92586 Computational Linguistics

Lesson 12. "More than One" Neuron

Alberto Barrón-Cedeño

Alma Mater Studiorum-Università di Bologna
a.barron@unibo.it @_albarron_

15/04/2021

Previously

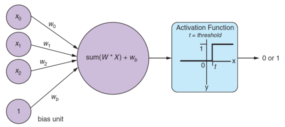
- The perceptron
- Intro to neural networks

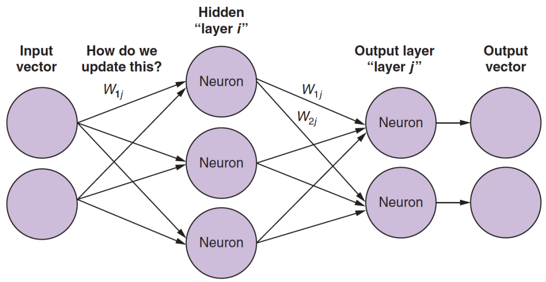
Table of Contents

Backpropagation (brief)

Weight Updating

Learning in a "simple" perceptron[1] vs a fully-connected network

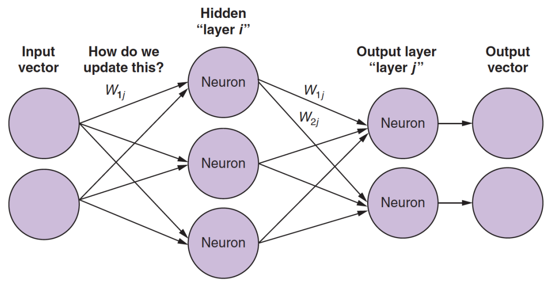




(Lane et al., 2019, p. 158, 168)

[1] Remember: aka linear regression

Backpropagation (of the errors)



- The error is computed on the output vector
- How much error did W_{1i} "contribute"?[2]
- "Path": $W_{1i} \rightarrow [W_{1j}, W_{2j}] \rightarrow output$

[2] Notice that the first W_{1j} should be W_{1i}

Backpropagation (of the errors)

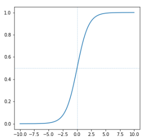
A better activation function

Step function: $f(\vec{x}) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} x_i w_i > threshold \\ 0 & \text{otherwise} \end{cases}$

Sigmoid function: non-linear[3] and continuously differentiable

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

Let us see



Non-linear $\rightarrow$ model non-linear relationships

Continuously differentiable $\rightarrow$ partial derivatives wrt various variables to update the weights

[3] The change of the output is not proportional to the change of the input.

Backpropagation

Differentiating to adjust

Squared error (in (Lane et al., 2019, p. 171) they say this is MSE; wrong)

$$SE = (y - f(x))^2 \tag{2}$$

Mean squared error

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y - f(x))^2 \tag{3}$$

Calculus chain rule

$$f(g(x))' = F'(x) = f'(g(x))\, g'(x) \tag{4}$$

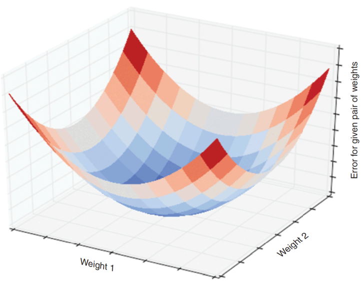
With (4) we can find the derivative of the actfunct $\forall$ neuron wrt its input.

Plain words: find the contribution of a weight to the error and adjust it!

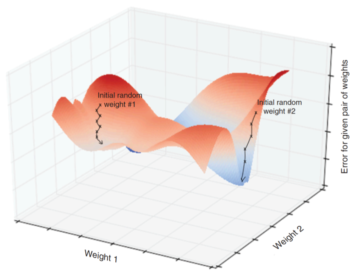
(no further math)

Backpropagation (of the errors)

∼Gradient descent: minimising the error



Convex error curve



Non-convex error curve

(Lane et al., 2019, p. 173–174)

Addressing Local minima

Batch learning

- Aggregate the error for the batch
- Update the weight at the end
- → hard to find global minimum

Stochastic gradient descent

- Look at the error for each single instance
- Update the weights right away
- → more likely to make it to the global minimum

Mini-batch

- Much smaller batch, combining the best of the two worlds
- → Fast as batch, resilient as stochastic gradient descent

Important parameter: learning rate α
A parameter to define at what extent should we "correct" the error

Keras

Some Available Libraries

There are many high- and low-level libraries in many languages

- **PyTorch**
  Community-driven; `https://pytorch.org/`
- **Theano**
  MILA (UdeM);
  `www.deeplearning.net/software/theano/`[4]
- **TensorFlow**
  Google Brain; `https://www.tensorflow.org/`
- **Others**

We will use **Keras**; `https://keras.io/`

[4] Non active

What is Keras

- High-level wrapper with an accessible API for Python
- Gives access to three alternative backends
  - Theano
  - TensorFlow
  - CNTK (MS)

Keras

Logical exclusive OR in Keras

input		output
0	0	0
0	1	1
1	0	1
1	1	0

📘 Let us see

- First dense layer
  - 2 inputs, 10 neurons
  - 30 parameters
  - $2 \times 10 \rightarrow 20$
  - But we also have the bias! That's 10 more weights
- Second dense layer
  - 10 inputs, 1 neuron
  - 11 parameters

Now we can compile the model

📘 Let us see

Some Guidelines

Design Decisions

Activation functions

Sigmoid

ReIU Rectified linear unit (and variations)

tanh Hyperbolic tangent

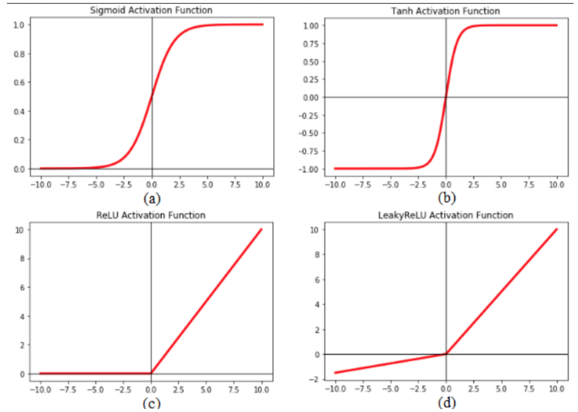


Figure source: (Kandel and Castelli, 2020)

Design Decisions

Activation functions

- Sigmoid
- RelU (rectified linear unit)
- tanh (hyperbolic tangent)

Learning rate

- Choosing one in advance
- Use **momentum** to perform dynamic adjustments

Dropout

- Ignore randomly-chosen weights in a training pass to prevent overfitting

Regularisation

- Dampen a weight from growing/shrinking too far from the rest to prevent overfitting

Normalisation

Example House classification.

Input number of bedrooms, last selling price

Output Likelihood of selling

Vector `input_vec = [2, 90000]`

All input dimensions should have comparable values

Ideally, all features should be in the range $[-1, 1]$ or $[0, 1]$

Typical normalisation: mean normalization, feature scaling, coefficient of variation

NLP uses TF-IDF, one-hot encoding, word2vec (already normalised!)

References

Kandel, I. and M. Castelli
2020. Transfer learning with convolutional neural networks for diabetic retinopathy image classification. a review. *Applied Sciences*, 10(6).

Lane, H., C. Howard, and H. Hapkem
2019. *Natural Language Processing in Action*. Shelter Island, NY: Manning Publication Co.