DL in Applied Mathematics
Lecture 5: Backpropagation

Backpropagation in neural networks is a fundamental algorithm used for training deep learning models.

It's essentially the backbone of learning in various types of neural networks including, but not limited to, feedforward neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs).

The Concept of Backpropagation

Backpropagation, short for "backward propagation of errors," is a method used to calculate the gradient of the loss function with respect to each weight in the network by applying the chain rule of calculus, moving backwards through the layers. It consists of two main phases: the forward pass and the backward pass.

Forward Pass

Input Layer: The process begins by feeding the input data into the network.

Hidden Layers: The input data then propagates forward through the network's hidden layers, where each neuron applies a weighted sum on the inputs, followed by a non-linear activation function.

Output Layer: The final output is calculated and compared against the target value to compute the loss (or error).

Backward Pass

- **Compute Gradient**: The backpropagation algorithm then calculates the gradient of the loss function with respect to each weight by propagating the loss backward through the network layers. This involves applying the chain rule to find out how much each weight contributed to the error.
- **Update Weights**: The computed gradients are then used to update the weights of the network. This is usually done using an optimization algorithm like Gradient Descent. The idea is to adjust each weight in the direction that most reduces the loss.

Key Components

- **Loss Function**: Measures the difference between the network's prediction and the actual target values. Common loss functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy for classification tasks.
- **Gradient Descent**: An optimization algorithm used to minimize the loss function by iteratively moving towards the minimum of the loss function.
- **Learning Rate**: A hyperparameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. Too small a learning rate may result in a long training process, while too large a learning rate may lead to overshooting the minimum.

Challenges and Solutions

- **Vanishing/Exploding Gradients**: As the error gradient is propagated back, it can diminish exponentially (vanish) or increase exponentially (explode), making it difficult for the model to learn. Techniques like normalization, proper weight initialization, and architectures designed to mitigate these issues (like LSTM units in RNNs) can help.
- **Overfitting**: This occurs when the model learns the noise in the training data to the extent that it performs poorly on unseen data. Techniques such as regularization, dropout, and early stopping are commonly used to combat overfitting.

Backpropagation has been crucial for the advancement of deep learning, enabling the training of deep neural networks that can learn from complex data and perform tasks ranging from image recognition to natural language processing.

Back Propagation. The problem statments

So we switch from the general vector of parameters, *p*, to the entries in the weight matrices and bias vectors.

Our task is to compute partial derivatives of the cost function with respect to each $\omega_{jk}^{[l]}$ and $b_j^{[l]}$.

We saw that the idea behind the stochastic gradient method is to exploit the structure of the cost function: because

$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N} \tfrac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2, \qquad (9)$$

is a linear combination of individual terms that runs over the training data the same is true of its partial derivatives.

We therefore focus our attention on computing those individual partial derivatives.

Hence, for a fixed training point we regard $C_{x^{\{i\}}}$ in (13)

$$C_{x^{\{i\}}} = \frac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2. \tag{13}$$

as a function of the weights and biases. So we may drop the dependence on $x^{\{\mathrm{i}\}}$ and simply write

$$C = \frac{1}{2}\|y - a^{[L]}\|_2^2. \tag{18}$$

We recall from (8)

$$a^{[l]} = \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}, \quad \text{for } l = 2, 3, \ldots, L. \tag{8}$$

that $a^{[L]}$ is the output from the artificial neural network. The dependence of *C* on the weights and biases arises only through $a^{[L]}$.

To derive worthwhile expressions for the partial derivatives, it is useful to introduce two further sets of variables. First we let

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l}, \qquad \text{for } l = 2, 3, \ldots, L. \tag{19}$$

We refer to $z_j^{[l]}$ as the *weighted input* for neuron *j* at layer l . The fundamental relation (8)

$$a^{[l]} = \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}, \quad \text{for } l = 2, 3, \ldots, L. \tag{8}$$

that propagates information through the network may then be written

$$a^{[l]} = \sigma\left(z^{[l]}\right), \quad \text{for } l = 2, 3, \ldots, L. \tag{20}$$

Second, we let $\delta^{[l]} \in \mathbb{R}^{n_l}$ be defined by

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \quad \text{for } 1 \leq j \leq n_l \quad \text{and} \quad 2 \leq l \leq L. \tag{21}$$

This expression, which is often called the *error* in the *j* th neuron at layer l , is an intermediate quantity that is useful both for analysis and computation.

However, we point out that this useage of the term error is somewhat ambiguous. At a general, hidden layer, it is not clear how much to "blame" each neuron for discrepancies in the final output

Also, at the output layer, *L*, the expression (21) does not quantify those discrepancies directly.

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \qquad \text{for } 1 \leq j \leq n_l \quad \text{and} \quad 2 \leq l \leq L. \tag{21}$$

The idea of referring to $\delta_j^{[l]}$ in (21) as an error seems to have arisen because the cost function can only be at a minimum if all partial derivatives are zero, so $\delta_j^{[l]} = 0$ is a useful goal.

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \qquad \text{for } 1 \le j \le n_l \quad \text{and} \quad 2 \le l \le L. \tag{21}$$

As we mention later , it may be more helpful to keep in mind that $\delta_j^{[l]}$ measures the sensitivity of the cost function to the weighted input for neuron *j* at layer l .

At this stage we also need to define the Hadamard, or componentwise, product of two vectors. If x, $y \in \mathbb{R}^n$, then $x \circ y \in \mathbb{R}^n$ is defined by $(x \circ y)_i = x_i y_i$. In words, the Hadamard product is formed by pairwise multiplication of the corresponding components.

With this notation, the following results are a consequence of the chain rule.

Lemma 1 We have

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^L - y), \tag{22}$$

$$\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]}, \quad \text{for } 2 \leq l \leq L-1, \tag{23}$$

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}, \quad \text{for } 2 \leq l \leq L, \tag{24}$$

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]}. \quad \text{for } 2 \leq l \leq L. \tag{25}$$

Proof We begin by proving (22).

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^L - y), \tag{22}$$

The relation (20)

$$a^{[l]} = \sigma\left(z^{[l]}\right), \quad \text{for } l = 2, 3, \ldots, L. \tag{20}$$

with $l = L$ shows that $z_j^{[L]}$ and $a_j^{[L]}$ are connected by $a^{[L]} = \sigma\left(z^{[L]}\right)$, and hence

$$\frac{\partial\, a_j^{[L]}}{\partial\, z_j^{[L]}} = \sigma'(z_j^{[L]}).$$

Also, from (18) ,

$$\frac{\partial C}{\partial a_j^{[L]}} = \frac{\partial}{\partial a_j^{[L]}} \frac{1}{2} \sum_{k=1}^{n_L} (y_k - a_k^{[L]})^2 = -(y_j - a_j^{[L]}).$$

So, using the chain rule,

$$\delta_j^{[L]} = \frac{\partial C}{\partial z_j^{[L]}} = \frac{\partial C}{\partial a_j^{[L]}} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = (a_j^{[L]} - y_j)\sigma'(z_j^{[L]}),$$

which is the componentwise form of (22).

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^L - y), \qquad (22)$$

To show (23),

$$\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]}, \qquad \text{for } 2 \le l \le L-1, \quad (23)$$

we use the chain rule to convert from $z_j^{[l]}$ to $\left\{z_k^{[l+1]}\right\}_{k=1}^{n_{l+1}}$.

Applying the chain rule, and using the definition (21),

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \qquad \text{for } 1 \le j \le n_l \quad \text{and} \quad 2 \le l \le L. \quad (21)$$

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}}. \quad (26)$$

Now, from (19) we know that $z_k^{[l+1]}$ and $z_j^{[l]}$ are connected via

$$z_k^{[l+1]} = \sum_{s=1}^{n_l} w_{ks}^{[l+1]} \sigma\left(z_s^{[l]}\right) + b_k^{[l+1]}.$$

Hence,

$$\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = w_{kj}^{[l+1]} \sigma'\left(z_j^{[l]}\right).$$

In (26) this gives

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}}. \tag{26}$$

$$\delta_j^{[l]} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} w_{kj}^{[l+1]} \sigma' \left(z_j^{[l]} \right),$$

which may be rearranged as

$$\delta_j^{[l]} = \sigma' \left(z_j^{[l]} \right) \left((W^{[l+1]})^T \delta^{[l+1]} \right)_j.$$

This is the componentwise form of (23).

$$\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]}, \qquad \text{for } 2 \leq l \leq L-1, \quad (23)$$

To show (24),

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}, \qquad \text{for } 2 \leq l \leq L, \quad (24)$$

we note from (19) and (20) that $z_j^{[l]}$ is connected to $b_j^{[l]}$ by

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l}, \qquad \text{for } l = 2, 3, \ldots, L. \quad (19)$$

$$a^{[l]} = \sigma\left(z^{[l]}\right), \qquad \text{for } l = 2, 3, \ldots, L. \quad (20)$$

$$z_k^{[l+1]} = \sum_{s=1}^{n_l} w_{ks}^{[l+1]} \sigma\left(z_s^{[l]}\right) + b_k^{[l+1]}.$$

Since $z^{[l-1]}$ does not depend on $b_j^{[l]}$, we find that $$\frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = 1.$$

Then, from the chain rule, $$\frac{\partial C}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} = \delta_j^{[l]},$$

using the definition (21). This gives (24).

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \qquad \text{for } 1 \le j \le n_l \quad \text{and} \quad 2 \le l \le L. \tag{21}$$

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}, \qquad \text{for } 2 \le l \le L, \tag{24}$$

Finally, to obtain (25)

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]}. \qquad \text{for } 2 \le l \le L. \qquad (25)$$

we start with the componentwise version of (19).

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l}, \qquad \text{for } l = 2, 3, \ldots, L. \qquad (19)$$

$$z_j^{[l]} = \sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]},$$

which gives $\dfrac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = a_k^{[l-1]},$ independently of *j* , (27)

and $$\frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = 0, \qquad \text{for } s \neq j. \tag{28}$$

In words, (27) and (28) follow because the *j* th neuron at layer l uses the weights from only the *j* th row of $W^{[l]}$, and applies these weights linearly. Then, from the chain rule, (27) and (28) give

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \sum_{s=1}^{n_l} \frac{\partial C}{\partial z_s^{[l]}} \frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} a_k^{[l-1]} = \delta_j^{[l]} a_k^{[l-1]},$$

where the last step used the definition of $\delta_j^{[l]}$ in (21). This completes the proof.

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \qquad \text{for } 1 \leq j \leq n_l \quad \text{and} \quad 2 \leq l \leq L. \tag{21}$$

There are many aspects of Lemma 1 that deserve our attention. We recall from (7), (19) and (20) that the output a [L] can be evaluated

$$a^{[1]} = x \in \mathbb{R}^{n_1}, \tag{7}$$

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l}, \qquad \text{for } l = 2, 3, \ldots, L. \tag{19}$$

$$a^{[l]} = \sigma\left(z^{[l]}\right), \qquad \text{for } l = 2, 3, \ldots, L. \tag{20}$$

from a *forward pass* through the network, computing $a^{[1]}$,$z^{[2]}$, $a^{[2]}$, $z^{[3]}$, . . . , $a^{[L]}$in order. Having done this, we see from (22) that $\delta^{[L]}$ is immediately available.

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^L - y), \tag{22}$$

Then, from (23),

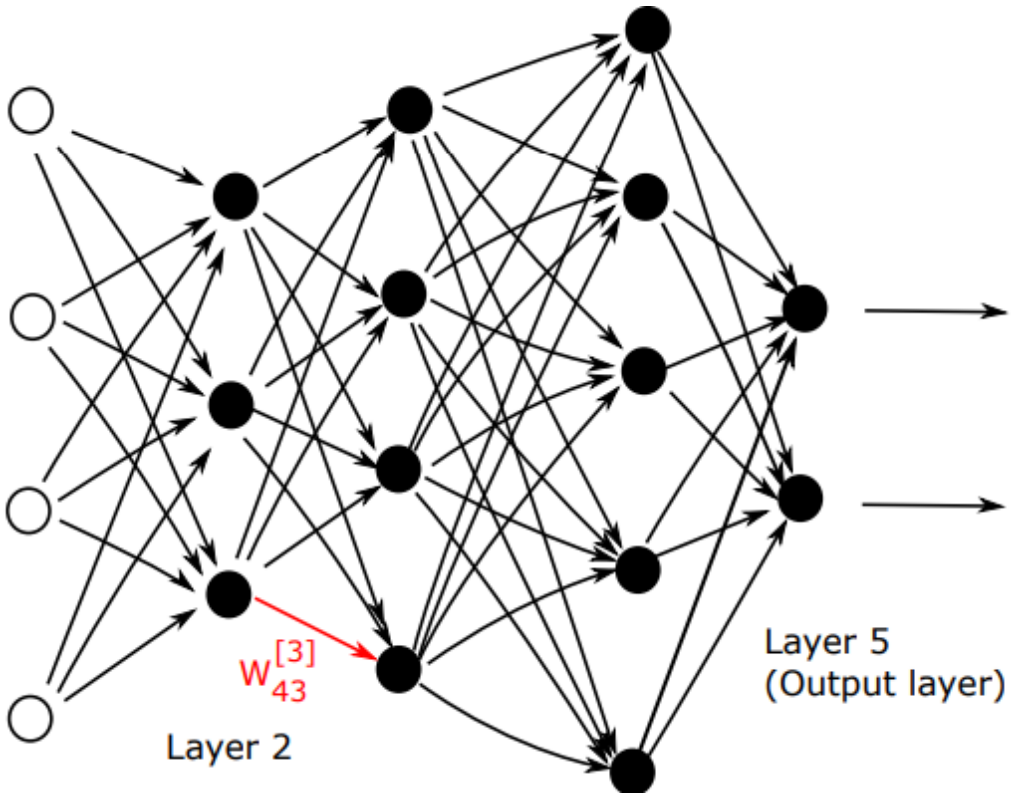
$$\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]}, \qquad \text{for } 2 \leq l \leq L-1, \quad (23)$$

$\delta^{[L-1]}, \delta^{[L-2]}, \ldots, \delta^{[2]}$ may be computed in a *backward pass*. From (24) and (25),

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}, \qquad \text{for } 2 \leq l \leq L, \quad (24)$$

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]}. \qquad \text{for } 2 \leq l \leq L. \quad (25)$$

we then have access to the partial derivatives. Computing gradients in this way is known as *back propagation*.



To gain further understanding of the back propagation formulas (24) and (25) in Lemma 1, it is useful to recall the fundamental definition of a partial derivative.

The quantity $\partial C / \partial \omega_{jk}^{[l]}$ measures how *C* changes whe0n we make a small perturbation to $\omega j_k^{[l]}$.

For illustration, Figure 6 highlights the weight $\omega_{43}^{[3]}$. It is clear that a change in this weight has no e-ffect on the output of previous layers. It is clear that a change in this weight has no effect on the output of previous layers.

It is clear that a change in this weight has no effect on the output of previous layers. So to work out $\partial C / \partial \omega_{43}^{[3]}$ we do not need to know about partial derivatives at previous layers. It should, however, be possible to express $\partial C / \partial \omega_{43}^{[3]}$ in terms of partial derivatives at subsequent layers.

More precisely, the activation feeding into the 4th neuron on layer 3 is $z_4^{[3]}$, and, by definition, $\delta_4^{[3]}$ measures the sensitivity of *C* with respect to this input. Feeding in to this neuron we have $\omega_{43}^{[3]} a_3^{[2]}$+ constant, so it makes sense that

$$\frac{\partial C}{\partial w_{43}^{[3]}} = \delta_4^{[3]} a_3^{[2]}.$$

Similarly, in terms of the bias, $b_4^{[3]}$+ constant is feeding in to the neuron, which explains why

$$\frac{\partial C}{\partial b_4^{[3]}} = \delta_4^{[3]} \times 1.$$

We may avoid the Hadamard product notation in (22) and (23) by introducing diagonal matrices.

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^L - y), \tag{22}$$

$$\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]}, \quad \text{for } 2 \leq l \leq L-1, \tag{23}$$

Let $D^{[l]} \in \mathbb{R}^{nl \times nl}$ denote the diagonal matrix with (i, i) entry given by $\sigma'\left(z_i^{[l]}\right)$. Then we see that $\delta^{[L]} = D^{[L]}\left(a^{[L]} - y\right)$ and $\delta^{[l]} = D^{[L]}\left(W^{[l+1]}\right)^T \delta^{[l+1]}$. We could expand this out as

$$\delta^{[l]} = D^{[l]}(W^{[l+1]})^T D^{[l+1]}(W^{[l+2]})^T \dots D^{[L-1]}(W^{[L]})^T D^{[L]}(a^{[L]} - y).$$

We also recall from (2) that σ $\sigma'(z)$ is trivial to compute.

The relation (24) shows that $\delta^{[l]}$ corresponds precisely to the gradient of the cost function with respect to the biases at layer l. If we regard $\partial C / \partial \omega_{jk}^{[l]}$ as defining the *(j, k)* component in a matrix of partial derivatives at layer l, then (25) shows this matrix to be the *outer product* $\delta^{[l]} a^{[l-1]^T} \in \mathbb{R}^{n_l \times n_{l-1}}$.

Putting this together, we may write the following pseudocode for an algorithm that trains a network using a fixed number, Niter, of stochastic gradient iterations.

For simplicity, we consider the basic version (15) where single samples are chosen with replacement.

$$p \to p - \eta \nabla C_{x\{i\}}(p). \qquad (15)$$

For each training point, we perform a forward pass through the network in order to evaluate the activations, weighted inputs and overall output $a^{[L]}$. Then we perform a backward pass to compute the errors and updates.

For counter = 1 upto Niter

Choose an integer k uniformly at random from {1, 2, 3, . . . , *N*}

$x^{\{k\}}$ is current training data point

$a^{[1]} = x^{\{k\}}$

For l = 2 upto *L*

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = \sigma\left(z^{[l]}\right)$$

$$D^{[l]} = diag\left(\sigma'\left(z^{[l]}\right)\right)$$

end

For l = *L* downto *2*

$$W^{[l]} \rightarrow W^{[l]} - \eta\delta^{[l]}a^{[l-1]^T}$$

$$b^{[l]} \rightarrow b^{[l]} - \eta\delta^{[l]}$$

end

end

Thank you for attention!