

الطبقة الأولى  $\rightarrow$  طبقة خروج، جميع نورونات طبقة خروج  $\rightarrow$  طبقة دارم  $\rightarrow$  طبقة إفراست  $\rightarrow$  طبقة تردد خرج.

label	$id = y_{target}$
A	1
B	0

$\rightarrow$



$$A = x_1, x_2, x_3, x_4 \\ = 1, 0, 0, 0$$

$$B = x_1, x_2, x_3, x_4 \\ = 0 \ 1 \ 0 \ 1$$

الدالة  $f(x) = \frac{1}{1+e^{-x}}$ , learning rate  $\alpha$ ، خط انتقال  $\rightarrow$  خط انتقال

input	out	NET	target	weight change	weight
1, 0, 0, 0, 0, 1	0.128	1	1	( $\frac{1}{1+e^{-0.128}}$ , $\frac{\partial f}{\partial w_1}$ )	( $\frac{1}{1+e^{-0.128}}$ , $\frac{\partial f}{\partial w_1}$ )
0, 1, 0, 1, 1	0.757	0	0	( $0.757 - 0.757^2 - 0.757$ , $-0.757$ ) / ( $0.757 - 0.757^2 - 0.757$ )	( $0.757 - 0.757^2 - 0.757$ , $-0.757$ ) / ( $0.757 - 0.757^2 - 0.757$ )
1, 0, 0, 0, 0, 1	0.1294	1	1	( $0.1294 - 0.1294^2 - 0.1294$ ) / ( $0.1294 - 0.1294^2 - 0.1294$ )	( $0.1294 - 0.1294^2 - 0.1294$ ) / ( $0.1294 - 0.1294^2 - 0.1294$ )
0, 1, 0, 1, 0, 1	0.1291	0	0	( $0.1291 - 0.1291^2 - 0.1291$ ) / ( $0.1291 - 0.1291^2 - 0.1291$ )	( $0.1291 - 0.1291^2 - 0.1291$ ) / ( $0.1291 - 0.1291^2 - 0.1291$ )
1, 0, 0, 0, 0, 1	0.1290	1	1	( $0.1290 - 0.1290^2 - 0.1290$ ) / ( $0.1290 - 0.1290^2 - 0.1290$ )	( $0.1290 - 0.1290^2 - 0.1290$ ) / ( $0.1290 - 0.1290^2 - 0.1290$ )
0, 1, 0, 0, 1, 1	0.1298	0	0	( $0.1298 - 0.1298^2 - 0.1298$ ) / ( $0.1298 - 0.1298^2 - 0.1298$ )	( $0.1298 - 0.1298^2 - 0.1298$ ) / ( $0.1298 - 0.1298^2 - 0.1298$ )
1, 0, 0, 0, 0, 1	0.1299	1	1	( $0.1299 - 0.1299^2 - 0.1299$ ) / ( $0.1299 - 0.1299^2 - 0.1299$ )	( $0.1299 - 0.1299^2 - 0.1299$ ) / ( $0.1299 - 0.1299^2 - 0.1299$ )
0, 1, 0, 0, 1, 0	0.1294	0	0	( $0.1294 - 0.1294^2 - 0.1294$ ) / ( $0.1294 - 0.1294^2 - 0.1294$ )	( $0.1294 - 0.1294^2 - 0.1294$ ) / ( $0.1294 - 0.1294^2 - 0.1294$ )

$$\frac{\partial \hat{y}_{t,i}}{\partial o_t} = \frac{(-\sum_j^r y_{t,j} \log(\hat{y}_{t,i}))}{\partial y_{t,r}} * \frac{\partial y_{t,r}}{\partial o_t} = -\sum_i^r \left( \frac{y_{t,r}}{\hat{y}_{t,r} \ln(1.0)} \times f'(o_t) \right)$$

$$\frac{\partial j_t}{\partial h_2} = \frac{\partial j_t}{\partial y_{t,r}} * \frac{\partial \hat{y}_{t,r}}{\partial o_t} * \frac{\partial o_t}{\partial h_3} * \frac{\partial h_r}{\partial o_3} * \frac{\partial z_3}{\partial h_2} = -\sum_i^r \frac{y_{t,r}}{\hat{y}_{t,r} \ln(1.0)} *$$

$$f'(o_t) * w_{yh} * g'(z_r) * w_{hh}$$

$$\frac{\partial j_t}{\partial w_{hh}} = - \left( \frac{\partial j_t}{\partial y_{t,r}} * \frac{\partial \hat{y}_{t,r}}{\partial o_t} * \frac{\partial o_t}{\partial h_3} * \frac{\partial h_3}{\partial z_3} * \frac{\partial z_3}{\partial w_{hh}} \right) *$$

$$\frac{\partial J_t}{\partial \hat{y}_{t,r}} * \frac{\partial \hat{y}_{t,r}}{\partial o_t} * \frac{\partial o_t}{\partial h_3} * \frac{\partial h_3}{\partial z_3} * \frac{\partial z_3}{\partial h_2} * \frac{\partial h_r}{\partial w_{hh}} +$$

$$\frac{\partial j_t}{\partial y_{t,r}} * \frac{\partial \hat{y}_{t,r}}{\partial o_t} * \frac{\partial o_t}{\partial z_3} * \frac{\partial z_3}{\partial h_r} * \frac{\partial h_r}{\partial z_2} * \frac{\partial z_2}{\partial h_1} * \frac{\partial h_1}{\partial w_{hh}}$$

$$-\left( \frac{y_{t,r}}{\hat{y}_{t,r} \ln(1.0)} * f'(o_t) * w_{yh} * g'(z_r) * h_r + \frac{y_{t,r}}{\hat{y}_{t,r} \ln(1.0)} * f'(o_t) * w_{yh} * g'(z_r) * w_{hh} * g'(z_r) * w_{hh} * h_o \right)$$

$$\frac{\partial J_t}{\partial w_{hh}} = - \sum_i^r \left( \frac{\partial J_t}{\partial w_{hh}} \right) \rightarrow \sum_i^r \overline{w_{hh}}$$

### 3. پرسپترون (Perceptron)

- ساختار: پرسپترون یک مدل ساده شبکه عصبی با یک لایه است که تنها می‌تواند مسائل خطی را حل کند.

- قابلیت تعمیم: به دلیل محدودیت در حل مسائل خطی، قابلیت تعمیم پرسپترون کمتر است. اگردادهای غیرخطی باشند، پرسپترون نمی‌تواند آنها را به درستی مدل کند.

:Adaline (Adaptive Linear Neuron)

- ساختار: Adaline مشابه پرسپترون است اما ازتابع هزینه مربعات خطأ استفاده می‌کند و با استفاده از الگوریتم گرادیان نزولی به روزرسانی می‌شود.

- قابلیت تعمیم: اگرچه Adaline بهبودهای نسبت به پرسپترون دارد، اما همچنان محدود به مسائل خطی است. بنابراین، قابلیت تعمیم آن نیز نسبت به مدل‌های پیچیده‌تر کمتر است.

:Madaline (Multiple Adaline)

- ساختار: Madaline از چندین واحد Adaline در یک لایه مخفی استفاده می‌کند و می‌تواند مسائل غیرخطی را نیز مدل کند.

- قابلیت تعمیم: Madaline به دلیل داشتن لایه‌های مخفی و استفاده از چندین واحد Adaline قابلیت تعمیم بهتری نسبت به پرسپترون و Adaline دارد. اما همچنان محدودیت‌هایی دارد و پیچیدگی آن کمتر از MLP است.

:MLP (Multilayer Perceptron)

- ساختار: MLP یک شبکه عصبی چند لایه است که حداقل یک لایه مخفی دارد و از توابع فعال‌سازی غیرخطی مانند سیگموئید یا ReLU استفاده می‌کند.

- قابلیت تعمیم: MLP به دلیل داشتن چندین لایه و توابع فعال‌سازی غیرخطی، قابلیت تعمیم بسیار بهتری نسبت به پرسپترون، Adaline و حتی Madaline دارد. این مدل می‌تواند روابط پیچیده و غیرخطی در داده‌ها را به خوبی مدل کند و بنابراین در مسائل پیچیده‌تر عملکرد بهتری دارد.

نتیجه‌گیری:

- بیشترین قابلیت تعمیم: MLP به دلیل ساختار چندلایه و استفاده از توابع غیرخطی، بیشترین قابلیت تعمیم را دارد.

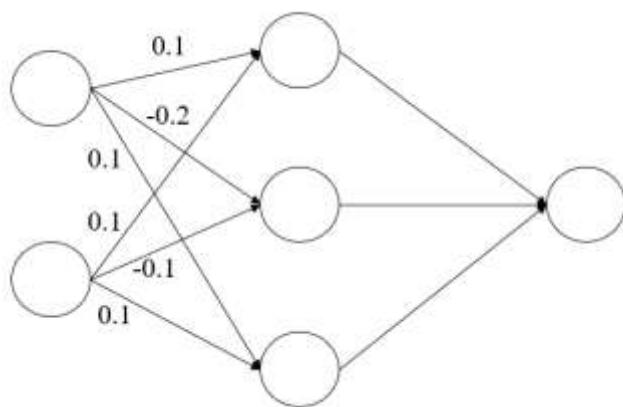
- کمترین قابلیت تعمیم: پرسپترون به دلیل محدودیت در حل مسائل خطی، کمترین قابلیت تعمیم را دارد. بنابراین، برای مسائل پیچیده و غیرخطی، استفاده از MLP توصیه می‌شود، در حالی که پرسپترون تنها برای مسائل ساده و خطی مناسب است.

4. میتوان از استفاده از یک شبکه عصبی پیشخور ساده (یک پرسپترون) برای طبقه بندی باینری استفاده کنیم.

ورودی ها : دایره:  $(0,0)$ ,  $(1,0)$ ,  $(1,1)$ ,  $(2,1)$  و مرربع:  $(0,2)$ ,  $(1,2)$

خروجی ها : دایره: 0 و مرربع: 1

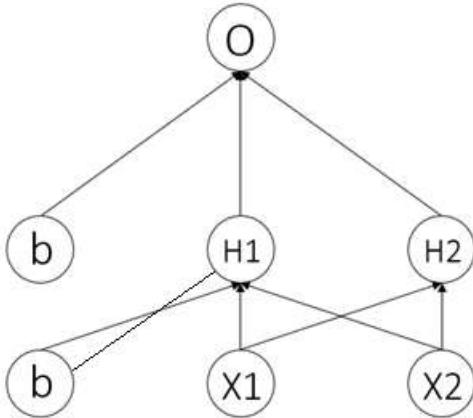
شبکه به شکل زیر است که دو نورون ورودی سه نورون مخفی و یک نورون خروجی دارد.



لایه مخفی از ReLU و لایه خروجی از sigmoid استفاده میکند. همچنین وزن ها به صورت تصادفی انتخاب می شوند و در طی فرآیند آموزش به روزرسانی می شوند. و نمونه ای از این وزن ها در تصویر بالا قابل مشاهده است.

آموزش شبکه هم به شکل backpropagation است که و با استفاده از تابع هزینه و بهینه سازی این کار را انجام میدهیم.

5. توضیحات مدل پیاده سازی شده زیر :



ورودی های  $X_1$  و  $X_2$  با وزن های مربوطه  $W_1$  تا  $W_4$  به لایه مخفی ارسال می شوند و خروجی های  $H_1$  و  $H_2$  با استفاده از تابع سیگموید محاسبه می شوند.  
خروجی های  $H_1$  و  $H_2$  با وزن های مربوطه  $W_5$  و  $W_6$  به لایه خروجی ارسال می شوند و خروجی نهایی  $O$  با استفاده از تابع سیگموید محاسبه می شود.

بعد از محاسبه می وزن ها در لایه های مخفی و خروجی(Feedforward)، باید مراحل Backpropagation) را انجام دهیم:

که ابتدا خطاهای را محاسبه کرده و بعد گرادیان خروجی و لایه مخفی را بدست آورده و سپس وزن ها را آپدیت میکنیم. این مراحل برای هر نمونه داده در مجموعه داده های آموزش تکرار می شوند. این مراحل برای تعداد مشخصی از دوره ها (epochs) تکرار می شوند تا زمانی که مدل به دقت مطلوبی برسد. در پایان، مدل قادر به پیش بینی خروجی های تابع XOR با دقت بالای خواهد بود.

توضیحات کد :

در اینجا ورودی و خروجی هایی که از مدل انتظار داریم، همچنین تعداد نورون های لایه مخفی و خروجی و ورودی را تعریف میکنیم :

```

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

input_layer_neurons = X.shape[1]
hidden_layer_neurons = 2
output_neurons = 1
    
```

سپس به مدل وزن های رندوم داده :

```
np.random.seed(42)
hidden_weights = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
hidden_bias = np.random.uniform(size=(1, hidden_layer_neurons))
output_weights = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
output_bias = np.random.uniform(size=(1, output_neurons))
```

در ادامه برای 1000 epoch گرادیان هارا بحسب آورده و وزن ها را اپدیت میکنیم :

```
learning_rate = 0.1
epochs = 10000

for _ in range(epochs):
    hidden_layer_activation = np.dot(X, hidden_weights) + hidden_bias
    hidden_layer_output = sigmoid(hidden_layer_activation)

    output_layer_activation = np.dot(hidden_layer_output, output_weights) + output_bias
    predicted_output = sigmoid(output_layer_activation)

    error = y - predicted_output

    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = d_predicted_output.dot(output_weights.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    output_weights += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    output_bias += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    hidden_weights += X.T.dot(d_hidden_layer) * learning_rate
    hidden_bias += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate
```

مزیت مدل NumPy : ساده تر و قابل فهم برای آموزش و یادگیری مفاهیم پایه شبکه های عصبی. مزیت مدل TensorFlow : استفاده از بهینه ساز های پیچیده تر مانند Adam که معمولاً به نتایج بهتری منجر می شود. همچنین، قابلیت های بیشتری برای توسعه و پیاده سازی مدل های پیچیده تر دارد.

این ساختار در هر دو پیاده سازی با NumPy و TensorFlow به کار گرفته شده است. در حالی که پیاده سازی با NumPy به صورت دستی انجام شده و جزئیات کامل مراحل فیدفوروارد و بک پراپگیشن در کد آورده شده، در پیاده سازی با TensorFlow از توابع کتابخانه ای برای انجام این مراحل استفاده شده است که باعث سهولت در کدنویسی و بهبود عملکرد مدل می شود.

6. توضیحات کد :

دیتاست CIFAR-10 شامل 60000 تصویر 32x32رنگی در 10 کلاس مختلف است که به دو بخش 50000 تایی برای آموزش و 10000 تایی برای آزمون تقسیم می شود.

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
```

شبکه ما شامل لایه‌های Dense می‌باشد که برای پردازش داده‌های تصویر به فرم یک بعدی شده استفاده می‌شوند.

لایه ورودی:

استفاده از flatten جهت تبدیل یک تنسور سه بعدی به یک بعدی به طول  $32 \times 32 \times 3$  است تا بتوان از آن در شبکه ای MLP استفاده کرد.

لایه‌های پنهان:

لایه‌های پنهان شامل 512، 256 و 128 نورون است و معمولاً از این ساختار استفاده می‌شود تا بتواند برای این دیتاست کار کند و پیچیدگی مناسبی دارد. 512 نورون: باعث ایجاد ظرفیت بالا می‌شود. 256 نورون: ظرفیت یادگیری خوبی دارد ولی پیچیدگی کمتر. 128 نورون: برای یادگیری جزئیات دقیق‌تر.

لایه خروجی:

دارای 10 نورون با فعال‌سازی softmax است که با 10 کلاس در مجموعه داده CIFAR-10 متناظر است.

```
model = models.Sequential()
model.add(layers.Flatten(input_shape=(32, 32, 3)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

سپس مدل را کامپایل کرده و بعد ترین میکنیم.

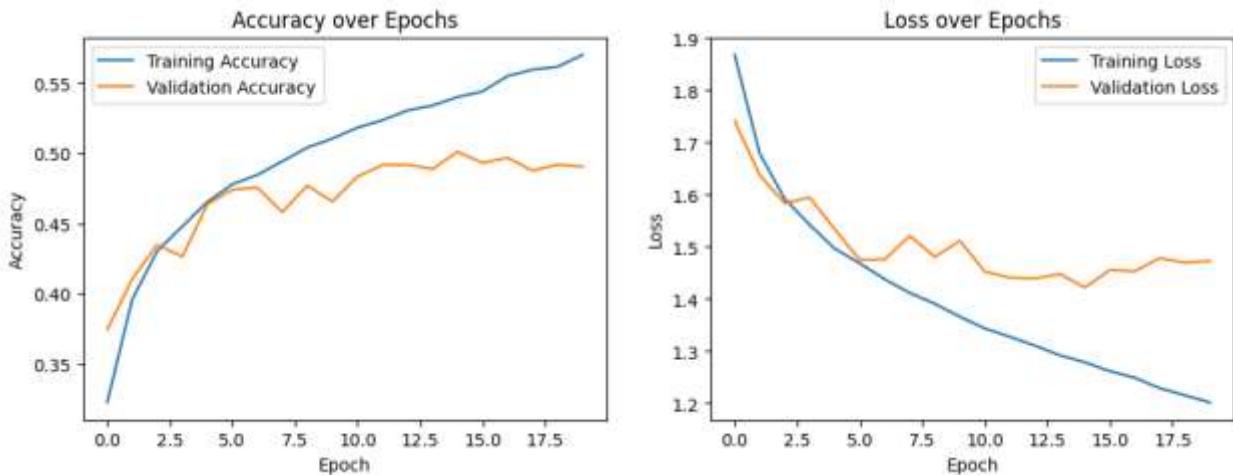
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=20,
                     validation_data=(test_images, test_labels))
```

نتیجه‌ی آموزش مدل با دقت زیر است :

```
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f"Accuracy: {test_accuracy * 100:.2f}%")
```

313/313 [=====] - 2s 5ms/step - loss: 1.4729 - accuracy: 0.4905



.7

داده‌های آموزشی و تست از فایل‌های CSV بارگذاری می‌شوند. ویژگی‌های آموزشی و متغیر هدف (قیمت فروش) از داده‌های آموزشی جدا می‌شوند.

```
features_train = train_df.drop(['SalePrice'], axis=1)
target_train = train_df['SalePrice']

features_test = test_df
```

داده‌های تست که شامل متغیر هدف نیستند، بارگذاری می‌شوند. و بعد ستون‌های عددی و دسته‌ای در داده‌های آموزشی شناسایی می‌شوند.

```
num_features = features_train.select_dtypes(include=['int64', 'float64']).columns.tolist()
cat_features = features_train.select_dtypes(include=['object']).columns.tolist()

num_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

یک پایپ‌لاین پیش‌پردازش برای داده‌های عددی ایجاد می‌شود که شامل پرکردن مقادیر گم شده با میانگین و مقیاس‌بندی داده‌ها است. و یک پایپ‌لاین پیش‌پردازش برای داده‌های دسته‌ای ایجاد می‌شود

که شامل پر کردن مقادیر گم شده با مقدار پر تکرار و تبدیل به کدگذاری یک گانه است. و پایپ لاین های پیش پردازش عددی و دسته ای ترکیب می شوند تا یک پیش پردازش کلی برای هر دو نوع داده داشته باشیم.

```
num_pipeline = Pipeline(steps=[  
    ('imputer', SimpleImputer(strategy='mean')),  
    ('scaler', StandardScaler())  
])  
  
cat_pipeline = Pipeline(steps=[  
    ('imputer', SimpleImputer(strategy='most_frequent')),  
    ('encoder', OneHotEncoder(handle_unknown='ignore'))  
])  
  
preprocess_pipeline = ColumnTransformer(  
    transformers=[  
        ('num', num_pipeline, num_features),  
        ('cat', cat_pipeline, cat_features)  
    ])
```

پیش پردازش داده های آموزشی و تست را با استفاده از پایپلاین ترکیبی انجام می دهیم. و مقادیر پردازش شده را به آرایه های Numpy تبدیل می کنیم) در صورتی که از نوع Sparse باشد.

```
features_train_prepared = preprocess_pipeline.fit_transform(features_train)  
features_test_prepared = preprocess_pipeline.transform(features_test)  
  
features_train_prepared = features_train_prepared.toarray()  
features_test_prepared = features_test_prepared.toarray()
```

مدل شبکه عصبی را تعریف می کنیم. این مدل شامل سه لایه است:

- لایه ورودی با ۶۴ نرون و تابع فعال سازی relu.
- یک لایه میانی با ۶۴ نرون و تابع فعال سازی relu.
- یک لایه خروجی با یک نرون (پیش بینی مقدار پیوسته).

```
neural_model = Sequential([  
    Dense(64, activation='relu', input_shape=(features_train_prepared.shape[1],)),  
    Dense(64, activation='relu'),  
    Dense(1)  
])
```

سپس مدل را کامپایل می کنیم. از بهینه ساز Adam و تابع زیان Mean Squared Error استفاده می کنیم. مدل را با داده های آموزشی آموزش می دهیم:

epochs=100: • تعداد ۱۰۰ دوره آموزش.

- `batch_size=32`: اندازه بج .۳۲.
  - `validation_split=0.2`: ۲٪ از داده‌های آموزشی به عنوان داده‌های اعتبارسنجی استفاده می‌شود.
  - `verbose=1`: نمایش جزئیات آموزش در خروجی.
- و بعد پیش‌بینی‌ها را برای داده‌های تست انجام می‌دهیم.