LDA and others, recap.

- Good source to recap is to look through the Data Mining course at Stanford University based on “Introduction to Statistical Learning” and “Elements of Statistical Learning”.
- http://web.stanford.edu/class/stats202/content/lectures.html
- **The best explanations usually are in tools here the python counterparts https://scikit-learn.org/stable/index.html**

Practical hands on recap : model building and validation

- Generate artificial data
  - Linearly separable rnorm , spherical
  - Linearly separable, overlapping classes rnorm , sperical and ellipsoids
  - Nonlinearly separable- banana data
  - Iris
- Attempt to visualize decision boundaries
- Run cross validation script – estimate generalization performance
  - Leave one out cross-validation
  - 10 fold cross validation
  - 5 fold cross validation

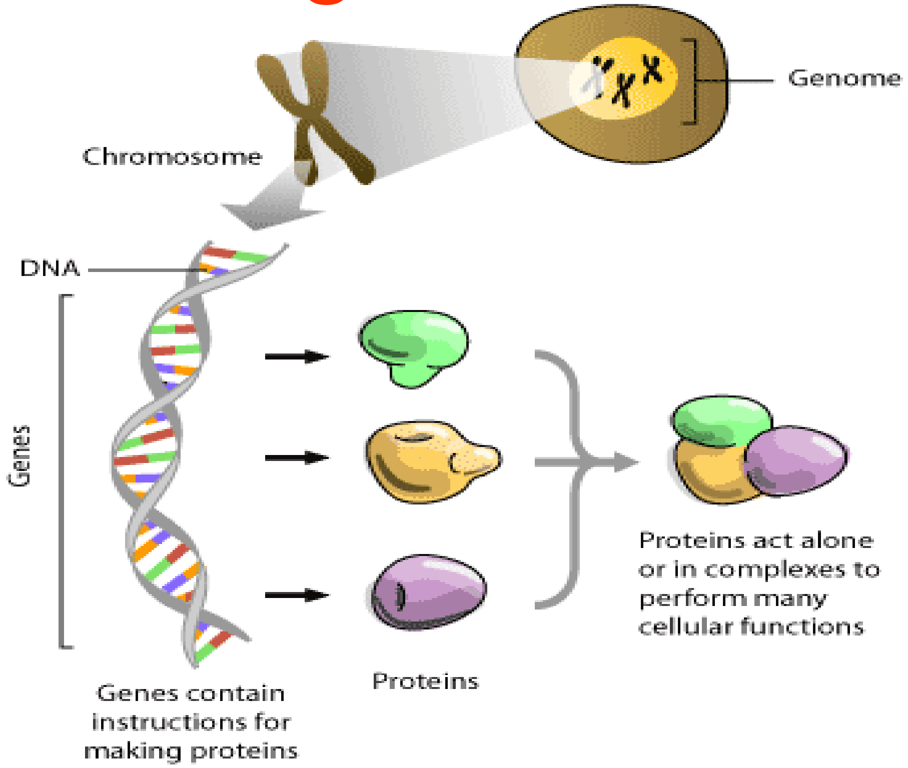
Decision boundaries, separating hyperplanes in short from Duda, Hart and Stork

- The text from the book directly
- 2.4 – 2.6 pages 13-45
- Need to explore the data configurations and discrimiant forms that are explained in this text using our script to generate artificial data

Support Vector Machine SVM. Underlying idea, software and application

Learning machine relies on a software, that makes possible learning and decision making.

Software relies on either a mathematical model or heuristics,
which usually also is based on mathematical principles of optimization.


Genome
Chromosome
DNA
Genes
Genes contain instructions for making proteins
Proteins
Proteins act alone or in complexes to perform many cellular functions

SVM is a machine learning algorithm, very successful in many practical applications including bioinformatics:

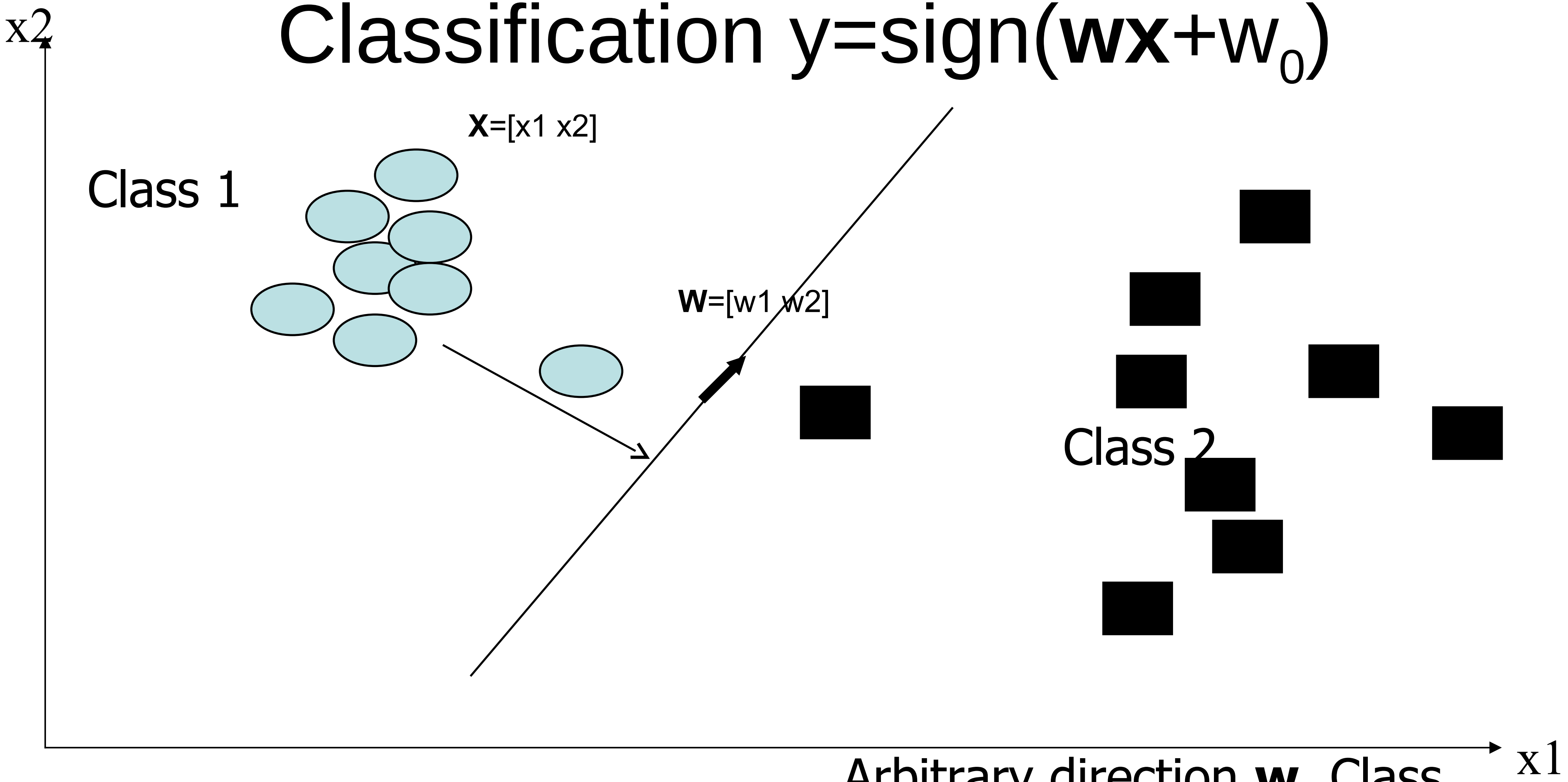
- Recognition of translation start sites
  - Protein remote homology detection
  - Protein fold recognition
- **Microarray gene expression analysis**
  - Functional classification of promoter regions
- Prediction of protein-protein interactions
  - Peptide identification from mass spectrometry data

Pavlidis P.,Wapinski I., Noble W.S. Support vector machine classification on the web. Bioinformatics. Vol. 20, No. 4, 2004.

Outline

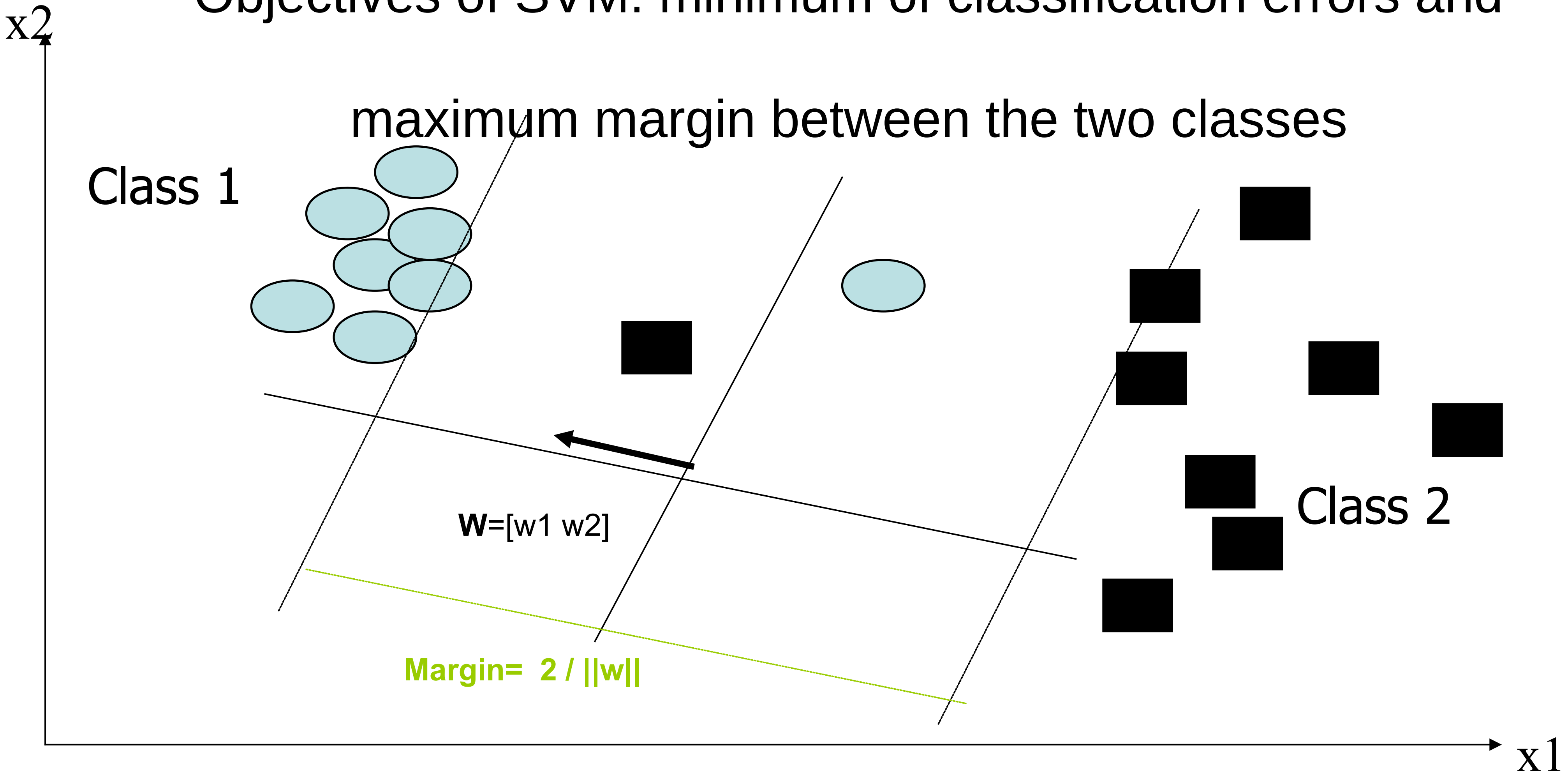
- Classification in simple terms
- Support Vector Machine classifier
- Formulation and computation of SVM classifier
- SVM software tools
- Application of SVM for mining genotype-phenotype association in sports genetics

Classification $y=\text{sign}(\mathbf{wx}+w_0)$



Arbitrary direction **w**, Class separation is arbitrary.

Objectives of SVM: minimum of classification errors and maximum margin between the two classes



Standard SVM formulation

Decision rule $$f(x) = \operatorname{sign}(\langle \mathbf{w}, \Phi(x)\rangle + b)$$

$$\begin{aligned}
\text{minimize} \quad & t(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i \\
\text{subject to} \quad & y_i(\langle \Phi(x_i), \mathbf{w}\rangle + b) \geq 1 - \xi_i \qquad (i = 1, \ldots, m) \\
& \xi_i \geq 0 \qquad (i = 1, \ldots, m) \qquad y_i = \pm 1
\end{aligned}$$

Solution $$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \boxed{\Phi(x_i)}$$

Mapping into higher dimensional space

Computation of SVM decision rule

Quadratic constrained optimization problem

$$\text{maximize} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

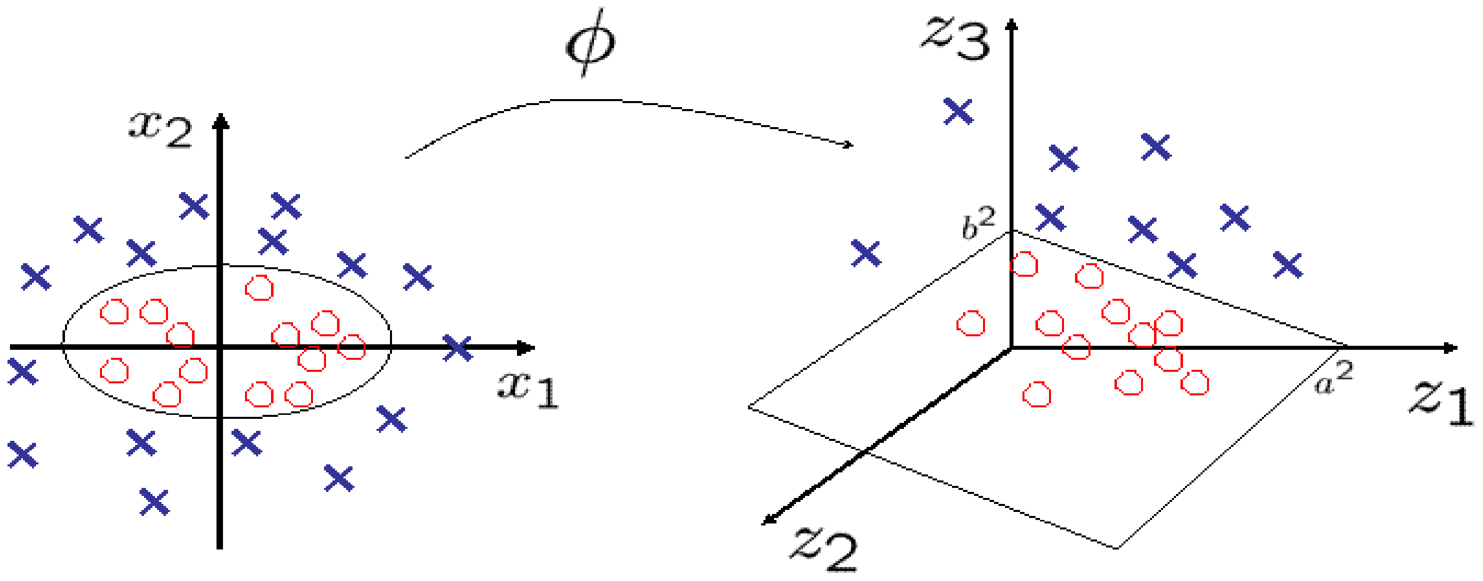
$$\text{subject to} \quad 0 \leq \alpha_i \leq \frac{C}{m} \qquad (i = 1, \ldots, m)$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$

KERNEL $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$

Karatzoglu A., Meyer D., Hornik K. Support Vector Machines in R.
Journal of Statistical Software. Vol. 26, Iss. 9. 2006.

Kernel trick



$$\phi : (x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Kernel entries depend only on training data. Kernel may be interpreted as a proximity metric of the data points/objects.

$$H_{ij} = y_i y_j k(x_i, x_j)$$

Kernels can be constructed for specific data types: strings, graphs. This property led to many successful applications of SVM in bioinformatics. Kernels can help to handle multimodal signal integration as well.

Vert J.P. Kernel methods in genomics and computational biology. arXiv-preprint, 2008.

SVM references and free software

- Websites devoted to the SVM topics
  - http://www.svms.org/
  - http://www.kernel-machines.org/

Summary

- Support Vector Machine is a classification algorithm highly successful in practical tasks.
- Classification rule of SVM represents a hyper-plane, minimizing classification errors and maximizing a margin between the classes.
- The “kernel trick” is applied in order to find a nonlinear separation boundary between the classes if needed.
- Kernel in SVM represents a similarity metrics of the data points.
- Kernels can be designed to handle structural data, which makes SVM very flexible.

Deep learning

Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.

1. LeCun Y, Bengio Y, Hinton G. Deep learning. nature. 2015 May;521(7553):436-44.
2. Angermueller C, Pärnamaa T, Parts L, Stegle O.Deep learning for computational biology. Molecular systems biology. 2016 Jul 1;12(7).
3. Pérez-Enciso M, Zingaretti LM. A Guide on Deep Learning for Complex Trait Genomic Prediction. Genes. 2019 Jul;10(7):553.

Angermueller C, Pärnamaa T, Parts L, Stegle O.Deep learning for computational biology. Molecular systems biology. 2016 Jul 1;12(7).

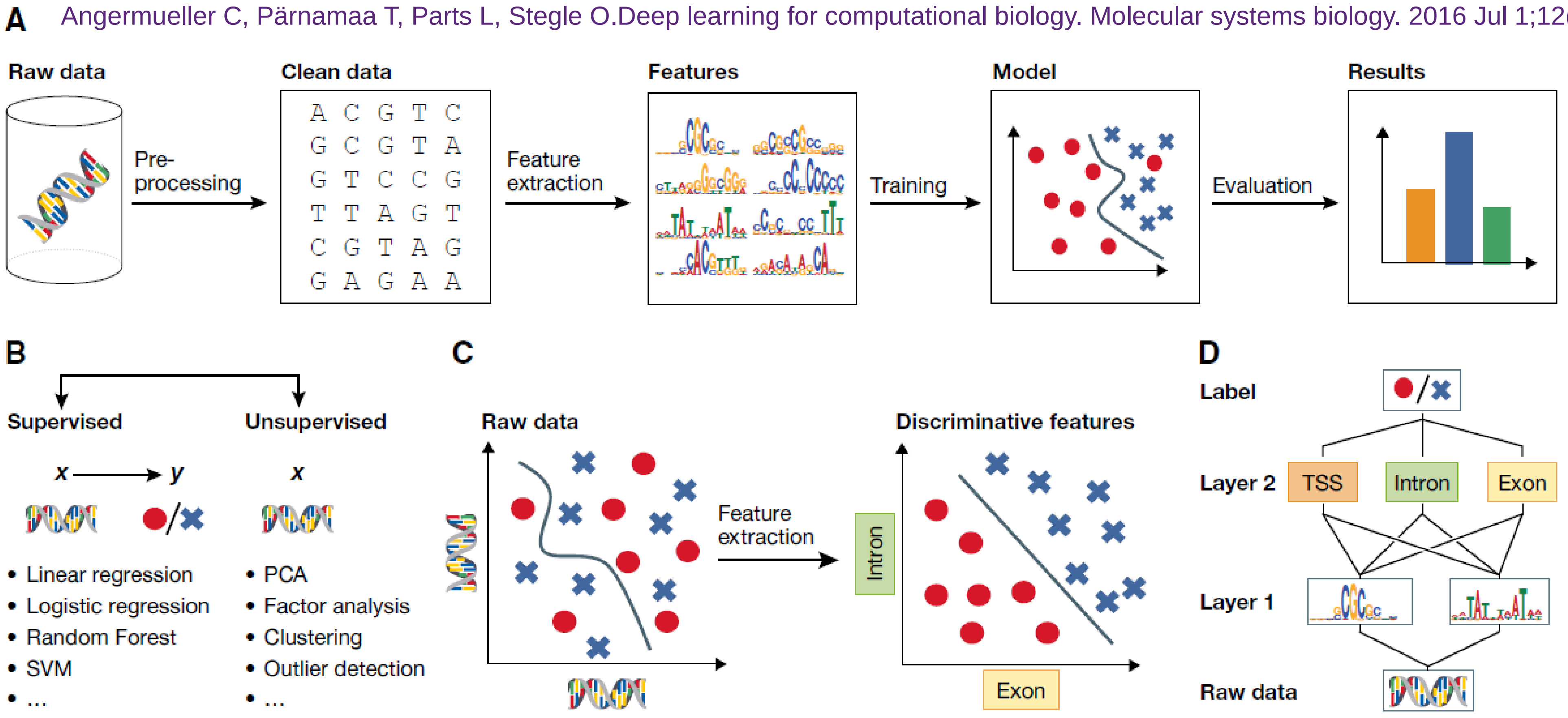
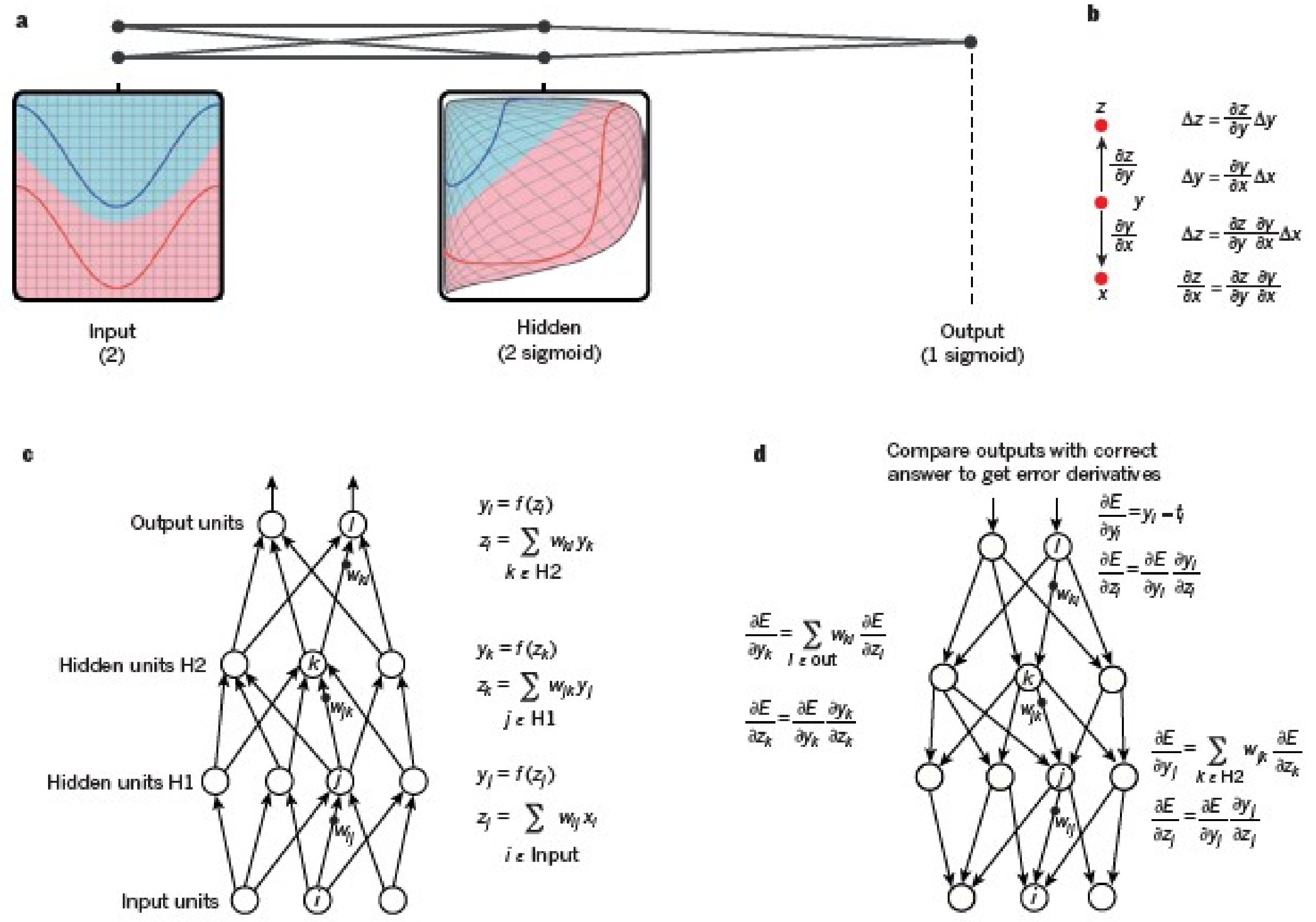


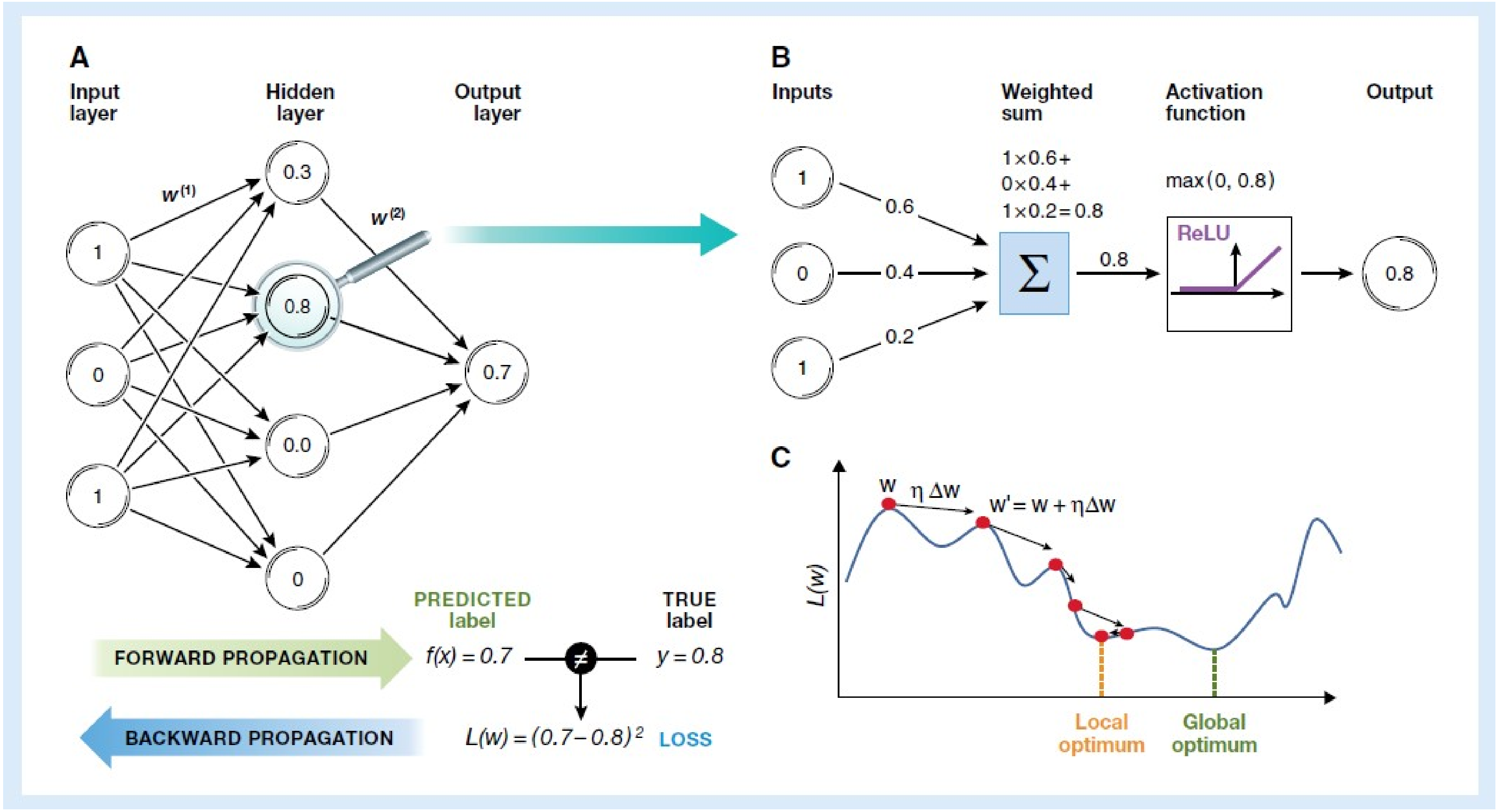
Figure 1. Machine learning and representation learning.
(A) The classical machine learning workflow can be broken down into four steps: data pre-processing, feature extraction, model learning and model evaluation. (B) Supervised machine learning methods relate input features *x* to an output label *y*, whereas unsupervised method learns factors about *x* without observed labels. (C) Raw input data are often high-dimensional and related to the corresponding label in a complicated way, which is challenging for many classical machine learning algorithms (left plot). Alternatively, higher-level features extracted using a deep model may be able to better discriminate between classes (right plot). (D) Deep networks use a hierarchical structure to learn increasingly abstract feature representations from the raw data.

Multilayer neural networks and back propagation

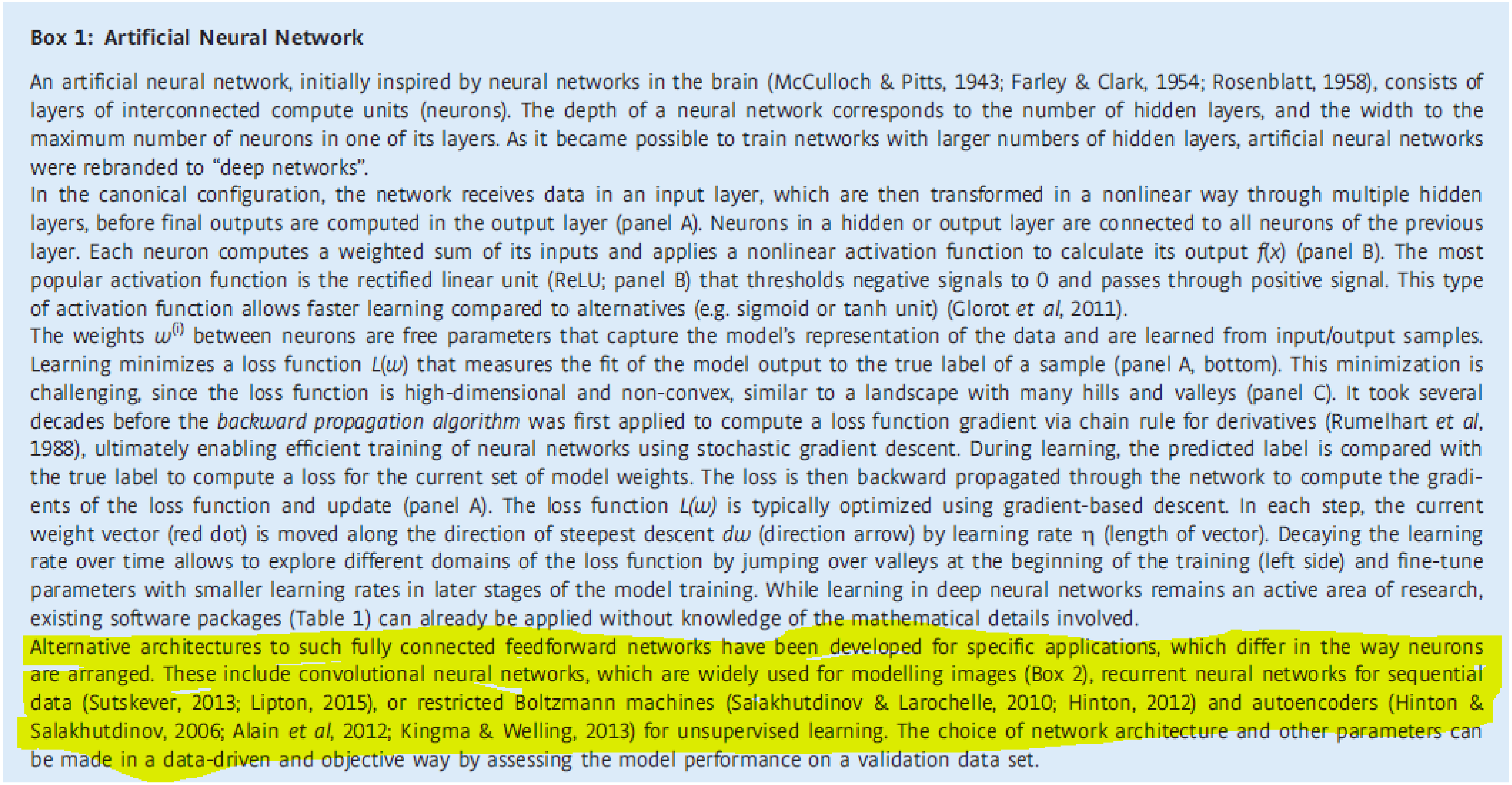
LeCun Y, Bengio Y, Hinton G. Deep learning. nature. 2015 May;521(7553):436-44.


a
Input (2)
Hidden (2 sigmoid)
Output (1 sigmoid)
b
z
y
x
$\frac{\partial z}{\partial y}$
$\frac{\partial y}{\partial x}$
$\Delta z = \frac{\partial z}{\partial y}\Delta y$
$\Delta y = \frac{\partial y}{\partial x}\Delta x$
$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$
c
Output units
Hidden units H2
Hidden units H1
Input units
l
k
j
i
w_{kl}
w_{jk}
w_{ij}
$y_l = f(z_l)$
$z_l = \sum_{k \in H2} w_{kl} y_k$
$y_k = f(z_k)$
$z_k = \sum_{j \in H1} w_{jk} y_j$
$y_j = f(z_j)$
$z_j = \sum_{i \in Input} w_{ij} x_i$
d
Compare outputs with correct answer to get error derivatives
$\frac{\partial E}{\partial y_l} = y_l - t_l$
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$
$\frac{\partial E}{\partial y_k} = \sum_{l \in out} w_{kl} \frac{\partial E}{\partial z_l}$
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk} \frac{\partial E}{\partial z_k}$
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$
w_{kl}
w_{jk}
w_{ij}

Angermueller C, Pärnamaa T, Parts L, Stegle O.Deep learning for computational biology. Molecular systems biology. 2016 Jul 1;12(7).



Angermueller C, Pärnamaa T, Parts L, Stegle O.Deep learning for computational biology. Molecular systems biology. 2016 Jul 1;12(7).



Box 1: Artificial Neural Network

An artificial neural network, initially inspired by neural networks in the brain (McCulloch & Pitts, 1943; Farley & Clark, 1954; Rosenblatt, 1958), consists of layers of interconnected compute units (neurons). The depth of a neural network corresponds to the number of hidden layers, and the width to the maximum number of neurons in one of its layers. As it became possible to train networks with larger numbers of hidden layers, artificial neural networks were rebranded to "deep networks".

In the canonical configuration, the network receives data in an input layer, which are then transformed in a nonlinear way through multiple hidden layers, before final outputs are computed in the output layer (panel A). Neurons in a hidden or output layer are connected to all neurons of the previous layer. Each neuron computes a weighted sum of its inputs and applies a nonlinear activation function to calculate its output $f(x)$ (panel B). The most popular activation function is the rectified linear unit (ReLU; panel B) that thresholds negative signals to 0 and passes through positive signal. This type of activation function allows faster learning compared to alternatives (e.g. sigmoid or tanh unit) (Glorot *et al*, 2011).

The weights $\omega^{(i)}$ between neurons are free parameters that capture the model's representation of the data and are learned from input/output samples. Learning minimizes a loss function $L(\omega)$ that measures the fit of the model output to the true label of a sample (panel A, bottom). This minimization is challenging, since the loss function is high-dimensional and non-convex, similar to a landscape with many hills and valleys (panel C). It took several decades before the *backward propagation algorithm* was first applied to compute a loss function gradient via chain rule for derivatives (Rumelhart *et al*, 1988), ultimately enabling efficient training of neural networks using stochastic gradient descent. During learning, the predicted label is compared with the true label to compute a loss for the current set of model weights. The loss is then backward propagated through the network to compute the gradients of the loss function and update (panel A). The loss function $L(\omega)$ is typically optimized using gradient-based descent. In each step, the current weight vector (red dot) is moved along the direction of steepest descent $d\omega$ (direction arrow) by learning rate η (length of vector). Decaying the learning rate over time allows to explore different domains of the loss function by jumping over valleys at the beginning of the training (left side) and fine-tune parameters with smaller learning rates in later stages of the model training. While learning in deep neural networks remains an active area of research, existing software packages (Table 1) can already be applied without knowledge of the mathematical details involved.

Alternative architectures to such fully connected feedforward networks have been developed for specific applications, which differ in the way neurons are arranged. These include convolutional neural networks, which are widely used for modelling images (Box 2), recurrent neural networks for sequential data (Sutskever, 2013; Lipton, 2015), or restricted Boltzmann machines (Salakhutdinov & Larochelle, 2010; Hinton, 2012) and autoencoders (Hinton & Salakhutdinov, 2006; Alain *et al*, 2012; Kingma & Welling, 2013) for unsupervised learning. The choice of network architecture and other parameters can be made in a data-driven and objective way by assessing the model performance on a validation data set.

Feed forward artificial neural net – multilayer perceptron

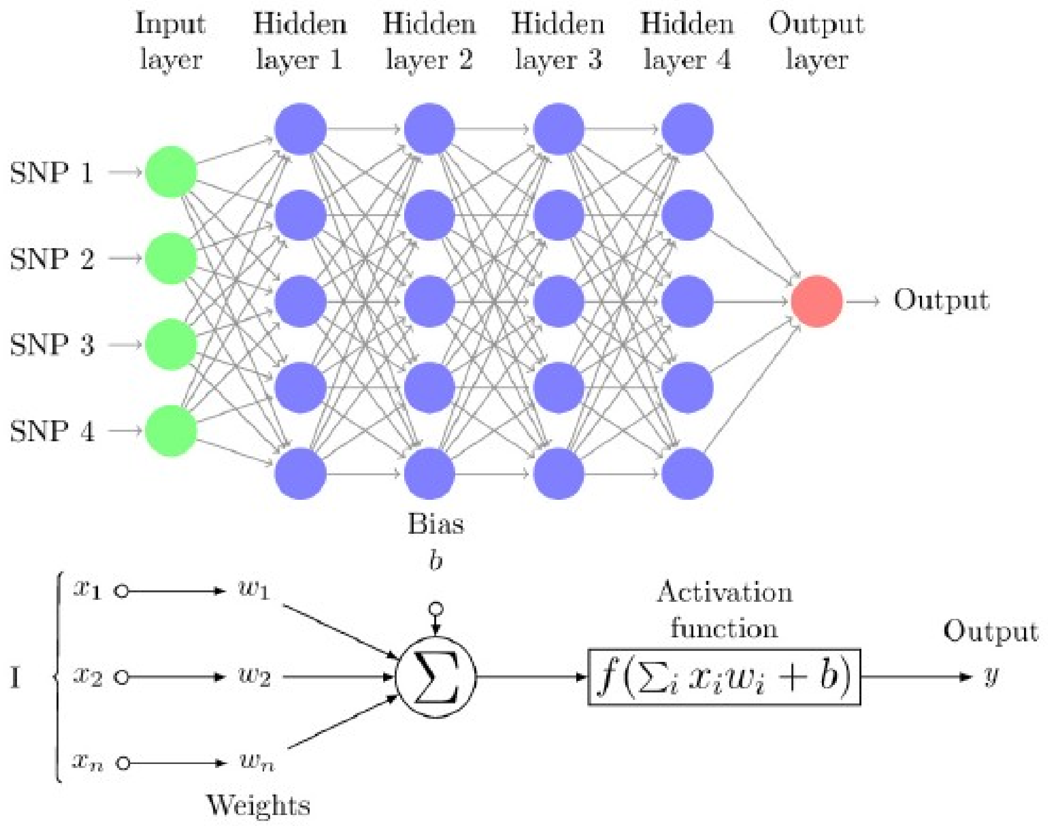


Figure 1. Multi-Layer Perceptron (MLP) diagram with four hidden layers and a collection of single nucleotide polymorphisms (SNPs) as input and illustrates a basic "neuron" with n inputs. One neuron is the result of applying the nonlinear transformations of linear combinations (x_i, w_i, and biases b). These figures were redrawn from tikz code in http://www.texample.net/tikz/examples/neural-network.

Pérez-Enciso M, Zingaretti LM. A Guide on Deep Learning for Complex Trait Genomic Prediction. Genes. 2019 Jul;10(7):553.

Convolutional neural network

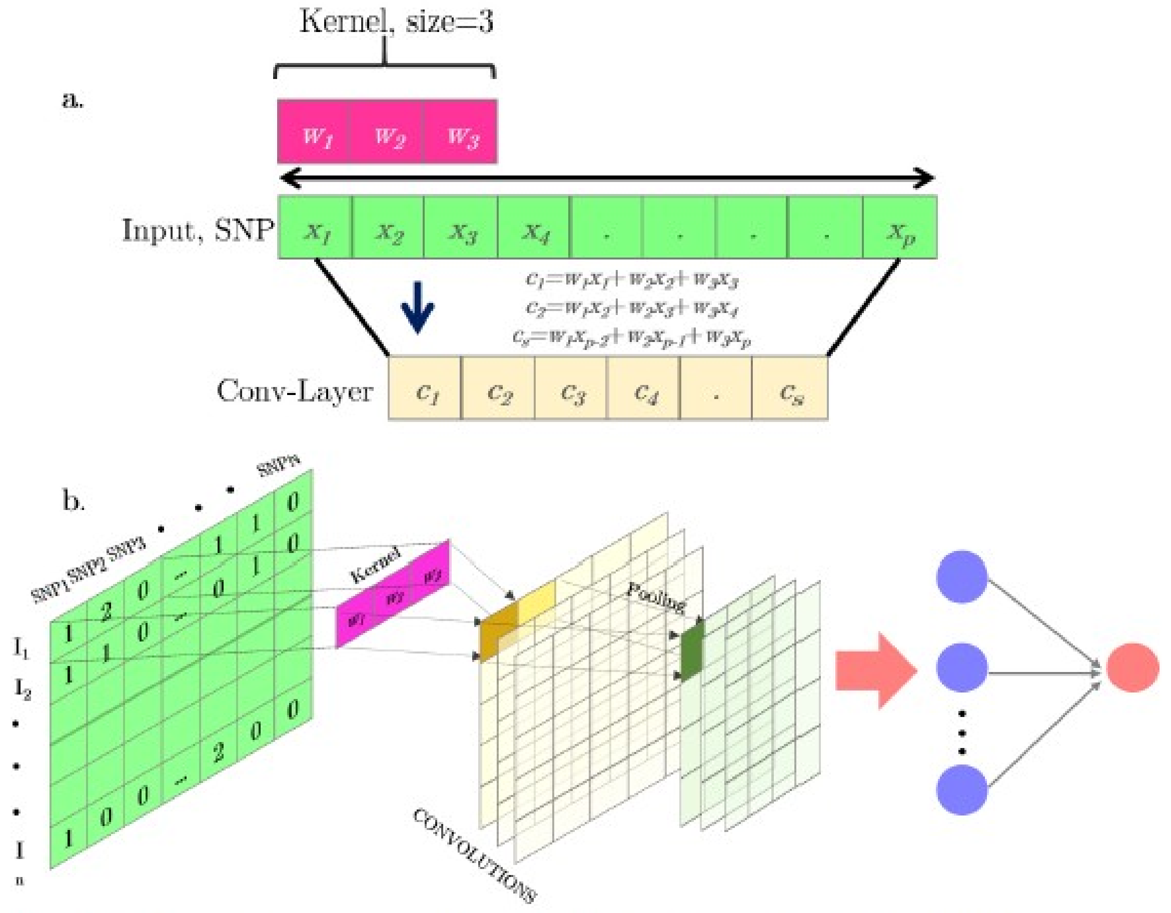

Kernel, size=3
a.
w_1 w_2 w_3
Input, SNP
x_1 x_2 x_3 x_4 . . . . x_p
$c_1 = w_1x_1 + w_2x_2 + w_3x_3$
$c_2 = w_1x_2 + w_2x_3 + w_3x_4$
$c_s = w_1x_{p-2} + w_2x_{p-1} + w_3x_p$
Conv-Layer
c_1 c_2 c_3 c_4 . c_s
b.
SNP1 SNP2 SNP3
SNPs
I_1
I_2
I_n
Kernel
w_1 w_2 w_3
Pooling
CONVOLUTIONS

Figure 2. (**a**) Simple scheme of a one-dimension (1D) convolutional operation. (**b**) Full representation of a 1D convolutional neural network for a SNP-matrix. The convolution outputs are represented in yellow. Pooling layers after convolutional operations combining the output of the previous layer at certain locations into a single neuron are represented in green. The final output is a standard MLP.

Pérez-Enciso M, Zingaretti LM. A Guide on Deep Learning for Complex Trait Genomic Prediction. Genes. 2019 Jul;10(7):553.

Recurrent neural network

Pérez-Enciso M, Zingaretti LM.
A Guide on Deep Learning for Complex Trait Genomic Prediction.
Genes. 2019 Jul;10(7):553.

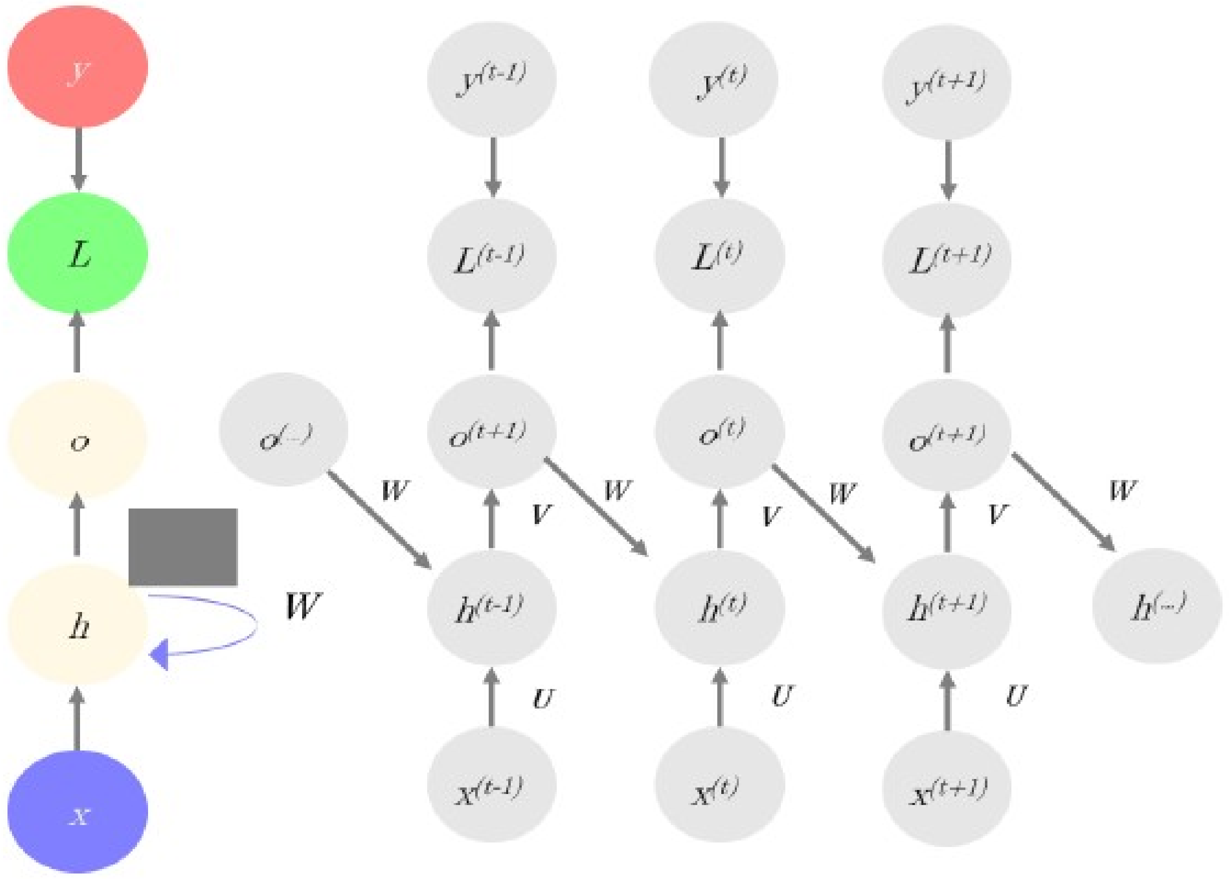


Figure 3. Scheme of recurrent neural networks (RNNs): The left part of the image (in colors) shows the whole network structure; whereas, the recursive structure of the network is shown in the right, where x represents inputs, h are the hidden layers, $o = Vh^{(t)} + b$ are the outputs (Equation (4b)), y are the target variables, and L is the loss function.

Generative adversarial neural network

Pérez-Enciso M, Zingaretti LM.
A Guide on Deep Learning for
Complex Trait Genomic Prediction.
Genes. 2019 Jul;10(7):553.

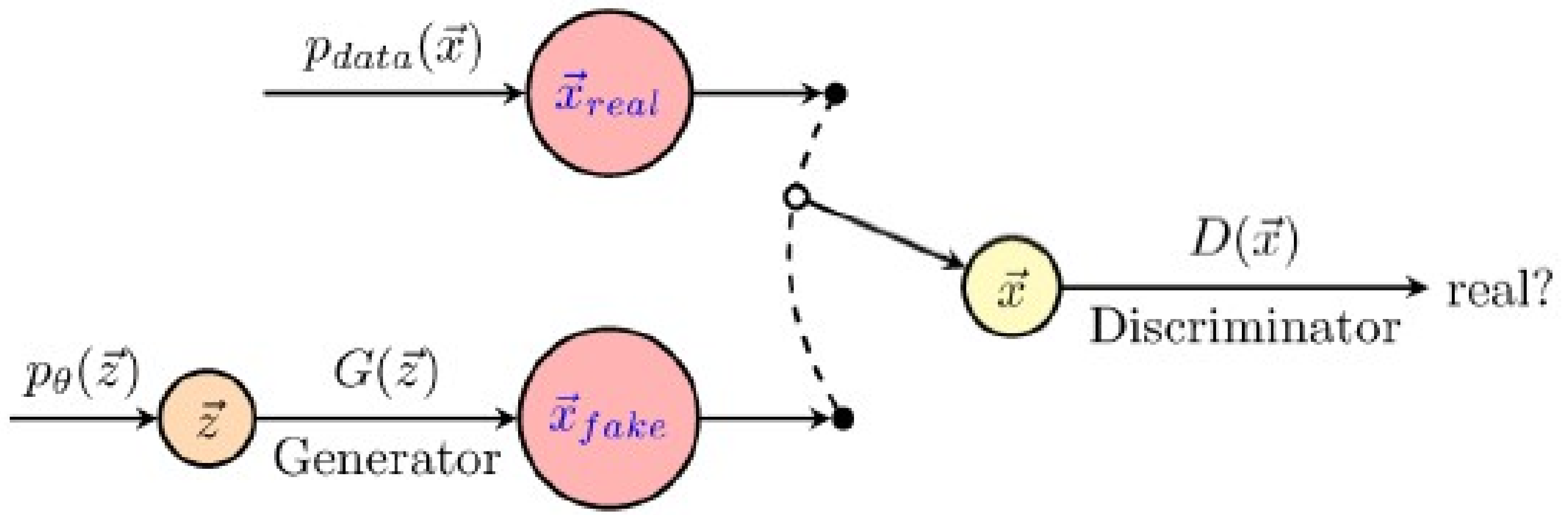
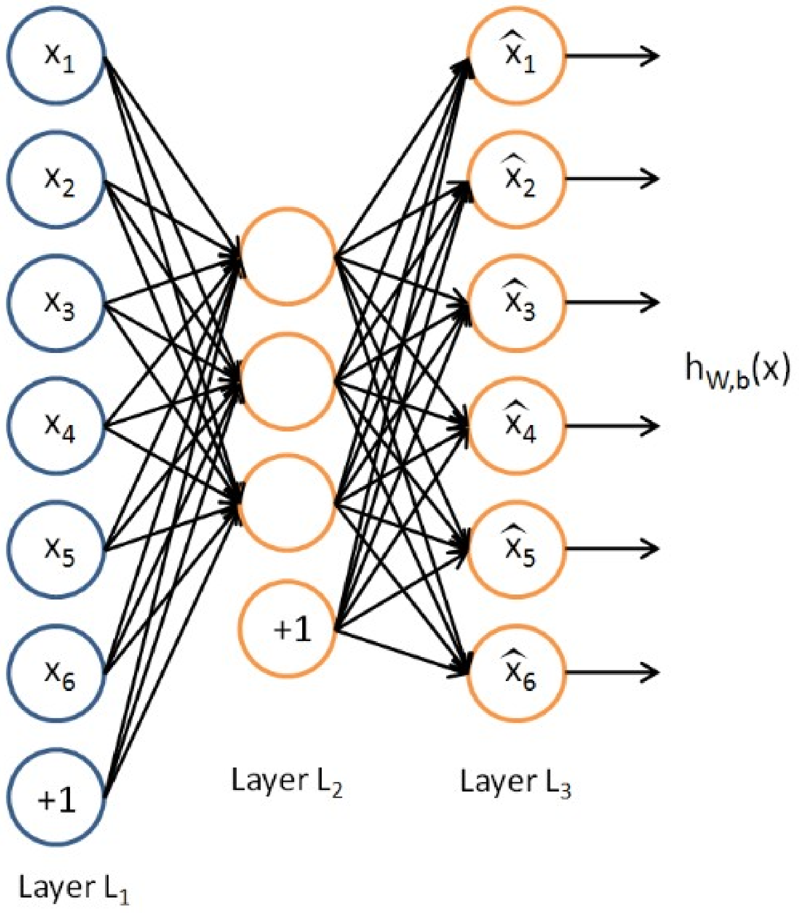


Figure 4. Scheme of generative adversarial networks (GANs). The generator (G), defines a probability distribution based on the information from the samples, whereas, the discriminator (D) distinguishes data produced by G from the real data. The figure was redrawn using code from http://www.texample.net/tikz/examples/neural-network.



Autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs

http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Off the shelf tools (C/C++)

- Caffe – specialized for CNN
- Theano – (NN declared as computational graph) well suited for RNN
  - Keras wrapper
  - Lasagne wrapper
- Torch wriiten in LuaJIT – interpretable , good for CNN
- TensorFlow (NN declared as computational graph) native support for parralelization (CPUs, GPUs) most efficient for RNN

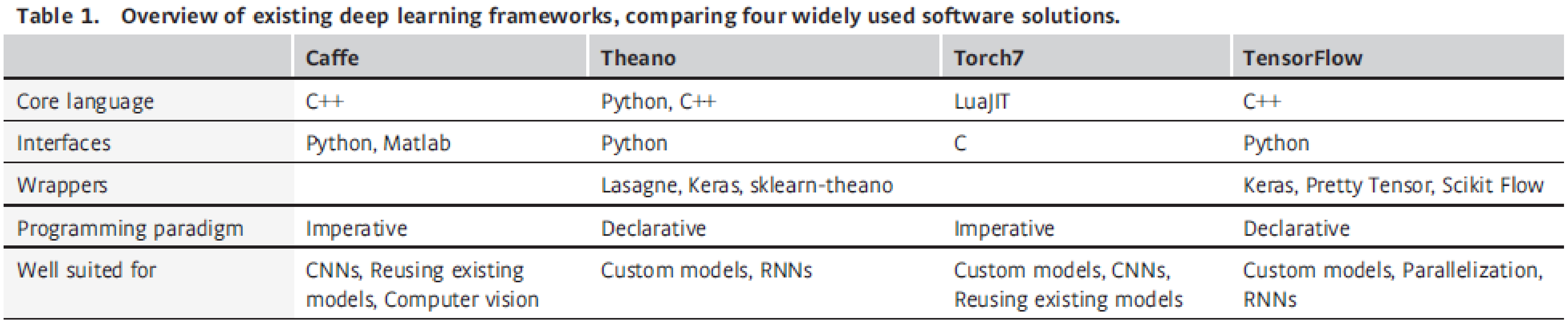


Table 1. Overview of existing deep learning frameworks, comparing four widely used software solutions.

	Caffe	Theano	Torch7	TensorFlow
Core language	C++	Python, C++	LuaJIT	C++
Interfaces	Python, Matlab	Python	C	Python
Wrappers		Lasagne, Keras, sklearn-theano		Keras, Pretty Tensor, Scikit Flow
Programming paradigm	Imperative	Declarative	Imperative	Declarative
Well suited for	CNNs, Reusing existing models, Computer vision	Custom models, RNNs	Custom models, CNNs, Reusing existing models	Custom models, Parallelization, RNNs

Angermueller C, Pärnamaa T, Parts L, Stegle O.Deep learning for computational biology. Molecular systems biology. 2016 Jul 1;12(7).

Data Transformations And Model Trainng

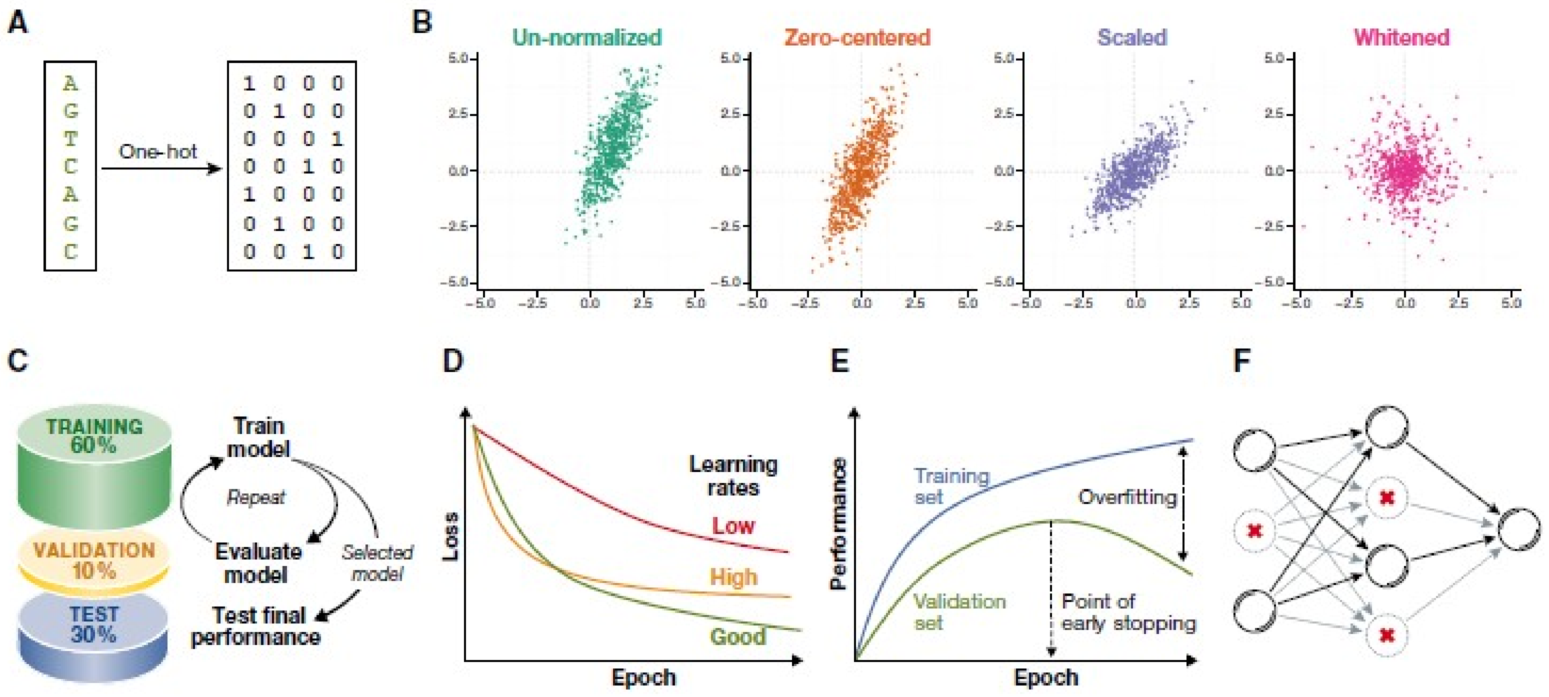


Figure 5. Data normalization for and pre-processing for deep neural networks.
(A) DNA sequence one-hot encoded as binary vectors using codes A = 1000, G = 0100, C = 0010 and T = 0001. (B) Continuous data (green) after zero-centring (orange), scaling to unit variance (blue) and whiting (purple). (C) Holdout validation partitions the full data set randomly into training (~60%), validation (~10%) and test set (~30%). Models are trained with different hyper-parameters on the training set, from which the model with the highest performance on the validation set is selected. The generalization performance of the model is assessed and compared with other machine learning methods on the test set. (D) The shape of the learning curve indicates if the learning rate is too low (red, shallow decay), too high (orange, steep decay followed by saturation) or appropriate for a particular learning task (green, gradual decay). (E) Large differences in the model performance on the training set (blue) and validation set (green) indicate overfitting. Stopping the training as soon as the validation set performance starts to drop (early stopping) can prevent overfitting. (F) Illustration of the dropout regularization. Shown is a feedforward neural network after randomly dropping out neurons (crossed out), which reduces the sensitivity of neurons to neurons in the previous layer due to non-existent inputs (greyed edges).

Articles too choose

- Kelley (2016) – deep learning to predict DNAse 1 hypersensitivity across multiple cell types (Basset)
- Angermueller (2016) CNN to predict DNA methylation states in single cell bi-suphite sequencing
- Lee (2015) RNN to predict DNA sequence (at limited extent)
- Fakoor (2013) Autoencoders to classify cancer cases using gene expression data
- Asgari & Mofrad (2015) Restricted Botzman machines to classify proteins
- Zhu and Troyanskaya (2015) - CNN to predict chromatin marks
- Poplin (2018) Google ai DeepVariant – variant calling based on images application of CNN
- Other articles from the list on the github

Where to search for more information

- https://canvas.stanford.edu/courses/70852
- http://deeplearning.net/tutorial/contents.html
- https://scikit-learn.org/stable/index.html