Backpropagation in MLP

(Tensor-level Computation)

Tirtharaj Dash

Dept. of CS & IS and APPCAIR
BITS Pilani, Goa Campus

September 7, 2021

Where we are:

Summary of the previous lecture:

- Backpropagation with scalar-level computation

Today: Backpropagation with tensor-level computation

- Forward propagation
- Backwatd propagation
- Computational graphs

Introduction

- We saw the complex mathematical derivations for backpropagation at a scalar-level. Note that this used to be the case in many research works before the era of PyTorch, Tensorflow and suchlike.
- Note that in today's deep learning era, you don't have to write the derivation computation codes yourself. Thanks to automatic differentiation engines such as Autograd in PyTorch.
- The vectorisation method of linear algebra allows us to represent these complex scalar-level operations at a tensor-level. This in turn allows us to use available computational hardware to effeciently obtain computed results in parallel.
- In this lecture, we stick to the same architecture (that is, 1-hidden layer MLP).

Forward Propagation I

- Let's assume that the input example is $\mathbf{x} \in \mathbb{R}^d$, and for simplicity, we assume that the hidden layer does not include a bias term.
- There are some notational differences with regard to our previous lecture:

Notation for	Prev. Lecture	Today
Number of units in the hidden layer:	m	h
Hidden layer activation function:	$\sigma^{(1)}$	ϕ
Hidden layer activations:	$\mathbf{a}$	$\mathbf{h}$
Number of units in the output layer:	c	q
Output layer activation function:	$\sigma^{(2)}$	linear
Outputs:	$\hat{\mathbf{y}}$	$\mathbf{o}$

At least to me, our previous lecture notations are more convenient. However, let's stay consistent with our textbook 1.

Forward Propagation II

- Now the net input to the hidden layer is:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$$

  where $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ is the matrix representing the weight parameters of the hidden layer.

- The hidden activations (or, hidden representation) are then computed by running $\mathbf{z} \in \mathbb{R}^{h}$ through the hidden layer activation $\phi(\cdot)$:

$$\mathbf{h} = \phi(\mathbf{z})$$

Forward Propagation III

- Let the output layer weights be denoted by the matrix $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$. Then the output vector is (assuming linear activation function):

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$$

Forward Propagation IV

- The loss function l for a single data instance is:

$$L = l(\mathbf{o}, y)$$

where y is the example label.

- Additionally, let's assume we have L_2-regularisation with hyperparameter λ:

$$s = \frac{\lambda}{2}\left(||\mathbf{W}^{(1)}||_F^2 + ||\mathbf{W}^{(2)}||_F^2\right)$$

where the Frobenius norm of the matrix is simply the L_2 norm applied after flattening the matrix into a vector.

We will study the theory on regularisation in the next lecture.

Forward Propagation V

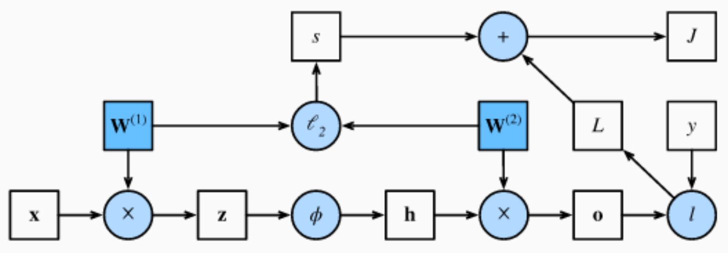
- Finally, the model's regularised loss on a given data example is:

$$J = L + s$$

We refer to J as the objective function.

Computational Graph of Forward Propagation

- Computational graph allows us to visualise the dependencies of operators and variables with the calculation.



- Here □s denote variables and ◯s denote the operators. $\rightarrow$s and $\uparrow$s show direction of data-flow.

Backpropagation I

Chain-rule at tensor-level:

- Assume that we have functions $\mathsf{Y} = f(\mathsf{X})$ and $\mathsf{Z} = g(\mathsf{Y})$, in which the input and the output $\mathsf{X}, \mathsf{Y}, \mathsf{Z}$ are tensors of arbitrary shapes.
- By using the chain rule of derivatives, we can compute the derivative of Z w.r.t. X via:
$$\frac{\partial \mathsf{Z}}{\partial \mathsf{X}} = \mathrm{prod}\left(\frac{\partial \mathsf{Z}}{\partial \mathsf{Y}}, \frac{\partial \mathsf{Y}}{\partial \mathsf{X}}\right)$$
- The operator prod hides all these notation overheads such as transposition, swapping input positions, etc.

Note: We have already discussed gradient of a vector w.r.t. another vector in our tutorial on optimisation (Refer Tutorial 3).

Backpropagation II

Backpropation for our 1-hidden layer MLP:

- Our goal is to calculate: $\frac{\partial J}{\partial \mathbf{W}^{(1)}}$ and $\frac{\partial J}{\partial \mathbf{W}^{(2)}}$
- We already know how chain-rule applies in backpropagation. So let's get started with the first step.

Backpropagation III

- Calculating the gradients of the objective function $J = L + s$ w.r.t. the loss term L and the regularisation term s.

$$\frac{\partial J}{\partial L} = 1$$

and

$$\frac{\partial J}{\partial s} = 1$$

Backpropagation IV

- Next, we compute the gradient of J w.r.t. output vector according to the chain rule:

$$\frac{\partial J}{\partial \mathbf{o}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}\right) = \frac{\partial L}{\partial \mathbf{o}} \in \mathbb{R}^q$$

Backpropagation V

- Next, we calculate the gradients of the regularisation term w.r.t. the weight matrices:

$$\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)}$$

$$\frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}$$

Backpropagation VI

- Now we are able to calculate the gradients $\frac{\partial J}{\partial \mathbf{W}^{(2)}} \in \mathbb{R}^{q \times h}$ of the model parameters closest to the output layer.
- Using the chain-rule yeilds:

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{W}^{(2)}}\right) + \text{prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(2)}}\right)$$
$$= \frac{\partial J}{\partial \mathbf{o}} \mathbf{h}^{\mathsf{T}} + \lambda \mathbf{W}^{(2)}$$

Backpropagation VII

- To calculate the gradients w.r.t. $\mathbf{W}^{(1)}$ we need to continue backpropagation along the output layer to the hidden layer.
- Gradient w.r.t. hidden layer activation $\frac{\partial J}{\partial \mathbf{h}} \in \mathbb{R}^h$ is:

$$\frac{\partial J}{\partial \mathbf{h}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{h}}\right) = \mathbf{W}^{(2)\mathsf{T}} \frac{\partial J}{\partial \mathbf{o}}$$

Backpropagation VIII

- Since the activation function ϕ applies elementwise, calculating the gradient $\frac{\partial J}{\partial \mathbf{z}} \in \mathbb{R}^h$ requires that we use the elementwise multiplication operator, which we denote by $\odot$:

$$\frac{\partial J}{\partial \mathbf{z}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{h}}, \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right) = \frac{\partial J}{\partial \mathbf{h}} \odot \phi'(\mathbf{z})$$

- Finally, we can obtain the gradients $\frac{\partial J}{\partial \mathbf{W}^{(1)}} \in \mathbb{R}^{h \times d}$ of the model parameters closest to the inputs.
- According to the chain-rule we get:

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}^{(1)}} &= \text{prod}\left(\frac{\partial J}{\partial \mathbf{z}}, \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}}\right) + \text{prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(1)}}\right) \\ &= \frac{\partial J}{\partial \mathbf{z}} \mathbf{x}^{\mathsf{T}} + \lambda \mathbf{W}^{(1)}\end{aligned}$$

Backpropagation X

- Parameter update then proceeds as per gradient-descent update rule:

$$\mathbf{w}^{(\ell)} \leftarrow \mathbf{w}^{(\ell)} - \eta \frac{\partial J}{\partial \mathbf{W}^{(\ell)}}$$

  where $\ell = \{1, 2\}$ in our 1-hidden layer MLP case; and this expression can be generalised to any ℓ as seen before.
- η is some learning rate $\in (0, 1]$.