Sieci rekurencyjne (RNNs)

Ministerstwo Cyfryzacji

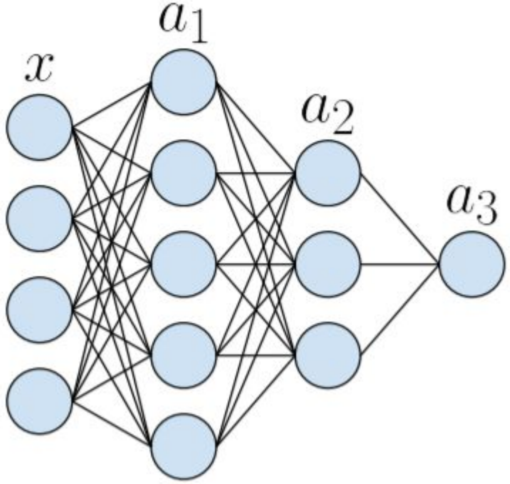
Michał Stypułkowski
February 2025

Wrocław
miasto spotkań

Wprowadzenie

Liniowe sieci neuronowe

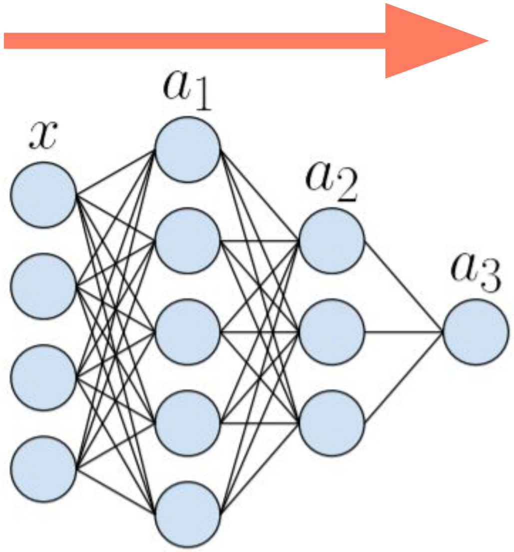
- Składają się z warstw i neuronów



Liniowe sieci neuronowe

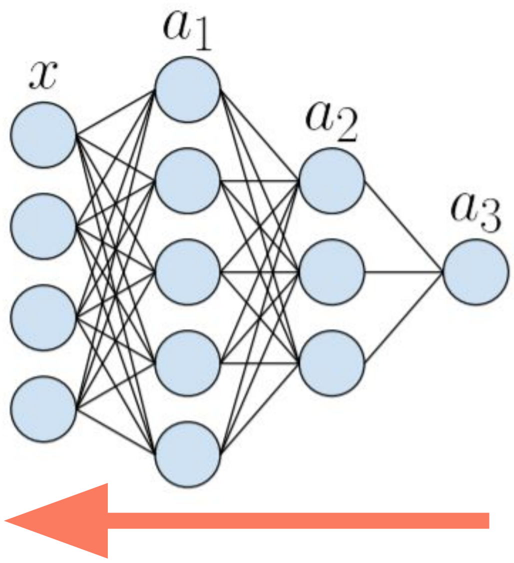
- Składają się z warstw i neuronów

- Forward propagation



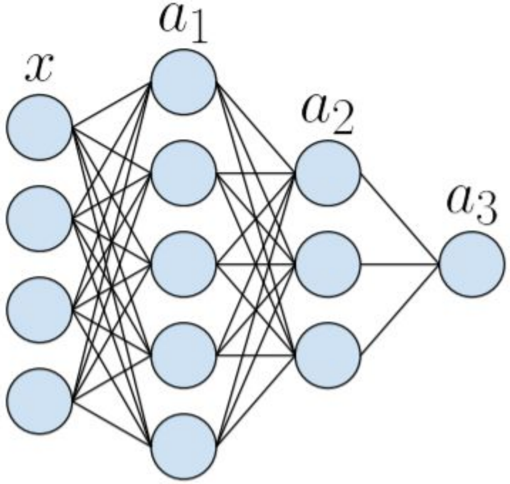
Liniowe sieci neuronowe

- Składają się z warstw i neuronów
- Forward propagation
- Backward propagation



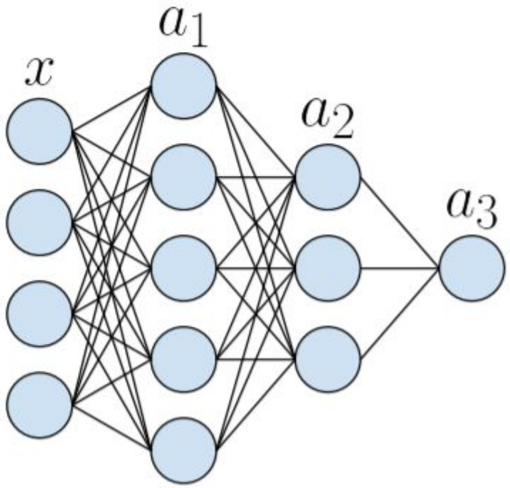
Liniowe sieci neuronowe

- Składają się z warstw i neuronów
- Forward propagation
- Backward propagation
- Ustalony rozmiar wejścia



Liniowe sieci neuronowe

- Składają się z warstw i neuronów
- Forward propagation
- Backward propagation
- Ustalony rozmiar wejścia
- Dobre dla obrazków, danych tabelarycznych, itd.



Co z danymi sekwencyjnymi?

Dane sekwencyjne

- Tekst, audio, szeregi czasowe, …

Dane sekwencyjne

- Tekst, audio, szeregi czasowe, …

- Kolejność ma znaczenie!

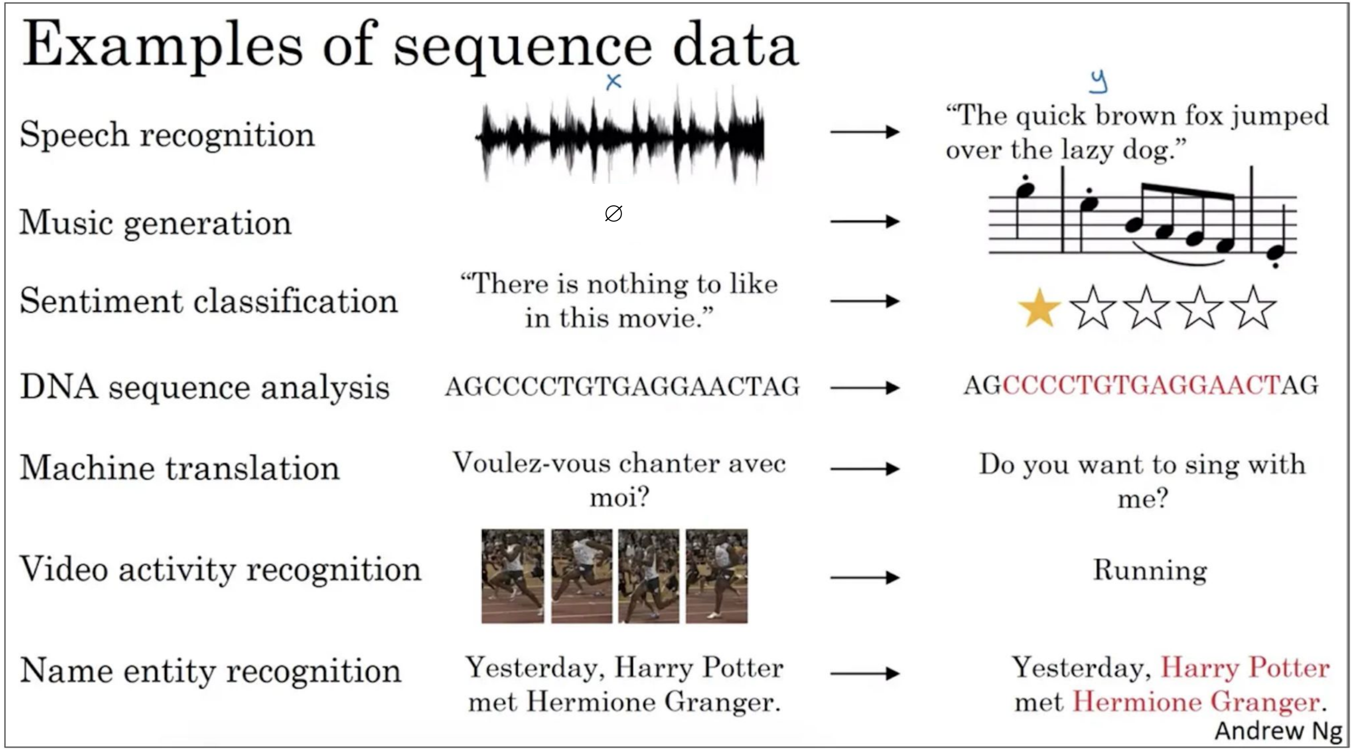
Dane sekwencyjne

- Tekst, audio, szeregi czasowe, …
- Kolejność ma znaczenie!
- Zmienna długość sekwencji

Dane sekwencyjne

- Tekst, audio, szeregi czasowe, …
- Kolejność ma znaczenie!
- Zmienna długość sekwencji
- Kontekst i pamięć są często potrzebne

Dane sekwencyjne



Source: [link]

Dlaczego liniowe sieci nie radzą sobie z danymi sekwencyjnymi?

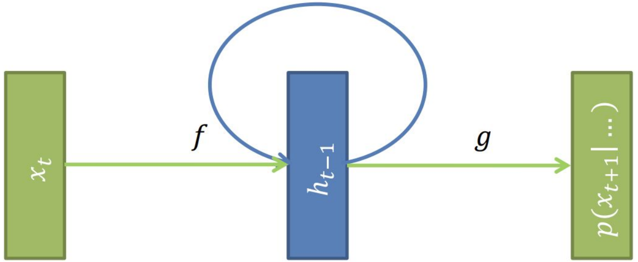
Dane sekwencyjne

- Tekst, audio, szeregi czasowe, …
- Kolejność ma znaczenie!
- Zmienna długość sekwencji
- Kontekst i pamięć są często potrzebne

Sieci rekurencyjne
(RNNs)

RNNs

RNNs are networks with state



$$h_t = f(h_{t-1}, x_t)$$
$$p(x_{t+1}|x_1, \dots, x_t) = g(h_t)$$

f, g are implemented as feedforward neural nets.

Source: [link]

RNNs

RNN Example

Input: a sequence of bits $1,0,1,0,0,1$

Output: the parity

Solution:

The hidden state will be just 1 bit – parity so far:

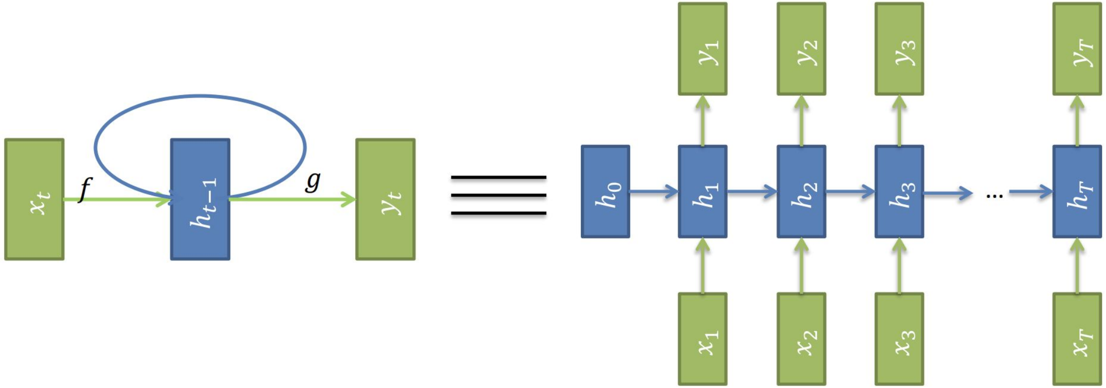
$$h_0 = 0$$
$$h_t = XOR(h_{t-1}, x_t)$$
$$y_T = h_T$$

Source: link

RNNs

RNNs are dynamical systems

Time is discrete, we can unroll:



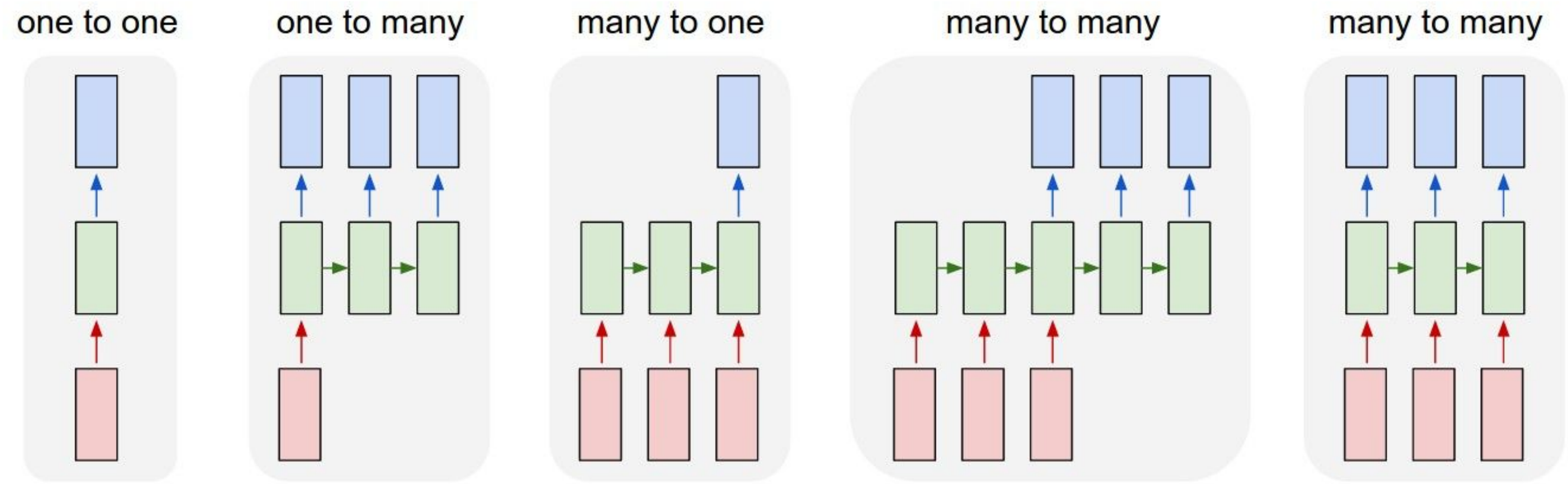
Thus the RNN is a very deep network, with same "arrows" computing the same function!

Source: link

Kod w PyTorchu

```
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleRNN, self).__init__()
        self.hidden_size = hidden_size

        # Learnable parameters
        self.Wxh = nn.Linear(input_size, hidden_size, bias=False)  # Input to hidden
        self.Whh = nn.Linear(hidden_size, hidden_size, bias=False) # Hidden to hidden
        self.Why = nn.Linear(hidden_size, output_size, bias=False) # Hidden to output

    def forward(self, x, hidden):
        seq_length = x.size(1)  # Get the sequence length
        outputs = []
        for t in range(seq_length):
            hidden = torch.tanh(self.Wxh(x[:, t, :]) + self.Whh(hidden))
            output = self.Why(hidden)
            outputs.append(output)

        outputs = torch.stack(outputs, dim=1)  # Stack along sequence dimension
        return outputs, hidden

    def init_hidden(self, batch_size):
        return torch.zeros(batch_size, self.hidden_size)
```

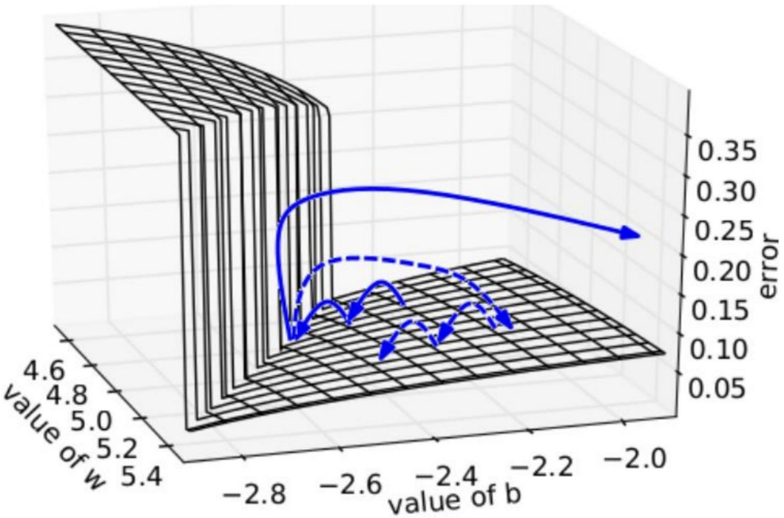
Rodzaje sieci rekurencyjnych



Source: [link]

Eksplodujący gradient

$$h_0 = \sigma(0.5)$$
$$h_t = \sigma(wh_{t-1} + b)$$
$$L = (h_{50} - 0.7)^2$$



Source: [link]

Eksplodujący gradient

Exploding gradient solution

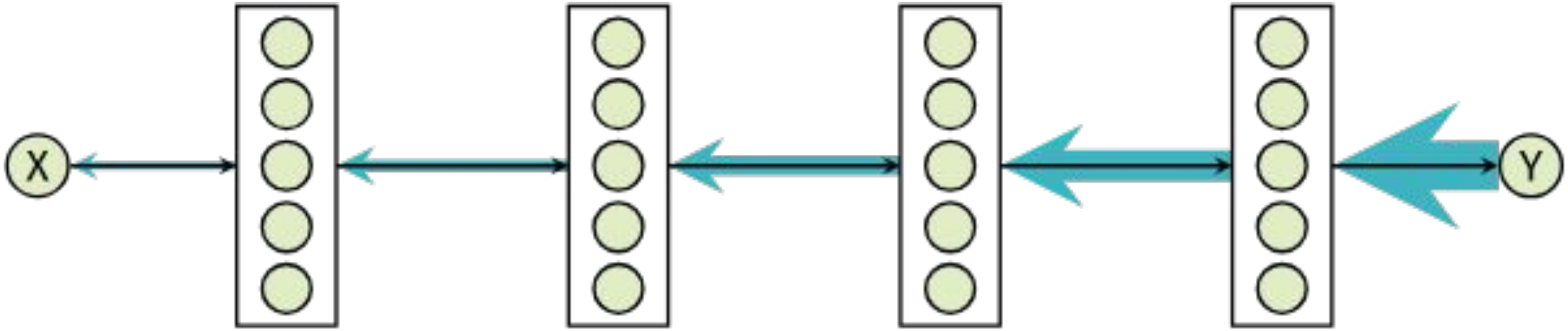
Don’t do large steps.

Pick a gradient norm threshold and scale down all larger gradients.

This prevents the model from doing a large learning update and destroying itself.

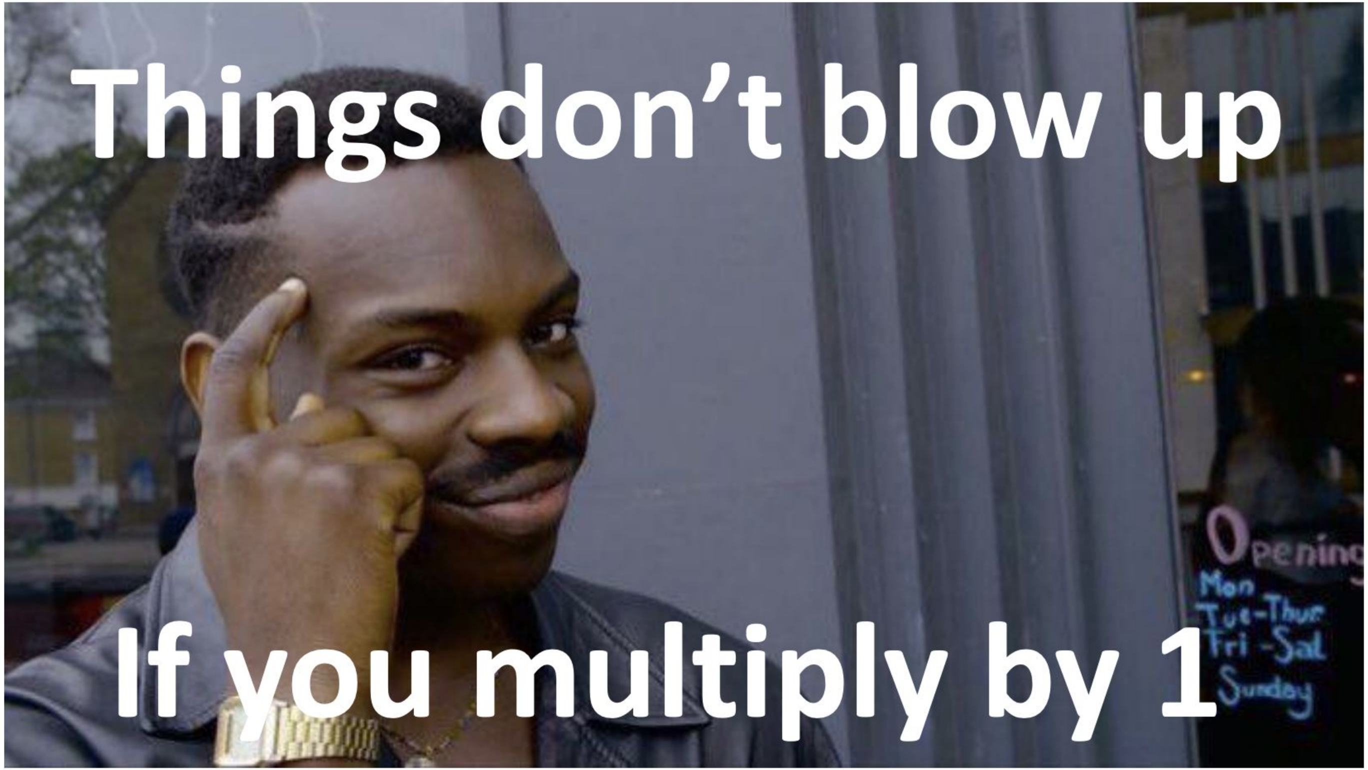
Source: [link]

Zanikający gradient



Source: [link]()

LSTM



Source: [link]

LSTM

LSTM intuitions

Recall the scalar case

$$h_t = wh_{t-1}$$

It maximally preserves information when $w = 1$

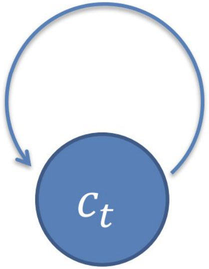
The LSTM introduces a memory cell c_t that will keep information forever:

$$c_t = 1 \cdot c_{t-1}$$

Source: link

LSTM

Memory cell



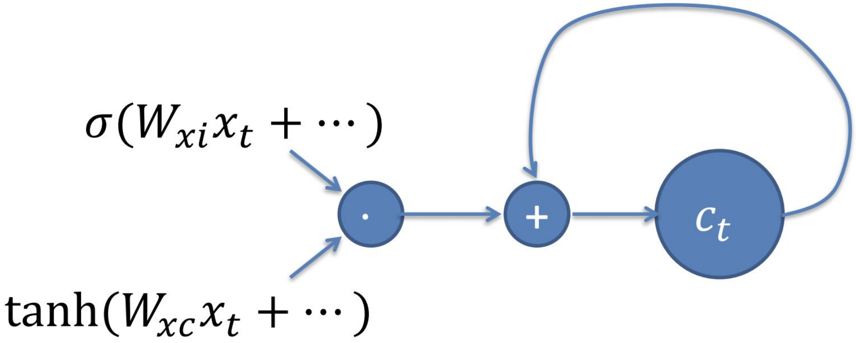
Memory cell preserves information

$$c_t = c_{t-1}$$
$$\frac{\partial c_T}{\partial c_t} = 1$$

Source: [link]

LSTM

Gates



Gates selectively load information into the memory cell:

$$i_t = \sigma(W_{xi}x_t + \cdots)$$

$$c_t = c_{t-1} + i_t \cdot \tanh(W_{xc}x_t + \cdots)$$

Source: [link]

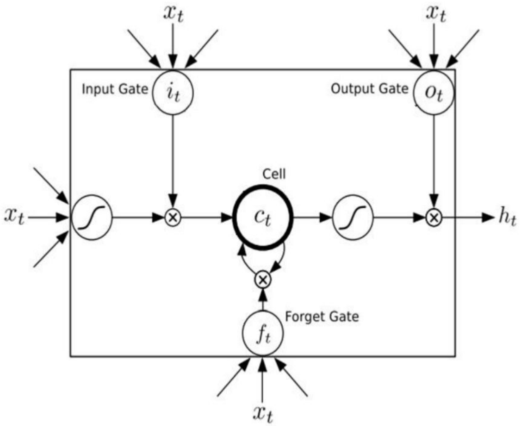
LSTM

LSTM: the details

Hidden state is a pair of:
-c_t information in the cell, hidden from the rest of the network
-h_t information extracted form the cell into the network

Update equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$c_t = i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) + f_t c_{t-1}$$
$$h_t = o_t \tanh c_t$$



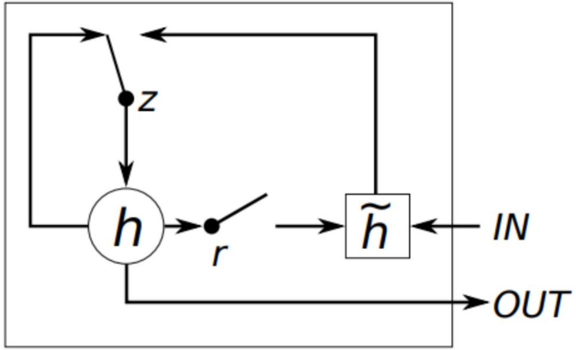
Source: [link]

GRU

The GRU cell – an LSTM alternative

The GRU is similar to the LSTM, but:

- uses only two gates: reset (r) and update (z)
- Doesn't have a separate c_t from h_t.



$$r_t^j = \sigma\left(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1}\right)^j$$

$$\tilde{h}_t^j = \tanh\left(W \mathbf{x}_t + U\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)\right)^j$$

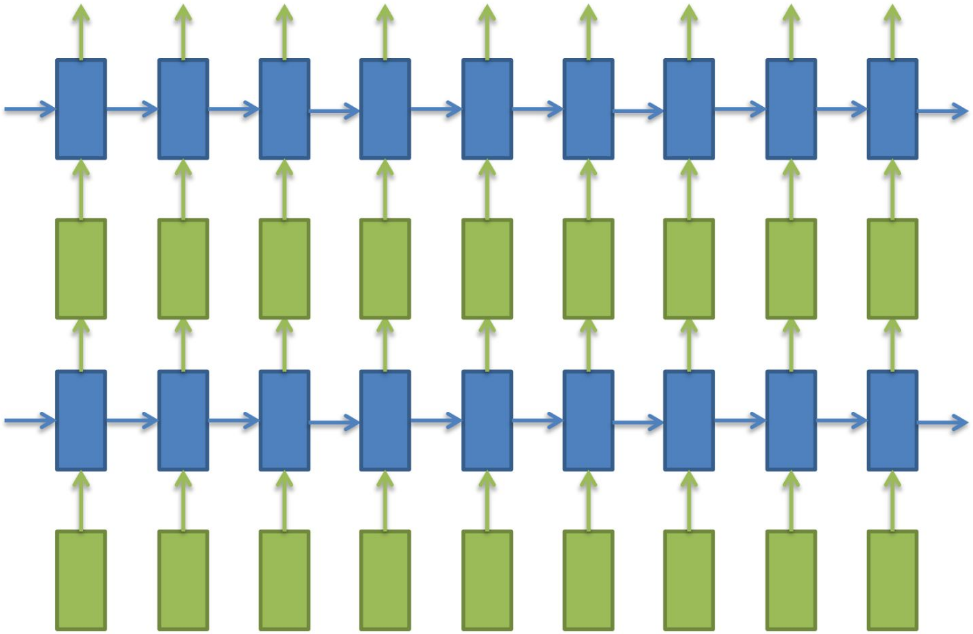
$$z_t^j = \sigma\left(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1}\right)^j$$

$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \tilde{h}_t^j$$

Source: [link]

Wielowarstwowe sieci rekurencyjne

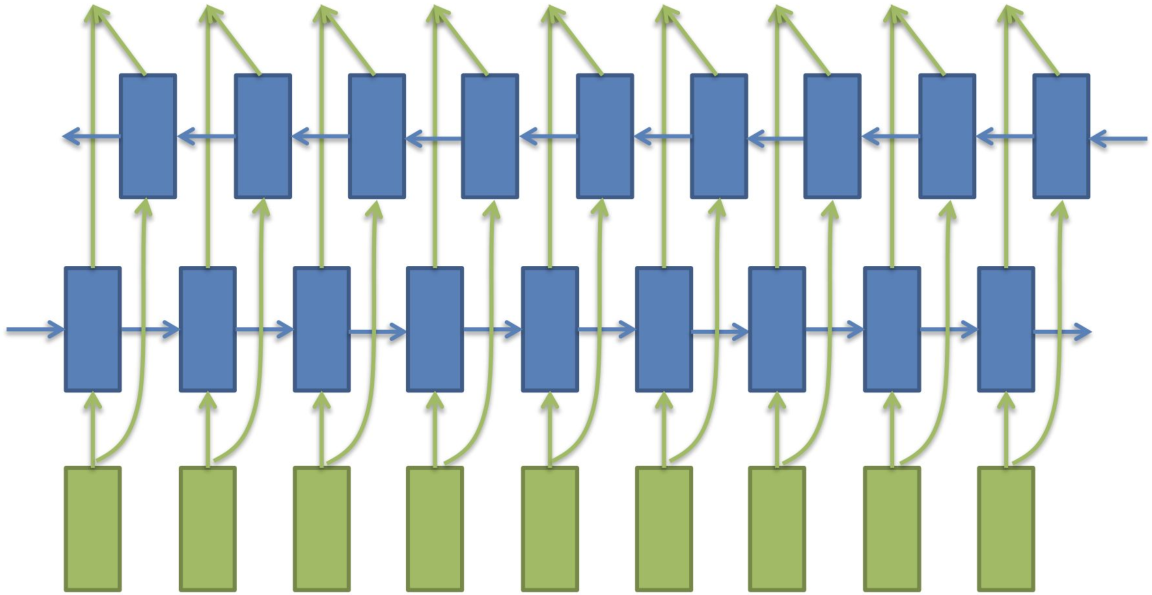
You can create a deep RNN by stacking



Source: [link]

Dwukierunkowe sieci rekurencyjne

Concatenate two RNNs: one going forward in time, one back in time.



Source: [link]

Dziękuję za uwagę!