Neural Networks and Deep Learning

Mohamed Hesham Ibrahim Abdalla, Slim Abdennadher, and Alia Elbolock

German University in Cairo, Cairo, Egypt

Abstract. The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: Neural Networks · Deep Learning · Machine Learning

1 Introduction

Artificial Neural Networks (ANNs) and perceptrons are intelligent units that have taken inspiration from biology, especially the brain [16]. ANNs work by taking labeled inputs and then trying to find a mathematical rule or function that could answer the question of which label belongs to which input, and later identify labels of new inputs that have never been seen before by the network. For example, inputs could be human faces (as images) and the labels are a binary value for the gender of that face.

The history of ANNs and perceptrons goes back to the 50's and the 60's when the first known perceptrons was created. In 1958 Frank Rosenblatt began working on the perceptrons. This perceptrons was build completely on hardware and the idea behind it is very similar to modern neurons or perceptrons (Figure 1). It was used to classify continuous inputs by by applying a weighted sum of the inpus and subtracting a bias (constant value) and then the output is given as a number from two possible values (0 or 1).

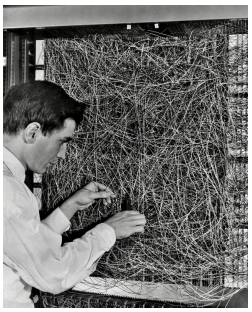


Fig. 1. Frank Rosenblatt with a Mark I Perceptron computer [4].

Shortly after the first perceptrons, the research on ANNs stopped until 1981, due to fear and unfulfilling results. After this peroid, many institutions and

researchrs began working on ANNs again, such as US-Japan Joint Conference on Cooperative/ Competitive Neural Networks [13], Schmidhuber & Hochreiter [17] and Yann LeCun [21].

2 Mathematical Background and Concept

ANNs learn how to accomplish a task by excuting two main steps: forward propagation and back propagation. Forward propagation is the process of predicting labels and computing how deviated these labels from the ones provided in the input data. On the other hand, back propagation tries to correct the predictions by minimizing the diffrence between the input labels and the predicted labels.

2.1 Forward Propagation

ANNs consist of layers where each layer has neurons which are connected via links (Figure 2.1). Each neuron gets inputs and produces an output. Each input is given a certain weight which makes that specific input has more or less effect on the output of the neuron.

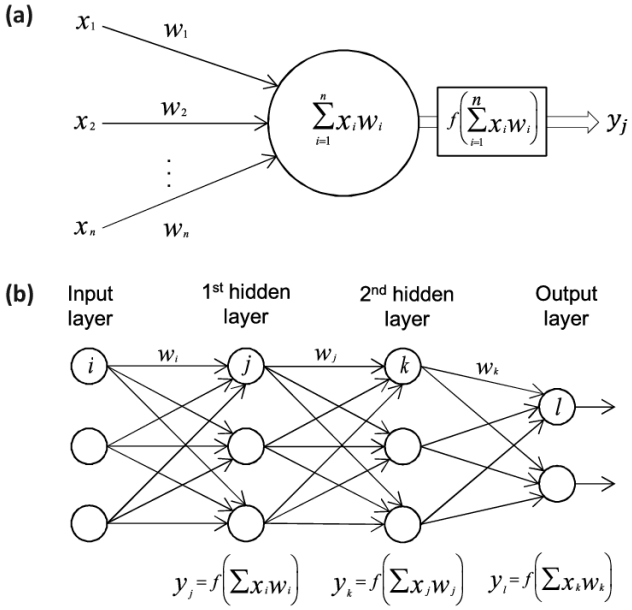


Fig. 2. (a) A single-layer ANN where the first layer has only one neuron. (b) A multi-layer ANN with 2 hidden layers [12].

Neurons calculate their output by multiplying their inputs by their weights and applying a bias to the multiplication. Eqaution 1 shows a linear mapping of

a single input.

$$z^i = W^T x^i + b \tag{1}$$

where z^i is the linear mapping of the ith example, $W \in R^{1 \times n_x}$ is the weight vector of the form $[w_1, w_2, ...w_{n_x}]$, $x^i \in R^{n_x \times 1}$ is the input vector of the form $[x_1, x_2, ...x_{n_x}]$ of the ith example and n_x is the number of features of the input vector (input size).

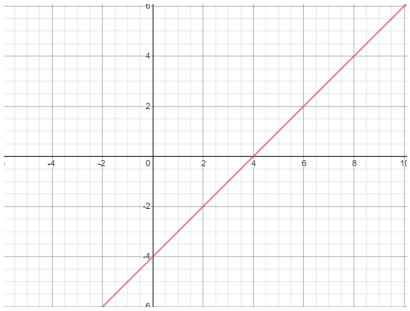


Fig. 3. A Figure that shows a sepration line with $n_x = 1$, $W = 1$, $b =$ -4 $(1x - 4)$.

This mapping is then forwaded to an activation function (Eqaution 2). Activation functions are used to limit the linear transformation output.

$$a^i = g(z^i) \tag{2}$$

where a^i is the output of the activation function (unit activation) on the linear mapping of the ith example.

To avoid manaully looping over each example and summing the output, all of the variables are used as vectors.

$$Z^{[l]} = W^{T[l]} A^{[l-1]} + b^{[l]} \tag{3}$$

$$A^{[l]} = g^{[l]}(Z^{[l]}) \tag{4}$$

where $Z^{[l]}$ is the linear mapping of the lth layer, $W^{T[l]} \in R^{n^{[l]} \times n^{[l-1]}}$ is the weight vector of the lth layer, $b^{[l]} \in R^{n^{[l]}}$ is the bias vector of the lth layer, $A^{[l-1]}$ is the activation unit of the previous layer ($A^{[0]} = X$), $A^{[l]}$ is the activation unit of the lth layer, $g()^{[l]}$ is the activation function of the lth layer, $n^{[l]}$ number of units (neurons) in the l layer and $n^{[l-1]}$ is the number of units in the previous layer $(l-1)$th layer.

The choice of the activation function vary depending on the given data (inputs). In general, there are three main categories of the activation function: binary, linear and non-linear. Binary or threshold functions output a binary value depending on the input. For example, the step function outputs +1 in case

z^i is greater than or eqaul to 0 and -1 otherwise (Eqaution 5). Binary functions can not deal with categorical data, therefore they are not widely used.

$$f(x) = \begin{cases} +1 & z^i \leq 0 \\ -1 & \text{otherwise} \end{cases} \tag{5}$$

Another type of activation functions is linear. Linear functions forward the input directly to the output without any transformation. This is useful in problems where the output is continuous. For example, predicting house prices.

Although all of the previous functions are useful for some sitautions, they fail to find a pattern if the data is non-linearly separable, since the function will only be able to draw a linear decesion boundary that can only devide the data into two groups. Therefore, other functions are used to find non-linear sepration between the data. (Figure 2.2) shows some widely used non-linear activation functions.

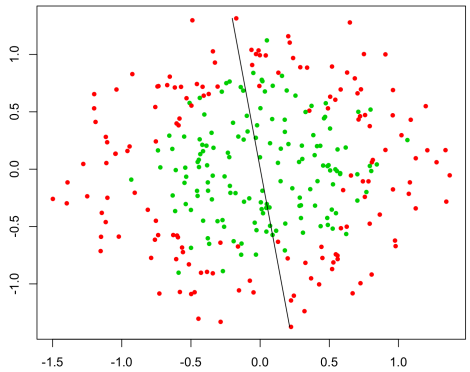


Fig. 4. Example of a linear activation function on non-linearly separable data [7].

After calculating the activation unit for the inputs, a loss function L is calculated for each prediction to measure how different the prediction is from the real data. the formula of the loss function varies based on the type of the labels. For example, if the labels are binary values (consisting of two classes), Log Loss could be used. This loss function pentalize the wrong prediction by applying the negative sign to the log, since the log of the values between 0.0 and 1.0 will output negative numbers (increse the loss for wrong predictions) (Eqaution 6).

$$L_i = -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i) \tag{6}$$

where L_i is the loss function of the ith example, y_i is the label of the ith example, $\hat{y}_i$ is the prediction of the ith example.

Alternatively, if the labels are continuous, Mean Absolute Error (MAE) or Mean Squared Error (MSE) are used (Eqaution 7 and 8).

$$MAE_i = |y_i - \hat{y}_i| \tag{7}$$

$$MSE_i = (y_i - \hat{y}_i)^2 \tag{8}$$

where MAE_i is the MAE of the ith example, MSE_i is the MSE of the ith example, y_i is the label of the ith example, $\hat{y}_i$ is the prediction of the ith example.

2.2 Back Propagation

Back propagation is the process of tunning the input weights of each neuron to correctly predict new labels. In other words, finding the weights that minimize the loss function. These weights are calculated through an algorithm called gradient descent [22]. Gradient descent is the process of iteratively updating the parameters of a function (input weights) in the direction of the steepest descent until a local minima is reached. To give a better idea, we can imagine the parameter space as a surface from where we follow the direction of the slope downhill until we reach a valley (Figure 2.2). Moreover, the slope is calculated by finding the first partial derivative of a function J in respect to every parameter of the function (Eqaution 10).

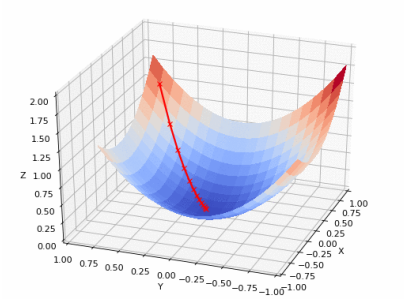


Fig. 5. Gradient descent visualization where the function is moving from a local maxima to a local minima [6].

$$w_i = w_i - \eta \frac{\partial J(W)}{\partial w_i} \tag{9}$$

where w_i is the ith parameter, η is the learning rate (a parameter used to control how large a step is).

in case of a multi-layer ANN with L layers, the gradient is instead calculated with the chain rule where the partial derivative is applied on each of the hidden layers till the first layer is reached.

$$w_i = w_i - \eta \left(\frac{\partial J(g_i(W))}{\partial g_i(W)} \frac{\partial g_i(W)}{\partial W}\right) \tag{10}$$

Since in the multi-layer ANNs the derivative is also applied to the activation functions, the choice of which activation function to be used is crucial. For example, using tanh or sigmoid functions may result in a slower learning process, as the first derivative of these functions is not so steep on extremely high or extremely low values (Figure 2.2).

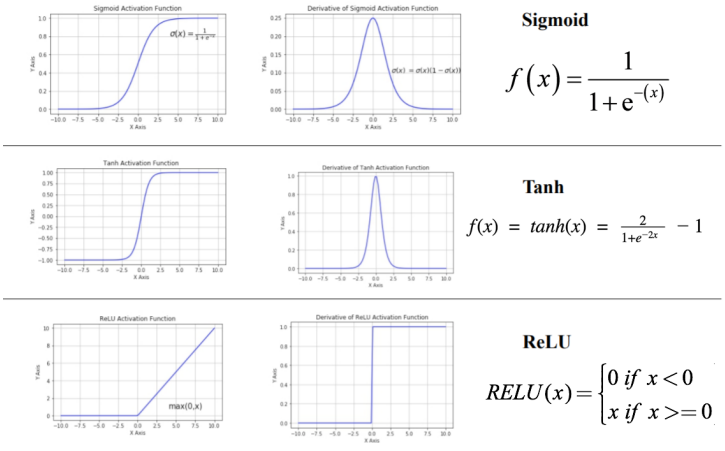


Fig. 6. A figure of some activation functions where the first one is sigmoid, the second is Tanh and the last is ReLU [5].

3 Applications and Architectures

In this section, we are going to discuss some of the architectures of ANNs and their applications.

3.1 Convolutional Neural Networks

While machines can calculate and do other tasks faster than humans, they can not sense or see objects. Therefore, many researchers tried to make machines mimic human vision through the field of classical Computer Vision [18]. However, this approach was hard to implement since an explicit algorithm has to be written. For example, if we want to do a task like face detection, we have to define what a face is and how far the eyes are away from each other and define many other measurements. This is not an easy task, because to generalize well, the algorithm has to cover every feature a human face can ever have and its measurements. Deep Learning and ANNs can solve this problem by just providing enough data (face images) and applying non-linearity to it. Images are presented as pixels and each pixel has a certain value. an ANN can take the pixels as parameters for the network and connect them to the neurons. To take

all the pixels from one image, the ANN has to provide many parameters. For example, a gray scale image with 28x28 size will be presented in 784 parameters, since the image has to be flatten before being processed by the ANN. Therefore, other approaches were used to process images with lower dimensionality. Convolutional Neural Networks (CNNs) use the "convolution" operator [21]. The main purpose of that operator is to extract only the important features from an image while still preserving the spatial relationship between its pixels. This is done using a sqaure of weights (filter) which gets multiplied by the image values (index-wise) and then producing an output called a feature map (Figure 3.1).

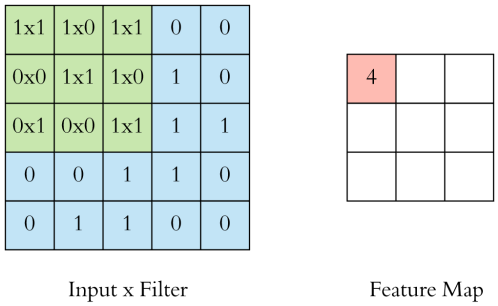


Fig. 7. A filter of size 3x3 applied on a 5x5 binary image [2].

A non-linear function is then applied to the feature map (Section 2.1). To further reduce dimensionality, CNNs uses the spatial pooling operator. This operator has diffrent types: Max, Average, Sum etc. When Max Pooling is used, the largest element is chosen from each window. Figure 3.1 shows a 2x2 Max Pooling operator.

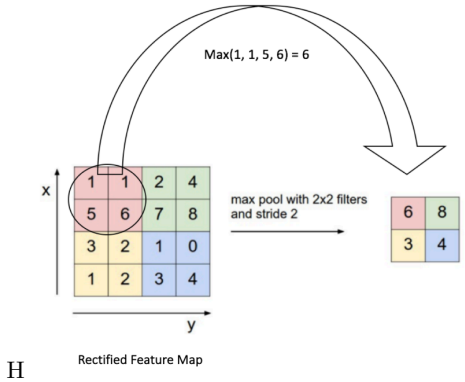


Fig. 8. a 2x2 Max Pooling operator on a 4x4 gray scale image [3].

The learning part in this ANN is the filter weights. However, we still need to define the height and width parameters of the filter on each layer, how many layers will the CNNs use and many other parameters. In general, the more layers the ANN has, the more it will be able to detect complex relations between pixels (shapes).

After finishing the previous operator, CNNs are usaully connected with a dense ANN to be able to predict/ classify a certain image. Figure 3.1 shows an example process of a CNN.

H

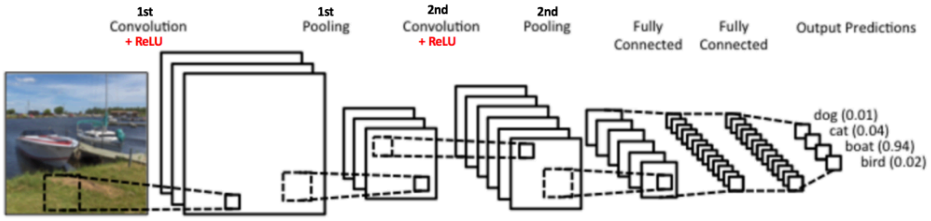
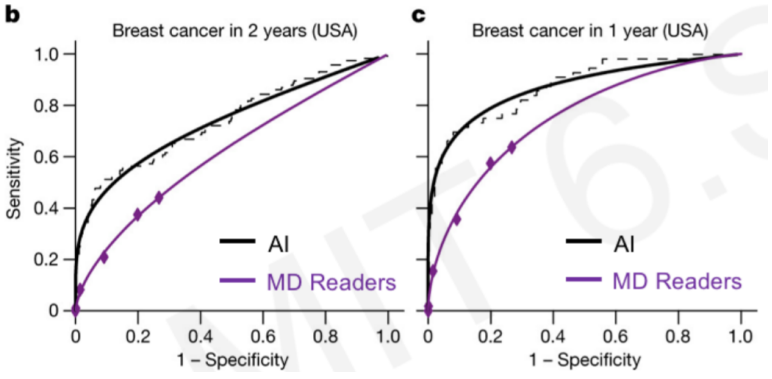


Fig. 9. A CNN applied on a boat image to detect if it falls in dog, cat, boat or bird categories [8].

CNNs have many applications, especially in healthcare. For instance, CNNs outperformed expert radiologists at detection breast cancer [23] (Figure 3.1).



H

Fig. 10. A figure showing that AI and CNNs can outperform RD Readers [10].

3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) is an ANN architecture which is used to predict a step in a sequence [17]. For example, consider the sentence "Mathe-

matics is my favorite subject, therefore I want to study _____ in college". A valid guess for the blank can be Mathematics or Engineering. Without seeing what is before the comma, we would not have gussed the word correctly. normal ANNs architectures (Section 2.1) propagate the inputs in one way and therefore the inputs are not aware of each other (parameter sharing) (not able to see what is before the comma because it process each word separately). RNNs address this problem by having an internal state (also called a hidden state) which gets updated every time when a new input is recieved (Eqaution 11).

$$h_t = f_W(h_{t-1}, x_t). \tag{11}$$

where h_t is the current state at time step t , f_W is a function based of the input weights W, the previous state and the current input, h_{t-1} is the previous state at time step $t-1$ and x_t is the current input at time step t.

In addition to updating the internal state, the network produces an input at each time step t by applying weights to the current state (Figure 3.2) (Eqaution 12).

$$\hat{y}_t = g(W_{hy}^T, h_t). \tag{12}$$

where $\hat{y}_t$ is the prediction at time step t, $g()$ is a non-linear function, W_{hy} is the weight vector between the state and the output, and h_t is the current state.

While the forward path is similar to the normal ANN forward propagate, the backward path is slightly diffrent. First of all, the loss function L must be calculated at each time step. Also, the back propagation has to go through all input weights in every time step. This particular process is called Back Propagation Through Time (BPTT) [24].

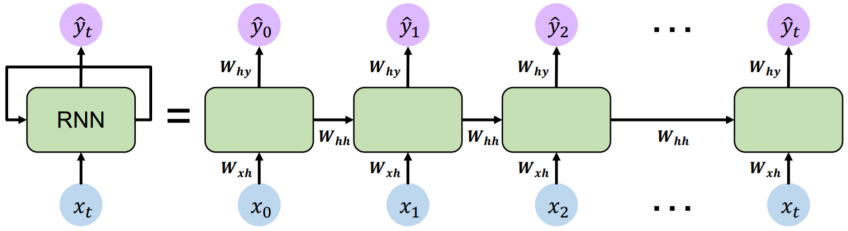


Fig. 11. An unfolded RNN where W_{hy} is the weight vector between the state and the output , W_{xh} is the weight vector between the current input x_t and the state h_t [9].

RNNs proved to be useful in many applications. For example, text prediction [20], Self-Driving Cars [15], and climate analysis and prediction [1].

4 Conclusion

In conclusion, ANNs have many applications and usages that could change our society. Unfortunately, we could not cover all of the architectures and applications of ANNs, since there are many of them. To name a few, Variational Autoencoders (VAE) which is used to generate data and reconstruct images [19] [25], and Deep Reinforcement Learning (DRL) which is seen in many applications, such as playing competitive computer games [14] and solving the Rubik cube [11].

References

1. a global map of wind, weather and ocean conditions. https://earth.nullschool.net/, accessed: 2020-04-25
2. Applied deep learning - part 4: Convolutional neural networks - arden dertat. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
3. Cs231n convolutional neural networks for visual recognition - stanford university. https://cs231n.github.io/convolutional-networks/
4. Frank rosenblatt with a mark i perceptron computer in 1960. https://www.reddit.com/r/EngineeringPorn/comments/e8a7x8/frank_rosenblatt_with_a_mark_i_perceptron/, accessed: 25.04.2020
5. German university in cairo (guc) - csen 1094 deep learning for self-driving cars. slim abdennadher. http://met.guc.edu.eg/Courses/CourseEdition.aspx?crsEdId=980
6. Gradient descent - ai wiki. https://docs.paperspace.com/machine-learning/wiki/gradient-descent
7. Interactive visualization of non-linear logistic regression decision boundaries with shiny. https://www.r-bloggers.com/interactive-visualization-of-non-linear-logistic-regression-decision-boundaries-with-shiny/
8. An intuitive explanation of convolutional neural networks - ujjwalkarn. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
9. Mit intro to deep learning (2019) - lecture 2. http://introtodeeplearning.com/2019/materials/2019_6S191_L2.pdf
10. Mit intro to deep learning (2020) - lecture 3. http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf
11. Openai solving the rubik's cube. https://openai.com/blog/solving-rubiks-cube, accessed: 25.04.2020
12. researchgate - sandra vieira. https://www.researchgate.net/figure/a-The-building-block-of-deep-neural-networks-artificial-neuron-or-node-Each-input-x_fig1_312205163
13. Amari, S.i., Arbib, M.A.: Competition and cooperation in neural nets. Springer Lecture Notes in Biomathematics **45** (1982)
14. Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al.: Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 (2019)
15. Gu, Z., Li, Z., Di, X., Shi, R.: An lstm-based autonomous driving model using a waymo open dataset. Applied Sciences **10**(6), 2046 (2020)

16. Hassoun, M.H., et al.: Fundamentals of artificial neural networks. MIT press (1995)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
18. Hoffmann, C.M.: Computer vision, descriptive geometry, and classical mechanics. In: Computer graphics and mathematics, pp. 229–243. Springer (1992)
19. Hou, X., Shen, L., Sun, K., Qiu, G.: Deep feature consistent variational autoencoder. In: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 1133–1141. IEEE (2017)
20. Jagannatha, A.N., Yu, H.: Structured prediction models for rnn based sequence labeling in clinical text. In: Proceedings of the conference on empirical methods in natural language processing. conference on empirical methods in natural language processing. vol. 2016, p. 856. NIH Public Access (2016)
21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
22. Lemaréchal, C.: Cauchy and the gradient method. Doc Math Extra **251**, 254 (2012)
23. McKinney, S.M., Sieniek, M., Godbole, V., Godwin, J., Antropova, N., Ashrafian, H., Back, T., Chesus, M., Corrado, G.C., Darzi, A., et al.: International evaluation of an ai system for breast cancer screening. Nature **577**(7788), 89–94 (2020)
24. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE **78**(10), 1550–1560 (1990)
25. Yan, X., Yang, J., Sohn, K., Lee, H.: Attribute2image: Conditional image generation from visual attributes. In: European Conference on Computer Vision. pp. 776–791. Springer (2016)