Softmax Regression

Asking not "how much" but "which one"

- Does this email belong in the spam folder or the inbox?
- Is this customer more likely **to sign up** or **not to sign up** for a subscription service?
- Does this image depict a donkey, a dog, a cat, or a rooster?
- Which movie is Aston most likely to watch next?

Classification

- Interested only in **hard assignments** of examples to categories (classes)
- Wish to make **soft assignments**, e.g. to assess the **probability** that each category applies

Classification Problem

2 x 2 grayscale image

x_1	x_2
x_3	x_4

Each pixel value with a single scalar, **four features**

Label

- How to represent the labels {dog, cat, chicken}?
- $y \in \{1, 2, 3\}$
- $y \in \{(1,0,0), (0,1,0), (0,0,1)\}$ **"one-hot encoding"**

Network Architecture

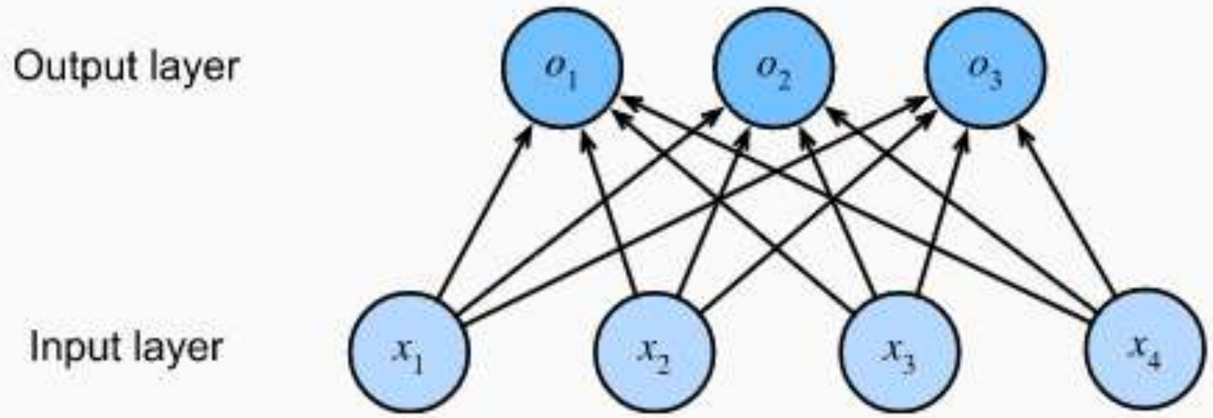


Fig. 3.4.1 Softmax regression is a single-layer neural network.

Affine function computing **logits**

$$o_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1$$
$$o_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2$$
$$o_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3$$

or

$$\boldsymbol{o} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$ (Vector form)

Softmax Operation

- Interpret the outputs of our model as **probabilities**
- Any output $\hat{y_j}$ to be interpreted as the **probability** that a given item belongs to **class** j
- Can choose the class with the **largest** output value as **our prediction** $argmax_j y_j$
- E.g. $\widehat{y_1}, \widehat{y_2}$ and $\widehat{y_3}$ are 0.1, 0.8, 0.1, then predict category 2

Why not to use logits directly as our outputs?

- Nothing constrains these numbers to **sum to 1**
- It can take **negative** values (violate basic axioms of probability)
- To transform our logits such that they become **nonnegative** and **sum to 1**, while requiring **differentiable**

$$\hat{\boldsymbol{y}} = softmax(\boldsymbol{o}) \quad where \quad \hat{y_j} = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$

Loss Function

Log-Likelihood

- $\hat{\boldsymbol{y}}$, can interpret as estimated **conditional probabilities** of each class **given any input** x
- e.g. $\widehat{y_1} = P(y = cat|x)$
- Suppose that the entire dataset {X, Y} has n examples

$$P(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{i=1}^{n} P(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)})$$

- Maximizing P(Y|X) is equivalent to **minimizing negative log-likelihood**

$$-logP(\boldsymbol{Y}|\boldsymbol{X}) = \sum_{i=1}^{n} -logP\left(\boldsymbol{y}^{(i)}\middle|\boldsymbol{x}^{(i)}\right) = \sum_{i=1}^{n} l(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)})$$

$$l(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) = -\sum_{j=1}^{q} y_j log\hat{y}_j$$

$$P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}) = \prod_j [P(y_j^{(i)} \mid \mathbf{x}^{(i)})]^{y_j} = \prod_j \hat{y}_j^{y_j}$$

Softmax and Derivatives

Digging equation

$$l(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) = -\sum_{j=1}^{q} y_j log \hat{y}_j$$

Cross-entropy loss

$$l(y, \hat{y}) = -\sum_{j=1}^{q} y_j log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)}$$

$$= \sum_{j=1}^{q} y_j log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j \, o_j$$

$$= log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j$$

$$\partial_{oj} l(y, \hat{y}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = softmax(\boldsymbol{o})_j - y_j$$

Derivative is the difference between the **probability** and **one-hot label**

Information Theory Basics

Entropy

- $H[P] = \sum_j -P(j) log P(j)$
- Quantify the information content in data (Hard limit to compress the data)
- Encode data from distribution P, need at least H[P] to encode it

Surprisal

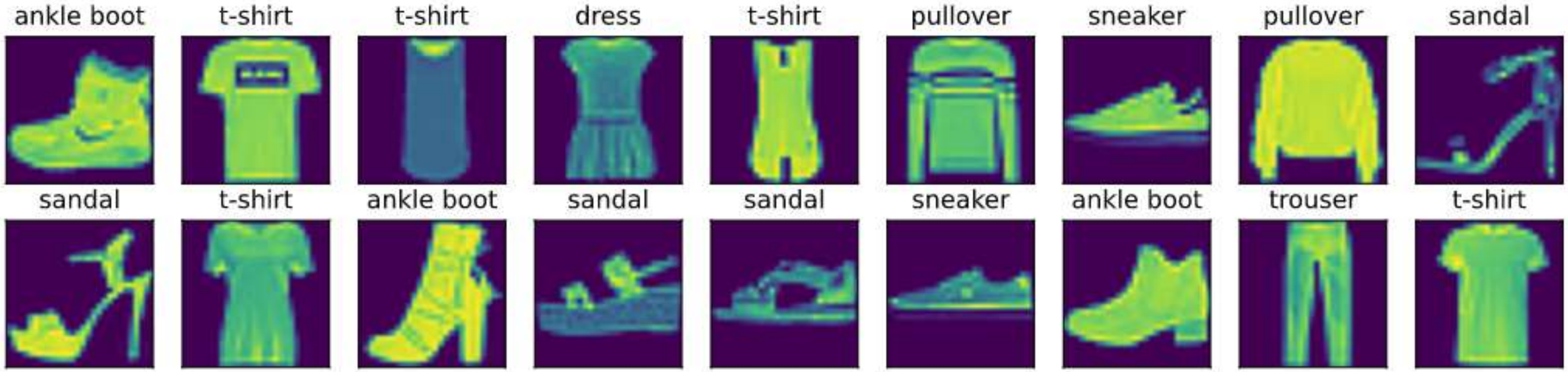
- Stream of data that we want to compress (Easy to predict, easy to compress)
- We can’t predict every event perfectly, then we might sometimes be surprised
- **Surprise is greater** when we assigned an **event lower probability**
- $log \frac{1}{P(j)} = -log P(j)$ to quantify one’s surprisal
- Entropy can be expressed as expected surprisal

Model Prediction and Evaluation

- After training the softmax regression model, given any example features, we can predict the probability
- Prediction is correct if it is **consistent with** the **actual class** (label)
- Accuracy, evaluation of model's performance
  - ratio between **the number of correct predictions** and **the total number of predictions**

The Image Classification Dataset

Fashion MNIST



- Even simple model achieve classification accuracy over 95% for MNIST dataset
- Fashion-MNIST consists of images from 10 categories
- Training set and test set contain 60000 and 10000 images respectively

```
mnist_train, mnist_test = tf.keras.datasets.fashion_mnist.load_data()
```

Implementation from Scratch

Initializing Model Parameters

```
num_inputs = 784
num_outputs = 10

W = tf.Variable(tf.random.normal(shape=(num_inputs, num_outputs),
                                 mean=0, stddev=0.01))
b = tf.Variable(tf.zeros(num_outputs))
```

Defining the Softmax Operation

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

$$\hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$

Implementation from Scratch

Defining the Model

```
def net(X):
    return softmax(tf.matmul(tf.reshape(X, (-1, W.shape[0])), W) + b)
```

Defining the Loss Function

```
y_hat = tf.constant([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y = tf.constant([0, 2])
tf.boolean_mask(y_hat, tf.one_hot(y, depth=y_hat.shape[-1]))
```

```
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([0.1, 0.5], dtype=float32)>
```

```
def cross_entropy(y_hat, y):
    return -tf.math.log(tf.boolean_mask(
        y_hat, tf.one_hot(y, depth=y_hat.shape[-1])))

cross_entropy(y_hat, y)
```

$$l(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) = -\sum_{j=1}^{q} y_j log \hat{y}_j$$

Implementation from Scratch

Training

```
def train_epoch_ch3(net, train_iter, loss, updater):  #@save
    """The training loop defined in Chapter 3."""
    # Sum of training loss, sum of training accuracy, no. of examples
    metric = Accumulator(3)
    for X, y in train_iter:
        # Compute gradients and update parameters
        with tf.GradientTape() as tape:
            y_hat = net(X)
            # Keras implementations for loss takes (labels, predictions)
            # instead of (predictions, labels) that users might implement
            # in this book, e.g. `cross_entropy` that we implemented above
            if isinstance(loss, tf.keras.losses.Loss):
                l = loss(y, y_hat)
            else:
                l = loss(y_hat, y)
        if isinstance(updater, tf.keras.optimizers.Optimizer):
            params = net.trainable_variables
            grads = tape.gradient(l, params)
            updater.apply_gradients(zip(grads, params))
        else:
            updater(X.shape[0], tape.gradient(l, updater.params))
        # Keras loss by default returns the average loss in a batch
        l_sum = l * float(tf.size(y)) if isinstance(
            loss, tf.keras.losses.Loss) else tf.reduce_sum(l)
        metric.add(l_sum, accuracy(y_hat, y), tf.size(y))
    # Return training loss and training accuracy
    return metric[0] / metric[2], metric[1] / metric[2]
```

Concise Implementation

High-level APIs (TF2)

```
net = tf.keras.models.Sequential()
net.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

weight_initializer = tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.01)
net.add(tf.keras.layers.Dense(10, kernel_initializer=weight_initializer))
```

```
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
trainer = tf.keras.optimizers.SGD(learning_rate=.1)
```