EngSci 721

Inverse Problems and Learning From Data

Oliver Maclaren (oliver.maclaren@auckland.ac.nz)

1. *Basic concepts* [5 lectures + 1 Tutorial]

   Forward vs inverse problems. Well-posed vs ill-posed problems. Algebra and calculus of inverse problems (left and right inverses, generalised and pseudo inverses, resolution operators, matrix calculus). Representing higher dimensional problems (image data etc).

2. *Instability and regularisation in linear and nonlinear problems* [6 lectures + 1 Tutorial]

   Instability and related issues for generalised inverses. Introduction to regularisation and trade-offs. Tikhonov regularisation. Higher-order Tikhonov regularisation. Sparsity and regularisation using different norms. Truncated singular value decomposition. Iterative regularisation, including stochastic/mini-batch gradient descent.

3. *Further topics* [3 lectures + 1 Tutorial]

   Regularisation parameter choice, including statistical and machine learning views of regularisation. Confidence sets for linear and nonlinear models. Physics-informed machine learning and neural networks.

Module overview

Inverse Problems and Learning From Data (*Oliver Maclaren*)
[~14 lectures/3 tutorials]

Lecture 11: Stochastic/Minibatch Gradient Descent

Topics:

- Recent developments in iterative methods in machine learning: stochastic/minibatch gradient descent

Eng Sci 721 : Lecture 11.

Stochastic (minibatch) gradient descent

or

'Recent developments in iterative methods'

- Gradient descent eg Landweber iteration

  vs

  Stochastic / minibatch gradient descent

Recall

Zeros & Fixed Points

[Note: here using θ for unknown]

'Theory of everything' (LeCun)

$\boxed{F(\theta) = 0}$ for some F, θ

(everything is a zero of some function)

Alternative:

$\boxed{T(\theta) = \theta}$ for some T, θ.

(everything is a fixed point of some function)

Relationship

If eg

$\boxed{T(\theta) := \alpha F(\theta) + \theta}$, $\alpha \neq 0$

Then

$$\boxed{\begin{array}{c} F(\theta) = 0 \\ \Leftrightarrow \\ T(\theta) = \theta \end{array}}$$

Recall

Fixed point iterations : (define goal as fixed point of some map.

Define:

$$\theta_{k+1} = T(\theta_k)$$

If $\theta_k \to \theta^*$ as $k \to \infty$

Then get $\theta^* = T(\theta^*)$ & hence solves $F(\theta) = 0$

Contraction mappings

The above works if eg

$$\|T(\theta_1) - T(\theta_2)\| \leq a \|\theta_1 - \theta_2\| \quad \text{for all } \theta_1, \theta_2$$

& $0 < a < 1$

More detail: functional analysis & dynamical systems

Difficult/interesting area for eg nonlinear T!

Recall

Normal equations for $y = X\theta$ (data vector; parameter/unknown vector; design matrix / forward map of data 'features' (known/measured))

- $X^T X\theta = X^T y$
- $F(\theta) = X^T y - X^T X\theta = 0$ (zero eqn)
- $T(\theta) = \alpha F(\theta) + \theta = \theta$ (fixed point eqn)

Define:

$$\theta_{k+1} = T(\theta_k)$$
$$= \theta_k + \alpha F(\theta_k)$$
$$= \theta_k + \alpha X^T (y - X\theta_k)$$

→ 'Landweber iteration'

→ Note:

$$\nabla_\theta \|y - X\theta\|^2 = -2 X^T (y - X\theta)$$

⇒ gradient descent!

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \phi(\theta_k)$$

for $\phi = \|y - X\theta\|^2$,

α = 'learning rate'/ step size param.

Recall

Landweber iteration

$$\theta_{k+1} = \theta_k + \alpha X^T (y - X\theta_k)$$

&

$$0 < \alpha < 2/s_1^2$$

s_1: largest singular value

Stop when eg

- $\| X\theta_k - y \| \approx \epsilon$ = measure of noise level
- 'Flattens' off (monitor)

Note: only need matrix-vector products

→ no matrix factorisations

→ efficient for large sparse systems.

Recall

Landweber iteration

Can show in linear case iterates have form

$$\theta_k = V F_k \Lambda^{-1} U^T y$$

V: model space basis; coefficients = Filter factors × usual SVD

$$F_k = \begin{bmatrix} f_1^k & & \\ & \ddots & \\ & & f_n^k \end{bmatrix}$$

$$f_i^k = 1 - (1 - \alpha s_i^2)^k$$

↑ using s_i for singular values

ie 'damps out singular values'

similar to Tikhonov/TSVD

Recall

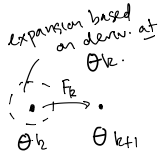
Iterative regularisation : more sophisticated

→ Landweber iteration is basically (first-order) gradient descent (for linear systems)

→ just as in optimisation, we can choose a search/update direction based on higher-order information

↳ more derivatives, 'larger' local neighbourhood

General $\theta_{k+1} = \theta_k + F_k(\theta_k, y)$



where F_k can take into account higher-derivatives* etc.

[* Think: Taylor series for $\theta(k+1) = T(\theta(k))$]

Trade-off: don't want it to optimise too well!
→ want to avoid $\theta^\dagger = X^\dagger y$ limit !

Recall

Iterative regularisation : → prev. lecture

Aster et al. (Chapter 6) Example 6.3 (see attached)

→ Deblurring 2D image (200 x 200 pixels)

→ with & without explicit regularisation

→ use iterative method called 'conjugate gradient least squares' (see eq 7.68?)

→ Blurring operator :
- 40,000 x 40,000
- very sparse
  ↳ SVD produces denser matrices & requires a lot of storage

  ↳ CGLS works well & only requires access to matrix-vector products

→ CGLS w/wout explicit regularisation
→ doesn't matter if only iterative!
→ much cheaper!

Gradient descent in machine learning?

- Gradient descent is also widely used in machine learning (ML), particularly in training (deep) neural networks
- While first-order optimisers (ie based on first derivatives) are not very popular in the pure optimisation community, & improved methods have also been favoured in inverse problems, first order gradient descent methods are surprisingly widely-used & competetive in ML!

  ↳ in particular, a family of variants usually called 'stochastic gradient descent' (SGD) is the 'default'/'go to' approach

  → improvements have been proposed but it's still pretty good!

---

let's look at ingredients!

'Per-data-point' squared errors

So far we have focussed on 'least squares' type objectives, $\|y - X\theta\|^2$ or $\|y - f(x;\theta)\|^2$ (θ: parameter to estimate.)

These can be written as eg

$$\|y - X\theta\|^2 = \sum_{i=1}^{m} (y_i - (X\theta)_i)^2 \qquad \text{row}_i X \equiv x_i^T$$

$$= \sum_{i=1}^{m} (y_i - (\text{row}_i X)\theta)^2 = \sum_{i=1}^{m} (y_i - x_i^T\theta)^2$$

ie sums of squared error over individual data points, per data point errors: $y_i - x_i^T\theta$

$$\Rightarrow \|y - X\theta\|^2 = \sum_{i=1}^{m} \text{'squared error for } i\text{th data point'}$$

---

why/where? →

Expected squared errors?

Minimising $\|y - X\theta\|^2 = \sum_{i=1}^{m} (y_i - x_i^T\theta)^2$ is the same as minimising $\frac{1}{m}\sum_{i=1}^{m} (y_i - x_i^T\theta)^2$

(extra factor)

This can be thought of as an empirical/sample-based estimate of the 'population' expected value

$$\mathbb{E}\left[(Y - x^T\theta)^2\right] \approx \frac{1}{m}\sum_{i=1}^{m} (y_i - x_i^T\theta)^2$$

(expected value; Random vars; Sample realisations)

Many decision/fitting problems can be phrased in terms of minimising expected 'loss' (760)

→ loss? →

→ Hence we often get objectives that can be written as sums over each data point loss

Loss functions? Data misfit measure!

More generally we can define other 'data misfit' functions that can be written as the expected value of the 'per data point' losses/misfits.

→ 'empirical risk minimisation', 'M-estimation' etc (← max likelihood-like.)

→ see eg 760

$$\mathbb{E}\left[\ell(Y, f(X;\theta)\right] \approx \frac{1}{m}\sum_{i=1}^{m} \ell(y_i, f(x_i;\theta))$$

(random vars; empirical estimate/training error; target (population) error)

$\ell(y_i, f(x_i;\theta))$ is the 'loss' (≥ 0) for the i^{th} data point given parameter θ.

we solve the 'empirical loss minimisation':

$$\min_{\theta} \frac{1}{m}\sum_{i=1}^{m} \ell(y_i, f(x_i;\theta))$$

Here we focus on $\ell(y_i, f(x_i;\theta)) = (y_i - x_i^T\theta)^2$

$$\iff \sum_{i=1}^{m} \ell(y_i, f(x_i;\theta)) = \|y - X\theta\|^2$$

ie squared error loss

Expected gradients

When we have a data misfit of the form:

$$\mathbb{E}[\ell(Y, f(X;\theta))] \approx \frac{1}{m}\sum_{i=1}^{m} \ell(y_i, f(x_i;\theta))$$

It holds in general that

$$\nabla_\theta \mathbb{E}[\ell(Y, f(X;\theta))] = \mathbb{E}[\nabla_\theta \ell(y_i, f(x_i;\theta))]$$

in the 'population' ('infinite dataset') &

$$\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} \ell(y_i, f(x_i;\theta)) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \ell(y_i, f(x_i;\theta))$$

in the 'training' (observed) dataset

ie the gradient of the total/expected loss
= the total/expectation of the gradients of the individual losses &

Eg $\nabla_\theta \mathbb{E}[\ell(Y, f(X;\theta))] \approx \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \ell(y_i, f(x_i;\theta))$

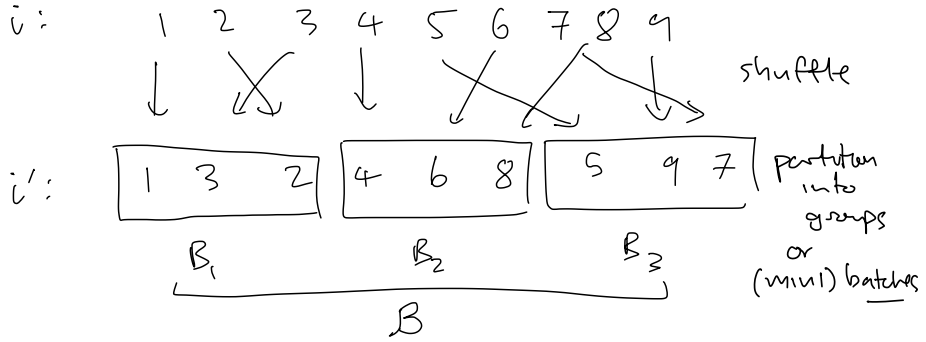
estimate of grad. = expected value of data point grads.

Decompositions

Additive loss functions & their gradients

$$\sum_{i=1}^{m} (y_i - x_i^T\theta)^2 \quad \& \quad \sum_{i=1}^{m} \nabla_\theta (y_i - x_i^T\theta)^2$$

can clearly be computed by summing in any order &/or in 'groups' to be added together, eg



$$\sum_{i=1}^{m} (y_i - x_i^T\theta)^2 = \sum_{B_j \in \mathcal{B}} \sum_{i \in B_j} (y_i - x_i^T\theta)^2$$

Sum over sets of data points (groups) in partition $\mathcal{B}$

Sum over data points in group B_j

Stochastic & minibatch gradient descent

The term 'stochastic gradient descent' is used for both a general family of methods & a special case of this family

More precisely we should call these

1. Standard or 'batch' gradient descent
2. Stochastic gradient descent
3. 'mini-batch' gradient descent

But we often call 2 & 3 'stochastic gradient descent' for short.

Idea

The key idea is simple: instead of solving

$$\min_{\theta} \sum_{i=1}^{m} (y_i - f(x_i;\theta))^2$$

or computing

$$\nabla_{\theta} \sum_{i=1}^{m} (y_i - f(x_i;\theta))^2 \quad \text{etc.}$$

over the whole (training) dataset, we instead compute them over various subsets of the data called 'minibatches'

Note: when being precise we call

full dataset $\longleftrightarrow$ batch

subset of data $\longleftrightarrow$ minibatch

but again sometimes people say 'batch' eg 'batch size' instead of 'minibatch' (which is what they should say)

Gives:

- batch gradient descent: full data
- minibatch gradient descent: subsets of data
- stochastic gradient descent: single data points

Using mini batches in gradient descent

Instead of updates like (for linear least sq.)

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \|y - A\theta\|^2$$

(over full dataset)

We do a series of updates over subsets (minibatches) of data, chosen randomly (hence 'stochastic')

eg an update looks like

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \|y - A\theta\|^2$$

(over minibatch)

→ can think of doing updates using different (noisy) estimates of corresponding expected loss / expected gradient

Minibatch gradient descent

- We will randomly divide training data into non-overlapping subsets called minibatches
- loop through minibatches, updating as we go based on minibatch gradient until all minibatches used
- That gives n_mbatch='number minibatches' updates or iterations and one 'epoch' (all minibatches seen)
- we then repeat: redivide training data, iterate through minibatches
- do this for n_epoch = 'number of epochs'
- gives n_mb × n_epoch total parameter updates (iterations)

Algorithm : Mini-batch gradient descent for $\|y - f(x;\theta)\|^2$ (nonlinear least sq).

$\alpha \leftarrow$ learning rate (fixed here; can vary in general)

$\theta_0 \leftarrow$ initial parameter-estimate for iteration 1

For $i = 1 : n\_epoch$

Randomly divide dataset into n_mbatch minibatches

For $j = 1 : n\_mbatch$

iteration $k = (i-1) \cdot n\_mbatch + j$

$\theta_k^0 = \theta_{k-1}$ intial guess

Compute $g_k = \nabla_\theta \|y - f(x; \theta_k^0)\|^2$ for minibatch j of epoch i

Update $\theta_k \leftarrow \theta_k^0 - \alpha g_k$

record loss $l_{ij} = l_k = \|y - f(x;\theta_k)\|^2$ for minibatch

Epoch loss$(i) = \sum_j l_{ij}$

Notes

- don't need to load whole dataset into memory
- actually improves 'generalisation' ie regularisation towards good out-of-sample prediction models
- can analyse via estimators of expectations but...
- still not well-understood!
- gradients computed for large neural networks via 'fancy chain rule'
  - pytorch & tensorflow etc implement derivative propagation for you!
- Many improvements, tricks etc possible!

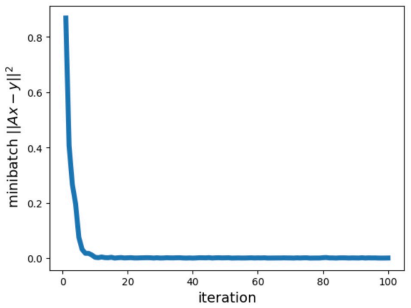
Example : deconvolution revisited

```
nepoch = 100
nmb = 100
xs = np.zeros((nepoch*nmb,len(x)))
xs0 = np.zeros(len(x))
x_norms = np.zeros(nepoch*nmb)
data_norms = np.zeros(nepoch*nmb)
data_norms_matrix = np.zeros((nepoch,nmb))
epoch_data_norms = np.zeros(nepoch)

obs_indices = np.arange(0,len(y_noisy),dtype=int)
np.random.shuffle(obs_indices)
minibatch_partition = np.array_split(obs_indices,nmb)

ind = minibatch_partition[0]
ind.sort()
A_mb = A[ind,:]

U, S, Vh = np.linalg.svd(A_mb, full_matrices=False)
s1 = S[0]
print(s1)

for i in range(0,nepoch):
    alpha = 1/(s1**2)
    np.random.shuffle(obs_indices)
    minibatch_partition = np.array_split(obs_indices,nmb)
    for j in range(0,nmb):
        mb_indices = minibatch_partition[j]
        mb_indices.sort()
        A_mb = A[mb_indices,:]
        k = i*nmb + j
        #update rule
        xs[k,:] = xs0 + alpha*A_mb.T@(y_noisy[mb_indices]-A_mb@xs0)
        xs0 = xs[k,:]
        #calc norms
        x_norms[k] = np.linalg.norm(xs0,2)
        #minibatch norm
        data_norms[k] = np.linalg.norm(y_noisy[mb_indices] - A_mb@xs0)
        data_norms_matrix[i,j] = np.linalg.norm(y_noisy[mb_indices] - A_mb@xs0)

    epoch_data_norms[i] = np.sum(data_norms_matrix[i,:])
```

Example : 100 minibatches of size 10

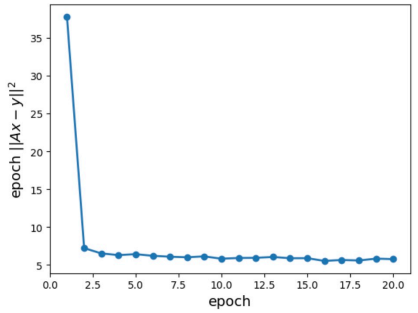
```
n = 100 #iterations
plt.plot(np.linspace(1,n,n),data_norms[0:n]**2,linewidth=5)
plt.ylabel(r' minibatch $||Ax-y||^2$',fontsize=14)
plt.xlabel(r'iteration',fontsize=14)
plt.show()
```

✓ 0.1s



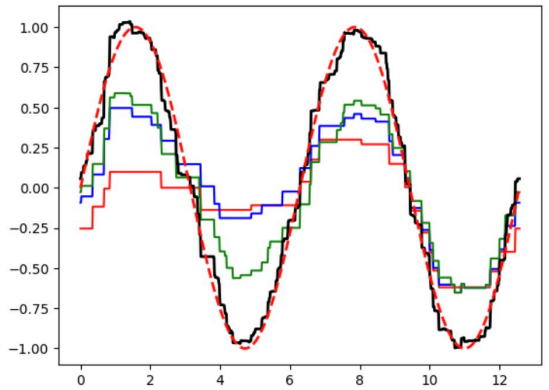

```
# epochs
n = 20
plt.plot(np.linspace(1,n,n),epoch_data_norms[0:n]**2,linewidth=2,marker='o')
plt.ylabel(r'epoch $||Ax-y||^2$',fontsize=14)
plt.xlabel(r'epoch',fontsize=14)
plt.xlim(0,n+1)
plt.show()
```

✓ 0.1s




```
plt.plot(t,xs[0,:],'r')
plt.plot(t,xs[1,:],'b')
plt.plot(t,xs[2,:],'g')
plt.plot(t,xs[15,:],'k',linewidth=2)
plt.plot(t,x,'r--',linewidth=2)
plt.show()
```

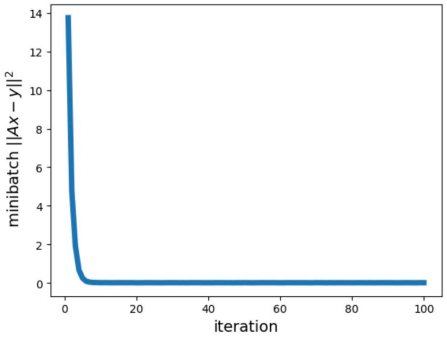
✓ 0.1s

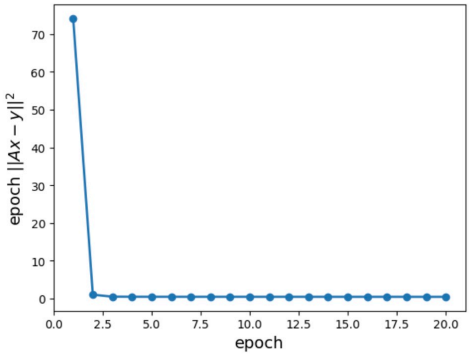


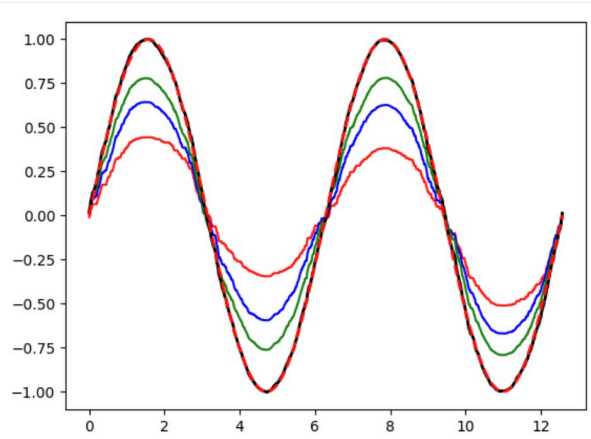
Here: piecewise linear? updates are linear combo of subset of A^T col (A rows)

Example : minibatches of size 200

→ back to Landweber / full grad. descent style.







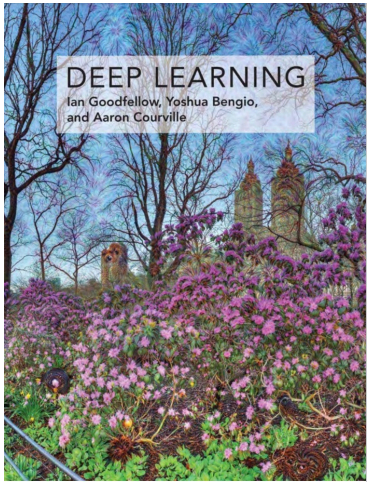
Exercise:

Try on polynomial (x^{19}) example!

Further info & extensions

References:

(Goodfellow, Bengio & Courville)



Covers extensions eg

momentum

general models & losses

further alg.

+ much more!