DEEP LEARNING

DEVANSH TRIPATHI

Abstract. We shall learn Deep Learning.

Part 1.
Shallow Neural Network

1. Overflow and Underflow

1.1. Underflow.

- It is the one form of rounding error.
- It occurs when numbers near zero are rounded to zero.
- Many function can blow up at 0 but they maybe defined at some small positive number.

1.2. Overflow.

- It occure when very large number are rounded off as ∞ or $-\infty$ and for further arithmatic will treat them as NaN.

1.2.1. *Avoiding underflow and overflow.* We should use some trick that stabilize the functions that are present in our algorithm. For example- if we have $softmax$ function,

$$\text{softmax}(\mathbf{x})_\mathbf{i} = \frac{\exp(\mathbf{x_i})}{\sum_{\mathbf{j=1}}^{\mathbf{n}} \exp(\mathbf{x_j})}$$

then we can use shifting to avoid the any number being too small or too large. With simple manipulation, it can shown that $\text{softmax}(\mathbf{x}) = \text{softmax}(\mathbf{x} - \mathbf{c})$ where c is $\max_i x_i$.

Problem will arise when all the entries in the $\mathbf{x}$ are either too small or too large. But then shifting will make sure at least one element becomes 0 while subtracting the $\max_i x_i$ namely the maximum term

itself. Then $\exp(0) = 1$ will avoid the underflow and subtractng $\max_i x_i$ will make sure all the terms become small to avoid overflow.

2. Gradient Descent

For $f(x - \epsilon \operatorname{sign}(f'(x)))$ is less than $f(x)$ for small enough ϵ. We can thus reduce $f(x)$ by moving x in small steps in the direction opposite to the derivative. This technique is called *gradient descent*. (Cauchy, 1847).

For $f : \mathbb{R}^n \to \mathbb{R}$, *directional derivative* in the direction $\mathbf{u}$ (a unit vector) is the slope of f in the direction u.

To minimize f, we need to find the direction in which f decreases fastest. So we want the change to be maximum. Since final value will be less than the initial, the change is negative.

$$\mathbf{x}' - \mathbf{x} = -\epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$$

where $\mathbf{x}'$ is the final value and $\mathbf{x}$ is the initial value. In order to maximize RHS, we need to minimize $\nabla_{\mathbf{x}} f(\mathbf{x})$ since ϵ is constant (learning rate).

$$\begin{aligned} &= \min_{u, u^\top u = 1} \nabla_x f(x) \cdot u \quad \text{(directional derivative)} \\ &= \min_{u, u^\top u = 1} \|u\|_2 \|\nabla_x f(x)\|_2 \cos\theta \end{aligned}$$

where θ is the angle between u and the gradient. For minimizing this, we need minimal value of $\cos\theta$ which is -1. Hence, the direction just opposite of gradient is the steepest descent direction. This method is called *method of steepest descent* or *gradient descent*.

NOTE: Gradient descent is limited to optimization in continuous spaces. But general concept of making small moves (that are approximately the best small move) towards better confiurations can be generalized towards discrete spaces. TODO: Prove/how this can be generalized.

NOTE: Ascending an objective function of discrete parameters is called *hill climbing*. (Russel and Norvig, 2003)

2.1. **Line search.** TODO

3. Shallow Neural Networks

Consider the case with D hidden units where the d th hidden unit is:

$$h_d = a[\theta_{d0} + \theta_{d1}x],$$

where $a[\cdot]$ is the *activation function* and these are combined linearly to create the output:

$$y = \phi_0 + \sum_{d=1}^{D} \phi_d h_d.$$

The hidden layer has just x because there is only one input. In case of multivariate input, hidden layer will be function of each of the components of the input.

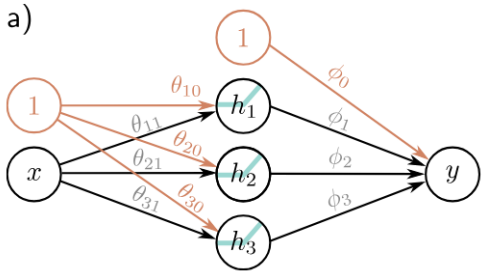

FIGURE 1. Neural nets

> **Network Capacity:** The number of hidden units in shallow network is a measure of network capacity.

With ReLU activation functions, the output of a network with D hidden units has atmost D joints and it is a piecewise linear function with $D + 1$ linear regions.

> **Universal Approximation Theorem:** This thoerem states that there exists a network with one hidden layer containing finite number of hidden units that can approximate any specified continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy.

3.1. **Multivariate inputs and outputs.**

3.1.1. *Multivariate Outputs.* To extend the network to multivariate outputs $\mathbf{y}$, we use different linear function of the hidden layer for each output. For example: A network with a scalar input x and 4 hidden layers and a $2D$ output $\mathbf{y} = [y_1, y_2]^\top$ would defined as

$$\begin{aligned} h_1 &= a[\theta_{10} + \theta_{11}x] \\ h_2 &= a[\theta_{20} + \theta_{21}x] \\ h_3 &= a[\theta_{30} + \theta_{31}x] \\ h_4 &= a[\theta_{40} + \theta_{41}x] \end{aligned}$$

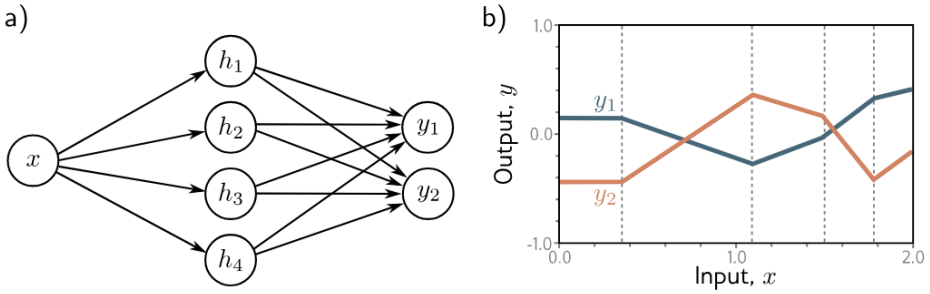

Figure 3.6 Network with one input, four hidden units, and two outputs. a) Visualization of network structure. b) This network produces two piecewise linear functions, $y_1[x]$ and $y_2[x]$. The four "joints" of these functions (at vertical dotted lines) are constrained to be in the same places since they share the same hidden units, but the slopes and overall height may differ.

FIGURE 2. Neural nets

The outputs may look like

$$\begin{aligned} y_1 &= \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4 \\ y_2 &= \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4 \end{aligned}$$

Since we are taking the combination of linear functions, the slope of the linear regions can change according to the value ϕ's **but the location of joints will be same in both outputs**. ϕ_0 can cause translation in y axis.

Exercise 1. *Why the location of joints will not change in the graph of both the outputs which are the linear combination of same hidden layers but for different values of scalar?*

Solution. They are linear combination of same hidden layers (linear functions) with different scalars. When we apply ReLU on them the point of joint gets fixed. Then we create different functions for each

output by multiplying the output after ReLU with ϕ's which results in change of slope of the line which has non-zero slope. And no change in slope for 0 slope. (Basically, roots does not change when we multiply slope and intercept by the same number.)

3.1.2. *Multivariate Inputs.* In the case of multivariate input $\mathbf{x} = [x_1, x_2]^\top$, the hidden will be defined as

$$h_d = a[\theta_{d0} + \theta_{d1}x_1 + \theta_{d2}x_2]$$

Then these hidden layers are combined in the usual way of linear combination.

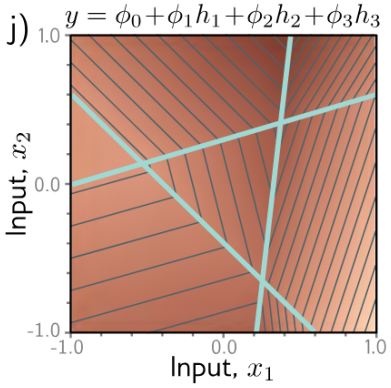

FIGURE 3. Convex Region

If we had same number of hidden units as input dimensions D_i then we can align each hyperplane with the coordinate axis but in shallow neural networks we usually have more hidden units as compared to input dimensions (hence more than 2^{D_i}).

3.2. **Shallow Neural Network: General case.** Shallow Neural network $\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ maps the multidimensional input $\mathbf{x} \in \mathbb{R}^{D_i}$ to a multidimensional output $\mathbf{y} \in \mathbb{R}^{D_o}$ using $\mathbf{h} \in \mathbb{R}^D$ hidden units. Each hidden unit is computed as:

$$h_d = a\left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di}x_i\right]$$

The final output we get is after linearly combining them:

$$y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd}h_d$$

The model has parameters $\boldsymbol{\phi} = \{\theta_{\bullet,\bullet}, \phi_{\bullet,\bullet}\}$. The activation functions permits the model to describe non-linear relations as they itself are non-linear.

With the ReLU activations, the network divides the input space into **convex polytopes** (polygon in higher dimensions) defined by the intersection of hyperplanes computed by joints in the ReLU functions. Each convex polytope contains a different linear function (whichever linear functions are active (nonzero slope) in the region, will contribute to overall polytope). The overall polytope's edge's slope is the summation of the slope of linear functions in the area.

In case of multiple outputs, the polytopes are the same for each output, but the linear function they contain may differ.

4. Terminology

Definition 1 (Pre-activations)**.** When we pass data through the network, the values of the inputs to the hidden layer (i.e. before the ReLU functions are applied) are termed as *pre-activations*.

The value at the hidden layer (i.e. after the ReLU functions) are termed as *activations*.

Definition 2 (Feed forward NN)**.** Neural networks in which the connections form an acyclic graph (i.e. graph with no loops, as e.g. in previous figures) are referred to as *feed forward networks*.

Number of linear regions: Consider a shallow network of input dimension $D_i \geq 2$ and D hidden units. The **maximum** number of linear regions created by D hyperplanes in the D_i dimension input space where D_i is less than D-dimension input space, is $\sum_{\mathbf{j=0}}^{\mathbf{D_i}} \binom{\mathbf{D}}{\mathbf{j}} \leq \mathbf{2^D}$. Hence, as a rule of thumb, **shallow neural networks always have a large number of hidden units D than the input space dimension D_i** and create between 2^{D_i} and 2^D linear regions.

Definition 3 (Width, depth and Hyperparameters)**.**

- Number of hidden units in each layer is referred to as the *width* of the network.
- Number of hidden layers as the *depth*.
- The total number of hidden units is a measure of the network's *capcaity*.

- The number of layers and number of hidden units in each layer is called *hyperparameters*.

Part 2. Deep Neural Network

Theoretically, shallow neural network can describe any arbitrary complex functions with increasing number of hidden units but for some functions the number of hidden units needed can be impractically large.

Deep Neural network can describe those functions with less number of hidden units and produce more linear regions for the given number of parameters.

5. Composing neural network

We can compose two shallow neural network such that output of the first network is the input of the other network.

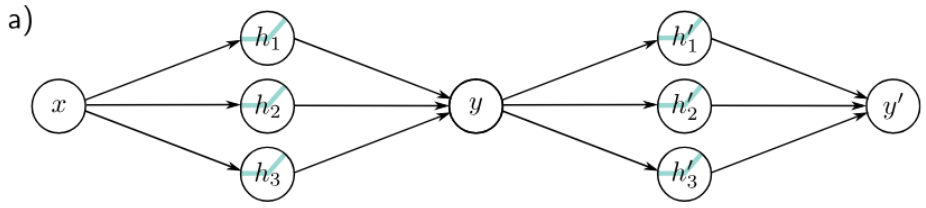

Figure 4. Composing neural nets

$$\begin{aligned}
h'_1 &= \mathrm{a}[\theta'_{10} + \theta'_{11}y] = \mathrm{a}[\theta'_{10} + \theta'_{11}\phi_0 + \theta'_{11}\phi_1 h_1 + \theta'_{11}\phi_2 h_2 + \theta'_{11}\phi_3 h_3] \\
h'_2 &= \mathrm{a}[\theta'_{20} + \theta'_{21}y] = \mathrm{a}[\theta'_{20} + \theta'_{21}\phi_0 + \theta'_{21}\phi_1 h_1 + \theta'_{21}\phi_2 h_2 + \theta'_{21}\phi_3 h_3] \\
h'_3 &= \mathrm{a}[\theta'_{30} + \theta'_{31}y] = \mathrm{a}[\theta'_{30} + \theta'_{31}\phi_0 + \theta'_{31}\phi_1 h_1 + \theta'_{31}\phi_2 h_2 + \theta'_{31}\phi_3 h_3],
\end{aligned}$$

which we can rewrite as:

$$\begin{aligned}
h'_1 &= \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3] \\
h'_2 &= \mathrm{a}[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3] \\
h'_3 &= \mathrm{a}[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3],
\end{aligned}$$

Figure 5

where $y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$. Here, if we compare above set of equation, we get the values of $\psi_{10} = \theta'_{10} + \theta'_{11}\phi_0$, $\psi_{11} = \theta'_{11}\phi_1$,

$\psi_{12} = \theta'_{11}\phi_2$, $\psi_{13} = \theta'_{11}\phi_3$ and similarly for other parameters. The above values of ψ_{ij} are the elements of first row of the outer product matrix of $u = [\theta'_{11}, \theta'_{21}, \theta'_{31}]^\top, v = [\phi_1, \phi_2, \phi_3]^\top$.

$$u \otimes v = \begin{bmatrix} \theta'_{11}\phi_1 & \theta'_{11}\phi_2 & \theta'_{11}\phi_3 \\ \theta'_{21}\phi_1 & \theta'_{21}\phi_2 & \theta'_{21}\phi_3 \\ \theta'_{31}\phi_1 & \theta'_{31}\phi_2 & \theta'_{31}\phi_3 \end{bmatrix}$$

Hence, when we compose two neural networks the values of the parameters ψ_{ij} are restricted to this outer product, while when we create a neural network with two hidden layers, we can have more values of the parameters ψ_{ij} hence it can represent a broader family of functions.