Supplementary Material A.

The concept of deep learning originated from artificial neural network research, in which feed-forward neural networks or multilayer perceptrons with many hidden layers are referred to as DNNs. These networks are generally trained by a gradient descent algorithm, designated by back-propagation. However, for deep networks, back-propagation alone has several problems, including local optima traps in the non-convex objective function (possible stagnation of algorithm in local minimum and thus insufficient learning outcomes) and vanish gradients (output signal decreases exponentially as information is back-propagated through layers).

In 2006, Hinton et al. proposed an unsupervised learning algorithm using a method called Contrastive Divergence, capable of training deep generative models known as deep belief networks[1]. A deep belief network is composed of a stack of Restricted Boltzmann Machines trained by a greedy, layer-by-layer learning algorithm (Contrastive Divergence). They are normally used for unsupervised tasks, but can be fine-tuned to perform supervised learning by attaching a softmax layer, which allows calculating the probability of output and thus measuring the classification, to the top layer.

There are now many deep learning architectures. The most common architectures are shown in Figure 1 and described in more detail below.

<u>Convolutional Neural Networks (CNN)</u> A CNN is composed by several blocks composed by different types of layers. Each module consists of a convolutional layer and a pooling layer[2]. These modules are often stacked up with one on top of another, or with a softmax logistic layer on top, to form a deep model. The CNN uses several tricks that make them well suited for image processing, such as weight sharing, adaptative filters and pooling. Pooling takes subsamples of the convolutional layer to feed the next layer, acting as a powerful regularizer. Weight sharing and pooling schemes (most usually a max pooling), allow the CNN to evolve invariance properties like translation invariance. CNNs have been found to be highly effective, with high selectivity and invariance,[1] and have been commonly used in computer vision and image recognition[3].

CNNs are also the most common DNN used for biomedical data. They operate on what should be considered a signal stream rather than a feature vector. As opposed to fully connected neural nets consisting of activation units bound to all inputs of a feature vector, where every unit has a weight specific to each feature in the input, convolutional layers utilize weight sharing by sliding a small (trainable) filter of weights across the input vector (or 2D input map, as CNNs are often used on images) and convolving each overlayed region of input with the filter.

In genomics, the concept of weight sharing is very appealing to learning about regulatory elements contained in the randomized regions of the DNA library, since these models can recognize regulatory elements regardless of exact position. Position should only play a minor role in prediction, and this is exactly what the convolutional layers provide: the filters will slide across the input DNA sequence, responding to specific motifs they have been trained to be sensitive towards. However, every overlayed region of the input signal relates to a specific location in the convolved output signal, meaning positional information remains preserved.

Deep autoencoder (DA): a DNN having as outputs the input data itself. If they are trained with some added noise, these architectures can act as generative models. An autoencoder can be trained with a greedy layer-wise mode, much like the DBNs, to form a deep model. Autoencoders can be stacked to form a deep network by forwarding the outputs of the autoencoder in the layer below as input to the layer above. The unsupervised pre-training is done one layer at a time and each layer is trained to minimize the error in reconstructing its input. After pre-training, the network can be fine-tuned by adding a softmax layer and applying supervised backpropagation, as if they were multilayer perceptrons. Deep autoencoders provide a more flexible and rich representation suitable for data with a complex structure, as biological data. The downside of this advantage is complication of the learning process and increasing of computational sources consumption. In addition, the ability to recover the parameters of the distribution function allows us to consider the autoencoder as a particular generative model that implements the possibility of sampling (generating observations of a random variable) synthetic examples of class representatives, the parameters of the autoencoder was able to learn.

Restricted Boltzmann Machine (RBM): a Boltzmann Machine, a stochastic recurrent neural network, formed by two layers - one of visible units and other of hidden units with no lateral connections. One of the most relatively simple to train NN that allows forming compact and high-level representation of objects what is beneficial for computational biologists with no expertise in Deep Learning. In contrast with clustering algorithms or decision trees algorithms (such as Random Forest) that based on local generalization, RBM based on distributed representations and so could perform generalization ability on area with no data[4].

Deep Belief Network (DBN): a probabilistic generative model formed by several stacked hidden layers (normally binary). The top two layers are symmetrically connected through indirect connections. Each layer receives a directed connection from the layer below. DBN are trained by a greedy layerwise strategy using the Contrastive Divergence (CD) algorithm proposed by Hinton et al.[1]. Apart from providing good initialization points, the DBN also have other interesting properties: i) entire data sets, even unlabeled can be used; ii) they can be considered a probabilistic generative model and iii) the over-fitting problem can be effectively alleviated by the pre-training step.

Recurrent Neural Networks (RNN) are a class of unsupervised or supervised architectures for learning temporal, or sequential, patterns, such as language . RNN can be used to predict the next data point in a sequence using the previous data samples. For instance, for text, a sliding window is used to predict the next word or set of words in a sentence. RNNs very powerful but difficult to train in capturing long-term dependencies due to well known gradient vanishing or gradient explosion problems.

(1) Hinton, G. E. Reducing the Dimensionality of Data with Neural Networks. *Science (80-.).* **2006**, *313* (5786), 504–507.

(2) Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. R. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *arXiv: 1207.0580* **2012**, 1–18.

(3) Ciresan, D. C.; Meier, U.; Gambardella, L. M.; Schmidhuber, J. Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition. *Neural Comput.* **2010**, *22* (12), 1–14.

(4) Bengio, Y.; Delalleau, O. On the Expressive Power of Deep Architectures; Kivinen, J., Szepesvári, C., Ukkonen, E., Zeugmann, T., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, **2011**; pp 18–36.

Supplementary Material B.

Framework	**Core language**	**Authors**	**Features**	**Benefits**	**Cons**
Caffe	C++ (Matlab, Python)	Yangqing Jia at BVLC	computer vision oriented	fast, architecture as a file (protobuf),	support only CNN and MLP, hard to extend, non-distributed
Theano	Python	AI community (University of Monreal)	computational graph, automatic differentiation	very flexible, large scientific community, has lots of libraries based on its (Lassagne, Keras, etc.)	Slow in both compilation and runtime, non-distributed
Torch	Lua	AI community	LuaJIT	fast, relatively flexible, easy to integrate	non-distributed, relatively hard to extend,
Chainer	Python	Preferred Networks Inc.	computational graph, automatic differentiation	relatively fast, relatively flexible, define-by-run concept	non-distributed, relatively hard to extend
Neon	Python	Nervana	computational graph, automatic differentiation	very fast, relatively flexible, support visualization tools	non-distributed, relatively hard to extend,small community
TensorFlow	C++ (Matlab, Python)	Google	computational graph, automatic differentiation	flexible, has lots of tools and algos (reinforcement learning, etc.), support visualization tools	slow, not commercially supported, non-distributed
Deeplearning4j (DL4J)	Java	Skymind	distributed library, business oriented	distributed computations, native to Hadoop, Spark	relatively slow, hard to extend,