Recitation 12

Neural Networks and Backprop

Vishakh

CDS

April 20, 2022

Announcement

- HW 6 is due on Friday night + HW 7(!) will be out and due in 2 weeks
- HW 5 grades by this weekend
- Regrades taking time

Agenda

1. Announcement
2. Feature Learning and Neural Networks
3. Backprop
4. Takeaways

Motivation

- Many problems are non-linear
- Previously we looked at kernel-based solutions

Motivation

- Many problems are non-linear
- Previously we looked at kernel-based solutions
- Essentially transforms features into another space and performs linear classification in the new space

Motivation

- Many problems are non-linear
- Previously we looked at kernel-based solutions
- Essentially transforms features into another space and performs linear classification in the new space
- Maybe you can hand-craft a kernel with domain knowledge but your learning algorithm does not influence the transformation (except some parameters)

Motivation

- Many problems are non-linear
- Previously we looked at kernel-based solutions
- Essentially transforms features into another space and performs linear classification in the new space
- Maybe you can hand-craft a kernel with domain knowledge but your learning algorithm does not influence the transformation (except some parameters) $\rightarrow$ Neural Networks

Neural Networks and Backprop

- Representation/Feature learning is now part of the algorithm
- Initial architecture motivated by ideas from neuroscience

Neural Networks and Backprop

- Representation/Feature learning is now part of the algorithm
- Initial architecture motivated by ideas from neuroscience
- Expressivity is key here $\rightarrow$ Universal approximation theorem

Universal approximation theorem

Theorem (Universal approximation theorem)

A neural network with one possibly huge hidden layer $\hat{F}(x)$ can approximate any continuous function $F(x)$ on a closed and bounded subset of $\mathbb{R}^d$ under mild assumptions on the activation function, i.e. $\forall \epsilon > 0$, there exists an integer N s.t.

$$\hat{F}(x) = \sum_{i=1}^{N} w_i \sigma(v_i^T x + b_i) \tag{1}$$

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

Universal approximation theorem

Theorem (Universal approximation theorem)

A neural network with one possibly huge hidden layer $\hat{F}(x)$ can approximate any continuous function $F(x)$ on a closed and bounded subset of $\mathbb{R}^d$ under mild assumptions on the activation function, i.e. $\forall \epsilon > 0$, there exists an integer N s.t.

$$\hat{F}(x) = \sum_{i=1}^{N} w_i \sigma(v_i^T x + b_i) \tag{2}$$

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

Activation Functions

- We want non-linearity $\rightarrow$ introduced via activation functions.
- The choice of activation function affects performance
  - Computationally cheap
  - Differentiable
  - Vanishing gradients
- Sigmoid, Tanh, ReLU

Activation Functions

- We want non-linearity $\rightarrow$ introduced via activation functions.
- The choice of activation function affects performance
  - Computationally cheap
  - Differentiable
  - Vanishing gradients
- Sigmoid, Tanh, ReLU
- Two-layer neural network (one hidden layer and one output layer) with K hidden units and sigmoid activation:

$$f(x) = \sum_{k=1}^{K} w_k h_k(x) = \sum_{k=1}^{K} w_k \sigma({v_k}^T x) \tag{3}$$

Question 1: Step Activation Function [1]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Can polynomials of degree one, i.e. $l(x) = ax + b$, be exactly represented by this neural network?

[1] From CMU

Question 1: Step Activation Function [1]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Can polynomials of degree one, i.e. $l(x) = ax + b$, be exactly represented by this neural network?

- No. $f(x) = w_0 + \sum_i w_i \mathbb{1}_{h_i \geq 0}$

[1] From CMU

Question 1: Step Activation Function [2]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Can polynomials of degree two, i.e. $l(x) = ax^2 + bx + c$, be exactly represented by this neural network?

[2] From CMU

Question 1: Step Activation Function [2]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Can polynomials of degree two, i.e. $l(x) = ax^2 + bx + c$, be exactly represented by this neural network?

- No. Same reason.

[2] From CMU

Question 1: Step Activation Function [3]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Can piece-wise constant functions, i.e. $l(x) = c; a \leq x \leq b$, be exactly represented by this neural network?

[3] From CMU

Question 1: Step Activation Function [3]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Can piece-wise constant functions, i.e. $l(x) = c; a \leq x \leq b$, be exactly represented by this neural network?

- Yes! $f(x) = c(g(x-a)) - c(g(x-b))$

[3] From CMU

Question 1: Step Activation Function [4]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = z$$

Can polynomials of degree one, i.e. $l(x) = ax + b$, be exactly represented by this neural network?

[4] From CMU

Question 1: Step Activation Function [4]

Suppose we have a neural network with one hidden layer.

$$f(x) = w_0 + \sum_i w_i h_i(x); \quad h_i(x) = g(b_i + v_i x),$$

where activation function g is defined as

$$g(z) = z$$

Can polynomials of degree one, i.e. $l(x) = ax + b$, be exactly represented by this neural network?

- Yes!
  $f(x) = w_0 + w_1 v_1 x + w_1 b_1$, i.e. choose $a = w_1 v_1$ and $b = w_0 + w_1 b_1$

[4] From CMU

Question 2: Power of ReLU [5]

Consider the following small NN:

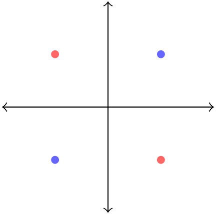
$$w_2^\top \mathbf{ReLU}\left(W_1 x + b_1\right) + b_2$$

where the data is 2D W_1 is 2 by 2, b_1 is 2D, w_2 is 2D and b_2 is 1D.

$$x_1 = (1,1) \;\; y_1 = 1; \quad x_2 = (1,-1) \;\; y_2 = -1;$$

$$x_3 = (-1,1) \;\; y_3 = -1; \quad x_4 = (-1,-1) \;\; y_4 = 1$$

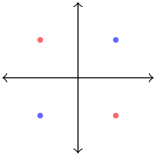
Find b_1, b_2, W_1, w_2 to solve the problem.



[5] From Harvard

Question 2: Power of ReLU

$$w_2^\top \textbf{ReLU}\left(W_1 x + b_1\right) + b_2$$

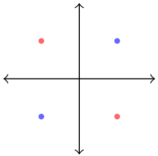


We want to find a mapping that differentiates $(1,1),(-1,-1)$ from $(-1,1),(1,-1)$. Also remember ReLU:

$$g(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Question 2: Power of ReLU

$$w_2^\top \textbf{ReLU}\left(W_1 x + b_1\right) + b_2$$



We want to find a mapping that differentiates $(1, 1), (-1, -1)$ from $(-1, 1), (1, -1)$. Also remember ReLU:

$$g(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

One choice is

$$W_1 = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}, b_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, w_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, b_2 = -1$$

Question 2: Power of ReLU

$$f(x) = w_2^\top \textbf{ReLU}\left(W_1 x + b_1\right) + b_2$$

Remember ReLU:

$$g(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Question 2: Power of ReLU

$$f(x) = w_2^\top \mathbf{ReLU}\left(W_1 x + b_1\right) + b_2$$

Remember ReLU:

$$g(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

One choice is

$$W_1 = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}, b_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, w_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, b_2 = -1$$

x	$h(x)$	$g(h(x))$	$f(x)$
$(1, 1)$	$[2 \quad -2]$	$[2 \quad 0]$	1
$(-1, 1)$	$[0 \quad 0]$	$[0 \quad 0]$	-1
$(1, -1)$	$[0 \quad 0]$	$[0 \quad 0]$	-1
$(-1, -1)$	$[-2 \quad 2]$	$[0 \quad 2]$	1

Backpropogation

- How do we find the right parameters? $\rightarrow$ Backprop
- Backprop enables you to calculate the gradients of all the parameters in your network systematically using the chain rule

Backpropogation

- How do we find the right parameters? $\rightarrow$ Backprop
- Backprop enables you to calculate the gradients of all the parameters in your network systematically using the chain rule
- Forward pass $\rightarrow$ calculate the loss on example (same as the different optimization algorithms we've seen)
- Backward pass $\rightarrow$ calculate partial dericative of the loss w.r.t each parameter, caching intermediate results

Question 3: Backpropagation [6]

Suppose we have a one hidden layer network and computation is:

$$h = \mathrm{ReLU}(z_1) = \mathrm{ReLU}(Wx + b_1)$$
$$\hat{y} = \mathrm{softmax}(z_2) = \mathrm{softmax}(Uh + b_2)$$
$$J = \text{Cross entropy}(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$$

The dimensions of the matrices are:

$$W \in \mathbb{R}^{m\times n} \quad x \in \mathbb{R}^n \quad b_1 \in \mathbb{R}^m \quad U \in \mathbb{R}^{k\times m} \quad b_2 \in \mathbb{R}^k$$

X is the input here, the rest are the model parameters that we want to optimize. Use backpropagation to calculate these four gradients

$$\frac{\partial J}{\partial b_2} \quad \frac{\partial J}{\partial U} \quad \frac{\partial J}{\partial b_1} \quad \frac{\partial J}{\partial W}$$

[6] From Stanford

Question 3: Backpropagation

$$z_2 = Uh + b_2$$
$$\hat{y} = \text{softmax}(z_2)$$

Question 3: Backpropagation

$$z_2 = Uh + b_2$$
$$\hat{y} = \text{softmax}(z_2)$$
$$\delta_1 = \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} = \hat{y} - y \quad \textit{(Reference)}$$

Question 3: Backpropagation

$$z_2 = Uh + b_2$$

$$\hat{y} = \mathrm{softmax}(z_2)$$

$$\delta_1 = \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} = \hat{y} - y \quad \textit{(Reference)}$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial z_2} \frac{\partial z_2}{\partial b_2} = \delta_1 \times 1 = \delta_1$$

Question 3: Backpropagation

$$z_2 = Uh + b_2$$

$$\hat{y} = \text{softmax}(z_2)$$

$$\delta_1 = \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_2} = \hat{y} - y \quad (Reference)$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial b_2} = \delta_1 \times 1 = \delta_1$$

$$\frac{\partial J}{\partial U} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial U} = \delta_1 h^T$$

Question 3: Backpropagation

$$z_2 = Uh + b_2$$

$$\hat{y} = \text{softmax}(z_2)$$

$$\delta_1 = \frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} = \hat{y} - y \quad (Reference)$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial z_2} \frac{\partial z_2}{\partial b_2} = \delta_1 \times 1 = \delta_1$$

$$\frac{\partial J}{\partial U} = \frac{\partial J}{\partial z_2} \frac{\partial z_2}{\partial U} = \delta_1 h^T$$

$$\frac{\partial J}{\partial h} = \frac{\partial J}{\partial z_2} \frac{\partial z_2}{\partial h} = U^T \delta_1$$

Question 3: Backpropagation

$$z_1 = Wx + b_1$$
$$h = \mathrm{ReLU}(z_1)$$

Question 3: Backpropagation

$$z_1 = Wx + b_1$$
$$h = \mathrm{ReLU}(z_1)$$
$$\delta_2 = \frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial h}\frac{\partial h}{\partial z_1} = (U^T\delta_1) \circ 1\{h > 0\}$$

Question 3: Backpropagation

$$z_1 = Wx + b_1$$
$$h = \mathrm{ReLU}(z_1)$$
$$\delta_2 = \frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial h}\frac{\partial h}{\partial z_1} = (U^T \delta_1) \circ 1\{h > 0\}$$
$$\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial b_1} = \delta_2 \times 1 = \delta_2$$

Question 3: Backpropagation

$$z_1 = Wx + b_1$$
$$h = \mathrm{ReLU}(z_1)$$
$$\delta_2 = \frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial h}\frac{\partial h}{\partial z_1} = (U^T \delta_1) \circ 1\{h > 0\}$$
$$\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial b_1} = \delta_2 \times 1 = \delta_2$$
$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial W} = \delta_2 x^T$$

Takeaways

- Neural networks are helpful tools to solve non-linear problems
- Activations are important to this non-linearity and the choice of activation affects performance
- NNs have a large number of parameters and we learn these via backpropagation