# ГЛАВА 1

# Виды нейронных сетей.

## 1.1 Детерминированные нейронные сети.

Сначала напомним, что такое обычные(детерминированные) нейронные сети и как они обучаются.

Основная задача обычных искусственных нейронных сетей($ANN$) в том, чтобы аппроксимировать некоторую зависимость выхода $y$ от входа $x$: $y = \Phi(x)$. Зависимость $\Phi(x)$ аппроксимируем через композицию последовательных преобразований.

Для простоты будем рассматривать обычные *полносвязные* сети со входом $x$, скрытыми(промежуточными) состояниями слоёв $\boldsymbol{h_i}$, функциями активации $a_i(\cdot)$ и выходом $y$:

$$\boldsymbol{h_0} = \boldsymbol{x}$$

$$\boldsymbol{h_i} = a_i(\boldsymbol{W_i} \cdot h_{i-1} + \boldsymbol{b_i}), i = \overline{1...n}$$

$$\boldsymbol{h_n} = \widehat{y}$$

$$L = \mathcal{L}(\widehat{y}, y),$$

где $\mathcal{L}(\cdot, \cdot)$ - функция ошибки.

Обозначим параметры модели на $i$-ом слое $\boldsymbol{\theta_i} = (\boldsymbol{W_i}, \boldsymbol{b_i})$, а параметры всей модели через $\boldsymbol{\Theta} = \{\boldsymbol{\theta_i} : \ i \ = \ \overline{1...n}\}$. Чаще всего нейронные сети принято рассматривать, как вычислительный граф/граф вычислений. Такой подход удобен с инженерной точки зрения, поскольку позволяет воспользоваться инструментом автоматического дифференцирования, и используется во всех современных фреймоворках: PyTorch, TensorFlow и прочие. Граф вычислений является ациклическим ориентированным графом, составленным из вершин-переменных и вершин-операций(Рисунок 1.1).

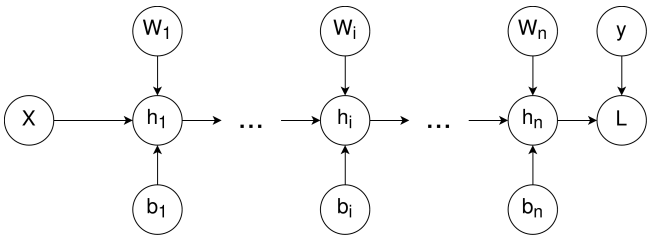


Рисунок 1.1 *Полносвязная* сеть в виде графа вычислений

Далее будем называть модели, основанные на графах вычислений, — графовыми моделями. Графы вычислений могут разных типов: статическими/динамическими, детерминированными/вероятностными и т.д. Для обучения/настройки параметров детерминированных графовых моделей используется метод *обратного распространения ошибки(back propagation)*, который широко используется в современном мире. Вкратце напомним алгоритм:

После прямого выполнения графа(*forward pass*), то есть в соответствии с направлениями рёбер на выходе мы получаем $L$ -значение функции ошибки, которые в зависимости от задач мы хотим либо минимизировать, либо максимизировать. Для этого мы пользуемся градиентными методами оптимизации, что требует вычисление градиентов $\frac{dL}{dW_i}, \frac{dL}{db_i}$ по нашим параметрам модели, где $i = \overline{1,n}$. В общем случае это трудная задача, однако в случае детерминированных графовых моделей мы можем использовать цепное правило(*chain rule*) для того, чтобы последовательно проталкивать градиенты, начиная с концевой вершины, содержащей $L$.

Например, для подсчёта градиентов $\frac{dL}{dW_i}, \frac{dL}{db_i}$ мы представим его в виде

$$\frac{dL}{dW_i} = \frac{dL}{dh_n} \cdot \frac{dh_n}{dW_i}$$

$$\frac{dL}{db_i} = \frac{dL}{dh_n} \cdot \frac{dh_n}{db_i}$$

Аналогично для всех остальных параметров модели мы будем проталкивать накопленный с концевой вершины градиент до соответствующих вершин и с помощью этого градиента высчитывать градиент по параметрам модели. Схему работы алгоритма обратного распространения ошибки можно увидеть на Рисунок 1.2.

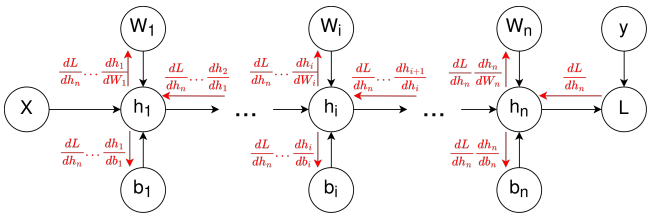


Рисунок 1.2 Обратное распространение ошибки по графу вычислений детерминированной полносвязной сети

Однако детерминированные нейронные сети обладают несколькими проблемами:

- Переобучение.
- Низкая интерпретируемость.
- Завышенная/заниженная уверенность модели в предсказаниях, даже если они неверные.
- Низкий уровень откалиброванности модели.

Указанные проблемы попытаемся решить с помощью байесовского подхода к нейронным сетям, который рассмотрим далее.