supervised training

Multiple neurons



- The derivatives of the logit, z, with respect to the inputs and the weights are very simple:

$$z = b + \sum_i x_i w_i$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

- The derivative of the output with respect to the logit is simple if you express it in terms of the output:

$$y = \frac{1}{1+e^{-z}}$$

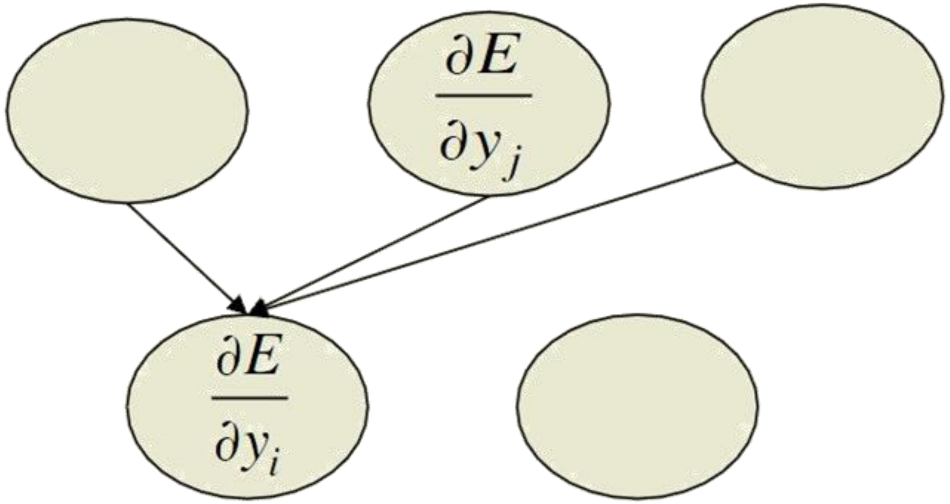
$$\frac{dy}{dz} = y\,(1-y)$$

Sketch of the backpropatation

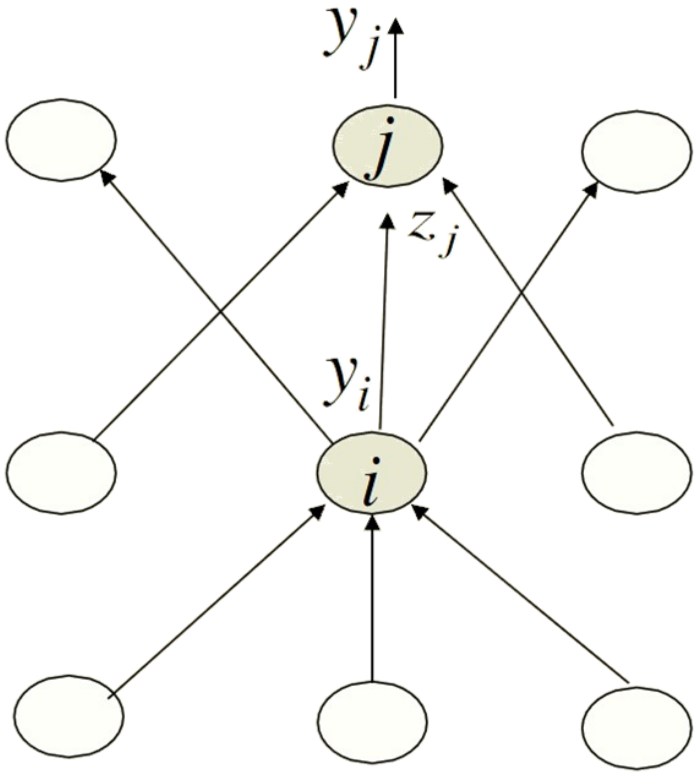
- First convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.
- Then use error derivatives *w.r.t.* activities to get error derivatives *w.r.t.* the incoming weights.

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$



Backpropagating



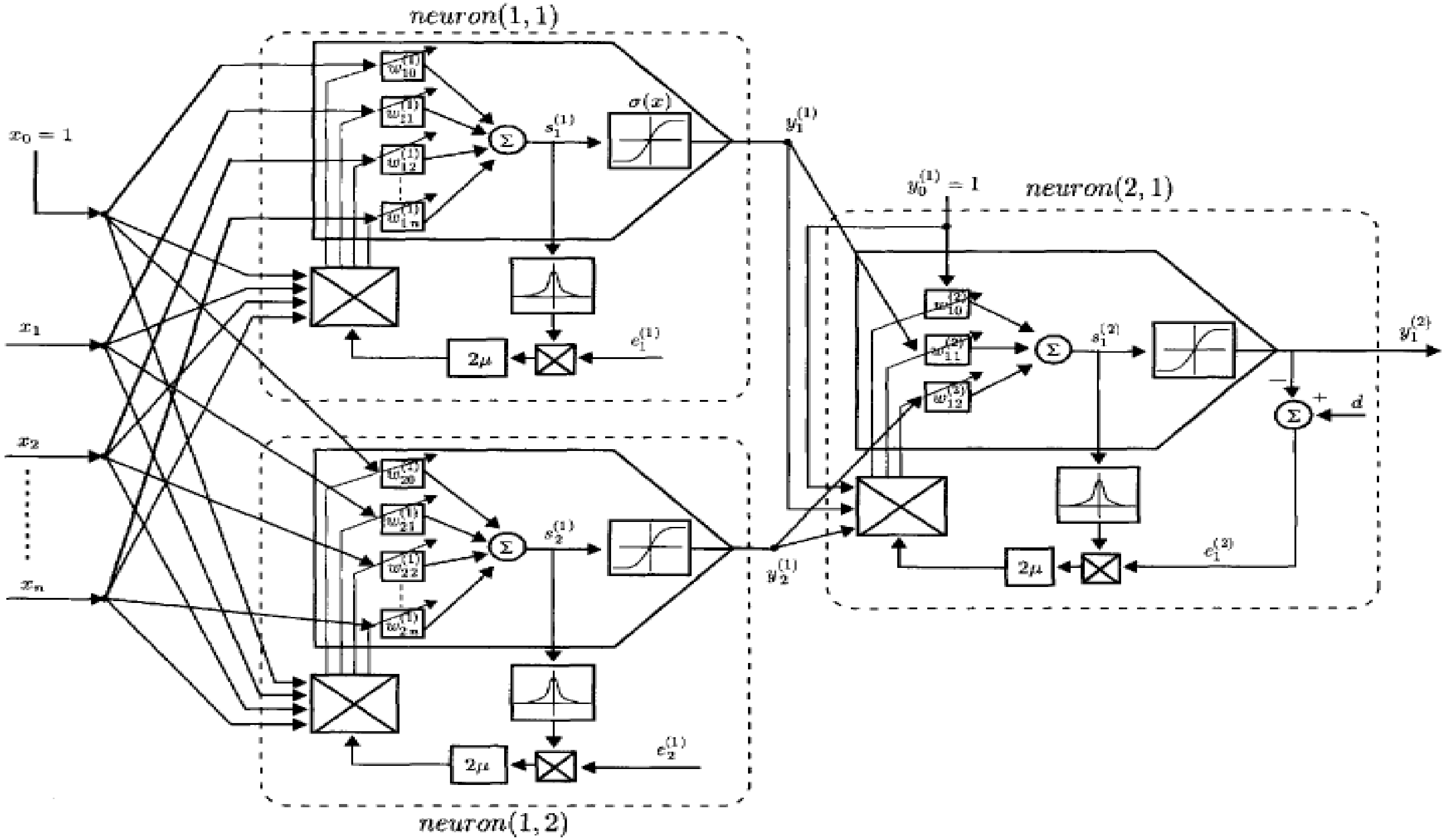
$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j\,(1-y_j)\,\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

5. Multi-layered feedforward neural networks

- Networks with multiple neurons
  - A two layered neural networks with parameter adoption



Multi-layered feedforward neural networks

- Networks with multiple neurons

$$
\begin{aligned}
\boldsymbol{w}_{a1}^{(1)}(k+1) &= \boldsymbol{w}_{a1}^{(1)}(k) - \mu \nabla_{\boldsymbol{w}_{a1}^{(1)}} \left(e^2(k) \right) \\
&= \boldsymbol{w}_{a1}^{(1)}(k) + 2\mu e_1^{(1)}(k) \sigma'(s_1^{(1)}(k)) \boldsymbol{x}_a(k)
\end{aligned}
$$

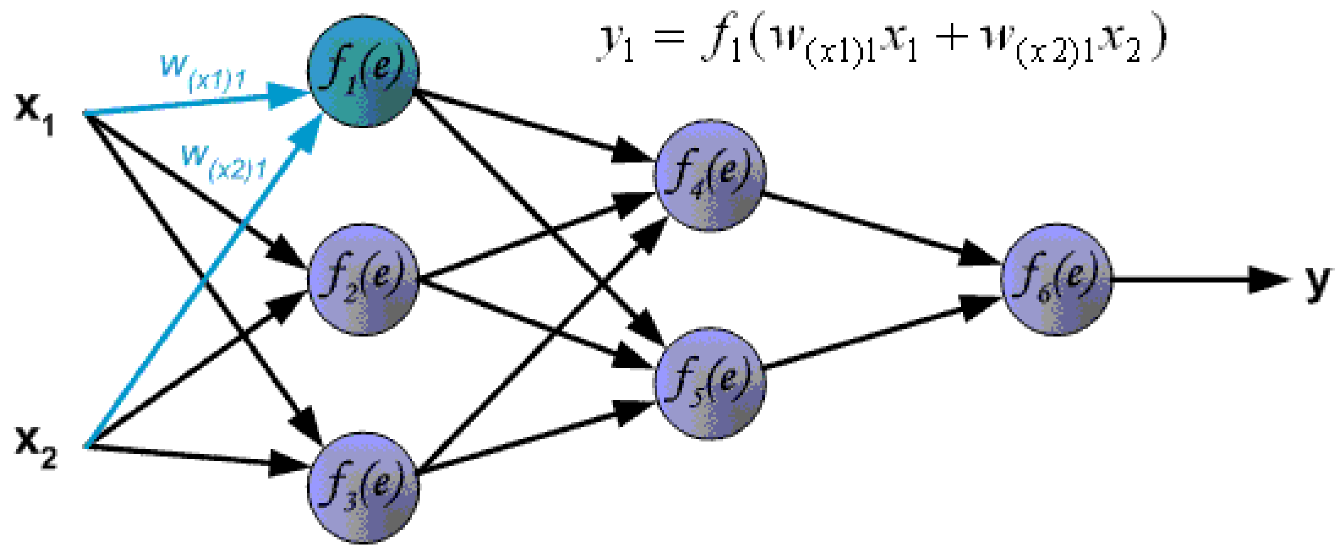
$$
\begin{aligned}
\boldsymbol{w}_{a2}^{(1)}(k+1) &= \boldsymbol{w}_{a2}^{(1)}(k) - \mu \nabla_{\boldsymbol{w}_{a2}^{(1)}} \left(e^2(k) \right) \\
&= \boldsymbol{w}_{a2}^{(1)}(k) + 2\mu e_2^{(1)}(k) \sigma'(s_2^{(1)}(k)) \boldsymbol{x}_a(k)
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{w}_{a1}^{(2)}(k+1) &= \boldsymbol{w}_{a1}^{(2)}(k) - \mu \nabla_{\boldsymbol{w}_{a1}^{(2)}} \left(e^2(k) \right) \\
&= \boldsymbol{w}_{a1}^{(2)}(k) + 2\mu e_1^{(2)}(k) \sigma'(s_1^{(2)}(k)) \boldsymbol{y}_a^{(1)}(k)
\end{aligned}
$$

$\mu>0$,learning rate parameter

Backpropagation

FP



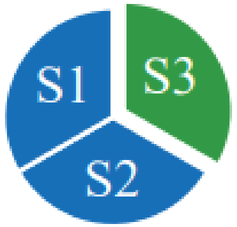
https://medium.datadriveninvestor.com/what-is-gradient-descent-intuitively-42f10dfb293f

Training set vs Test set

- Space of possible input values are usually infinite, and training set is only a FINITE subset
- Zero error on all training examples ≠ good results on whole space of possible inputs (generalization error ≠ empirical error…)
- Need to collect enough and representative examples (very critical in deep neural network)
- Essential to keep aside a subset of examples that shall be used only as TEST SET for estimating final generalization (when training finished)
- Need also to use some “validation set” independent from training set, in order to tune all hyper-parameters (layer sizes, number of iterations, etc…)

Optimize hyper-parameters by Validation

- To avoid over-fitting and maximize generalization, absolutely essential to use some VALIDATION estimation, for optimizing training hyper-parameters (and stopping criterion):
  - either use a separate validation dataset (random split of data into Training-set + Validation-set)
  - or use CROSS-VALIDATION:
    - Repeat k times: train on (k-1)/k proportion of data + estimate error on remaining 1/k portion
    - Average the k error estimations



3-fold cross-validation:

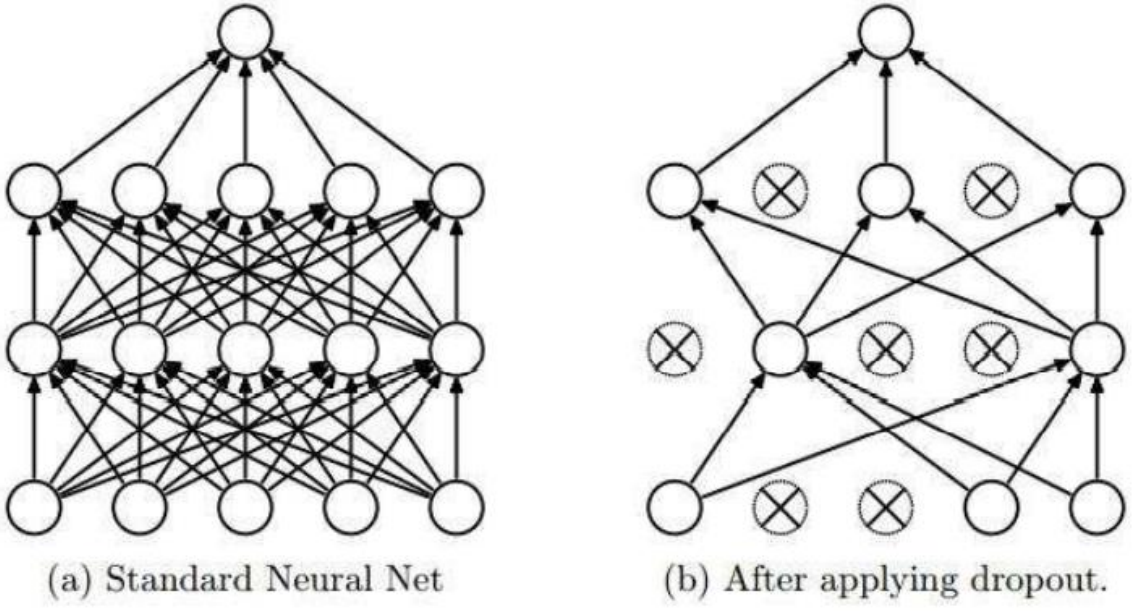
- Train on S1∪S2 then estimate errS3 error on S3
- Train on S1∪S3 then estimate errS2 error on S2
- Train on S2∪S3 then estimate errS1 error on S1
- Average validation error: (errS1+errS2+errS3)/3

Some Neural Networks training "tricks"

- Importance of input normalization
  - zero mean, unit variance
- Importance of weights initialization
  - random but SMALL and prop. to 1/sqrt(nb Inputs)
- Decreasing (or adaptive) learning rate
- Importance of training set size
  - If a Neural Net has a LARGE number of free parameters

  ➔train it with a sufficiently large training-set!
- Avoid overfitting by Early Stopping of training iterations
- Avoid overfitting by use of L1 or L2 regularization
- For ConvNet (or network with huge amount of training parameters)
  - Dropout technique to avoid overfitting
  - Batch normalization

Dropout



Dropout is proposed by Hinton et al., 2012, as a regularizer which randomly sets half of the activations to the fully connected layers to zero during training

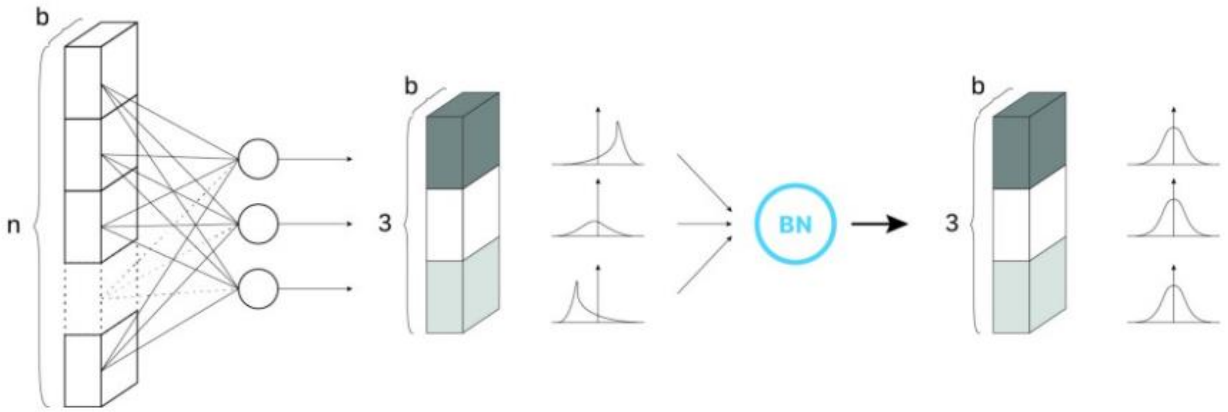
At each training stage, individual nodes can be temporarily "dropped out" of the net with probability p (usually ~0.5), or re-installed with last values of weights

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov (2012). Improving neural networks by preventing co-adaptation of feature detectors

Batch normalization (BN)

- Each layer of a NN has inputs with a corresponding distribution, which is affected during the training process by the randomness in the parameters initialization and input data ➔ internal covariate shift
- It is believed that NN with BN can use higher learning rate without vanishing or exploding gradients.
- It seems to have a regularizing effect and thus unnecessary to use dropout after.

[Szegedy, Christian (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“]



Given four trainable parameters $\mu_B, \sigma_B, \gamma, \beta$ (mean, variance, arbitrary scale, arbitrary bias, respectively). Use min-batch (B) with size m, input d-dimensional $x = \left[x^{(1)}, \dots x^{(d)}\right]$

Step1: : $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \;\; and \; \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$

Step2: normalized input

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)2} + \epsilon}}, \text{ where } k \in [1, d], and \; i \in [1, m]$$

Step3: Output of current layer

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

Step4: backpropagation with chain rule

(Optional) BN limitation

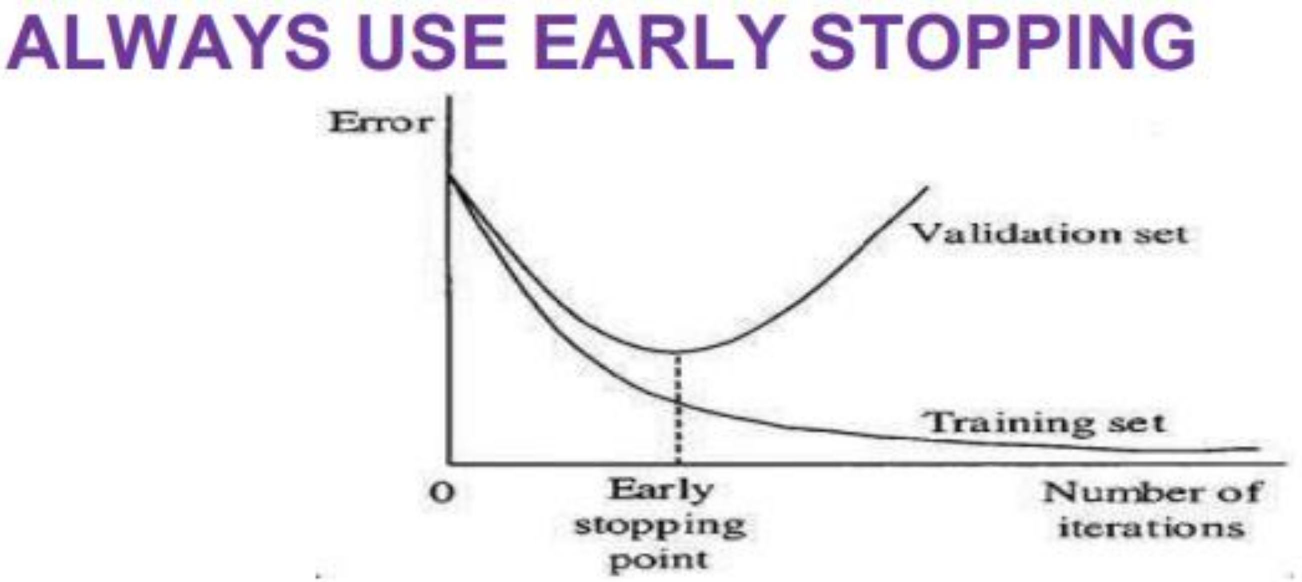
- BN has shown great results and has been used heavily in tons of SOTA methods. However, BN has the following concerns
  - Very expensive computational costs
  - Introduces a lot of extra hyper-parameters that need further fine-tuning
  - Causes a lot of implementation errors in distributed training
  - Performs poorly on small batch sizes, which are used often in training larger models
- Normalized free networks (NFNets, 2021) reaches ImageNet top1 accuracy 86.5% and 8.7 times faster than EfficientNET
  - without the use of BN
  - New idea of Adaptive Gradient Clipping (AGC) ➔ Gradient clipping with a way to automatically calculate the threshold hyperparameter based on the premise:
    - *the unit-wise ratio of the norm of the gradients to the norm of the weights of a layer provides a simple measure of how much a single gradient descent step will change the original weights.*
  - Sharpness-Aware Minimization (SAM) [Foret et al. 2012]

Brock, A., De, S., Smith, S. L., & Simonyan, K. (2021). *High-Performance Large-Scale Image Recognition Without Normalization*.

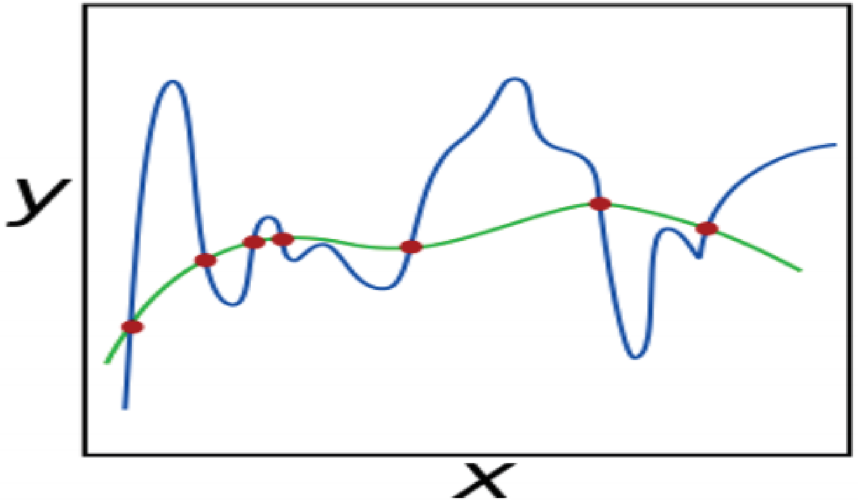
Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-aware minimization for efficiently improv- ing generalization. In 9th International Conference on Learning Representations, ICLR, 2021

Early stopping

- For Neural Networks, a first method to avoid overfitting is to STOP LEARNING iterations as soon as the validation_error stops decreasing
- Generally, not a good idea to decide the number of iterations beforehand. Better to <u>ALWAYS USE EARLY STOPPING</u>



Regularization penalty



Trying to fit too many free parameters with not enough information can lead to overfitting

Regularization = penalizing too complex models
Often done by adding a special term to cost function

For neural network, the regularization term is just norm L2 or L1 of vector of all weights:

$$K = \Sigma_m(\text{loss}(Y_m,D_m)) + \beta\, \Sigma_{ij} |W_{ij}|^p \quad \text{with } p=2 \text{ (L2) or } p=1 \text{ (L1)}$$

→ **name "Weight decay"**