

Deep Learning Systems: Algorithms and Implementation

ML Refresher / Softmax Regression

Tim Dettmers (this time) and Tianqi Chen
Carnegie Mellon University

Outline

How to use mupgrade

Basics of machine learning

Example: softmax regression

Outline

How to use mupgrade

Basics of machine learning

Example: softmax regression

Welcome

Sign in with Google to view your courses.



Enroll through <https://mugrade.org/courses/1?enroll=664EqzbhsV5p99rbBYf0>
Sign in with your andrew Google account

Deep Learning Systems

10-714 · Fall 2025



You should see Deep Learning Systems in your courses!

[Back to Courses](#)

Deep Learning Systems

10-714 · Fall 2025

[Show API Key](#)

Assignments

ASSIGNMENT NAME	DUE DATE	LAST SUBMISSION	LATE DAYS	SCORE
hw0	9/11/2025, 11:59:00 PM	8/24/2025, 11:22:24 AM	0	75 / 100 ⚠

Click “Show API Key” to retrieve your API key.

This API key is used in HW Colab notebooks to submit your answers



Back to Courses

Deep Learning Systems

10-714 · Fall 2025

Show API Key

Assignments

ASSIGNMENT NAME	DUE DATE	LAST SUBMISSION	LATE DAYS	SCORE
hw0	9/11/2025, 11:59:00 PM	8/24/2025, 11:22:24 AM	0	75 / 100 ⚠



Click on a specific assignment to see details

[Back to Course](#)

hw0

Release: 8/24/2025, 11:59:00 PM

Due: 9/11/2025, 11:59:00 PM

Your Score: 100 / 100

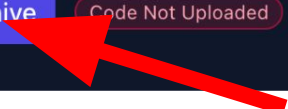
Problems

PROBLEM NAME	SUBMISSION DATE	SCORE
add	8/24/2025, 11:22:24 AM	10 / 10
parse_mnist	8/24/2025, 1:24:17 AM	10 / 10
softmax_loss	8/24/2025, 1:29:52 AM	10 / 10
softmax_regression_epoch	8/24/2025, 1:30:12 AM	20 / 20
nn_epoch	8/24/2025, 1:31:01 AM	25 / 25
softmax_regression_epoch_cpp	8/26/2025, 12:21:15 PM	25 / 25

After you have submitted all the problems, upload an archive with your code here. This is not used for autograding, but just as a reference for the instructors to check.

[Upload Code Archive](#)

Code Not Uploaded



Outline

How to use mupgrade

Basics of machine learning

Example: softmax regression

Machine learning as data-driven programming

Suppose you want to write a program that will classify handwritten drawing of digits into their appropriate category: 0,1,...,9

You *could*, think hard about the nature of digits, try to determine the logic of what indicates what kind of digit, and write a program to codify this logic

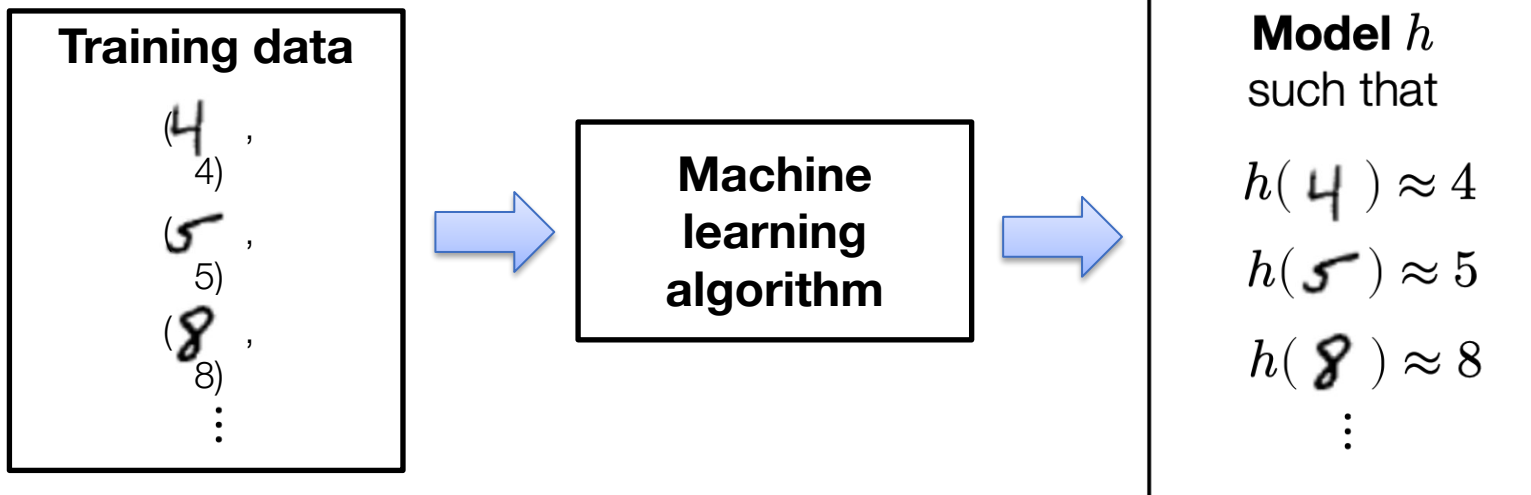
(Despite being a reasonable coder, I don't think I could do this very well)



MNIST Dataset

Machine learning as data-driven programming

The (supervised) ML approach: collect a *training set* of images with known labels and feed these into a *machine learning algorithm*, which will (if done well), automatically produce a “program” that solves this task



Three ingredients of a machine learning algorithm

Every machine learning algorithm consists of three different elements:

1. **The hypothesis class:** the “program structure”, parameterized via a set of *parameters*, that describes how we map inputs (e.g., images of digits) to outputs (e.g., class labels, or probabilities of different class labels)
2. **The loss function:** a function that specifies how “well” a given hypothesis (i.e., a choice of parameters) performs on the task of interest
3. **An optimization method:** a procedure for determining a set of parameters that (approximately) minimize the sum of losses over the training set

Outline

How to use mupgrade

Basics of machine learning

Example: softmax regression

Multi-class classification setting

Let's consider a *k-class classification setting*, where we have

- Training data: $x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \{1, \dots, k\}$ for $i = 1, \dots, m$
- n = dimensionality of the input data
- k = number of different classes / labels
- m = number of points in the training set

Example: classification of 28x28 MNIST digits

- $n = 28 \cdot 28 = 784$
- $k = 10$
- $m = 60,000$

Linear hypothesis function

Our hypothesis function maps inputs $x \in \mathbb{R}^n$ to k -dimensional vectors

$$h: \mathbb{R}^n \rightarrow \mathbb{R}^k$$

where $h_i(x)$ indicates some measure of “belief” in how much likely the label is to be class i (i.e., “most likely” prediction is coordinate i with largest $h_i(x)$).

A **linear hypothesis function** uses a *linear* operator (i.e. matrix multiplication) for this transformation

$$h_\theta(x) = \theta^T x$$

for *parameters* $\theta \in \mathbb{R}^{n \times k}$

Matrix batch notation

Often more convenient (and this is how you want to code things for efficiency) to write the data and operations in *matrix batch* form

$$X \in \mathbb{R}^{m \times n} = \begin{bmatrix} - & x^{(1)T} & - \\ & \vdots & \\ - & x^{(m)T} & - \end{bmatrix}, \quad y \in \{1, \dots, k\}^m = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Then the linear hypothesis applied to this batch can be written as

$$h_{\theta}(X) = \begin{bmatrix} - & h_{\theta}(x^{(1)})^T & - \\ & \vdots & \\ - & h_{\theta}(x^{(m)})^T & - \end{bmatrix} = \begin{bmatrix} - & x^{(1)T} \theta & - \\ & \vdots & \\ - & x^{(m)T} \theta & - \end{bmatrix} = X\theta$$

Loss function #1: classification error

The simplest loss function to use in classification is just the classification error, i.e., whether the classifier makes a mistake or not

$$\ell_{err}(h(x), y) = \begin{cases} 0 & \text{if } \operatorname{argmax}_i h_i(x) = y \\ 1 & \text{otherwise} \end{cases}$$

We typically use this loss function to assess the *quality* of classifiers

Unfortunately, the error is a bad loss function to use for *optimization*, i.e., selecting the best parameters, because it is not differentiable

Loss function #2: softmax / cross-entropy loss

Let's convert the hypothesis function to a "probability" by exponentiating and normalizing its entries (to make them all positive and sum to one)

$$z_i = p(\text{label} = i) = \frac{\exp(h_i(x))}{\sum_{j=1}^k \exp(h_j(x))} \iff z \equiv \text{softmax}(h(x))$$

Then let's define a loss to be the (negative) log probability of the true class: this is called *softmax* or *cross-entropy* loss

$$\ell_{ce}(h(x), y) = -\log p(\text{label} = y) = -h_y(x) + \log \sum_{j=1}^k \exp(h_j(x))$$

The softmax regression optimization problem

The third ingredient of a machine learning algorithm is a method for solving the associated optimization problem, i.e., the problem of minimizing the average loss on the training set

$$\underset{\theta}{\text{minimize}} \quad \frac{1}{m} \sum_{i=1}^m \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

For softmax regression (i.e., linear hypothesis class and softmax loss):

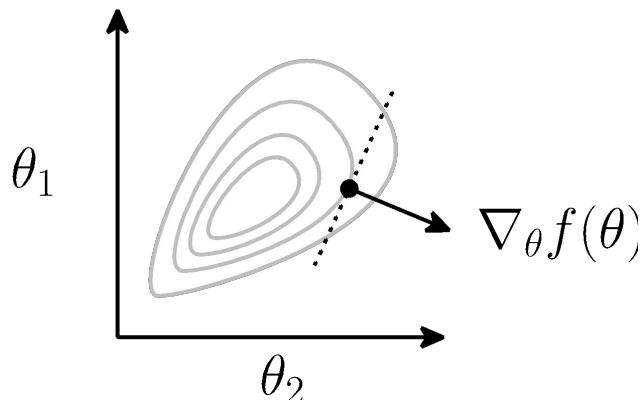
$$\underset{\theta}{\text{minimize}} \quad \frac{1}{m} \sum_{i=1}^m \ell_{ce}(\theta^T x^{(i)}, y^{(i)})$$

So how do we find θ that solves this optimization problem?

Optimization: gradient descent

For a matrix-input, scalar output function $f: \mathbb{R}^{n \times k} \rightarrow \mathbb{R}$, the *gradient* is defined as the matrix of partial derivatives

$$\nabla_{\theta} f(\theta) \in \mathbb{R}^{n \times k} = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_{11}} & \dots & \frac{\partial f(\theta)}{\partial \theta_{1k}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\theta)}{\partial \theta_{n1}} & \dots & \frac{\partial f(\theta)}{\partial \theta_{nk}} \end{bmatrix}$$



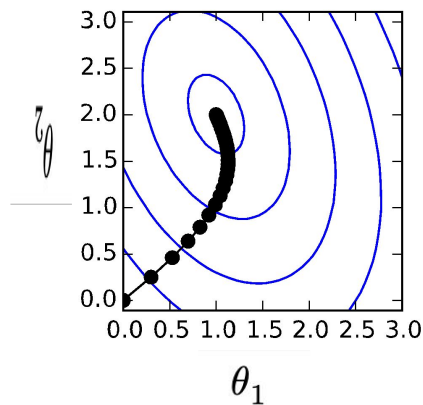
Gradient points in the direction that most *increases* f (locally)

Optimization: gradient descent

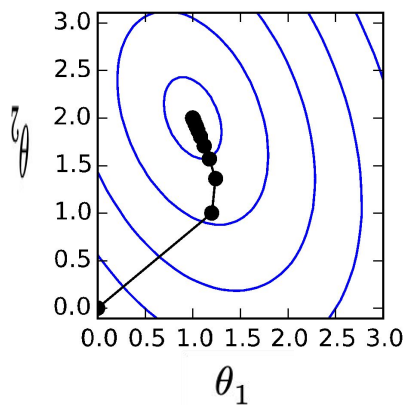
To *minimize* a function, the gradient descent algorithm proceeds by iteratively taking steps in the direction of the negative gradient

$$\theta := \theta - \alpha \nabla_{\theta} f(\theta)$$

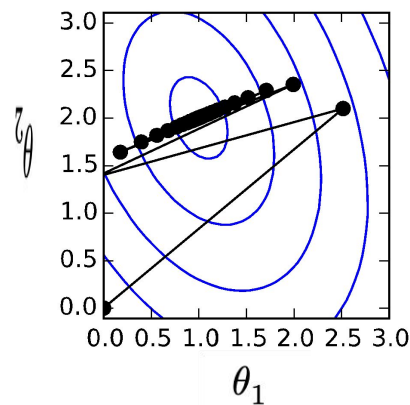
where $\alpha > 0$ is a *step size* or *learning rate*



$\alpha = 0.05$



$\alpha = 0.2$



$\alpha = 0.42$

Stochastic gradient descent

If our objective (as is the case in machine learning) is the *sum* of individual losses, we don't want to compute the gradient using all examples to make a single update to the parameters

Instead, take many gradient steps each based upon a *minibatch* (small partition of the data), to make many parameter updates using a single “pass” over data

Repeat:

Sample a minibatch of data $X \in \mathbb{R}^{B \times n}, y \in \{1, \dots, k\}^B$

Update parameters $\theta := \theta - \frac{\alpha}{B} \sum_{i=1}^B \nabla_{\theta} \ell(h_{\theta}(x^{(i)}), y^{(i)})$

The gradient of the softmax objective

So, how do we compute the gradient for the softmax objective?

$$\nabla_{\theta} \ell_{ce}(\theta^T x, y) = ?$$

Let's start by deriving the gradient of the softmax loss itself: for vector $h \in \mathbb{R}^k$

$$\begin{aligned} \frac{\partial \ell_{ce}(h, y)}{\partial h_i} &= \frac{\partial}{\partial h_i} \left(-h_y + \log \sum_{j=1}^k \exp h_j \right) \\ &= -1\{i = y\} + \frac{\exp h_i}{\sum_{j=1}^k \exp h_j} \end{aligned}$$

So, in vector form: $\nabla_h \ell_{ce}(h, y) = z - e_y$, where $z = \text{softmax}(h)$

The gradient of the softmax objective

So how do we compute the gradient $\nabla_{\theta} \ell_{ce}(\theta^T x, y)$?

- The chain rule of multivariate calculus ... but the dimensions of all the matrices and vectors get pretty cumbersome

Approach #1 (a.k.a. the right way): Use matrix differential calculus, Jacobians, Kronecker products, and vectorization

Approach #2 (a.k.a. the hacky quick way that everyone actually does): Pretend everything is a scalar, use the typical chain rule, and then rearrange / transpose matrices/vectors to make the sizes work 🤖 (and check your answer numerically)

The slide I'm embarrassed to include...

Let's compute the “derivative” of the loss:

$$\begin{aligned}\frac{\partial}{\partial \theta} \ell_{ce}(\theta^T x, y) &= \frac{\partial \ell_{ce}(\theta^T x, y)}{\partial \theta^T x} \frac{\partial \theta^T x}{\partial \theta} \\ &= (\underbrace{z - e_y}_{(k\text{-dimensional})})(\underbrace{x}_{(n\text{-dimensional})}), \quad \text{where } z = \text{softmax}(\theta^T x)\end{aligned}$$

So to make the dimensions work...

$$\nabla_{\theta} \ell_{ce}(\theta^T x, y) \in \mathbb{R}^{n \times k} = x(z - e_y)^T$$

Same process works if we use “matrix batch” form of the loss

$$\nabla_{\theta} \ell_{ce}(X\theta, y) \in \mathbb{R}^{n \times k} = X^T (Z - I_y), \quad Z = \text{softmax}(X\theta)$$

Putting it all together

Despite a fairly complex derivation, we should highly just how *simple* the final algorithm is

- Repeat until parameters / loss converges
 1. Iterate over minibatches $X \in \mathbb{R}^{B \times n}$, $y \in \{1, \dots, k\}^B$ of training set
 2. Update the parameters $\theta := \theta - \frac{\alpha}{B} X^T (Z - I_y)$

That is the entirety of the softmax regression algorithm

As you will see on the homework, this gets less than 8% error in classifying MNIST digits, runs in a couple seconds

Up next time: neural networks (a.k.a. fancier hypothesis classes)