Q/A @ CCN

2. February 2016

1 Ch 3 (Perceptron)

1. Rojas talks about duality between input and weight spaces (section 3.3.2). A quick internet search on the topic of "duality" points to fairly complicated looking mathematical stuff concerning vector spaces, linear functionals and topology. (https://en.wikipedia.org/wiki/Dual_space)

   Is this the same notion of duality Rojas is referring to? A related concept? Something totally different?

   A: It is the same duality. If we forget about the bias term for a moment and assume that the input is 2-dimensional, then before the non-linearity is applied, the perceptron performs the calculation

   $$[x_1, x_2] \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

   This means that the weight-vector $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ determines a linear map from $\mathbb{R}^2$ to $\mathbb{R}$. By definition, the sepace of all such linear maps is the dual space of $\mathbb{R}^2$. This works also the other way around: The above dot-product can be thought of defining either of the two linear functions $\mathbf{x} \mapsto \mathbf{x} \cdot \mathbf{w}$ or $\mathbf{w} \mapsto \mathbf{x} \cdot \mathbf{w}$.

   With the bias, one can think that the input is 3 dimensional but one coordinate is fixed to 1. The space of weights is 3-dimensional and defines the dual to $\mathbb{R}^3$ in the same way as above. Technically now the weights define a dual space to the (affine) 2-dimensional subspace of $\mathbb{R}^3$ and not to the whole $\mathbb{R}^3$. The only essential difference is that for a one dimensional subspace of the weight-space, namely those with $w_2 = w_3 = 0$ we get a trivial decomposition of the input space.

2. What are McCulloch-Pitts -units and how do they differ from perceptrons?

   A: A McCulloch-Pitts unit is a perceptron with weights equal to 1 and $-\infty$, i.e. the positive inputs are all summed up as 1's and if any inhibition is active, it inhibits the whole neuron.

3. In the chapter 3 and section 3.1.1, in the figure 3.1, I would like to know the reason why the connection to the second layer of computing elements and from the second to the third are stochastically selected?

   A: in order to make the model biologically plausible: randomness increases variability of different connection patterns which may enhance learning (that was the intuition).

4. In the page of 59, in the equation of calculating S, I would like to know how to define the value of the threshold value θ.

   A: The threshold can be anything and the argument still works. It is independent of the threshold.

5. How closely does a perceptron resemble an actual neuron? For instance, would the XOR problem apply also to a real neuron?

   A: Good question. Perceptron is a very simplistic version of a real neuron. At this moment I don't know if integration of inputs can be non-monotonic in real neurons (I only know it can be non-linear). But if it is monotonic with respect to a linear combination of the inputs, then it still cannot compute XOR. This means that even though the integration of information by a real neuron is much more complicated than linear, it might not be computationally more powerful as a unit.

6. How similar or how many similarities does the retina of the perceptron have with the biological retina?

7. Is there a technical reason for only using a single element instead of two? Or is the example of the XOR problem just there to show what would happen if we could use only one? With how many elements is it possible to compute XOR? **A:** Two perceptrons in a single layer would output two numbers. In order to integrate them into one, you need a third one; otherwise, how do you interpret the answer? If they are "one after another" in a multilayer fashion (input is first sent to perceptron 1 and then the output is sent to perceptron 2), then whatever you can do, you can already do with one perceptron (why?).

   BUT, you can implement XOR with two perceptrons if you send the input to both of them and also connect them to each other (how exactly?)

8. How exactly does the proof for the "no diameter limited perceptron can decide whether a geometric figure is connected or not"?

2 Ch 4 (Perceptron Learning)

1. Neural networks: how are cases handled if more than one output is needed? Multiple networks? **A:** There can be more than one neuron at the output.

2. Explain the worst case complexity problem of the greedy, local learning algorithm? (How) can you avoid or mitigate it?

3. What is "linear separability in the conventional sense"? In Rojas section 4.1.3, the author deals with absolute linear separability, but I don't see how this is different from just 'linear separability'. Where's the catch?

   A: It is the difference between "$\leq$" and "$<$". I agree that he spends too much ink on this issue.

4. Apart from simple perceptrons, what other computationally inexpensive predicates are generally useful, or, alternatively, perform exceptionally well in some class of applications?

5. What is the state of art in hardware implementations of networks and learning algorithms?

6. Is there a mathematical approach on how to track local minima in the error function and/or avoid converging to them?

   A: This is a difficult problem and there is no general solution. Many mathematical techniques are employed as much as possible. One way is to choose the activation function and the error function in such a way that the number of local minima becomes smaller. It depends on the network architecture and the task at hand.

7. How is training done in case of missing target information, i.e. unsupervised learning?

   A: For example through evolutionary strategies, i.e. roughly speaking by randomly assigning weights and then modifying them little by little according to some probability distributions and looking for best fitting answers. Another example is Hebbian learning, see Rojas Chapters 12, 13.

8. Perceptron learning starts with random weights. How big can these random weights be? I.e. is there an interval in which these values should be picked (e.g. decimals between 0 and 1)?

   A: It depends on the data, the learning constant γ and the error-function. A good thumb-rule is that it shouldn't be too large, because the minimum of the error-function is more likely to be found close to the origin than far from the origin. If the data for example is normalized to get values between 0 and 1, then a good interval is [0, 0.1].

9. About the vector math in perceptron learning algorithm, I'm unclear why $w \cdot x > 0$ means the angle between x and w is less than 90 degrees, and then how and why it is rotated away from x by subtracting x from w. (p 86.) **A:** Think about this characterization of the dot-product: $\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}||\mathbf{y}| \cos \alpha$ where α is the angle between $\mathbf{x}$ and $\mathbf{y}$.

10. Can the basic perceptron learning algorithm (Rojas algorithm 4.2.1) be used to train larger singlelayer and multilayer perceptrons?

11. Section 4.2 discusses the "pocket algorithm", a modification of the linear perceptron algorithm that terminates even when data is non linearly separable. What is the additional computing cost of such modification?

    A: At least memory.

3 Ch 7 (Backpropagation)

1. How to choose number of hidden layers. Is it bounded by number of input features?

2. On what basis we should choose the non linear function . Ex : logit / tanh etc.

   A: Different activation functions contribute differently into the convergece of learning algorithms. It is a matter of analysis in each situation what is the best choice.

3. In your opinion, for understanding deep neural network, should we focus more on researches based on physics theories like spin glass theory and re-normalization theory[2], or should we focus more on approximation like what people do in aerodynamics ? (it seems this topic has became one of fundamental question for doing contemporary deep learning/neural network research besides unsupervised deep learning and deep reinforcement learning.)

   A: Interesting question! I don't know :(

4. What are the possible target spaces (manifolds) that a neural network can find?

5. Are there target spaces that they cannot find?

6. Can you list all learning algorithms of neural network (as many as you know)?

   A: Backpropagation, Hebbian, perceptron, covariance matrix adaptation (evolution), simulated annealing, expectation-maximization, non-parametric methods and particle swarm optimization, competitive learning...

7. What is the difference between supervised and unsupervised learning algorithms?

   A: In supervised the intended output of the training data is known while in unsupervised it is unnkown. For example backpropagation requires the knowledge of the error function which can only be computed if the intended answer is known.

8. Is it possible to compute the gradient for a (small) neural network and find weights for a local minima analytically, instead of using the back-propagation algorithm and find the local minima one step at a time?

   A: http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

9. In a general feed forward neural network, is it typically so that every node in a layer is connected to all nodes in the next layer ?

   A: Not necessarily, see convolutional neural networks for a counterexample.

10. What are typical activation functions (in addition to sigmoid) ?

    A: step-function, sign-function, rectifier $\max(0, x)$, also see NIS (3.6), (3.7) (pp. 60-62)

11. What does a 'free parameter' of a network mean?

    A: Parameters that can be tuned such as weights.

12. I don't understand the concept and mathematical notation of the gradient descent (section 7.2, page 157). Why is derivative needed for the gradient descent (∂ seems to mean something about derivative)?

    A: Gradient is the same as multidimensional derivative. We want to go along the direction of least growth to find local minima.

13. Does deep learning make use of the standard back propagation algorithm, or are the methods different? If it does use back propagation, what distinguishes deep learning from a multilayer perceptron? Is it only the number of layers, or is there more to it?

14. To avoid overfitting, people usually add an extra term to the cost function, which is proportional to the sum of the square of all the weights (aka L2 regularization). The intuition of this term is to make the network to learn small weights. Why learning small weights can avoid overfitting?

    A: It reduces variability. Similarly as why in polynomial approximation the coefficients of large exponents should be kept small.

15. How to calculate the derivative of the function of a node in case, where the function is not differentiable at the input point?

    A: There is no way of calculating it by definition. One might look at one-sided derivative or consider the option that one is actually at a minimum if the function is not differentiable there.

16. (Sigmoid activation functions for backpropagation.) Rojas mentions that the reciprocal 1/c is called the 'temperature parameter' (where c is a constant parameter in a sigmoid function, see section 7.1.1. / equation on page 152). Any special reason for why it's called that?

    A: Probably comes from Fermi-Dirac distribution

17. For which problems is a feedforward ANN useful?

18. Can backpropagation be used for unsupervised learning?

    A: Hardly. But of course you can find anything on the Internet: http://arxiv.org/abs/1312.5394 Note that the authors of this are still "filling in the values" before they can backpropagate. As far as I understand.

19. What is an autoencoder and what are the main differences between it and a multi-layer perceptron?

    A: Stay tuned.

20. It's not clear for me how weights are updated by the backpropagation algorithm. In particular, where does this expression $\Delta w_{ij} = -\gamma o_i \delta_j$ (page 164 Rojas) come from and what does γ denote?

    A: γ is the learning constant. The smallet it is, the smaller step is taken towards the direction of greatest descent. It is more "careful": more precise, but is at risk of missing the big picture.

21. While solving the backpropagation algorithm in the second phase, at what point do we need to update the weights? Do we use updated weights to calculate things for $(n-1)$:th layer?

    A: No, at least in the standard b.p. algorithm described in Rojas all weights are updates simultaneously. The delta is first calculated for each weight in the network and then applied to all of them only after the entire calculation. In some cases, the layers may be trained separately, but that is more complicated.

22. in back-propagation section I don't understand why output values other than 0 or 1 would cause local minimas/maximas.

23. -The second case exposed in section 7.2.2 shows a network for the computation of the addition of two primitive functions. Would the subtraction of two primitive functions work in a similar way? And the product?

24. Does the backpropagation algorithm require that the error function is convex?

25. How well does gradient descent perform with sigmoid functions in practice? Does the algorithm get stuck in local minima most of the time? Are there any ways to improve the chance of finding the global minimum or a smaller local minimum, besides trying different random starting weights?

26. Back propagation: how do we construct the neural network with its hidden layer? Do we (a) just randomly play with different depths and number of intermediate nodes and pick the best or (b) can we actually construct a "better" base neural network based on specific topic?

27. Back propagation: (if I understood this correctly) the output of a path is done level-wise, For all node we sum for all incoming edges its source multiplied with the edge weight. Or in other words: we only use addition and multiplication. Can this then effectively model more complicated functions like a logarithm or a sinus?

    A: That is why we use the sigmoid function! So that there is not only addition and multiplication!