Minor_answers

Monday, 27 March 2023 10:21 AM

1 i) $\sigma(z) = \dfrac{1}{1+e^{-z}}$

$$\sigma'(z) = \frac{-(-e^{-z})}{(1+e^{-z})^2} = \frac{e^{-z}+1-1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2}.$$

$$= \sigma(z) - [\sigma(z)]^2.$$

$$= \sigma(z)[1-\sigma(z)]$$

⟨Hence, proved⟩

1) ii) $\sigma(z) = \dfrac{1}{1+e^{-z}}$

$$\Rightarrow \sigma(-z) = \frac{1}{1+e^{z}} = \frac{\left(\frac{1}{e^{-z}}\right)}{\left(\frac{1+e^{-z}}{e^{-z}}\right)} = \frac{e^{-z}}{e^{-z}+1} = \frac{e^{-z}+1-1}{e^{-z}+1}$$

$$= 1 - \frac{1}{1+e^{-z}}.$$

$$= 1-\sigma(z)$$

⟨Hence, proved⟩.

1] iii] lets say, $\sigma^{-1}(z) = \ln\left(\dfrac{z}{1-z}\right)$.

Now; $\sigma(z) = \dfrac{1}{1+e^{-z}}$

$\Rightarrow \sigma^{-1}\left(\dfrac{1}{1+e^{-z}}\right) = z$ ⟨we will try to prove this using first tone relation⟩

Therefore; $\sigma^{-1}\left(\dfrac{1}{1+e^{-z}}\right) = \ln \dfrac{\frac{1}{(1+e^{-z})}}{\left[\frac{e^{-z}}{1+e^{-z}}\right]} = \ln\left(\dfrac{1}{e^{-z}}\right) = z.$

⟨Hence, proved⟩

K

2) i] The size of the weight matrix $W^{(2)}$ is $(12,1)$.

ii] The total number of parameters needed to train the model (including bias) is $[\underbrace{36}_{\text{weights}}+\underbrace{12}_{\text{Bias.}}] = 48$

iii] $a_j^{(l)} = \sigma(z_j^{(l)})$ and

$$\Rightarrow z_k^{(l+1)} = \sum_{k=1}^{s_{l+1}} w_{kj}^{(l+1)} a_j^{(l)} + b_k^{(l+1)}. \quad \text{and} \quad z_k^{(l)} = \sum w_{kj}^{(l)} a_j^{(l-1)} + b_k^{(l)}.$$

iv] $$a_j^{(l)} = \sigma(z_j^{(l)}). \longrightarrow (1).$$

$$z_j^{(l)} = \sum w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} \longrightarrow (2).$$

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}$$

Now; $$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \frac{\partial}{\partial w_{jk}^{(l)}}\left[w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right] = a_k^{(l-1)}.$$

$$\therefore \frac{\partial J}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)}. \longrightarrow (3).$$

and $$\frac{\partial J}{\partial b_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \longrightarrow (4).$$

where $\delta_j^{(l)} \triangleq \dfrac{\partial J}{\partial z_j^{(l)}}$.

Hence; finally, $\frac{\partial J}{\partial W_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)}$ and $\frac{\partial J}{\partial b_j^{(l)}} = \delta_j^{(l)}$.

at output $(l=L)$ layer :— δ_j^L

$$\delta_j^L = \frac{\partial J}{\partial z_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \cdot \sigma'(z_j^{(L)}).$$

$$\therefore \quad \delta_j^{(L)} = \left(a_j^{(L)} - y_j\right) \sigma'\left(z_j^{(L)}\right) \quad \text{as} \quad J = \frac{1}{2}\left(a_j^{(L)} - y_j\right)^2.$$

Hence; $\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \frac{\partial}{\partial z_j^{(l)}} \sigma\left(z_j^{(l)}\right) = \sigma'\left(z_j^{(l)}\right)$ → for output layer only.

<u>Generalisation</u> :— to generalise expression of $\delta_j^{(l)}$ in an arbitrary layer (l), utilising the chain rule;

$$\delta_j^{(l)} = \sum_{k=1}^{S_{l+1}} \frac{\partial J}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$; k = no. of neurons in $(l+1)^{th}$ layer

$$= \sum_k \left\{ \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \delta_k^{(l+1)} \right\} \to (5).$$

with $a_j^{(l)} = \sigma(z_j^{(l)})$ and

$$z_k^{(l+1)} = \sum_{j=1}^{S_l} W_{kj}^{(l+1)} a_j^{(l)} + b_k^{(l+1)}$$

$$= \sum_j^{S_l} W_{kj}^{(l+1)} \sigma\left(z_j^{(l)}\right) + b_k^{(l+1)}. \to (6).$$

from Eq (6) we get

$$\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = W_{kj}^{(l+1)} \sigma'\left(z_j^{(l)}\right). \to (7).$$

Substituting Eq (7) to (5) :—

$$\delta_j^{(l)} = \sum_{k=1}^{s_{l+1}} w_{kj}^{(l+1)} \, \delta_k^{(l+1)} \, \sigma'(z_j^{(l)}) \longrightarrow (8).$$

3] a) Number of filters = 112.

b) Padding size of convolution layer = (258 × 258)

c) filter size for pooling layer :- (2×2).

4] i]

158	183	172
229	237	238
164	232	233

Dimension of the output (3×3).

ii] Dimension of $\frac{\partial L}{\partial X}$ is (1×1).

Dimension of $\frac{\partial L}{\partial W}$ is (3×3).

iii] $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \circledast W^{flip(180°)}$. convolution

$W^{flip(180°)} \Rightarrow$

3	8	3
2	7	9
5	0	2

$\Rightarrow$

5	0	2
2	7	9
3	8	3

Row 180° flip.

$\Rightarrow$

2	0	5
9	7	2
3	8	3

Complete row-column 180° flip.

$$\frac{\partial L}{\partial Y} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}_{3\times3}.$$

3	11	14	11	3.
5	20	32	27	12
10	25	39	29	14

2	0	5
9	7	2
3	8	3

1	1	1
1	1	1
1	1	1

3	11	14	11	3
5	20	32	27	1
10	25	39	29	1
7	14	25	18	1
5	5	7	2	2

$\therefore$

7	14	25	18	11
5	5	7	2	2

iv] $\frac{\partial L}{\partial W} = X \circledast \frac{\partial L}{\partial Y}$.

$$= \begin{bmatrix} 3 & 1 & 4 & 1 & 5 \\ 9 & 2 & 6 & 5 & 3 \\ 5 & 8 & 9 & 7 & 9 \\ 3 & 2 & 3 & 8 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix} \circledast \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 47 & 43 & 49 \\ 47 & 50 & 54 \\ 36 & 46 & 52 \end{bmatrix}$$

5] i) The issue with zero initialisation of weights and biases in a neural network is that it can lead to the "symmetry problem".

When wts and biases are initialized to zero; all neurons in a layer will compute the same output, making it difficult for the network to learn meaningful representations of the input data. This is because the weights will remain same during backpropagation and gradients will be identical for each neuron in a layer.

Zero Initialisation can also result in the vanishing gradient problem, where the gradients become very small as they propagate through multiple layers of the network, making it difficult for the network to learn.

ii] Scenarios where you might see vanishing gradients in a NN:—

1) as depth of NN increases, the gradient can become very small due to the repeated multiplication of small values.

2) Some activation functions example sigmoid function can saturate (i.e output values close to 0 or 1) for large or small inputs, which can lead to small gradients.

3] If weights are initialised to small values; the gradients can also become very small.

Tackle the Vanishing Gradient Problem :-

1] Initializing the weights with larger values [or use Xavier Initialization] can help to avoid small gradients.

2] Use activation functions that donot saturate; such as ReLU to avoid vanishing gradients.

3] Normalizing inputs to each layer of NN can help to stabilize the gradient magnitudes.

4] Using skip connections (i.e in a Residual network) can help gradients flow more easily in the network.

iii] The problem of exploding gradients can occur during the training of neural networks when the gradients become too large. When the gradients are too large, they can cause weights of NN to update in large increments, which can lead to unstable behaviour and cause the network to diverge during NN.

Exploding gradients can also occur while using activation functions that have very steep gradients such as Sigmoid function.

iv] In general; more parameters in a NN; greater the capacity to fit the training data, and more likely it is to overfit.

⇒ Hence, in 1st case there is more chance to overfit the training dataset. This is because the model have more capacity to fit perfectly in training dataset, but may not generalise well to new, unseen data.

v] Operator learning is a type of m/c learning where the goal is to learn a function or operator that maps the input data to output data. In other words, the focus is on learning the mathematical operators that transform input data into desired output.

Example:- In image processing, where goal is to learn function that maps a noisy image to a denoised version of same image. The operator is a function takes in noisy image as input and applies a set of mathematical operators to generate denoised image as output.

vi]
$$Loss_{(physics)} = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| \left(\frac{\partial^2}{\partial x^2} + \frac{\partial}{\partial x} + 1 \right) \hat{y}(x_i) \right|^2.$$

$$Loss_{BC/IC} = \frac{1}{N_b} \sum_{i=1}^{N_b} \left[\left| \hat{y}(0) - y_0 \right|^2 + \left| \hat{y}(L) - y_L \right|^2 \right]$$

vii] CNNs are less sensitive to spatial translation of objects within an image than FCNN because of their local connectivity and weight sharing properties.

In CNN, each layer contains a set of filters that learn to recognise patterns in the input data. These filters are applied to small regions of input image and same filter is applied to all regions. This is called weight sharing. Because of this property, the learned filters are able to recognize same pattern in different regions of the image.

Again, as filters are moved across the image, allows the network to detect the presence of a pattern regardless of where it is located in image. This property is called Local Connectivity.

Due to these two properties; the learned filters in CNN can detect same features in different regions of an image, regardless of the location of the feature. This makes network less sensitive to spatial translation of object within an image, as the same features will be detected regardless of where they appear in the image.

In contrast, FCNN do not have weight sharing or local connectivity properties.

viii] Zero-shot generalization is the ability of a model to perform a task without any specific training examples for that particular task. Instead, model is trained to perform related tasks and can use its knowledge to perform new task.

<u>Example</u>:- A language model has been trained over wide range of tasks e.g translation, summarization etc. If you give it a new task; such as generating a caption for an image, it may be able to perform this task without any specific training examples for image captioning. The model can use its understanding of language and its relationship with words to generate caption that accurately describes an image.