


```
!pip install d2l==1.0.3
```

 Show hidden output


✓ COSE474-2024F: Deep Learning

4.4. Softmax Regression Implementation from Scratch

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

✓ 4.4.1. The Softmax

```
X = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
X.sum(0, keepdims=True), X.sum(1, keepdims=True)
```

 (tensor([[5., 7., 9.]]),
tensor([[6.],
[15.]])

Computing the softmax requires three steps: (i) exponentiation of each term; (ii) a sum over each row to compute the normalization constant for each example; (iii) division of each row by its normalization constant, ensuring that the result sums to 1:


$$\text{softmax}(\mathbf{X})_{ij} = \frac{\exp(\mathbf{X}_{ij})}{\sum_k \exp(\mathbf{X}_{ik})}.$$

The (logarithm of the) denominator is called the (log) *partition function*. It was introduced in [statistical physics](#) to sum over all possible states in a thermodynamic ensemble. The implementation is straightforward:

```
def softmax(X):
    X_exp = torch.exp(X)
    partition = X_exp.sum(1, keepdims=True)
    return X_exp / partition # The broadcasting mechanism is applied here
```

For any input X , we turn each element into a nonnegative number. Each row sums up to 1, as is required for a probability. Caution: the code above is not robust against very large or very small arguments. While it is sufficient to illustrate what is happening, you should not use this code verbatim for any serious purpose. Deep learning frameworks have such protections built in and we will be using the built-in softmax going forward.

```
X = torch.rand((2, 5))
X_prob = softmax(X)
X_prob, X_prob.sum(1)
```

 (tensor([[0.2247, 0.1639, 0.1748, 0.1561, 0.2804],
[0.1595, 0.1795, 0.2647, 0.1535, 0.2428]]),
tensor([1., 1.]))

✓ 4.4.2. The Model

We now have everything that we need to implement **[the softmax regression model.]** As in our linear regression example, each instance will be represented by a fixed-length vector. Since the raw data here consists of 28×28 pixel images, **[we flatten each image, treating them as vectors of length 784.]** In later chapters, we will introduce convolutional neural networks, which exploit the spatial structure in a more satisfying way.

In softmax regression, the number of outputs from our network should be equal to the number of classes. **(Since our dataset has 10 classes, our network has an output dimension of 10.)** Consequently, our weights constitute a 784×10 matrix plus a 1×10 row vector for the biases. As with linear regression, we initialize the weights W with Gaussian noise. The biases are initialized as zeros.

```
class SoftmaxRegressionScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W = torch.normal(0, sigma, size=(num_inputs, num_outputs),
                                requires_grad=True)
        self.b = torch.zeros(num_outputs, requires_grad=True)

    def parameters(self):
        return [self.W, self.b]
```

Note that we flatten each 28×28 pixel image in the batch into a vector using `reshape` before passing the data through our model.

```
@d2l.add_to_class(SoftmaxRegressionScratch)
def forward(self, X):
    X = X.reshape((-1, self.W.shape[0]))
    return softmax(torch.matmul(X, self.W) + self.b)
```

✓ 4.4.3. The Cross-Entropy Loss

This may be the most common loss function in all of deep learning. At the moment, applications of deep learning easily cast as classification problems far outnumber those better treated as regression problems.

Recall that cross-entropy takes the negative log-likelihood of the predicted probability assigned to the true label. For efficiency we avoid Python for-loops and use indexing instead. In particular, the one-hot encoding in \mathbf{y} allows us to select the matching terms in $\hat{\mathbf{y}}$.

To see this in action we **[create sample data $\mathbf{y_hat}$ with 2 examples of predicted probabilities over 3 classes and their corresponding labels \mathbf{y} .]** The correct labels are 0 and 2 respectively (i.e., the first and third class). **[Using \mathbf{y} as the indices of the probabilities in $\mathbf{y_hat}$,]** we can pick out terms efficiently.

```
y = torch.tensor([0, 2])
y_hat = torch.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y_hat[[0, 1], y]
# tensor([0.1000, 0.5000])

def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()

cross_entropy(y_hat, y)
# tensor(1.4979)

@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
    return cross_entropy(y_hat, y)
```

✓ 4.4.4. Training

```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```

4.4.5. Prediction

```

| 1 | train loss |
X, y = next(iter(data.val_dataloader()))
preds = model(X).argmax(axis=1)
preds.shape

```

```

torch.Size([256])

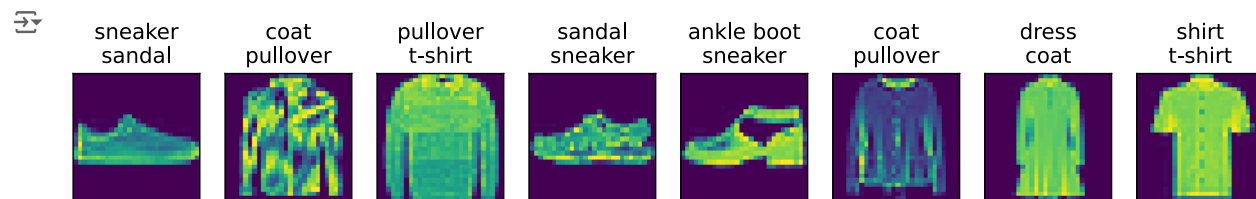
```

We are more interested in the images we label incorrectly. We visualize them by comparing their actual labels (first line of text output) with the predictions from the model (second line of text output).

```

wrong = preds.type(y.dtype) != y
X, y, preds = X[wrong], y[wrong], preds[wrong]
labels = [a+'\n'+b for a, b in zip(
    data.text_labels(y), data.text_labels(preds))]
data.visualize([X, y], labels=labels)

```



4.4.6. Summary

By now we are starting to get some experience with solving linear regression and classification problems. With it, we have reached what would arguably be the state of the art of 1960–1970s of statistical modeling. In the next section, we will show you how to leverage deep learning frameworks to implement this model much more efficiently.

Start coding or [generate](#) with AI.