

A Short Guide to Gradients, Parameter Space, and Embeddings

Research by: Jose Portillo

1. From Derivatives to Gradients

In one-variable calculus, a differentiable function $f : \mathbb{R} \rightarrow \mathbb{R}$ changes according to

$$f(x + h) \approx f(x) + f'(x)h,$$

where $f'(x)$ is the slope of the tangent line at x .

For a function of many variables $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the generalization of $f'(x)$ is the **gradient**, written $\nabla f(x)$, defined implicitly by the linear approximation

$$f(x + h) \approx f(x) + \nabla f(x) \cdot h,$$

for small $h \in \mathbb{R}^n$.

The gradient $\nabla f(x)$ is the unique vector whose dot product with any direction h gives the *first-order* rate of change of f along that direction. Its components are the partial derivatives:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right).$$

The **directional derivative** in the direction of a unit vector u is

$$D_u f(x) = \nabla f(x) \cdot u = \lim_{h \rightarrow 0} \frac{f(x + hu) - f(x)}{h}.$$

Thus, unlike the one-dimensional case, there is no single difference quotient representing the entire gradient. The gradient points in the direction of steepest ascent with respect to the standard Euclidean inner product, and its magnitude gives the corresponding rate of increase.

2. Gradients in Machine Learning

A neural network has a large collection of tunable numbers called **parameters**. Collectively these are denoted

$$\theta = (\theta_1, \dots, \theta_N) \in \mathbb{R}^N,$$

where N can be very large. Conceptually, all parameters can be flattened into a single vector.

Training uses a **loss function** $L(\theta)$, a scalar measuring prediction error. Backpropagation computes the gradient

$$\nabla_\theta L(\theta) \in \mathbb{R}^N,$$

which indicates the direction of steepest increase of the loss. Gradient descent updates are of the form

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta),$$

where η is the learning rate.

3. Embeddings: Vectors Representing Tokens

An **embedding** is a vector associated to a discrete token. If d is the embedding dimension, then each token t has a learned vector

$$e_t \in \mathbb{R}^d.$$

These vectors are stored in an embedding matrix, which is one part of the full parameter vector θ . Embeddings convert discrete symbols into continuous vectors so the rest of the model can operate using linear algebra.

Embeddings vs. the Parameter Vector

Although embeddings and the full parameter vector are both vectors, they differ in role:

Object	Symbol	Dimension	Role
Parameter vector	$\theta \in \mathbb{R}^N$	$N \gg 10^6$	Defines the model itself
Embedding of a token	$e_t \in \mathbb{R}^d$	$d \sim 10^2 - 10^4$	Represents semantic relations

4. Parameter Space vs. Embedding Space

4.1 Parameter Space

The parameter vector θ lives in a large space \mathbb{R}^N . Each point corresponds to a different model. The gradient $\nabla_\theta L$ indicates how an infinitesimal change of the model affects the loss.

This is the space in which optimization occurs.

4.2 Embedding Space

Embedding vectors live in a much smaller space \mathbb{R}^d . Distances and angles between embeddings often encode semantic similarity.

This is a *semantic representation space*, not primarily an optimization space, although embeddings are updated by gradients during training.

4.3 Distinct Roles of These Spaces

Although both spaces are Euclidean,

$$\theta \in \mathbb{R}^N, \quad e_t \in \mathbb{R}^d,$$

they represent different kinds of objects:

- \mathbb{R}^N : model (parameter) space.
- \mathbb{R}^d : semantic representation space.

Being Euclidean does not make them interchangeable.

Note. These are unfinished notes. An AI language model was used as a support tool for discussion, clarification of concepts, and stylistic refinement. All mathematical choices and final wording are the responsibility of the author.

Appendix A: Gradients as Linear Maps

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be differentiable at x . Then

$$f(x + h) = f(x) + \nabla f(x) \cdot h + o(\|h\|).$$

The gradient is the unique vector realizing this first-order approximation.

In machine learning,

$$L(\theta + h) \approx L(\theta) + \nabla_\theta L(\theta) \cdot h,$$

so the gradient determines how a small parameter update affects the loss.

Appendix B: Gradient Descent as a Discrete Dynamical System

Gradient descent updates

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta L(\theta_k)$$

define a discrete-time dynamical system in \mathbb{R}^N .

Embeddings receive updates of the same form within \mathbb{R}^d :

$$e_t \leftarrow e_t - \eta \frac{\partial L}{\partial e_t}.$$

Appendix C: Coordinates, Reparameterization, and Gradient Flow

C.1 What Is Reparameterization?

A **reparameterization** is a change of coordinates on parameter space. Instead of optimizing directly over θ , one introduces new parameters ϕ and a smooth map

$$\theta = \psi(\phi).$$

The function computed by the model is unchanged:

$$f_\theta(x) = f_{\psi(\phi)}(x).$$

Only the coordinates used to describe the same model differ.

C.2 Effect on Gradients

By the chain rule, gradients transform as

$$\nabla_\phi L = J_\psi(\phi)^\top \nabla_\theta L,$$

where J_ψ is the Jacobian matrix of ψ . Consequently:

- the loss function is unchanged;
- the represented model is unchanged;
- the *gradient direction and optimization trajectory* may change.

Thus, gradient descent is *coordinate-dependent*: the notion of “steepest descent” depends on the chosen parameterization.

C.3 What Reparameterization Is Not

Reparameterization is not a learned behavior of the model, nor a form of self-adaptation. It is fixed by the model designer before training begins and affects only how the optimizer traverses parameter space.

C.4 Geometric Interpretation

Standard gradient descent assumes the Euclidean inner product on parameter space. Changing coordinates implicitly changes how distances and angles are measured by the optimizer. More advanced methods (e.g. natural gradient) explicitly modify this geometry, but the basic loss function and model class remain the same.

Appendix D: Probability, Data, and Learning

Up to this point, neural networks have been described using linear algebra and calculus: vectors, matrices, gradients, and optimization in parameter space. All of these objects are deterministic.

Probability enters not by replacing these tools, but by explaining *what the model is trying to fit* and *why the loss functions make sense*.

D.1 Inputs, Outputs, and Data

Machine learning does not study abstract functions in isolation. It studies functions fitted to observed data.

Each data point is a pair:

$$(x, y)$$

- x is the **input** (for example: a vector, an image, a sentence).
- y is the **output** we want the model to produce (for example: a number, a label, or the next word).

Training data consists of many such pairs:

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n).$$

D.2 The Data-Generating Process

We assume that data is produced by some unknown process:

$$(x, y) \sim \mathcal{D}$$

Here \mathcal{D} denotes an **unknown data-generating distribution**. This does not mean that we know \mathcal{D} explicitly. It means only that the world produces data according to some rule we do not control. We never see \mathcal{D} directly. We only observe samples drawn from it.

D.3 Loss Functions and Uncertainty

A neural network defines a function:

$$f_\theta(x),$$

where θ is the parameter vector introduced earlier.

The loss function:

$$L(\theta; x, y)$$

measures how incorrect the model is on a single data point.

Many standard loss functions reflect uncertainty in the data.

Predicting Numbers

Suppose the output differs from the prediction by some error:

$$y = f_\theta(x) + \varepsilon.$$

If the error varies from example to example, minimizing

$$L(\theta) = (y - f_\theta(x))^2$$

is a reasonable objective.

This is the **mean squared error (MSE)** loss.

Predicting Categories

When the output is a category, the model produces scores that are interpreted as probabilities:

$$p_\theta(y | x).$$

The standard loss is:

$$L(\theta) = -\log p_\theta(y | x),$$

called **cross-entropy loss**.

D.4 Stochastic Gradient Descent

The ideal objective is an average over all possible data:

$$\mathbb{E}_{(x,y) \sim \mathcal{D}}[L(\theta; x, y)].$$

Since \mathcal{D} is unknown, this expectation is approximated using random minibatches of data.

As a result:

- gradient steps are noisy,
- training follows a stochastic path,
- optimization becomes stochastic without changing the gradient formula.

D.5 Embeddings and Probability

Embeddings are vectors:

$$e_t \in \mathbb{R}^d.$$

They are learned by encouraging the model to assign high probability to tokens that co-occur in data:

$$p(\text{context} | \text{token}).$$

Thus, embedding geometry reflects statistical structure.

D.6 A Laplace Perspective

Laplace imagined an intellect that could predict everything if all parameters were known. Neural networks operate under incomplete information. Probability appears not because the system is mysterious, but because our knowledge is limited.

D.7 Summary

- Gradients are deterministic.
- Probability enters through data and objectives.
- Geometry encodes statistical regularities.

Appendix E: Softmax, Exponentials, and the Geometry of Cross-Entropy

This appendix explains why classification models use exponentials, what the softmax function is, and how cross-entropy has a geometric meaning.

No prior knowledge of exponentials or probability is assumed.

E.1 What Is an Exponential?

The exponential function is written:

$$\exp(x) = e^x,$$

where $e \approx 2.718$ is a mathematical constant.

Key properties:

- $\exp(x) > 0$ for all x ,
- larger x produces much larger values,
- differences are amplified, not just preserved.

Exponentials turn additive differences into multiplicative contrasts.

E.2 From Scores to Probabilities

In classification, a model produces a vector of scores:

$$z = (z_1, z_2, \dots, z_K).$$

These scores can be any real numbers. They are not probabilities.

To convert them into probabilities, we use the **softmax** function:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)}.$$

Properties:

- $p_i \geq 0$,
- $\sum_i p_i = 1$,
- larger scores get disproportionately larger probabilities.

E.3 Why Exponentials Are Used

Exponentials ensure that:

- relative differences matter, not absolute values,
- probabilities are smooth functions of the scores,
- gradients are well-behaved for optimization.

Adding the same constant to all scores does not change probabilities:

$$\text{softmax}(z) = \text{softmax}(z + c).$$

This makes softmax compatible with dot products and embeddings.

E.4 Cross-Entropy as Geometry

Suppose the correct class is k . The loss is:

$$L = -\log p_k.$$

Geometric interpretation:

- embeddings and weights produce dot products (scores),
- softmax converts scores into probabilities,
- cross-entropy penalizes small alignment with the correct direction.

Minimizing cross-entropy means:

Increase the dot product between the input representation and the correct class, while decreasing it for others.

E.5 Connection to Gradients

The gradient of cross-entropy with softmax has a simple form. It pushes:

- the correct class score up,
- the incorrect class scores down.

Thus, probability, geometry, and gradients act together.

E.6 Summary

- Exponentials amplify differences.
- Softmax converts scores into probabilities.
- Cross-entropy measures geometric misalignment.
- Gradients reshape the space accordingly.