Машинное обучение в финансах

Лекция 12: Нейронные сети. Deep learning

Роман В. Литвинов*

*CRO
Финансовая Группа БКС

Высшая школа экономики, Июнь 2021

Содержание

Содержание

Нейрон. Перцептрон. Простейшая нейронная сеть

Интерлюдия: 'Can machine think like a human being?' Ричард Фейнман, леопард и автомобиль.

`https://www.youtube.com/watch?v=ipRvjS7q1DI`

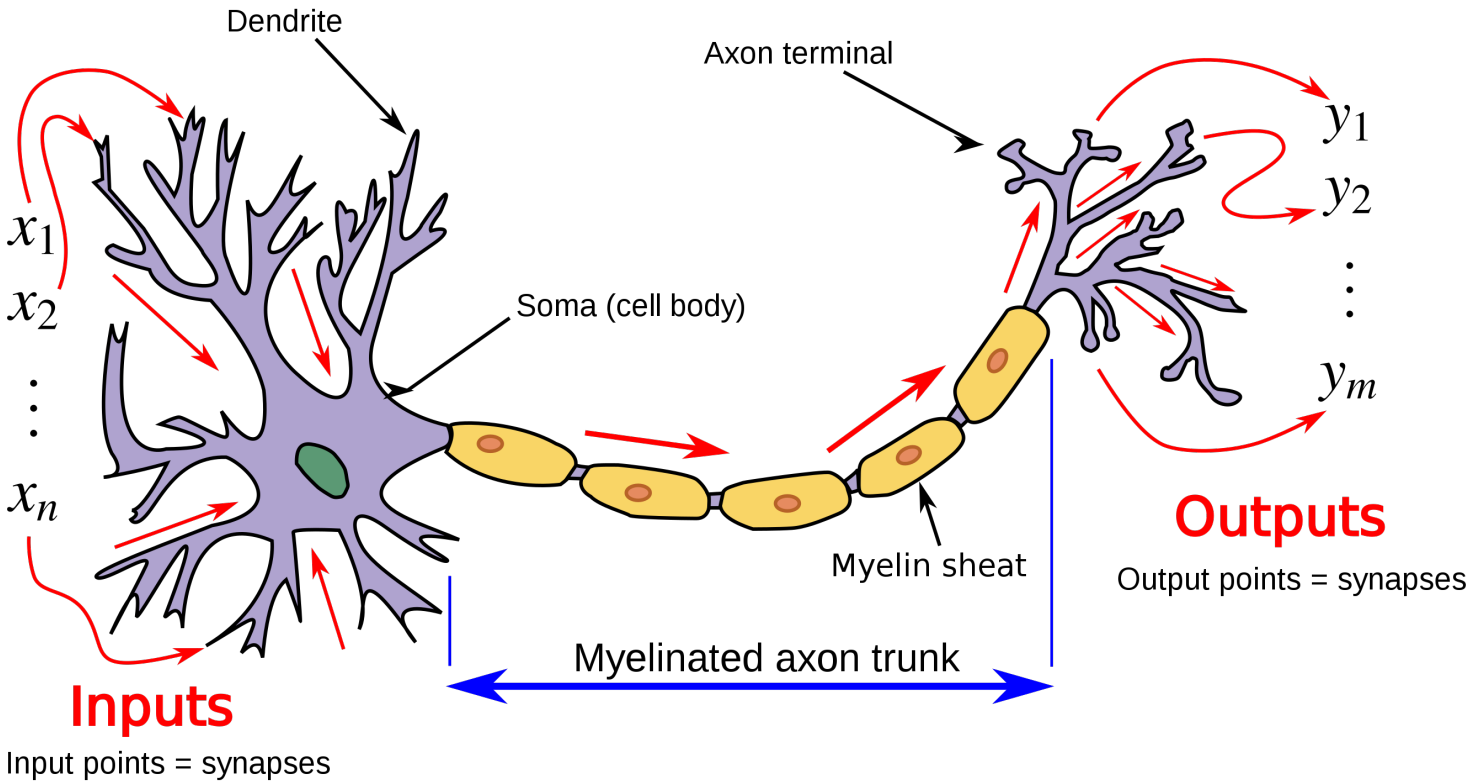
Сегодня мы рассмотрим финальную разновидность моделей машинного обучения нашего курса - нейронные сети.

Область применения этих моделей очень широка и не описывается только финансами. Сегодня нейронные сети и, в частности, глубокие нейронные сети успешно используются в широком спектре отраслей, начиная от беспилотных автомобилей и заканчивая машинным переводом, медициной и сельским хозяйством.

Они с успехом решают как задачи регрессии так и классификации.

Мы начнем рассмотрение нейронных сетей с простейшего базового блока - нейрона.

Нейрон. Перцептрон. Простейшая нейронная сеть



wikipedia.org

Нейрон. Перцептрон. Простейшая нейронная сеть

Для начала давайте попробуем представить как могла бы выглядеть простейшая модель одного нейрона. Она могла бы состоять из:

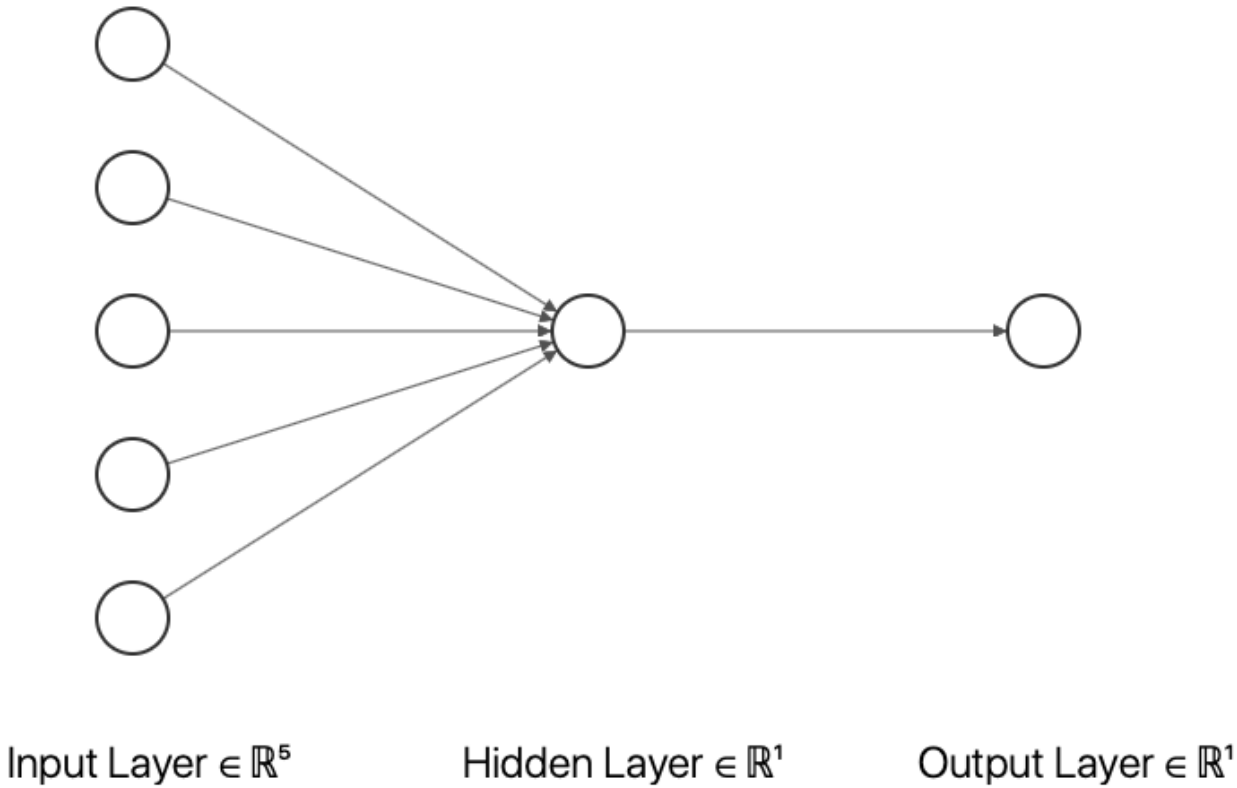
- синапсов - множества (вектора) X входных сигналов x_i, каждый из которых имел соотвествующий вес w_i;
- некого аналога клеточной мембраны, которая суммировала бы выходные сигналы/трансформировала в единый 'электрический' разряд

$$h = \sum_{i=1}^{n} w_i x_i$$

- функции активации (передаточной функции) (activation function), которая бы при заданном пороге активации решала бы 'загорелся'/ активировался нейрон или нет и возвращала 1 или 0, соотвественно.

Такая простейшая модель нейрона называется моделью МакКаллока-Питтса и визуализировать ее можно следующим образом:

Нейрон. Перцептрон. Простейшая нейронная сеть



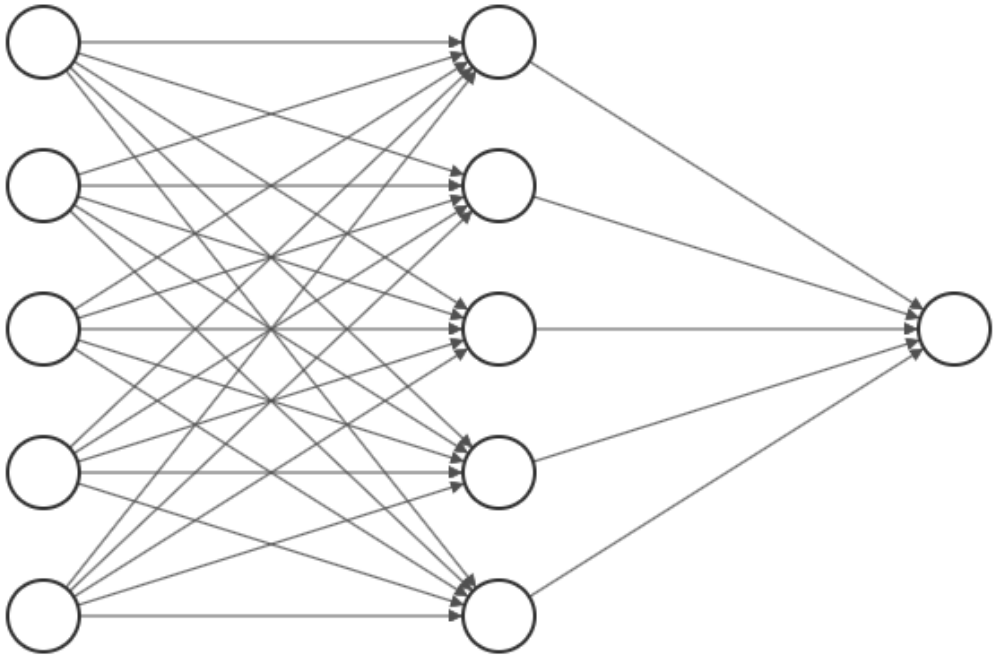
Нейрон. Перцептрон. Простейшая нейронная сеть

Перцептрон - это модель разрабатанная Фэнком Розенблаттом в 1957 году. Такая модель представляет собой не что иное как набор нейронов МаКаллока-Питтса и имеет следующую интерпретацию. Модель состоит из трех слоев/ типов элементов:

- первый слой - сенсоры/рецепторы - множество (вектор) X входных сигналов x_i;
- ассоциативный слой - каждому ассоциативному элементу соотвествует взвешенный набор сигналов с сенсоров. Ассоциативный элемент активируется (значение 1) только в случае, если количество сигналов на входе превысило некую пороговую величину.
- выходной слой - сигналы от ассоциативных элементов взвешиваются, сумируются и в случе превышения порогового значение на выходе регистрируется 1 (или 0 в противном случае).

Обучение происходит путем подбора оптимальных весов для сигналов сенсоров и ассоциативных элементов.

Нейрон. Перцептрон. Простейшая нейронная сеть



Input Layer $\in \mathbb{R}^5$ Hidden Layer $\in \mathbb{R}^5$ Output Layer $\in \mathbb{R}^1$

Нейрон. Перцептрон. Простейшая нейронная сеть

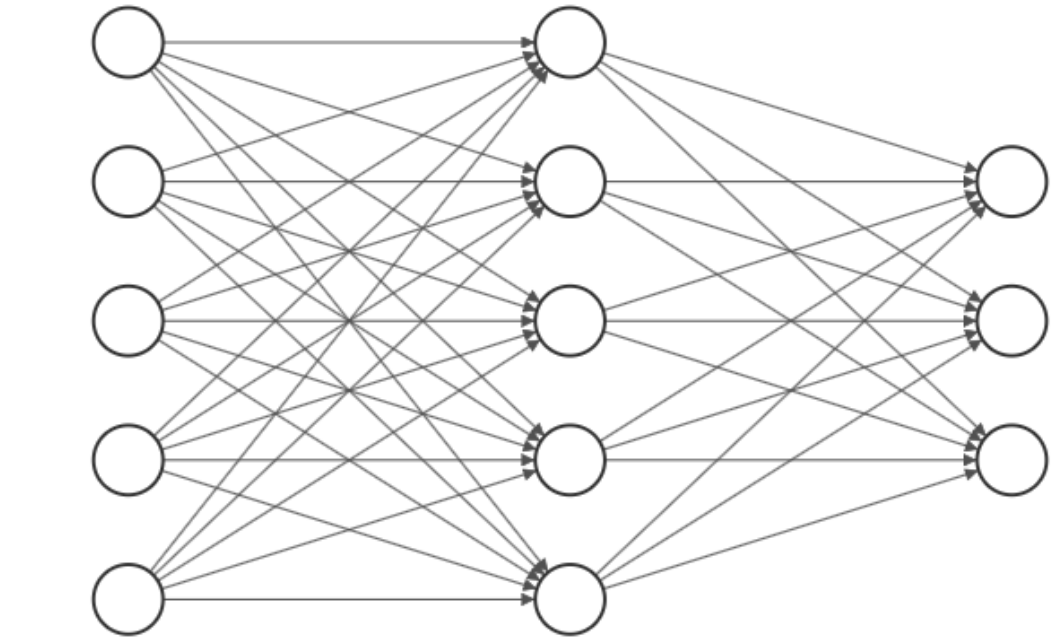
Давайте проанализируем полученную на предыдущих слайдах информацию.

Для начала очевидно, что важными составляющими такой простейшей модели являются веса элементов и бинарная функция активации. Также есть смысл отметить количество элементов в ассоциативном/скрытом (hidden) слое.

Изменив функцию активации, можно легко адаптировать модель к регрессионной задаче, когда на выходе мы будем получать не метку класса, а действительное число.

Например, если мы возьмем модель нейрона и изменим функцию активации на простое взвешивание входных сигналов и суммирование результата, то можем имитировать линейную регрессионную модель. В качестве критерия оптимизации здесь также можно использовать коэффициент SSE и минимизировать его значение с помощью подбора оптимальных весов.

Нейрон. Перцептрон. Простейшая нейронная сеть



Содержание

Нейронный сети с прямым распространением. Глубокие нейронные сети

Теперь двинемся в сторону усложнения нашей модели. Ранее мы рассматривали с вами (бинарную) функцию активации следующего вида:

$$f(h) = \begin{cases} 1 & h > \theta \\ 0 & h \leq \theta \end{cases} \tag{1}$$

где θ - пороговое значение функции активации.

Первое, что мы можем сделать, чтобы построить более изощренную модель это использовать вместо, или вместе с обозначенной выше функцией, другие функции активации.

Существует множество функций активации, и мы рассмотрим лишь самые распространенные.

Нейронный сети с прямым распространением. Глубокие нейронные сети

Такая, широкоиспользуемая функция, называется ReLU (от rectified linear unit):

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2}$$

Следущая, часто встречающаяся функция, - это гиперболический тангенс - tanh(x).

Также распространена логистическая функция активации или сигмоид:

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

Комбинация из таких функций активации позволяет построить разнообразные в тч нелинейные взаимосвязи между предикторами/ факторами и предсказывемой переменной.

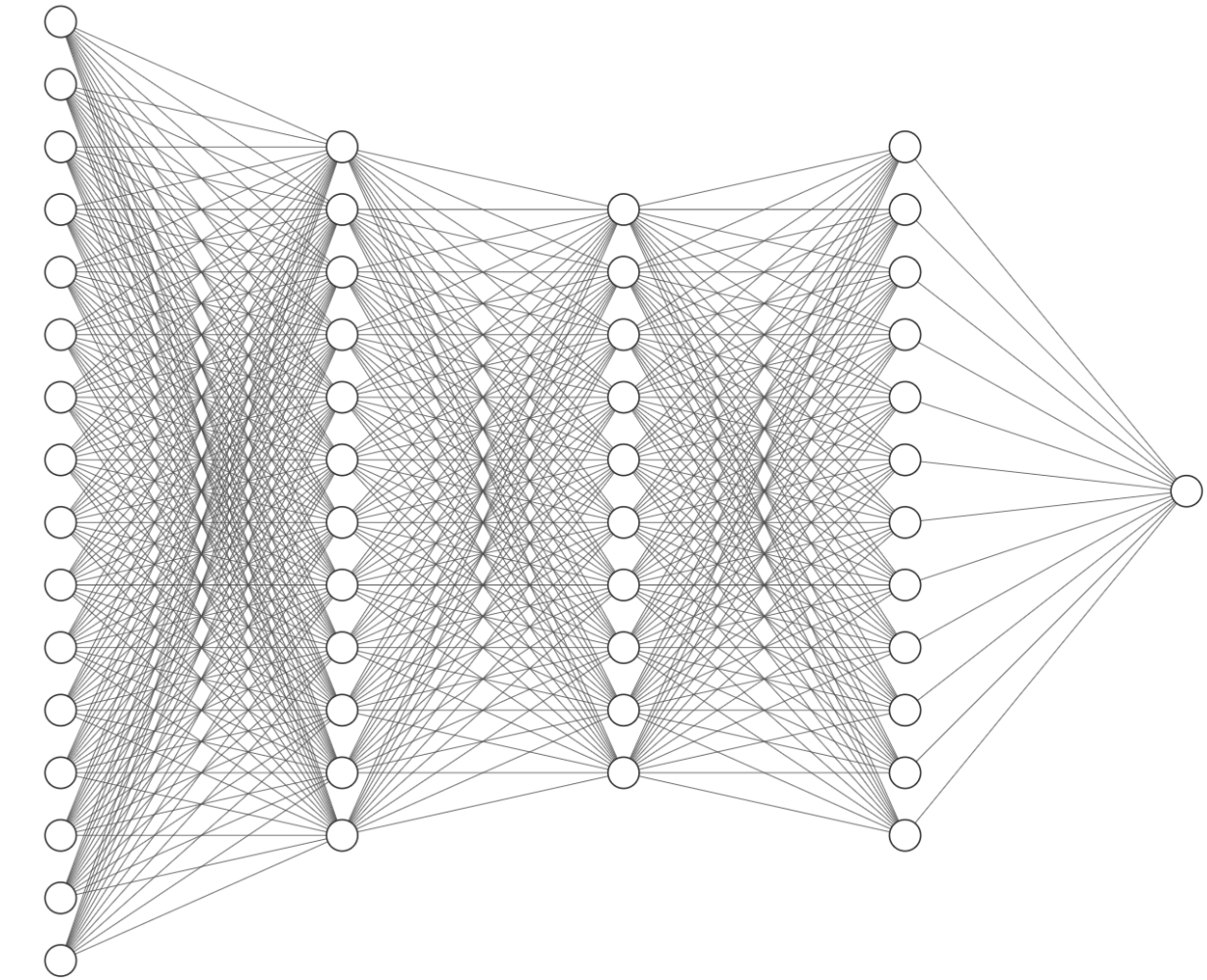
Нейронный сети с прямым распространением. Глубокие нейронные сети

Следущий этап в построении более продвинутой нейронной сети - это увеличение скрытых (hidden) слоев - слоев, которые находятся между входным и выходным слоями нашей нейронной сети.

Сигнал/информация в такой нейронной сети будет двигаться от входного слоя по нейронам до выходного слоя в одном направлении и не формируя циклов. Поэтому такая нейронная сеть называется сетью с прямым распространением (feed-forward neural network).

Чем большее количество скрытых слоев содержит нейронная сеть тем сложней (complexity) и изощренней становится наша модель. Собственно глубокими нейронными сетями (deep neural networks, DNNs) называют нейронные сети, содержащие большое количество скрытых (промежуточных) слоев нейронов.

Нейронный сети с прямым распространением. Глубокие нейронные сети



Содержание

Механизм обратного распространения (back propagation)

Понятно, что такая нейронная сеть обучается за счет поиска оптимальных весов, как для входных данных так и для нейронов в скрытых слоях. Но как?

Двумя основными механизмами процесса обучения нейронной сети являются обратное распространение (back propagation или backprop) и стохастический градиентный спуск (stochastic gradient descent). Давайте начнем с backprop.

Для начала разберем более подробно, что происходит в нейронной сети. На старте мы имеем данные размерности n с набором значений факторов/предикторов : $x_i \in R^n$. Т.е. это может быть как вектор (в нашем случае предсказания доходности, так и матрица, если например мы пытаемся классифицировать изображение). Обозначим этот вектор/матрицу как x.

Пусть W_k где $k \in (1, 2, ..., m)$ - матрица, определяющая веса 'соединяющие' k-тый и (k+1)-ый слой нашей нейронной сети.

Механизм обратного распространения (back propagation)

Обозначим входные данные k-того слоя нашей нейронной сети как x^k и как f_k функцию активации k-того слоя. Связь между слоями нейронной сети тогда будет определяться следующей рекурсивной формулой:

$$x^k = f_k(W_k, x^{k-1})$$

Первый скрытый слой будет тогда вычисляться по следующей формуле:

$$x^1 = f_1(W_1, x)$$

А последний, выходной слой, по формуле:

$$\hat{y} = f_k(W_k, x^{k-1})$$

Вся работа нейронной сети будет описываться следующей композицией (вложенных) функций:

$$\hat{y} = f_k(W_k, ...f_2(W_2, f_1(W_1, x))....)$$

Механизм обратного распространения (back propagation)

Теперь нам необходимо задать целевую функцию (loss function), характеризующую качество прогноза нашей нейронной сети, например следующую для задачи регрессии:

$$E(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Наша задача, как мы помним, это обучение нейронной сети - подбор оптимальных весов минимизирующих значение целевой функции. Алгоритм обратного распространения (backpropagation) используется именно для этой задачи.

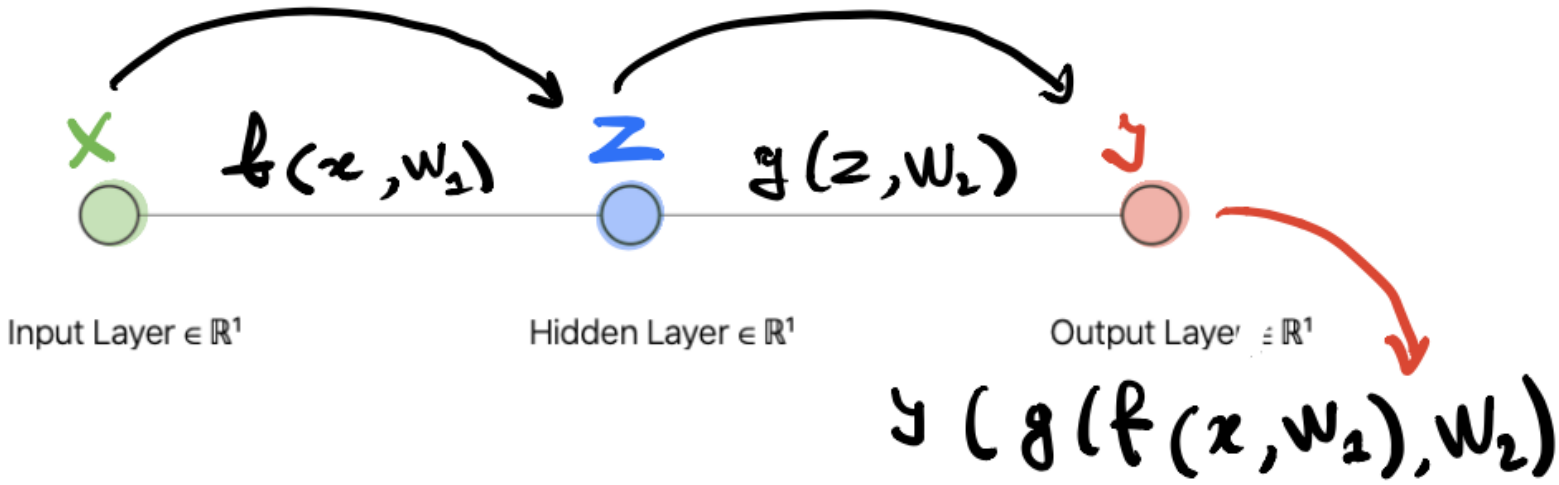
Обратное распространение базируется на уже знакомой нам идее дифференцирования сложной функции.

Для задачи подбора оптимальных весов (лекция 7) используется механизм градиентного спуска. Нам необходимо расчитать значения градиента функции и изменять веса двигаясь в направлении, противоположном знаку градиента.

Механизм обратного распространения (back propagation)

Для вычисления градиента нам потребуется вычисление частных производных функции описывающей динамику нейронной сети, а, как вы помните, она представляет собой сложную композицию из большого количества функций активации. Здесь нам и понадобится дифференцирование сложной функции.

Давайте рассмотрим механизм обратного распространения на следующем простом примере нейронной сети с одним скрытым слоем:



Механизм обратного распространения (back propagation)

Как мы видим, наша нейронная сеть характеризуется сложной функцией - функцией от функции (хотя и не такой громоздкой как в примере ранее):

$$\hat{y} = g(z, w_2) = g(f(x, w_1), w_2)$$

Нам необходимо вычислить частные производные и понять насколько оценка ошибки E чувствительна к изменению весов w_1 и w_2.

Используя правило дифференцирования сложной функции, получим:

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1}$$

Ошибочное влияние (вклад изменения заданного веса в оценку ошибки прогноза) распространяется справа налево - в обратном направлении информационного потока нейронной сети (backpropagation).

Механизм обратного распространения (back propagation)

Напомню о чем мы говорили на 7 лекции, когда разбирали как работает механизм градиентного спуска. Градиент (gradient) представляет собой вектор содержащий все частные производные функции. Обозначается в нашем случае $\nabla_w E(w)$.

Критическими точками нашей функции $E(w)$ будут точки, в которых каждый элемент градиента принимает нулевое значение.

Градиентный спуск будет работать в таком случае следующим образом. Новое значение вектора содержащего веса будет равно:

$$w' = w - \lambda \nabla_w E(w)$$

Параметр λ здесь представляет собой, так называемый, коэффициент скорости обучения (learning rate) и задает насколько агрессивно/быстро мы будем двигаться в направлении противоволожном знаку градиента/производной.

Механизм обратного распространения (back propagation)

Для нашего простого примера с двумя весовыми коэффциентами и градиентом состоящим из двух частных производных веса будут обновляться по следующим правилам:

$$w_1^{'} = w_1 - \lambda \frac{\partial E}{\partial w_1}$$

$$w_2^{'} = w_2 - \lambda \frac{\partial E}{\partial w_2}$$

а частные производные будут вычисляться с помощью механизма обратного распространения. Т.е.:

$$w_1^{'} = w_1 - \lambda \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1}$$

$$w_2^{'} = w_2 - \lambda \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2}$$

Содержание

Стохастический градиентный спуск (stochastic gradient descent)

Давайте вкратце суммируем механизм работы алгоритма градиентного спуска:

- Шаг 1: Случайным образом генерируется набор/матрица стартовых весов.
- Шаг 2: вычисляется градиент функции потерь $\nabla_w E(w)$
- Шаг 3: происходит процедура обновления/апдейта весов по правилу:

$$w' = w - \lambda \nabla_w E(w)$$

- Шаг 4: шаг 2 и 3 повторяются пока не будет достигнут критерий схождения алгоритма (количество итераций, стабилизация ошибки и тп)

Стохастический градиентный спуск (stochastic gradient descent)

Следует отметить, что вычисление полноценного градиента для всех n-точек тренировочного датасета в случае глубокой нейронной сети, содержащей большое количество слоев, а следовательно весов, это очень трудоемкое задание с вычислительной точки зрения.

Поэтому была придумана модификация алгоритма градиентного спуска называемая стохастический градиентный спуск (stochastic gradient descent) или пакетный стохастический градиентный спуск (batch gradient descent).

Суть модифицированного метода в том, что градиент вычисляется не для всех n точек, а только для одной или подмножества $k < n$ точек. Полученное значение градиента используется (в пакетном градиенте усредняется и используется) в качестве аппроксимации для других точек.

Стохастический градиентный спуск (stochastic gradient descent)

Алгоритм стохастического/пакетного стохастического градиентного спуска в общем виде выглядит так:

- Шаг 1: Случайным образом генерируется набор стартовых весов.
- Шаг 2: вычисляется градиент функции потерь в случайном образом выбранной точке i или подмножестве k-точек и результат усредняется
- Шаг 3: происходит процедура обновления/апдейта весов с использованием аппроксимации градиента полученной на шаге 2
- Шаг 4: шаг 2 и 3 повторяются пока не будет достигнут критерий схождения алгоритма.

Безусловно, для схождения алгоритма может потребоваться прогонка по всему массиву данных несколько раз, но при этом используется гораздо более простые и быстрые вычисления, нежели полноценная оценка градиента глубокой нейронной сети.

Содержание

Теорема Колмогорова-Арнольда о представлении непрерывных функций

Одним важным результатом из области математики о котором я хотел бы вам рассказать является теорема Колмогорова-Арнольда о представлении непрерывных функций.

Теорема (Колмогоров-Арнольд, 1957)

Пусть f — это многомерная непрерывная функция. Тогда f можно записать в виде конечной композиции непрерывных функций одной переменной и операции сложения:

$$f(x_1, x_2, ..., x_n) = \sum_{i=1}^{2n+1} f_i(\sum_{j=1}^{n} g_{ij}(x_j))$$

Она говорит о том, что любую многомерную функцию можно представить как суперпозицию более простых одномерных функций. Это вам ничего не напоминает?

Теорема Колмогорова-Арнольда о представлении непрерывных функций

Это фундаментальная теорема была доказана в 1957 году (и решила 13-ю проблему Гильберта), но неожиданно обрела свежее прочтение в последние несколько лет, когда начались оживленные дискуссии по поводу того, почему эффективны глубокие нейронные сети.

Действительно. Наша нейронная сеть и является композицией таких простых функций, с помощью которых мы пытаемся аппроксимировать более сложную функцию, отражающую взаимосвязь между набором предикторов/факторов и предсказываемой переменной. Т.е. нейронная сеть является своего рода универсальным аппроксиматором.

К сожалению, теорема Колмогорова-Арнольда ничего не говорит о том, как построить такую заданную функцию-аппроксиматор, только о том, что такая функция существует.

Содержание

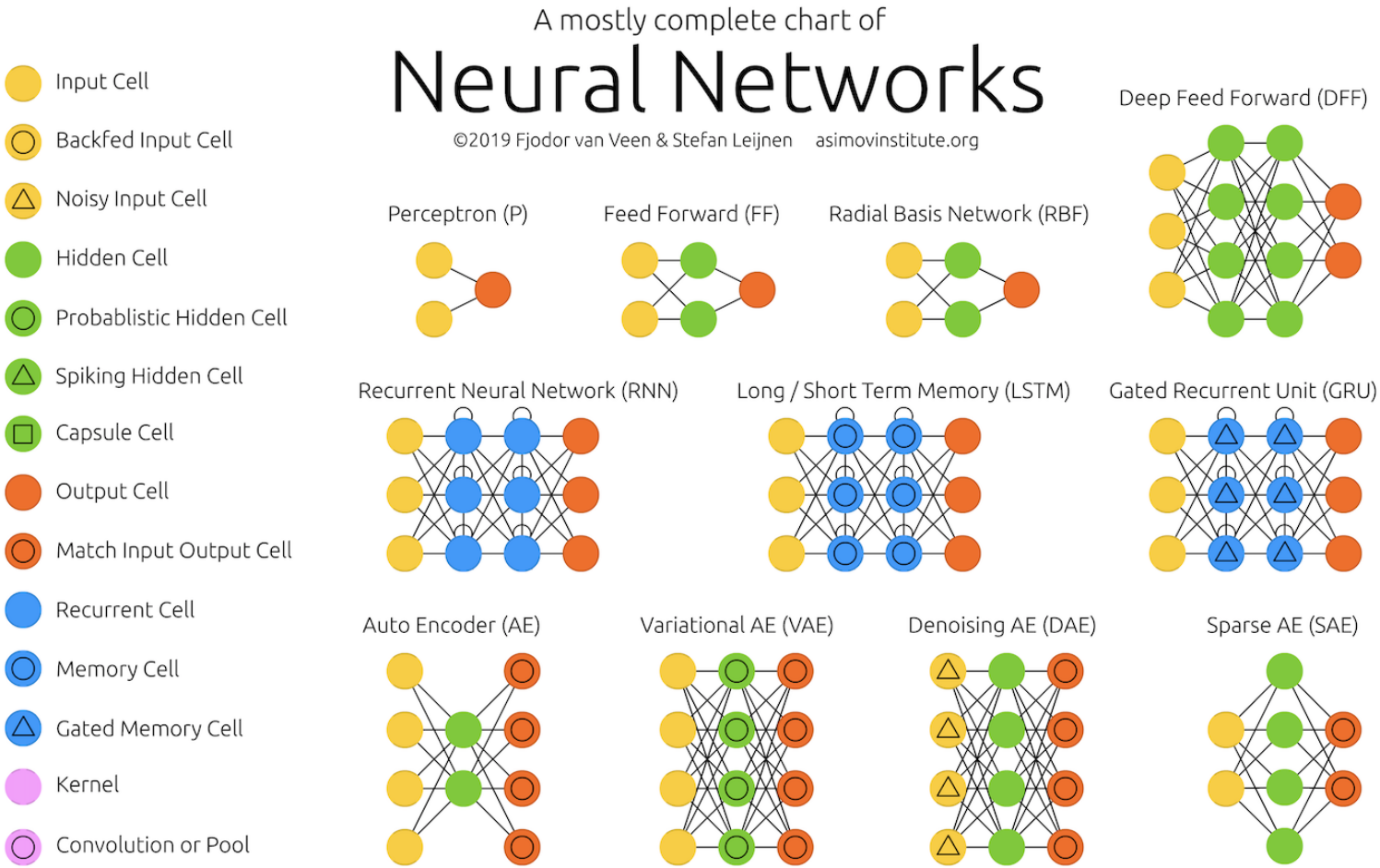
Топология нейронных сетей. Neural networks zoo

Мне еще много хотелось бы рассказать вам о нейронных сетях и эта тема действительно заслуживает отдельного по объему курса. Архитектура и топология нейронных сетей не заканчивается на примерах рассмотренных нами сегодня (deep feed-forward neural networks). В нейронных сетях могут использоваться петлевые связи, механизмы краткосрочной и долгосрочной памяти, сверточные механизмы и тп.

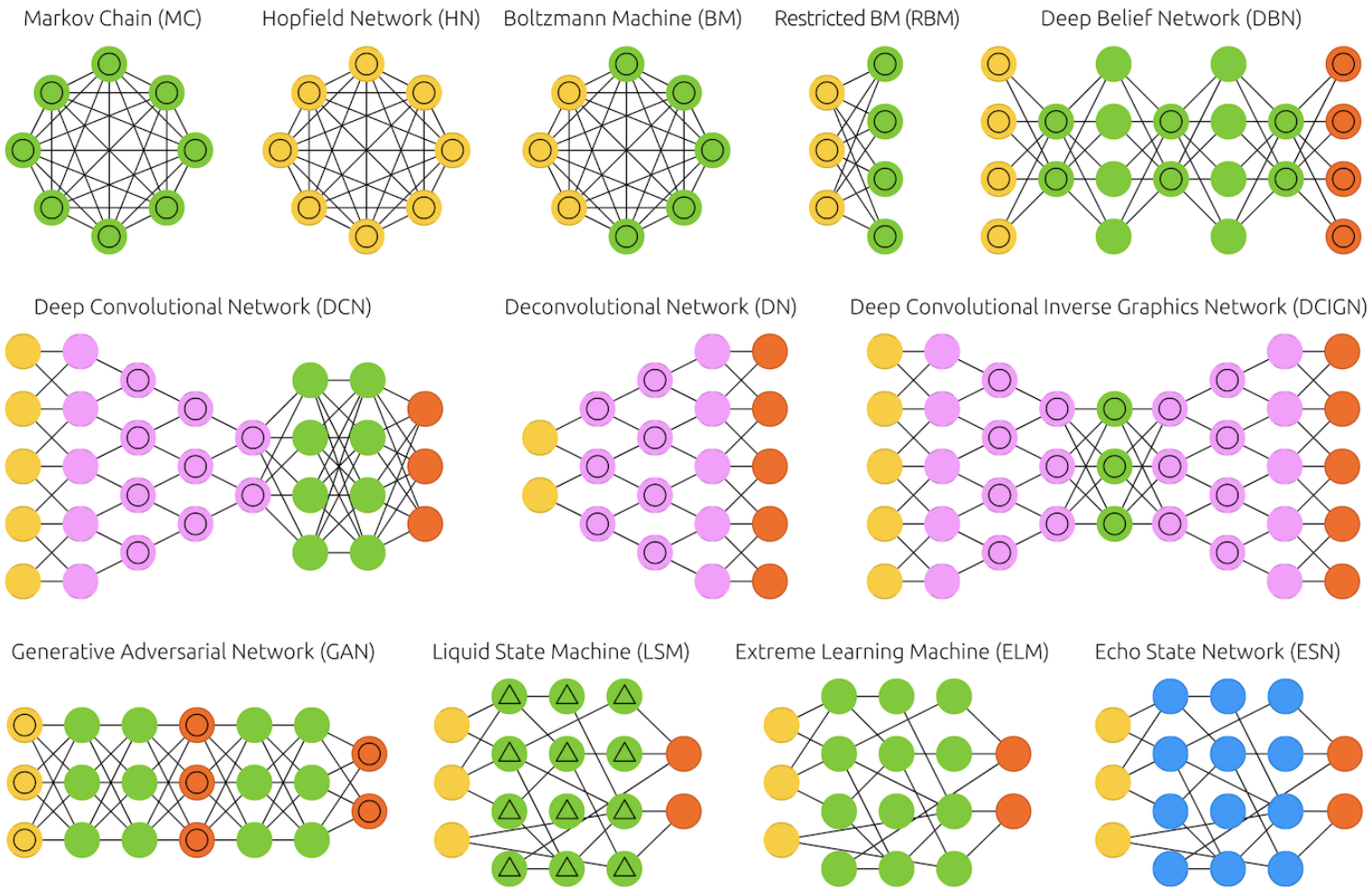
Структура нейронных сетей разнообразна и часто диктуется задачей, которая стоит перед инженером - распознование образов, речи или текста, машинный перевод, работа с видео, звуком и тп. Архитектор нейронных сетей - новая и хорошо оплачиваемая специальность.

Ниже приведены графические схемы распространенных нейронных сетей. Вы можете скачать и более подробно ознакомится с этим материалом по адресу:
`https://www.asimovinstitute.org/neural-network-zoo/`

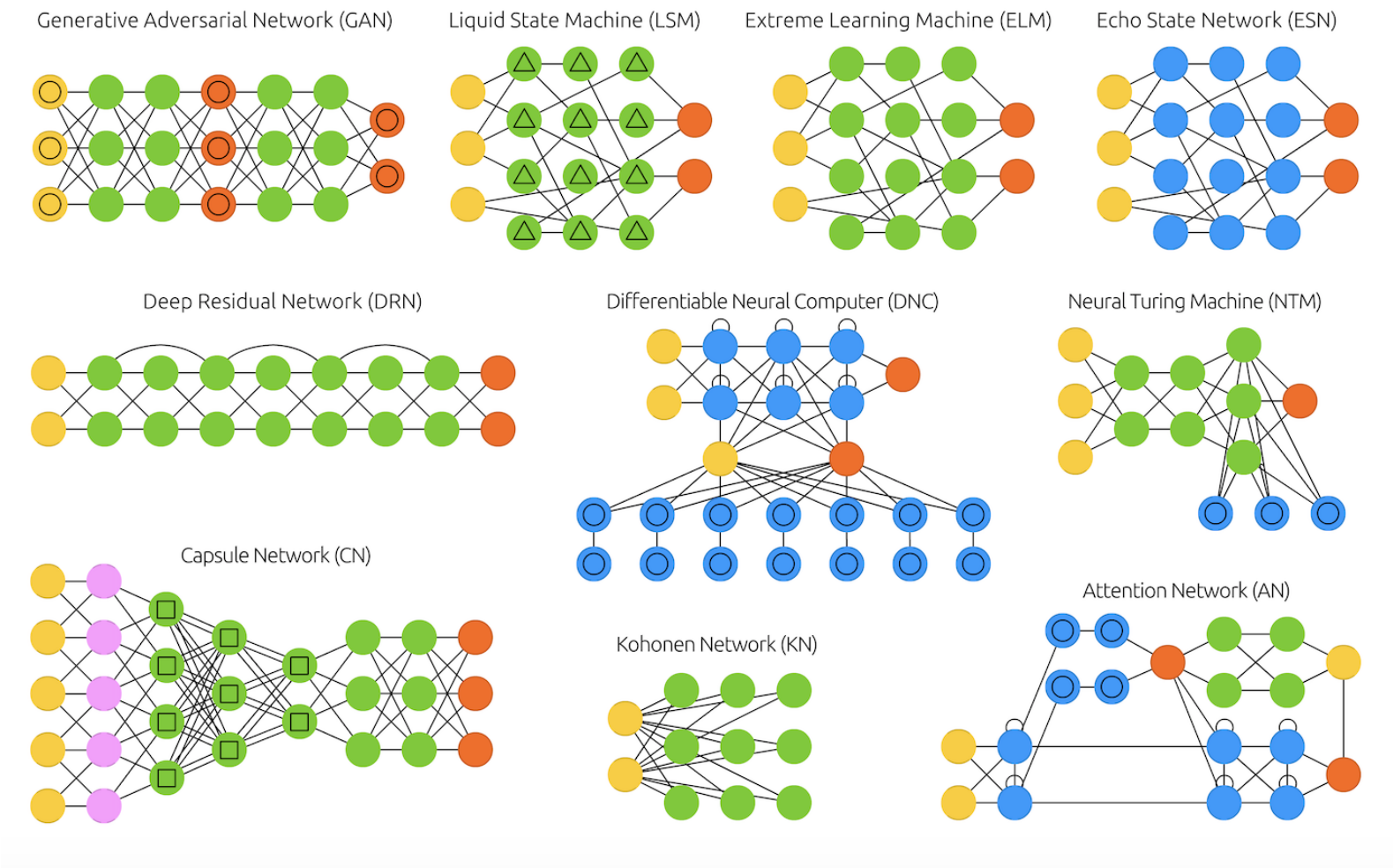
Топология нейронных сетей. Neural networks zoo



Топология нейронных сетей. Neural networks zoo



Топология нейронных сетей. Neural networks zoo



Распространенными пакетами для работы с нейронными сетями являются TensorFlow, Pythorch и библиотека Sklearn.

Литература

Дополнительная литература к сегодняшней лекции:

- Колмогоров, А.Н. (1957) 'О представлении непрерывных функций нескольких переменных в виде суперпозиций непрерывных функций одного переменного и сложения', Докл. АН СССР, 114:5, 953–956
- Deisenroth, M. Fisal, A., Ong, C.S. (2020) Mathematics for machine learning. Cambridge.
- Goodfellow, I. Bengio, Y., Courville, A. (2016). Deep learning. MIT. (есть на русском)
- Kroese, D., Botev, Z. (2019) Data Science and Machine Learning. Chapman and Hall/CRC
- Marsland, S. (2015) Machine learning. An algorithmic perspective. Chapman and Hall/CRC
- Trask, A. (2019) Grokking deep learning. Manning (есть на русском)