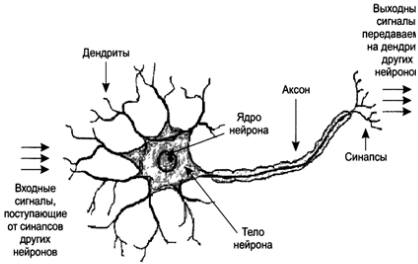
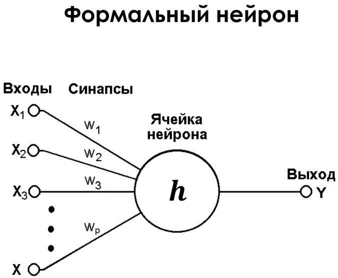
Neural Nets (NN), с элементами Deep Learning

Е. Ларин, Ф. Ежов, И. Кононыхин

Санкт-Петербургский государственный университет
Прикладная математика и информатика
Вычислительная стохастика и статистические модели

Математическая модель нейрона

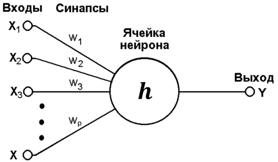




$$h = \sum_i x_i * w_i \qquad y = f(h)$$

Перцептрон

Формальный нейрон



$$h = \sum_i x_i * w_i \qquad y = f(h)$$

Перцептрон(perceptron) — одна из простейших архитектур ANN, придуманная Фрэнком Розенблаттом в 1957 году.

- p — количество признаков
- $\{x_1, \cdots, x_p\}$ — индивид.
- $\{w_1, \cdots, w_p\}$ — веса нейрона.
- $$f(h) = \begin{cases} 0 & h < 0 \\ 1 & h \geqslant 0 \end{cases}$$

  — функция активации.
- y — выход сети.

Перцептрон: Проблемы

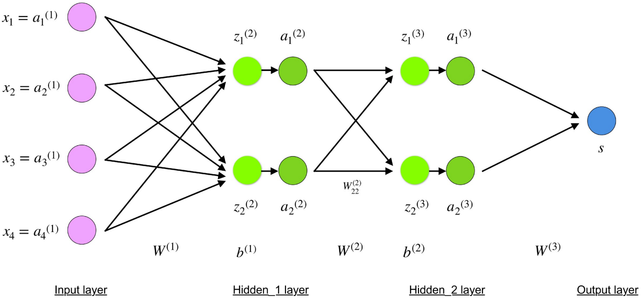
Перцептрон может решать только линейно разделимые задачи.

Проблема

Перцептрон не способен решить некоторые тривиальные задачи, например задачу классификации на основе исключающего "ИЛИ" (Exclusive OR - XOR).

MultiLayer Perceptron (MLP)

Каждый слой кроме выходного включает нейрон смещения и полностью связан со следующим слоем.



MultiLayer Perceptron (MLP): Теорема Цыбенко

Функция $\sigma(z)$ — сигмоида, если $\lim\limits_{z \to -\infty} \sigma(z) = 0$ и $\lim\limits_{z \to +\infty} \sigma(z) = 1$.

Теорема Цыбенко

Если функция активации $\sigma(z)$ — непрерывная сигмоида, то для любой непрерывной на $[0, 1]^p$ функции $f(x)$ существуют такие значение параметров $w_h \in \mathbb{R}^p$, $w_0 \in \mathbb{R}$, $\alpha_h \in \mathbb{R}$, что двухслойная сеть

$$a(x) = \sum_{h=1}^{H} \alpha_h \sigma(\langle x, w_h \rangle + w_0)$$

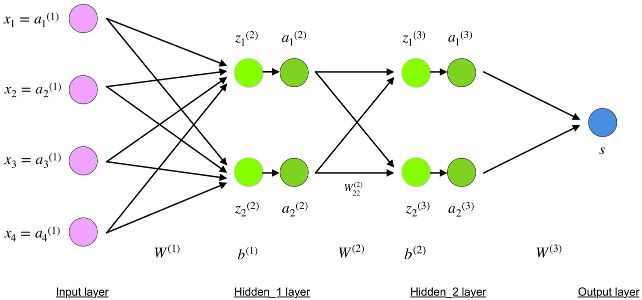
равномерно приближает $f(x)$ с любой точность ε:

$$|a(x) - f(x)| < \varepsilon, \text{для всех} x \in [0, 1]^p$$

MultiLayer Perceptron (MLP): Функции представимые нейросетью

- Двухслойная сеть в $\mathbb{R}^p$ позволяет отделить произвольный выпуклый многогранник.
- Трехслойная сеть в $\mathbb{R}^p$ позволяет отделить произвольную многогранную область, не обязательно выпуклую, и даже не обязательно связную.
- С помощью линейных операций и одной нелинейной функции активации можно приблизить любую непрерывную функцию с любой желаемой точностью.

Обучение MLP



Минимизация средних потерь на обучающей выборке:

$$Q(w) := \frac{1}{n}\sum_{i=1}^{n}\mathcal{C}_i(w) \to \min_w$$

$\mathcal{C}$ — любая дифференцируемая функция потерь.

Алгоритм SG

Задача минимизации: $Q(w) := \frac{1}{n}\sum_{i=1}^{n}\mathcal{C}_i(w) \to \min\limits_{w}$

Вход: выборка X^n; «скорость» обучение ϵ; параметр λ;

Выход: оптимизированные значение весов w.

1. инициализация весов w и текущую оценку $Q(w)$
2. повторять
3. выбрать объект x_i из X^n;
4. вычислить потерю $\mathcal{C}_i := \mathcal{C}_i(w)$
5. градиентный шаг: $w := w - \epsilon\mathcal{C}_i'(w)$
6. оценить значение функционала: $Q := (1-\lambda)Q + \lambda\mathcal{C}_i$
7. пока значение Q и/или весов w не стабилизируются

Обучение MLP: forward-propagation

$x = a^{(1)}$ *Input layer*

$z^{(2)} = W^{(1)}x + b^{(1)}$ *neuron value at $Hidden_1$ layer*

$a^{(2)} = f(z^{(2)})$ *activation value at $Hidden_1$ layer*

$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$ *neuron value at $Hidden_2$ layer*

$a^{(3)} = f(z^{(3)})$ *activation value at $Hidden_2$ layer*

$s = W^{(3)}a^{(3)}$ *Output layer*

$$C = cost(s, y)$$

s — полученная оценка с помощью нейронной сети. y — правильный ответ. $\mathcal{C}$ — значение ошибки между s и y по какой-нибудь метрики (например MSE).

Обучение MLP: back-propagation

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \qquad \textit{chain rule}$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad \textit{by definition}$$

m – *number of neurons in* $l-1$ *layer*

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad \textit{by differentiation (calculating derivative)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1} \qquad \textit{final value}$$

Обучение MLP: процесс изменения весов

$$while\ (termination\ condition\ not\ met)$$

$$w := w - \epsilon \frac{\partial C}{\partial w}$$

$$b := b - \epsilon \frac{\partial C}{\partial b}$$

$$end$$

back-propagation algorithm: в итоге

Преимущества:

- быстрое вычисление градиента
- обобщение на любые функции активации, функции потерь и количество слоев
- возможность потокового обучения
- возможность распараллеливания

Недостатки:

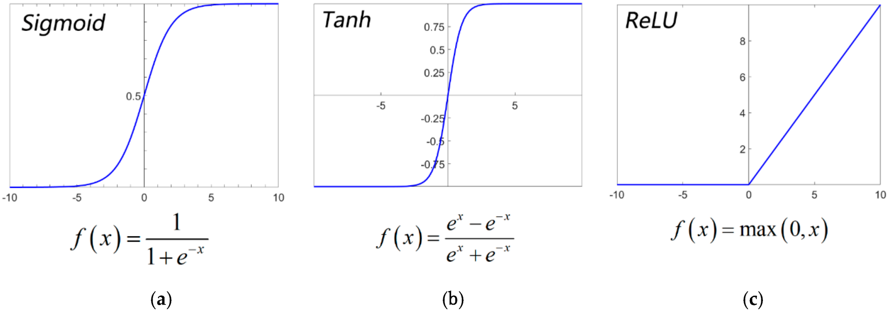
- медленная сходимость
- застревания в локальных экстремумах
- проблема переобучения

Методы решения задачи минимизации

- Stochastic Gradient (SG)
- Метод накопления импульса (Momentum)
- NAG (Nesterov's accelerated gradient)
- RMSProp (running mean square)
- AdaDelta (adaptive learning rate)
- Adam (adaptive momentum)
- Nadam (Nesterov-accelerated adaptive momentume)

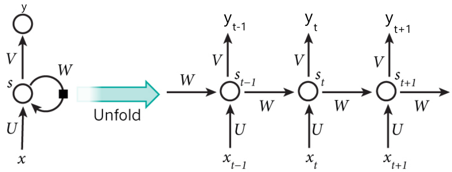
Функции активации

Функции активации добавляют в модель нелинейность.



Recurrent neural network(RNN)

x_t — входной вектор в момент времени t
s_t — вектор скрытого слоя в момент времени t
y_t — выходной вектор (в некоторых случаях $y_t \equiv s_t$)



Задача:

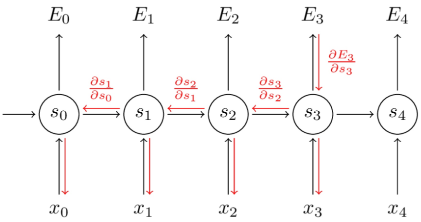
$$\sum_{t=0}^{T} \mathcal{C}_t(U, V, W) \to \min_{U,V,W}$$

$\mathcal{C}_t(U, V, W) = \mathcal{C}(y_t(U, V, W))$ — потеря от предсказания y_t.

Обучение RNN

Back-propagation Through Time (BPTT)

$$\frac{\partial \mathcal{C}_t}{\partial W} = \frac{\partial \mathcal{C}_t}{\partial y_t}\frac{\partial y_t}{\partial s_t}\sum_{k=0}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial s_i}{\partial s_{i-1}}\right)\frac{\partial h_k}{\partial W}$$

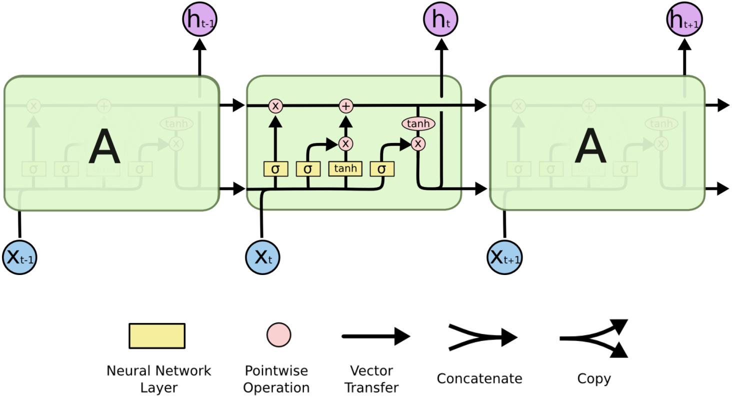


Где используется RNN?

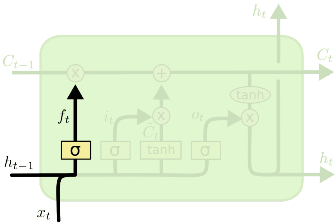
- Прогнозирование временных рядов
- Управление технологическими процессами
- Классификация текстов или их фрагментов
- Анализ тональности документа / предложений / слов
- Машинный перевод
- Распознавание речи
- Синтез речи
- Синтез ответов на вопросы, разговорный интеллект
- Генерация подписей к изображениям
- Генерация рукописного текста
- Интерпретация генома и другие задачи биоинформатики

Long short-term memory (LSTM)

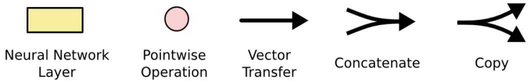
Мотивация LSTM: сеть должна долго помнить контекст, какой именно — сеть должна выучить сама.
Вводится C_t — вектор состояния сети в момент t.



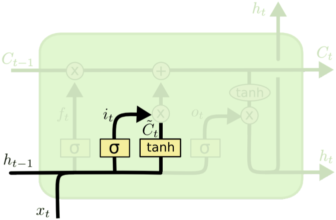
Объясняя LSTM (часть 1)



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

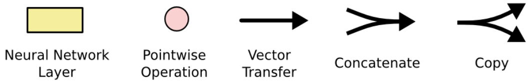


Объясняя LSTM (часть 2)

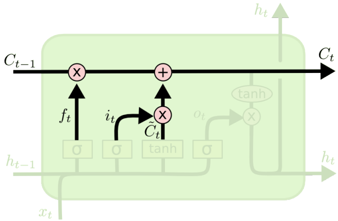


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

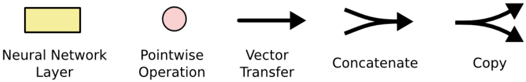
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



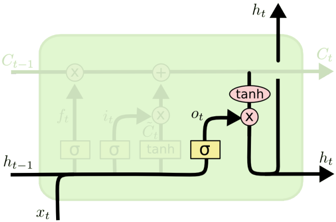
Объясняя LSTM (часть 3)



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

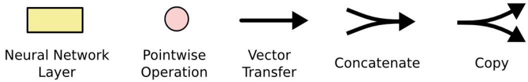


Объясняя LSTM (часть 4)



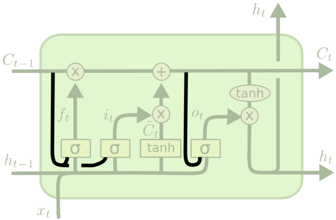
$$o_t = \sigma\left(W_o\ \left[h_{t-1}, x_t\right]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



Модификации LSTM

LSTM с «замочными скважинами» (peepholes)



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

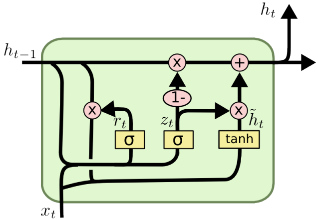
Все фильтры «подглядывают» вектор состояния C_{t-1} или C_t.

Увеличивается число параметров модели.

Замочную скважину можно использовать не для всех фильтров.

Модификации LSTM

LSTM Gated recurrent units (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Используется только состояние h_t, вектор C_t не вводится.

Фильтр обновления (update gate) вместо входного и забывающего.

Фильтр перезагрузки (reset gate) решает, какую часть памяти нужно перенести дальше с прошлого шага.

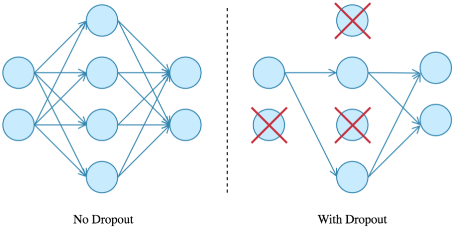
Dropout

Обучение: обнуляем выход h-ого нейрона ℓ-го слоя с вероятностью p_ℓ.

$$a_{ih}^{\ell+1} = \xi_h^\ell f_h^\ell \Big(\sum_j w_{jh} a_{ij}^\ell \Big), \; P(\xi_h^\ell = 0) = p_\ell$$

Применение: вводим поправку

$$a_{ih}^{\ell+1} = (1 - p_\ell) f_h^\ell \Big(\sum_j w_{jh} a_{ij}^\ell \Big)$$



Dropout

Inverted Dropout:

$$a_{ih}^{\ell+1} = \frac{1}{(1-p_\ell)} \xi_h^\ell f_h^\ell \left(\sum_j w_{jh} a_{ij}^\ell\right)$$

- регуляризация: из всех сетей выбираем более устойчивую к утрате pN нейронов.
- сокращаем переобучение, заставляя разные части сети решать одну и ту же исходную задачу.

Batch Normalization

$B = x_i$ — пакеты (mini-batch) данных. Усреднение градиентов $\mathcal{C}(w)$ по пакету ускоряет сходимость.
$B^\ell = \{u_i^\ell\}$ — векторы объектов x_i на выходе ℓ-го слоя.
Batch Normalization:

1. Нормировать каждую j-ю компоненту вектора u_i^ℓ по пакету:

$$\hat{u}_{ij}^\ell = \frac{u_{ij}^\ell - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}; \quad \nu_j = \frac{1}{|B|}\sum_{x_i \in B} u_{ij}^\ell; \quad \sigma_j^2 = \frac{1}{|B|}\sum_{x_i \in B}(u_{ij}^\ell - \nu_j)^2.$$

2. Добавить линейный слой с настраиваемыми весами:

$$\tilde{u}_{ij}^\ell = \gamma_j^\ell \hat{u}_{ij}^\ell + \beta_j^\ell$$

3. Параметры γ_j^ℓ и β_j^ℓ настраиваются с помощью back-propagation.