

Deep Learning (Fall 2023)

Ikbeom Jang

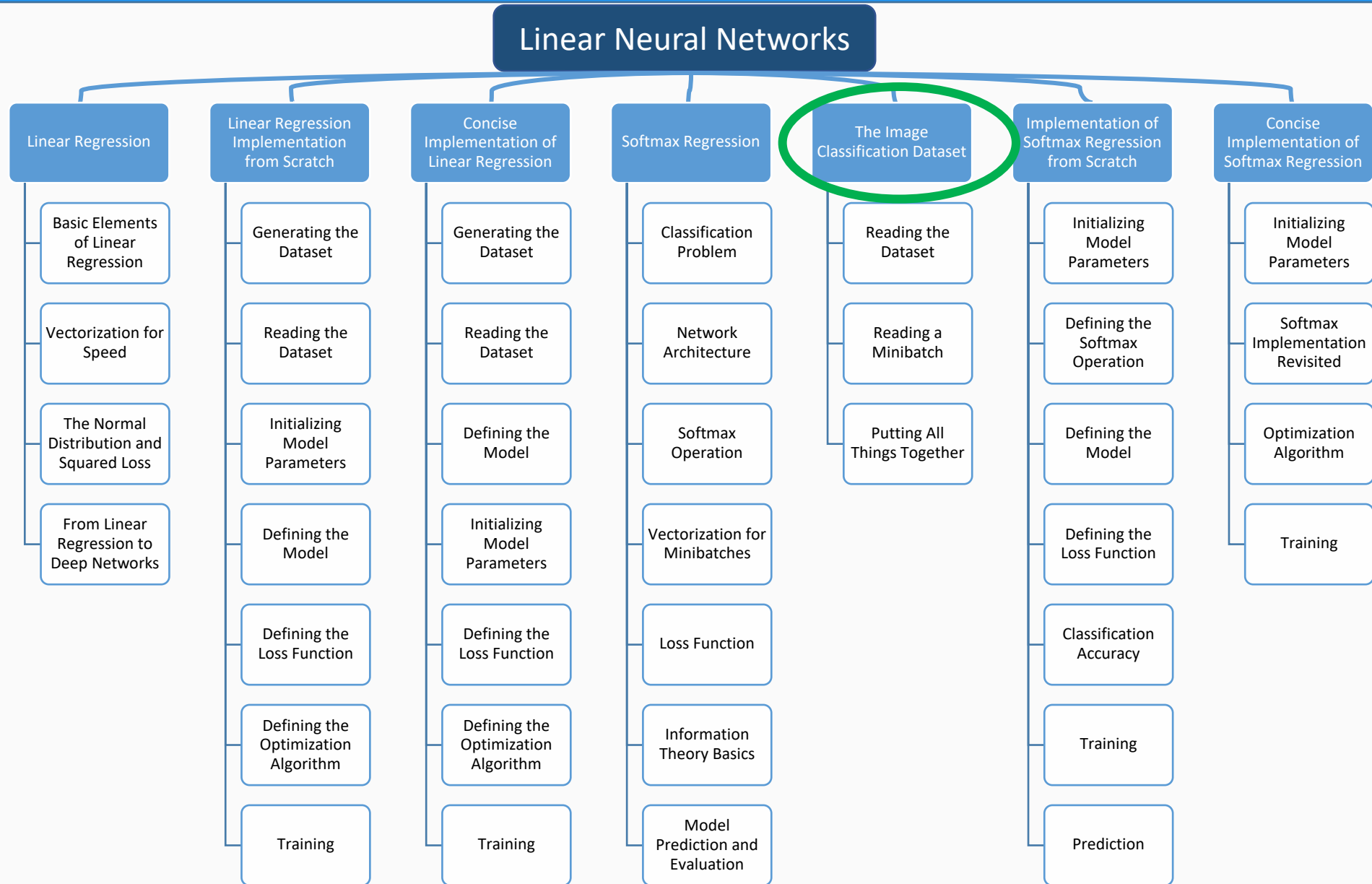
ijang@hufs.ac.kr

CES HUFS

Linear Neural Networks

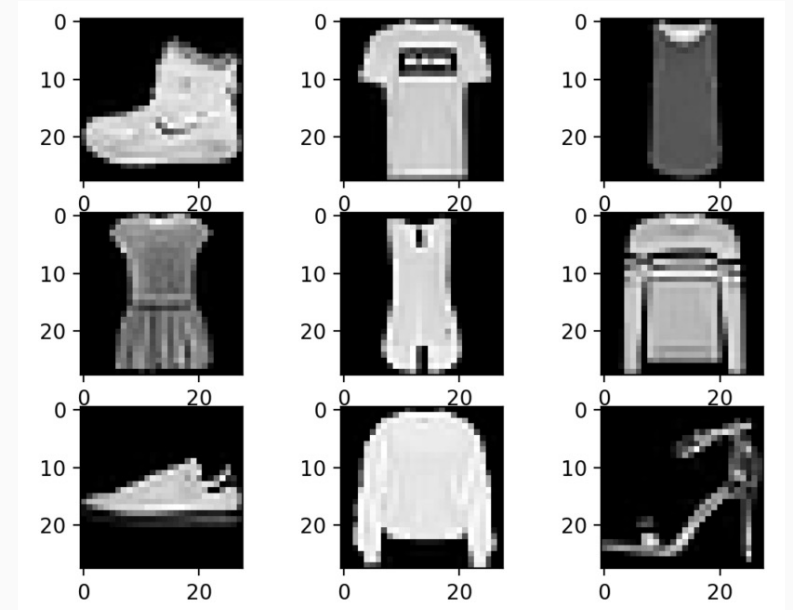
- **The Image Classification Dataset**
- **The Base Classification Model**

Contents



The Image Classification Dataset

- One of the widely used dataset for image classification is the MNIST dataset.
 - Simple models achieve classification accuracy over 95%, making it unsuitable for distinguishing between stronger models and weaker ones.
- We will focus on the qualitatively similar, but comparatively complex Fashion-MNIST dataset.



```
%matplotlib inline
import time
import torch
import torchvision
from torchvision import transforms
from d2l import torch as d2l

d2l.use_svg_display()
```

loss function 0.2
softmax 0.2

linear-classification-dataset

Loading the Dataset

- We can download and read the Fashion-MNIST dataset into memory via the built-in functions in the framework.

```
class FashionMNIST(d2l.DataModule): #@save
    """The Fashion-MNIST dataset."""
    def __init__(self, batch_size=64, resize=(28, 28)):
        super().__init__()
        self.save_hyperparameters()
        trans = transforms.Compose([transforms.Resize(resize), transforms.ToTensor()])
        self.train = torchvision.datasets.FashionMNIST(root=self.root, train=True, transform=trans, download=True)
        self.val = torchvision.datasets.FashionMNIST(root=self.root, train=False, transform=trans, download=True)
```

resize → ToTensor()
훈련 데이터를 어떻게 변환할 것인지
self.val =
self.val =

- Fashion-MNIST consists of images from 10 categories, each represented by 60000 images in the training dataset and by 10000 in the test dataset.
 - A test dataset (or test set) is used for evaluating model performance and not for training.
 - Consequently the training set and the test set contain 60000 and 10000 images, respectively.

```
data = FashionMNIST(resize=(32, 32))
len(data.train), len(data.val)
```

```
(60000, 10000)
```

Loading the Dataset

- The images are originally 28 x 28 pixels in resolution.
 - The dataset consists of grayscale images, whose number of channels is 1.
 - They upscaled to 32 x 32 pixels

```
data.train[0][0].shape
```

```
torch.Size([1, 32, 32])
```

- The categories of Fashion-MNIST have human-understandable names.
 - The following convenience method converts between numeric labels and their names.

```
@d2l.add_to_class(FashionMNIST) #@save
def text_labels(self, indices):
    """Return text labels."""
    labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [labels[int(i)] for i in indices]
```

Reading a Minibatch

- We use the built-in data iterator rather than creating one from scratch.
 - At each iteration, a data loader reads a minibatch of data with size *batch_size* each time.
 - We also randomly shuffle the examples for the training data iterator.

```
@d2l.add_to_class(FashionMNIST) #@save
def get_dataloader(self, train):
    data = self.train if train else self.val
    return torch.utils.data.DataLoader(data, self.batch_size, shuffle=train, num_workers=self.num_workers)
```

- Let's load a minibatch of images using *train_dataloader*

```
X, y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, y.shape, y.dtype)
```

```
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

Reading a Minibatch

- Let's look at the time it takes to read the training data.

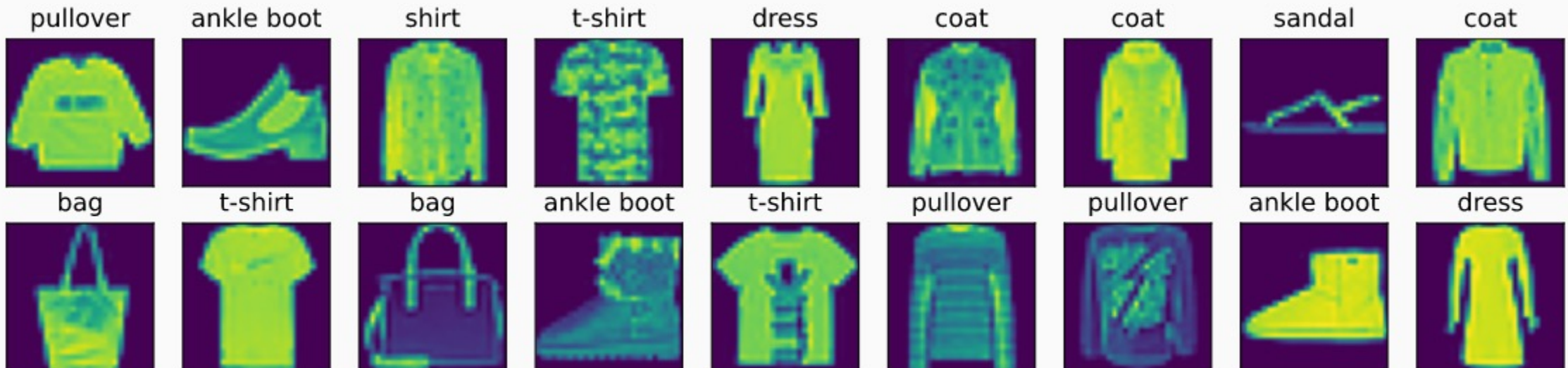
```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

```
'4.69 sec'
```


Visualization

- In general, it is a good idea to visualize and inspect data that you are training on.
- A convenience function *show_images* can be used to visualize the images and the associated labels.

```
@d2l.add_to_class(FashionMNIST) #@save
def visualize(self, batch, nrows=1, ncols=8, labels=[]):
    X, y = batch
    if not labels:
        labels = self.text_labels(y)
    d2l.show_images(X.squeeze(1), nrows, ncols, titles=labels)
batch = next(iter(data.val_dataloader()))
data.visualize(batch)
```

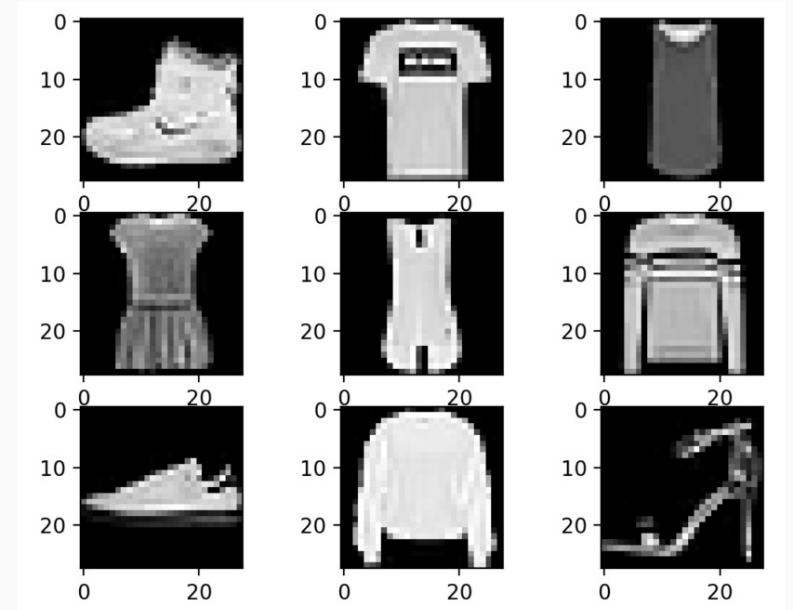


Summary

- Fashion-MNIST is an apparel classification dataset consisting of images representing 10 categories. We will use this dataset in subsequent sections and chapters to evaluate various classification algorithms.
- We store the shape of any image with height h width w pixels as $h \times w$ or (h, w) .
- Data iterators are a key component for efficient performance. Rely on well-implemented data iterators that exploit high-performance computing to avoid slowing down your training loop.

The Base Classification Model

- We saw that the implementations from scratch and the concise implementation using framework functionality were quite similar in the case of regression.
- The same is true for classification.
- This section provides a base class for classification models to simplify future code.



```
import torch
from d2l import torch as d2l
```

The Classifier Class

- We define the Classifier class below.
- In the *validation_step* we report both the loss value and the classification accuracy on a validation batch.
- We draw an update for every *num_val_batches* batches.
 - This has the benefit of generating the averaged loss and accuracy on the whole validation data.
 - These average numbers are not exactly correct if the final batch contains fewer examples, but we ignore this minor difference to keep the code simple.

```
class Classifier(d2l.Module): #@save
    """The base class of classification models."""
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        self.plot('loss', self.loss(Y_hat, batch[-1]), train=False)
        self.plot('acc', self.accuracy(Y_hat, batch[-1]), train=False)
```

- By default, we use a stochastic gradient descent optimizer, operating on minibatches, just as we did in the context of linear regression.

```
@d2l.add_to_class(d2l.Module) #@save
def configure_optimizers(self):
    return torch.optim.SGD(self.parameters(), lr=self.lr)
```

The Classifier Class

- Given the predicted probability distribution $y_{\hat{}}$, we typically choose the class with the highest predicted probability whenever we must output a hard prediction.
- Many applications require that we make a choice.
 - For instance, Gmail must categorize an email into “Primary”, “Social”, “Updates”, “Forums”, or “Spam”.
 - It might estimate probabilities internally, but at the end it has to choose one among the classes.
- When predictions are consistent with the label class y , they are correct.
 - The classification accuracy is the fraction of all predictions that are correct.
- Although it can be difficult to optimize accuracy directly (it is not differentiable), it is often the performance measure that we care about the most.
 - As such, we will nearly always report it when training classifiers.

The Classifier Class

- Accuracy is computed as follows.
 - First, if y_hat is a matrix, we assume that the second dimension stores prediction scores for each class.
 - We use `argmax` to obtain the predicted class by the index for the largest entry in each row.
 - Then we compare the predicted class with the ground truth y elementwise.
 - Since the equality operator `==` is sensitive to data types, we convert y_hat 's data type to match that of y .
 - The result is a tensor containing entries of 0 (false) and 1 (true).
 - Taking the sum yields the number of correct predictions.

```
@d2l.add_to_class(Classifier) #@save
def accuracy(self, Y_hat, Y, averaged=True):
    """Compute the number of correct predictions."""
    Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
    preds = Y_hat.argmax(axis=1).type(Y.dtype)
    compare = (preds == Y.reshape(-1)).type(torch.float32)
    return compare.mean() if averaged else compare
```

Summary

- Classification is a sufficiently common problem that it warrants its own convenience functions.
- Of central importance in classification is the accuracy of the classifier.
- Note that while we often care primarily about accuracy, we train classifiers to optimize a variety of other objectives for statistical and computational reasons.
- Regardless of which loss function was minimized during training, it is useful to have a convenience method for assessing the accuracy of our classifier empirically.