Introduction to deep learning in computational biology

María Rodríguez Martínez

Technical Lead Systems Biology, IBM, Zürich Research Lab

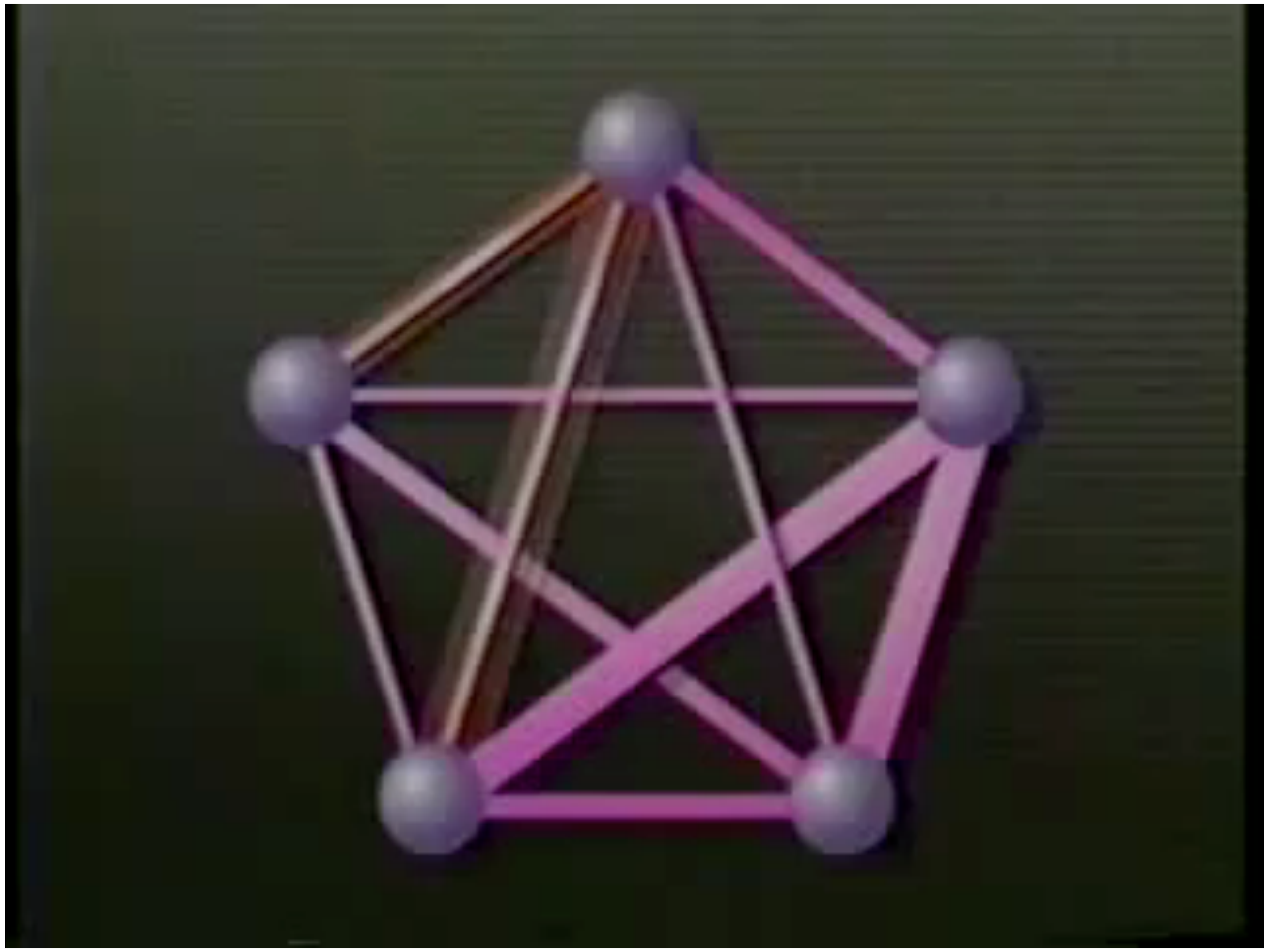
Overview

- Introduction to deep learning
  - History and motivation
  - Activations functions
  - Cost functions
  - Backpropagation
  - Regularization
  - Optimization
- Multi-Layer Perceptron (MLP)
- Auto-enconders (AE)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)

Overview

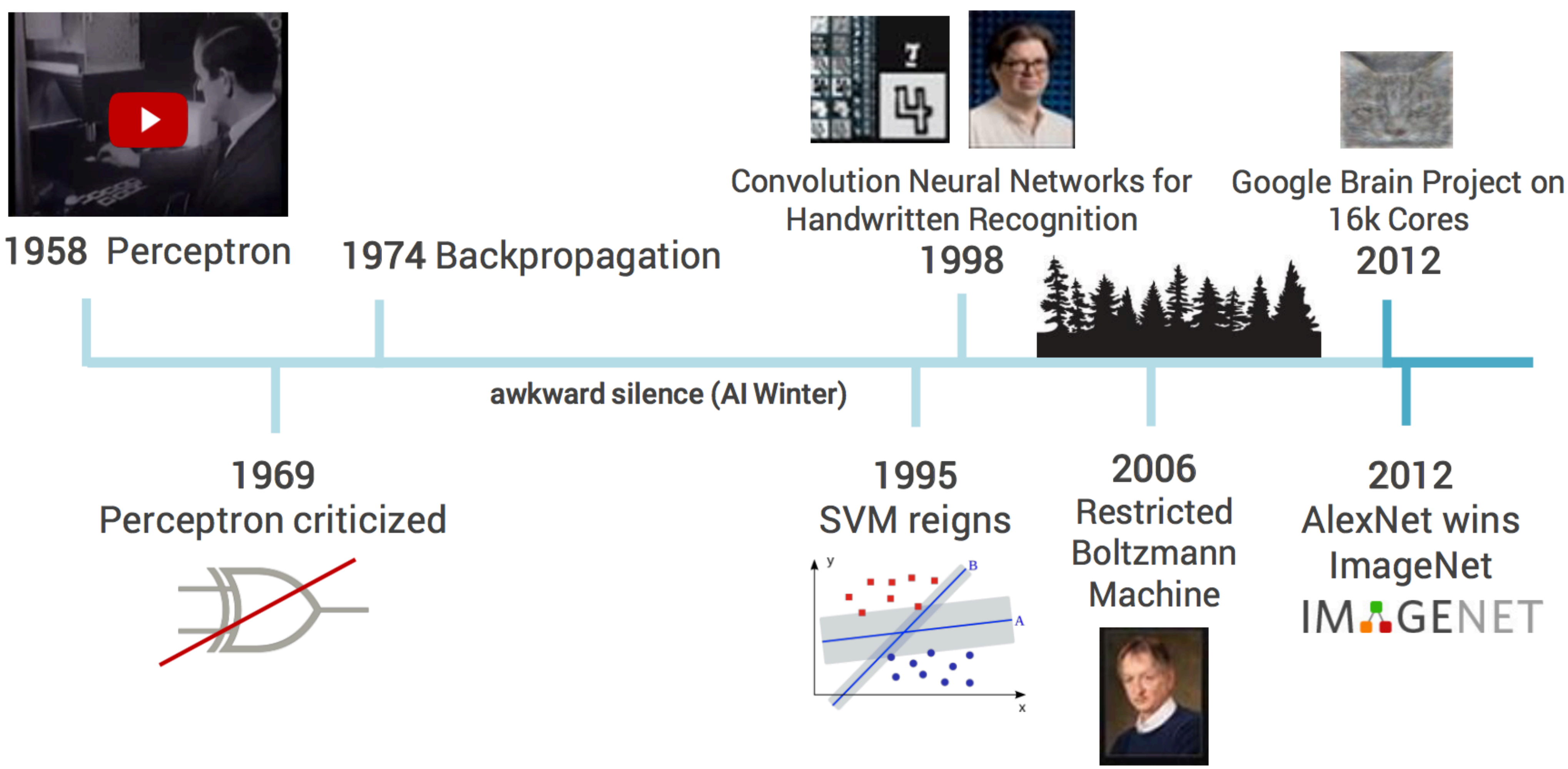
- Introduction to deep learning
  - History and motivation
  - Activations functions
  - Cost functions
  - Backpropagation
  - Regularization
  - Optimization
- Multi-Layer Perceptron (MLP)
- Auto-enconders (AE)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)

The beginnings: perceptron.



https://www.youtube.com/watch?v=cNxadbrN_aI

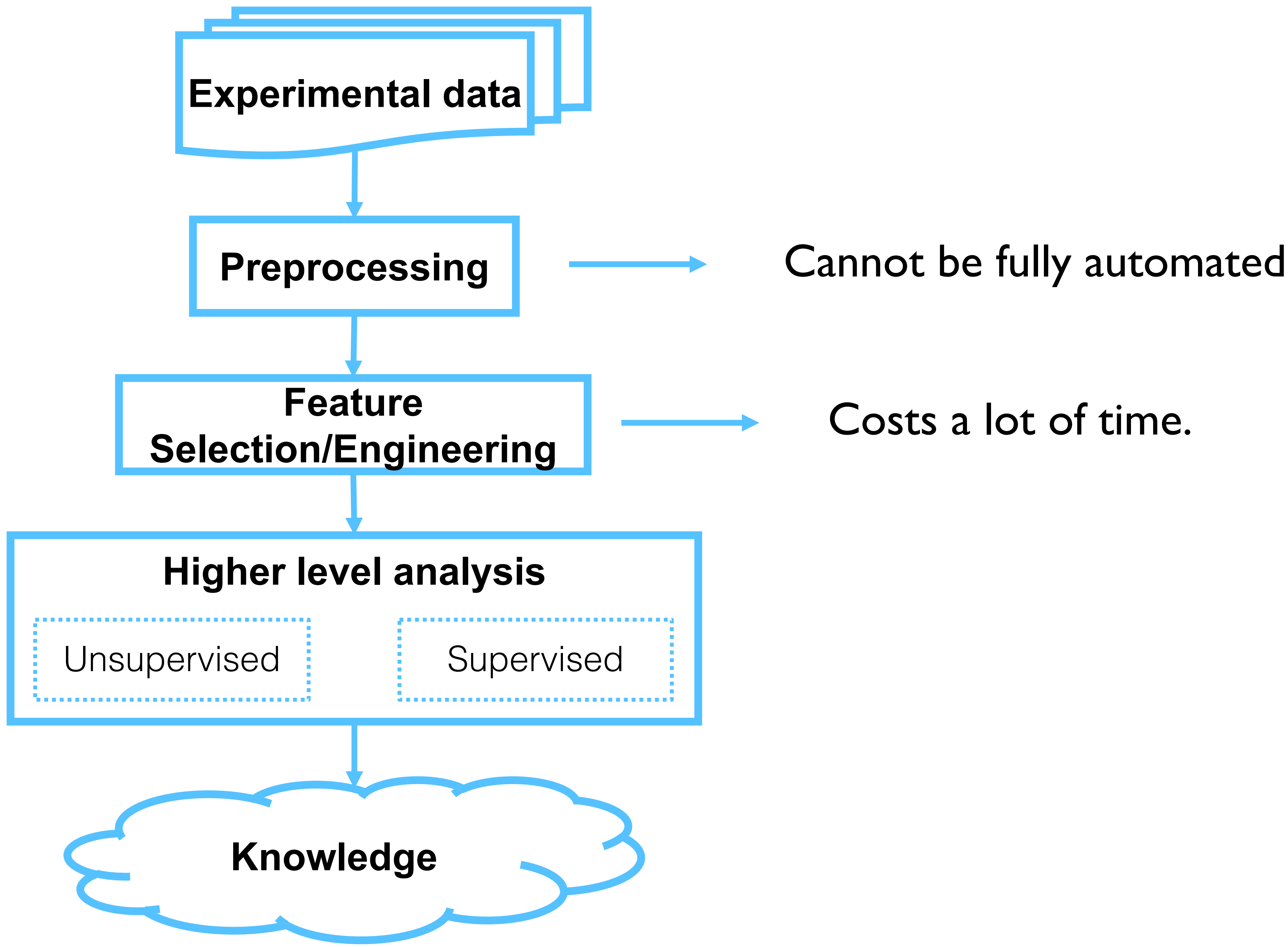
A brief history of deep learning



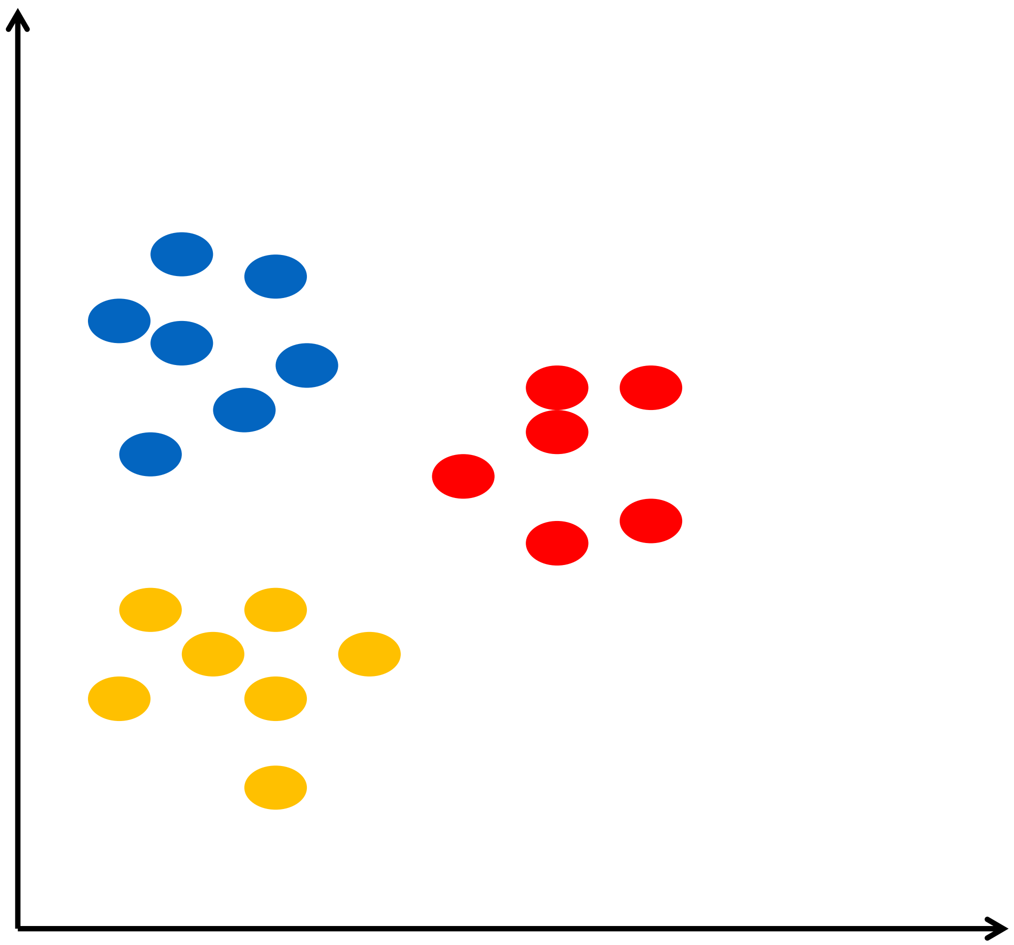
https://www.slideshare.net/LuMa921/deep-learning-a-visual-introduction

Machine learning

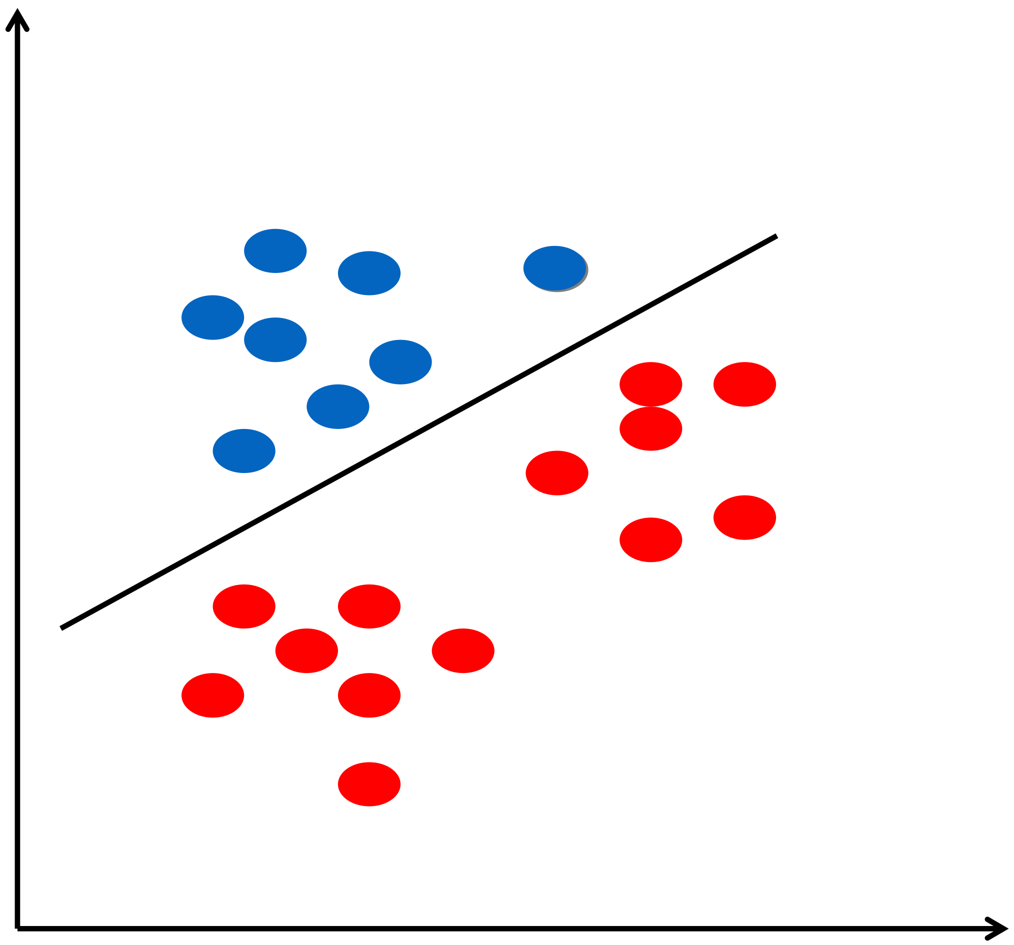
Machine Learning is a type of Artificial Intelligence that provides computers with the ability to learn without being explicitly programmed.



Learning approaches



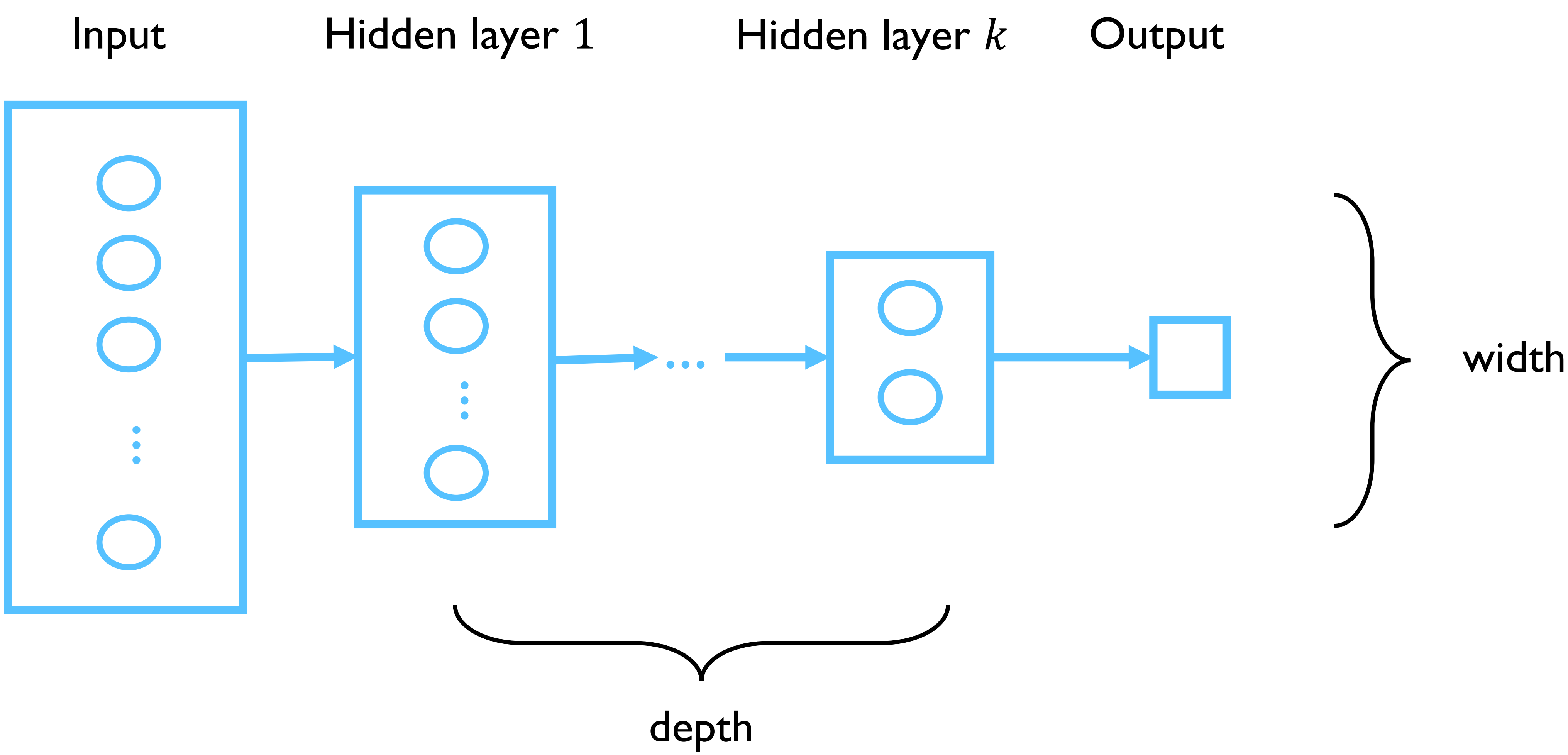
Dimensionality reduction: e.g. PCA, tSNE
Clustering: e.g. Phenograph, FlowSOM



Classification: SVMs, Random Forests

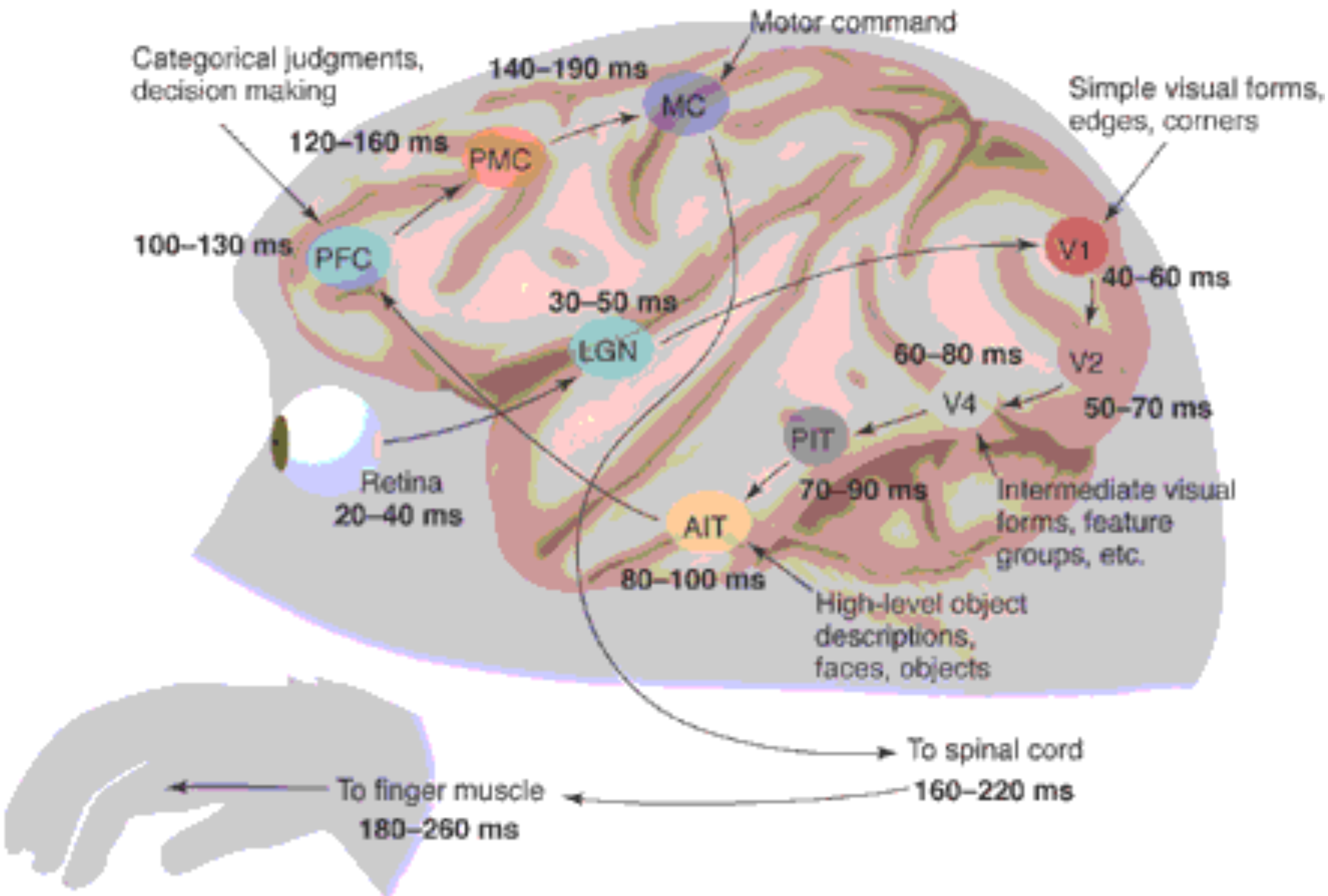
Supervised Learning: Learning with a labeled training set.
E.g. email spam detector with training set of already labeled emails.
Unsupervised Learning: Discovering patterns in unlabeled data.
E.g. cluster similar documents based on the text content .
Reinforcement Learning: learning based on feedback or reward.
E.g. learn to play chess by winning or losing.

Neural networks



- Learn data representations. Exceptional effective at learning patterns.
- Use a hierarchy of layers that mimic the neural networks of our brain.
- Can learn highly complex patterns if sufficient data is available for training.

The mammalian visual cortex is hierarchical

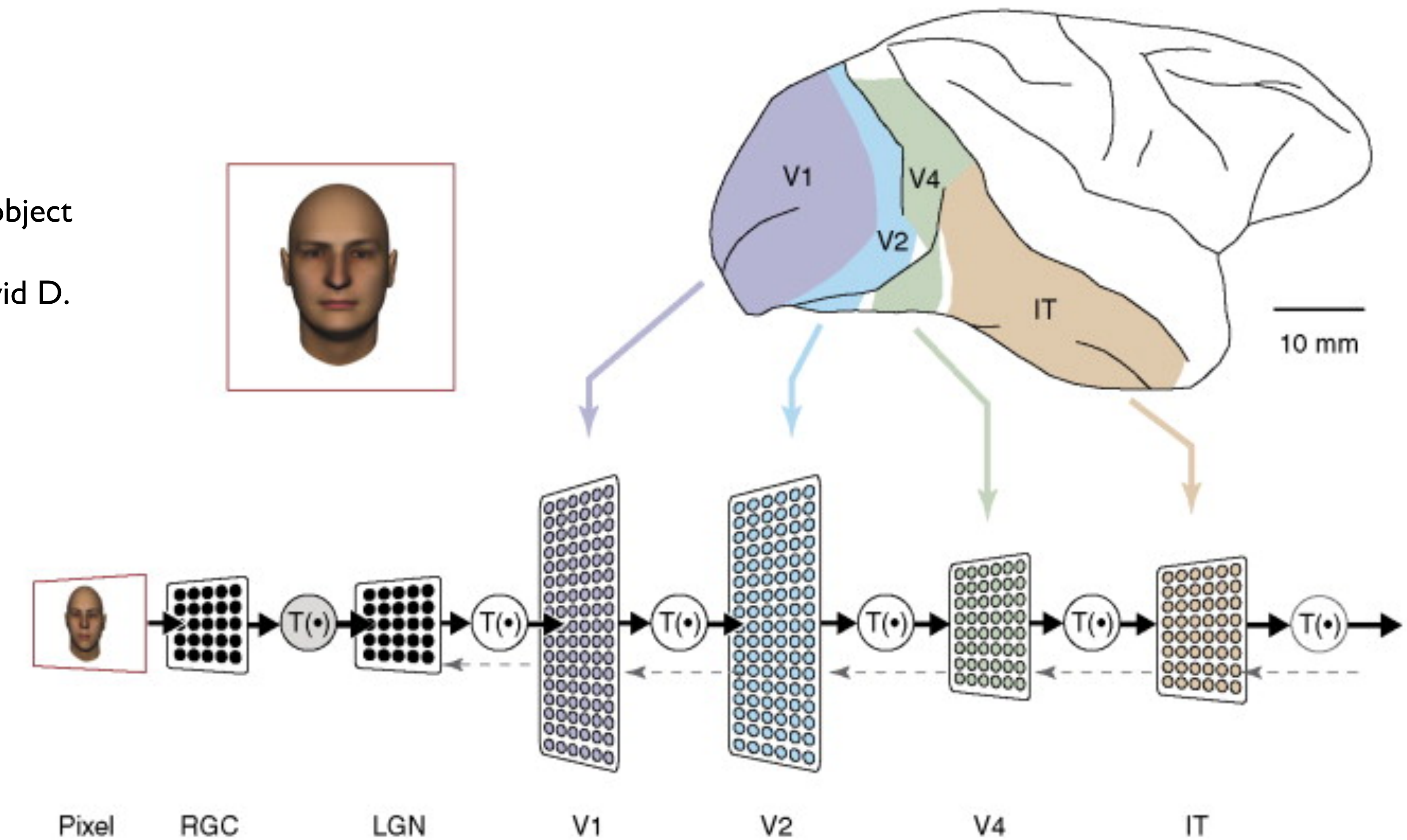


Simon J. Thorpe, Michèle Fabre-Thorpe, *Science* 2001

- First hierarchy of neurons are sensitive to edges.
- Brain regions further down the visual pipeline are sensitive to more complex structures (e.g. faces).
- The strength of the connections between neurons represents long term knowledge.

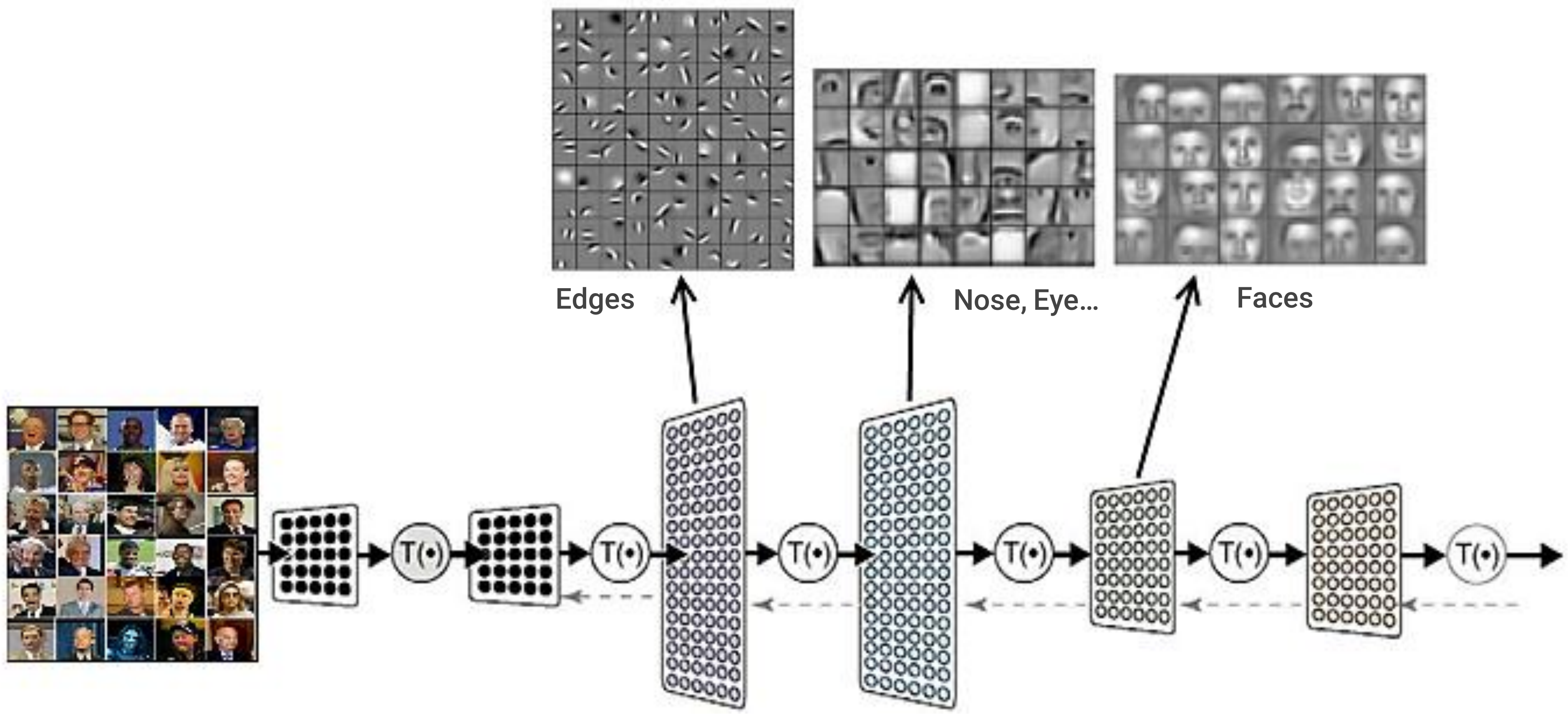
DNNs mimic the neuronal hierarchical connectivity.

Untangling invariant object recognition
James J. DiCarlo, David D. Cox, 2007



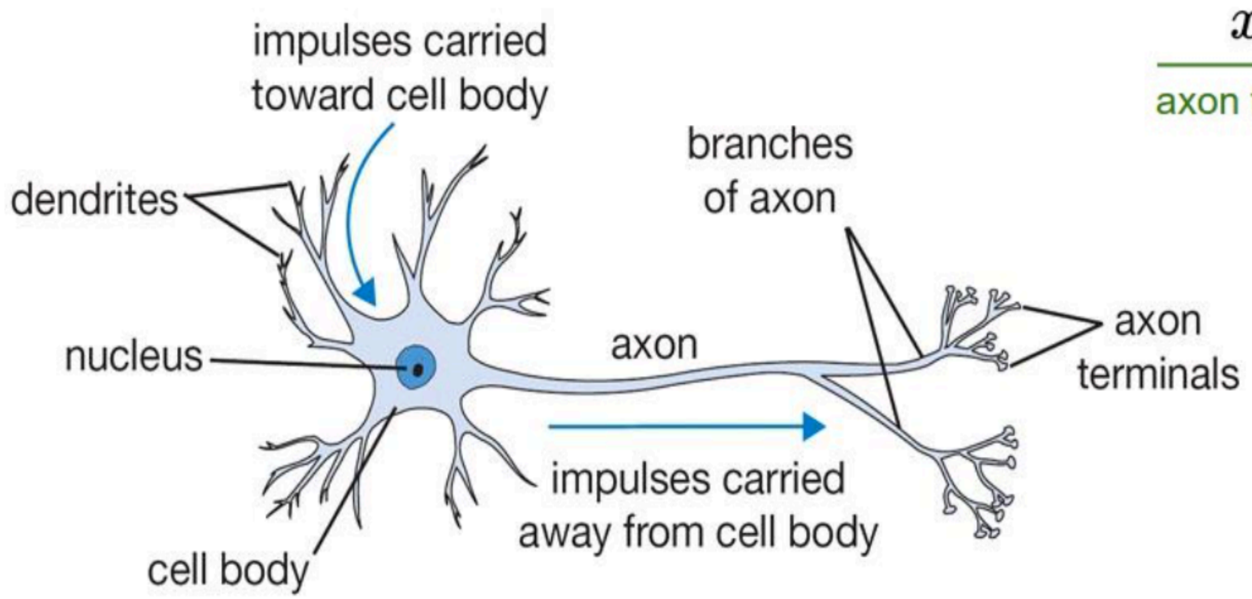
- Deep neural networks (DNNs) consists of a hierarchy of layers.
- Each layer transforms the input data into more abstract representations:
  e.g. edge -> nose -> face.
- The output layer combines those features to make predictions.

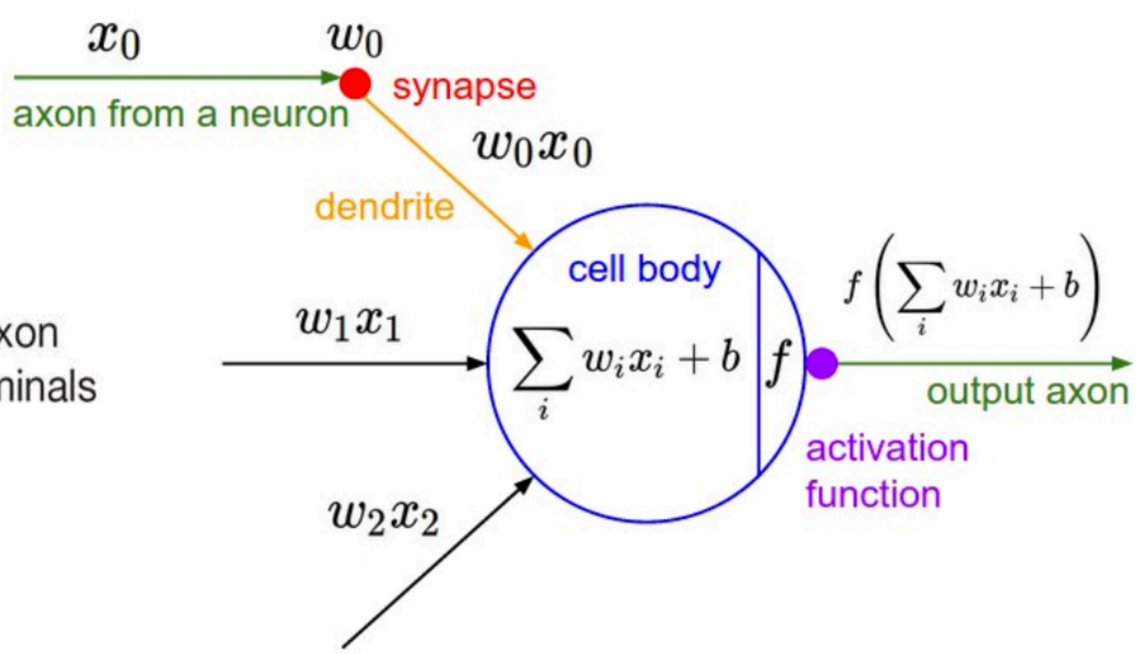
DNNs mimic the neuronal hierarchical connectivity.



- Deep neural networks (DNNs) consists of a hierarchy of layers.
- Each layer transforms the input data into more abstract representations:
  e.g. edge -> nose -> face.
- The output layer combines those features to make predictions.

Biological vs. artificial neurons

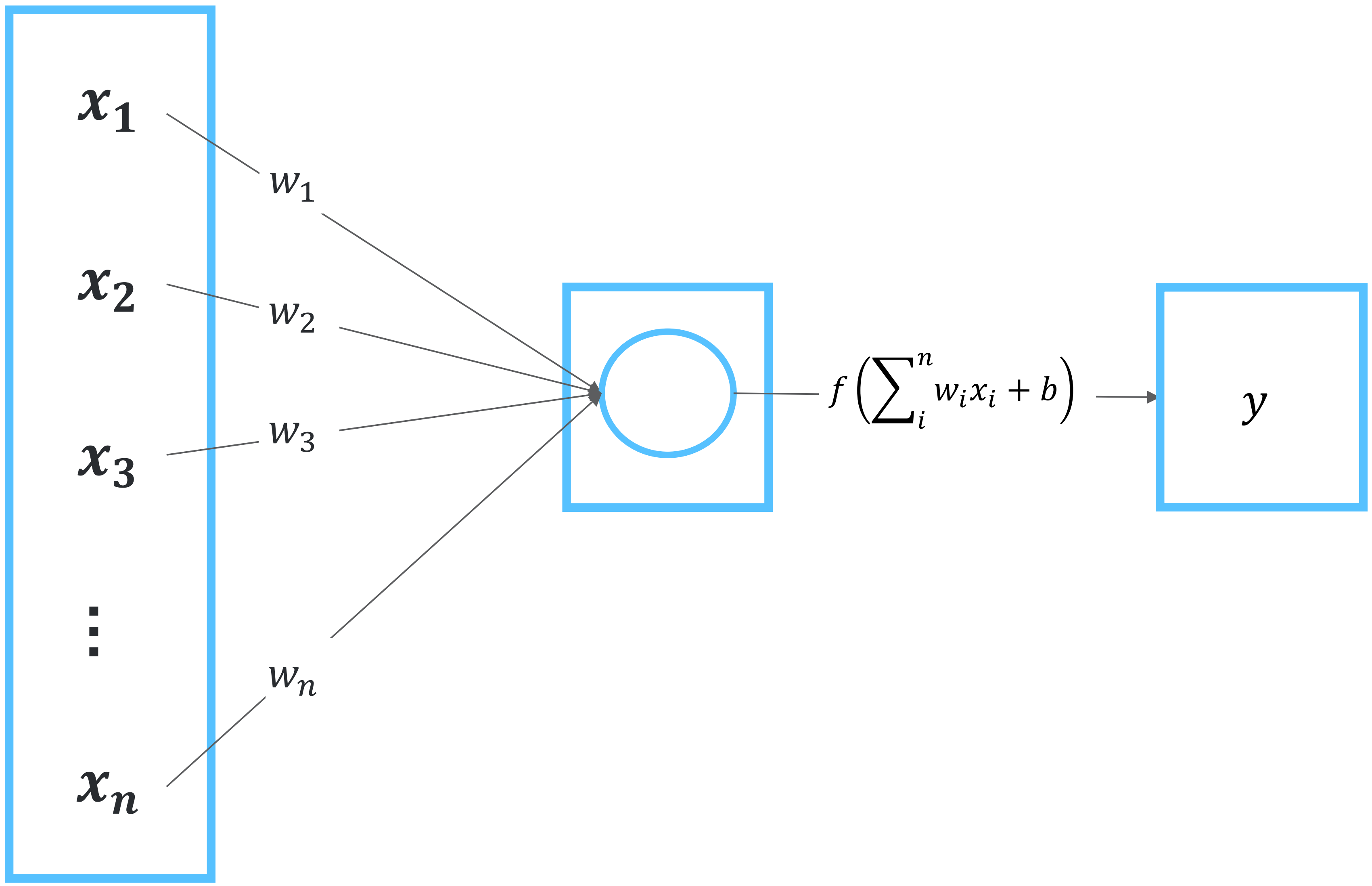




http://cs231n.github.io/neural-networks-1/

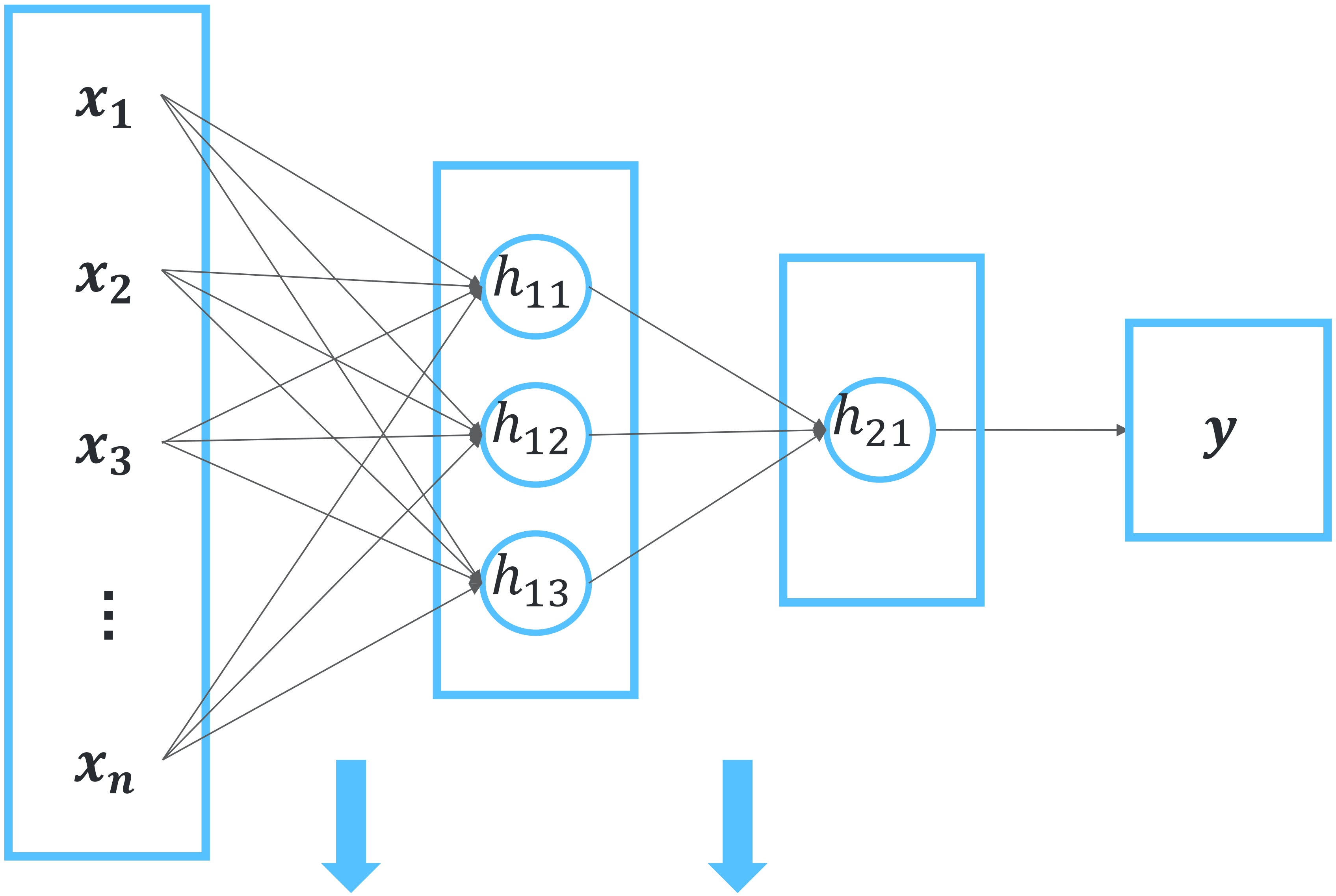
- Neurons filter and detect specific features or patterns (e.g. edge, nose) by receiving a weighted input, transforming it with the activation function and passing it to the outgoing connections.
  - Each neuron performs a dot product with the input and its weights, adds the bias and applies the activation function.
- Artificial neurons mimic brain neurons.

Simplest neural network



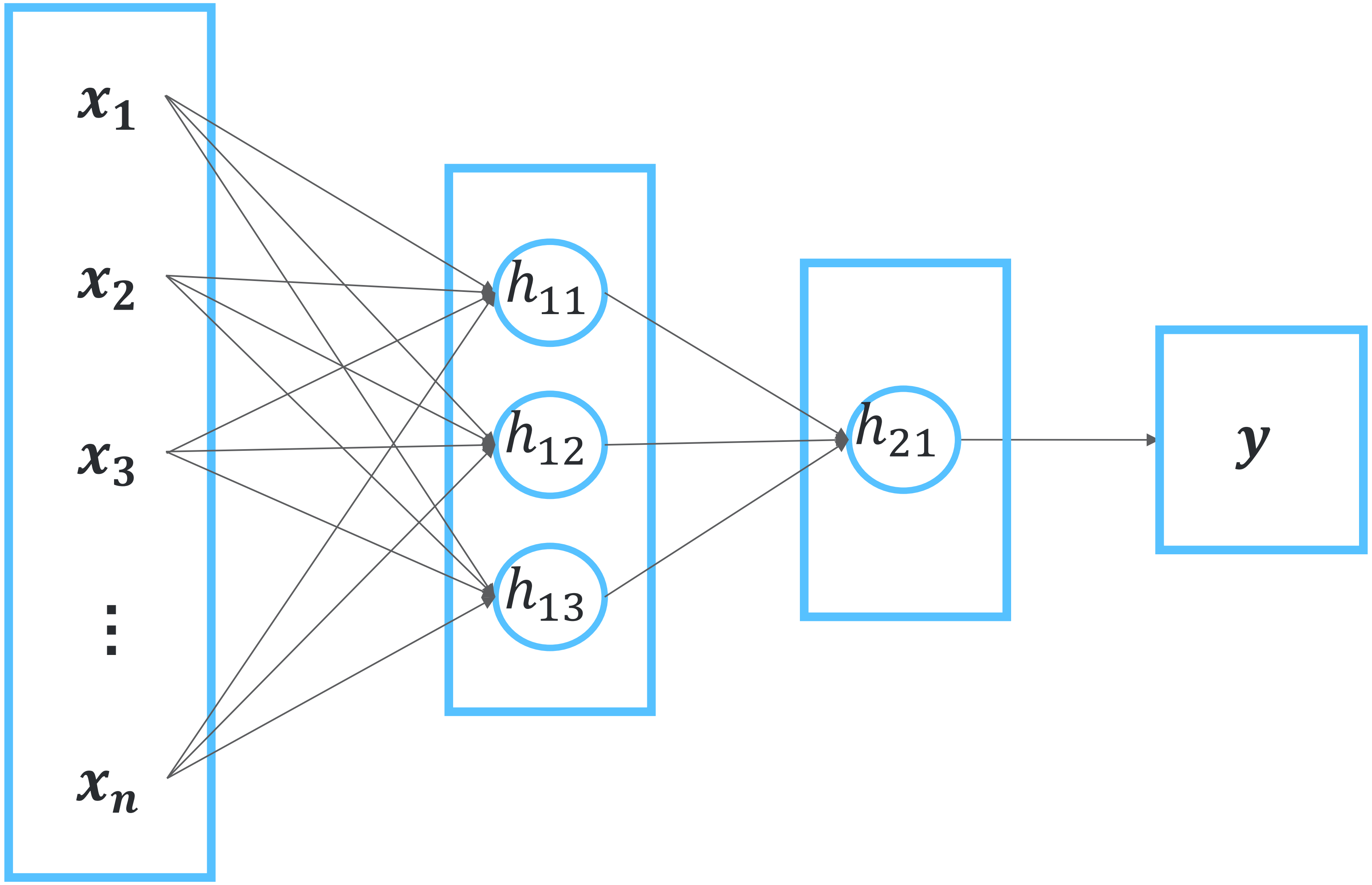
- Weights and biases are the learnable parameters.
- **Weight**: controls the strength of the connection. Weights near zero mean changing this input will not change the output.
- **Bias**: measure of how easy it is to get a node to fire. A node with a large bias will tend produce large positive outputs.

A more realistic example



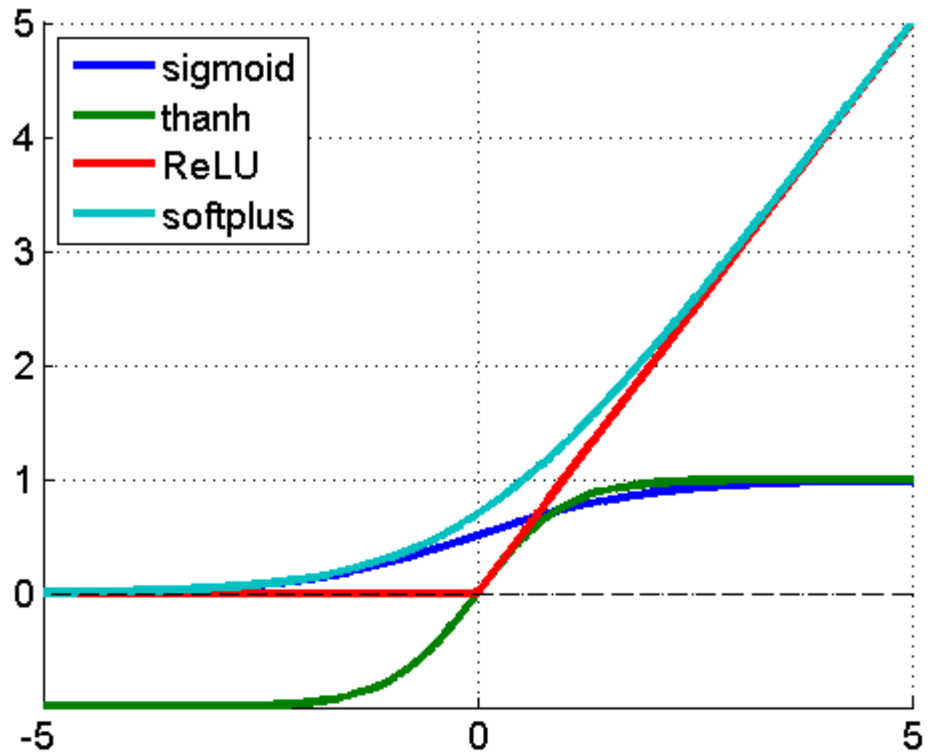
$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ w_{n1}^1 & w_{n2}^1 & w_{n3}^1 \end{bmatrix}, b = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix} \qquad W^2 = \begin{bmatrix} w_{11}^2 \\ w_{12}^2 \\ w_{13}^2 \end{bmatrix}, b = [b_1^2]$$

A more realistic example



- Each DNN consists of one input, one output and multiple fully-connected hidden layers in between.
- Each layer is represented as a series of neurons that progressively extract higher-level features of the input until the final layer makes a decision about what the input shows.
- The more layers the network has, the more abstract features it can learn.

Commonly used activation functions



https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearities/

$$\mathrm{Sigmoid(x)} = \frac{1}{1 + e^{-x}}$$

$$\mathrm{Tanh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

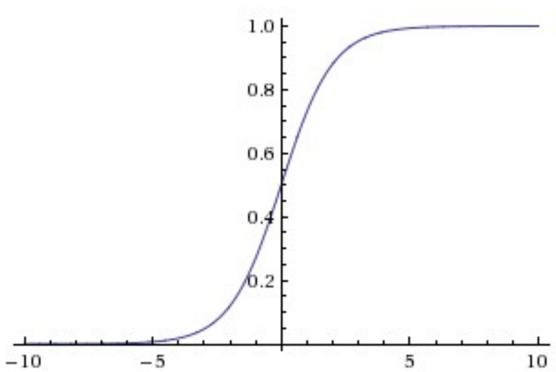
$$\mathrm{ReLU(x)} = \max(0, \mathrm{x})$$

$$\mathrm{Softplus(x)} = \log(1 + e^{x})$$

- Activations functions are non-linear. Non-linearity is needed to learn complex representations of data, otherwise the DNN would be just a linear function (analogous to PCA).
- Most deep networks use ReLU in hidden layers:
  - it trains much faster (constant derivative),
  - improve discriminative performance,
  - prevents the gradient vanishing problem.

Activation functions

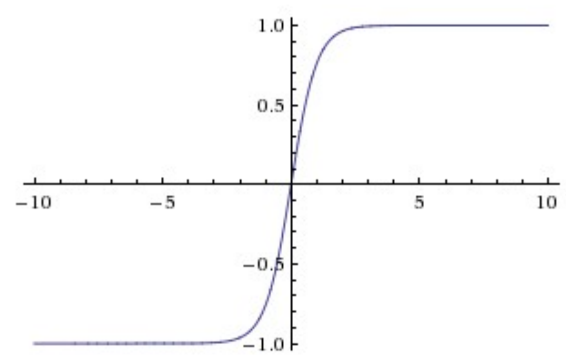
Sigmoid



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

+	Resembles neuronal firing
—	Saturation → zero gradients
—	Sigmoid outputs are not zero-centered

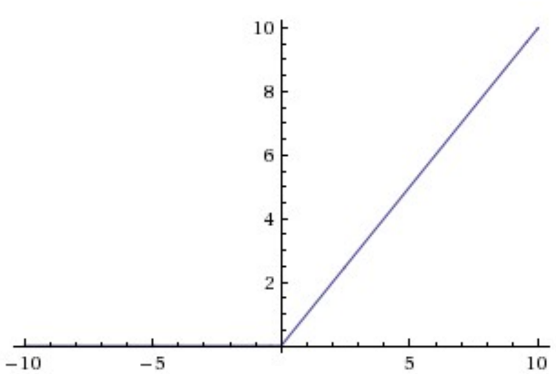
Hyperbolic Tangent (tanh)



$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} = 2\sigma(2x) - 1$$

+	Outputs are zero-centered
—	Saturation → zero gradients

Rectified Linear Unit



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

+	Cheap operation
+	Accelerates convergence
—	Large gradients → Dying ReLUs

Additional activation functions

- **Leaky ReLUs**: solves the dying ReLUs issue, results are not consistent
- **Maxout:** generalization of ReLUs, no saturation, no dying, very expensive to compute

Examples of cost functions

- A cost function measures how well a neural network predicts the expected outputs given the training samples.
- A cost function is single valued function:

$$C_i(W, B, S_i, O_i)$$

Weights Biases Sample i Expected output of sample i

- **Cost function requirements:**
  - The cost function C must be able to be written as an average over individual training samples:

$$C(W, B, S, O) = \frac{1}{n}\sum_{i=1}^{n} C_i(W, B, S_i, O_i)$$

  - The cost function C must not depend on any network activation value besides the activation value of the output layer, a_j^L.

- $C(W, B, S, O) \approx 0$ means the DNN is well trained.

Cost functions

- **Mean square error** (aka maximum likelihood and sum squared error):

$$C(W, B, S, O) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(a_j^L - O_{ij}\right)^2$$

- **Cross-entropy** (aka Bernoulli negative log-likelihood and binary cross-entropy):

$$C(W, B, S, O) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\left[O_{ij}\log a_j^L + \left(1 - O_{ij}\right)\log\left(1 - a_j^L\right)\right]$$

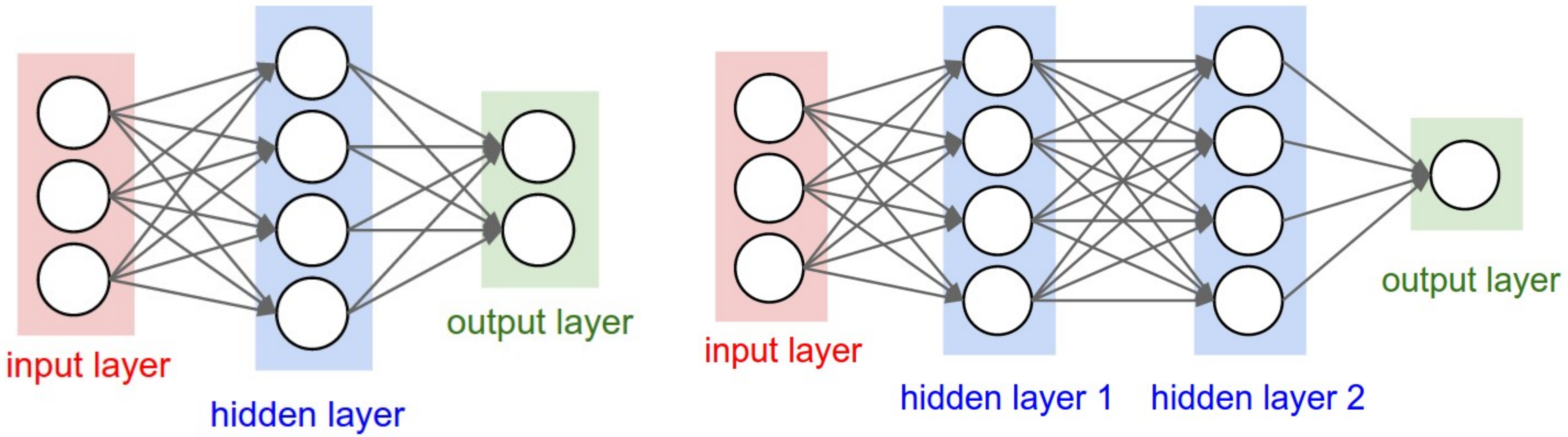
- **Kullback–Leibler divergence** (aka information divergence, information gain and relative entropy):

$$C(W, B, S, O) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} O_{ij}\log\frac{O_{ij}}{a_j^L}$$

- **Hellinguer distance**

$$C(W, B, S, O) = \frac{1}{n\sqrt{2}}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(\sqrt{a_j^L} - \sqrt{O_{ij}}\right)^2$$

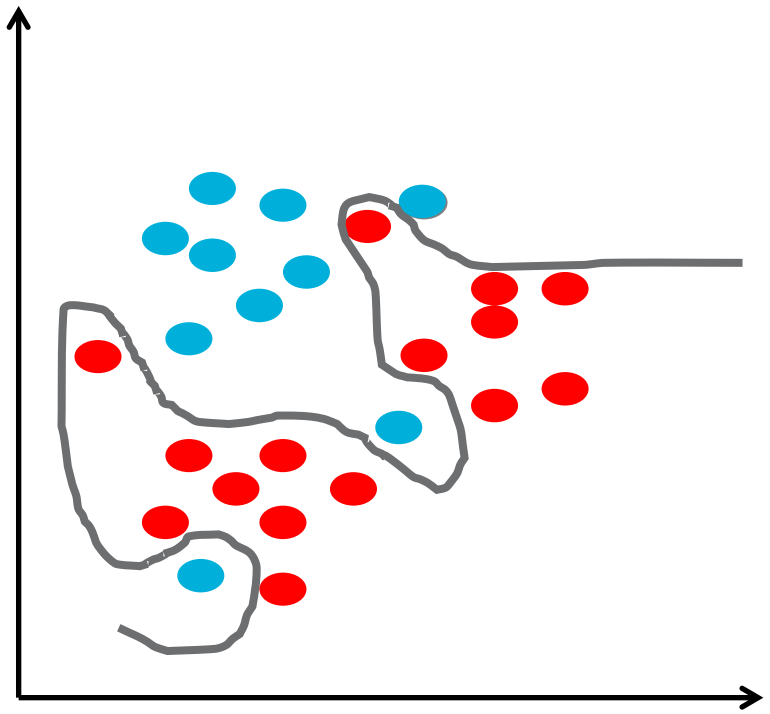
The importance of network architecture

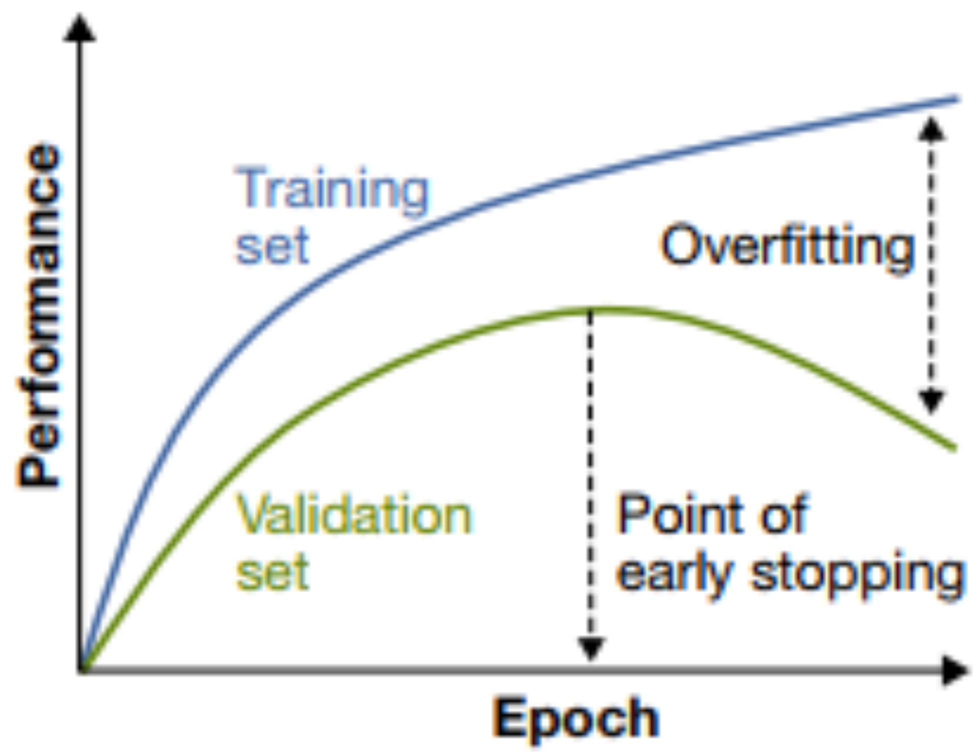


- The capacity of a network can be increased with the number of layers and units per layer.
- As a rule of thumb, going deeper results in more expressive networks, while going wider may lead to overfitting
    - more layers lead to more nested functions and non-linearities that increase the abstraction power, while more units in the same layer usually add features of the same complexity, which might lead to redundancy.

http://cs231n.github.io/neural-networks-1/#bio

Overfitting

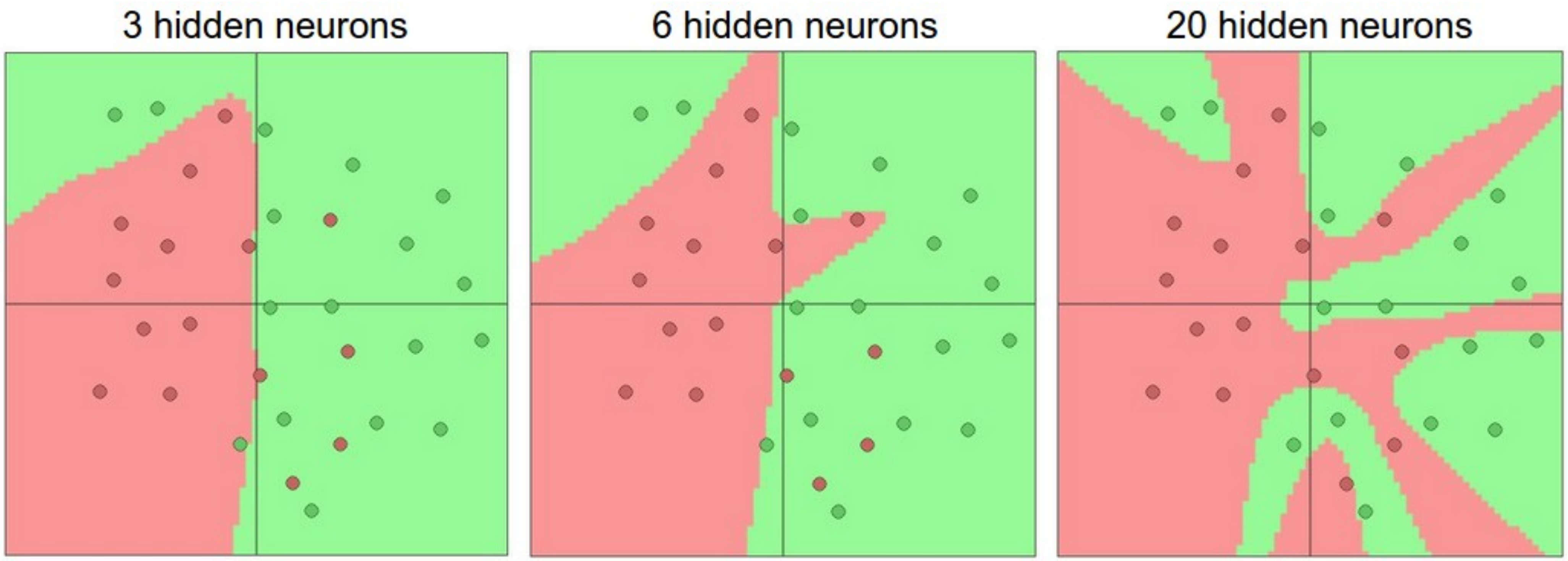




- **Overfitting** occurs when a model with high capacity fits the noise in the data instead of the (assumed) underlying relationship.

Source: https://livingthing.danmackinlay.name/deep_learning.html

Overfitting



- Larger DNNS can represent more complicated functions. Should therefore we go always very deep?
  - No, DNNs with more neurons can express more complicated functions, however, large networks trained on scarce data might lead to overfitting.
- When data is scarce, it is essential to implement methods to prevent overfitting (L2 regularization, dropout, input noise, etc).
- In practice, it is always better to use methods to control overfitting instead of reducing the number of neurons.

http://cs231n.github.io/neural-networks-1/#bio

How to prevent overfitting.

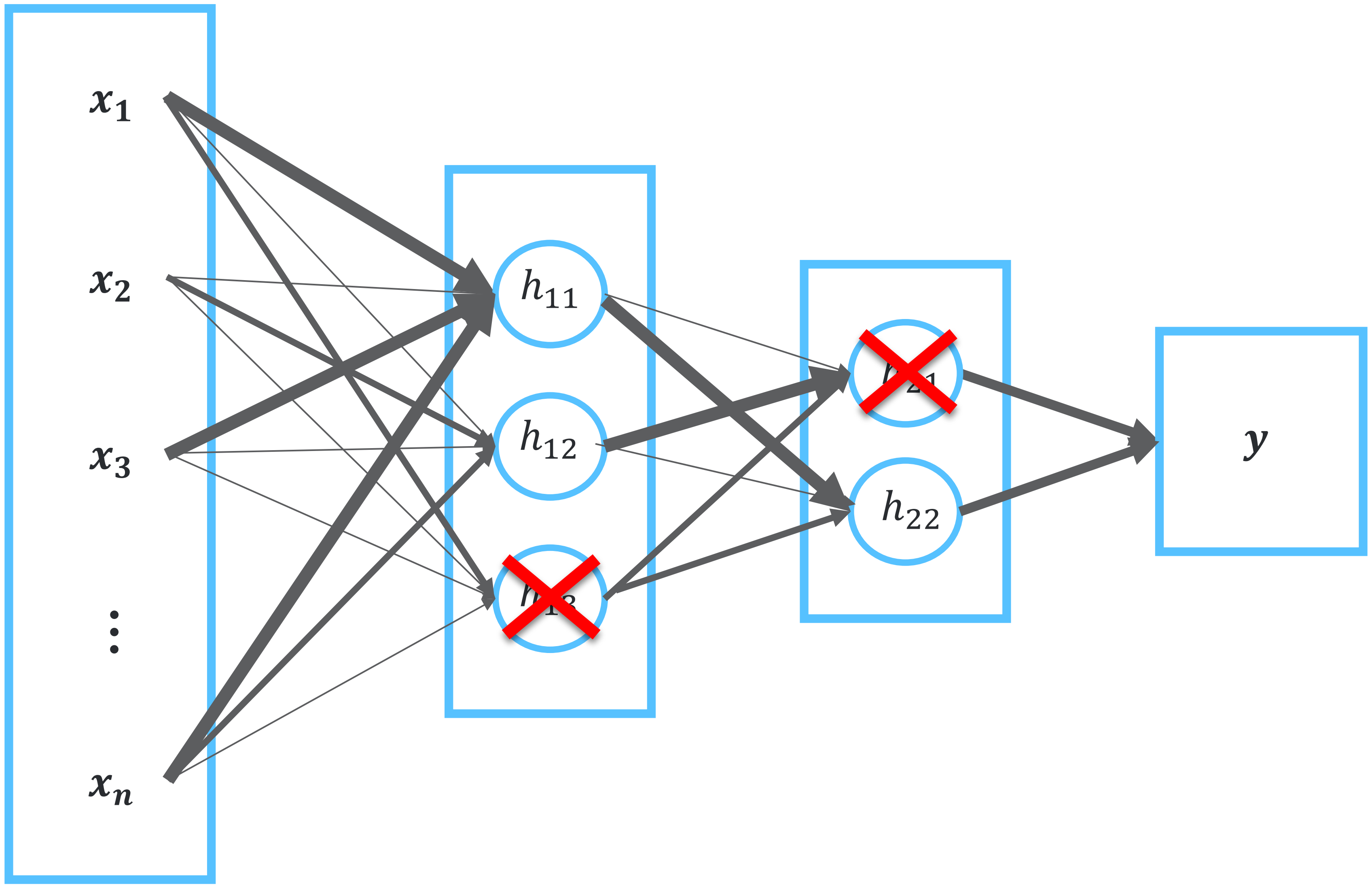
- Early stopping:
  - Stop training as so on as the error on the validation set is higher than it was the last time it was checked.

- Noise addition:
  - Dropout: dropping out units (both hidden and visible) in a neural network.
  - Add noise to data (e.g. denoising autoencoders): we train the network to reconstruct the input from a corrupted version of it.

- Regularization penalties :
  - Create weight penalties L1 and L2.

- Dataset augmentation:
  - Create fake data and add it to the training set.

Dropout



- At each training iteration a dropout layer randomly removes some nodes in the network with probability p along with all of their incoming and outgoing connections.
- Dropout can be applied to hidden or input layer.
- Why it works:
  - Prevents co-adaptation between neurons.
  - Dropout is an example of ensemble technique, where multiple thinned networks with shared parameters are averaged out.

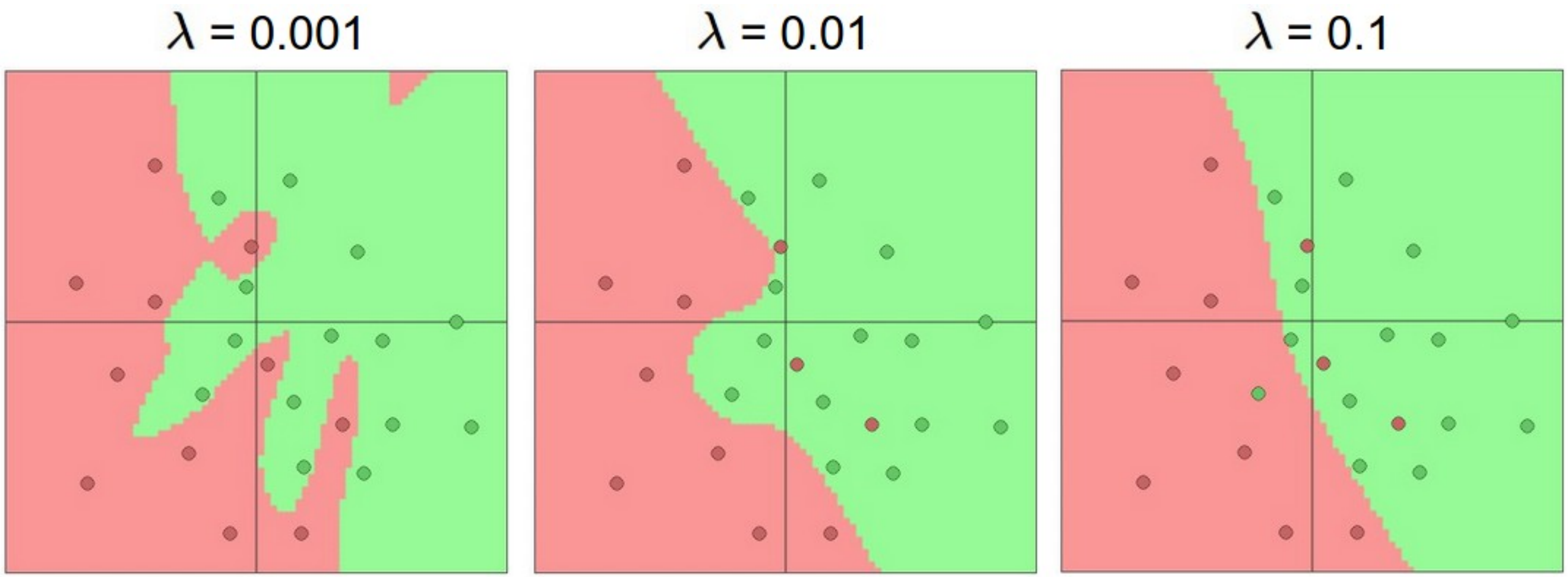
Weight regularization

- L2 norm
  - penalizes the square value of the weight (p = 2).
  - tends to drive all the weights to smaller values.

- L1 norm
  - penalizes the absolute value of the weight (p = 1)
  - tends to drive some weights to exactly zero (introducing sparsity in the model), while allowing some weights to be big.

$$\text{C(W, B, S, O)}_{regularized} = \text{C(W, B, S, O)} + \lambda \sum_{i,j} \left| w_{ij} \right|^{p}$$

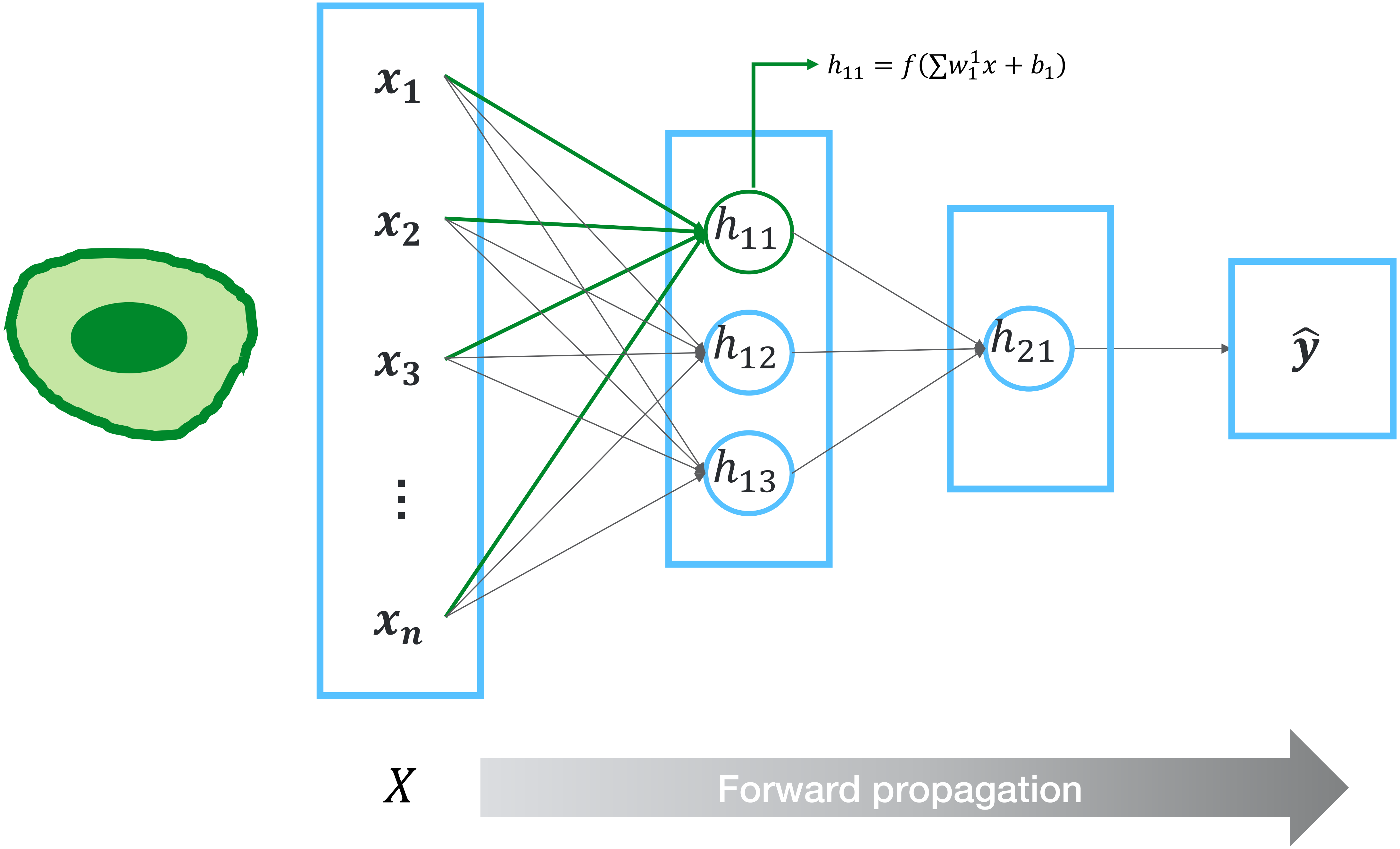
The effect of regularization



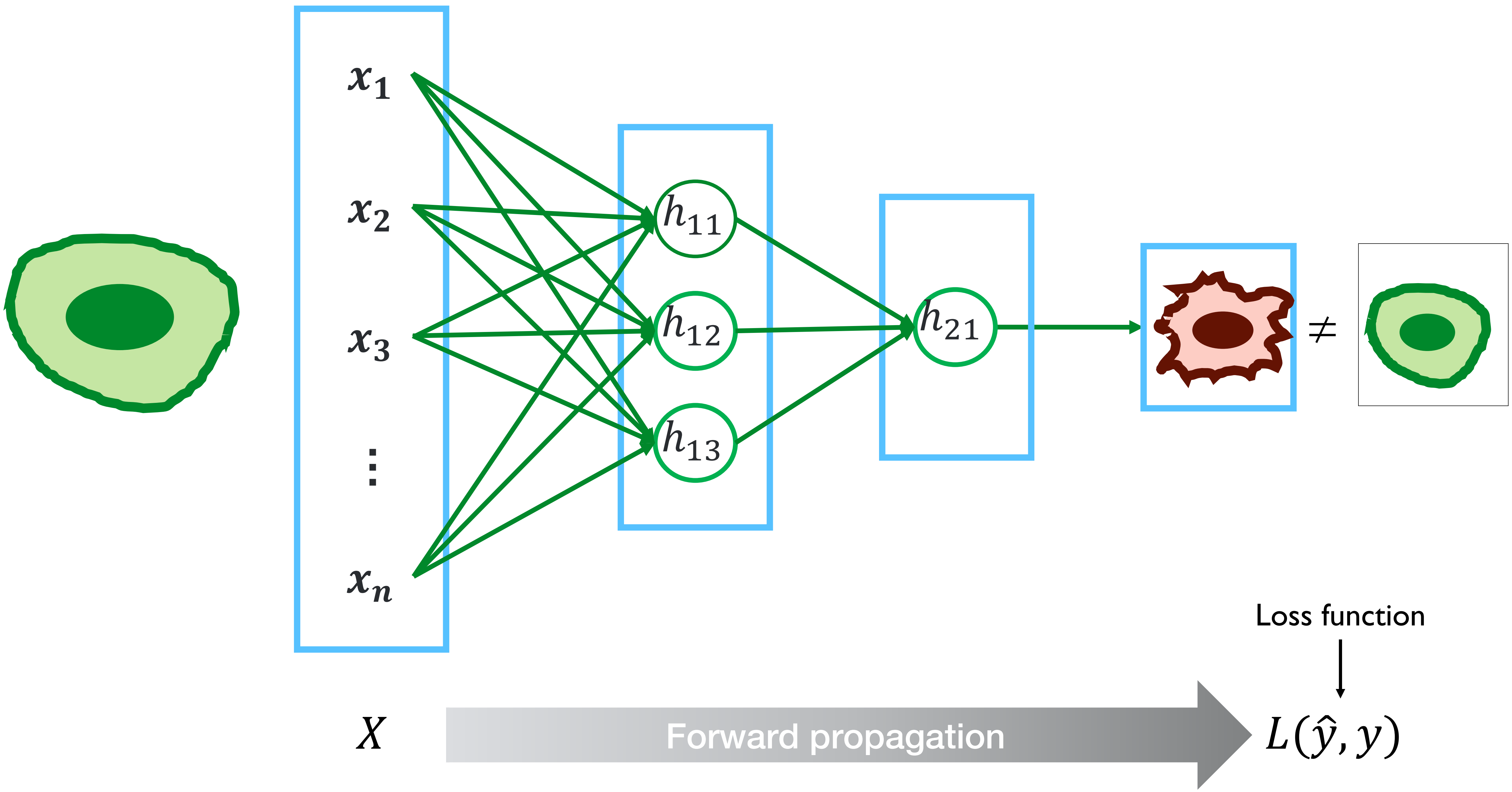
- **The effects of regularization strength**:
  - Each neural network above has 20 hidden neurons, but increasing the regularization strength makes its final decision regions smoother.

http://cs231n.github.io/neural-networks-1/#bio

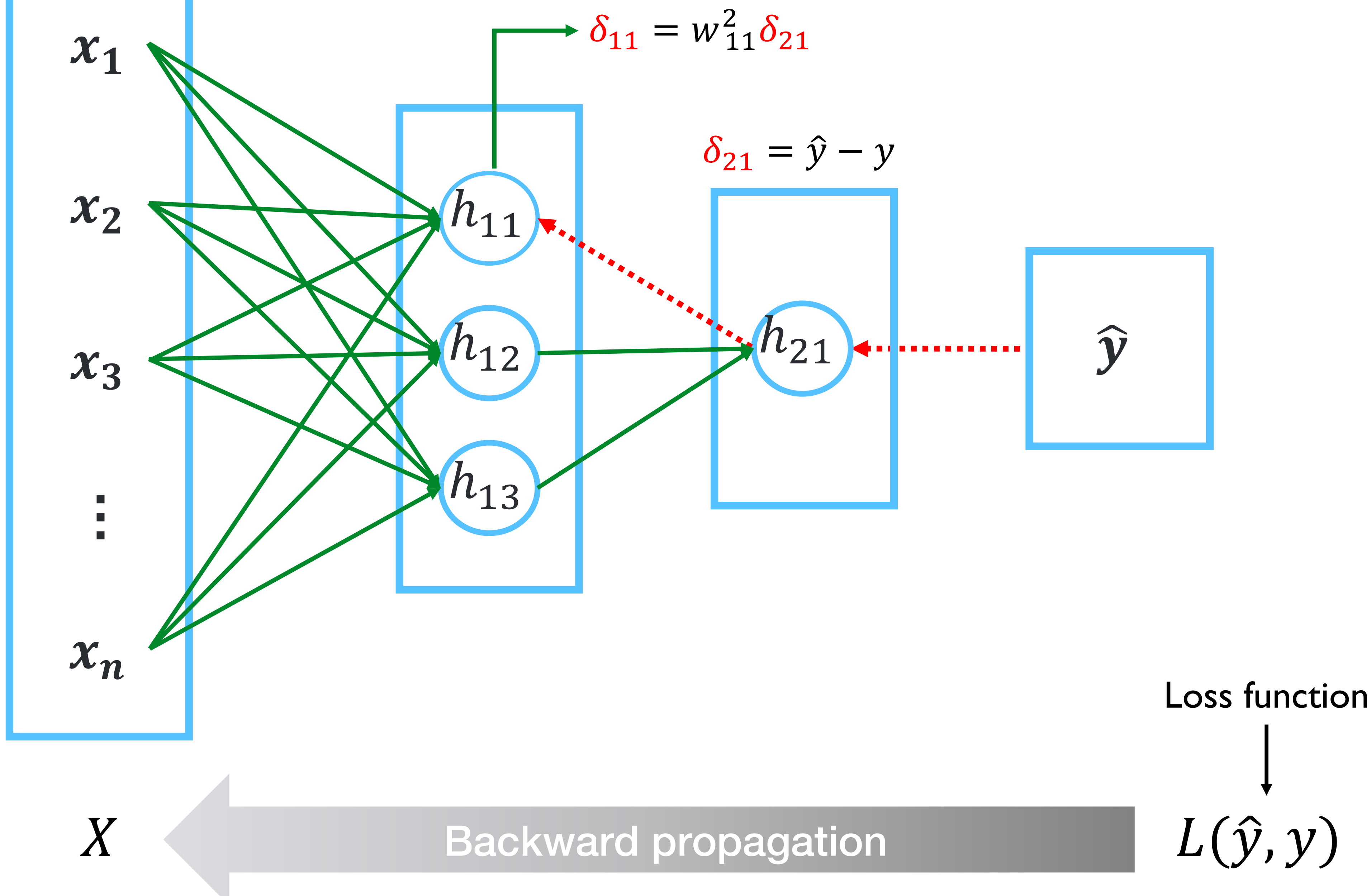
Backpropagation



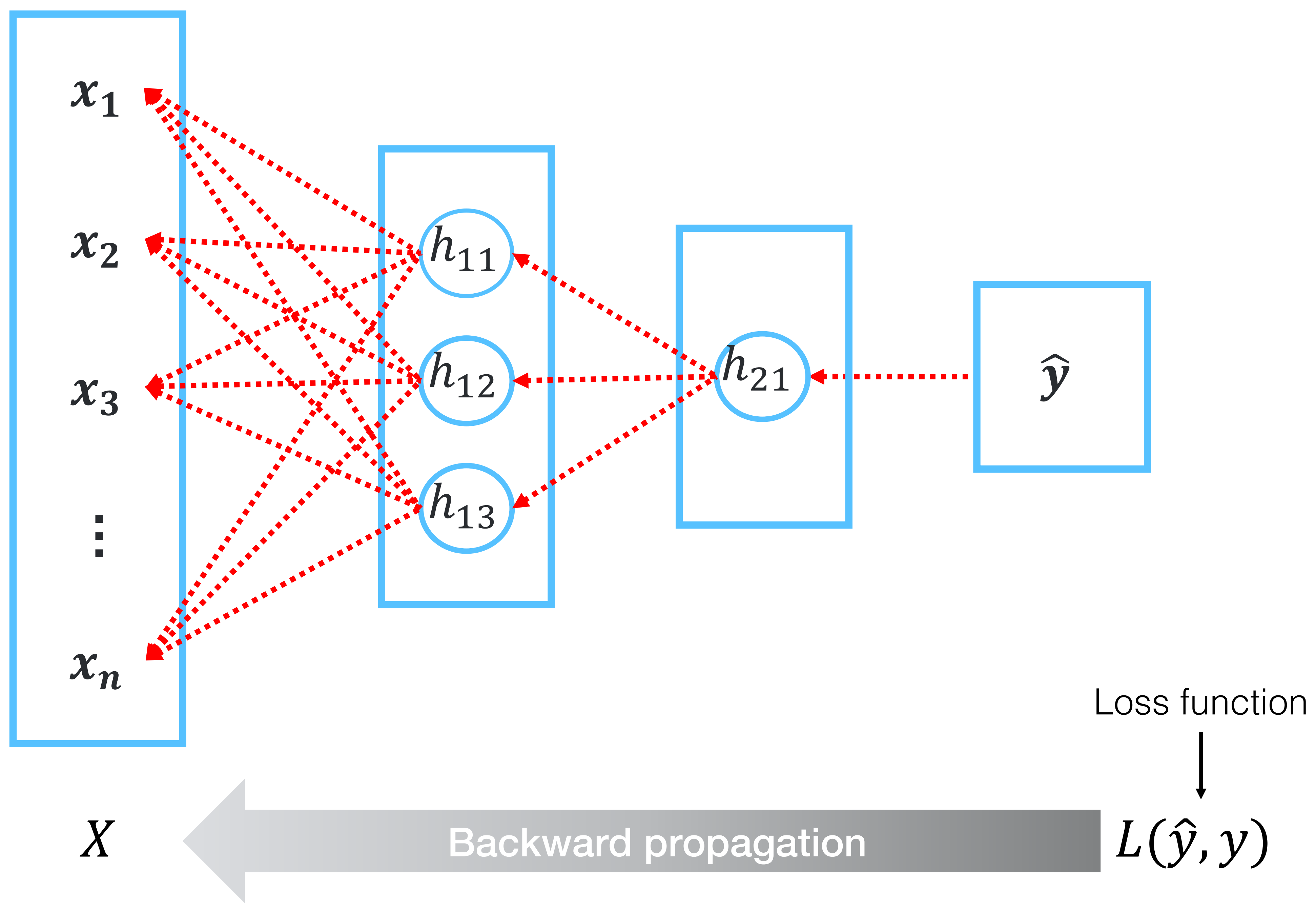
Backpropagation



Backpropagation



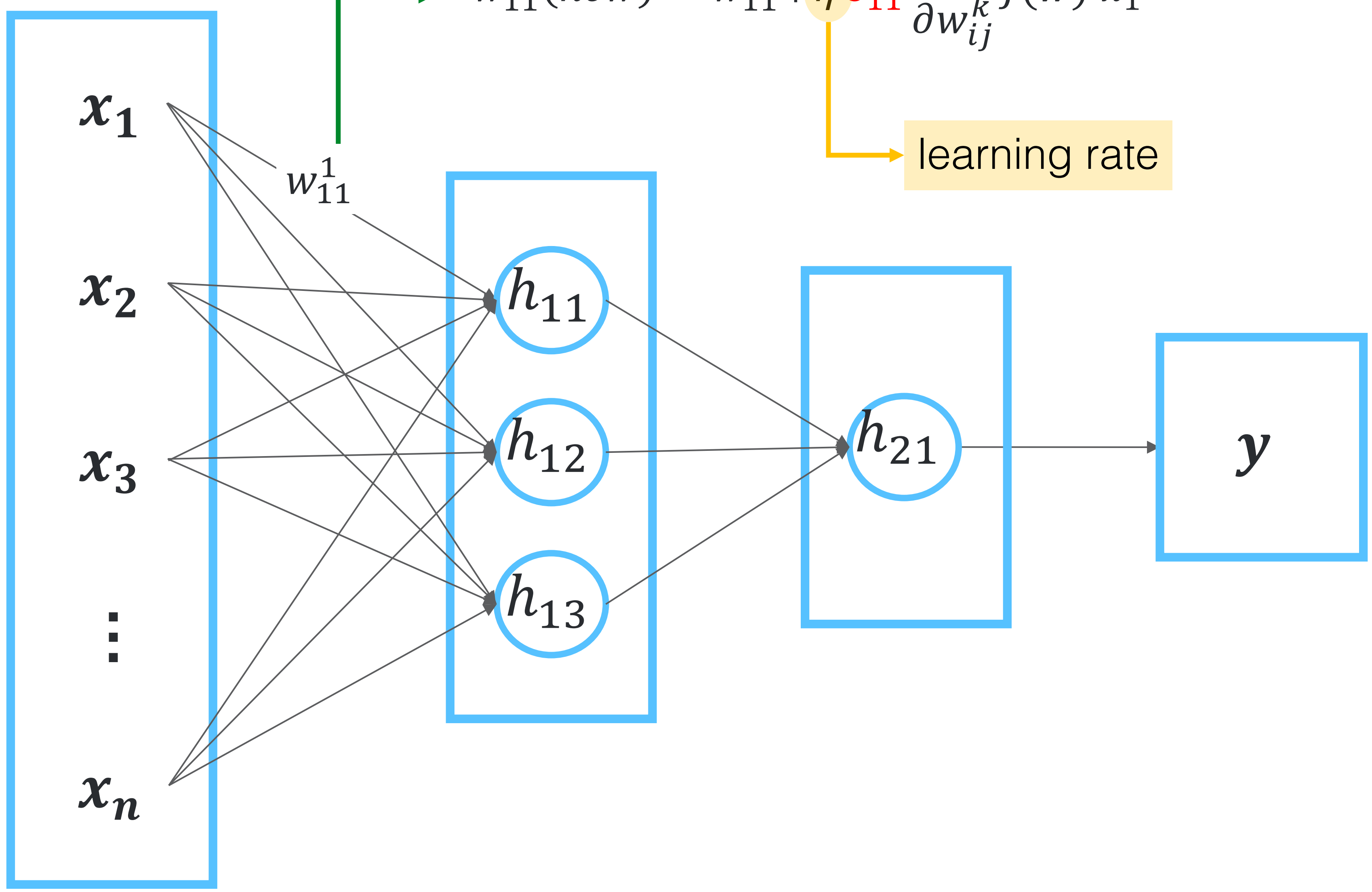
Backpropagation



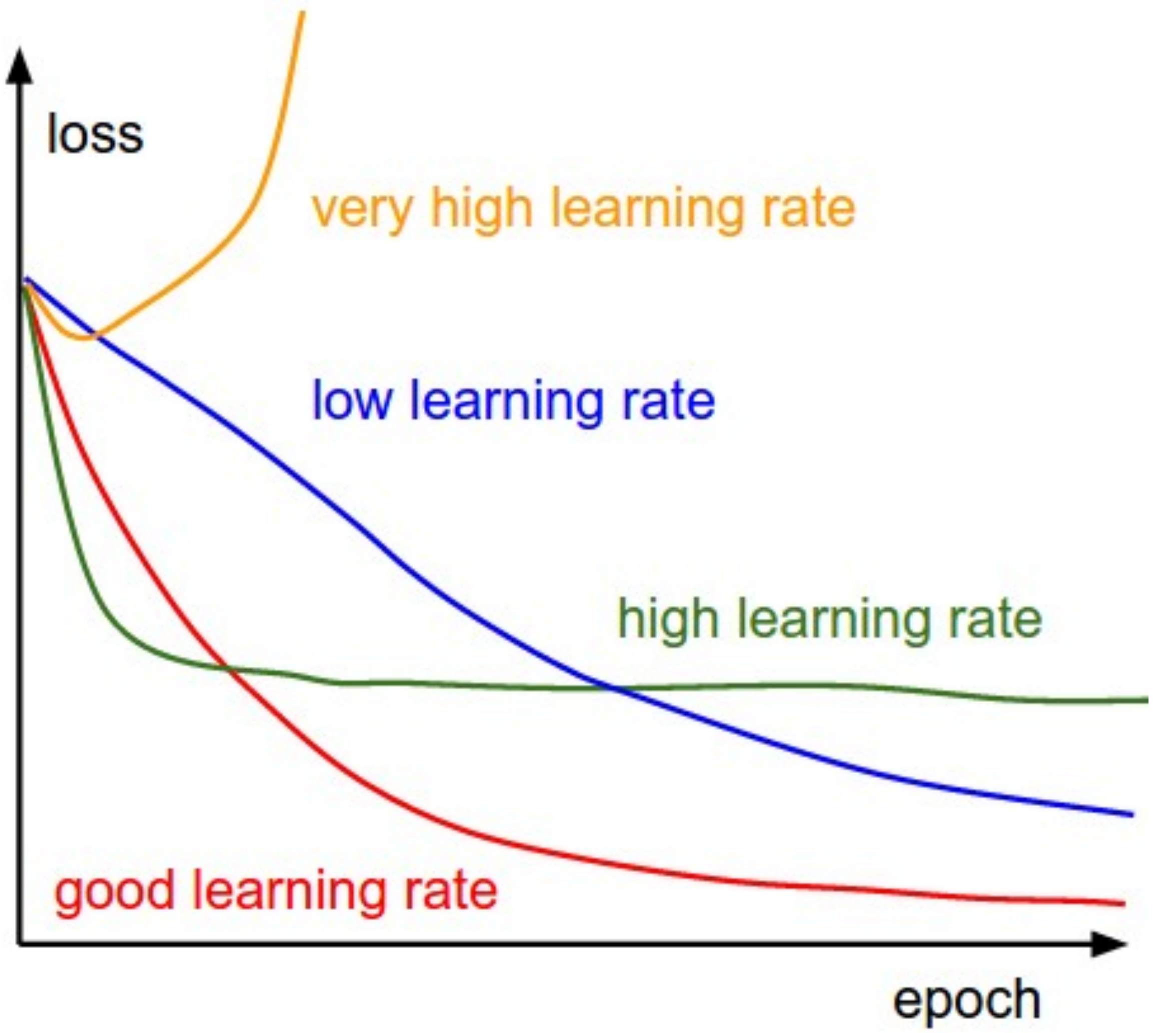
Weight update

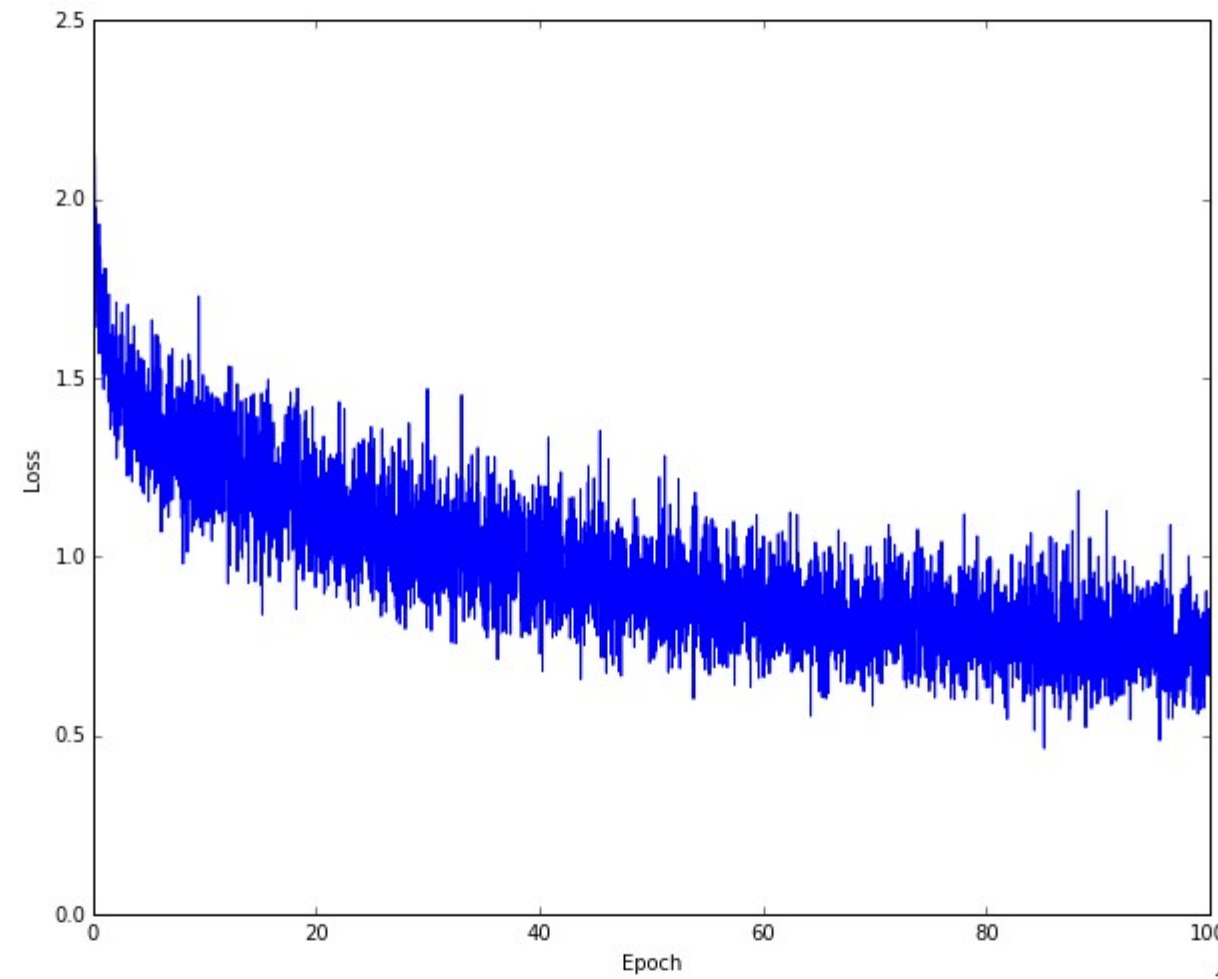
$$w_{11}^{1}(new) = w_{11}^{1} + \eta\, \delta_{11} \frac{\partial}{\partial w_{ij}^{k}} J(\boldsymbol{w})\; x_1$$

learning rate



The effect of different learning rates





- Low learning rates decrease linearly (slow convergence).
- High learning rates initially decrease exponentially, but saturate at higher values: there is too much "energy" in the optimization and the parameters keep bouncing chaotically, unable to settle in a good minimum.

http://cs231n.github.io/neural-networks-3/

Optimizers

Stochastic gradient descent:

1. Choose an initial vector of parameters
2. Repeat until convergence:
- Randomly shuffle training examples
- Move the weight vector towards the direction of steepest descent by learning rate η

Advanced choices:

- **Momentum:** save the update at each iteration, and determine the next update as a linear combination of the gradient and the previous update
- **Adaptive learning rate methods:** RMSprop, Adagrad, Adam

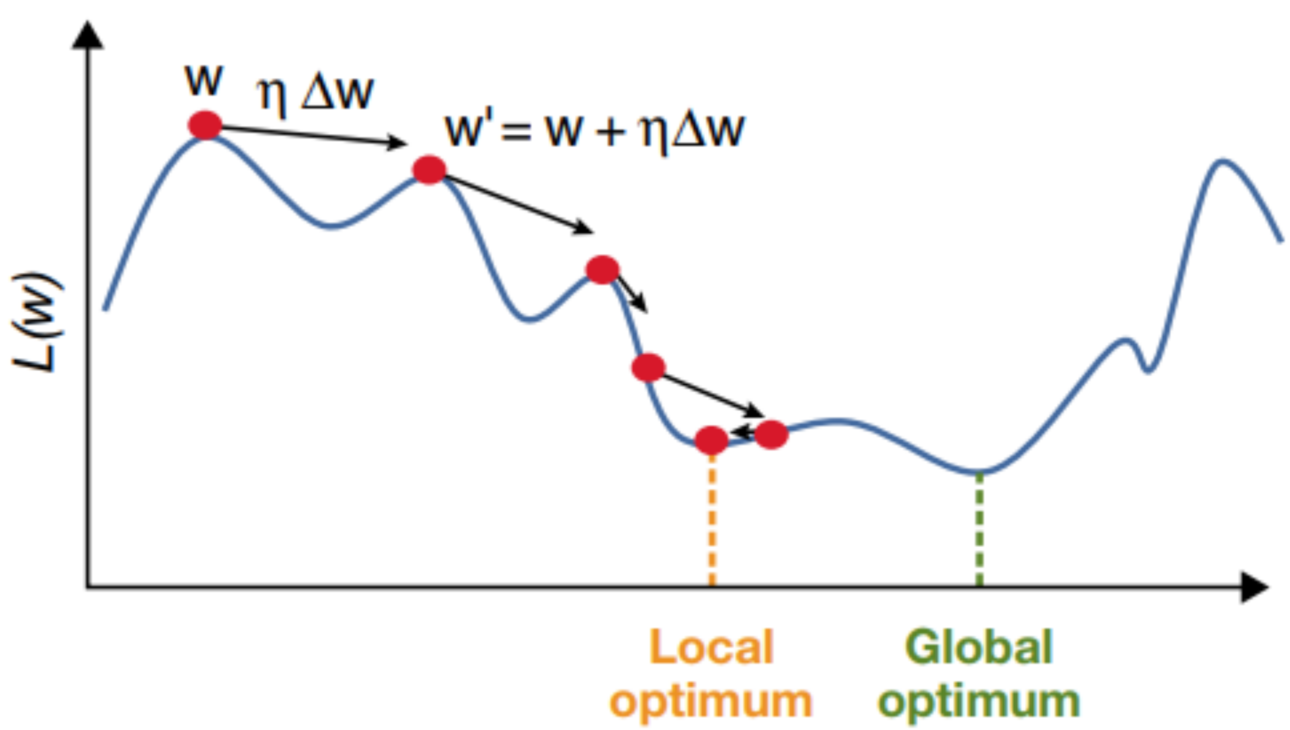
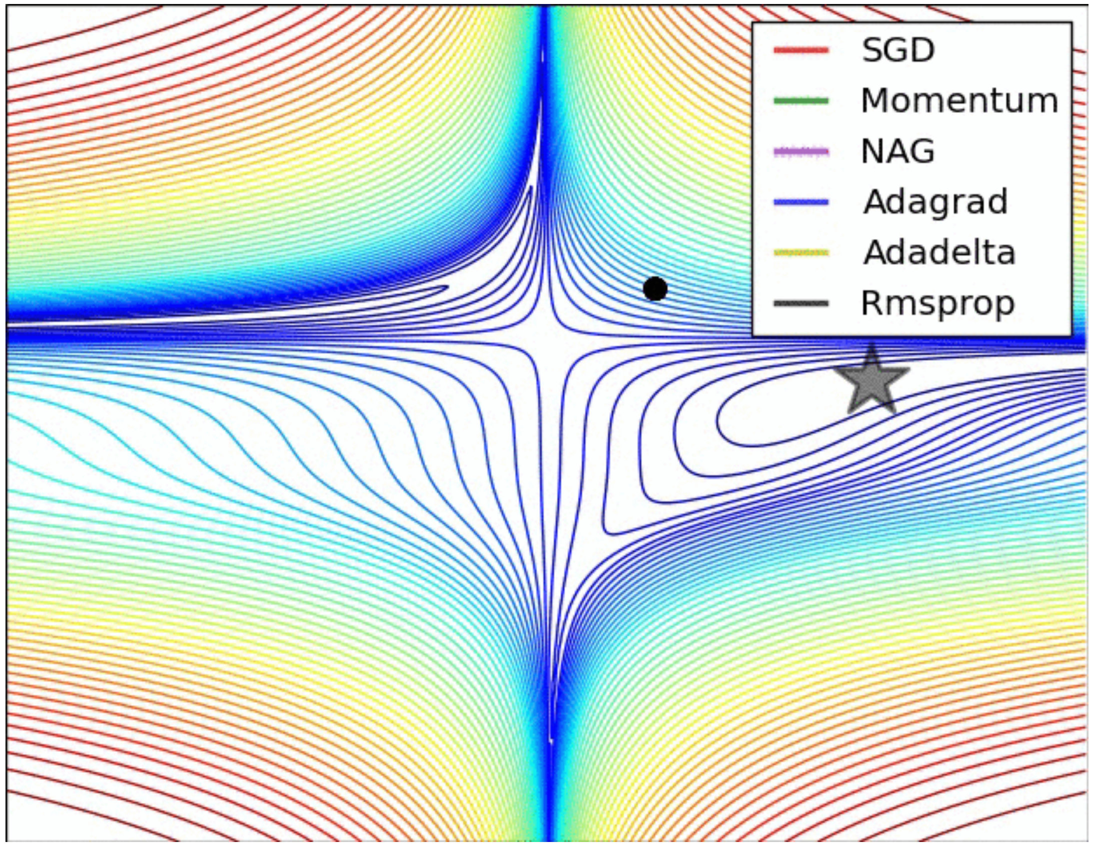
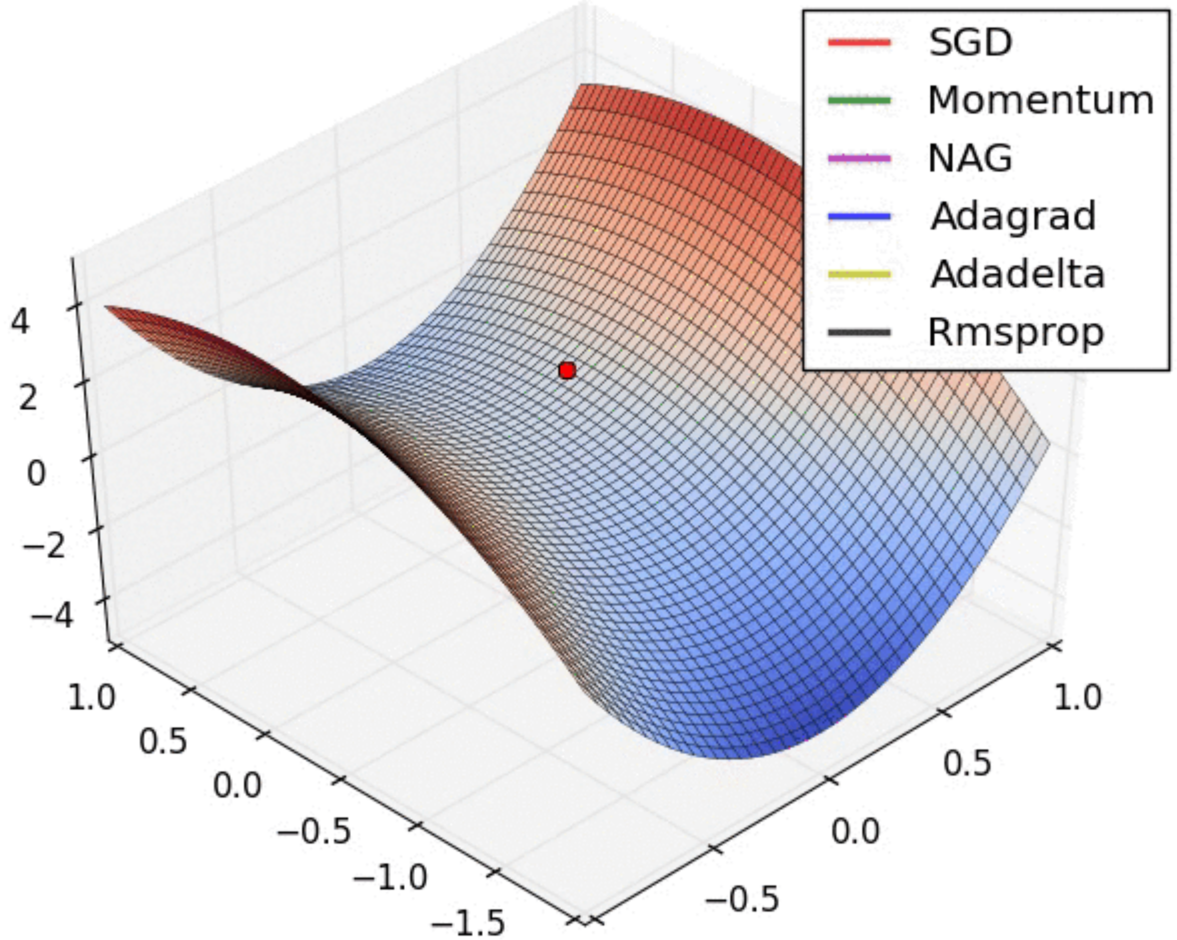


Figure from Angermueller et al., Mol Syst Biol. (2016) 12: 878

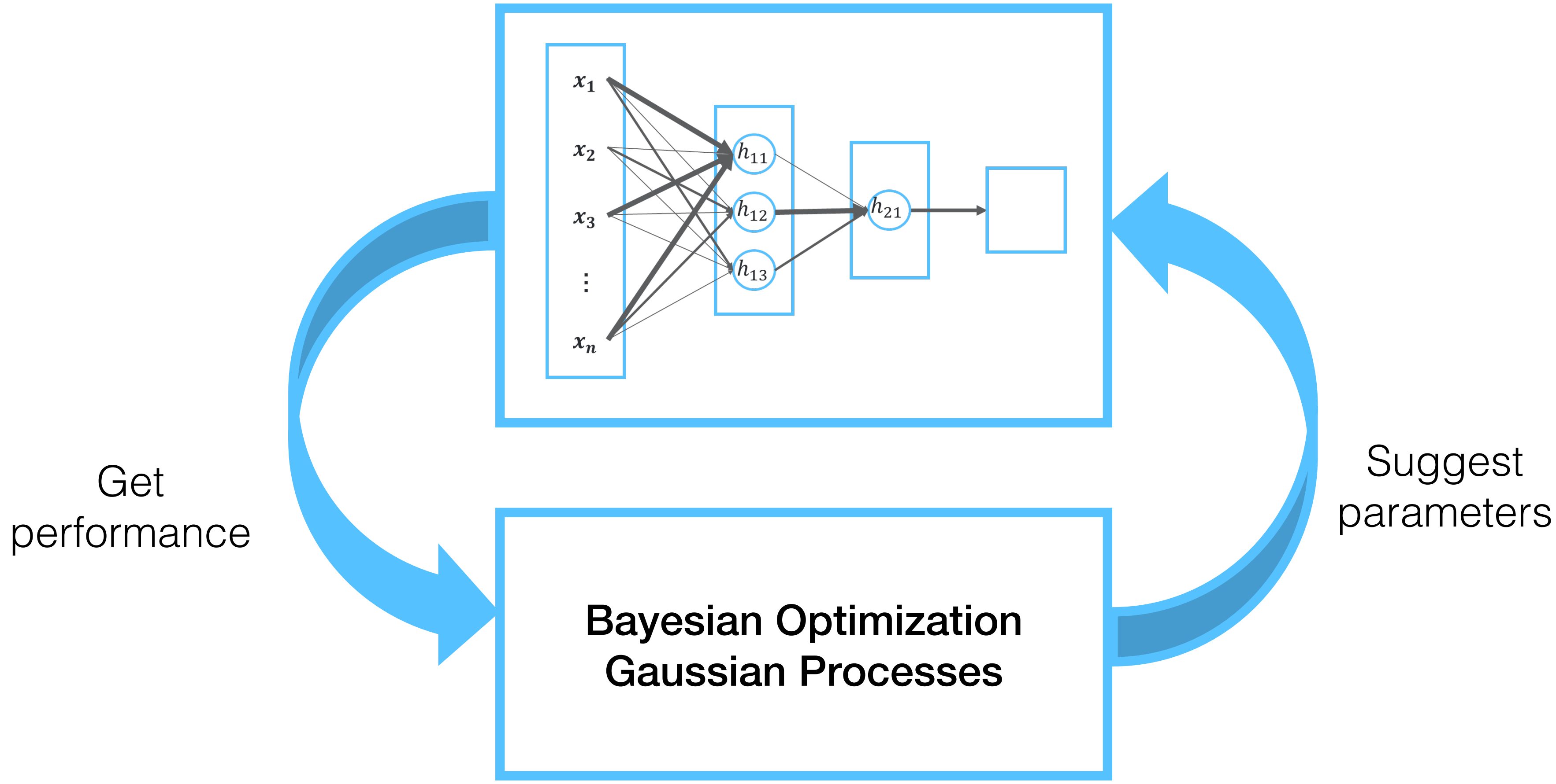
Different optimizers achieve very different convergence rates





Images credit: Alec Radford.

Hyperparameter tuning and reproducibility

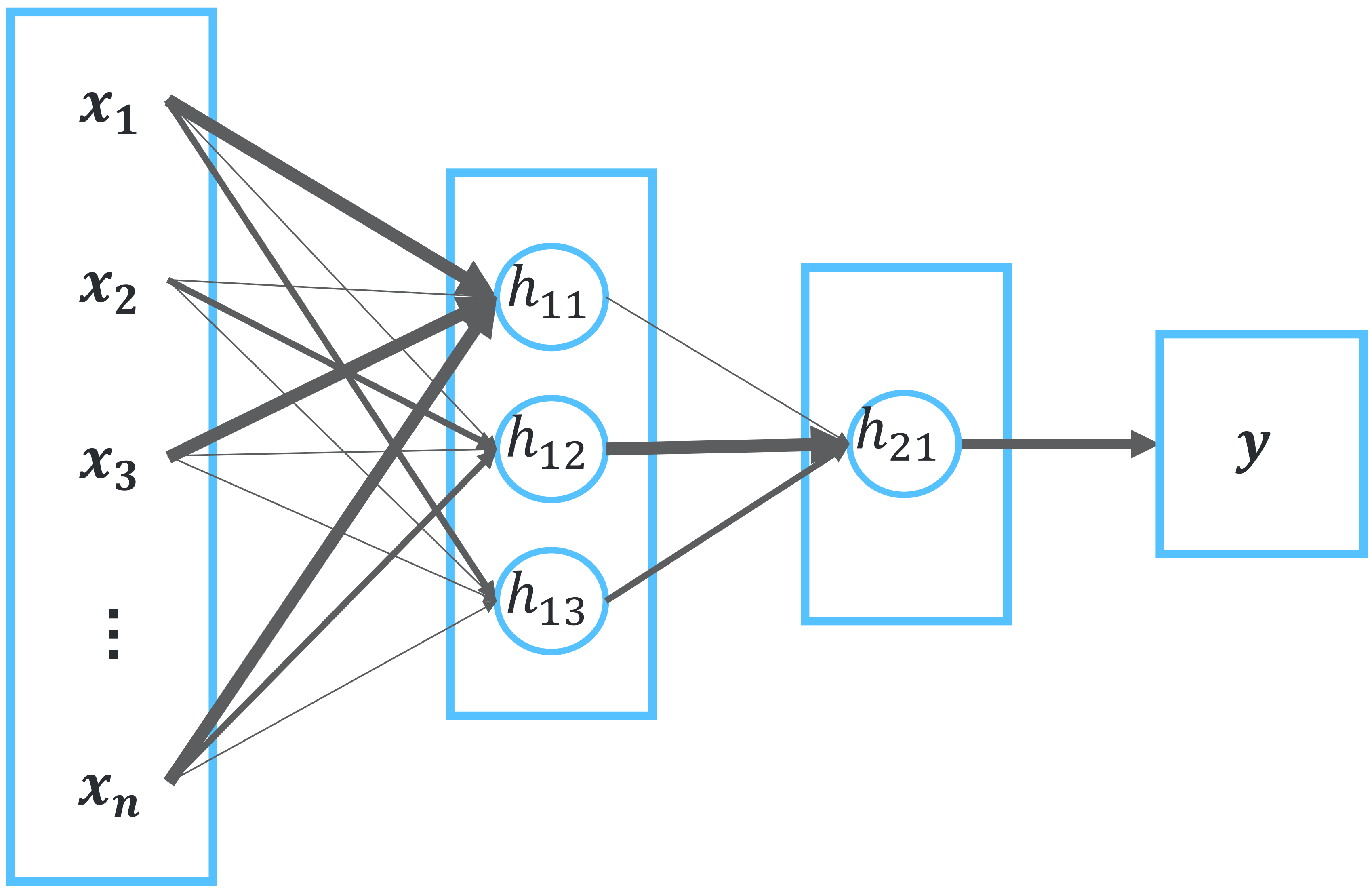


DNNs can involve many hyperparameters. The most common include:

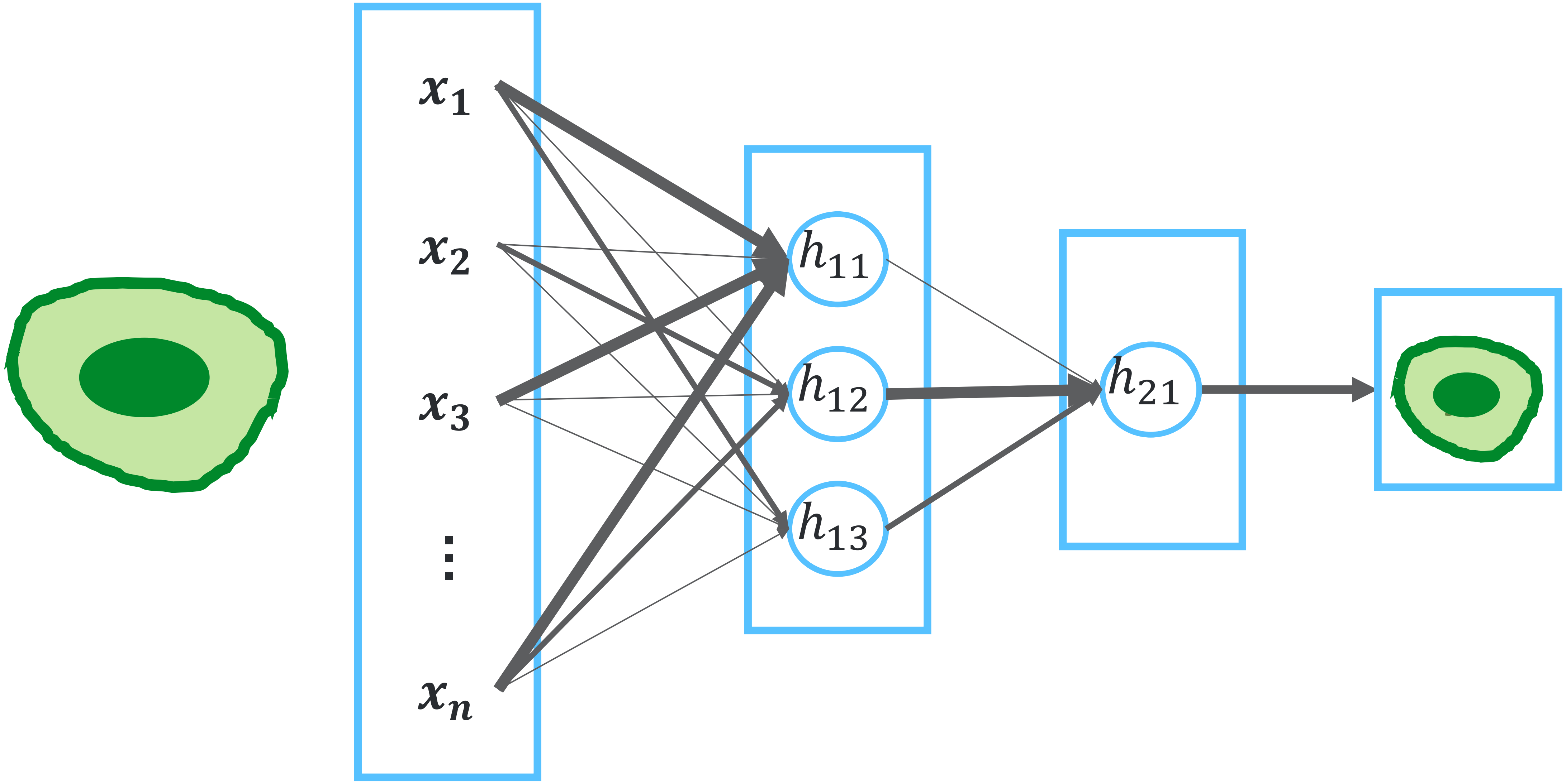
- initial learning rate
- momentum
- regularization strength (L2 penalty, dropout strength, etc)

A grid search exploration of all possible parameter combination might not be the most efficient way of tuning the DNN!

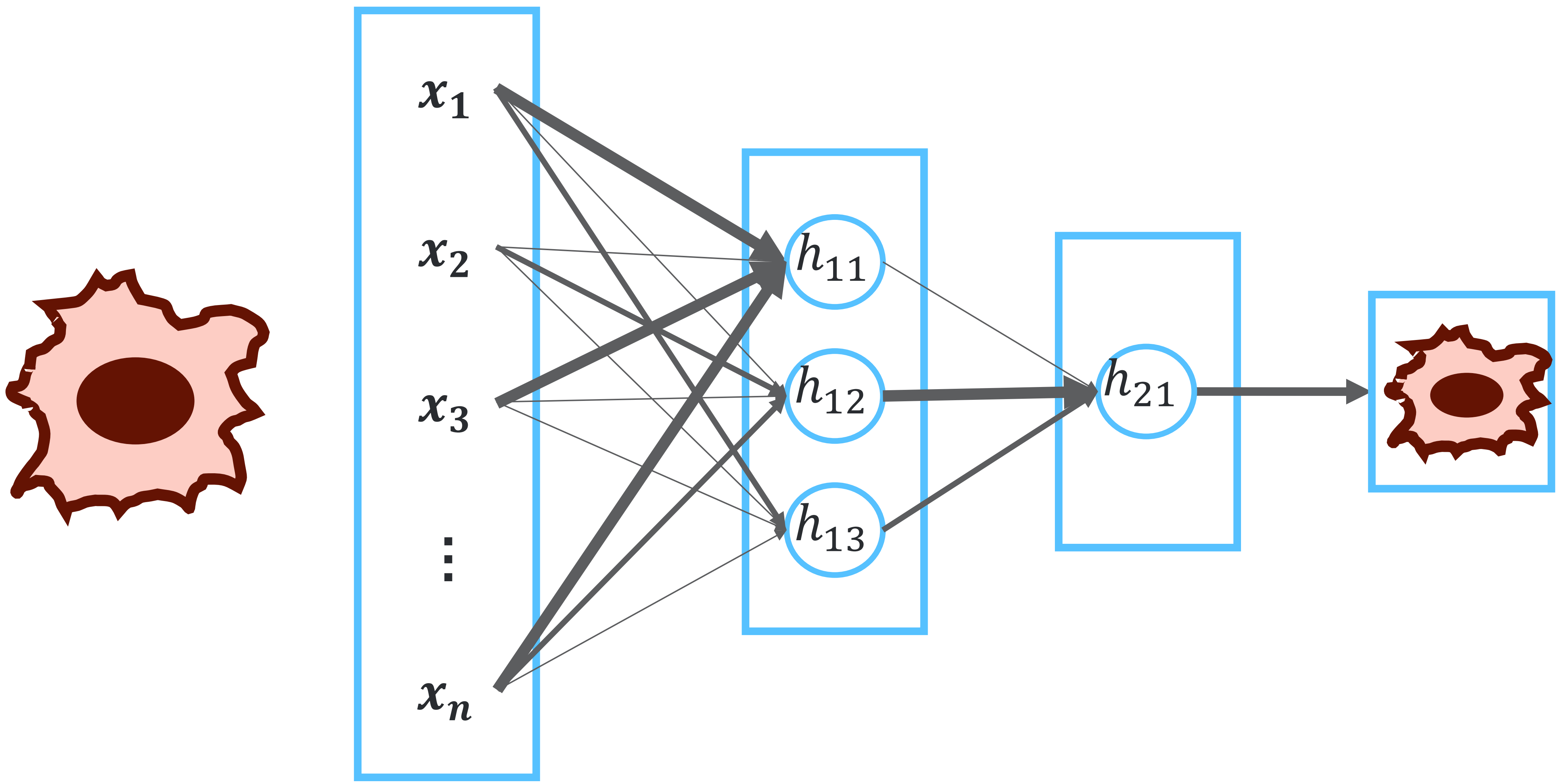
Trained network



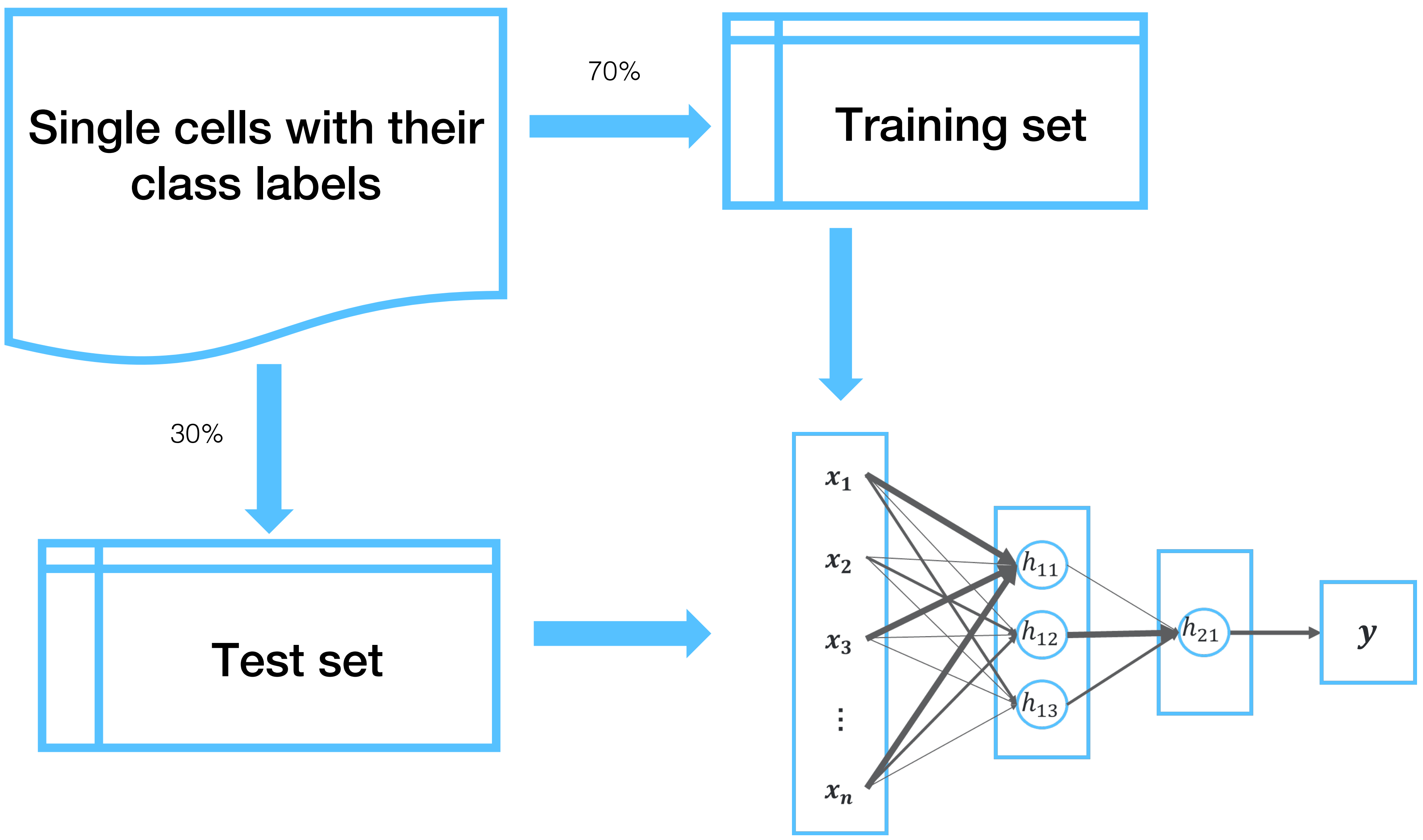
Trained network



Trained network



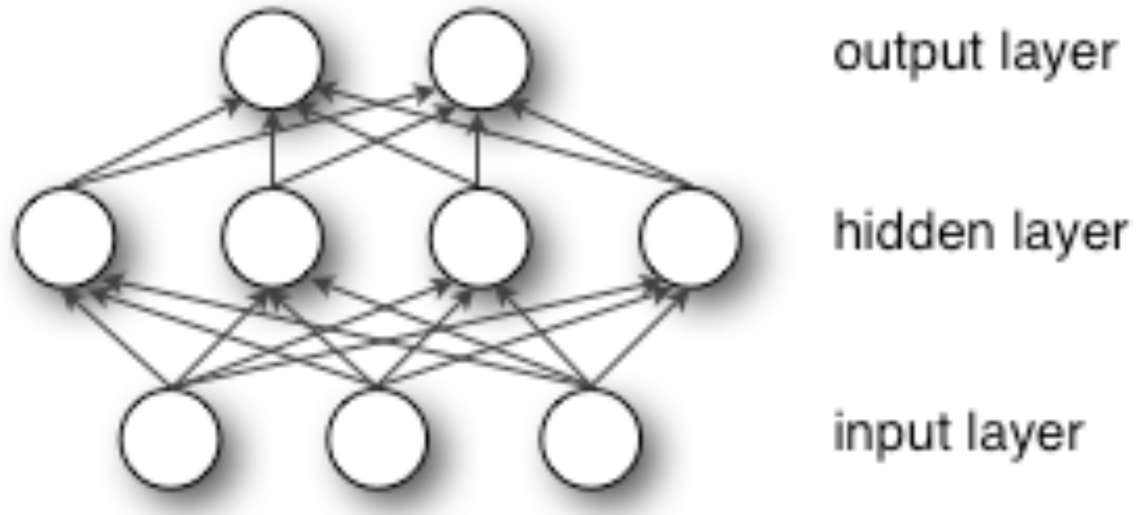
Training and testing



Overview

1. Introduction to deep learning
   - History and motivation
   - Activations functions
   - Cost functions
   - Backpropagation
   - Regularization
   - Optimization
2. Multi-Layer Perceptron (MLP)
3. Auto-enconders (AE)
4. Convolutional Neural Networks (CNN)
5. Recurrent Neural Networks (RNN)

Multi-layer perceptron (MLP)



- An MLP is a DNN that:
  - Consists at least of three layers of nodes (i.e. there is at least one hidden layer).
  - It is always feedforward (no loops are allowed).
  - Consecutive layers are fully connected.

- A single hidden layer is sufficient to make MLPs **a universal approximator**. However usually there are substantial benefits to using more than one hidden layer.

http://deeplearning.net/tutorial/mlp.html

Deep learning in genomics

Article | OPEN

Deep Learning in Label-free Cell Classification

Claire Lifan Chen, Ata Mahjoubfar, Li-Chia Tai, Ian K. Blaby, Allen Huang, Kayvan Reza Niazi & Bahram Jalali

NATURE METHODS | BRIEF COMMUNICATION

Predicting effects of noncoding variants with deep learning–based sequence model

Jian Zhou & Olga G Troyanskaya

RESEARCH ARTICLE

The human splicing code reveals new insights into the genetic determinants of disease

Hui Y. Xiong[1,2,3,*], Babak Alipanahi[1,2,3,*], Leo J. Lee[1,2,3,*], Hannes Bretschneider[1,3,4], Daniele Merico[5,6,7], Ryan K. C. Yuen[5,6,7], Yimin Hua[8], Serge Gueroussov[2,7], Hamed S. Najafabadi[1,2,3], Timothy R. Hughes[2,3,7], Quaid Morris[1,2,3,7], Yoseph Barash[1,2,9], Adrian R. Krainer[8], Nebojsa Jojic[10], Stephen W. Scherer[3,5,6,7], Benjamin J. Blencowe[2,5,7], Brendan J. Frey[1,2,3,4,5,7,10,†]

A deep learning framework for modeling structural features of RNA-binding protein targets

Sai Zhang[1], Jingtian Zhou[2,†], Hailin Hu[2,†], Haipeng Gong[3,4], Ligong Chen[2], Chao Cheng[5,*] and Jianyang Zeng[1,4,*]

Gene expression inference with deep learning

Yifei Chen[1,4,†], Yi Li[1,†], Rajiv Narayan[2], Aravind Subramanian[2] and Xiaohui Xie[1,3,*]

PDF version

Deep biomarkers of human aging: Application of deep neural networks to biomarker development

Evgeny Putin [1,2], Polina Mamoshina[1,3], Alexander Aliper[1], Mikhail Korzinkin[1], Alexey Moskalev[1,4], Alexey Kolosov[5], Alexander Ostrovskiy[5], Charles Cantor[6], Jan Vijg[7], and Alex Zhavoronkov[1,3]

Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks

David R Kelley[1], Jasper Snoek and John Rinn

Deep learning of the tissue-regulated splicing code

Michael K. K. Leung[1,2], Hui Yuan Xiong[1,2], Leo J. Lee[1,2] and Brendan J. Frey[1,2,3,*]

Deep learning frameworks

High-level frameworks make deep learning easier

Deep Learning Frameworks:

- Keras
- Lasagne
- Caffe

Graph compilers:

- Theano
- Tensor Flow

Linear Algebra Libraries:

- PyCuda (python)
- CUDAMat (python)
- JCuda (java)

Thank you!

CompSysBio team @ IBM Research