COMS0018: Lab3 Practical

Dima Damen
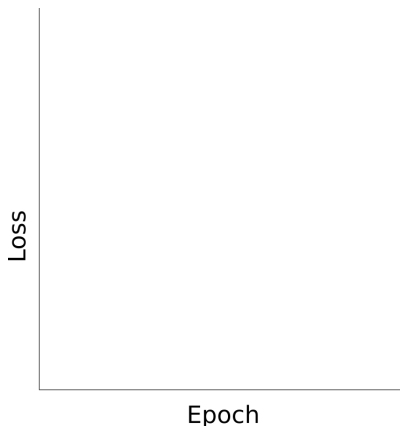
Dima.Damen@bristol.ac.uk

Bristol University, Department of Computer Science
Bristol BS8 1UB, UK

October 19, 2018
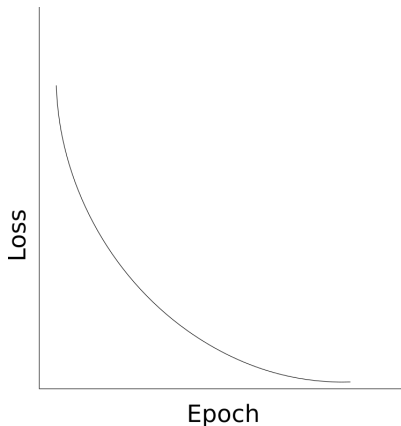
Evaluation Metrics

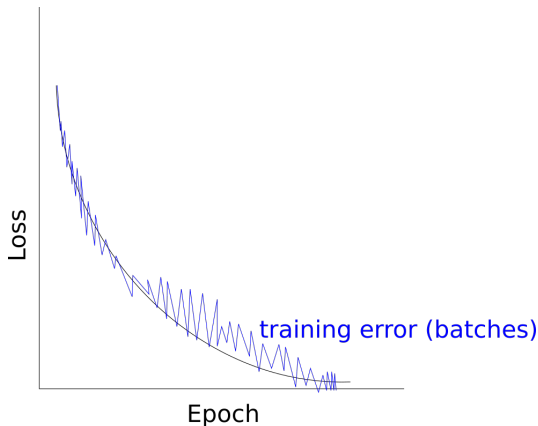▶ Let's look again at the curves you are getting during training

Evaluation Metrics

▶ Optimally, the loss decreases after each iteration

Evaluation Metrics

▶ Practically, due to mini-batch optimisation, you see this wiggly curve

Evaluation Metrics - What if??

▶ Training Error starts to go up?!

training error (batches)

Evaluation Metrics - What if??

▶ Training error wiggles A LOT?!

training error (batches)
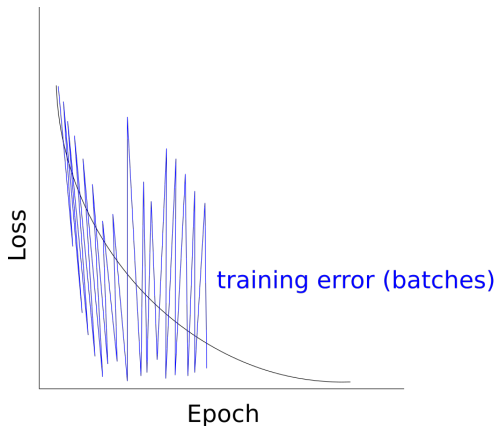
What did you choose to change in your first model?

▶ There are too many parameters to change. Which one did you feel you should change first?

Hyperparameters in this practical

▶ In this practical, we will NOT revisit the architecture - will stick to it
▶ We will though think about the critical parameters to change, for a given architecture
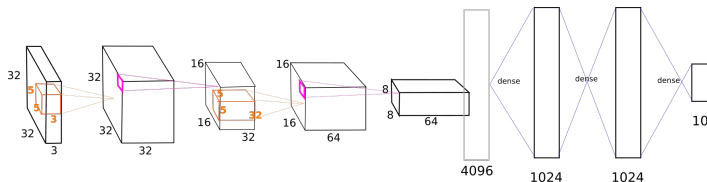
Selecting hyperparameter values

- ▶ For each hyperparameters, one must understand the relationship between its value and each of the following:
  - ▶ Training error/loss
  - ▶ Testing error/loss (generalisation)
  - ▶ Computational resources (memory and runtime)

1. Batch-size Training

- ▶ Recall from lecture 2, that there are two extremes when performing gradient descent
  - ▶ Calculating the gradient from a single example
  - ▶ Calculating the gradient for the whole dataset
- ▶ Neither is ideal, thus we typically calculate the gradient from a number of data points
- ▶ This number is referred to as the 'batch size'

1. Batch-size Training

- ► Recall from lecture 2, that there are two extremes when performing gradient descent
  - ► Calculating the gradient from a single example
  - ► Calculating the gradient for the whole dataset
- ► Neither is ideal, thus we typically calculate the gradient from a number of data points
- ► This number is referred to as the 'batch size'
- ► To correctly approximate the loss from a batch, it is crucial that samples are selected randomly

1. Batch-size Training

- ▶ Recall from lecture 2, that there are two extremes when performing gradient descent
    - ▶ Calculating the gradient from a single example
    - ▶ Calculating the gradient for the whole dataset
- ▶ Neither is ideal, thus we typically calculate the gradient from a number of data points
- ▶ This number is referred to as the 'batch size'
- ▶ To correctly approximate the loss from a batch, it is crucial that samples are selected randomly
- ▶ However, there are approaches that sample data for a purpose **(read about hard mining)**

1. Batch-size Training

- ▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset
- ▶ The amount of wiggle in the training loss is related to the batch size.

1. Batch-size Training

▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset

▶ The amount of wiggle in the training loss is related to the batch size.

▶ However there are general guidelines:

1. Batch-size Training

▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset

▶ The amount of wiggle in the training loss is related to the batch size.

▶ However there are general guidelines:

▶ Larger batches indeed provide better approximation of the full-dataset gradient

1. Batch-size Training

- ▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset
- ▶ The amount of wiggle in the training loss is related to the batch size.
- ▶ However there are general guidelines:
  - ▶ Larger batches indeed provide better approximation of the full-dataset gradient
  - ▶ However, as batch size increases, the increase in accuracy or the decrease in training time is NOT linear

1. Batch-size Training

- ▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset
- ▶ The amount of wiggle in the training loss is related to the batch size.
- ▶ However there are general guidelines:
  - ▶ Larger batches indeed provide better approximation of the full-dataset gradient
  - ▶ However, as batch size increases, the increase in accuracy or the decrease in training time is NOT linear
  - ▶ Due to parallel processing, the typical limitation to batch size is the GPU memory

1. Batch-size Training

▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset

▶ The amount of wiggle in the training loss is related to the batch size.

▶ However there are general guidelines:

    ▶ Larger batches indeed provide better approximation of the full-dataset gradient

    ▶ However, as batch size increases, the increase in accuracy or the decrease in training time is NOT linear

    ▶ Due to parallel processing, the typical limitation to batch size is the GPU memory

    ▶ There's a minimum size below which the gradient is too noisy

1. Batch-size Training

- ▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset
- ▶ The amount of wiggle in the training loss is related to the batch size.
- ▶ However there are general guidelines:
    - ▶ Larger batches indeed provide better approximation of the full-dataset gradient
    - ▶ However, as batch size increases, the increase in accuracy or the decrease in training time is NOT linear
    - ▶ Due to parallel processing, the typical limitation to batch size is the GPU memory
    - ▶ There's a minimum size below which the gradient is too noisy
    - ▶ We are always searching for this sweet spot between too small and too large

1. Batch-size Training

► The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset

► The amount of wiggle in the training loss is related to the batch size.

► However there are general guidelines:

  ► Larger batches indeed provide better approximation of the full-dataset gradient

  ► However, as batch size increases, the increase in accuracy or the decrease in training time is NOT linear

  ► Due to parallel processing, the typical limitation to batch size is the GPU memory

  ► There's a minimum size below which the gradient is too noisy

  ► We are always searching for this sweet spot between too small and too large

  ► To make the most of the GPU architectures, batches that are a power of 2 provide the best runtime - 128, 256, ...

1. Batch-size Training

▶ The effect of changing the batch-size on the accuracy of a dataset, depends on the dataset

▶ The amount of wiggle in the training loss is related to the batch size.

▶ However there are general guidelines:

  ▶ Larger batches indeed provide better approximation of the full-dataset gradient

  ▶ However, as batch size increases, the increase in accuracy or the decrease in training time is NOT linear

  ▶ Due to parallel processing, the typical limitation to batch size is the GPU memory

  ▶ There's a minimum size below which the gradient is too noisy

  ▶ We are always searching for this sweet spot between too small and too large

  ▶ To make the most of the GPU architectures, batches that are a power of 2 provide the best runtime - 128, 256, ...

  ▶ For small-sized data samples, batches between 32 and 256 are typical

  ▶ For large-sized data samples, batches of 8 or 16 are common

2. Learning Rate

► The learning rate is the **most important hyperparameter** to set

"If you have time to tune only one parameter, tune the learning rate"[1]
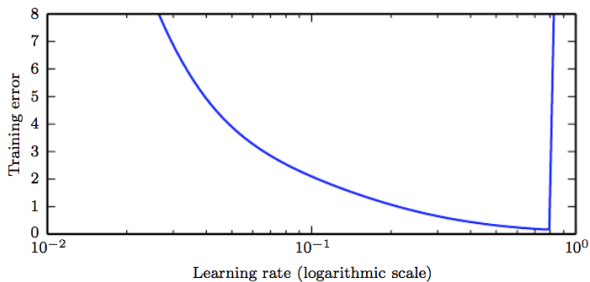
---

[1] Goodfellow et al, p 424

Dima Damen
Dima.Damen@bristol.ac.uk

2. Learning Rate

Figure 11.1: Typical relationship between the learning rate and the training error.
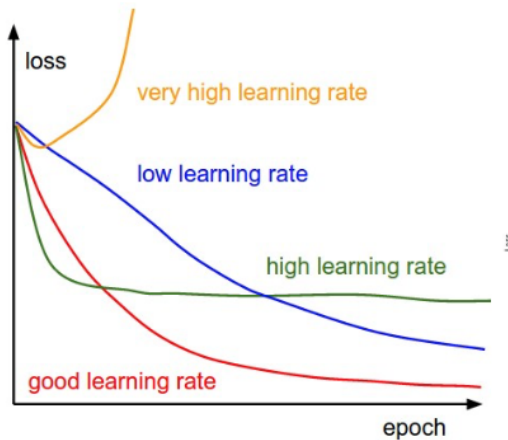
Goodfellow et al p 425

2. Learning Rate

2. Learning Rate

- ► You have learnt about decaying the learning rate
- ► Remind yourself about the Adam optimiser (Lec 3) to use it in the Lab work

3. Parameter initialisation

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1]   ILYASU@GOOGLE.COM
James Martens   JMARTENS@CS.TORONTO.EDU
George Dahl   GDAHL@CS.TORONTO.EDU
Geoffrey Hinton   HINTON@CS.TORONTO.EDU

ICML 2013

3. Parameter initialisation

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1]   ILYASU@GOOGLE.COM
James Martens   JMARTENS@CS.TORONTO.EDU
George Dahl   GDAHL@CS.TORONTO.EDU
Geoffrey Hinton   HINTON@CS.TORONTO.EDU

Abstract

Deep and recurrent neural networks (DNNs
and RNNs respectively) are powerful mod-
els that were considered to be almost impos-
sible to train using stochastic gradient de-
scent with momentum. In this paper, we
show that when stochastic gradient descent
with momentum uses a well-designed random
initialization and a particular type of slowly
increasing schedule for the momentum pa-
rameter, it can train both DNNs and RNNs
(on datasets with long-term dependencies) to
levels of performance that were previously
achievable only with Hessian-Free optimiza-
tion. We find that both the initialization
and the momentum are crucial since poorly
initialized networks cannot be trained with
momentum and well-initialized networks per-
form markedly worse when the momentum is
absent or poorly tuned.

ICML 2013

3. Parameter initialisation

- ▶ The cost function in a DNN is non-convex
- ▶ Any non-convex optimisation algorithm will depend on the initial parameters

3. Parameter initialisation

- ▶ The cost function in a DNN is non-convex
- ▶ Any non-convex optimisation algorithm will depend on the initial parameters
- ▶ In the lab you'll learn about the *Xavier initialisation* as a practical way to initialise the weights in a DNN

3. Parameter initialisation

- ▶ The cost function in a DNN is non-convex
- ▶ Any non-convex optimisation algorithm will depend on the initial parameters
- ▶ In the lab you'll learn about the *Xavier initialisation* as a practical way to initialise the weights in a DNN
- ▶ However, more practically, it's best to start from some pre-trained model on a relevant problem with larger datasets

3. Parameter initialisation

- ▶ The cost function in a DNN is non-convex
- ▶ Any non-convex optimisation algorithm will depend on the initial parameters
- ▶ In the lab you'll learn about the *Xavier initialisation* as a practical way to initialise the weights in a DNN
- ▶ However, more practically, it's best to start from some pre-trained model on a relevant problem with larger datasets
- ▶ In images for example, it's customary to start from a model pre-trained on ImageNet
- ▶ This gives an informative initial set of parameters to train CNNs when images are targeted

4. Batch Normalisation

▶ Another powerful practical modification to the baseline training algorithm is known as **batch normalisation**

4. Batch Normalisation

- ▶ Another powerful practical modification to the baseline training algorithm is known as **batch normalisation**
- ▶ Recall the standardisation approach to data

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma}$$

4. Batch Normalisation

▶ Another powerful practical modification to the baseline training algorithm is known as **batch normalisation**

▶ Recall the standardisation approach to data

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma}$$

▶ The distribution of $\hat{\mathbf{x}}$ would then be zero-meaned with a standard deviation of 1

4. Batch Normalisation

- ▶ Another powerful practical modification to the baseline training algorithm is known as **batch normalisation**
- ▶ Recall the standardisation approach to data

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma}$$

- ▶ The distribution of $\hat{\mathbf{x}}$ would then be zero-meaned with a standard deviation of 1
- ▶ When training using $\hat{\mathbf{x}}$ instead of $\mathbf{x}$, the training converges faster.
- ▶ You can compensate for that effect on the output by learnt variables

$$y = \boldsymbol{\lambda}\hat{\mathbf{x}} + \boldsymbol{\beta}$$

4. Batch Normalisation

► When applied to **each** layer in the network, all layers are given the chance to converge faster

► Using batch normalisation, higher learning rates can be used

And now....

READY....

STEADY....

GO...

Dima Damen
Dima.Damen@bristol.ac.uk