AI Research Labs

Section 6 of 11 · 4 lessons

Loss Functions and Output Layers

How training minimises a loss, how MSE, MAE and cross-entropy behave, and how to pair output layers with losses.

6.1 Training as Minimising a Loss

6.2 MSE, MAE and Asymmetric Losses

6.3 Softmax and Cross-entropy

6.4 Binary, Multi-class and Logits in PyTorch



Free course with quizzes, progress tracking and Scoopy, an AI tutor:
airesearchlabs.si/learn/dl

Deep Learning Foundations · Edition November 2026 · © 2026 AI Research Labs. Free for personal study.

Where this section fits

Every lesson is a step in building a model. These are the steps this section teaches; the Big picture PDF and the online map show how they connect to the rest.

5

BUILDING AND TRAINING A NETWORK · STEP 5

Choose the output and loss

How do I measure how wrong the network is?

Lessons here: 6.1, 6.2, 6.3, 6.4 · feeds into Compute the gradients

Training as Minimising a Loss

Training has two steps: choose a loss that measures how wrong the model is, then minimise it. Worked through for a straight line with mean squared error: the gradients, one update, and the loss going down.

Two decisions

1. **Choose a loss** $J(\theta)$ that measures how badly the model with parameters θ does on the data.
2. **Minimise it**, usually with gradient descent: $\theta \leftarrow \theta - \eta \nabla J(\theta)$.

Mean squared error for a line

For $\hat{y}^{(i)} = mx^{(i)} + c$ and N examples:

$$J(m, c) = \frac{1}{N} \sum_{i=1}^{N} \left(mx^{(i)} + c - y^{(i)}\right)^2$$

Differentiate with the chain rule:

$$\frac{\partial J}{\partial m} = -\frac{2}{N} \sum_i x^{(i)} \left(y^{(i)} - mx^{(i)} - c\right), \qquad \frac{\partial J}{\partial c} = -\frac{2}{N} \sum_i \left(y^{(i)} - mx^{(i)} - c\right)$$

WORKED EXAMPLE: ONE GRADIENT STEP

Data $(1, 2), (2, 4), (3, 5)$. Start at $m = 1$, $c = 0$; residuals $y - mx - c$ are $1, 2, 2$.

- $J = (1 + 4 + 4)/3 = 3$.
- $\partial J/\partial m = -\frac{2}{3}(1 \cdot 1 + 2 \cdot 2 + 3 \cdot 2) = -\frac{22}{3} \approx -7.33$.
- $\partial J/\partial c = -\frac{2}{3}(1 + 2 + 2) = -\frac{10}{3} \approx -3.33$.
- With $\eta = 0.05$: $m = 1 + 0.367 = 1.367$, $c = 0 + 0.167 = 0.167$.
- New predictions $1.53, 2.90, 4.27$; new $J \approx 0.66$, down from 3.

Neural networks do exactly this, with millions of parameters and backpropagation (Section 7) to get the gradients.

Common doubts

FIRST PASS **Why are the gradients negative when the line is too low?**

The residuals $y - \hat{y}$ are positive (the line is below the points), so $\partial J/\partial m$ and $\partial J/\partial c$ come out negative. Gradient descent subtracts them, so m and c **increase** and the line moves up towards the data, which is what we want.

FIRST PASS **What is the difference between a loss function and an objective function?**

Strictly:

- **Loss** $\ell(\hat{y}_i, y_i)$: the error on **one** example.
- **Cost**, or empirical risk: the average loss over the training set, $\frac{1}{n}\sum_i \ell(\hat{y}_i, y_i)$.
- **Objective**: whatever training actually minimises. That is the cost, plus a penalty if you regularise: $J(\theta) = \frac{1}{n}\sum_i \ell + \lambda\,\Omega(\theta)$.

Books and frameworks often use the words loosely (PyTorch calls everything a "loss"), and this lesson calls J the loss. Here J is the average squared error with no penalty, so it is both the cost and the objective. In an exam, define the terms you use.

GOING DEEPER **Does the objective function represent the error?**

Mostly. With no penalty, the objective is the average error on the **training** data. Two caveats:

- With a regularisation penalty, it is error plus a cost for large weights, so a lower objective doesn't always mean a lower training error.
- It is training error, not the error on new data that you actually care about. That is why a validation set is kept aside (Section 8).

Cheatsheet

MSE gradients for a line

$J = \frac{1}{N}\sum(mx + c - y)^2$; $\partial_m J = -\frac{2}{N}\sum x(y - mx - c)$; $\partial_c J = -\frac{2}{N}\sum(y - mx - c)$.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/losses-and-outputs/training-as-minimising-a-loss

MSE, MAE and Asymmetric Losses

Regression losses compared: MSE punishes big errors hard and estimates the mean; MAE is robust to outliers and estimates the median; pinball losses price over- and under-prediction differently.

The regression losses

LOSS	FORMULA (AVERAGE OVER N)	PROPERTY
MSE (L2)	$\frac{1}{N}\sum(\hat{y}_i - y_i)^2$	Estimates the **mean**; large errors dominate
MAE (L1)	$\frac{1}{N}\sum\lvert\hat{y}_i - y_i\rvert$	Estimates the **median**; robust to outliers
Pinball (asymmetric)	αr if $r \geq 0$, $-\beta r$ if $r < 0$, with $r = \hat{y} - y$	Different prices for over- and under-prediction

Why MAE resists outliers



The loss for one prediction.



Its derivative: MSE's grows with the error.

MSE's gradient is proportional to the error, so one huge residual can dominate the whole update. MAE's gradient has the same size (±1) however big the error, so a few outliers have limited pull. MAE isn't differentiable at exactly zero; a valid **subgradient** (any value in $[-1, 1]$, usually 0) is used there.

WORKED EXAMPLE: MEAN VS MEDIAN

Targets $1, 2, 3, 4, 100$. The single number that minimises MSE is the mean, 22. The one that minimises MAE is the median, 3. One outlier drags the MSE answer far from the typical value.

Asymmetric losses

When over-predicting and under-predicting cost different amounts (running out of stock versus holding extra), the pinball loss weights them differently: with $\alpha = 1$ and $\beta = 3$, under-predicting costs three times as much as over-predicting, so the model learns to aim high.

For any regression loss, use a **linear output**: a sigmoid would restrict the predictions to (0, 1).

Common doubts

GOING DEEPER **If MAE is more robust, why is MSE so common?**

MSE is smooth everywhere, its gradient shrinks near the optimum (which helps convergence), and it matches the assumption of Gaussian noise. MAE is preferable when the data has outliers you don't want to chase. Many practitioners use a hybrid (the Huber loss) that is quadratic near zero and linear far away.

Cheatsheet

MSE vs MAE

MSE: estimates the mean; gradient $2e$, outliers dominate. MAE: estimates the median; gradient ± 1, robust; subgradient at 0.

Pinball loss

$r = \hat{y} - y$: αr if $r \geq 0$, $-\beta r$ if $r < 0$. Different costs for over- and under-prediction.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/losses-and-outputs/mse-mae-and-asymmetric-losses

Softmax and Cross-entropy

Softmax turns class scores into probabilities; cross-entropy scores those probabilities. A worked example, the numerical-stability trick, and the derivation of the famously simple gradient: predicted minus target.

Softmax: scores to probabilities

The last layer of a K-class classifier produces K raw scores s_j, called **logits**. Softmax turns them into probabilities:

$$o_j = \frac{e^{s_j}}{\sum_{k=1}^{K} e^{s_k}}$$

Every o_j is positive and they sum to 1. The target z is **one-hot**: 1 for the correct class, 0 for the rest.

Cross-entropy

$$H = -\sum_j z_j \log o_j$$

With a one-hot target only one term survives: $H = -\log(\text{probability given to the correct class})$.

WORKED EXAMPLE

Logits $s = (2, 1, 0)$ for classes A, B, C; the correct class is A.

- $e^2 = 7.389$, $e^1 = 2.718$, $e^0 = 1$; sum $= 11.107$.
- Softmax: $o = (0.665,\ 0.245,\ 0.090)$.
- Loss: $H = -\ln 0.665 \approx 0.408$.
- Gradient (below): $o - z = (-0.335,\ 0.245,\ 0.090)$. It pushes A's score up and the others down.

For comparison, a probability of 0.5 on the correct class costs $-\ln 0.5 \approx 0.693$, 0.95 costs 0.051, and 0.01 costs 4.6. Confident mistakes are punished hard.

Deriving the gradient

First the softmax derivative. For $j = k$: $\partial o_j / \partial s_j = o_j(1 - o_j)$; for $j \neq k$: $\partial o_j / \partial s_k = -o_j o_k$. Both cases: $\partial o_j / \partial s_k = o_j(\delta_{jk} - o_k)$. Then:

$$\frac{\partial H}{\partial s_k} = -\sum_j \frac{z_j}{o_j} o_j(\delta_{jk} - o_k) = -z_k + o_k \sum_j z_j = o_k - z_k$$

using $\sum_j z_j = 1$. So

$$\boxed{\frac{\partial H}{\partial s_j} = o_j - z_j}$$

Predicted probability minus target. The softmax derivative is already folded in; there is no extra activation-derivative factor. Sigmoid with binary cross-entropy gives the same result, $\hat{y} - y$.

Numerical stability

e^s overflows for large logits. Subtract the largest logit first: $o_j = e^{s_j - \max s} / \sum_k e^{s_k - \max s}$. The common factor $e^{-\max s}$ cancels, so the probabilities are unchanged: $(2, 1, 0)$ and $(0, -1, -2)$ give the same softmax.

Common doubts

FIRST PASS **Why is cross-entropy better than MSE for classification?**

Two reasons. It punishes confident wrong answers heavily ($-\ln 0.01 \approx 4.6$), and with a softmax or sigmoid output its gradient is simply $o - z$, so it doesn't vanish when the output unit saturates. MSE through a sigmoid has an extra $\hat{y}(1 - \hat{y})$ factor that goes to zero exactly when the model is confidently wrong.

GOING DEEPER **Where did the $1/o_j$ go in the derivation?**

$\partial(-z_j \log o_j)/\partial o_j = -z_j/o_j$, and the softmax derivative contributes a factor o_j, so they cancel. That cancellation is why the softmax–cross-entropy pair is so convenient, and why frameworks compute them together.

Cheatsheet

Softmax and cross-entropy

$o_j = e^{s_j} / \sum_k e^{s_k}$; $H = -\sum_j z_j \log o_j$; gradient $\partial H / \partial s_j = o_j - z_j$.

Stable softmax

Subtract $\max_k s_k$ from every logit before exponentiating. Same probabilities, no overflow.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/losses-and-outputs/softmax-and-cross-entropy

Binary, Multi-class and Logits in PyTorch

Which output layer and loss go with which task, binary cross-entropy worked out, why accuracy can't be the training loss, the best accuracy any classifier can reach, and the logits rule that trips people up in PyTorch.

Output layer and loss by task

TASK	OUTPUT LAYER	LOSS	PYTORCH
Regression	Linear	MSE	`nn.MSELoss`
Binary classification	One sigmoid unit	Binary cross-entropy	`nn.BCEWithLogitsLoss`
Multi-class	Softmax over K classes	Categorical cross-entropy	`nn.CrossEntropyLoss`

Binary cross-entropy

$$\ell(\hat{y}, y) = -\big[y \log \hat{y} + (1-y)\log(1-\hat{y})\big], \qquad \hat{y} = \sigma(z) = P(y=1 \mid x)$$

For $y=1$ it's $-\log \hat{y}$; for $y=0$, $-\log(1-\hat{y})$. With $y=1$: $\hat{y}=0.9$ costs 0.105 and $\hat{y}=0.1$ costs 2.303. A threshold (0.5 by default) turns $\hat{y}$ into a class.

The logits rule in PyTorch

RETURN LOGITS, NOT PROBABILITIES

`nn.CrossEntropyLoss` and `nn.BCEWithLogitsLoss` take **raw logits** and apply log-softmax or sigmoid internally, in a numerically stable way. If your model also applies a softmax at the end, the loss applies it a second time: training still runs but the gradients are wrong and learning is poor. Leave the final layer linear; apply softmax only when you want probabilities for reporting.

Why not train on accuracy?

Accuracy only changes when a prediction crosses the threshold, so as a function of the weights it's flat almost everywhere: its gradient is zero. Train on cross-entropy, which changes smoothly, and **report** accuracy.

The best any classifier can do

WORKED EXAMPLE: OVERLAPPING CLASSES

Class A is uniform on the square $[0, 5]^2$; class B uniform on $[0, 10]^2$; equal priors. Inside the small square the densities are $1/25$ (A) and $1/100$ (B), so $P(A \mid x) = \frac{1/25}{1/25+1/100} = 0.8$. Outside it, only B is possible.

- A point lands in the small square with probability $0.5 \cdot 1 + 0.5 \cdot 0.25 = 0.625$. The best guess there is A, right 80% of the time.
- Outside (probability 0.375) the best guess is B, right every time.
- Best possible accuracy: $0.625 \times 0.8 + 0.375 \times 1 = 0.875$.
- Lowest possible cross-entropy: $0.625 \times (-0.8 \ln 0.8 - 0.2 \ln 0.2) \approx 0.313$ nats.

No network, however large, can beat this **Bayes-optimal** limit, because the classes genuinely overlap.

Common doubts

FIRST PASS **Do I need a sigmoid layer before BCEWithLogitsLoss?**

No, and you shouldn't add one. As the name says, it expects logits and applies the sigmoid itself. If you want probabilities at prediction time, apply `torch.sigmoid` to the logits then.

GOING DEEPER **Why can't my model reach 100% accuracy on overlapping classes?**

If the same input can belong to either class, any rule that picks one class for that input must be wrong some of the time. The Bayes-optimal classifier picks the more probable class everywhere, and its accuracy (0.875 in the example) is a ceiling for every model on that data distribution.

Cheatsheet

Task → output → loss

Regression: linear + MSE. Binary: sigmoid + BCE (BCEWithLogitsLoss). Multi-class: softmax + CE (CrossEntropyLoss).

Logits rule

CrossEntropyLoss / BCEWithLogitsLoss take raw logits. Don't put softmax/sigmoid in the model when using them.

Accuracy as a loss

Accuracy is piecewise constant: zero gradient. Train on cross-entropy; report accuracy.

Online version with the interactive quiz: airesearchlabs.si/learn/dl/losses-and-outputs/binary-multiclass-and-logits