Neural Networks

Fabio G. Cozman - fgcozman@usp.br

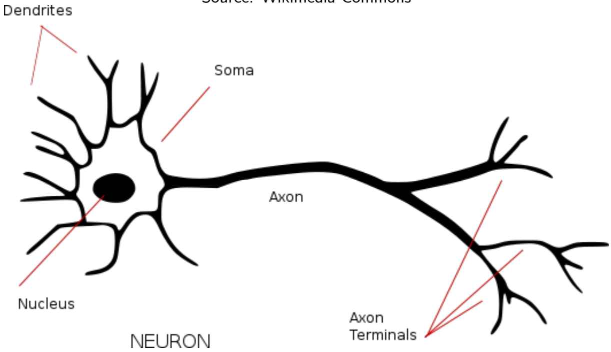
November 1, 2022

A bit of history about neural networks

- Interest in neural networks started decades ago.
  - McCulloch-Pitts neuron model: 1943.
  - Perceptrons (Rosenblatt): 1958.
  - Adaline (Widrow and Hoff): 1960.
- Minsky and Papert blew perceptrons in 1969.
- Multi-layer perceptrons and backpropagation, Hopfield networks, radial basis functions, during the eighties.
- Big thing during the 90s.
- Big crash around 2000.
- Big fever since 2012 (deep learning).

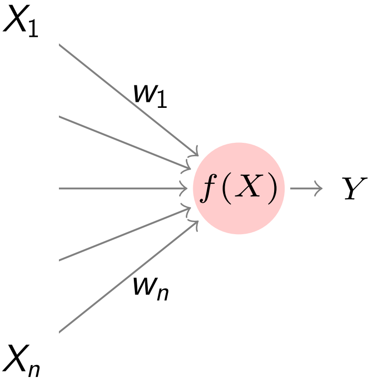
The idea

Source: Wikimedia Commons



$$\sum \text{inputs} \;\rightarrow\; \text{nonlinear function} \;\rightarrow\; \text{outputs}$$

Perceptron

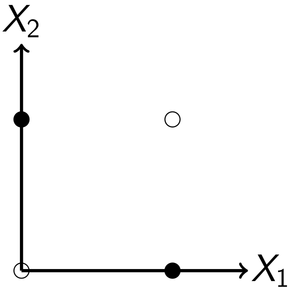


- Output: $Y = f(w_0 + \sum_{i=1}^{n} w_i X_i)$
  (w_0 is the "bias").
- Function $f(X) = 1$ if $X \geq 0$, and -1 otherwise.

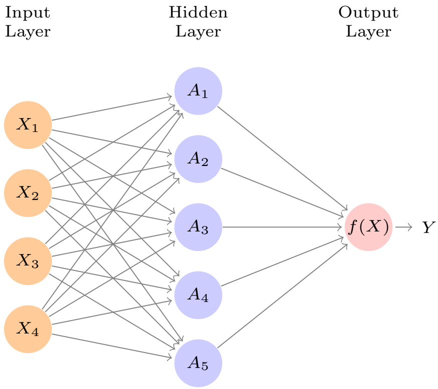
Representation power

- Perceptrons can solve linearly separable problems.
- Non-linearly separable functions... fail.
- Famous example: XOR (exclusive-OR).

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



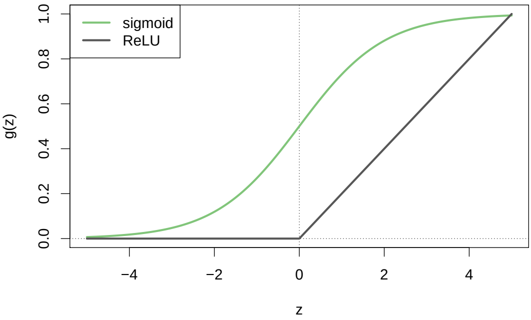
Multi-Layer Perceptron (MLP)



$$Y = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

Units and their activation functions

- Classic: sigmoid.
- Modern: rectified linear (ReLU).

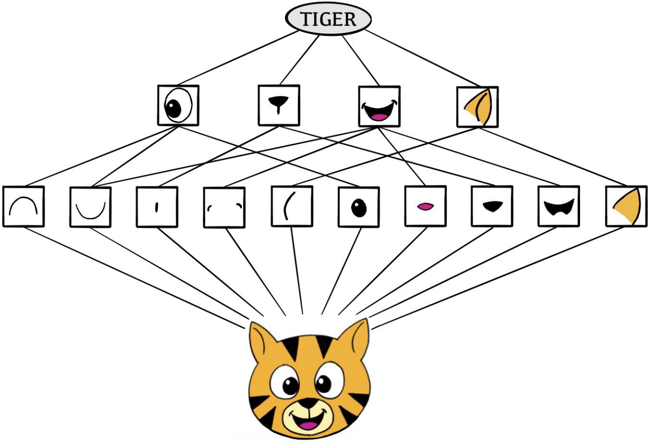


Some notes

- With a single hidden layer:
  $Y = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$.
- Activation function $g(\cdot)$ must be non-linear; otherwise we get a linear structure.
- Activation function generates a "derived feature" by combining features.

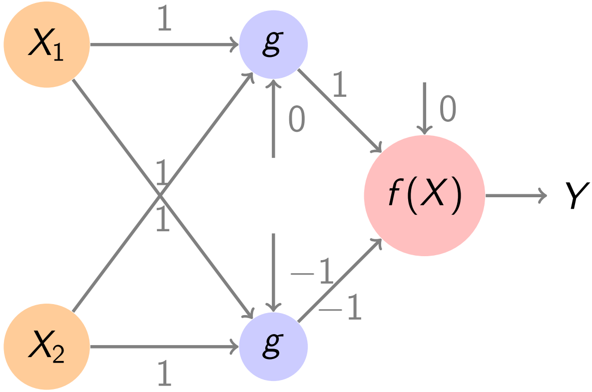
- Universal Approximation Result (1989): an MLP with a single hidden layer can approximate any continuous function arbitrarily well.

Hidden units as derived features



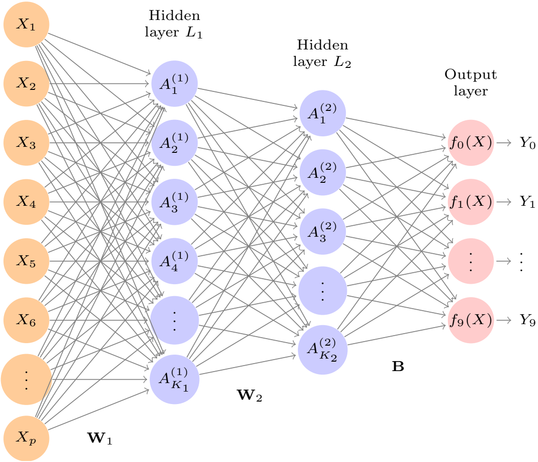
Example

- Take $g(z) = 1$ if $z \geq 1/2$, 0 otherwise.



- What is Y for binary X_1 and X_2?

Many outputs



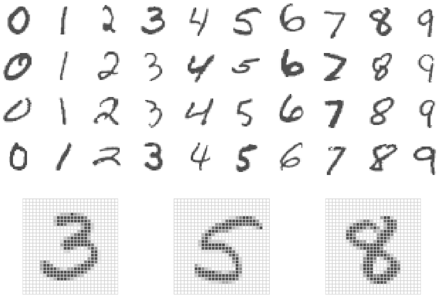
Often, output $f_m(X)$ is interpreted as $\mathbb{P}(Y = m|X)$.

Softmax function

- Define $Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}$.
- Compute:
$$f_m(X) = \frac{e^{Z_m}}{\sum_t e^{Z_t}}.$$
- Interpret $f_m(X)$ as $\mathbb{P}(Y = m|X)$.

Example: MNIST

- Famous dataset with handwritten digits: 60K train, 10K test images (28×28, greyscale).
- Each pixels is a feature, each label is a number from 0 to 9.



Example: MNIST results

- Linear Discriminant Analysis: 12.7% empirical error.
- MLP with two hidden layers (256 units then 128 units; total of 235146 weights!): 1.8% empirical error with backpropagation and dropout regularization.

Learning an MLP

- First, select number of layers, width of layers, activation functions.
- Then, minimize a *loss function*.
- For regression problems, popular loss function:

$$(1/2)\sum_{j=1}^{N}(y_j - f(x_j))^2 ,$$

where $f(x_j)$ is the output of the MLP for the jth training input.

Minimizing loss: Gradient descent

- Goal: find set w of weights that minimize $R_w = (1/2)\sum_{j=1}^{N}(y_j - f(x_j))^2$.

- Start with a set of weights (guess).
- Then iterate:
  - Find a small change, using gradient, to the set of weights such that the new weights reduce the loss.

- Note: given non-linear functions in MLP, no guarantees of global optimality.

Gradient descent

- Gradient vector at iteration t,

$$\nabla R_w^t = \left.\frac{\partial R_w}{\partial w}\right|_{\theta^t}$$

  is the vector of derivatives with respect to weights at iteration t.
- Gradient vector points uphill, so we must take

$$w^{t+1} \leftarrow w^t - \rho \nabla R_w^t,$$

  where ρ is the *learning rate*.

Backpropagation

- Assume loss is a sum over observations; then focus on each observation and add results.
  - For instance, for $R_w = (1/2)\sum_{j=1}^{N}(y_j - f(x_j))^2$, focus on $R_{w,j} = (1/2)(y_j - f(x_j))^2$.
- Key insight: for efficient calculation of derivatives, go backwards from output.

Backpropagation: One hidden layer

- Then $R_{w,j} = (1/2)(y_j - f_w(x_j))^2$, so

$$R_{w,j} = (1/2)\left(y_j - \beta_0 - \sum_{k=1}^{k} \beta_k\, g(z_{jk})\right)^2,$$

where $z_{jk} = \left(w_{k0} + \sum_{i=1}^{P} w_{ki} x_{ji}\right)$.

- Hence:

$$\frac{\partial R_{w,j}}{\partial \beta_k} = \frac{\partial R_{w,j}}{\partial f_w(x_j)} \times \frac{\partial f_w(x_j)}{\partial \beta_k} = -(y_j - f_w(x_j))g(z_{jk}).$$

Backpropagation: One hidden layer

- Then
  $R_{w,j} = (1/2)\left(y_j - \beta_0 - \sum_{k=1}^{k} \beta_k \, g(z_{jk})\right)^2$,
  where $z_{jk} = \left(w_{k0} + \sum_{i=1}^{P} w_{ki}x_{ji}\right)$.
- Hence:

$$\begin{aligned}\frac{\partial R_{w,j}}{\partial w_{ki}} &= \frac{\partial R_{w,j}}{\partial f_w(x_j)} \times \frac{\partial f_w(x_j)}{\partial g(z_{jk})} \times \frac{\partial g(z_{jk})}{\partial z_{jk}} \times \frac{\partial z_{jk}}{\partial w_{ki}} \\ &= -(y_j - f_w(x_j))\beta_k \frac{\partial g(z_{jk})}{\partial z_{jk}} x_{ji}.\end{aligned}$$

Backpropagation: Chain rule

- Output can be viewed as a composition of (vector-valued) functions:

  $$Y = \beta_0 + W_0 g_1(\beta_1 + W_1 g_2(\beta_2 + W_2 g_3(\dots))),$$

  so we can use chain rule to combine intermediate values.

- This matrix-based representation is useful to understand operations; implementation usually resorts to message passing schemes in the network.

Lots of tricks

- Gradient descent must have small steps; the process is (very) slow!
- Early stopping seems to help against overfitting (stop when things are still improving).

Lots of tricks

- Gradient descent must have small steps; the process is (very) slow!
- Early stopping seems to help against overfitting (stop when things are still improving).
- Stochastic gradient: compute the gradient with respect to a subset of the training datasets, selected randomly (this dataset is called "minibatch", usually small).

Lots of tricks

- Gradient descent must have small steps; the process is (very) slow!
- Early stopping seems to help against overfitting (stop when things are still improving).
- Stochastic gradient: compute the gradient with respect to a subset of the training datasets, selected randomly (this dataset is called "minibatch", usually small).
- It is very often useful to augment data.
- It is very often useful to regularize:
    - Penalty on weights (for instance, ridge regularization).
    - Popular: dropout learning.

Dropout learning

- At each step of stochastic gradient descent, "drop out" units.
- Drop out a unit with probability p by setting weights related to it to zero.
- Scale up the weights of the remaining units by $1/(1-p)$.

Classification

- For classification, previous loss is not the best.
  - For instance, for the MNIST dataset.

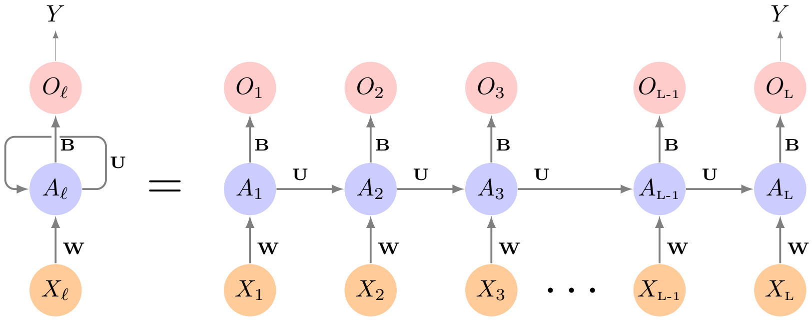
- Option: negative multinomial log-likelihood (also known as *cross-entropy*).

$$-\sum_{j=1}^{N}\sum_{m=0}^{M} y_{jm} \ln f_m(x_j),$$

where M is the number of labels.

Recurrent Neural Networks (RNNs)

- Consider an indexed sequence of observations.
  - Indexed by time, by position in sentence, etc.
  - For instance, time-series of inflation.
- RNN architecture:



Note: weights are the same for all steps.

Deep Learning in practice

- Deep learning has surprising success when datasets are large (overfitting does not seem to be so important).
- For "small" datasets, with "large" noise, simpler models do well, often better.

A note

Some of the figures in this presentation are taken from *An Introduction to Statistical Learning, with applications in R, second edition* (Springer, 2021) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.