

Computation Graphs, Automatic Differentiation, and Backpropagation

Class Notes

1 Why Do We Need These Methods?

Many machine learning models are built by combining simple mathematical operations:

- addition and multiplication;
- powers and exponentials;
- activation functions such as sigmoid and ReLU;
- sums and averages used to calculate a loss.

To train a model, we need to know how changing each parameter changes the final loss. In other words, we need derivatives such as

$$\frac{\partial L}{\partial w_1}, \quad \frac{\partial L}{\partial w_2}, \quad \frac{\partial L}{\partial b}.$$

A computation graph represents a calculation as a sequence of simple operations. Automatic differentiation computes derivatives by applying the chain rule to this graph.

Backpropagation is the efficient procedure used to propagate derivatives backward through the graph.

2 A First Computation Graph

Consider the expression

$$z = (x + y)w.$$

Instead of treating this as one large expression, break it into steps:

$$\begin{aligned} a &= x + y, \\ z &= aw. \end{aligned}$$

The computation graph is:

$$\boxed{x, y} \longrightarrow \boxed{a = x + y} \longrightarrow \boxed{z = aw}$$

with w also feeding into the final multiplication.

Each node represents either an input, an intermediate value, or an operation. The arrows show which values are used to calculate other values.

For example, if

$$x = 2, \quad y = 3, \quad w = 4,$$

then

$$a = 2 + 3 = 5, \quad z = 5 \cdot 4 = 20.$$

The graph describes the forward computation: inputs move through the operations to produce the output.

3 Local Derivatives

The key idea is that every operation has a simple derivative with respect to its inputs. These are called local derivatives.

3.1 Addition

Let

$$z = x + y.$$

The local derivatives are

$$\boxed{\frac{\partial z}{\partial x} = 1}, \quad \boxed{\frac{\partial z}{\partial y} = 1}.$$

Changing either input by a small amount changes the output by the same amount.

Example:

$$z = 2 + 3 = 5.$$

If x increases from 2 to 2.1, then z increases from 5 to 5.1.

3.2 Addition with a Constant

Let

$$z = x + 5.$$

Then

$$\frac{dz}{dx} = 1.$$

The constant does not affect the derivative.

3.3 Multiplication

Let

$$z = xy.$$

The local derivatives are

$$\boxed{\frac{\partial z}{\partial x} = y}, \quad \boxed{\frac{\partial z}{\partial y} = x}.$$

Example: if $x = 2$ and $y = 3$, then $z = 6$, and

$$\frac{\partial z}{\partial x} = 3, \quad \frac{\partial z}{\partial y} = 2.$$

A small change in x has approximately three times that effect on z , while a small change in y has approximately twice that effect.

3.4 Multiplication by a Constant

Let

$$z = 5x.$$

Then

$$\frac{dz}{dx} = 5.$$

For example, increasing x by 0.1 increases z by 0.5.

3.5 Powers

For

$$z = x^n,$$

the derivative is

$$\boxed{\frac{dz}{dx} = nx^{n-1}}.$$

Examples:

Function	Derivative
$z = x^2$	$2x$
$z = x^3$	$3x^2$
$z = x^4$	$4x^3$
$z = \sqrt{x} = x^{1/2}$	$\frac{1}{2\sqrt{x}}$

At $x = 3$, for example, the derivative of x^2 is $2(3) = 6$.

3.6 Exponential and Logarithm

For the exponential function,

$$z = e^x \quad \Longrightarrow \quad \boxed{\frac{dz}{dx} = e^x}.$$

For the natural logarithm,

$$z = \ln(x) \quad \Longrightarrow \quad \boxed{\frac{dz}{dx} = \frac{1}{x}}, \quad x > 0.$$

4 Activation Functions

Neural networks combine weighted sums with activation functions. The activation function introduces nonlinearity.

4.1 Sigmoid

The sigmoid function is

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

It maps any real input to a value between 0 and 1.

Its derivative is

$$\boxed{\sigma'(x) = \sigma(x)(1 - \sigma(x))}.$$

This form is convenient because the derivative can be calculated directly from the sigmoid output.

Examples:

$$\sigma(0) = 0.5, \quad \sigma'(0) = 0.5(1 - 0.5) = 0.25.$$

For $x = 2$,

$$\sigma(2) \approx 0.881,$$

so

$$\sigma'(2) \approx 0.881(1 - 0.881) \approx 0.105.$$

For $x = -2$,

$$\sigma(-2) \approx 0.119,$$

and

$$\sigma'(-2) \approx 0.119(1 - 0.119) \approx 0.105.$$

Notice that the sigmoid derivative is largest near zero and becomes small for very positive or very negative inputs.

4.2 ReLU

The Rectified Linear Unit (ReLU) is

$$\text{ReLU}(x) = \max(0, x).$$

Equivalently,

$$\text{ReLU}(x) = \begin{cases} 0, & x < 0, \\ x, & x \geq 0. \end{cases}$$

Its derivative is

$$\text{ReLU}'(x) = \begin{cases} 0, & x < 0, \\ 1, & x > 0. \end{cases}$$

At $x = 0$, the ordinary derivative is not defined. In practice, software libraries use a chosen subgradient, commonly zero.

Examples:

x	$\text{ReLU}(x)$	$\text{ReLU}'(x)$
-3	0	0
-0.5	0	0
2	2	1
5	5	1

4.3 Comparing Sigmoid and ReLU

Function	Output	Derivative
Sigmoid	between 0 and 1	$\sigma(x)(1 - \sigma(x))$
ReLU	nonnegative	0 or 1, except at 0

Both functions are nonlinear. This matters because stacking only linear operations would still produce a linear function.

5 The Chain Rule

A computation graph is useful because complex derivatives can be calculated from local derivatives.

Suppose

$$z = f(y), \quad y = g(x).$$

Then

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

This is the chain rule.

In words:

Derivative of the final output with respect to the input equals the product of the local derivatives along the path.

5.1 Example 1: A Simple Composition

Let

$$y = x^2, \quad z = 3y.$$

Substituting gives $z = 3x^2$, so

$$\frac{dz}{dx} = 6x.$$

Using the chain rule:

$$\frac{dz}{dy} = 3, \quad \frac{dy}{dx} = 2x.$$

Therefore,

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = 3(2x) = 6x.$$

At $x = 4$,

$$\frac{dz}{dx} = 24.$$

5.2 Example 2: Multiplication Followed by Addition

Consider

$$a = xy, \quad z = a + 5.$$

The graph has two operations:

$$(x, y) \longrightarrow a = xy \longrightarrow z = a + 5.$$

The local derivatives are

$$\frac{\partial z}{\partial a} = 1, \quad \frac{\partial a}{\partial x} = y, \quad \frac{\partial a}{\partial y} = x.$$

Thus,

$$\boxed{\frac{\partial z}{\partial x} = y}, \quad \boxed{\frac{\partial z}{\partial y} = x}.$$

The addition of 5 does not change the derivatives.

6 Backpropagation: Work Backward Through the Graph

Backpropagation is the application of the chain rule to a computation graph, starting at the output and moving backward toward the inputs.

The forward pass calculates values. The backward pass calculates derivatives.

A useful notation is

$$\bar{v} = \frac{\partial L}{\partial v},$$

where L is the final loss. The quantity $\bar{v}$ is sometimes called the gradient or adjoint of v .

For an intermediate variable v , we ask:

How much would the final loss change if v changed?

6.1 Example 3: A Complete Scalar Backward Pass

Consider

$$a = x + y, \quad z = aw,$$

and let the loss be

$$L = z^2.$$

Use the values

$$x = 2, \quad y = 3, \quad w = 4.$$

Forward pass

$$a = x + y = 2 + 3 = 5,$$

$$z = aw = 5 \cdot 4 = 20,$$

$$L = z^2 = 20^2 = 400.$$

Now calculate derivatives backward.

Step 1: Start at the loss

Since

$$L = z^2,$$

we have

$$\frac{\partial L}{\partial z} = 2z = 40.$$

Step 2: Move through the multiplication

Since

$$z = aw,$$

the local derivatives are

$$\frac{\partial z}{\partial a} = w = 4, \quad \frac{\partial z}{\partial w} = a = 5.$$

Apply the chain rule:

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial a} = 40 \cdot 4 = 160,$$

and

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w} = 40 \cdot 5 = 200.$$

Step 3: Move through the addition

Since

$$a = x + y,$$

we have

$$\frac{\partial a}{\partial x} = 1, \quad \frac{\partial a}{\partial y} = 1.$$

Therefore,

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial x} = 160 \cdot 1 = 160,$$

and similarly,

$$\frac{\partial L}{\partial y} = 160.$$

Final result

$$\boxed{\frac{\partial L}{\partial x} = 160, \quad \frac{\partial L}{\partial y} = 160, \quad \frac{\partial L}{\partial w} = 200}$$

The important pattern is that we reuse intermediate derivatives. We do not need to repeatedly differentiate the entire expression from scratch.

7 When a Variable Has Multiple Paths

A node may influence the final output through more than one path. In that case, the contributions from all paths are added.

7.1 Example 4: A Variable Used Twice

Let

$$a = x^2, \quad b = 3x, \quad L = a + b.$$

The same input x affects the loss through both a and b .

The derivative is

$$\frac{dL}{dx} = \frac{da}{dx} + \frac{db}{dx} = 2x + 3.$$

At $x = 2$,

$$\frac{dL}{dx} = 4 + 3 = 7.$$

The two paths contribute:

$$\underbrace{4}_{\text{through } a} + \underbrace{3}_{\text{through } b} = 7.$$

This is why backpropagation accumulates gradients when paths merge.

8 Backpropagation Through Sigmoid

Consider a simple model:

$$z = wx + b, \quad p = \sigma(z).$$

Here p may represent the predicted probability that an example belongs to class 1.

Suppose the loss is squared error:

$$L = (p - y)^2,$$

where y is the target, either 0 or 1.

We want to calculate

$$\frac{\partial L}{\partial w}, \quad \frac{\partial L}{\partial b}.$$

The computation graph is

$$(w, x, b) \longrightarrow z = wx + b \longrightarrow p = \sigma(z) \longrightarrow L = (p - y)^2.$$

Apply the chain rule:

$$\frac{\partial L}{\partial p} = 2(p - y),$$

$$\frac{\partial p}{\partial z} = p(1 - p),$$

$$\frac{\partial z}{\partial w} = x, \quad \frac{\partial z}{\partial b} = 1.$$

Therefore,

$$\boxed{\frac{\partial L}{\partial w} = 2(p - y)p(1 - p)x}$$

and

$$\boxed{\frac{\partial L}{\partial b} = 2(p - y)p(1 - p)}.$$

8.1 Numerical Example

Let

$$x = 2, \quad w = 0.5, \quad b = 0, \quad y = 1.$$

Forward pass:

$$z = wx + b = 0.5(2) = 1,$$

$$p = \sigma(1) \approx 0.731,$$

$$L = (0.731 - 1)^2 \approx 0.072.$$

Backward pass:

$$\frac{\partial L}{\partial p} = 2(0.731 - 1) \approx -0.538,$$

$$\frac{\partial p}{\partial z} = 0.731(1 - 0.731) \approx 0.197.$$

Thus,

$$\frac{\partial L}{\partial z} \approx (-0.538)(0.197) \approx -0.106.$$

Finally,

$$\begin{aligned}\frac{\partial L}{\partial w} &= \frac{\partial L}{\partial z} x \approx -0.106(2) = -0.212, \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial z} \approx -0.106.\end{aligned}$$

With learning rate $\eta = 0.1$, gradient descent gives

$$\begin{aligned}w_{\text{new}} &= w - \eta \frac{\partial L}{\partial w} = 0.5 - 0.1(-0.212) = 0.5212, \\ b_{\text{new}} &= b - \eta \frac{\partial L}{\partial b} = 0 - 0.1(-0.106) = 0.0106.\end{aligned}$$

The negative gradients cause the parameters to increase in this example.

9 Backpropagation Through ReLU

Consider

$$z = wx + b, \quad h = \text{ReLU}(z), \quad L = h^2.$$

The derivatives are

$$\frac{\partial L}{\partial h} = 2h,$$

and

$$\frac{\partial h}{\partial z} = \begin{cases} 0, & z < 0, \\ 1, & z > 0. \end{cases}$$

Also,

$$\frac{\partial z}{\partial w} = x, \quad \frac{\partial z}{\partial b} = 1.$$

Therefore,

$$\begin{aligned}\frac{\partial L}{\partial w} &= 2h \text{ReLU}'(z)x, \\ \frac{\partial L}{\partial b} &= 2h \text{ReLU}'(z).\end{aligned}$$

Case 1: Let $z = 3$. Then $h = 3$ and $\text{ReLU}'(z) = 1$. The gradient can flow backward.

Case 2: Let $z = -2$. Then $h = 0$ and $\text{ReLU}'(z) = 0$. The gradient through this ReLU is zero.

This illustrates why a ReLU unit with a negative input does not pass a gradient backward through that unit.

10 The General Backpropagation Rule

Suppose a node v is calculated from earlier nodes:

$$v = f(u_1, u_2, \dots, u_n).$$

If the final loss is L , then

$$\boxed{\frac{\partial L}{\partial u_i} += \frac{\partial L}{\partial v} \frac{\partial v}{\partial u_i}}$$

The symbol $+=$ means that we add this contribution to any gradient already accumulated at u_i .

This gives the basic algorithm:

1. Run the forward computation and save intermediate values.
2. Start at the final output or loss.
3. Calculate the local derivative at the current operation.
4. Multiply by the incoming gradient.
5. Pass the result to earlier nodes.
6. Add contributions when multiple paths reach the same node.

11 Why the Backward Pass Is Efficient

Suppose a neural network has thousands or millions of parameters. We want the derivative of one loss with respect to every parameter.

Computing each derivative independently would repeat much of the same work.

Backpropagation avoids this repetition by:

- calculating intermediate values once in the forward pass;
- calculating local derivatives during the backward pass;
- reusing gradients as they move through the graph;
- accumulating contributions at shared nodes.

For a scalar loss L and parameters $\theta_1, \dots, \theta_n$, the result is the gradient

$$\nabla_{\theta} L = \left[\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \dots, \frac{\partial L}{\partial \theta_n} \right].$$

Gradient descent then updates all parameters:

$$\boxed{\theta_i \leftarrow \theta_i - \eta \frac{\partial L}{\partial \theta_i}}$$

Backpropagation calculates the gradients. Gradient descent uses those gradients to update the parameters. They are related, but they are not the same operation.

12 From Scalars to Vectors and Tensors

So far, every variable has been a single number. Neural networks usually operate on vectors, matrices, and higher-dimensional tensors.

The same ideas still apply.

12.1 Vector Example: Elementwise Addition

Let

$$\mathbf{z} = \mathbf{x} + \mathbf{y},$$

where

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

Then

$$\mathbf{z} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}.$$

If a scalar loss is

$$L = z_1^2 + z_2^2,$$

then

$$\frac{\partial L}{\partial \mathbf{z}} = \begin{bmatrix} 2z_1 \\ 2z_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 12 \end{bmatrix}.$$

Because $\mathbf{z} = \mathbf{x} + \mathbf{y}$,

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{z}}, \quad \frac{\partial L}{\partial \mathbf{y}} = \frac{\partial L}{\partial \mathbf{z}}.$$

Thus,

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} = \begin{bmatrix} 8 \\ 12 \end{bmatrix}.$$

The scalar rule for addition applies element by element.

12.2 Vector Example: Dot Product

Let

$$z = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^n w_i x_i.$$

For example,

$$\mathbf{w} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}.$$

Then

$$z = 2(4) + 3(5) = 23.$$

The gradients are

$$\boxed{\frac{\partial z}{\partial \mathbf{w}} = \mathbf{x}}, \quad \boxed{\frac{\partial z}{\partial \mathbf{x}} = \mathbf{w}}.$$

If $L = z^2$, then

$$\frac{\partial L}{\partial z} = 2z = 46.$$

Using the chain rule,

$$\frac{\partial L}{\partial \mathbf{w}} = 46\mathbf{x} = \begin{bmatrix} 184 \\ 230 \end{bmatrix}.$$

12.3 Matrices and Tensors

A tensor is a generalization of a scalar, vector, or matrix:

- scalar: zero dimensions;
- vector: one dimension;
- matrix: two dimensions;
- tensor: potentially three or more dimensions.

For example, an image may be represented by a tensor with dimensions

$$\text{height} \times \text{width} \times \text{channels}.$$

In tensor computations:

- addition and multiplication may operate elementwise;
- matrix multiplication combines many scalar products;
- activation functions are often applied elementwise;
- gradients have shapes corresponding to the values being differentiated.

The chain rule still applies. The main difference is that derivatives may be vectors, matrices, or higher-dimensional objects.

A derivative of a vector with respect to another vector can be represented as a Jacobian matrix. For a scalar loss, the gradient with respect to a tensor has the same shape as that tensor.

Modern libraries avoid explicitly constructing large Jacobian matrices when they can compute the needed products more efficiently.

13 A Tiny Introduction to PyTorch Autograd

PyTorch provides automatic differentiation through its `autograd` system.

When a tensor has `requires_grad=True`, PyTorch tracks operations involving that tensor. It builds a computation graph behind the scenes.

Calling `backward()` calculates gradients by traversing the graph backward.

13.1 Scalar Example

```
import torch

x = torch.tensor(2.0, requires_grad=True)
y = torch.tensor(3.0, requires_grad=True)

z = x * y + x**2

z.backward()

print(z.item()) # 10.0
print(x.grad.item()) # 7.0
print(y.grad.item()) # 2.0
```

Why are the gradients 7 and 2?

The expression is

$$z = xy + x^2.$$

Therefore,

$$\frac{\partial z}{\partial x} = y + 2x = 3 + 4 = 7,$$

and

$$\frac{\partial z}{\partial y} = x = 2.$$

PyTorch calculated these derivatives automatically.

13.2 A Small Gradient Descent Example

The following example trains a single scalar parameter w to approximate the relationship $y = 2x$.

```
import torch

x = torch.tensor(3.0)
target = torch.tensor(6.0)

w = torch.tensor(0.0, requires_grad=True)
learning_rate = 0.1

for step in range(5):
    prediction = w * x
    loss = (prediction - target)**2

    loss.backward()

    with torch.no_grad():
        w -= learning_rate * w.grad

    w.grad.zero_()

    print(step, w.item(), loss.item())
```

Important details:

- `requires_grad=True` tells PyTorch to track operations involving w .
- `loss.backward()` computes the gradient.
- `w.grad` contains the derivative of the loss with respect to w .
- `torch.no_grad()` prevents the parameter update itself from being added to the computation graph.
- `w.grad.zero_()` clears the stored gradient before the next iteration.

PyTorch accumulates gradients by default. If `backward()` is called again without clearing the gradient, the new gradient is added to the old one.

13.3 Tensor Example

The same mechanism works with vectors:

```
import torch

x = torch.tensor([1.0, 2.0, 3.0], requires_grad=True)

y = x**2
loss = y.sum()

loss.backward()

print(y)
print(x.grad)
```

Here,

$$L = x_1^2 + x_2^2 + x_3^2.$$

Therefore,

$$\nabla_x L = \begin{bmatrix} 2x_1 \\ 2x_2 \\ 2x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}.$$

PyTorch returns a gradient tensor with the same shape as x .

14 Key Ideas to Remember

- A computation graph breaks a calculation into simple operations.
- The forward pass computes values from inputs to outputs.
- Each operation has local derivatives.
- The chain rule combines local derivatives to calculate derivatives of the final loss.

- Backpropagation applies the chain rule backward through the computation graph.
- When a variable affects the loss through multiple paths, gradient contributions are added.
- Automatic differentiation performs these calculations systematically; it is not numerical finite-difference approximation.
- PyTorch autograd builds and traverses computation graphs automatically.
- Backpropagation computes gradients; an optimizer such as gradient descent uses those gradients to update parameters.

15 Quick Practice

1. Let $z = x + y$. Calculate $\partial z / \partial x$ and $\partial z / \partial y$.
2. Let $z = xy$. Calculate both partial derivatives. Evaluate them at $x = 2, y = 5$.
3. Let $z = (x + 2)^2$. Use the chain rule to calculate dz/dx . Evaluate at $x = 3$.
4. Let $p = \sigma(z)$. Write the derivative dp/dz in terms of p .
5. Let $h = \text{ReLU}(z)$. What is dh/dz when $z = -1$? What is it when $z = 4$?
6. Let $a = x + y$, $b = aw$, and $L = b^2$. Draw the computation graph and calculate $\partial L / \partial x$, $\partial L / \partial y$, and $\partial L / \partial w$.
7. In your own words, explain the difference between the forward pass, backpropagation, and gradient descent.