Лабораторная работа №27

Нейронные сети

Семейство алгоритмов, известное как нейронные сети, недавно пережило свое возрождение под названием «глубокое обучение». Несмотря на то что глубокое обучение сулит большие перспективы в различных сферах применения машинного обучения, алгоритмы глубоко обучения, как правило, жестко привязаны к конкретным случаям использования. В данном разделе мы рассмотрим лишь некоторые относительно простые методы, а именно многослойные персептроны для классификации и регрессии, которые могут служить отправной точкой в изучении более сложных методов машинного обучения. Многослойные персептроны (**MLP**) также называют простыми (**vanilla**) нейронными сетями прямого распространения, а иногда и просто нейронными сетями.

Модель нейронной сети

MLP можно рассматривать как обобщение линейных моделей, которое прежде чем прийти к решению выполняет несколько этапов обработки данных.

Вспомним, что в линейной регрессии прогноз получают с помощью следующей формулы:

$$\hat{y} \; w[0] * x[0] \;\; w[1] * x[1] \;\; \ldots \;\; w[p] * x[p] \;\; b$$

Говоря простым языком, $\hat{y}$ - это взвешенная сумма входных признаков $x[0] \ldots x[p]$. Входные признаки взвешены по вычисленным в ходе обучения коэффициентам $w[0] \ldots w[p]$. Мы может представить их графически, как показано на рис. 1:

```
display(mglearn.plots.plot_logistic_regression_graph())
```

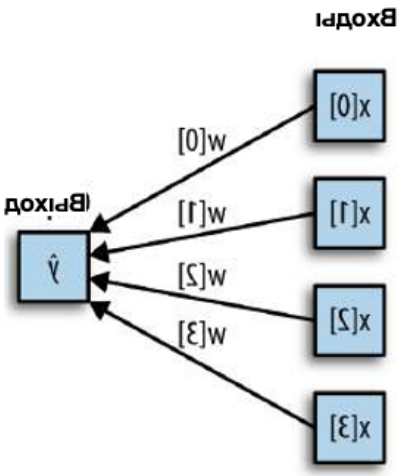

Рис. 1 Визуализация логистической регрессии, в которой входные признаки и прогнозы показаны в виде узлов, а коэффициенты – в виде соединений между узлами

Здесь каждый узел, показанный слева, представляет собой входной признак, соединительные линии – коэффициенты, а узел справа – это выход, который является взвешенной суммой входов.

MLP процесс вычисления взвешенных сумм повторяется несколько раз. Сначала вычисляются скрытые элементы (**hidden units**), которые представляют собой промежуточный этап обработки. Они вновь объединяются с помощью взвешенных сумм для получения конечного результата (рис. 2):

```
display(mglearn.plots.plot_single_hidden_layer_graph())
```

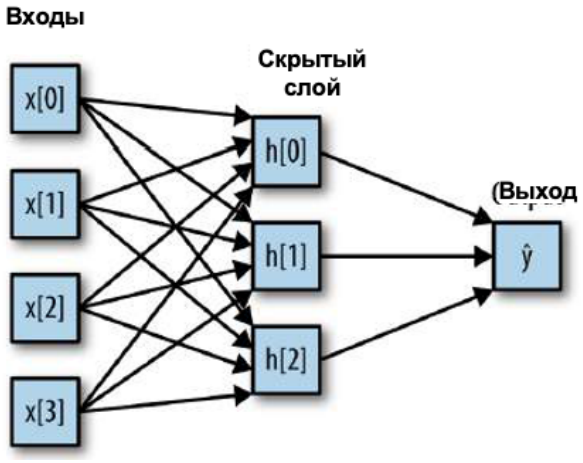

Рис. 2 Иллюстрация многослойного персептрона с одним скрытым слоем

Этой модели гораздо больше вычисляемых коэффициентов (также называемых весами): коэффициент между каждым входом и каждым скрытым элементом (которые образуют скрытый слой или **hidden layer**) коэффициент между каждым элементом скрытого слоя и выходом

С математической точки зрения вычисление серии взвешенных сумм – это то же самое, что вычисление лишь одной взвешенной суммы, таким образом, чтобы эта модель обладала более мощной прогнозной силой, чем линейная модель, нам нужен один дополнительный трюк. Поясним его на примере нейронной сети с одним скрытым слоем. Входной слой просто передает входы скрытому слою сети, либо без преобразования, либо выполнив сначала стандартизацию входов. Затем происходит вычисление взвешенной суммы входов для каждого элемента скрытого слоя, к ней применяется **функция активации** – обычно используются нелинейные функции выпрямленный линейный элемент (**rectified linear unit** или **relu**) или гиперболический тангенс (**hyperbolic tangent** или **tanh**).

В итоге получаем выходы нейронов скрытого слоя. Эти промежуточные выходы могут считаться нелинейными преобразованиями и комбинациями первоначальных входов. Они становятся входами выходного слоя. Снова вычисляем взвешенную сумму входов, применяем функцию активации и получаем итоговые значения целевой переменной. Функции активации **relu** и **tanh** показаны на рис. 3 **Relu** отсекает значения ниже нуля, в то время как **tanh** принимает значения от –1 до 1 (соответственно для минимального и максимального значений входов). Любая из этих двух нелинейных функций позволяет нейронной сети в отличие от линейной модели вычислять гораздо более сложные зависимости.

```
line = np.linspace(-3, 3, 100)
plt.plot(line, np.tanh(line), label="tanh")
plt.plot(line, np.maximum(line, 0), label="relu")
plt.legend(loc="best")
plt.xlabel("x")
plt.ylabel("relu(x), tanh(x)")
```

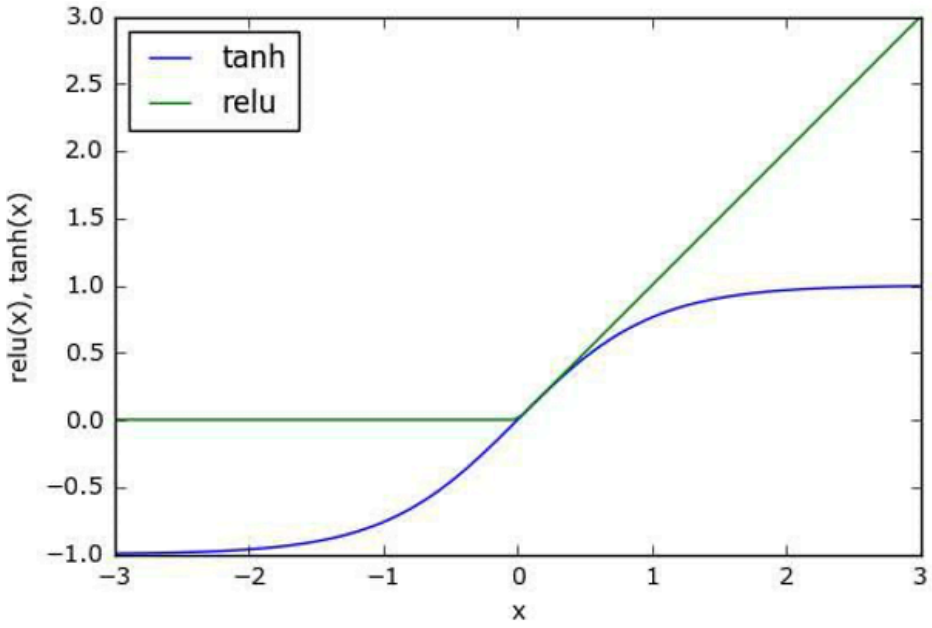

Рис. 3 Функция активации гиперболический тангенс и функция активации выпрямленного линейного элемента

Для небольшой нейронной сети, изображенной на рис. 2, полная формула вычисления $\hat{y}$ в случае регрессии будет выглядеть так (при использовании **tanh**):

$h[0]$ tanh($w[0,0]$* $x[0]$ $w[1,0]$* $x[1]$ $w[2,0]$* $x[2]$ $w[3,0]$* $x[3]$) $h[1]$
tanh($w[0,0]$* $x[0]$ $w[1,0]$* $x[1]$ $w[2,0]$* $x[2]$ $w[3,0]$* $x[3]$) $h[2]$
tanh($w[0,0]$* $x[0]$ $w[1,0]$* $x[1]$ $w[2,0]$* $x[2]$ $w[3,0]$* $x[3]$) $\hat{y}$ $v[0]$* $h[0]$
$v[1]$* $h[1]$ $v[2]$* $h[2]$

Здесь w – веса между входом x и скрытом слоем h, а v – весовые коэффициенты между скрытым слоем h и выходом $\hat{y}$. Веса v и w вычисляются по данным, x являются входными признаками, $\hat{y}$ - вычисленный выход, а h – промежуточные вычисления. Важный параметр, который должен задать пользователь – количество узлов в скрытом слое. Его значение может быть маленьким, например, 10 для очень маленьких или простых наборов данных или же большим, например, 10000 для очень сложных данных. Кроме того, можно добавить дополнительные скрытые слои, как показано на рис. 4:

```
mglearn.plots.plot_two_hidden_layer_graph()
```

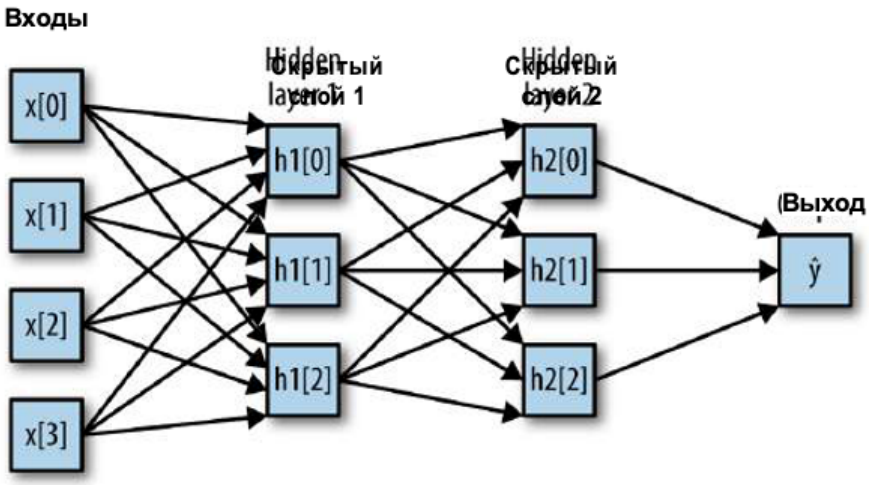

Рис. 4 Многослойный персептрон с двумя скрытыми слоями

Построение больших нейронных сетей, состоящих из множества слоев вычислений, вдохновило специалистов ввести в обиход термин «глубокое обучение» («deep learning»).

Настройка нейронных сетей

Давайте посмотрим, как работает **MLP**, применив **MLPClassifier** к набору данных two_moons, который мы использовали ранее в этой главе. Результаты показаны на рис. 5:

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    stratify=y,
    random_state=42
)

mlp = MLPClassifier(
    solver='lbfgs',
    random_state=0
).fit(
    X_train,
    y_train
)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

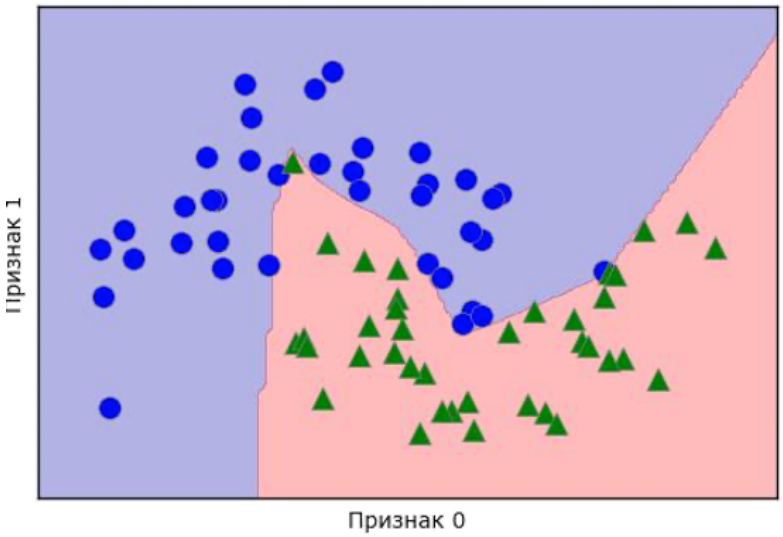

Рис. 5 Граница принятия решений, построенная нейронной сетью со 100 скрытыми элементами на наборе данных two_moons

Как видно из рис., нейронная сеть построила нелинейную, но относительно гладкую границу принятия решений. Мы использовали **solver='lbfgs'**, который рассмотрим позднее.

По умолчанию **MLP** использует 100 скрытых узлов, что довольно много для этого небольшого набора данных. Мы можем уменьшить число (что снизит сложность модели) и снова получить хороший результат (рис. 6):

```
mlp = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

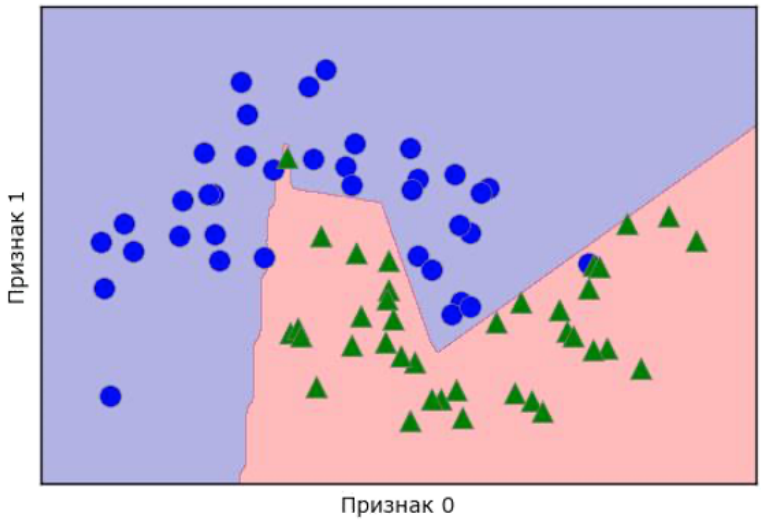

Рис. 6 Граница принятия решений, построенная нейронной сетью с 10 скрытыми элементами на наборе данных two_moons

При использовании лишь 10 скрытых элементов граница принятия решений становится более неровной. По умолчанию используется функция активации **relu**, показанная на рис. 6. При использовании одного скрытого слоя решающая функция будет состоять из 10 прямолинейных отрезков. Если необходимо получить более гладкую решающую границу, можно добавить большее количество скрытых элементов (как показано на рис. 6), добавить второй скрытый слой (рис. 7), или использовать функцию активации tanh (рис. 8):

```
# использование двух скрытых слоев по 10 элементов в каждом
mlp = MLPClassifier(solver='lbfgs', random_state=0,
hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

```
# использование двух скрытых слоев по 10 элементов в каждом,
# на этот раз с функцией tanh
mlp = MLPClassifier(solver='lbfgs', activation='tanh',
random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

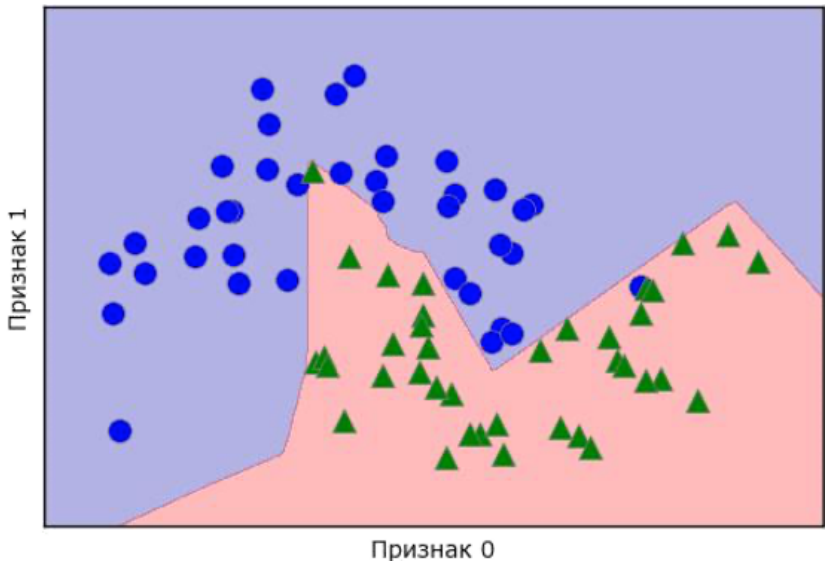

Рис. 7 Граница принятия решений, построенная нейронной сетью с 2 скрытыми слоями по 10 элементов в каждом и функцией активации выпрямленный линейный элемент

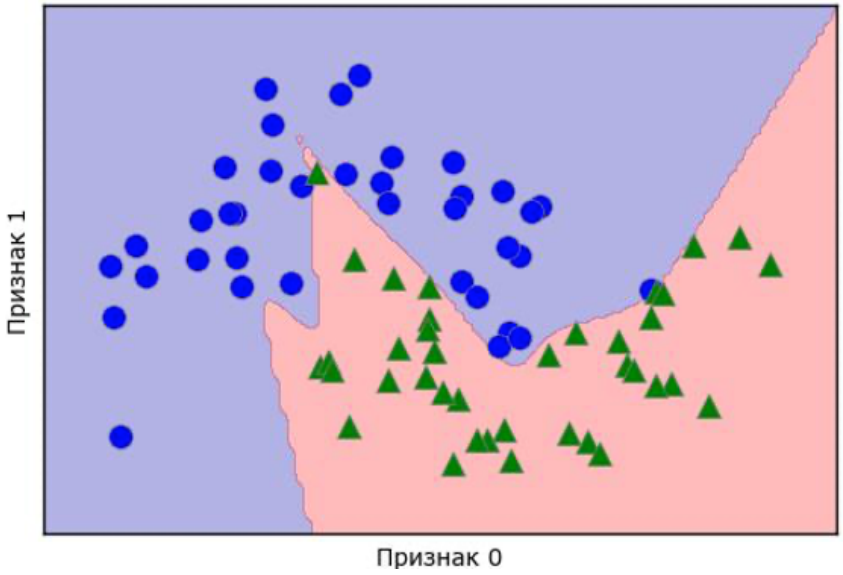

Рис. 8 Граница принятия решений, построенная нейронной сетью с 2 скрытыми слоями по 10 элементов в каждом и функцией активации гиперболический тангенс

И, наконец, мы можем дополнительно настроить сложность нейронной сети с помощью **l2** штрафа, чтобы сжать весовые коэффициенты до близких к нулю значений, как мы это делали в гребневой регрессии и линейных классификаторов. В **MLPClassifier** за это отвечает параметр **alpha** (как и в моделях линейной регрессии), и по умолчанию установлено очень низкое значение (небольшая регуляризация). На рис. 9 показаны результаты применения к набору данных two_moons различных значений **alpha** с использованием двух скрытых слоев с 10 или 100 элементами в каждом

```
fig, axes = plt.subplots(2, 4, figsize=(20, 8))
for axx, n_hidden_nodes in zip(axes, [10, 100]):
    for ax, alpha in zip(axx, [0.0001, 0.01, 0.1, 1]):
        mlp = MLPClassifier(
            solver='lbfgs',
            random_state=0,
            hidden_layer_sizes=[n_hidden_nodes, n_hidden_nodes],
            alpha=alpha
        )
        mlp.fit(X_train, y_train)
        mglearn.plots.plot_2d_separator(
            mlp, X_train, fill=True, alpha=.3, ax=ax
        )
        mglearn.discrete_scatter(
            X_train[:, 0], X_train[:, 1], y_train, ax=ax
        )
        ax.set_title("n_hidden=[{}, {}]\nalpha={:.4f}"
            .format(n_hidden_nodes, n_hidden_nodes, alpha)
        )
```

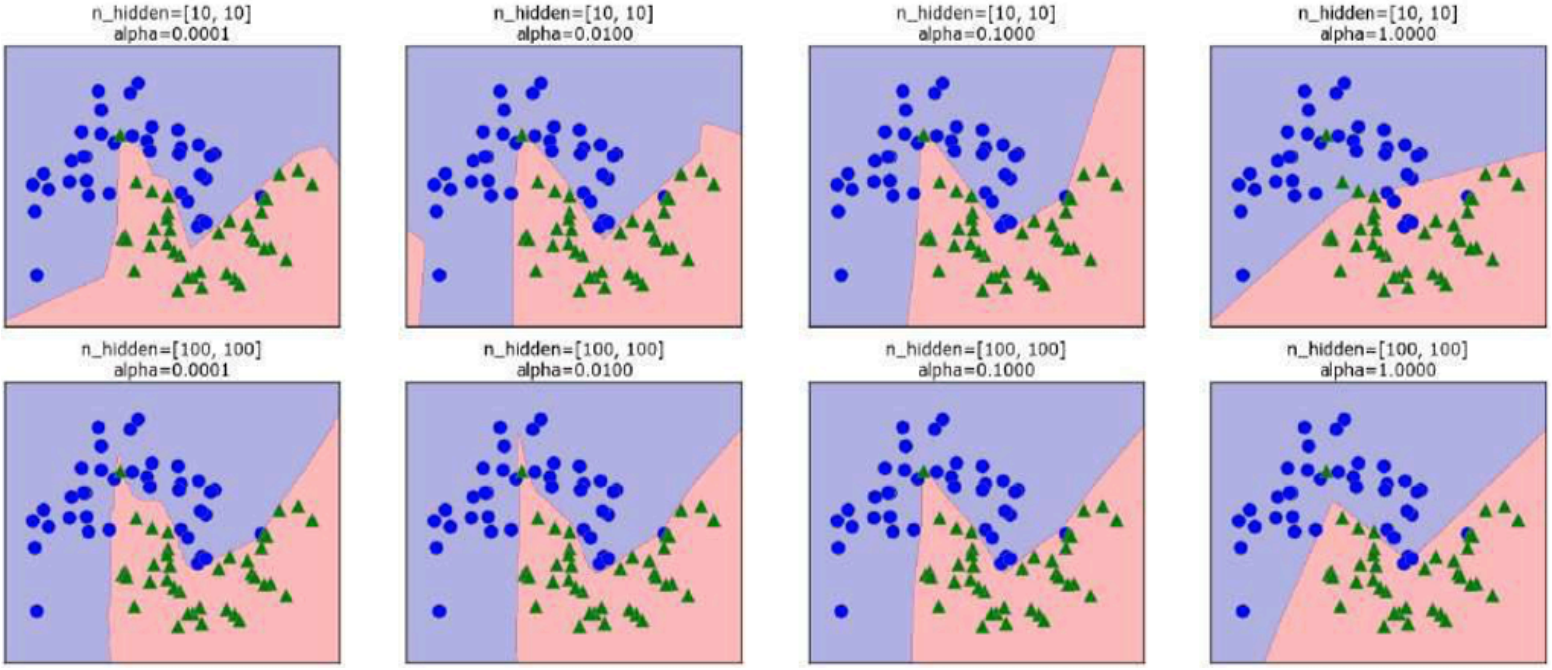

Рис. 9 Границы принятия решений для различного количества скрытых элементов и разных значений параметра alpha

Как вы, наверное, уже поняли, существуют различные способы регулировать сложность нейронной сети: количество скрытых слоев, количество элементов в каждом скрытом слое и регуляризация (**alpha**). На самом деле их гораздо больше, но мы не будем здесь вдаваться в подробности.

Важным свойством нейронных сетей является то, что их веса задаются случайным образом перед началом обучения и случайная инициализация влияет на процесс обучения модели. Это означает, что даже при использовании одних и тех же параметров мы можем получить очень разные модели, задавая разные стартовые значения генератора псевдослучайных чисел. При условии, что сеть имеет большой размер и сложность настроена правильно, данный факт не должен сильно влиять на правильность, однако о нем стоит помнить (особенно при работе с небольшими сетями). На рис. 10 представлены графики нескольких моделей, обученных с использованием тех же самых значений параметров:

```
fig, axes = plt.subplots(2, 4, figsize=(20, 8))
for i, ax in enumerate(axes.ravel()):
    mlp = MLPClassifier(
        solver='lbfgs', random_state=i, hidden_layer_sizes=[100, 100]
    )
    mlp.fit(X_train, y_train)
    mglearn.plots.plot_2d_separator(
        mlp, X_train, fill=True, alpha=.3, ax=ax
    )
    mglearn.discrete_scatter(
        X_train[:, 0], X_train[:, 1], y_train, ax=ax
    )
```

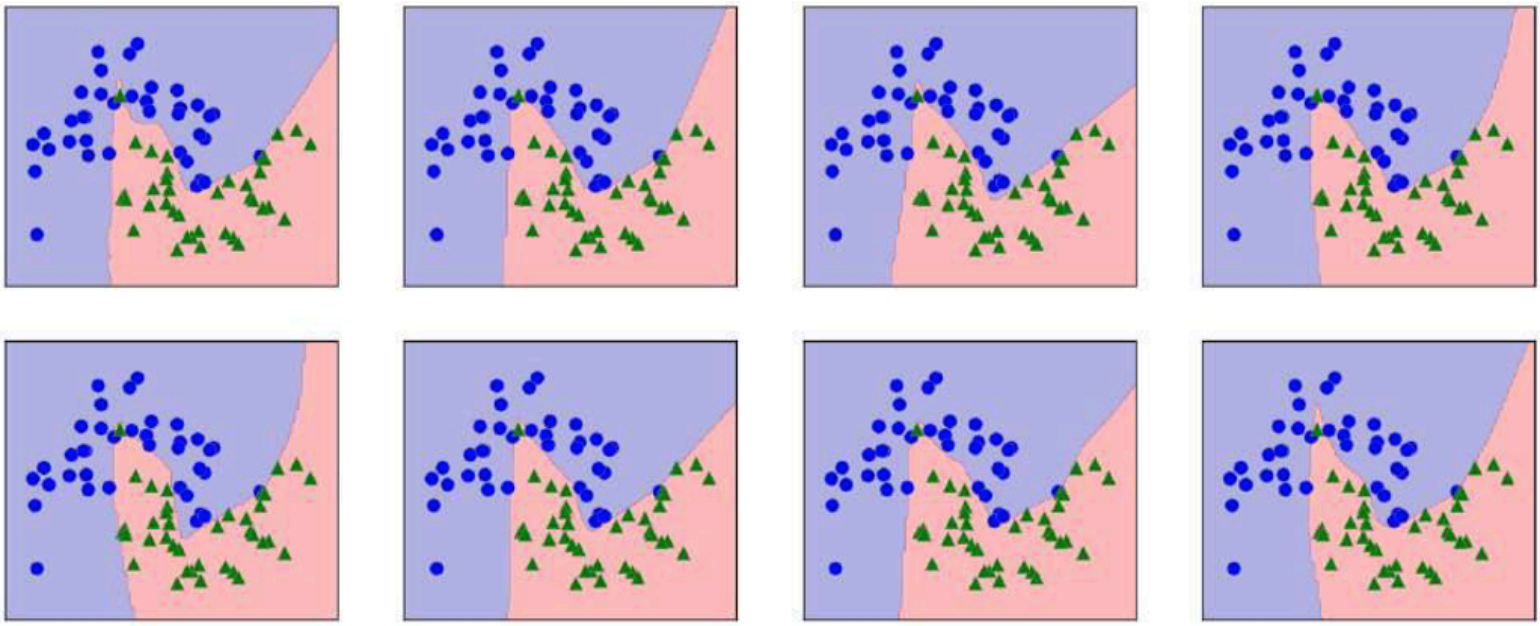

Рис. 10 Границы принятия решений, полученные с использованием тех же самых параметров, но разных стартовых значений

Чтобы лучше понять, как нейронная сеть работает на реальных данных, давайте применим **MLPClassifier** к набору данных Breast Cancer. Мы начнем с параметров по умолчанию:

```
print("Максимальные значения характеристик:\n{}".format(cancer.data.max(axis=0)))
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0
)

mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train)

print("Правильность на обучающем наборе: {:.2f}".format(mlp.score(X_train,
y_train)))
print("Правильности на тестовом наборе: {:.2f}".format(mlp.score(X_test, y_test)))
```

```
Максимальные значения характеристик:
[    28.110        39.280        188.500
     0.201         0.304         0.097
     0.031         0.135         0.396
     49.540        251.200       4254.000
     0.664         0.207
]
Правильность на обучающем наборе: 0.92
Правильность на тестовом наборе: 0.90
```

MLP демонстрирует довольно неплохую правильность, однако не столь хорошую, если сравнивать с другими моделями. Как и в предыдущем примере с **SVC**, это, вероятно, обусловлено масштабом данных. Нейронные сети также требуют того, чтобы все входные признаки были измерены в одном и том же масштабе, в идеале они должны иметь среднее 0 и дисперсию 1. Мы должны отмасштабировать наши данные так, чтобы они отвечали этим требованиям. Опять же, мы

будем делать это вручную, однако это можно делать автоматически с помощью **StandardScaler**.

```
# вычисляем среднее для каждого признака обучающего набора
mean_on_train = X_train.mean(axis=0)
# вычисляем стандартное отклонение для каждого признака обучающего набора
std_on_train = X_train.std(axis=0)
# вычитаем среднее и затем умножаем на обратную величину
# стандартного отклонения mean=0 и std=1
# используем ТО ЖЕ САМОЕ преобразование
# (используем среднее и стандартное отклонение обучающего набора)
# для тестового набора
X_test_scaled = (X_test - mean_on_train) / std_on_train

mlp = MLPClassifier(random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Правильность на обучающем наборе: {:.3f}"
    .format( mlp.score(X_train_scaled, y_train))
)
print("Правильность на тестовом наборе: {:.3f}"
    .format(mlp.score(X_test_scaled, y_test))
)
```

```
Правильность на обучающем наборе: 0.991
Правильность на тестовом наборе: 0.965
```

После масштабирования результаты стали намного лучше и теперь уже вполне могут конкурировать с результатами остальных моделей. Впрочем, мы получили предупреждение о том, что достигнуто максимальное число итераций. Оно является неотъемлемой частью алгоритма adam и сообщает нам о том, что мы должны увеличить число итераций:

```
mlp = MLPClassifier(max_iter=1000, random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Правильность на обучающем наборе: {:.3f}"
    .format( mlp.score(X_train_scaled, y_train))
)
print("Правильность на тестовом наборе: {:.3f}"
    .format(mlp.score(X_test_scaled, y_test))
)
```

```
Правильность на обучающем наборе: 0.995
Правильность на тестовом наборе: 0.965
```

Увеличение количества итераций повысило правильность лишь на обучающем наборе. Тем не менее модель имеет достаточно высокое значение правильности. Поскольку существует определенный разрыв между правильностью на обучающем наборе и правильностью на тестовом наборе, мы можем попытаться уменьшить сложность модели, чтобы улучшить обобщающую способность. В данном случае мы увеличим параметр **alpha** (довольно сильно с 0.0001 до 1), чтобы применить к весам более строгую регуляризацию:

```
mlp = MLPClassifier(max_iter=1000, alpha=1, random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Правильность на обучающем наборе: {:.3f}"
    .format( mlp.score(X_train_scaled, y_train))
)
print("Правильность на тестовом наборе: {:.3f}"
    .format(mlp.score(X_test_scaled, y_test))
)
```

```
Правильность на обучающем наборе: 0.988
Правильность на тестовом наборе: 0.972
```

Это дает правильность, сопоставимую с правильностью лучших моделей.

Несмотря на то что анализ нейронной сети возможен, он, как правило, гораздо сложнее анализа линейной модели или модели на основе дерева. Один из способов анализа нейронной сети заключается в том, чтобы исследовать веса модели. Образец такого анализа вы можете увидеть в галерее примеров scikit-learn. Применительно к набору данных Breast Cancer такой анализ может быть немного сложен. Следующий график (рис. 11) показывает весовые коэффициенты, которые были вычислены при подключении входного слоя к первому скрытому слою. Строки в этом графике соответствуют 30 входным признакам, а столбцы – 100 скрытым элементам. Светлые цвета соответствуют высоким положительным значениям, в то время как темные цвета соответствуют отрицательным значениям:

```
plt.figure(figsize=(20, 5))
plt.imshow(mlp.coefs_[0], interpolation='none', cmap='viridis')
plt.yticks(range(30), cancer.feature_names)
plt.xlabel("Столбцы матрицы весов")
plt.ylabel("Входная характеристика")
plt.colorbar()
```

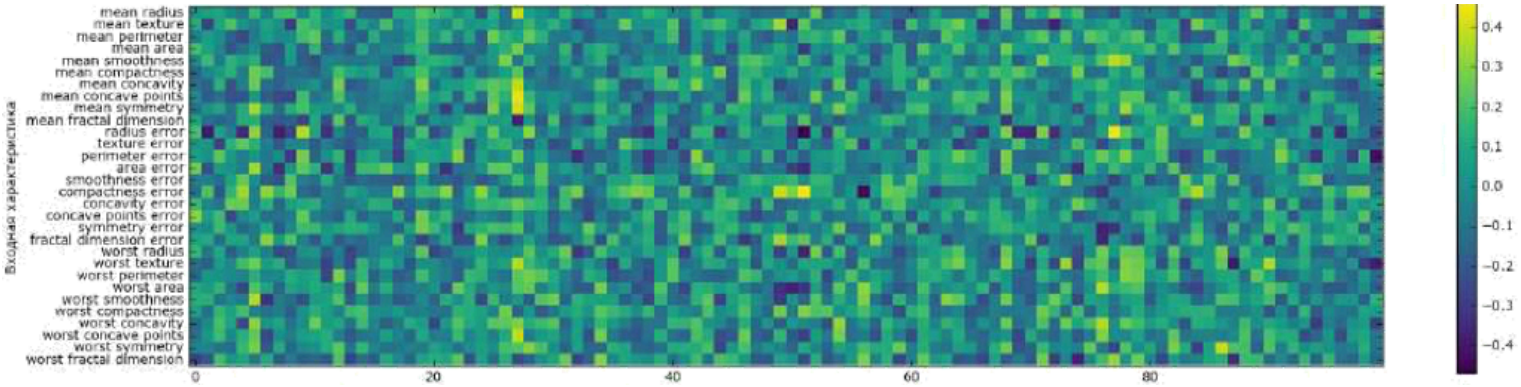

Рис. 11 Теплокарта для весов первого слоя нейронной сети, обученной на наборе данных Breast Cancer

Один из возможных выводов, который мы можем сделать, заключается в том, что признаки с небольшими весами скрытых элементов «менее важны» в модели. Мы можем увидеть, что «mean smoothness» и «mean compactness» наряду с признаками, расположенными между «smothness error» и «fractal dimension error», имеют относительно низкие веса по сравнению с другими признаками. Это может означать, что эти признаки являются менее важными или, возможно, мы не преобразовали их таким способом, чтобы их могла использовать нейронная сеть.

Кроме того, мы можем визуализировать веса, соединяющие скрытый слой с выходным слоем, но их еще труднее интерпретировать.

Несмотря на то что для наболее распространенных архитектур нейронных сетей **MLPClassifier** и **MLPRegressor** предлагают легкий в использовании интерфейс, они представляют лишь небольшой набор возможных средств, позволяющих строить нейронные сети. Если вас интересует работа с более гибкими или более масштабными моделями, мы рекомендуем вам не ограничиваться возможностями библиотеки scikit-learn и обратиться к фантастическим по своими возможностями библиотекам глубокого обучения. Для пользователей Python наиболее устоявшимися являются **keras**, **lasagna** и **tenzor-flow**. lasagna построена на основе библиотеки theano, тогда как keras может использовать либо tensor-flow, либо theano. Эти библиотеки предлагают гораздо более гибкий интерфейс для построения нейронных сетей и обновляются в соответствии с последними достижениями в области глубокого обучения. Кроме того, все популярные библиотеки глубокого обучения позволяют использовать высокопроизводительные графические процессоры (GPU), которые в scikit-learn не поддерживаются. Использование графических процессоров позволяет ускорить вычисления от 10 до 100 раз, и они имеют важное значение для применения методов глубого обучения для крупномасштабных наборов данных.

Параметры, недостатки и параметры

Нейронные сети вновь «вышли на сцену» во многих сферах применения машинного обучения в качестве передовых методов. Одно из их главных преимуществ заключается в том, что они способны обрабатывать информацию, содержащуюся в больших объемах данных, и строить невероятно сложные модели. При наличии достаточного времени вычислений, данных и тщательной настройки параметров нейронные сети часто превосходят другие алгоритмы машинного обучения (для задач классификации и регрессия).

Это дает и свои минусы. Нейронные сети, особенно крупные нейронные сети, как правило, требуют длительного времени обучения. Как мы видели здесь, они также требуют тщательной предварительной обработки данных. Аналогично **SVM**, нейронные сети лучше всего работают с «однородными» данными, где все признаки измерены в одном том же масштабе. Что касается данных, в которых признаки имеют разный масштаб, модели на основе дерева могут дать лучший результат. Кроме того, настройка параметров нейронной сети – это само по себе искусство. В наших экспериментах мы едва коснулись возможных способов настройки и обучения нейросетевых моделей.

Оценка сложности в нейронных сетях

Наиболее важными параметрами являются ряде слоев и число скрытых блоков в одном слое. Вы должны начать с одной или двумя скрытыми слоями, и, возможно, расширить оттуда. Количество узлов на скрытом уровне часто аналогично числу входных функций, но редко выше, чем в низких до средних тысяч.

Полезным показателем, позволяющим судить о сложности нейронной сети, является количество вычисляемых в ходе обучения весов или коэффициентов. Если вы работаете с 2-классовым набором данных, содержащим 100 признаков, и используете нейронную сеть, состоящую из 100 скрытых элементов, то между входным и первым скрытым слоем будет 100 * 100 = 10000 весов. Кроме того, между скрытым слоем и выходным слоем будет 100 * 1 = 100 весов, что в итоге даст около 10100 весов. Если использовать второй скрытый слой размером 100 скрытых элементов, то 100 * 100 = 10000 весов из первого скрытого слоя добавятся ко второму скрытому слою, что в итоге составит 20100 весов. Если вместо этого вы будете использовать один слой из 1000 скрытых элементов, вы вычислите 100*1000 = 100000 весов на пути от входного слоя к скрытому слою и 1000*1 весов на пути из скрытого слоя к выходному слою, всего 101000 весов. Если использовать второй скрытый слой, вы добавите еще 1000*1000 = 1000000 весов, получив колоссальную цифру 1101000, что в 50 раз больше количества весов, вычисленных для модели с двумя скрытыми слоями по 100 элементов в каждом.

Общераспространенный способ настройки параметров в нейронной сети – сначала построить сеть достаточно большого размера, чтобы она обучилась. Затем, убедившись в том, что сеть может обучаться, сжимаете веса сети или увеличиваете **alpha**, чтобы добавить регуляризацию, которая улучшит обобщающую способность.

В наших экспериментах мы сосредоточились главным образом на спецификации модели: количестве слоев и узлов в слое, регуляризации нелинейных функциях активации. Эти параметры задают модель, которую мы хотим обучить. Кроме того, встает вопрос о том, как обучить модель или алгоритм, который используется для вычисления весов и задается с помощью параметра **solver**. Существует два простых в использовании алгоритма. Алгоритм **'adam'**, выставленный по умолчанию, дает хорошее качество в большинстве ситуаций, но весьма чувствителен к масштабированию данных (поэтому важно отмасштабировать ваши данные так, чтобы каждая характиристика имела среднее 0 и дисперсию 1). Другой алгоритм **'lbfgs'** вполне надежен, но может занять много времени в случае больших моделей или больших массивов данных. Существует также более продвинутая опция **'sgd'**, которая используется многими специалистами по глубокому обучению. Опция **'sgd'** имеет большее количество дополнительных параметров, которые нужно настроить для получения наилучших результатов. Вы можете найти описания всех этих параметров в руководстве пользователя. Начиная работать с **MLP**, придерживайтесь алгоритмов **'adam'** и **'l-bfgs'**.

Код к лабораторной работе:

```
import mglearn
import sklearn
import matplotlib.pyplot as plt
import numpy as np
import graphviz

from IPython.display import display
display(mglearn.plots.plot_single_hidden_layer_graph())


line = np.linspace(-3, 3, 100)
plt.plot(line, np.tanh(line), label="tanh")
plt.plot(line, np.maximum(line, 0), label="relu")
plt.legend(loc="best")
plt.xlabel("x")
plt.ylabel("relu(x), tanh(x)")
plt.show()


mglearn.plots.plot_two_hidden_layer_graph()
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)

mlp = MLPClassifier(solver='lbfgs', random_state=0).fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()


mlp = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()


mlp = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
```

```
plt.ylabel("Признак 1")
plt.show()


mlp = MLPClassifier(solver='lbfgs', activation='tanh',
    random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()


fig, axes = plt.subplots(2, 4, figsize=(20, 8))
for axx, n_hidden_nodes in zip(axes, [10, 100]):
    for ax, alpha in zip(axx, [0.0001, 0.01, 0.1, 1]):
        mlp = MLPClassifier(solver='lbfgs', random_state=0,
            hidden_layer_sizes=[n_hidden_nodes, n_hidden_nodes], alpha=alpha)
        mlp.fit(X_train, y_train)
        mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3, ax=ax)
        mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train, ax=ax)
        ax.set_title("n_hidden=[{}, {}]\nalpha={:.4f}".format(
            n_hidden_nodes, n_hidden_nodes, alpha))
plt.show()


fig, axes = plt.subplots(2, 4, figsize=(20, 8))
for i, ax in enumerate(axes.ravel()):
    mlp = MLPClassifier(solver='lbfgs', random_state=i, hidden_layer_sizes=[100, 100])
    mlp.fit(X_train, y_train)
    mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3, ax=ax)
    mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train, ax=ax)
plt.show()


from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("Максимальные значения характеристик:\n{}".format(cancer.data.max(axis=0)))


X_train, X_test, y_train, y_test = train_test_split( cancer.data, cancer.target, random_state=0)
mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(mlp.score(X_train, y_train)))
print("Правильности на тестовом наборе: {:.2f}".format(mlp.score(X_test, y_test)))


min_on_training = X_train.min(axis=0)
range_on_training = (X_train - min_on_training).max(axis=0)
X_train_scaled = (X_train - min_on_training) / range_on_training
```

```
mean_on_train = X_train.mean(axis=0)
std_on_train = X_train.std(axis=0)
X_test_scaled = (X_test - mean_on_train) / std_on_train
mlp = MLPClassifier(random_state=0)
mlp.fit(X_train_scaled, y_train)
print("Правильность на обучающем наборе: {:.3f}".format( mlp.score(X_train_scaled, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(mlp.score(X_test_scaled, y_test)))


mlp = MLPClassifier(max_iter=1000, random_state=0)
mlp.fit(X_train_scaled, y_train)
print("Правильность на обучающем наборе: {:.3f}".format( mlp.score(X_train_scaled, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(mlp.score(X_test_scaled, y_test)))


mlp = MLPClassifier(max_iter=1000, alpha=1, random_state=0)
mlp.fit(X_train_scaled, y_train)
print("Правильность на обучающем наборе: {:.3f}".format( mlp.score(X_train_scaled, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(mlp.score(X_test_scaled, y_test)))


plt.figure(figsize=(20, 5))
plt.imshow(mlp.coefs_[0], interpolation='none', cmap='viridis')
plt.yticks(range(30), cancer.feature_names)
plt.xlabel("Столбцы матрицы весов")
plt.ylabel("Входная характеристика")
plt.colorbar()
plt.show()
```