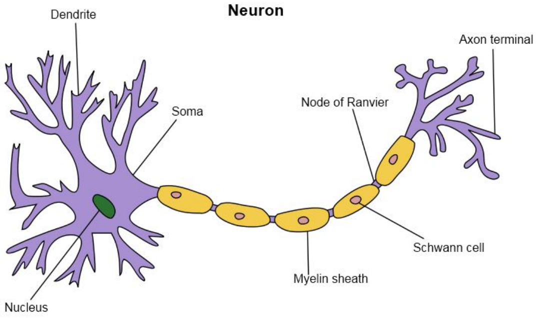
Evolutionary algorithms

Neural networks

About the structure of the brain

The smallest unit of the nervous system is the neuron. Neuron is a combination of the nerve cell and all of its protuberances. Neurons are irritable cells that specialize in stimulating and driving nerves.

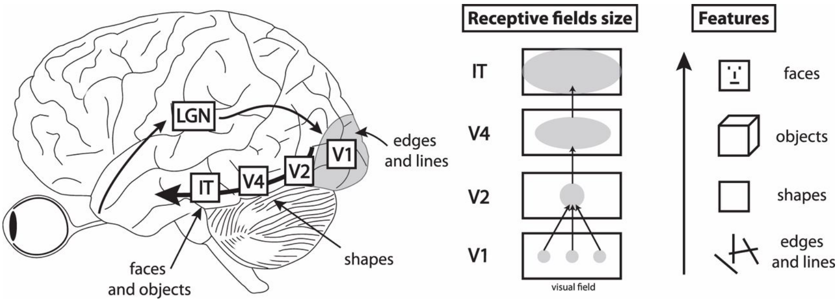


About the structure of the brain II

The information arrives via dendrites. There is a single output of the cell called the axon. The brain consists of a network of such nerve cells, partly with known topology (areas that perform a particular function, and within it, can be divided into levels).

For example, about one third of the brain surface consists of neurons specialized in image processing, which are organized into different levels. At lower levels, the brain recognizes and determines the direction, recognizing forms and objects at higher levels.

Seeing



Mauro Manassi; Bilge Sayim; Michael H. Herzog (Journal of Vision)

Neural networks

The very simple artificial imitation of the brain is the neural network, the most advanced image and text processing tool used today. From a mathematical point of view, he approximates a nonlinear function with the composition of simple functions.

The neural network consists of a linked topology of localized processing elements (neurons).

When constructing a neural network, we must plan the elemental neurons and the connection (topology) between them. This network still has many free parameters. The optimization of free parameters is called training. With the given parameters calculating the output from the input is the retrieval.

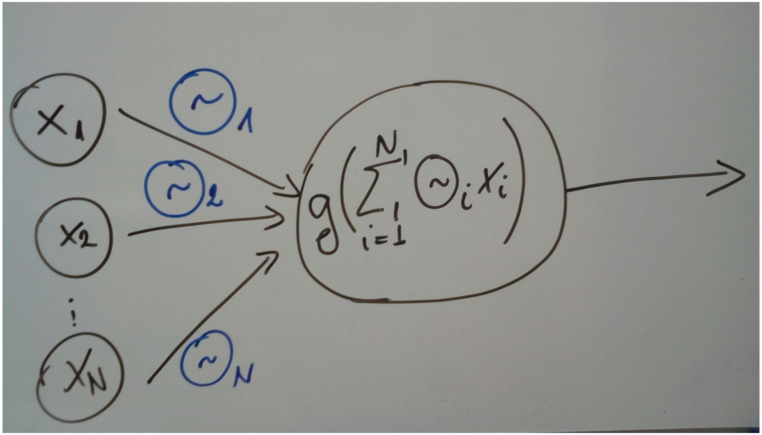
MLP networks

An elemental neuron has several inputs and one output. In the simplest case, the output depends on the input through a linear combination of inputs has the form of

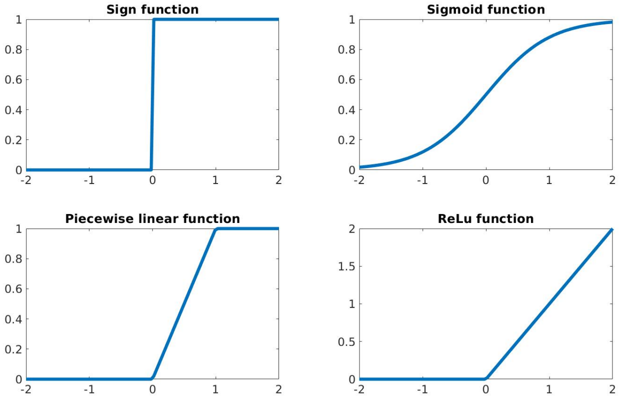
$$g\left(\sum_{i=1}^{N}\theta_i x_i\right)$$

θ_i are the weights, $g$is the activation function. The latter may be, for example, a sign function or a smoother, sigmoid type. There is a linear (but continuous) function, which can be considered as a transition between the two.

A neuron



Activation function



Topology

The topology of the network can be described by a directed graph that tells the input and output of each neuron. Based on the type of edges, one neuron can be

- input (they get the input of the network and forward to the other neurons),
- output (forward information to environment)
- hidden (their input and output can only be another neuron).

Topology

In the simplest case (forward multilayer network), the same type of neurons form layers: there is an input, an output, or any number of hidden layers, from which all edges are forward (from the input layer to the output layer).

This structure means a lot of unknown independent weight. However, in some tasks, it is possible to simplify: for example, when image recognition is performed by certain groups on the same part of the image, we can take the same weights in each of these groups.

Theorem

Theorem (Cybenko, 1989)
Let $f(\underline{x})$ be a continuous function on $K \subset \mathbb{R}^n$ compact set, and $\varepsilon > 0$ is arbitrary. Then there is a neural network with one hidden layer and sigmoid activation function, which has the precision:

$$\max_{x \in K} |f(\underline{x}) - h_\theta(\underline{x})| < \varepsilon$$

The idea of the proof is the uniform continuity of the $f(\underline{x})$ function, which makes it approachable with a step function. The step function can be approached with the neural network, but the number of required neurons is an open question. Later, this theorem was significantly generalized by others. Although the proof is constructive, it can't be used in practice to build a net, because the number of necessary neurons might be to much.

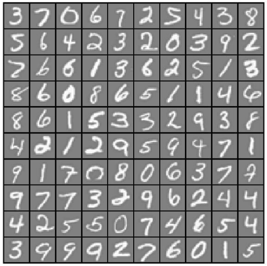
CNN

Neural nets designed for a specific problem mimic the hierarchy of the brain area that solves the problem: for example, convolutional nets are used in image processing have many identical smaller units are connected. A low-level component solves a simpler problem for a portion of the image (such as the upper left corner): for example, it detects whether there is a vertical line in a given sub-picture. The output of these low-level components is used by the next level of the neural network as input.

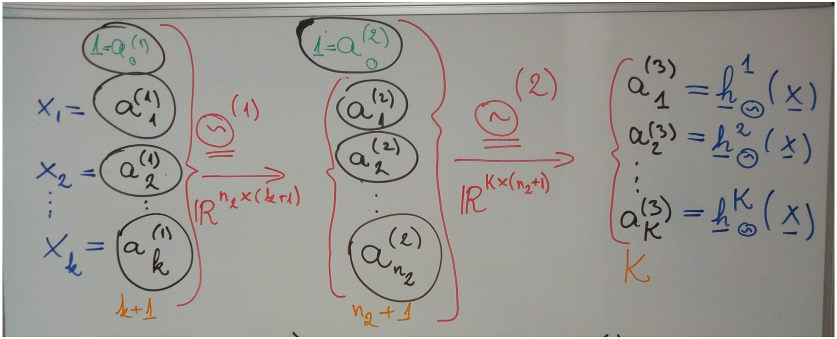
A little information about how many neurons and layers in a neural network today: CNN

Classification: an example

Let k be the dimension of the input. We have a classification task, that is, we want to determine which of the K classes our input belongs to. An example: we have 20×20 pixel gray scale pictures. We want to find out which digit is on the picture. There are a total of 100 sample inputs here, each has a label, which tell us what digit is on the picture. This is our training set.

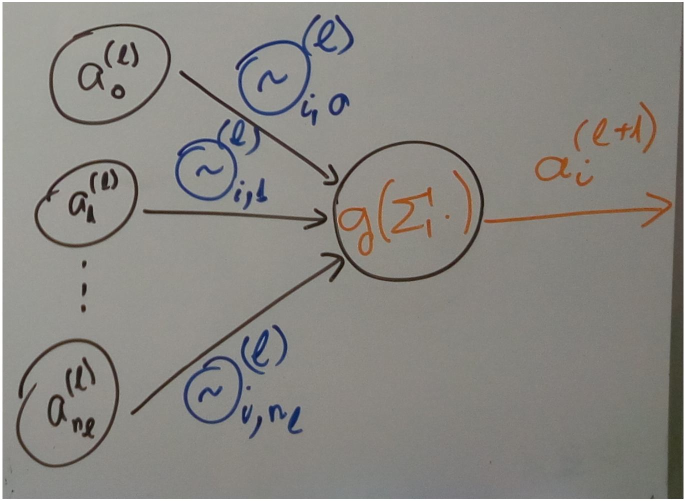


A NN with one hidden layer



Green labeled neurons are the so-called 'bias' units. In the example, $k = 400$ and $K = 10$. The output on each of the 10 neurons is a number between 0 and 1, which is the highest, that will be our guess for the class.

A NN with one hidden layer



This is how one neuron looks like.

Finding the optimal weights

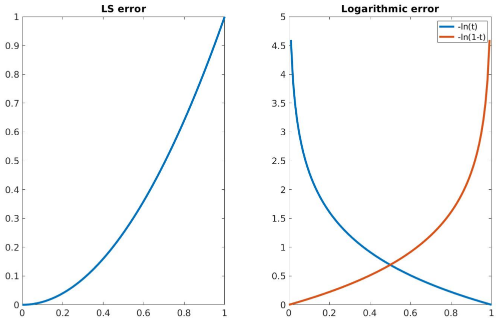
If the net topology is given, the determination of unknown weights (all θ matrices) is the next task. The value of the function to be approximated at each of the (typically many) training points is known (in our example the 0-9 digits).

Thus, it is possible to determine the total error of the net with the given weights, our aim is to minimize it. This is a function optimization task in high dimension.

In this example, we have a total of $401 \cdot 25 + 26 \cdot 10 = 10285$ θ-s, this is the dimension of the task.

Measuring the error

To solve the function optimization task, we have to assign a metric to the deviation of the neural network output and the desired output. In the case of a sigmoid activation function, the logarithmic penalty function is appropriate. In other cases, the smallest square deviation comes into play.



Measuring the error

Suppose we have m learning points, for each we know which class they belong. The value of y is 1 for this class, the rest is 0. Then the cost function is:

$$J(\theta) = \frac{-1}{m} \sum_{\text{trainingpoints}} \sum_{k=1}^{\text{classes}} y \log(\text{k.output}) + (1-y) \log(1-\text{k.output})$$

If the net perfectly matches our data (which is not a goal!) then it would be 0, otherwise it is positive. Our goal is to find the θ weights for which this cost is minimal.

Finding the optimal weights

An option (and today considered as the most advanced method) is to calculate optimal weights using (stochastic) gradient method. To do this, we need to calculate the derivative of all $\theta_{i,j}^{(l)}$ of the above cost function.

For the derivatives we can write a recursive formula using the chain rule, starting from the output layer and going backwards on the layers (that is why it is called 'back-propagation of error').

Backpropagation

If we use the $g(z) = \frac{1}{1+\exp(-z)}$ sigmoid activation function with the cost-function above then we have the following recursion for the partial derivatives:

$$\underline{\delta}^{(L)} = \underline{h}_\theta(\underline{x}) - \underline{y}$$

$$\underline{\delta}^{l} = \underline{\underline{\hat{\theta}}}^{(l)} \underline{\delta}^{(l+1)} .* g'(\underline{\underline{\theta}}^{(l-1)} \underline{a}^{(l-1)}),$$

where the $\underline{\underline{\hat{\theta}}}^{(l)}$ matrix is obtained by deleting the weights of the bias units. Using these $\underline{\delta}$ quantities:

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} = \frac{1}{m} \sum_{trainingexamples} \delta_i^{(l+1)} a_j^{(l)}$$

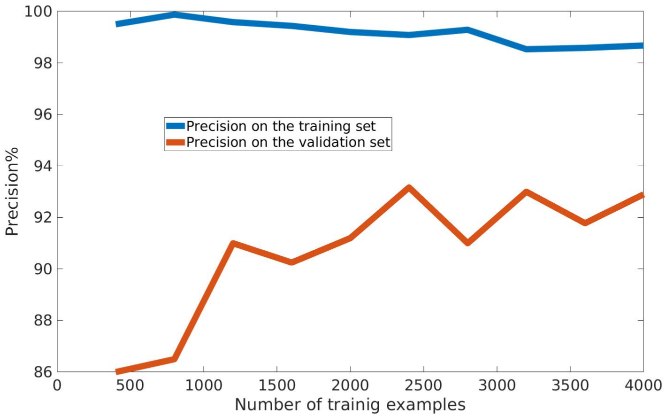
Validation

At the end of the learning process, we test on a validation set how well the 'optimal' weights solve the problem. To do this, we divide our learning points into two parts at first. Some of the data (about 75-80%) is used to adjust the weights. The remainder is used for validation, the error on this set is the net error of the net.

A large net is particularly exposed to the risk of overfitting. Although the cost will be 0, we have not learned the task, but the test-set. To avoid this, one term is added to the cost function, the sum of the squares of all weights weighted by some parameter. Of course, the gradient must be recalculated.

Learning curve

It is worth observing how changing the number of examples will change the validation error. Here is the precision of a net with 25 hidden neurons, depending on the number of examples.



Learning curve

If the validation error and the error on the training set are still far from each other, then we probably have many variables compared to the number of examples (high variance). In this example, we have a total of 10285θ and 4000 training points. From a larger data set, we can expect better alignment of parameters, but it is costly to acquire, and more time to teach the net.

If the validation error and the error on the training set are close to each other but equally high, then the model is bad, no further learning data can be expected. (high bias)

Measuring the performance

Suppose we have the task of deciding whether or not the 9 digit is in the picture. Would we be satisfied with a 90% performance? However, if the numbers are uniformly distributed and the net responds 'not 9' regardless of the data, then it achieves this performance.

That's why we introduce the F_1 score to measure performance. Consider a classification problems with two classes where only positive / negative answers has to be given. The 'recall' of the model is the ratio of the true positive to all positive cases. 'Precision' is the ratio of true positive to all positive predicted cases. The harmonic mean of the two ratios is F_1 score.

F_1 score

Predicted \ Reality	Positive	Negative
Positive	True Positive (TP)	False Positive (FP)
Negative	False negative (FN)	True negative (TN)

The recall:

$$R = \frac{TP}{TP + FN}$$

precision:

$$P = \frac{TP}{TP + FP}$$

and the F_1 score is:

$$F_1 = \frac{2PR}{P + R}.$$