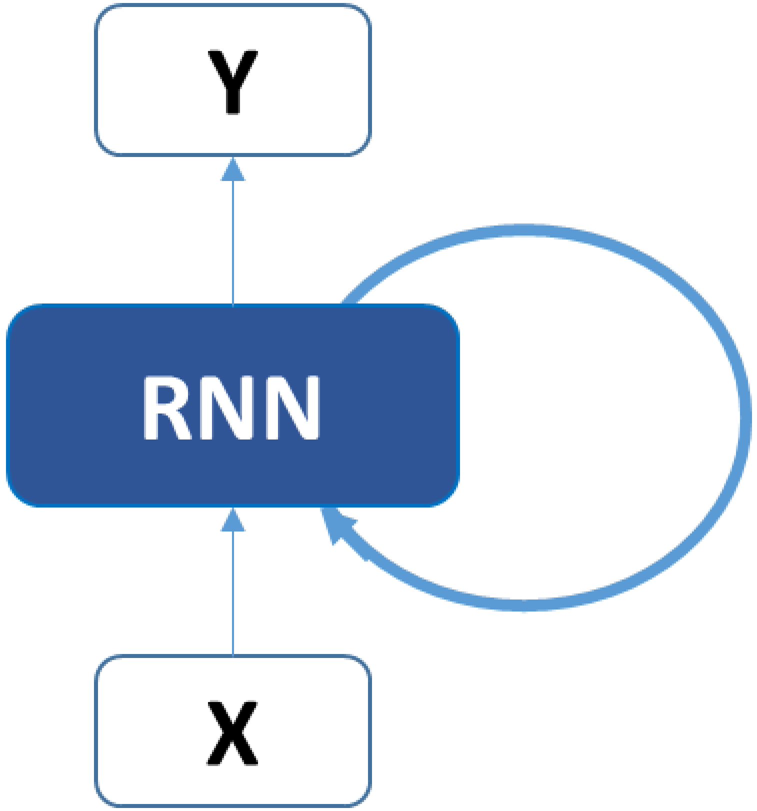
Deloitte.

Analytics Institute



Введение в рекуррентные нейронные сети

Кирилл Цыганов, Ведущий инженер анализа данных

cisdai.ru

Нейрон – это функция со многими входами и одним выходом (скаляром)

Формула

функция активации

$$Neuron\left(\begin{pmatrix} X1 \\ X2 \\ X3 \\ X4 \end{pmatrix}\right) = f\left((w_1\ w_2\ w_3\ w_4\ b) \cdot \begin{pmatrix} X1 \\ X2 \\ X3 \\ X4 \\ 1 \end{pmatrix} \right) = f\underbrace{(w_1 \cdot X1 + w_2 \cdot X2 + w_3 \cdot X3 + w_4 \cdot X4 + b)}_{}$$

Аргументом функции активации является ***скаляр***

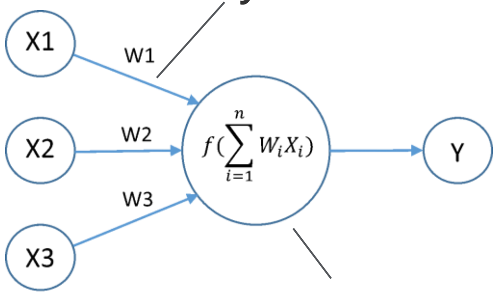
или

$$Neuron\,(X) = f\,(WX + b)$$

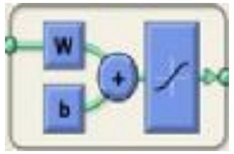
$$\begin{pmatrix} w_1 & w_2 & w_3 & w_4 & b \\ & & * & & \\ & & * & & \\ & & & & \\ & & & & \end{pmatrix} \times \begin{pmatrix} X1 \\ X2 \\ X3 \\ X4 \\ 1 \end{pmatrix}$$

Схематичное изображение

Стрелка интерпретируется, как ***умножение на коэф.*** *w1*

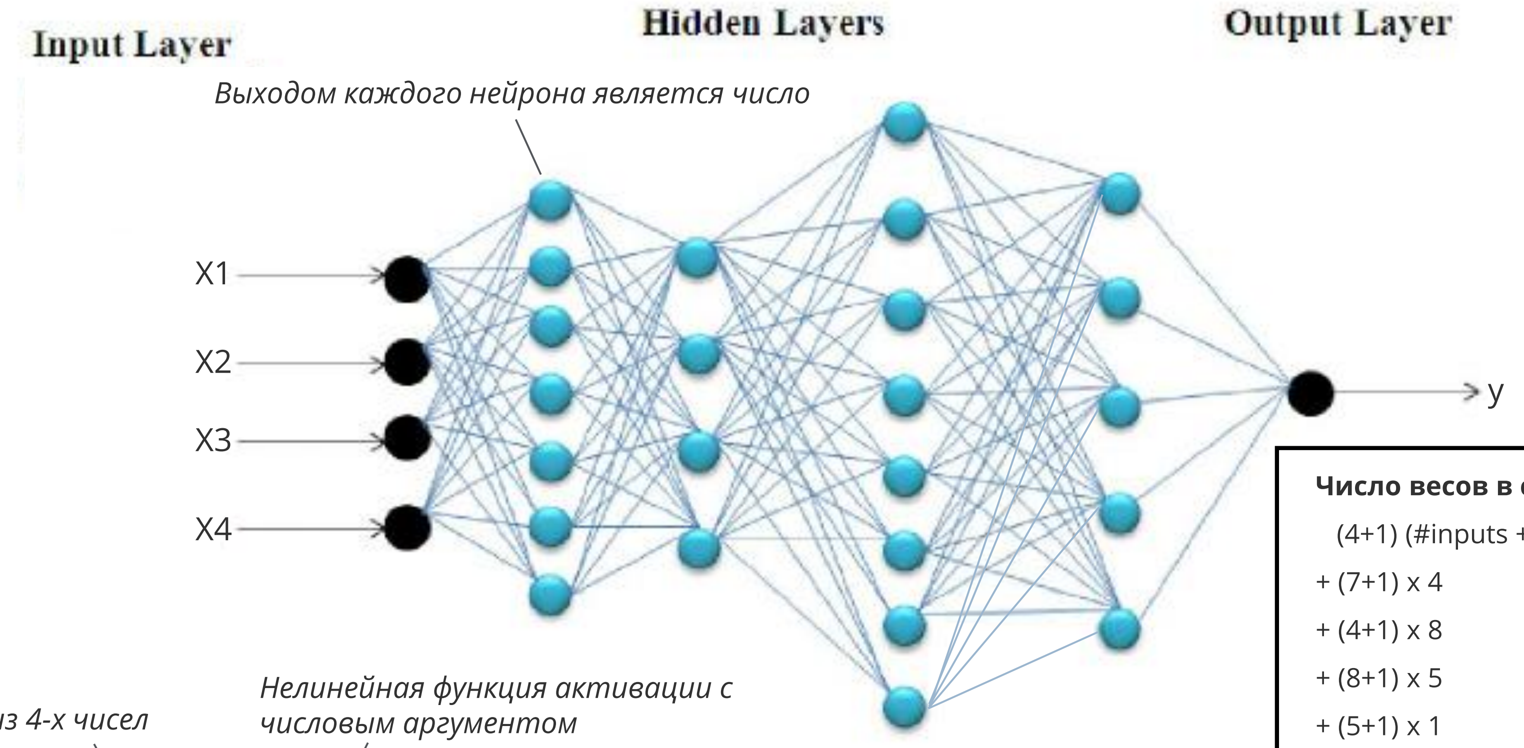


или



Круг интерпретируется, как ***применение ф-ии активации с сумме входных значений помноженных на соответствующие коэф.***

Многослойный перцептрон – нейронная сеть, в которой нейроны каждого слоя связаны только с нейронами соседних слоев



Число весов в сети:

(4+1) (#inputs + bias) x 7(#neurons in layer 1)

\+ (7+1) x 4

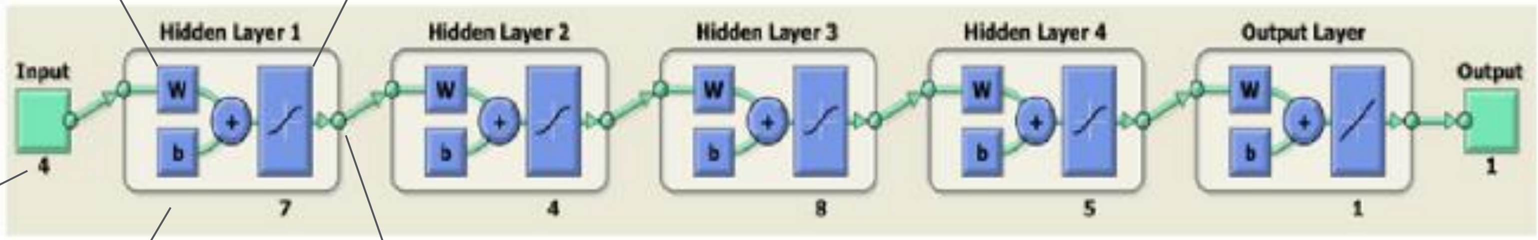
\+ (4+1) x 8

\+ (8+1) x 5

\+ (5+1) x 1

Вектор весов из 4-х чисел

Нелинейная функция активации с числовым аргументом

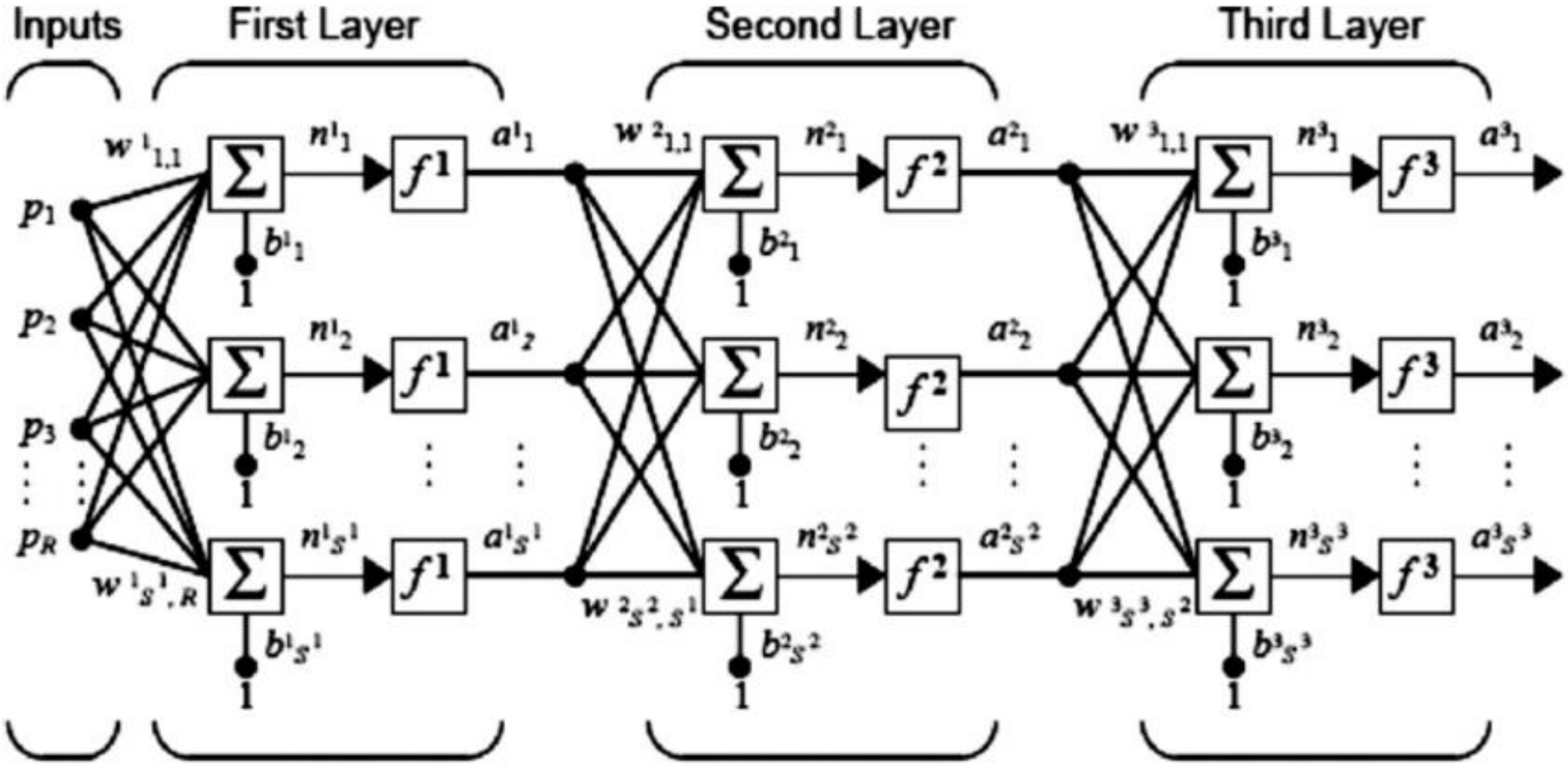


Вектор из 4-х чисел

Число

Вектор из 7-ми чисел – нейронов

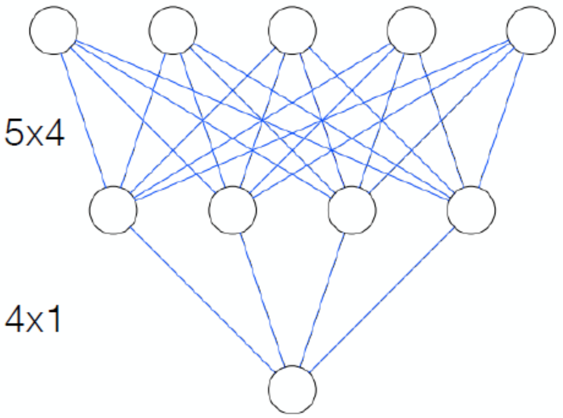
Любая нейронная сеть – это композиция функций



$$a^1 = f^1(w^1 p + b^1) \qquad a^2 = f^2(w^2 p + b^2) \qquad a^3 = f^3(w^3 p + b^3)$$

$$a^3 = f^3(w^3 f^2(w^2 f^1(w^1 p + b^1) + b^2) + b^3)$$

Именно нелинейная функция активации позволяет нейронной сети аппроксимировать нелинейные зависимости



Linear model: $\hat{y} = XW$

Perceptron:

Hidden: $h = XW_1$

Output: $\hat{y} = hW_2$

But: $\hat{y} = hW_2 = XW_1W_2 = XA$

Linear again! damn.

Idea

Add some non-linear activation in the hidden layer

$$h = f(XW_1)$$

$$\hat{y} = hW_2 = f(XW_1)W_2 \neq XA$$

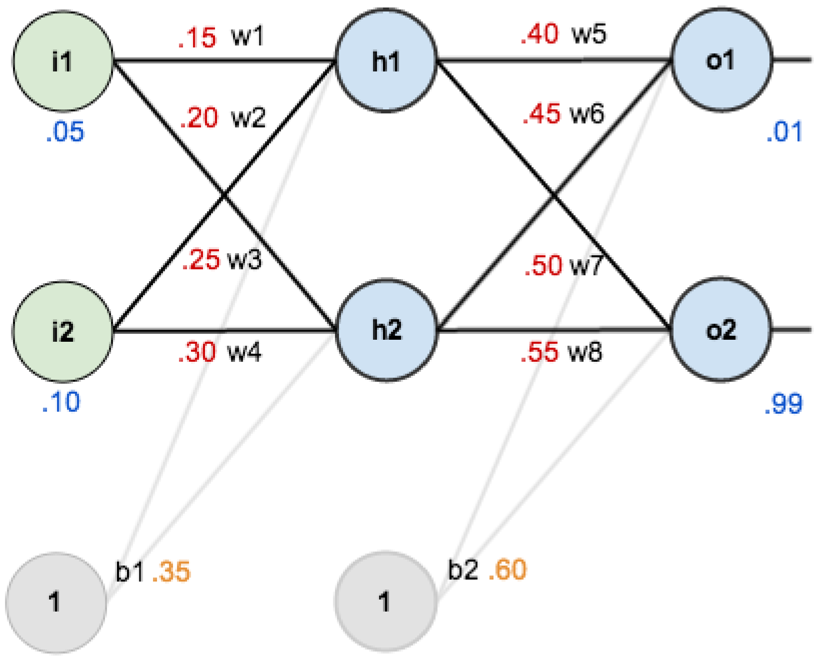
Источник: лекции Ильи Езепова (https://github.com/iezepov)

Forward pass (прямой проход) на численном примере

Задано:

- сеть (архитектура + веса)
- входы X и выходы y

training inputs/outputs initial weights biases



1. Расчет выходов 1-го скрытого слоя

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

2. Расчет выходов выходного слоя

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

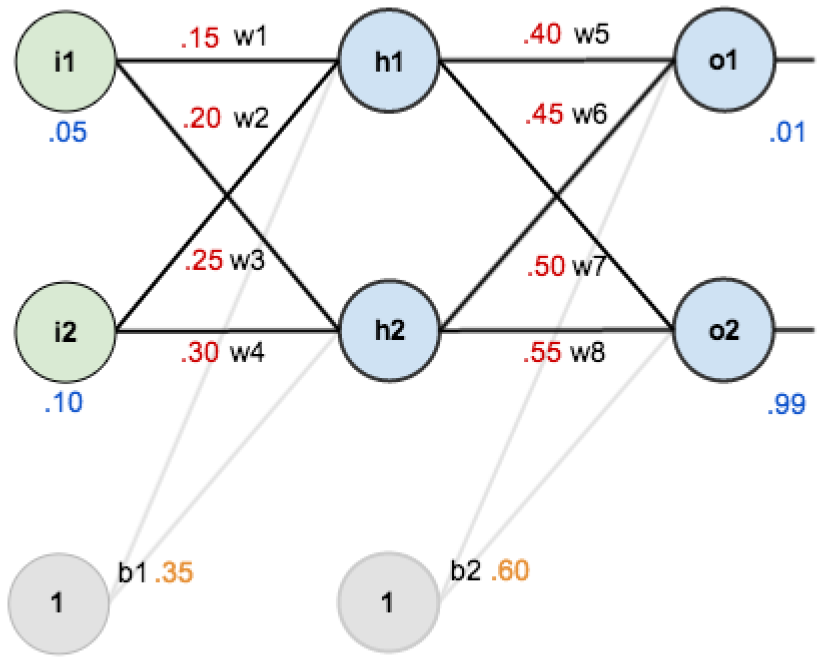
Источник: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Обучение сети на численном примере, backward pass (1/5)

Задано:

- сеть (архитектура + веса)
- входы X и выходы y

training inputs/outputs initial weights biases



3. Подсчет ошибки

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

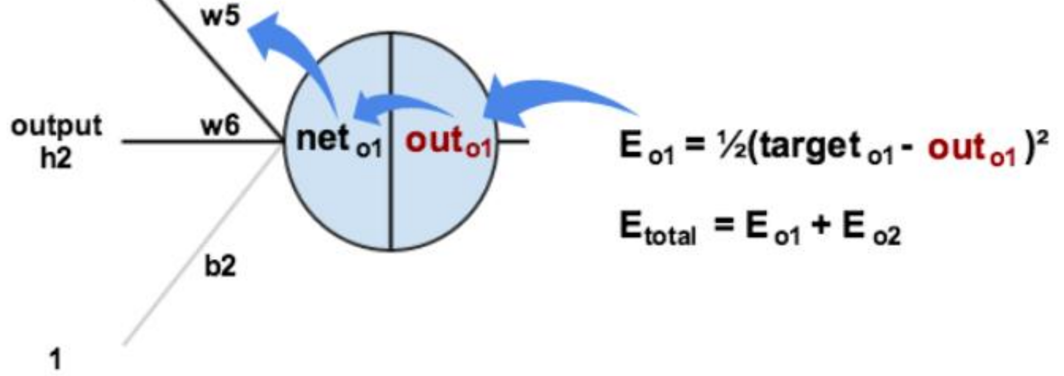
$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

4. Обратное распространение ошибки (или как влияет w_i на итоговую ошибку?) $\frac{\partial E_{total}}{\partial w_5}$

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \boxed{\frac{\partial E_{total}}{\partial w_5}}$$



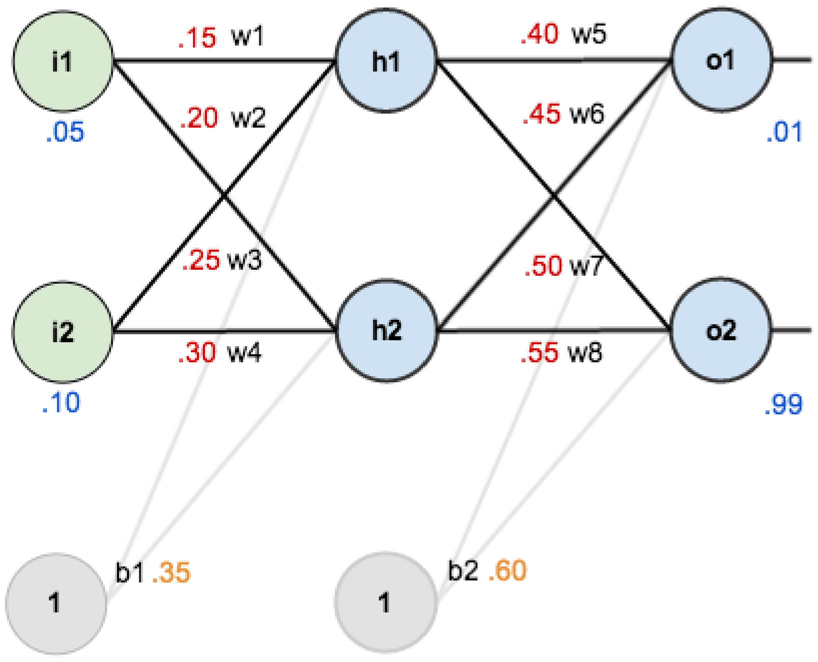
Источник: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Обучение сети на численном примере, backward pass (2/5)

Задано:

- сеть (архитектура + веса)
- входы X и выходы y

training inputs/outputs initial weights biases



4. Обратное распространение ошибки (или как влияет w_i на итоговую ошибку?)

$$\frac{\partial E_{total}}{\partial w_5} = \boxed{\frac{\partial E_{total}}{\partial out_{o1}}} * \boxed{\frac{\partial out_{o1}}{\partial net_{o1}}} * \boxed{\frac{\partial net_{o1}}{\partial w_5}}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\boxed{\frac{\partial E_{total}}{\partial out_{o1}}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

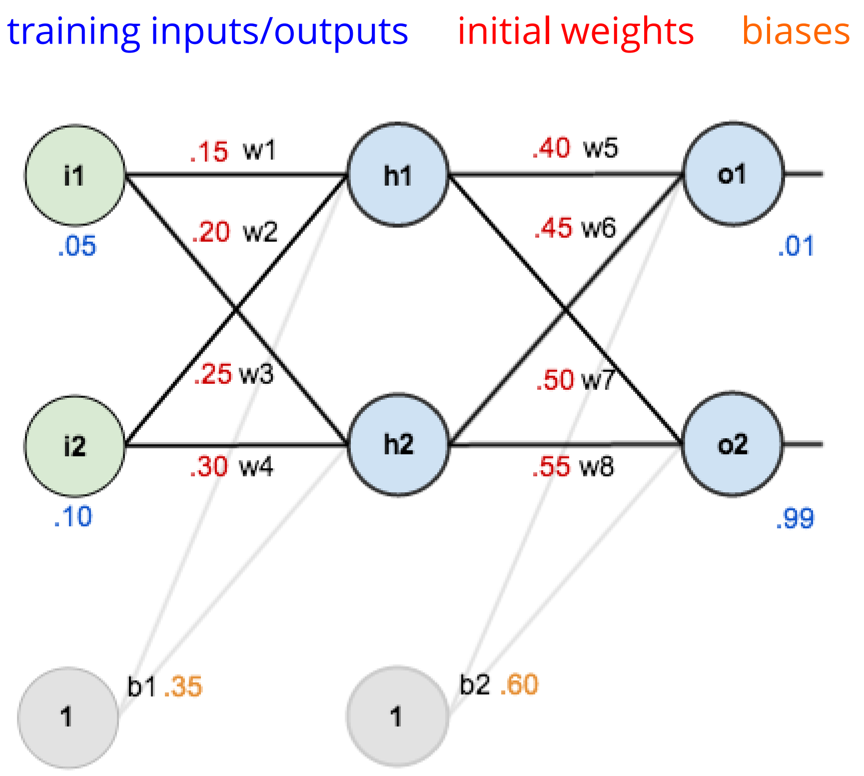
$$\boxed{\frac{\partial out_{o1}}{\partial net_{o1}}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Источник: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Обучение сети на численном примере, backward pass (3/5)

Задано:

- сеть (архитектура + веса)
- входы X и выходы y



4. Обратное распространение ошибки (или как влияет w_i на итоговую ошибку?)

$$\frac{\partial E_{total}}{\partial w_5} = \boxed{\frac{\partial E_{total}}{\partial out_{o1}}} * \boxed{\frac{\partial out_{o1}}{\partial net_{o1}}} * \boxed{\frac{\partial net_{o1}}{\partial w_5}}$$

$$\boxed{\frac{\partial E_{total}}{\partial out_{o1}}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$\boxed{\frac{\partial out_{o1}}{\partial net_{o1}}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\boxed{\frac{\partial net_{o1}}{\partial w_5}} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

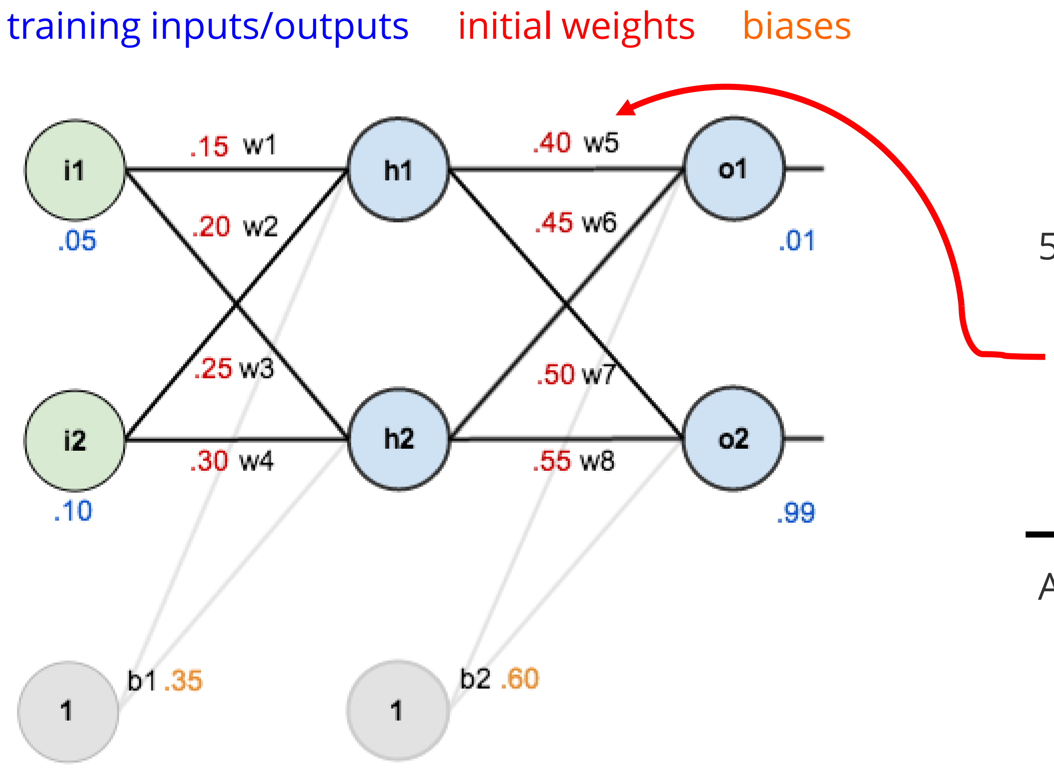
$$\frac{\partial E_{total}}{\partial w_5} = \boxed{0.74136507} * \boxed{0.186815602} * \boxed{0.593269992} = 0.082167041$$

Источник: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Обучение сети на численном примере, backward pass (4/5)

Задано:

- сеть (архитектура + веса)
- входы X и выходы y



4. Обратное распространение ошибки (или как влияет w_i на итоговую ошибку?)

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

5. Обновление весов

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

learning rate

Аналогично для ω_1

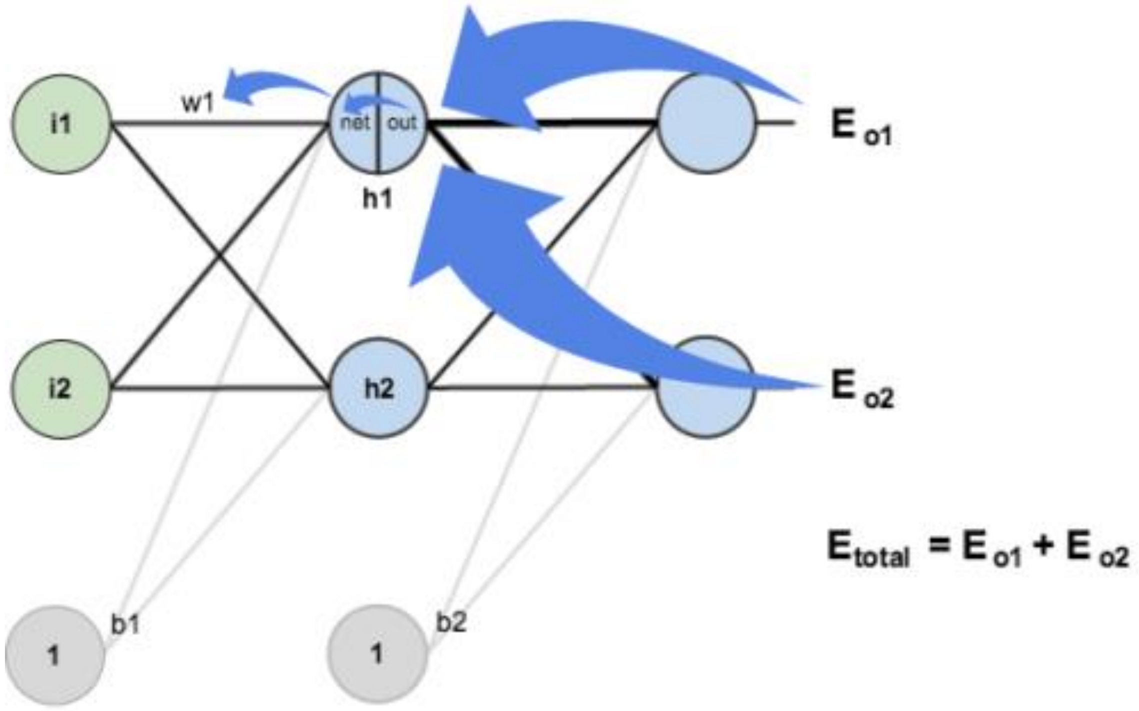
Источник: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Обучение сети на численном примере, backward pass (5/5)

Аналогично для ω_1

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1})$$

уже считали ранее

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}}$$

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Источник: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Перцептрон – универсальный аппроксиматор (может аппроксимировать любую функцию)

The Perceptron Convergence Theorem

Frank Rosenblatt, 1965

- Perceptron can solve **any** problem
- Perceptron will converge in finite time
- Result is independent of initial weights

Great. Why to implement anything else?

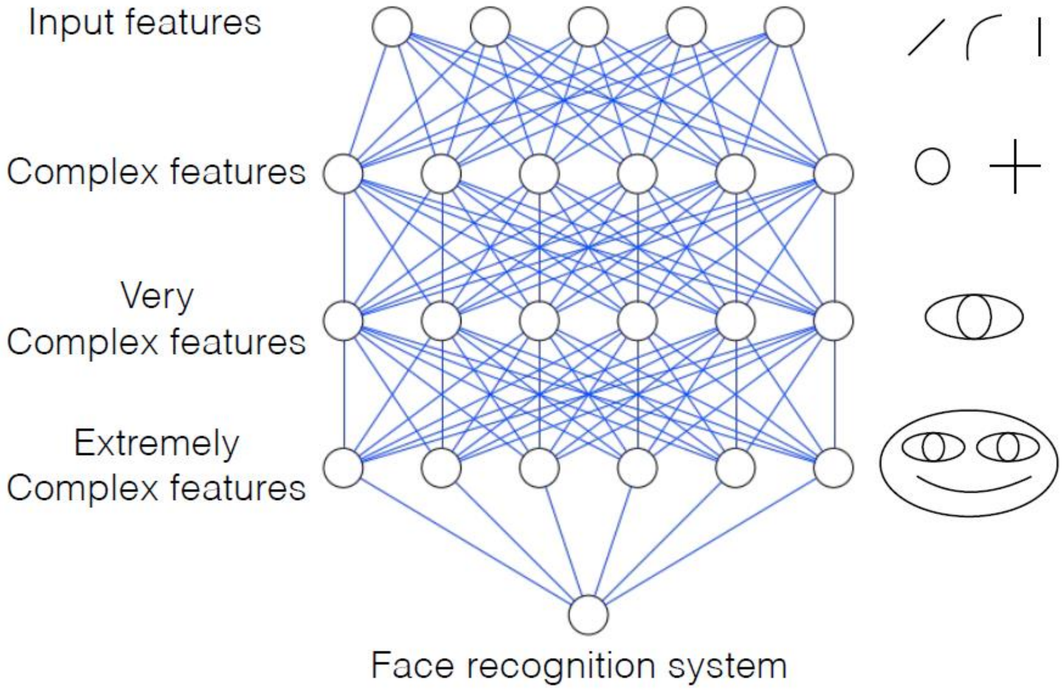
Источник: лекции Ильи Езепова (https://github.com/iezepov)

Почему сети делают глубокими, а не широкими?

Motivation: computational cost

Wide enough perceptron can solve any problem, but the cost will be very high

It's cheaper to go deeper than wider



Источник: лекции Ильи Езепова (https://github.com/iezepov)

Техники избавления от переобучения в нейронных сетях

Thousands of parameters may (and will) cause overfitting

- Data augmentation if possible *(NNs need more data)*
- L2 penalty (add sum of weights squares to loss)
- Dropout

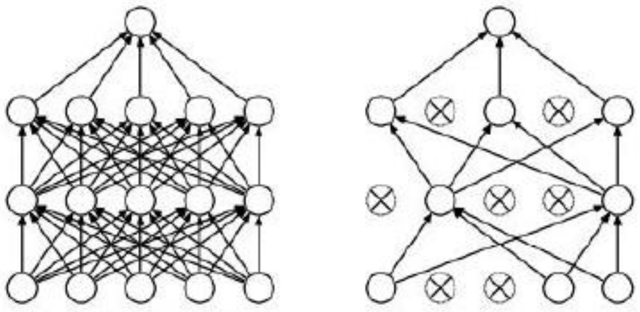
Dropout

Introduced in 2014

In terms of layers as functions:

$$f_{train}(x) = \begin{cases} 0, & \text{with probability } p \\ x, & \text{with probability } 1-p \end{cases}$$

$$f_{test}(x) = (1-p)x$$



(a) Standard Neural Net (b) After applying dropout.

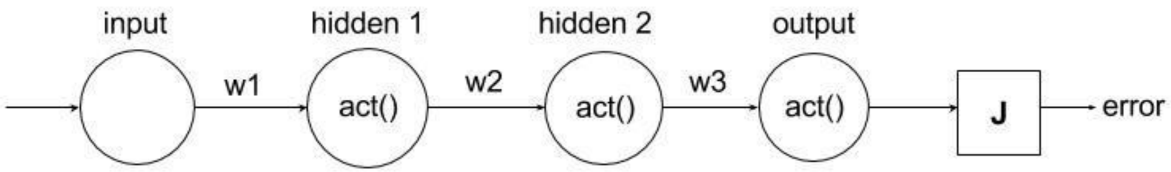
Data augmentation

Cat

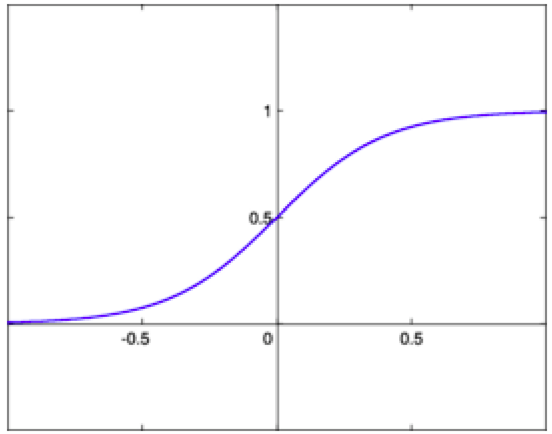
Still a cat — we can learn on them!

Источник: лекции Ильи Езепова (https://github.com/iezepov)

Проблема затухания градиентов



$$Sigmoid = S(\alpha) = \frac{1}{1+e^{-\alpha}}$$



$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

$$output = Sigmoid(hidden2 \ * w3)$$

$$z_1 = hidden2 * w3$$

$$\frac{\partial output}{\partial hidden2} = \frac{\partial Sigmoid(z_1)}{\partial z_1} w3$$

$$hidden2 = Sigmoid(hidden1 \ * w2)$$

$$z_2 = hidden1 * w2$$

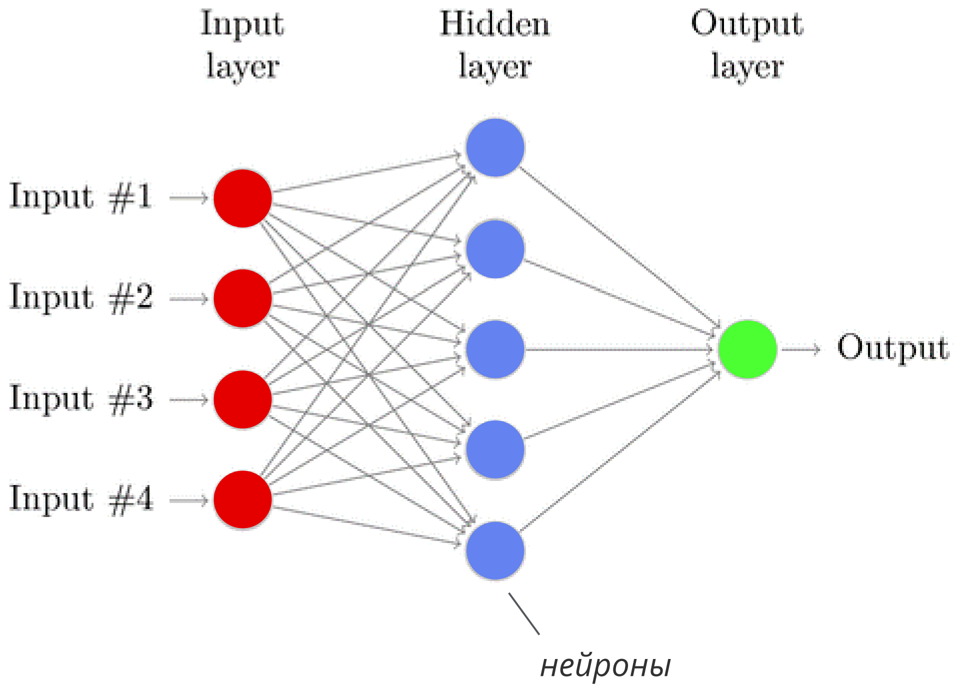
$$\frac{\partial hidden2}{\partial hidden1} = \frac{\partial Sigmoid(z_2)}{\partial z_2} w2$$

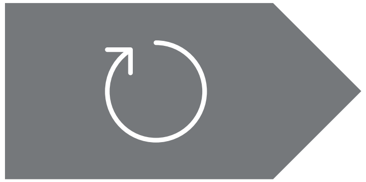
$$\left(\frac{1}{1+e^{-\alpha}}\right)' = \frac{1}{1+e^{-\alpha}}\left[1 - \frac{1}{1+e^{-\alpha}}\right]$$

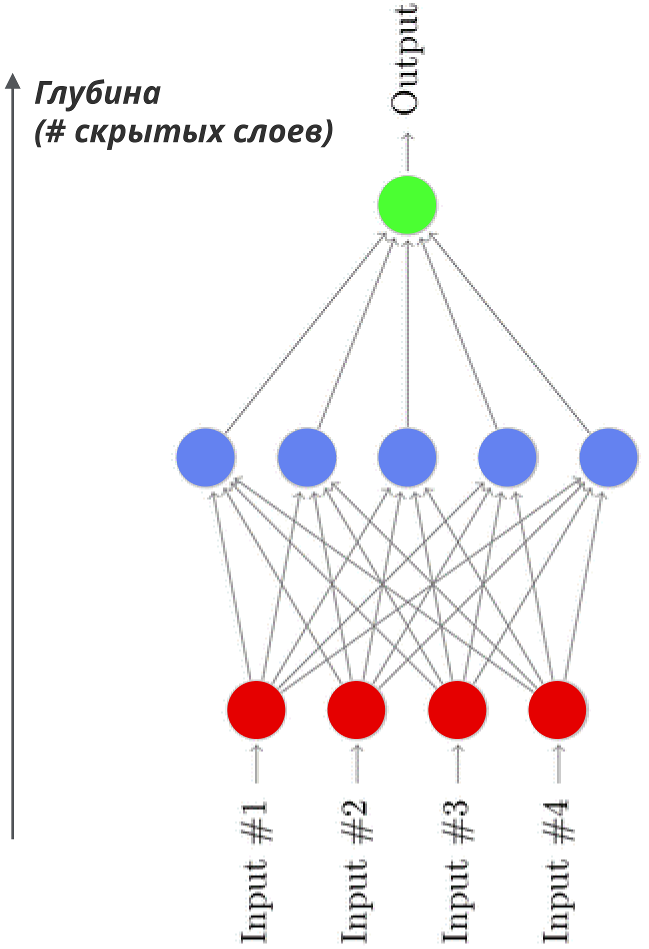
$$\frac{\partial output}{\partial hidden2}\frac{\partial hidden2}{\partial hidden1} = \overbrace{\frac{\partial Sigmoid(z_1)}{\partial z_1}}^{<1/4} \overbrace{w3}^{<1} * \overbrace{\frac{\partial Sigmoid(z_2)}{\partial z_2}}^{<1/4} \overbrace{w2}^{<1} \longrightarrow 0$$

Источник: https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b

Перцептрон имеет одну абстрактную размерность – глубину (число слоев)



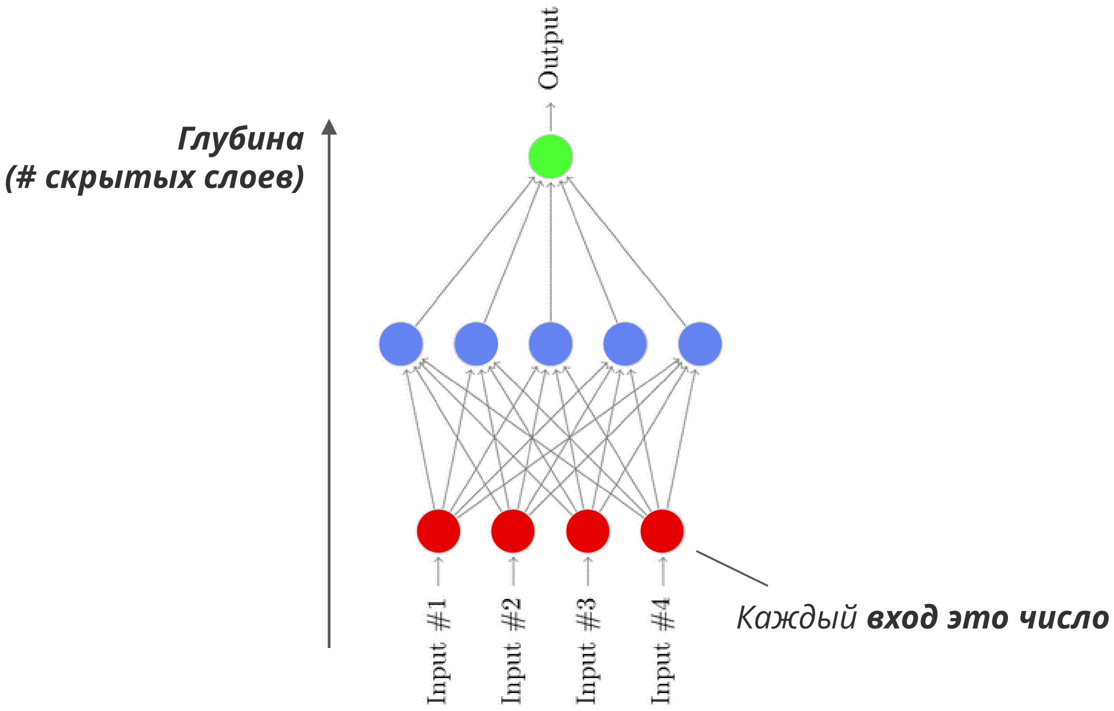




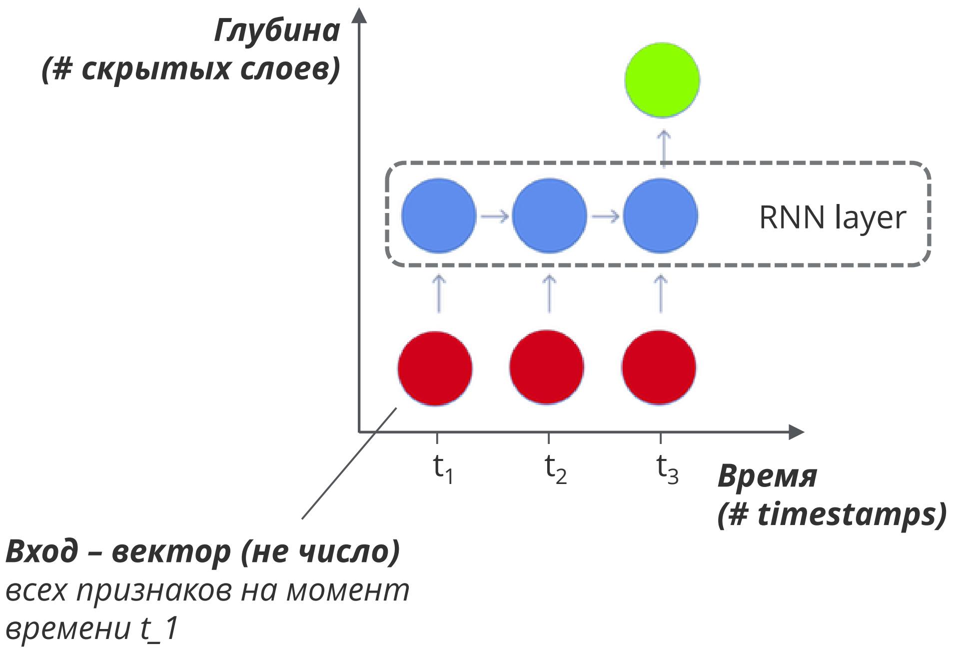
$$\mathbf{Neuron_i}(input_1, input_2, input_3, input_4) =$$
$$= \tanh\left(\mathbf{w_0^i} + input_1 \cdot \mathbf{w_1^i} + input_2 \cdot \mathbf{w_2^i} + input_3 \cdot \mathbf{w_3^i} + input_4 \cdot \mathbf{w_4^i}\right)$$

Рекуррентная нейронная сеть имеет два измерения – глубина сети и время

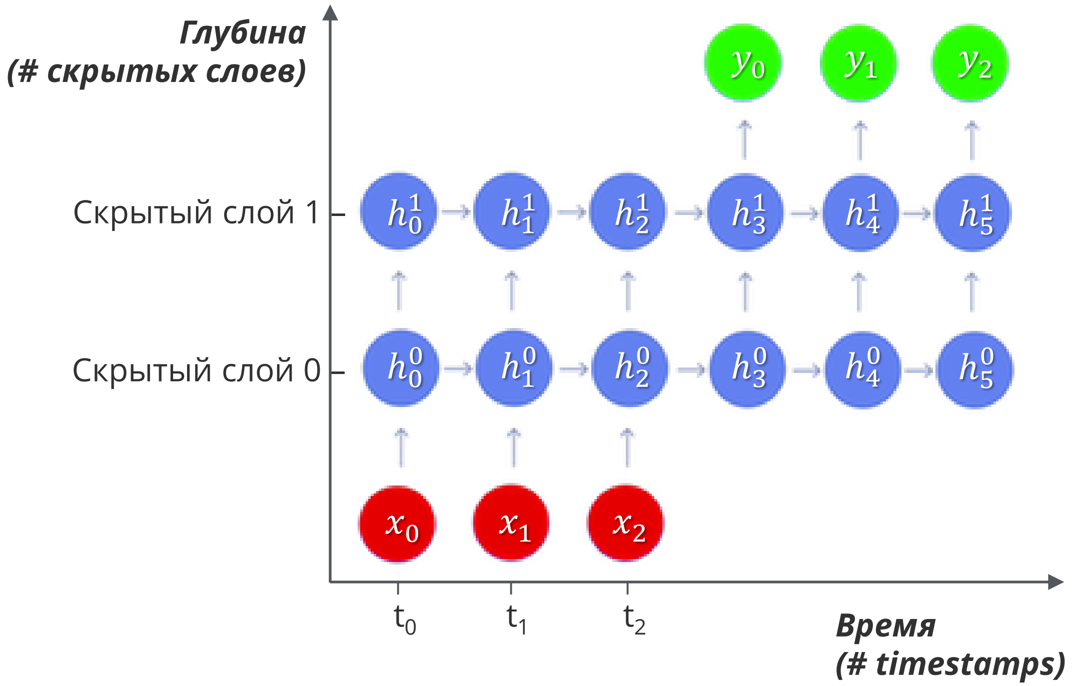
Перцептрон (DNN)



Recurrent Neural Network (RNN)



Что из себя представляет RNN?



Input $\rightarrow x_t$

Output $\rightarrow y_t$

Hidden $\rightarrow h^{\ell}_t$

input to hidden $\rightarrow W_{xh}$

hidden to hidden in time $\rightarrow W^{\ell}_{hht}$

hidden to hidden in depth $\rightarrow W^{\ell}_{hhd}$

hidden to output $\rightarrow W_{hy}$

biases $\rightarrow b^{\ell}_h,\ b^{\ell}_y$

Расчет скрытых слоев RNN

$$h^{\ell}_t = \begin{cases} f_W(h^{\ell}_{t-1},\ x_t) \text{ for } \ell = 1 \\ f_W(h^{\ell}_{t-1},\ h^{\ell-1}_t) \text{ for } \ell > 1 \end{cases}$$

Расчет выходов RNN

$$y_t = W_{hy} h^{\ell}_t + b_y$$

$$h^{\ell}_t = f_W(h^{\ell}_{t-1},\ h^{\ell-1}_t) \text{ for } \ell > 1$$
$$= \tanh(W^{\ell}_{hht} h^{\ell}_{t-1} + W^{\ell}_{hhd} h^{\ell-1}_t + b^{\ell}_h)$$

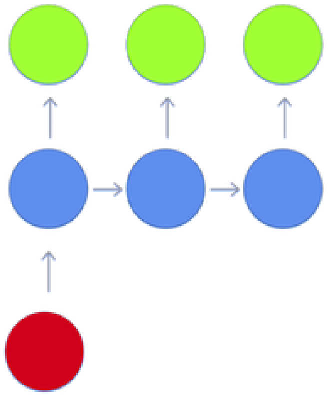
Любое произв. ***W·h****, является вектором (а не скаляром как в DNN)*

$$h^{\ell}_t = f_W(h^{\ell}_{t-1},\ x_t) \text{ for } \ell = 1$$
$$= \tanh(W^{\ell}_{hht} h^{\ell}_{t-1} + W_{xh} x_t + b^{\ell}_h)$$

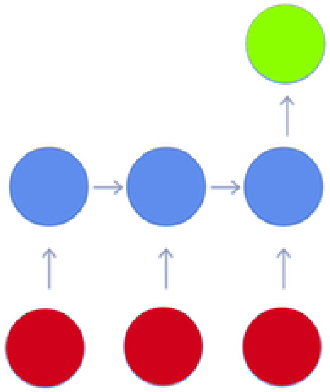
$h^{\ell}_0 = 0$ for any layer ℓ

Различные постановки задач требуют разные архитектуры RNN

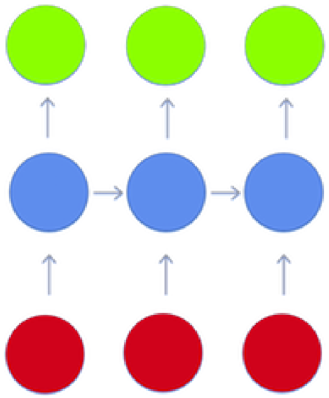
One-to-many



Many-to-one

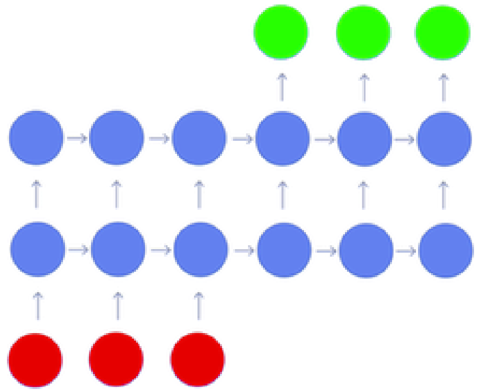


Many-to-many “synchronized”



General structure

Many-to-many “non-synchronized”



Example

One-to-many

Training set:

Sample 1:
- — Input (X): [1]
- — Output (y): [2, 3, 4]

Sample 2:
- — Input (X): [2]
- — Output (y): [3, 4, 5]

Many-to-one

Training set:

Sample 1:
- — Input (X): [1, 2, 3]
- — Output (y): [4]

Sample 2:
- — Input (X): [2, 3, 4]
- — Output (y): [5]

Many-to-many “synchronized”

Training set:

Sample 1:
- — Input (X): [1, 2, 3]
- — Output (y): [4, 5, 6]

Sample 2:
- — Input (X): [2, 3, 4]
- — Output (y): [5, 6, 7]

Many-to-many “non-synchronized”

Training set:

Sample 1:
- — Input (X): [1, 2, 3]
- — Output (y): [4, 5, 6]

Sample 2:
- — Input (X): [2, 3, 4]
- — Output (y): [5, 6, 7]

Бэкпроп для RNN (Backpropagation Through Time)

RNN:
$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(Vs_t)$$

Три матрицы весов, которые надо найти

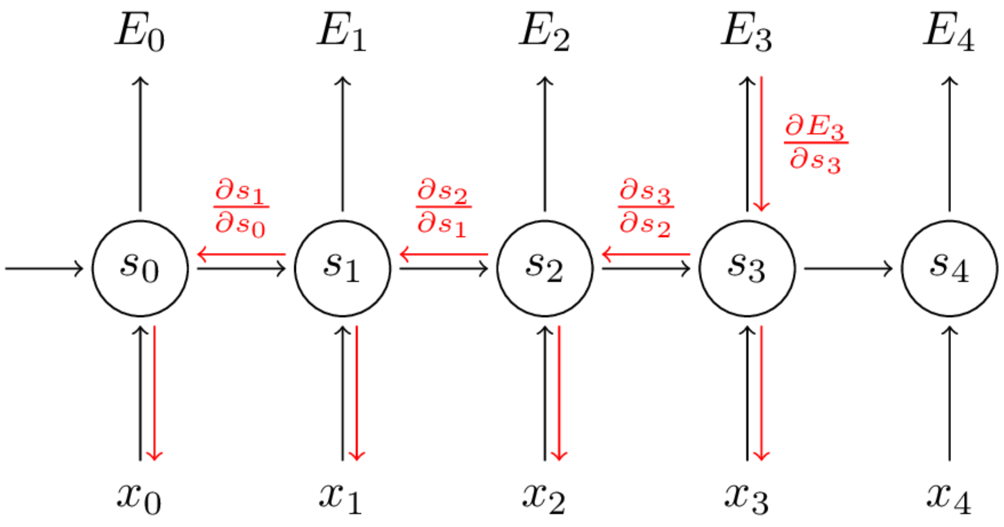
Loss:
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$
$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$

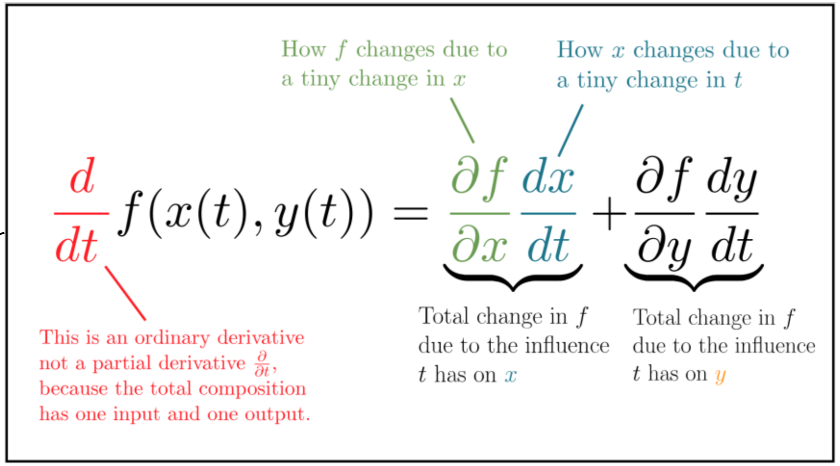
Расчет градиента по *W*: $$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

$$S_t = S_t(W, S_{t-1}(W), \dots)$$

Поэтому применяем правило дифференцирования сложной функции от двух аргументов

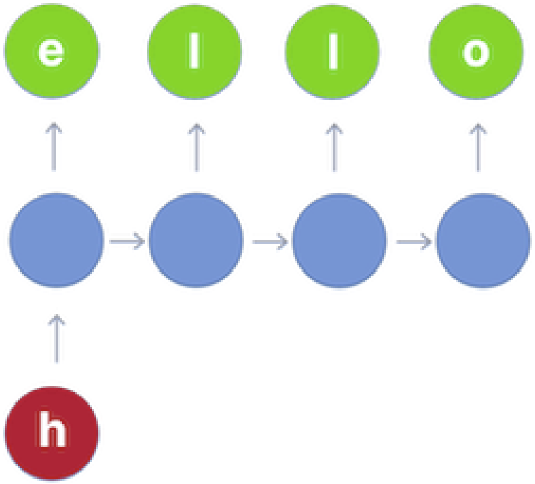




Расчет градиентов по V и U делается аналогично

Пример генерации последовательности

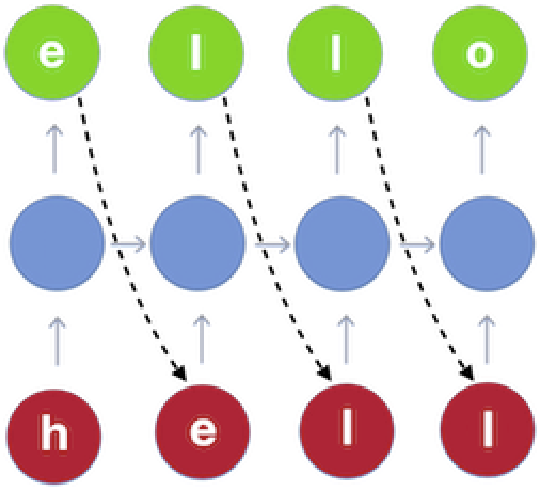
Хотим по первым буквам сгенерировать оставшееся слово



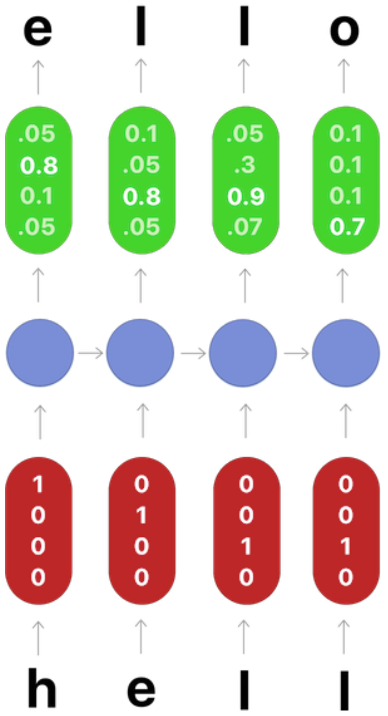
Будем делать это рекуррентно:

1. Подставим **первую букву** в сеть (RNN ячейку)
2. Из выхода сети получим **вектор внутреннего состояния** и **вторую букву**
3. Теперь подставим **вторую букву** и **вектор скрытого состояния**

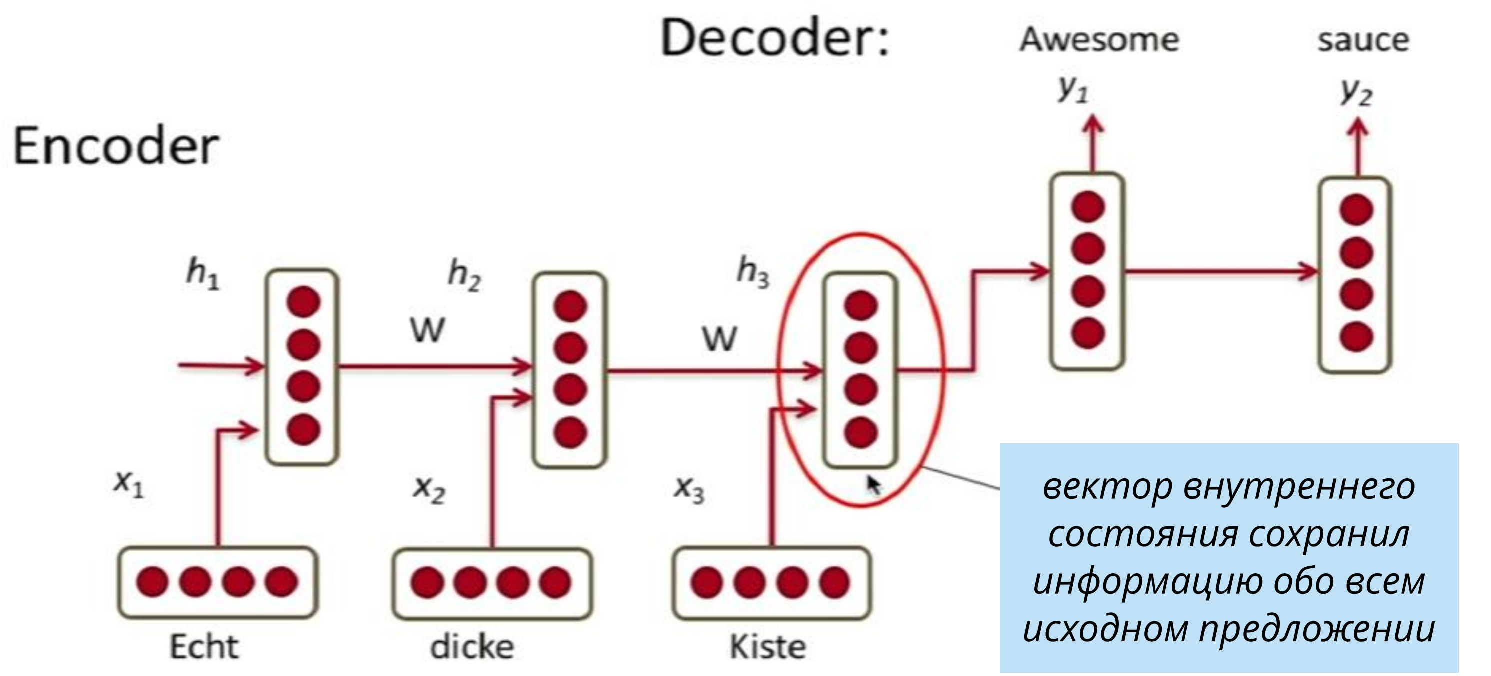
…



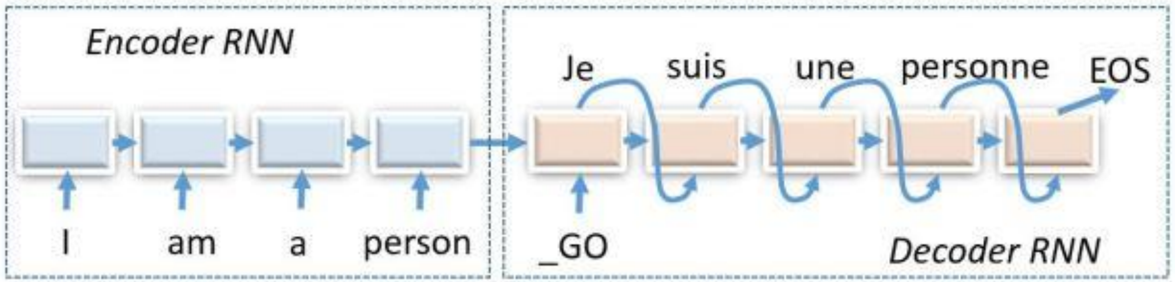
$$"h" = \begin{bmatrix}1\\0\\0\\0\end{bmatrix};\ "e" = \begin{bmatrix}0\\1\\0\\0\end{bmatrix};\ "l" = \begin{bmatrix}0\\0\\1\\0\end{bmatrix};\ "o" = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$



Перевод с помощью Encoder-Decoder RNN



вектор внутреннего состояния сохранил информацию обо всем исходном предложении



Source: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/recurrent_neural_networks/machine-translation-using-rnn.html

Рекуррентные нейронные сети в Keras

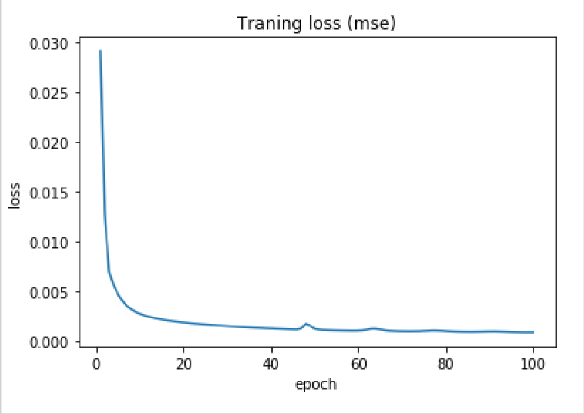
```
model = Sequential()
# input_shape = (time_steps, input_dim) - where input_dim is the number of features X
model.add(SimpleRNN(units=100, input_shape=(None, data.shape[1]), return_sequences=True, activation='tanh'))
model.add(TimeDistributed(Dense(data.shape[1])))
model.add(Activation('linear'))
model.compile(loss='mse', optimizer='rmsprop')
```

```
history = model.fit(X, y, epochs=100, verbose=0)
```

```
input_seq = X[0][:2]
output_seq = model.predict(np.array([input_seq]))[0]
```

```
Input sample:
[[0.006]
 [0.014]]

Output:
[[0.00632539]
 [0.00203975]]
```



Jupyter notebook: https://github.com/uselessskills/tutorials/blob/master/rnn/simple_rnn_forecast.ipynb

Источники и полезные ссылки

- Лекции Ильи Езепова: https://github.com/iezepov
- Хорошее введение в RNN с нуля: https://ayearofai.com/rohan-lenny-3-recurrent-neural-networks-10300100899b
- RNN для ряда с трендом: https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html
- Числовой пример backprop на LSTM: https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9
- Видео-лекция про RNN от Brandon Rohrer: https://www.youtube.com/watch?v=WCUNPb-5EYI
- Интерактивная статья про расширения для RNN: https://distill.pub/2016/augmented-rnns/
- Вывод backprop для RNN: http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/, http://songhuiming.github.io/pages/2017/08/20/build-recurrent-neural-network-from-scratch/
- Что такое Attention: https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129
- Еще немного про attention: http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/

Deloitte.

deloitte.ru