Lecture 11:Feature learning, neural networks and back propagation

wbg231

December 2022

1 introduction

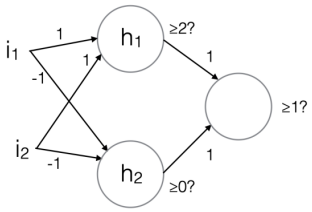
Feature engineering

- many problems are non-linear
- we can express certain non linear problems as a linear combination of a feature map
$$f(x) = w^t \phi(x)$$
- if we are explicitly specifying a feature map then the task comes down to decomposing our problem into sub-problems that can be combined in a linear way

perceptron's as logic gates

- perceptrons learn a hyperplane that separates linearly sparable data
- this works to repent simple and or logic gates.
- note that this will not work for tasks that are non linearly sparable
- for example if we have $x \in \mathbb{R}^2$ and we want to know if $x_1 = x_2$ then $w^t x = w_1(x) + w_2(x_2) = w_1 + w_2 > 0$ if they are both 1, and 0 if they are both 0 then $w_1 + w_1 = 0$ so in other words we can not make a line (in 2 d space) separating the classes

- Fire when the two pixels have the same value ($i_1 = i_2$)



			Hidden layer input		Hidden layer output		
	i_1	i_2	h_1	h_2	h_1	h_2	o
x_1	0	0	0	0	0	1	1
o_1	0	1	1	-1	0	0	0
o_2	1	0	1	-1	0	0	0
x_2	1	1	2	-2	1	0	1

(for x_1 and x_2 the correct output is 1;
for o_1 and o_2 the correct output is 0)

- so if we add a a second perceptron we can separate these classes
- how you can think of this is that the first perceptron is in effect a feature map, which sends $x \to \phi(x)$ where $\phi(x)$ is a space where the classes can be separated

neural networks

- the key idea is to learn the intermediate features as opposed to explicitly building them
- feature engineering medially specifying a feature map ϕ based on domain knowledge then learn weights w

$$f(x) = w^t \phi(x)$$

- feature learning learn both the features (K hidden units) and the weights

$$h(x) = [h_1(x) \cdots h_k(x)]$$

$$f(x) = w^t h(x)$$

activation function

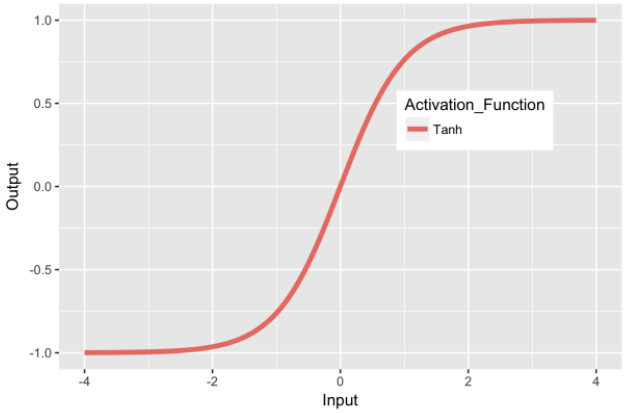
- think of hidden layers as feature representations, we like to think of these feature representations as either absent (ie zero) or if they exist passed a certain threshold taking some value
- so the activation function encodes this it applies non-linearity on the inputs and fires after some threshold

$$h_i(x) = \sigma(v_i^t x)$$

- so we can write a two layer networks as

$$f(x) = \sum_{k=1}^{k} w_k h_k(x) = \sum_{k=1}^{k} w_k \sigma(v_k^t x)$$

- the hyperbolic tangent function is a common activation function



note that this function basically gives activates when the magintude of it's input are away from zero

- relu activation function

$$\sigma(x) = max(0, x)$$

does not fire until x is greater than zero and then fires linearly after that

universal approximation theorem

- universal approximation theorem a neural net with one possibly huge hidden layer $\hat{F}(x)$ can approximate any continuous function a closed and bounded subset under mild $\forall \epsilon > 0$ there exists an integer N such that

$$\hat{F}(x) = \sum_{i=1}^{n} w_i \sigma(v_i^t x + b)$$

satisfies

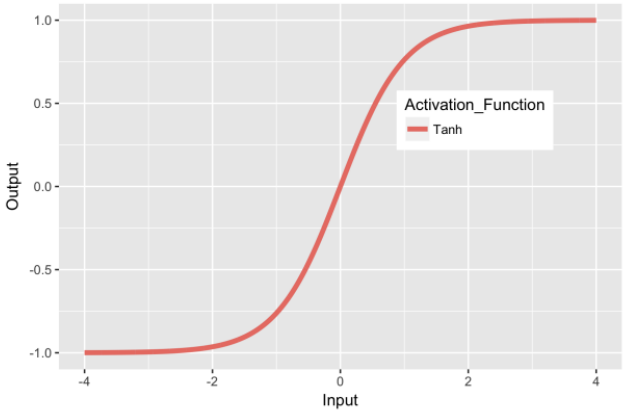
$$||\hat{F}(x) - F(x)|| < \epsilon$$

- the take away is as long as the function is continuous (ie it has a non-infinite rate) on some subset we can in theory approximate it using neural networks

- note that in this set up the number of hidden units is exponential in d

deep neural networks

- a deep neural network is one that can be both wide (ie have hidden layers) with many hidden units, as well as deep ie have hidden many hidden layers



-
- Multi layer perceptron definition
- input space $x \in \mathbb{R}^d$ action space $A = \mathbb{R}^k$ (for a k class classification task)
- let $\sigma : \mathbb{R} \to \mathbb{R}$ be an activation function
- suppose we have L hidden layers each having M hidden units
- the first hidden layer is given by

$$h^1(x) = \sigma(W^1 x + \beta)$$

  where $W^1 \in \mathbb{R}^{m \times d}$ $b \in \mathbb{R}^m$ and σ is applied to each entry of it's argument

- each of the following hidden layers is passed $o \in \mathbb{R}^m$ which is the output and produces

$$h^j(o^{j-1}) = \sigma(W^j o^{j-1} + b^j)$$

  where $W^j \in \mathbb{R}^{m \times m}$

- and the last layer (output layer is an affine function) that is with no activation function

$$a(o^l) = W^{L+1} O^l + b^{l+1}$$

  where $W^{l+1} \in \mathbb{R}^{k \times m}$ and $b^{l+1} \in \mathbb{R}^k$

- the last layer gives us our scores

- then we try to maximize a non-linear score function that maps our scores to probabilities like the soft max function

$$argmax_{f_1 \cdots f_k} \sum_{i=1}^{n} log(softmax(f_1(x) \cdots f_k(y))_{y_i})$$

- so we are in effect maximizing the log likelihood of representations of the training data
- so keep in mind the input layer has no learnable parameters it is just the inputs
- the hidden layer is affine plus some non-linear activation function
- the output layer is an affine function that is then passed to some scoring function

fitting parameters for MLP

- suppose $X = \mathbb{R}$ that is we have one dimensional input data
- our action adn output space are real numbers
- out hypothesis space is a mlp with 3 hidden node layes

$$f(x) = w_0 + w_1 h_1(x) + w_2 h_2(x) + w_3 g_3(x)$$

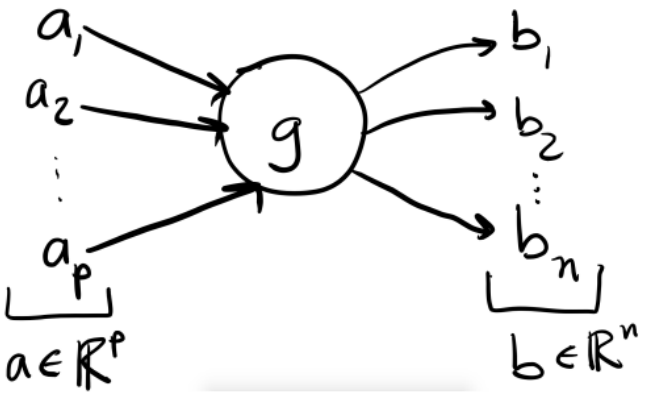
  where $h_i(x) = \sigma(v_i x + b_i)$
- we need to fit $b_1, b_2, b_3, v_1, v_2, v_3, w_0, w_1, w_2, w_3 \in \mathbb{R}$
- think of all parameters tighter as $\theta \in \mathbb{R}^{10}$ our goal is to find

$$\hat{\theta} = argmin_{\theta \in \mathbb{R}^{10}} \frac{1}{n} \sum_{i=1}^{n} (f(x_i, \theta) - y_i)^2$$

- we can do gradient descent and extend it to back propagation which is a systematic and efficient way to get gradient

computation graph

- we can represent each component of the network as a node tat takes a set of inputs and produces outputs



- suppose we have this computation graph
- let $g(x) = Mx + c$ for $M \in \mathbb{R}^{n \times p}$ and $c \in \mathbb{R}$
- $letb = g(a) = Ma + c$
- what is bi?

$$b_i = \sum_{k=1}^{p} M_{i,k} a_k + c_i$$

- note that $\frac{\partial b_i}{\partial a_j} = M_{i,j}$

least squares example

- hypothesis space

$$\{f(x) = w^t x + b | w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

  so affine functions
- dataset $((x_1, y_1), \cdots, (x_n, y_n)) \in \mathbb{R}^d \times \mathbb{R}$
- our loss function in this contest is

$$\ell_i(w, b) = [(w^t x_i + b) - y_i]^2$$

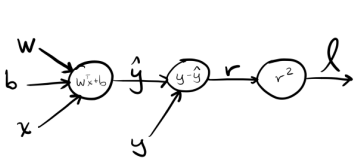
- in stochastic gradient descent we take steps

$$w_j \leftarrow w_j - \eta \frac{\partial \ell_i(w, b)}{\partial w_j} \forall j \in [1, d]$$

  and

$$b \leftarrow b - \eta \frac{\partial \ell_i(w, b)}{\partial b}$$

- for training point $\ell(w, b) = (w^t x + b - y)^2 = (r)^2$



$$\frac{\partial \ell}{\partial r} = 2r$$
$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial r}$$
$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial \hat{y}}$$
$$\frac{\partial \ell}{\partial w_j} = \frac{\partial \ell}{\partial \hat{y}}$$

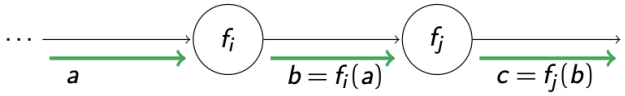
- then we can find the partial darivatives for this question as

backpropigation example

- to learn we need to run gradient descent to find the parameters that minimize our objective
- backpropigation we compute the gradient wrt to each trainable parameter
- this has to steps
  1. compute intermediate function value ie output of each node
  2. compute the partial derivative of j with respect to all intermediate values and model parameters
- we can optimize this with path sharing each node cashes it's intermediate results and we dont need to compute them multiple times (this is dynamic programming :)

forward pass

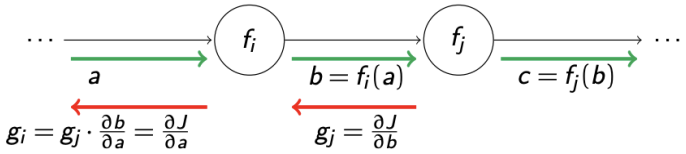
- order the nodes by topological sort (ie every node appears before it's children)
- for each node compute the output given the input



- so the forward pass from $f_i \to f_j$ looks like this

backwards pass

- order the nodes in reverse topological order so every child comes before every parent
- for each node find it's partial derivatives with respect to it's inputs, multiplied by the partial derivatives of it's children (the chain rule)



-
- it is better to do backwards since, we have a scaler output and vector input so it takes less memory to store
- local mins, snaffle points,flat regions, and high curvature areas are all issues
- learning rates are an important parameter to pay attention to in practice