VISSZACSATOLT (REKURRENS) NEURONHÁLÓK

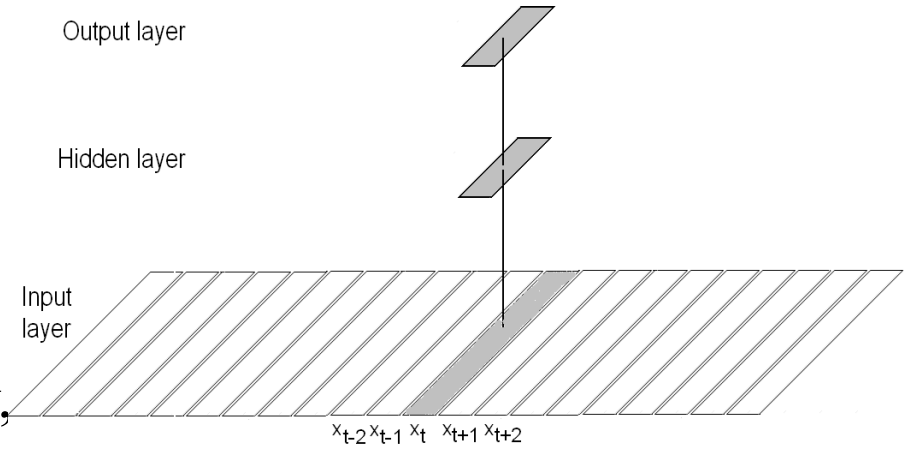
A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.

Időbeli sorozatok modellezése

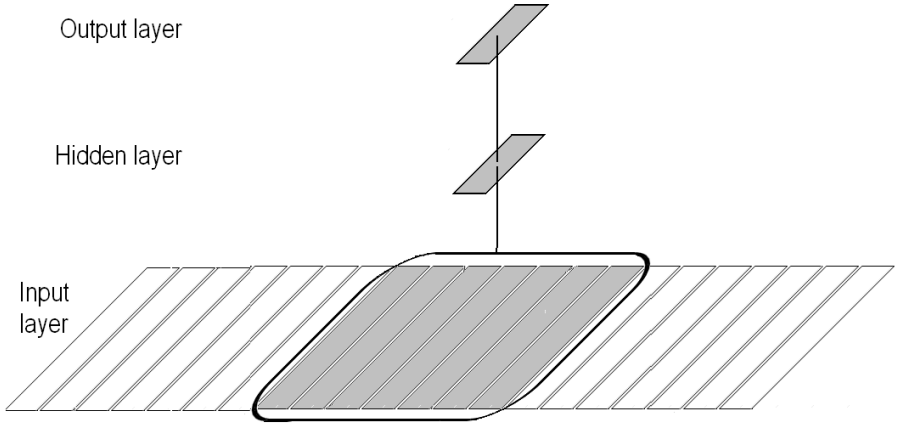
- Eddig feltételeztük, hogy az egymás után következő példák függetlenek egymástól
  - Ez a legtöbb alakfelismerési feladat esetén igaz is
    - Pl. képi alakfelismerés
- Vannak azonban feladatok, ahol a minták egymásutánisága fontos plusz információt hordoz
  - Tipikusan időbeli sorozatok modellezésénél fordul elő
  - Pl.: beszédfelismerés, nyelvi feldolgozás, kézírás-felismerés, videók elemzése, árfolyambecslés,…
- Kérdés, hogy hogyan tudjuk úgy módosítani a hálót, hogy a szomszédokat is figyelembe vegye
  - Előrecsatolt háló több szomszédos inputon: Time-delay neural network
  - Visszacsatolt hálózat: recurrent neural network
  - Visszacsatolt háló hosszú távú memóriával: Long short-term memory network

Szomszédos inputok figyelembe vétele

- Az eredeti (egyetlen input vektort feldolgozó) hálónk így nézett ki (a vonal teljes kapcsolatot jelez):

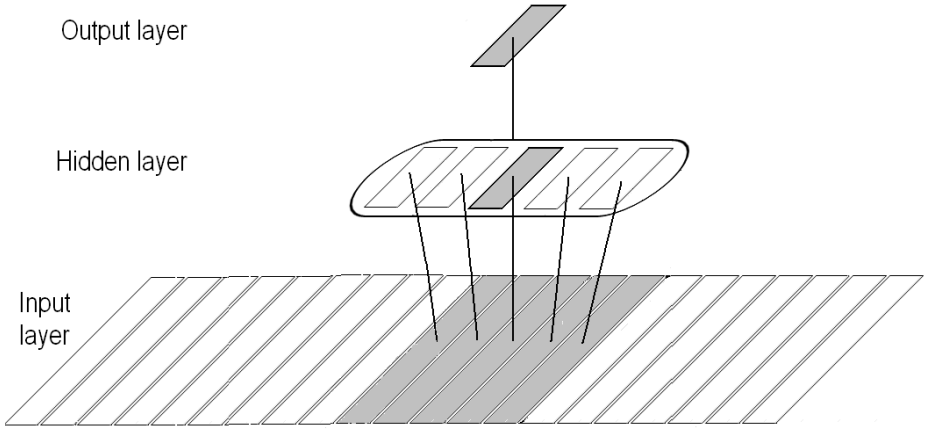


- Ezt egyszerűen módosíthatjuk úgy, hogy az input több szomszédos input vektorból álljon:
  - Előny: semmi módosítást nem igényel sem a hálózatban, sem a tanításában
  - Hátrány 1: az input mérete nagyon megnő
  - Hátrány 2: továbbra is véges környezetet néz



Time-Delay Neural Network

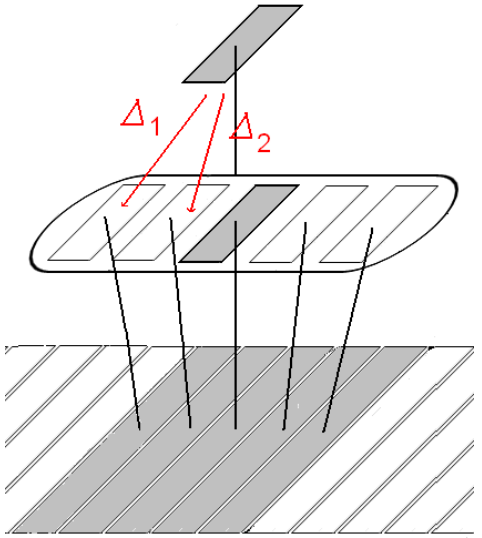
- Több szomszédos inputot is fel akarunk dolgozni
  - De a különböző input vektorokon ugyanazt a feldolgozást végezzük el (legalábbis alacsony szinten)
  - A feldolgozás eredményét csak magasabb szinten egyesítjük
- A rejtett réteg 5 blokkja ugyanazokat a súlyokat használja →"Weight sharing"



- Előny: a feldolgozott input vektorok számának növelésével a rejtett réteg neuronjainak száma nem nő
- Ha a rejtett réteg kicsi, akkor a fölső réteg inputszáma sem nő gyorsan
- Természetesen az alsó és felső feldolgozó rész is állhat több rétegből is

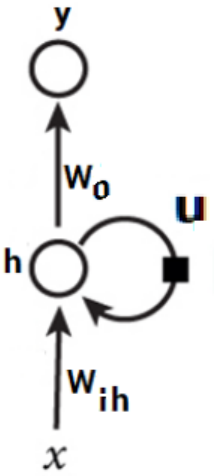
Time-Delay Neural Network 2

- A TDNN előrecsatolt háló
  - A backpropagation tanítás ugyanúgy használható
  - A „weight sharing” okoz némi technikai nehézséget
- Kiértékelés („forward pass”):
  - Ugyanúgy, mint eddig, csak ugyanazt a súlymátrixot többször is felhasználjuk
- Tanítás („backward pass”):
  - a különböző útvonalak mentén kapott delta értékek ugyanazokhoz a súlyokhoz tartoznak, ezért össze kell adni őket, és a súlyokat ezzel módosítani
- A TDNN közeli rokona a konvolúciós hálónak (ld. később)



Visszacsatolt (rekurrens) háló (RNN)

- Valódi visszacsatolást viszünk a hálózatba
- Azaz egy adott pillanatban az input nem pusztán az input vektorból áll, hanem az eggyel korábbi kimenetet is tartalmazza
  - De a kimentet helyett inkább a rejtett réteget szokás visszacsatolni
  - Ezzel memóriával látjuk el a hálózatot, „emlékezni fog” az eggyel korábbi rejtett állapotára is

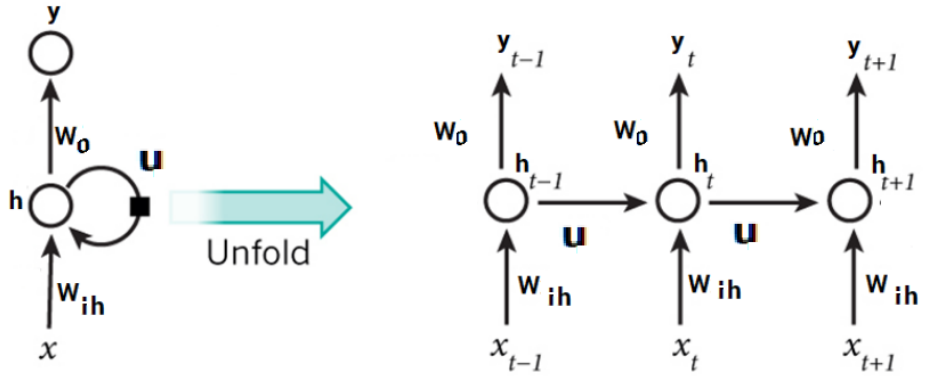


$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_{ih} \cdot \boldsymbol{x}_t + \boldsymbol{U} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b})$$
$$\hat{\boldsymbol{y}}_t = \sigma(\boldsymbol{W}_o \cdot \boldsymbol{h}_t)$$

$\boldsymbol{x}_t$	Input (feature) vector at time t
$\hat{\boldsymbol{y}}_t$	Network output vector at time t
$\boldsymbol{h}_t$	Network internal (hidden) states vector at time t
$\boldsymbol{W}_{ih}$	Weight matrix from input to hidden
$\boldsymbol{W}_o$	Weight matrix from hidden to output
$\boldsymbol{U}$	Weight matrix from hidden to hidden
$\boldsymbol{b}$	Bias parameter vector

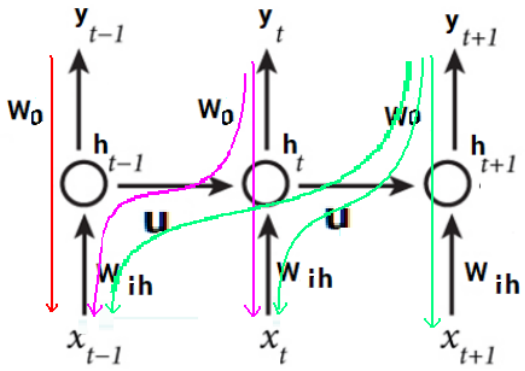
Backpropagation through time

- Hogyan lehet az RNN-t kiértékelni?
  - Az időben előre, azaz balról jobbra, vektorról vektorra haladva
  - Nem tudunk átugrani vektorokat
  - A legelső lépésnél valahogy inicializálni kell h_{t-1}-et
- Hogyan lehet az RNN-t betanítani?
  - Elvileg minden lépésben kell az előző h_{t-1} → végtelen rekurzió
  - A gyakorlatban azonban minden tanító adat véges
    - Vagy mesterségesen is elvághatjuk…
  - Így a hálózatot időben „kiteríthetjük” (unfolding)
  - És taníthatjuk back-propagation-nel



Backpropagation through time

- Mik a „backpropagation through time” nehézségei?
  - A különböző pozíción levő „másolatokhoz” ugyanazok a súlyok tartoznak!
    - A Δ-kat össze kell gyűjteni, ugyanúgy mint „weight sharing” esetén láttuk



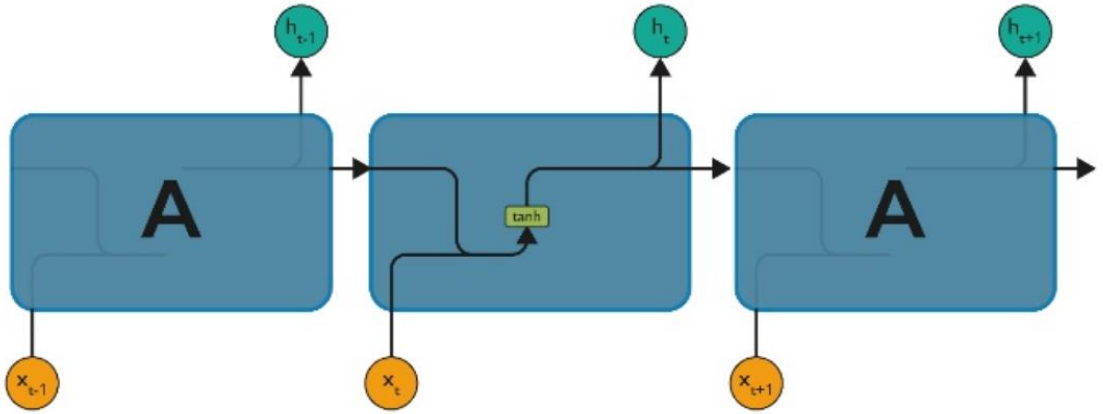
  - Hosszú visszacsatolási útvonalak vannak
    - Elvileg az adott pillanatban adott kimenetet az összes korábbi bemenet befolyásolja
    - Gyakorlatilag azonban a hosszú útvonalak mentén a gradiensek elveszhetnek vagy „felrobbanhatnak”
    - Az RNN tanítása során instabilitási problémák jelentkezhetnek
    - Ha sikerül betanítani, akkor sem tud igazán hosszú távú összefüggéseket megtanulni
    - Tkp. ugyanaz a probléma, mint mély hálók tanításánál!

Long short-term memory (LSTM) háló

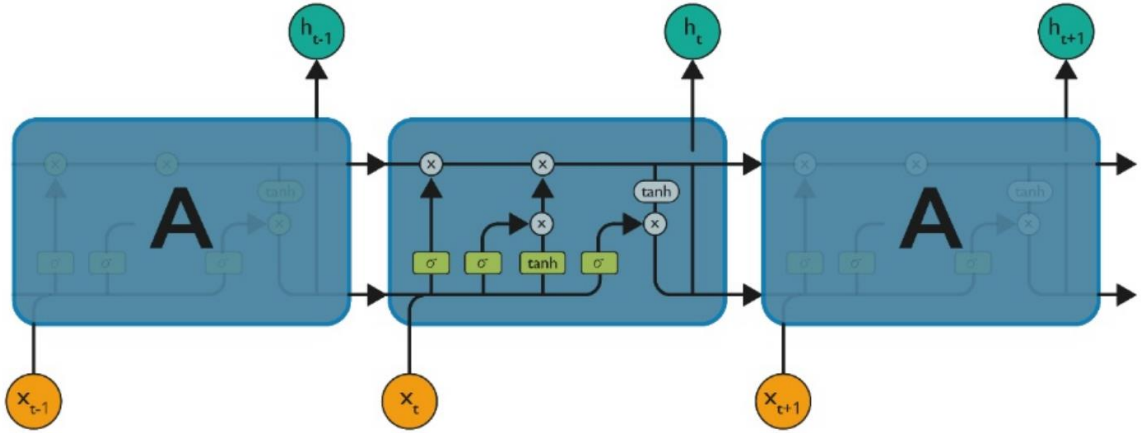
- Tanulhatóvá szeretnénk tenni, hogy az egyes visszacsatolt neuronok mely korábbi inputokat felejtsenek el, és melyekre emlékezzenek
  - Ezzel a hosszú távú emlékezés problémáját is megoldanánk
- Egy külön belső állapotot hozunk létre, amely memóriaként fog működni, és kevésbé lesz érzékeny a backpropagation lépésekre
- Az információ több párhuzamos útvonalon fog áramlani
  - Külön „kapu” fogja szabályozni a memória törlését („forget gate”)
  - Azt, hogy az adott inputból mire kell emlékezni („input gate”)
  - És hogy hogyan álljon össze a kimenet („output gate”)
- Az így kapott modell lesz az LSTM neuron, ilyenekre fogjuk lecserélni a korábbi visszacsatolt neuronokat

RNN vs. LSTM neuron

- RNN (tanh aktivációval):

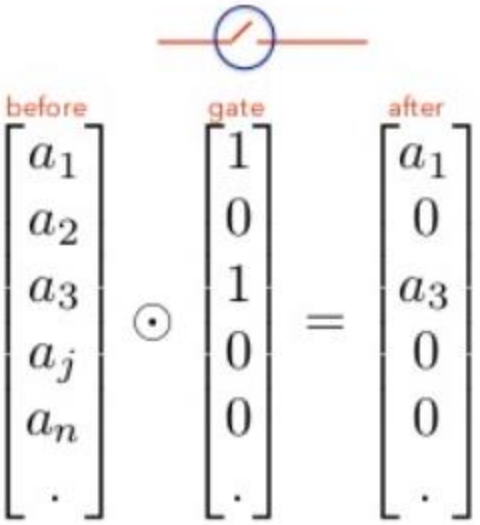


- LSTM:



A kapuk működése

- A kapuk súlyai komponensenként szorozzák meg a bemenet értékeit
- A kapuk súlyait 0 és 1 közt tartjuk szigmoid függvénnyel
- Az optimális súlyértékeket tanulással határozzuk meg
- Példa 0 és 1 kapu-súlyokkal:

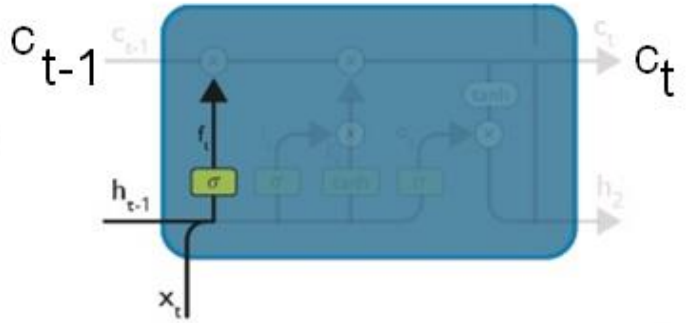


$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_j \\ a_n \\ . \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ . \end{bmatrix} = \begin{bmatrix} a_1 \\ 0 \\ a_3 \\ 0 \\ 0 \\ . \end{bmatrix}$$

LSTM neuron

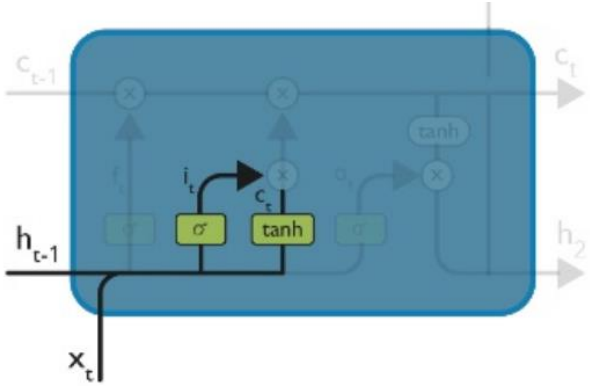
- Forget gate: mely komponenseket kell elfelejteni a C memóriából

$w_f = Weight$
$h_{t-1} = Output\ from\ the\ previous\ time\ stamp$
$x_t = New\ input$
$b_f = Bias$



$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

- Input gate: mely input komponenseket kell eltárolni C-be



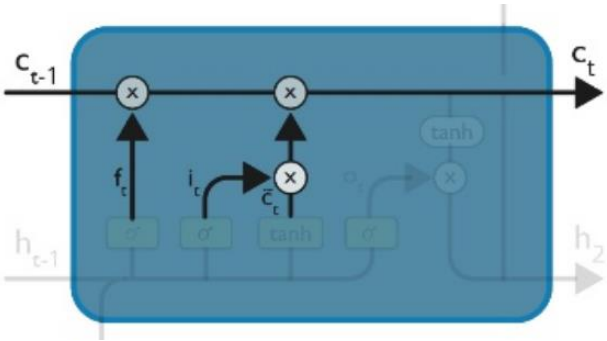
$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c)$$

In the next step, we'll combine these two to update the state.

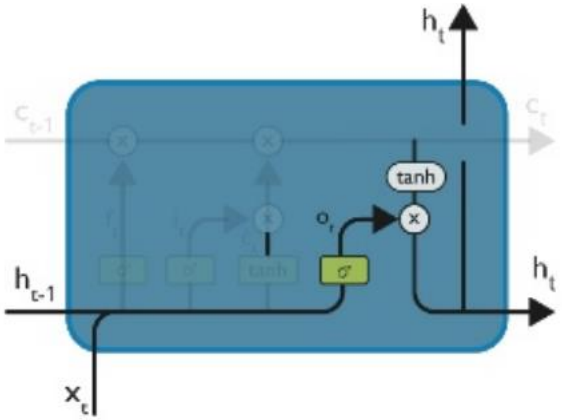
LSTM neuron

- A cell state („memória”) frissítése a forget gate és az input gate segítségével:



$$c_t = f_t * c_{t-1} + i_t^* \tilde{c}_t$$

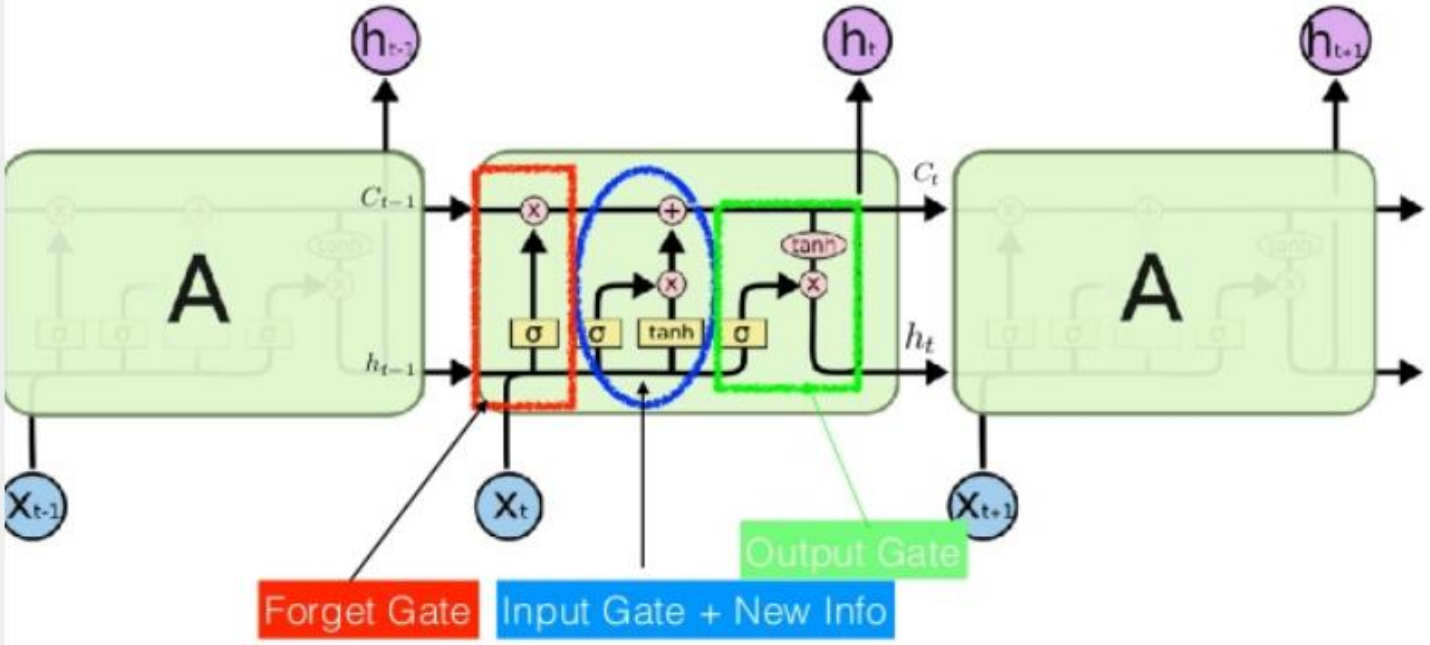
- A rejtett állapot kimeneti értékének kiszámolása:



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

$$h_t = o_t * \tanh(c_t)$$

LSTM összegezve



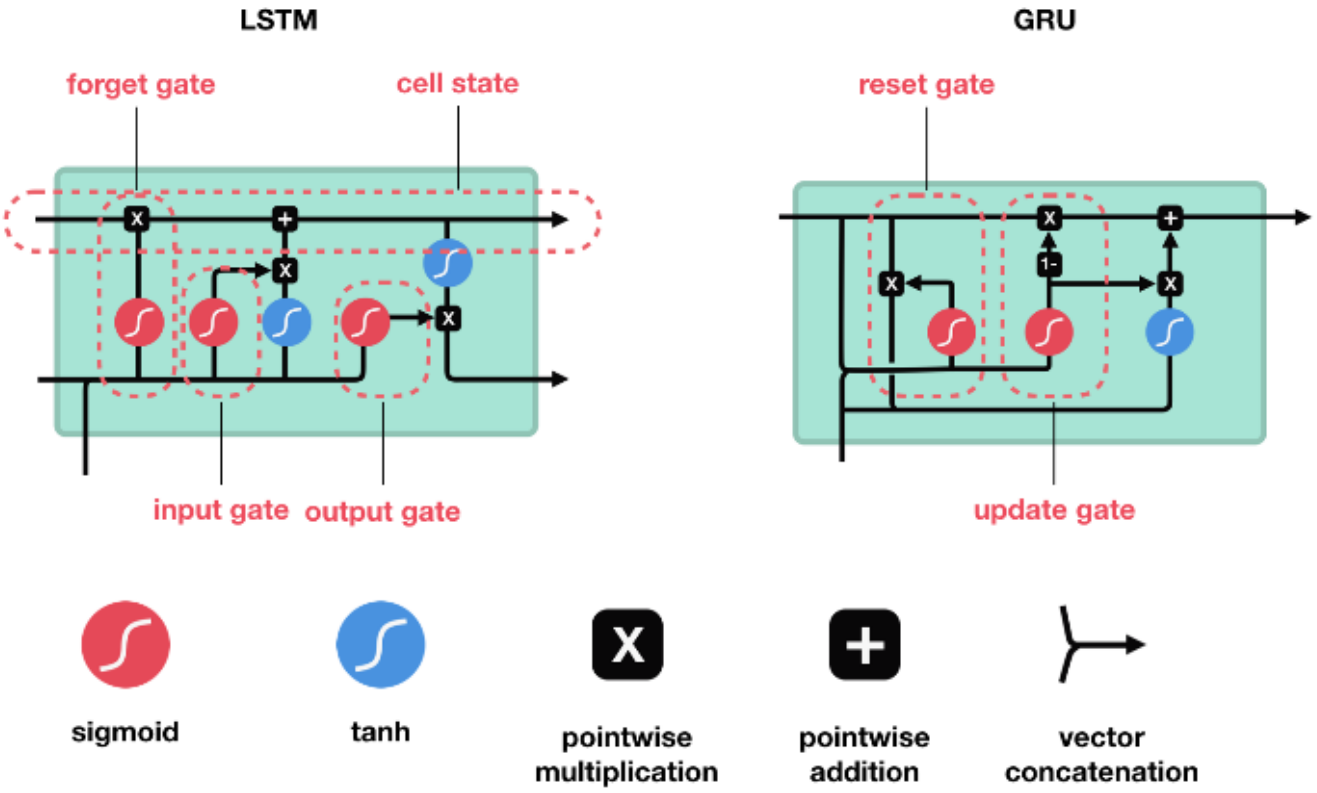
$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad \textit{forget gate}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad \textit{input gate}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad \textit{output gate}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \quad \textit{New cell memory}$$
$$h_t = o_t \circ \sigma_h(c_t) \quad \textit{New Hidden}$$

LSTM hálózat

- LSTM cellákból ugyanúgy építhetünk hálózatot, mint a sima visszacsatolt neuronokból
  - A betanítás persze bonyolultabb és lassabb lesz
  - De a legtöbb feladaton tényleg jobb eredményeket ad
- LSTM variánsai:
  - Vannak még több útvonalat tartalmazó változatok
    - LSTM „peephole” kapcsolatokkal
  - Vagy épp ellenkezőleg, egyszerűsített változatok
    - Pl. GRU – gated recurrent unit (ld. következő 2 dia)
  - Jelenleg az egyik legaktívabb kutatási terület

- https://www.slideshare.net/LarryGuo2/chapter-10-170505-l
- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM vs. GRU

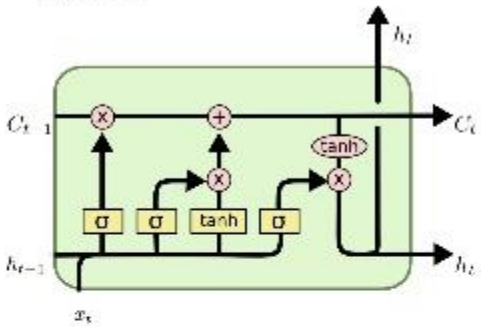


- A GRU kevesebb kaput használ, de a tapasztalatok szerint könnyebben, gyorsabban tanítható, és hasonló pontosságra képes, mint az LSTM

LSTM vs. GRU

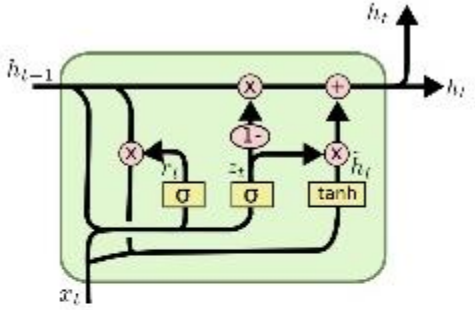
- A kevesebb kapu miatt egyetlen GRU egység kevesebb egyenlettel írható le, mint az LSTM

- LSTM



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o [h_{t-1}, x_t] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- GRU



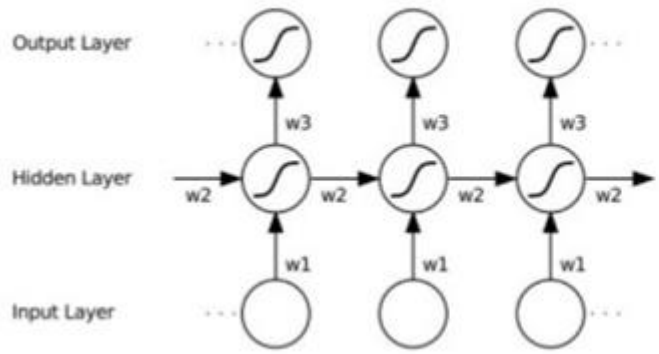
(Biases are omitted.)

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

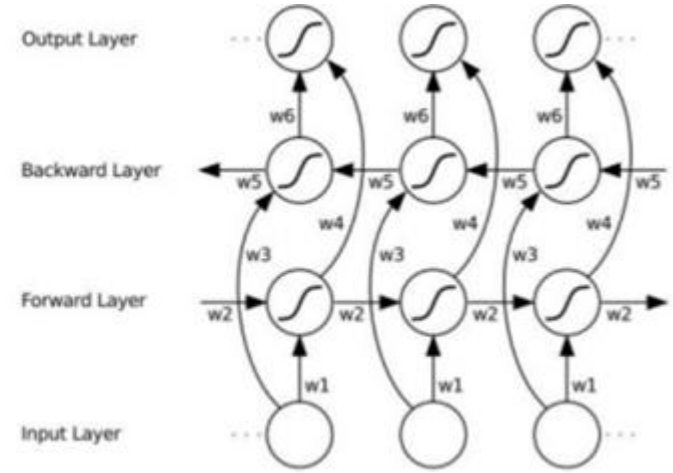
Kétirányú (bidirectional) rekurrens háló

- A jelet mindkét irányban feldolgozzuk
  - Mindkét irányhoz tartozni fog 1-1 rejtett réteg (egymástól függetlenek!)
  - A kimeneti réteg felhasználja mindkét rejtett réteget
    - Előny: az előző és a későbbi kontextust is figyelembe veszi
    - Hátrány: valós időben nem megy

RNN:

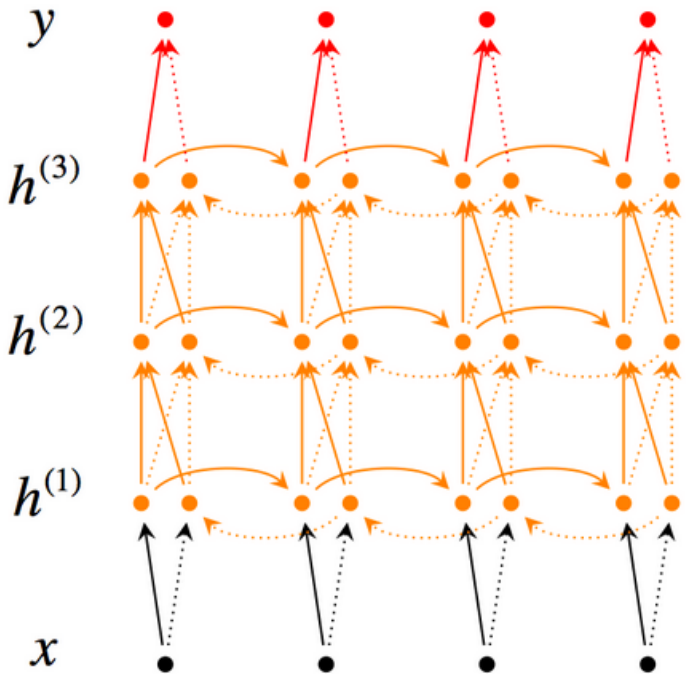


BRNN:



Mély rekurrens hálók

- Természetesen rekurrens hálónál is lehet több réteget egymásra pakolni
  - Akár kétirányút is
  - De kevésbé (illetve kevesebb réteget) szokás, mint előrecsatolt rétegekkel
    - A rekurrens hálónak eleve nagyobb a reprezentációs ereje
    - A tanítása viszont lassabb és bonyodalmasabb



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

KÖSZÖNÖM A FIGYELMET!

A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.