Искусственные нейронные сети (ANN)

Искусственные нейронные сети (Artificial Neural Networks, ANN)

Искусственные нейронные сети (ANN) – это математические модели, **вдохновлённые биологическими нейронами**.

- Они используются для решения **сложных задач**, таких как **распознавание образов, обработка естественного языка и прогнозирование**.
- Основной принцип работы – **обучение на примерах**, изменение весов связей между нейронами и постепенное улучшение точности.

Ключевые особенности нейросетей

✅ Могут **извлекать скрытые закономерности** из данных.

✅ Обучаются на **примерах** без явного программирования.

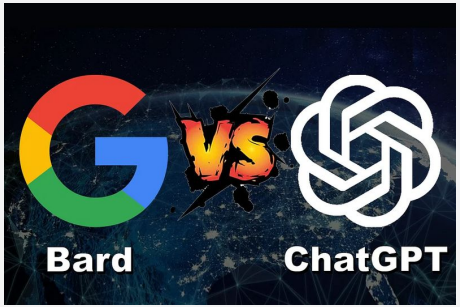
✅ Используются в **глубоком обучении (Deep Learning)**.

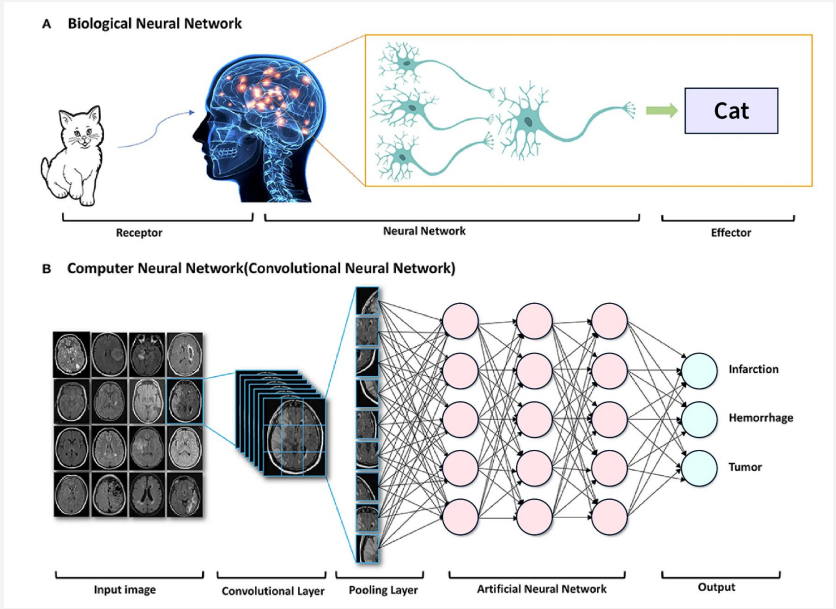
Примеры использования

В медицине – распознавание опухолей на снимках.

В автомобилях – автопилоты Tesla Waymo.

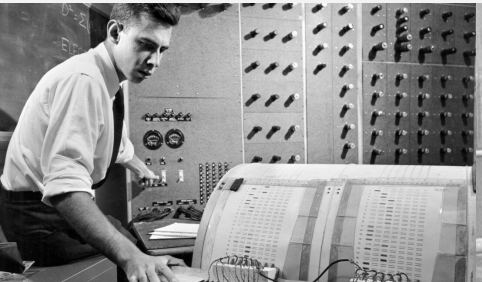
В чат-ботах – ChatGPT, Google Bard.

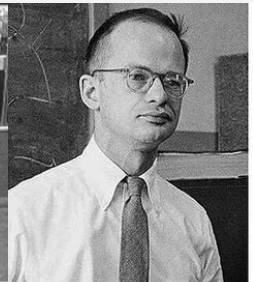
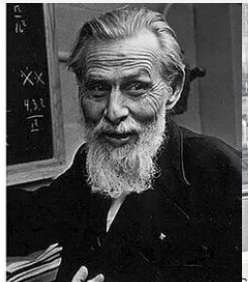




История и развитие нейронных сетей

- Развитие **искусственных нейронных сетей (ANN)** прошло несколько этапов – от первых теоретических моделей до современных **глубоких нейросетей (Deep Learning)**.
- Первые модели ANN были вдохновлены **работой биологических нейронов**.




Ключевые этапы развития нейронных сетей

1️⃣ 1943 – Перцептрон МакКаллока-Питтса

- Первая математическая модель нейрона, работающая по бинарному принципу (0 или 1).

2️⃣ 1957 – Перцептрон Розенблатта

- Первая нейросеть, способная **обучаться** и распознавать простые паттерны.

Ключевые этапы развития нейронных сетей

3️⃣ 1970-е – Многослойные сети и метод обратного распространения ошибки

- Решена проблема простого перцептрона: теперь сеть может обучаться на сложных задачах.

4️⃣ 1990-е – Прорыв в машинном обучении

- Нейросети начинают активно применяться в реальных задачах, таких как **распознавание рукописного текста**.

Ключевые этапы развития нейронных сетей

5️⃣ 2010-е – Эра глубокого обучения (Deep Learning)

- Развитие **глубоких нейросетей (DNN, CNN, RNN)** благодаря **ускорителям (GPU), большим данным и новым алгоритмам**.
- Нейросети достигают **человеческого уровня в распознавании изображений и речи**.

Примеры успеха нейросетей

2012 – нейросети впервые побеждают в ImageNet (конкурс компьютерного зрения).

2016 – AlphaGo побеждает чемпиона мира в игре го.

2023 – GPT-4 и другие языковые модели достигают **человеческого уровня понимания текста**.

Как устроен искусственный нейрон?

- **Искусственный нейрон** – это математическая модель, вдохновлённая работой биологического нейрона.
- Каждый нейрон принимает **входные данные**, умножает их на **веса (коэффициенты важности)**, суммирует и передаёт результат через **функцию активации**.

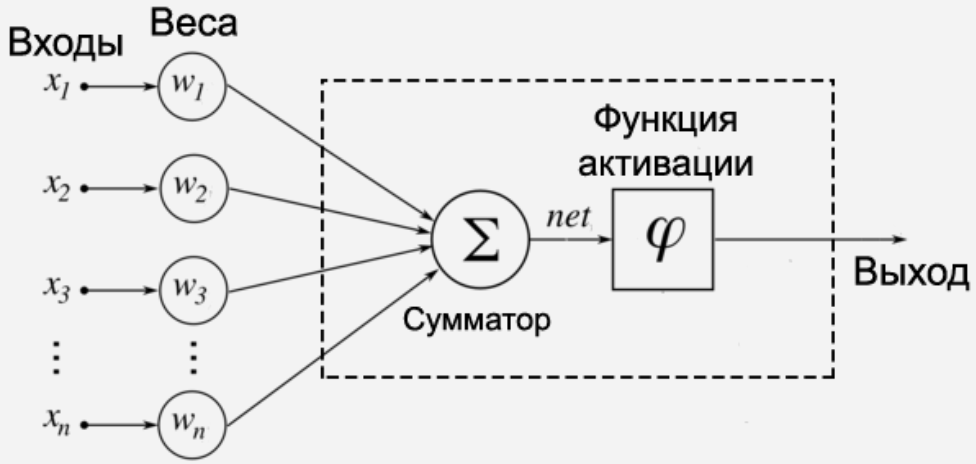
Основные компоненты нейрона

1. ***Входы (Inputs)***
2. ***Весовые коэффициенты (Weights, w)***
3. ***Сумматор (Summation Function)***
4. ***Функция активации (Activation Function, f(S))***

ReLU (Rectified Linear Unit),

Sigmoid, Tanh.

5. ***Выход (Output)***



Входы (Inputs)

***Входные данные* —** это информация, поступающая в нейрон. Они представляют собой числовые значения (например, пиксели изображения, параметры датчиков, признаки объекта).

- Количество входов зависит от задачи (например, изображение 28x28 пикселей = 784 входа).
- Входные данные могут быть нормализованы (например, значения в диапазоне [0,1] или [-1,1]).
-

Весовые коэффициенты (Weights, w)

Каждый вход связан с весом, который определяет значимость этого входа для результата. Весовые коэффициенты изменяются во время обучения сети.

- ***Вес w*** определяет, насколько важен соответствующий вход.
- ***Инициализация весов*** — случайная или с помощью специальных стратегий (например, He или Xavier).
- Обновляются с помощью алгоритмов оптимизации (например, градиентный спуск).

Сумматор (Summation Function)

Сумматор объединяет все входные данные с учетом их весов, вычисляя сумму:

$$S = \sum_{i=1}^{n} w_i x_i + b$$

где **b** — смещение (bias), которое помогает сети лучше обучаться.

Функции активации

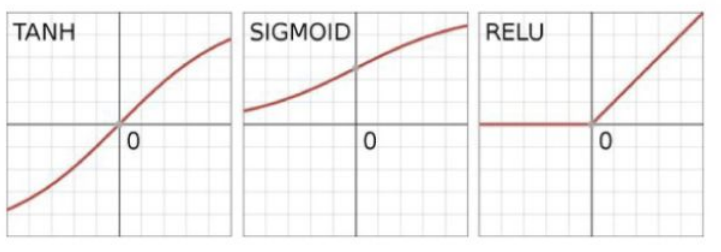
Функция активации добавляет **нелинейность** в работу нейрона, позволяя модели **учиться сложным зависимостям**.

- Без функции активации нейросеть была бы **обычным линейным классификатором**, неспособным решать сложные задачи.
- Разные функции активации используются в **разных слоях** нейросети в зависимости от задачи.

Основные функции активации

1. **Sigmoid (σ)**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



2. **Tanh (гиперболический тангенс)**

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

3. **ReLU (Rectified Linear Unit)**

$$ReLU(x) = \max(0, x)$$

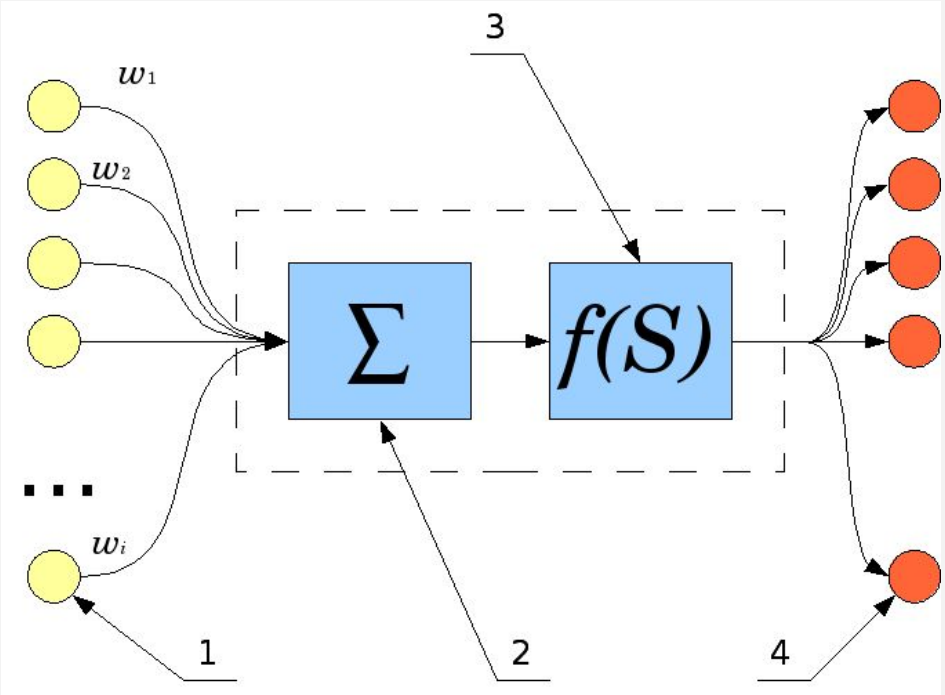
Выход (Output)

Выход нейрона **—** это значение, передаваемое дальше в сеть (следующему слою) или на финальный результат.

- В многослойных сетях выход одного слоя является входом для следующего.
- В последнем слое выход может быть интерпретирован как:
  - **Классификация:** вероятность класса (например, softmax).
  - **Регрессия:** непрерывное значение.
  - **Бинарное решение:** 0 или 1 (например, с помощью сигмоиды).

Математическая модель нейрона

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$



Как работает искусственный нейрон?

1. **Получает входные данные** (например, изображение пикселей).
2. **Умножает их на веса** (учитывает значимость каждого признака).
3. **Добавляет смещение bbb** (bias корректирует модель).
4. **Пропускает через функцию активации** (чтобы учесть сложные зависимости).
5. **Передаёт результат дальше** по сети.

Примеры

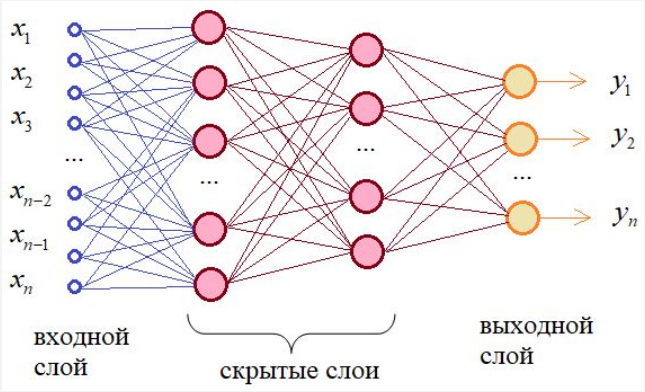
- В обработке изображений нейрон может **реагировать на края объектов**.
- В распознавании текста **разные нейроны активируются на буквы и слова**.
- В прогнозировании цен **учитываются экономические факторы**.

Пример работы искусственного нейрона

```
import numpy as np

inputs = np.array([0.5, 0.3, 0.2])

weights = np.array([0.8, -0.6, 0.4])
bias = 0.1

output = np.dot(inputs, weights) + bias

activation = max(0, output)
print("Выход нейрона:", activation)
```

```
Выход нейрона: 0.4
```

Архитектура нейронных сетей (Multilayer Perceptron - MLP)



1. **Входной слой (Input Layer)**
   - Принимает данные.
   - ***Пример***: пиксели изображения.
2. **Скрытые слои (Hidden Layers)**
   - Выполняют преобразования и извлекают признаки.
   - Слои могут быть глубокими (глубокое обучение).
3. **Выходной слой (Output Layer)**
   - Возвращает результат.
   - ***Пример:*** вероятность того, что изображение содержит кошку.

Соединения между слоями

1. Полносвязные (Fully Connected, Dense)
2. Разреженные (Sparse Connections)
3. Связи с повторным использованием (Shared Weights)
4. Прямые (Feedforward) соединения
5. Обратные (Recurrent) соединения

Полносвязные (Fully Connected, Dense)

- Каждый нейрон **соединен со всеми нейронами** следующего слоя.
- Используется в **классических многослойных перцептронах (MLP)**.
- Весовые коэффициенты определяют важность входных данных.

Плюсы: простота реализации, хорошо работает с небольшими наборами данных.

Минусы: требует большого количества параметров, что увеличивает вычислительную сложность, плохо масштабируется на большие сети.

Разреженные (Sparse Connections)

- Только часть нейронов **соединена** со следующими слоями.
- Используется в **нейросетях с весовыми ограничениями**, например, в **сверточных сетях (CNN)**.

Плюсы:

✅ Снижает вычислительную сложность.

✅ Улучшает обобщающую способность модели.

Минусы: может потерять полезную информацию при недостаточном количестве соединений.

Связи с повторным использованием (Shared Weights)

🔹 Один и тот же набор весов применяется к нескольким нейронам.

🔹 Используется в **сверточных нейронных сетях (CNN)**.

Плюсы:

✅ Снижает число параметров.

✅ Улучшает обобщающую способность модели.

Минусы:

❌ Менее гибкий, чем полносвязные сети.

Прямые (Feedforward) соединения

🔹 Данные передаются **только в одном направлении**: **от входного слоя к выходному**.

🔹 Используется в **стандартных MLP**.

Плюсы:

✅ Простая архитектура.

✅ Легко обучается с использованием обратного распространения ошибки.

Минусы: не учитывает последовательность входных данных (плохо для временных рядов).

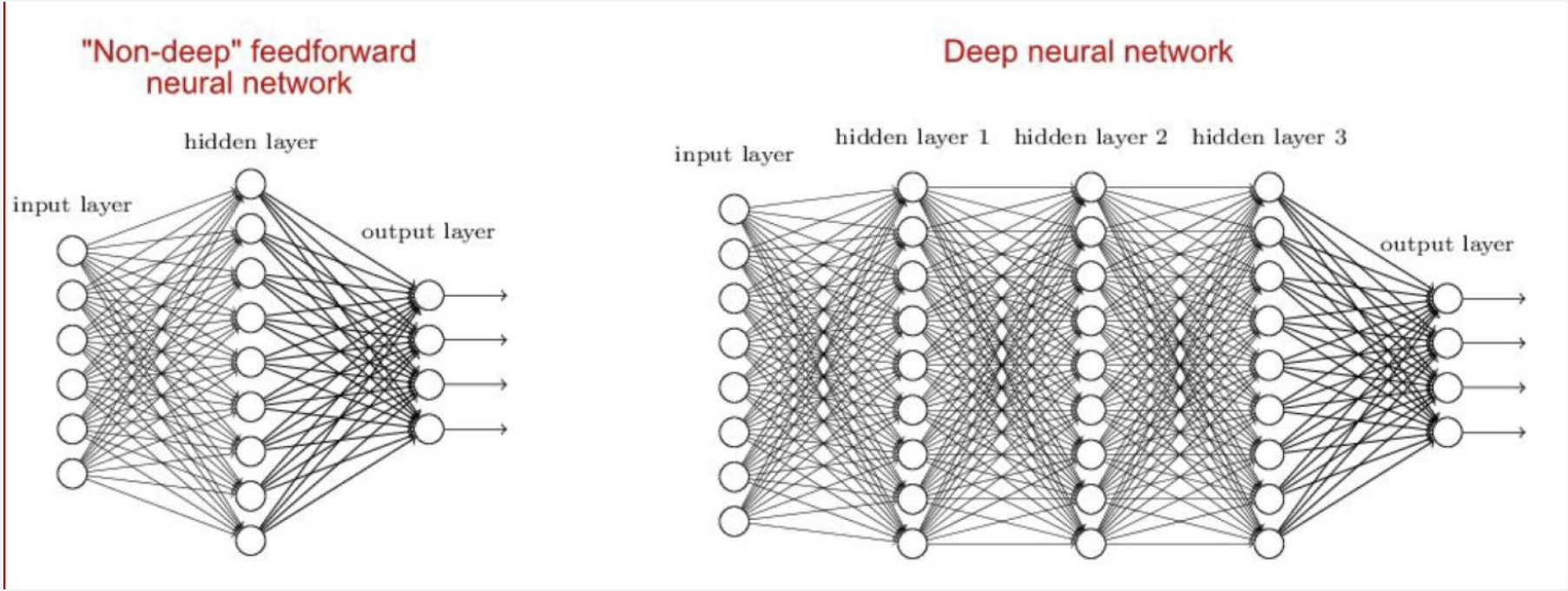
Обратные (Recurrent) соединения

- Нейроны могут **передавать информацию назад**, формируя **петли**.
- Используется в **рекуррентных нейронных сетях (RNN, LSTM, GRU)**.

Плюсы: учитывает зависимость во времени (используется для временных рядов, NLP).

Минусы: может страдать от затухания градиента, сложнее обучать по сравнению с обычными нейросетями.

Искусственная нейросеть



Код в Python (Keras)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    Dense(10, activation='softmax')
])

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

Типы нейронных сетей

1. ***Полносвязные нейронные сети*** (Dense, FCNN - Fully Connected Neural Networks)
2. ***Свёрточные нейронные сети*** (CNN - Convolutional Neural Networks)
3. ***Рекуррентные нейронные сети*** (RNN - Recurrent Neural Networks)
4. ***Автокодировщики*** (Autoencoders)

Полносвязные нейронные сети

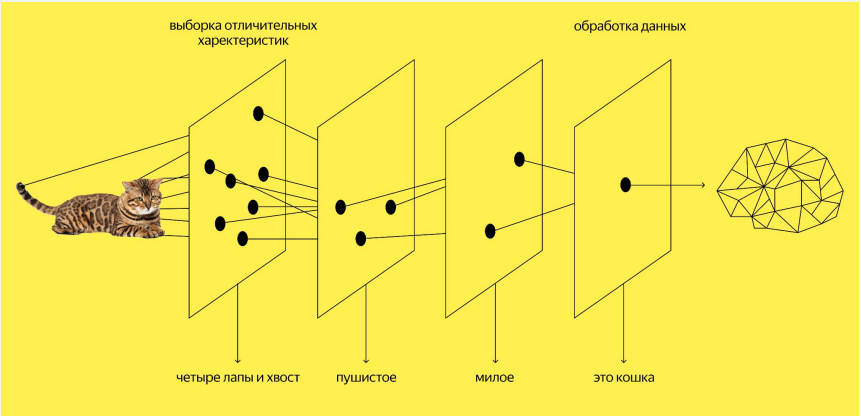
- Каждый нейрон соединён со всеми нейронами предыдущего и следующего слоя.
- Эти сети просты в реализации и используются в большинстве задач машинного обучения.
- Основной недостаток – **большое количество параметров**, что требует много ресурсов.

Пмерри кода
(Создание полносвязной сети с Keras)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(64, activation='relu', input_shape=(10,)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.summary()
```

Свёрточные нейронные сети

- Используются для обработки **изображений** и **видео**.
- Включают **свёрточные** и **пулинговые** слои, которые помогают находить особенности объектов.
- Уменьшают количество параметров за счёт обработки локальных областей изображения.



Пример кода (Создание CNN для обработки изображений)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten, Dense

model = Sequential([
Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)),
    MaxPooling2D(2,2),  Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')])
model.summary()
```

Рекуррентные нейронные сети

- В отличие от обычных сетей, **имеют память** – используют выход предыдущего шага как вход для следующего.
- Применяются для работы с **последовательными данными** (текст, аудио, временные ряды).
- Основная проблема – **затухающий градиент**, который решается с помощью **LSTM** и **GRU**.

Пример кода
(Создание RNN для обработки текста)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

model = Sequential([
                  SimpleRNN(50,     activation='relu',
input_shape=(100, 1)),
    Dense(1, activation='sigmoid')
])

model.summary()
```

Автокодировщики

- Используются для **снижения размерности данных**, устранения шума, обнаружения аномалий.
- Состоят из **кодировщика** (Encoder) и **декодировщика** (Decoder).
- Кодировщик уменьшает размерность данных, а декодировщик восстанавливает оригинальные данные.

Пример кода (Простой автокодировщик)

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input

input_layer = Input(shape=(784,))
encoded = Dense(32, activation='relu')(input_layer)
decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(input_layer, decoded)
autoencoder.summary()
```

Вывод

Каждый тип нейронной сети решает свою задачу:

- **FCNN** – универсальные, но требуют много ресурсов.
- **CNN** – работают с изображениями и видео.
- **RNN** – обрабатывают последовательные данные.
- **Autoencoders** – для сжатия данных и обнаружения аномалий.

Примеры применения ANN

Пример 1: Распознавание изображений

Задача: Определить, есть ли на изображении кошка.

Данные: Набор изображений с размеченными категориями.

Модель: CNN.

Пример 2: Обработка естественного языка (NLP)

Задача: Перевод текста.

Модель: Рекуррентные сети (RNN).

Пример 3: Диагностика заболеваний

Данные: Рентгеновские снимки.

Модель: CNN для классификации.

Ограничения нейронных сетей

1. Требование большого количества данных: для обучения ANN нужны большие наборы данных.

2. Высокие вычислительные затраты: обучение сложных моделей может занимать часы и даже дни.

3. Проблемы интерпретации: нейронные сети работают как "чёрный ящик", и сложно понять, как они принимают решения.