

Machine Learning

Textbook: Machine Learning Refined, by Watt, Borhani, and Katsaggelos, 2nd edition:
MLRefined.com

Previous Lecture

- ◆ Softmax transforms K real-valued scores into positive class probabilities that sum to one.
- ◆ Softmax preserves the score ordering, so $\arg \max_k p_k = \arg \max_k f_k$.
- ◆ Adding the same constant to every score leaves the softmax probabilities unchanged and enables stable computation.
- ◆ Categorical cross-entropy is $-\sum_k y_k \log p_k$, which reduces to $-\log p_y$ for a one-hot label.
- ◆ Softmax plus categorical cross-entropy gives the simple score gradient $\nabla_{\mathbf{f}} \ell = \mathbf{p} - \mathbf{y}$.
- ◆ Binary logistic regression is the $K = 2$ special case of softmax regression.



§7.8 - Stochastic and Mini-Batch Optimization

So far:

when computing a gradient,
we usually used every training observation.

That is reasonable for a small dataset.

But what if

$$P = 10^6$$

or

$$P = 10^9?$$

Computing the exact full gradient

before every parameter update can be expensive.

Today:

use only part of the data at each update.

Machine-Learning Costs Are Often Finite Sums

For many supervised-learning problems:

$$g(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P g_p(\mathbf{w}).$$

Here

$$g_p(\mathbf{w})$$

is the loss contributed by observation p .

Examples:

squared-error loss;

logistic loss;

categorical cross-entropy;

hinge loss.

The total training objective is an average of observation-level losses.

The Full Gradient

Differentiate the finite sum:

$$\nabla g(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P \nabla g_p(\mathbf{w}).$$

Full-batch gradient descent uses

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla g(\mathbf{w}_k).$$

Every update requires contributions from all P observations.

Advantage:

exact gradient of the training objective.

Disadvantage:

potentially expensive update.

Three Choices

At one parameter update we may use:

Method	Observations used in one gradient estimate
Full batch	all P
Stochastic	one observation
Mini-batch	b observations

where

$$1 < b < P$$

for a genuine mini-batch.

These methods do not change the model.

They change how we estimate the gradient.

Stochastic Gradient Descent

Choose an observation index

$$i_k.$$

Use its gradient:

$$\nabla g_{i_k}(\mathbf{w}_k).$$

Update:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla g_{i_k}(\mathbf{w}_k).$$

This is **stochastic gradient descent**,
or **SGD**.

One update can be much cheaper
than a full-gradient update.

But:
the direction is noisy.

Why “Stochastic”?

Suppose
 i_k
is chosen randomly.

Then the gradient used at step k
is a random vector.

Different selected observations
can point in different directions.

The update path therefore fluctuates.

Unlike full-batch gradient descent:
the cost need not decrease at every step.

The algorithm trades
exactness per update
for cheap frequent updates.

The Stochastic Gradient Is Unbiased

Suppose an index
 I
is selected uniformly from
 $\{1, \dots, P\}$.

Then
 $P(I = p) = \frac{1}{P}$.

Therefore
$$\begin{aligned} E[\nabla g_I(\mathbf{w})] &= \sum_{p=1}^P P(I = p) \nabla g_p(\mathbf{w}) \\ &= \frac{1}{P} \sum_{p=1}^P \nabla g_p(\mathbf{w}). \end{aligned}$$

Thus
 $E[\nabla g_I(\mathbf{w})] = \nabla g(\mathbf{w})$.

The stochastic gradient is noisy,
but correct on average.

A Simple Quadratic Example

Consider observations

$$y = (1, 3, 5, 7).$$

Fit the constant model

$$\hat{y} = w.$$

Use observation loss

$$g_p(w) = \frac{1}{2}(w - y_p)^2.$$

Then

$$g(w) = \frac{1}{4} \sum_{p=1}^4 \frac{1}{2}(w - y_p)^2.$$

Individual gradient:

$$g'_p(w) = w - y_p.$$

The Full Gradient in the Example

Average the four individual gradients:

$$\begin{aligned} g'(w) &= \frac{1}{4}[(w - 1) + (w - 3) + (w - 5) + (w - 7)] \\ &= \frac{1}{4}(4w - 16) \\ &= w - 4. \end{aligned}$$

Therefore the minimizer is

$$w^* = 4.$$

As expected:

least squares for a constant model chooses the mean.

Full Gradient Versus Stochastic Gradients

At

$$w = 0:$$

full gradient:

$$g'(0) = -4.$$

Individual gradients:

y_p	1	3	5	7
$g'_p(0)$	-1	-3	-5	-7

Their average:

$$\begin{aligned} &\frac{-1-3-5-7}{4} \\ &= -4. \end{aligned}$$

Each observation gives a different estimate.

The average gives the exact gradient.

One Step Can Differ Substantially

Take

$$\alpha = 0.1$$

and

$$w_0 = 0.$$

Full-batch step:

$$\begin{aligned} w_1 &= 0 - 0.1(-4) \\ &= 0.4. \end{aligned}$$

If SGD happens to select

$$\begin{aligned} y &= 1: \\ w_1 &= 0 - 0.1(-1) \\ &= 0.1. \end{aligned}$$

If SGD selects

$$\begin{aligned} y &= 7: \\ w_1 &= 0 - 0.1(-7) \\ &= 0.7. \end{aligned}$$

Different random observations
produce different trajectories.

Even at the Exact Minimizer, Individual Gradients Need Not Vanish

At

$$w^* = 4:$$

full gradient:

$$g'(4) = 0.$$

But individual gradients are

y_p	1	3	5	7
$g'_p(4)$	3	1	-1	-3

So SGD with a fixed step size
can keep moving
even after reaching the full-batch minimizer.

This is one source of persistent stochastic jitter.

A Stochastic Step Can Increase the Full Cost

At

$$w = 4,$$

suppose SGD selects

$$y = 1$$

with

$$\alpha = 0.1.$$

Then

$$\begin{aligned} w_{\text{new}} &= 4 - 0.1(3) \\ &= 3.7. \end{aligned}$$

But $w = 4$ minimizes the full objective.

Therefore

$$g(3.7) > g(4).$$

Indeed:

$$g(4) = 2.5,$$

while

$$g(3.7) \approx 2.545.$$

SGD does not promise descent at every individual update.

Mini-Batch Gradients

Instead of one observation,

choose a set

$$B_k$$

containing

$$b$$

observations.

Define the mini-batch gradient

$$\nabla g_{B_k}(\mathbf{w}) = \frac{1}{b} \sum_{p \in B_k} \nabla g_p(\mathbf{w}).$$

Update:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla g_{B_k}(\mathbf{w}_k).$$

Special cases:

$$b = 1$$

gives SGD.

$$b = P$$

gives full-batch gradient descent.

Mini-Batches Average Away Some Noise

Return to

$$w = 4.$$

Individual gradients:

3, 1, -1, -3.

Batch

$$\{y = 1, y = 3\}:$$

average gradient:

$$\frac{3+1}{2}$$
$$= 2.$$

Batch

$$\{y = 5, y = 7\}:$$

average gradient:

$$\frac{-1-3}{2}$$
$$= -2.$$

Batch

$$\{y = 1, y = 7\}:$$

average gradient:

$$0.$$

Mini-batches still fluctuate,
but averaging usually reduces the fluctuation.

Mini-Batch Gradients Are Also Unbiased

Suppose observations are sampled uniformly
into a batch.

The batch gradient is an average
of unbiased observation-level gradient estimates.

Therefore

$$E[\nabla g_B(\mathbf{w})] = \nabla g(\mathbf{w})$$

under standard random-sampling schemes.

Increasing b generally reduces the variance
of the gradient estimate.

So batch size controls a continuum:

noisy / cheap updates

$\leftrightarrow$

stable / expensive updates.

Batch Size Tradeoff

Small batch:

cheaper individual updates;

more updates per pass through the data;

noisier gradient estimates.

Large batch:

more work per update;

fewer updates per pass;

less gradient noise.

Hardware also matters.

Vectorized matrix operations can make
moderate-size mini-batches computationally efficient.

This is one reason mini-batches dominate modern practice.

What Is an Epoch?

One **epoch**

means approximately one complete pass
through the training dataset.

If

$$P$$

observations are split into batches of size

$$b,$$

then the number of updates per epoch is approximately

$$\left\lceil \frac{P}{b} \right\rceil.$$

Example:

$$P = 1000,$$

$$b = 100.$$

Updates per epoch:

$$10.$$

With

$$b = 1:$$

$$1000$$

updates per epoch.

The Last Batch May Be Smaller

Suppose

$$P = 103$$

and

$$b = 20.$$

Five full batches use
100
observations.

The final batch contains
3.

Therefore:
 $\lceil 103/20 \rceil$
= 6
updates per epoch.

Software may:
keep the smaller final batch,
or occasionally discard it.

The choice should be explicit.

Why Shuffle?

Suppose data are stored in an ordered way:
all class 1 observations first,

then all class 2 observations,

then all class 3 observations.

Sequential batches could then be systematically biased.

A common procedure:

1. randomly shuffle observation order;
2. partition into mini-batches;
3. process every batch;
4. reshuffle for the next epoch.

Shuffling helps batches resemble
random samples from the full training set.

Sampling With Replacement Versus Shuffled Epochs

Two common stochastic schemes:

With replacement:
choose a random observation independently at each step.

An observation may appear repeatedly

before another is selected.

Without replacement:

shuffle the dataset,
then process each observation once during the epoch.

Theoretical analyses often begin with independent sampling.

Practical training commonly uses shuffled epochs.

Learning Rate Matters Even More With Noisy Gradients

Full-batch gradient descent:

gradient noise is absent.

SGD / mini-batch descent:

gradient estimates fluctuate.

With a fixed learning rate:

the iterates may continue wandering
around a minimizer.

Reducing the learning rate over time

can reduce this late-stage jitter.

Thus one often uses

$$\alpha_k \downarrow 0.$$

A Classical Decreasing-Step Condition

In classical stochastic approximation,

a common theoretical condition is

$$\sum_{k=0}^{\infty} \alpha_k = \infty$$

and

$$\sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

Interpretation:

steps remain large enough in total
to keep making progress,

but squared step sizes become small enough

to control accumulated noise.

Example:

$$\alpha_k = \frac{1}{k+1}.$$

Then

$$\sum \alpha_k$$

diverges,

while

$$\sum \alpha_k^2$$

converges.

These are theoretical sufficient conditions
in particular stochastic settings,
not a universal practical prescription.

Constant Learning Rate Versus Decay

Constant learning rate:

- often moves rapidly;

- maintains stochastic variation;

- may hover near a minimizer.

Decreasing learning rate:

- large early steps;

- smaller late steps;

- can settle more tightly.

Modern optimization also uses

- adaptive learning-rate methods.

But the basic tradeoff begins here:

- speed versus late-stage stability.

Noise Is Not Necessarily Bad

A stochastic gradient is less accurate
than the full gradient.

But noisy updates can still be useful.

They allow:

- frequent updates before a full pass through the data;

- online learning as observations arrive;

- movement through some flat or shallow regions.

However:

- randomness does not guarantee escape from every poor solution.

Noise is a computational and statistical feature,
not magic.

Cost Per Epoch Versus Cost Per Update

Suppose every observation is used once per epoch.

Then full batch,
SGD,
and mini-batch methods
all process roughly
 P
observations per epoch.

The difference is:
how often parameters are updated.

Full batch:
1 update per epoch.

SGD:
 P updates per epoch.

Mini-batch:
approximately P/b updates per epoch.

So “cheaper update”
is different from
“less data processed per epoch.”

Why Mini-Batches Work Well on Modern Hardware

Matrix operations are efficient.

A mini-batch allows us to process
many observations simultaneously.

For example:

- matrix multiplication on CPUs / GPUs;
- vectorized probability calculations;
- parallel gradient contributions.

Batch size therefore reflects both
optimization mathematics
and hardware efficiency.

The fastest algorithm in wall-clock time
need not use the fewest arithmetic operations.

Mini-Batch Logistic Regression

Binary cross-entropy gradient:

$$\nabla g = \frac{1}{P} X^T (\mathbf{p} - \mathbf{y}).$$

For a batch

B

of size

b :

$$\nabla g_B = \frac{1}{b} X_B^T (\mathbf{p}_B - \mathbf{y}_B).$$

Same formula.

Just use the rows belonging to the current batch.

No new model derivation is required.

Mini-Batch Softmax Regression

From Lesson 1.20:

$$\nabla_W g = \frac{1}{P} (\mathbf{P} - \mathbf{Y})^T \mathbf{X}.$$

For a mini-batch:

$$\nabla_W g_B = \frac{1}{b} (\mathbf{P}_B - \mathbf{Y}_B)^T \mathbf{X}_B.$$

Again:

same derivative,
smaller set of rows.

This pattern extends to much more complicated models.

Later neural networks will also be trained
using mini-batches.

Full-Batch Gradient Descent in Python

For the constant-model example:

```
y = np.array([1., 3., 5., 7.])

def full_grad(w):
    return np.mean(w - y)

w = 0.0
alpha = 0.1

for k in range(50):
    w = w - alpha*full_grad(w)
```

Every update computes
all four observation gradients.

The iterates move smoothly toward

$$w^* = 4.$$

SGD in Python

Use a shuffled pass through the data:

```
rng = np.random.default_rng(0)

w = 0.0
alpha = 0.1

for epoch in range(20):
    order = rng.permutation(len(y))

    for p in order:
        grad = w - y[p]
        w = w - alpha*grad
```

New Python ideas:

`np`
creates a random-number generator.

`rng`
returns a random ordering of the indices.

The order changes each epoch.

Mini-Batch Gradient Descent in Python

Use batches of size 2:

```
rng = np.random.default_rng(0)

w = 0.0
alpha = 0.1
batch_size = 2

for epoch in range(20):
    order = rng.permutation(len(y))

    for start in range(0, len(y), batch_size):
        idx = order[start:start + batch_size]

        grad = np.mean(w - y[idx])

        w = w - alpha*grad
```

The slice

`order[start:start + batch_size]`
selects the current mini-batch.

This code also works

if the final batch is smaller.

Mini-Batches for Matrix Data

Suppose the training data are

`X`
and
`y`.

For current batch indices

```
idx:
    Xb = X[idx]
    Yb = Y[idx]
```

Then run the same formulas

```
on
Xb
and
Yb.
```

For softmax regression:

```
scores = Xb @ W.T
P = softmax_rows(scores)

grad_W = (P - Yb).T @ Xb / len(Xb)

W = W - alpha*grad_W
```

The optimization method changes
by changing which rows participate in each update.

Do Not Judge SGD by One Update

Because a stochastic step may increase
the full training objective,

one-step behavior can be misleading.

Better diagnostics include:

- full loss evaluated periodically;
- moving averages of mini-batch loss;
- validation performance;
- progress across epochs.

The stochastic trajectory is noisy.

We judge the trend,
not every microscopic fluctuation.

Terminology Warning

Strictly:

SGD often means batch size
 $b = 1$.

Mini-batch gradient descent means

$$1 < b < P.$$

But in modern practice:

“SGD” is often used broadly
for stochastic optimization with mini-batches.

Always check what batch size

a source or software package actually uses.

The Main Continuum

Batch size	Gradient noise	Work/update	Updates/epoch
1	high	low	P
medium b	moderate	moderate	$\approx P/b$
P	none	high	1

There is no universally best batch size.

The choice depends on:

dataset size;

hardware;

model;

learning rate;

desired optimization behavior.

What did we learn?

- ◆ Many machine-learning objectives are finite sums: $g = (1/P) \sum_p g_p$.
- ◆ Full-batch gradient descent uses every observation before each update; SGD uses one; mini-batch methods use b .
- ◆ A uniformly sampled stochastic gradient is unbiased: $E[\nabla g_t] = \nabla g$.
- ◆ Mini-batches average several stochastic gradients, generally reducing gradient variance while retaining frequent updates.
- ◆ One epoch is one pass through the data; shuffled epochs prevent systematic ordering from dominating batches.
- ◆ With noisy gradients, learning-rate choice and decay affect the tradeoff between rapid progress and late-stage stability.

Next

- We now have the optimization machinery needed for large datasets.
- Next we shift from supervised prediction to linear structure in unlabeled data.
- Projection geometry will become the foundation for principal component analysis.
- **1.22 - Orthogonal Projection and Data**