Applied Deep Learning

Chapter 2: Feedforward Neural Networks

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2026

Full-Batch Training

Batch $\equiv$ *the dataset reserved for training*

In full-batch training, we compute gradients at all data-points in the batch: *so, we need to wait till forward and backward pass are over for all B data-points*

This can be a huge burdensome!

+ *Wait a moment! Don't we use all the dataset for training?*
– No! As you may have noticed in the assignments, *we reserve a part of it for testing*
+ *And, why should it be a burdensome?*
– OK! Let's get more into datasets!

Public Datasets

Let's consider our example of image recognition: *we want to recognize the hand-written number in an image*. For this, we need to have access *to images of hand-written numbers*. This has been done before by people at *National Institute of Standards and Technology* and collected in a database called

Modified National Institute of Standards and Technology (MNIST)

that is available for public on internet

There are several of such public databases; *some well-known examples are*

- *CIFAR-10 and CIFAR-100 by Canadian Institute For Advanced Research*
- *ImageNet initiated by Fei-Fei Li at Princeton University*
- *Caltech-101 and Caltech-256 compiled at Caltech*
- *Fashion MNIST that collects fashion images and labels them*

You can find out more about public datasets online

Public Datasets: *Accessing via PyTorch*

PyTorch provides us a simple tool to access these public datasets, e.g.,

```
>> import torchvision.datasets as DataSets

>> dataset = DataSets.MNIST( ... )
>> dataset = DataSets.CIFAR10( ... )
```

In the example of MNIST, we load the dataset which contains the pixel vectors of the images of size 28×28. This means that we load a *list of pairs were each pair contains*

a 784-dimensional vector of pixel values and a label that is in $\{0, 1, \dots, 9\}$

Public Datasets: *How Do They Look?*

Public datasets include a large amount of data-points with their labels

> *MNIST includes 70,000 images of hand-written numbers with their true labels: from these 70,000 we use 60,000 for training and 10,000 for test*

This means that once we load the MNIST dataset, we make a batch of 60,000 images to train our FNN. Once the training is over, we test the performance of the trained FNN on the remaining 10,000 images

---

Back to our problem, this means that *our full-batch training performs each iteration of the gradient descent after*

60,000 forward and backward passes over the FNN

which sounds a lot!

Full-Batch Training: *Complexity*

Given the example of MNIST, let's see roughly how long it takes to do a full-batch training: *if we need 100 iterations of gradient descent, we need to pass back and forth for 6×10^6 times!*

+ *But do we really need to do this much? This sounds impossible in large NNs!*
- No! We really don't need! We can do the training much faster

---

The full-batch training is really not practical: *in practice, we use stochastic (mini-batch) gradient descent to train our NN with feasible complexity*

Let's take a look at these approaches!

Sample-Level Training

The most primary idea is to apply one step of gradient descent after each forward and backward pass: *in our FNN this means that we do the following*

```
SampLevel_GradientDescent():
1:  Initiate with some initial values $\{\mathbf{W}_1^{(0)}, \mathbf{W}_2^{(0)}, \mathbf{W}_3^{(0)}\}$ and set a learning rate $\eta$
2:  Start at $b = 1$
3:  while weights not converged do
4:      if $b > B$ then
5:          Update $b \leftarrow 1$                                  # start over with the dataset
6:      end if
7:      $\texttt{NN.values} \leftarrow \texttt{ForwardProp}\left(\boldsymbol{x}_b, \{\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{W}_3^{(t)}\}\right)$
8:      $\{\mathbf{G}_{1,b}, \mathbf{G}_{2,b}, \mathbf{G}_{3,b}\} \leftarrow \texttt{BackProp}\left(\boldsymbol{x}_b, v_b, \{\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{W}_3^{(t)}\}, \texttt{NN.values}\right)$
9:      Update $\mathbf{W}_\ell{}^{(t+1)} \leftarrow \mathbf{W}_\ell{}^{(t)} - \eta\, \mathbf{G}_{\ell,b}$             # sample_level update
10:     Update $b \leftarrow b + 1$                               # go for next data-point
11: end while
```

We call this approach *sample-level training*

Sample-Level Training: *Meaning*

+ *But what does it mean in the sense of empirical risk minimization? Aren't we now doing something different from the standard gradient descent?!*
– Yes! We are in fact performing an approximative gradient descent

---

Consider the an ideal scenario in which

$$\mathbf{G}_{\ell,1} = \mathbf{G}_{\ell,2} = \ldots = \mathbf{G}_{\ell,B}$$

In this case, we do not need to wait for the batch to be fully over, since

$$\mathbf{G}_{\ell,1} = \texttt{mean}\ \left(\mathbf{G}_{\ell,1}, \mathbf{G}_{\ell,2}, \ldots, \mathbf{G}_{\ell,B}\right)$$

In other words, in this case

sample-level training $\equiv$ *full-batch training*

Sample-Level Training: *Meaning*

In practice, at each data-point we calculate a noisy-version of a ground truth gradient $\bar{\mathbf{G}}_\ell$. In other words, we can think of $\mathbf{G}_{\ell,b}$ for each b as

$$\mathbf{G}_{\ell,b} = \bar{\mathbf{G}}_\ell + \text{Noise}$$

If this noise is small enough, we can say that

$$\mathbf{G}_{\ell,b} \approx \bar{\mathbf{G}}_\ell \approx \texttt{mean}(\mathbf{G}_{\ell,1}, \mathbf{G}_{\ell,2}, \ldots, \mathbf{G}_{\ell,B})$$

and therefore, we can conclude that

sample-level (worse approx. $\bar{\mathbf{G}}_\ell$) $\approx$ full-batch (better approx. $\bar{\mathbf{G}}_\ell$)

In this case, we say that $\mathbf{G}_{\ell,b}$ is an estimator of the ground truth gradient

Sample-Level Training: *Repetitive Cycle Issue*

Naive sample-level update can trap us into a repetitive cycle: *in simple words, we can end up with our initial point at the end of the batch. For instance, consider the following dummy (but possible) scenario in our three-layer FNN*

We start with $\mathbf{W}_\ell^{(0)}$ and get into the batch for the first time

- *We update $\mathbf{W}_\ell^{(0)}$ after the first data-point to $\mathbf{W}_\ell^{(1)}$*
- *We update $\mathbf{W}_\ell^{(1)}$ after the second data-point to $\mathbf{W}_\ell^{(2)}$*
- $\ldots$
- *We update $\mathbf{W}_\ell^{(B-1)}$ after the last data-point to $\mathbf{W}_\ell^{(B)}$*

Now, assume that $\mathbf{W}_\ell^{(B)} = \mathbf{W}_\ell^{(0)}$ for all layers again!

In the above dummy example, further looping over the batch is *useless*, since *we always get back to the initial point*: *this is the most basic example of the repetitive cycle issue*

Stochastic Sample-Level Training: *SGD*

+ *How can we avoid such cyclic behaviors?*
- We can use *Stochastic Gradient Descent (SGD)*

Each time we are to loop over our training batch, *we shuffle the data-points randomly: this way we avoid next loop behave like the previous one*

This idea is called Stochastic Gradient Descent (SGD)

SGD is the most common algorithm for training of NNs!

What does random shuffling mean?

It means randomly permuting the data-points

Stochastic Gradient Descent

```
SGD():
 1: Initiate with some initial values $\{\mathbf{W}_1^{(0)}, \mathbf{W}_2^{(0)}, \mathbf{W}_3^{(0)}\}$ and set a learning rate $\eta$
 2: Randomly shuffle the batch and start at $b = 1$
 3: while weights not converged do
 4:     if $b > B$ then
 5:         Randomly shuffle the batch and set $b \leftarrow 1$          # random shuffling
 6:     end if
 7:     NN.values ← ForwardProp $(\boldsymbol{x}_b, \{\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{W}_3^{(t)}\})$
 8:     $\{\mathbf{G}_{1,b}, \mathbf{G}_{2,b}, \mathbf{G}_{3,b}\}$ ← BackProp $(\boldsymbol{x}_b, v_b, \{\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{W}_3^{(t)}\}$, NN.values)
 9:     Update $\mathbf{W}_\ell^{(t+1)} \leftarrow \mathbf{W}_\ell^{(t)} - \eta\, \mathbf{G}_{\ell,b}$          # sample_level update
10:     Update $b \leftarrow b + 1$          # go for next data-point
11: end while
```

+ *But, doesn't sample-level training lead to any drawback?*
– Sure! But we accept this drawback as a cost we pay for less complexity

Let's see how this trade-off looks like

Recap: *Variance*

For random variable x with mean μ, the variance is defined as

$$\mathbb{V}\mathrm{ar}\left\{x\right\} = \mathbb{E}\left\{(x-\mu)^2\right\} = \mathbb{E}\left\{x^2\right\} - \mu^2$$

Clearly, when x is zero-mean, we can say $\mathbb{V}\mathrm{ar}\left\{x\right\} = \mathbb{E}\left\{x^2\right\}$

Properties of Variance

For any random variable x and constant c, we have

$$\mathbb{V}\mathrm{ar}\left\{cx\right\} = c^2 \mathbb{V}\mathrm{ar}\left\{x\right\}$$

Let $x_1, \dots, x_N$ be N independent random variables; then, we have

$$\mathbb{V}\mathrm{ar}\left\{\sum_{n=1}^{N} x_n\right\} = \sum_{n=1}^{N} \mathbb{V}\mathrm{ar}\left\{x_n\right\}$$

Recap: *Variance*

Now, assume $x_1, \ldots, x_N$ are N independent zero-mean random variables all with variance σ^2: *let $\bar{x}$ be the arithmetic average of $x_1, \ldots, x_N$, i.e.,*

$$\bar{x} = \mathtt{mean}\left(x_1, \ldots, x_N\right) = \frac{1}{N} \sum_{n=1}^{N} x_n$$

We could then say

$$\begin{aligned}\mathbb{Var}\left\{\bar{x}\right\} &= \mathbb{Var}\left\{\frac{1}{N} \sum_{n=1}^{N} x_n\right\} = \frac{1}{N^2} \mathbb{Var}\left\{\sum_{n=1}^{N} x_n\right\} \\ &= \frac{1}{N^2} \sum_{n=1}^{N} \underbrace{\mathbb{Var}\left\{x_n\right\}}_{\sigma^2} = \frac{1}{N^2}\left(N\sigma^2\right) \\ &= \frac{\sigma^2}{N}\end{aligned}$$

variance of average drops by $1/\#$ samples

Complexity-Accuracy Trade-off of SGD

Now, let's get back to our problem: *when we talked about the meaning of symbol level update, we said*

> *In practice, at each data-point we calculate a noisy-version of a ground truth gradient $\bar{\mathbf{G}}_\ell$. In other words, we can think of $\mathbf{G}_{\ell,b}$ for each b as*
>
> $$\mathbf{G}_{\ell,b} = \bar{\mathbf{G}}_\ell + \text{Noise}$$

and called $\mathbf{G}_{\ell,b}$ an estimator of the ground truth

---

Let's make the above statement a bit more formal: *we assume that* Noise *for each b is a matrix with independent zero-mean entries all with variance σ^2*, i.e.,

$$\mathbf{G}_{\ell,b} = \bar{\mathbf{G}}_\ell + \mathbf{N}_{\ell,b}$$

where we define $\bar{\mathbf{G}}_\ell$ to be the gradient of the true risk

Complexity-Accuracy Trade-off of SGD

What is the true risk? If you remember, when we started with training

Our goal was to minimize the risk $R(\mathbf{w})$

However, we could not do this: *since we did not know (1) the true function, and (2) the data distribution.* Thus,

we approximated the true risk $R(\mathbf{w})$ *with empirical risk* $\hat{R}(\mathbf{w})$

We assume that $\bar{\mathbf{G}}_\ell$ is the gradient of true risk with respect to $\mathbf{W}_\ell$, i.e.,

$$\bar{\mathbf{G}}_\ell = \nabla_{\mathbf{W}_\ell} R(\mathbf{w})$$

+ *Can we determine this gradient?*
– Of course not! We can only approximate it with $\nabla_{\mathbf{W}_\ell} \hat{R}(\mathbf{w})$

Complexity-Accuracy Trade-off of SGD

Let's see accurate the gradient is approximated, when we do full-batch training

In full-batch training, we determine the gradient as

$$\hat{\mathbf{G}}_\ell^{\text{batch}} = \texttt{mean}(\mathbf{G}_{\ell,1}, \mathbf{G}_{\ell,2}, \ldots, \mathbf{G}_{\ell,B}) = \frac{1}{B}\sum_{b=1}^{B}\left(\bar{\mathbf{G}}_\ell + \mathbf{N}_{\ell,b}\right)$$

$$= \bar{\mathbf{G}}_\ell + \underbrace{\frac{1}{B}\sum_{b=1}^{B}\mathbf{N}_{\ell,b}}_{\hat{\mathbf{N}}_\ell^{\text{batch}}} = \bar{\mathbf{G}}_\ell + \hat{\mathbf{N}}_\ell^{\text{batch}}$$

Recall that by *arithmetic averaging variance drops by $1/\#$ of samples*

In full-batch training the approximated gradient is different from the true gradient by an error whose variance drops as σ^2/B

Complexity-Accuracy Trade-off of SGD

In full-batch training the approximated gradient is different from the true gradient by an error whose variance drops as σ^2/B

Now, let's compare it to SGD

In SGD, we approximate the gradient with a sample gradient, i.e.,

$$\hat{\mathbf{G}}_\ell^{\mathrm{SGD}} = \mathbf{G}_{\ell,b} = \bar{\mathbf{G}}_\ell + \mathbf{N}_{\ell,b}$$

SGD is still approximating the true gradient but with much larger variance: *entries of $\mathbf{N}_{\ell,b}$ have all variance σ^2*

For instance, consider MNIST with $60,000$ samples: *by full-batch training we get gradient values whose difference from the entries of the true gradient is approximately 1.67×10^{-5} times smaller that those gradient entries calculated by SGD!*

Complexity-Accuracy Trade-off of SGD

In the context of ML, we often say: *in the analyses of last slides,*

SGD and full-batch training are both unbiased estimators of $\bar{\mathbf{G}}_\ell$

We call them unbiased, since $\mathbb{E}\left\{\hat{\mathbf{G}}_\ell^{\mathrm{SGD}}\right\} = \mathbb{E}\left\{\hat{\mathbf{G}}_\ell^{\mathrm{batch}}\right\} = \bar{\mathbf{G}}_\ell$

Complexity-Accuracy Trade-off

Assume that a forward and backward pass takes time T and that gradient of the risk at each sample be an unbiased estimators of the true gradient; then,

1. *each step of SGD takes time T while each step of full-batch training takes BT with B being the batch size*
2. *if we denote the variance of estimation given by SGD by σ^2, the variance of full-batch estimator is σ^2/B*

Training via Mini-Batches

- + *But, can't we play with this trade-off? For instance, increase a bit the complexity to improve the accuracy!*
- – Yes! This is the idea of mini-batch training

In mini-batch training, we divide the whole batch of data into mini-batches:

- *after each mini-batch is over, we average the gradients over the mini-batch*
- *we apply one step of gradient descent using this averaged gradient*

To avoid cyclic behavior, we still *shuffle the dataset randomly each time we start a new loop over it*. This training approach is hence often called

Mini-Batch Stochastic Gradient Descent = *Mini-Batch SGD*

Mini-Batch SGD

```
mBatchSGD():
 1: Initiate with some initial values $\{\mathbf{W}_\ell^{(0)}\}$ and set a learning rate $\eta$
 2: Randomly shuffle the batch and divide it into mini-batches of size $\Omega$
 3: Denote the number of mini-batches by $\Xi = \lceil B/\Omega \rceil$ and start at $\xi = 1$
 4: while weights not converged do
 5:     if $\xi > \Xi$ then
 6:         Randomly shuffle the batch and divide it into mini-batches of size $\Omega$
 7:         Set $\xi \leftarrow 1$                                # start over with the dataset
 8:     end if
 9:     for $\omega = 1, \ldots, \Omega$ do
10:         NN.values $\leftarrow$ ForwardProp $(\boldsymbol{x}_\omega, \{\mathbf{W}_\ell^{(t)}\})$
11:         $\{\mathbf{G}_{\ell,\omega}\}$ $\leftarrow$ BackProp $(\boldsymbol{x}_\omega, v_\omega, \{\mathbf{W}_\ell^{(t)}\}$, NN.values$)$
12:     end for
13:     Update $\mathbf{W}_\ell^{(t+1)} \leftarrow \mathbf{W}_\ell^{(t)} - \eta$ mean $(\mathbf{G}_{\ell,1}, \ldots, \mathbf{G}_{\ell,\Omega})$
14:     Update $\xi \leftarrow \xi + 1$                           # go for next mini-batch
15: end while
```

Complexity-Accuracy Trade-off

It is easy to see that

- *mini-batch training reduces to full-batch training when we set the size of mini-batches* to B, i.e., $\Omega = B$
- *mini-batch training reduces to SGD when we set the size of mini-batches to* 1, *i.e.,* $\Omega = 1$

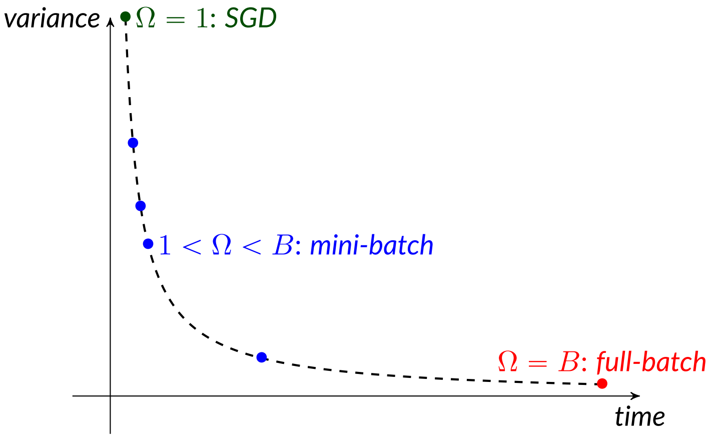
Complexity-Accuracy Trade-off

Assume that a forward and backward pass takes time T and that gradient of the risk at each sample be an unbiased estimators of the true gradient; then,

1. *each step of SGD takes time T while each step of mini-batch* training takes ΩT with Ω being the mini-batch size
2. *if we denote the variance of estimation given by SGD by σ^2, the variance of mini-batch* estimator is σ^2/Ω

Complexity-Accuracy Trade-off

The complete trade-off can be visualized as



$\boxed{\textit{Mini-batch size is what specifies the trade-off point}}$

Few Definitions: *Epoch and Iteration*

In the language of deep learning there are few terms that we must know

Batch Size

Through time, the term mini-batch has been transformed to batch, and the complete batch is referred to as training dataset or the full batch. People hence call the size of each mini-batch, i.e., Ω, the batch size

Iteration

When we take one step of gradient descent, we take one iteration. So, one iteration is over when we finish with a mini-batch

Epoch

An epoch is over when we finish once with the whole training dataset

Few Definitions: *Epoch and Iteration*

We can annotate these definitions in our algorithm

```
mBatchSGD():
 1: Initiate with some initial values $\{\mathbf{W}_\ell^{(0)}\}$ and set a learning rate $\eta$
 2: Randomly shuffle training dataset and make mini-batches of size $\Omega \equiv$ batch-size
 3: Denote the number of mini-batches by $\Xi = \lceil B/\Omega \rceil$ and start at $\xi = 1$
 4: while weights not converged do
 5:     if $\xi > \Xi$ then
 6:         Randomly shuffle the batch and divide it into mini-batches of size $\Omega$
 7:         Set $\xi \leftarrow 1$          $\leftarrow$ one epoch is over, we start another epoch
 8:     end if
 9:     for $\omega = 1, \ldots, \Omega$ do
10:         NN.values $\leftarrow$ ForwardProp $(\boldsymbol{x}_\omega, \{\mathbf{W}_\ell^{(t)}\})$ going through a min-batch
11:         $\{\mathbf{G}_{\ell,\omega}\} \leftarrow$ BackProp $(\boldsymbol{x}_\omega, v_\omega, \{\mathbf{W}_\ell^{(t)}\}, $ NN.values$)$
12:     end for
13:     Update $\mathbf{W}_\ell{}^{(t+1)} \leftarrow \mathbf{W}_\ell{}^{(t)} - \eta$ mean $(\mathbf{G}_{\ell,1}, \ldots, \mathbf{G}_{\ell,\Omega}) \leftarrow$ one iteration
14:     Update $\xi \leftarrow \xi + 1$                    # go for next mini-batch
15: end while
```

Few Definitions: *Epoch and Iteration*

We can consider a simple example: *say we train our FNN over MNIST using mini-batch SGD with batch size $\Omega = 100$. Our training dataset has $60,000$ data-points; thus, we have*

$$\Xi = \frac{60,000}{100} = 600$$

mini-batches. Each time we finish with a mini-bacth, we do one iteration of gradient descent. After 600 iterations, we finish with a single epoch

So, if we have trained the FNN for 10 epochs, it means that

we have done $600 \times 10 = 6000$ iterations of gradient descent

Testing NNs with New Data-Point

+ *Say we are over with the training; then, what should we do?*
– We need to test it with the data we reserved for testing

After training, we need to test our trained NN: *say we get a new data-point $\boldsymbol{x}_{\rm new}$ with label $\boldsymbol{v}_{\rm new}$. We can test our NN for this new test data-point by evaluating classical metrics*

1. **Test Risk** also called **Test Loss**: *we pass $\boldsymbol{x}_{\rm new}$ forward through our trained NN and get $\boldsymbol{y}_{\rm new}$. We then calculate the test loss as $\mathcal{L}\left(\boldsymbol{y}_{\rm new}, \boldsymbol{v}_{\rm new}\right)$ using the same loss function $\mathcal{L}$ we used for training*
2. **Test Accuracy**: *we use $\boldsymbol{y}_{\rm new}$ to classify $\boldsymbol{x}_{\rm new}$. We then compare it to the true class of $\boldsymbol{x}_{\rm new}$. If they are the same; then, the test accuracy is 1, if not, it is 0*

Testing NNs over Test Dataset

Testing for a single new point is not reliable: *this is why we had reserved the test dataset*.

Given the test dataset, we go through every single test data-point

- *we pass the data-point forward through the trained NN*
- *we compute the test loss and test accuracy*
- *we average them over the whole test dataset*

Therefore, we get

- *an average loss that approximates the risk*
- *a test accuracy between 0 and 1 that says how accurate our trained NN is*

Learning Curves

+ *What you said gives us two numbers! But, I have seen curves!*
– Yes! They are learning curves

In practice, the SGD can take very long to converge, i.e., to stop iterating

it needs too many iterations to get too close to the minimum

But, it might be not really needed to get that close! So,

we test our NN once every epoch

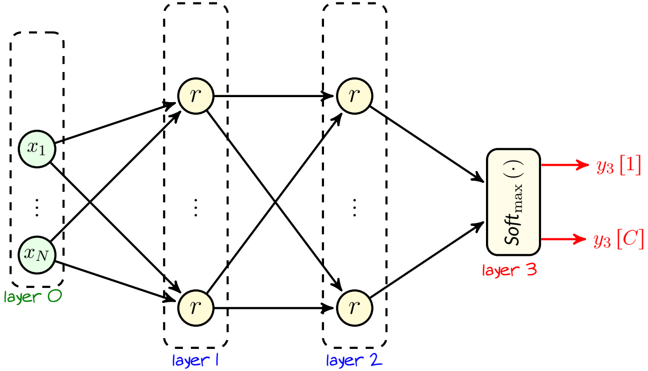
We then plot the test risk and test accuracy against number of epochs in a curve: *these curves are often called learning curves*

if we see that learning curves are saturating, we can stop the training

In practice: $\boxed{\textit{we always perform the training for a fixed number of epochs}}$

Learning Curves: *Example*

Let's see an example: *recall our three-layer FNN. Say, we train it for image classification over MNIST which has 60,000 data-points for and 10,000 for test*



In MNIST, we have 10 classes, so $C = 10$*. We use cross-entropy as loss function*

Learning Curves: *Example*

We agree to do the following: *we use mini-batch SGD with batch size* $\Omega = 100$ *and train the FNN for 100 epochs*.

In epoch $\xi = 1, \ldots, 100$

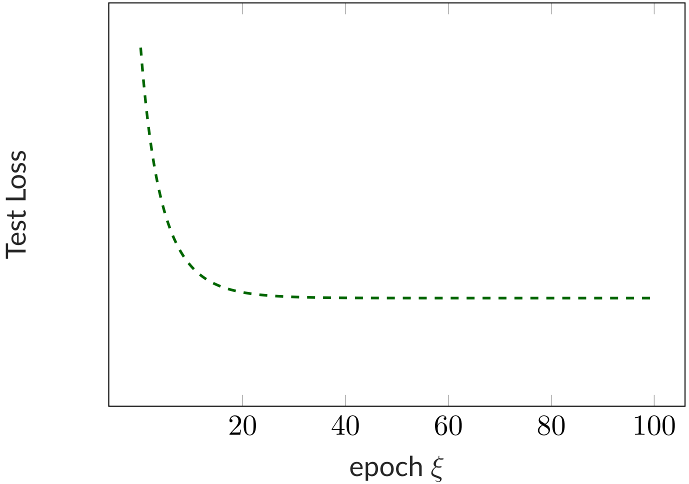
1. *we perform* 600 *iterations of gradient descent*
2. *we fix the weights to what we computed at last iteration of the epoch*
3. *for each test data-point:* we pass it forward and determine $\mathbf{y}_3$
    1. we compute $\mathrm{CE}\left(\mathbf{y}_3, \mathbf{1}_v\right)$, where v is the true class of test data-point
    2. we find the index of maximum term in $\mathbf{y}_3$ and compare it to v
        - ↳ *if they are the same, we set accuracy to* 1; *otherwise, we set it* 0
4. *we average test loss and accuracy*

Now, for each epoch

we have a test loss and test accuracy: *we plot them against* ξ

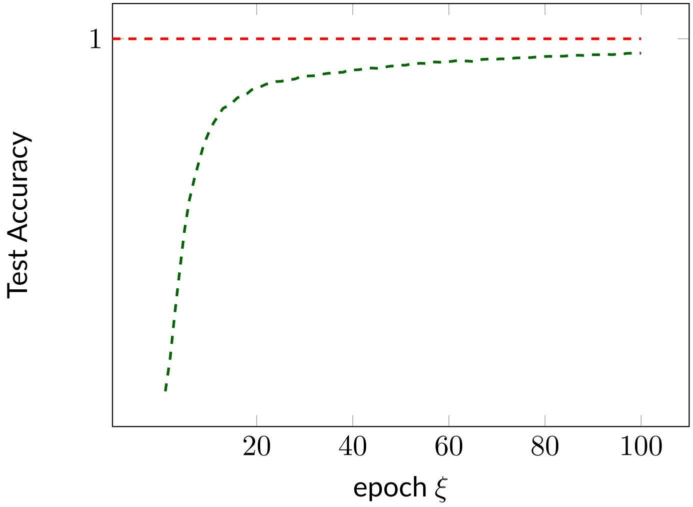
Learning Curves: *Example*

How should the learning curves look? A typical curve for test loss is



Learning Curves: *Example*

How should the learning curves look? A typical curve for test accuracy is



Summary of This Chapter

- To train a NN we need *gradients*
  - ↳ *We can calculate gradient by forward and backpropagation over the NN*
  - ↳ *In FNNs, forward propagation uses simple linear and nonlinear operations*
  - ↳ *Backpropagation is readily derived using computation graph*
- We tried Classification via FNNs
  - ↳ *Better to work with probabilities instead of exact labels*
  - ↳ *For multiclass classification, we should use vector-activated neurons*
- To minimize the exact empirical risk, we have to do full-batch training
  - ↳ *This requires huge computation complexity*
  - ↳ *We can hugely reduce this cost by SGD which does sample-level training*
  - ↳ *SGD versus full-batch describes a complexity-accuracy trade-off*
  - ↳ *We can tune this trade-off by mini-batch SGD*