LSTM

Kognitiv teknologi och artificiell intelligens
729G83

LINKÖPINGS UNIVERSITET

Översikt

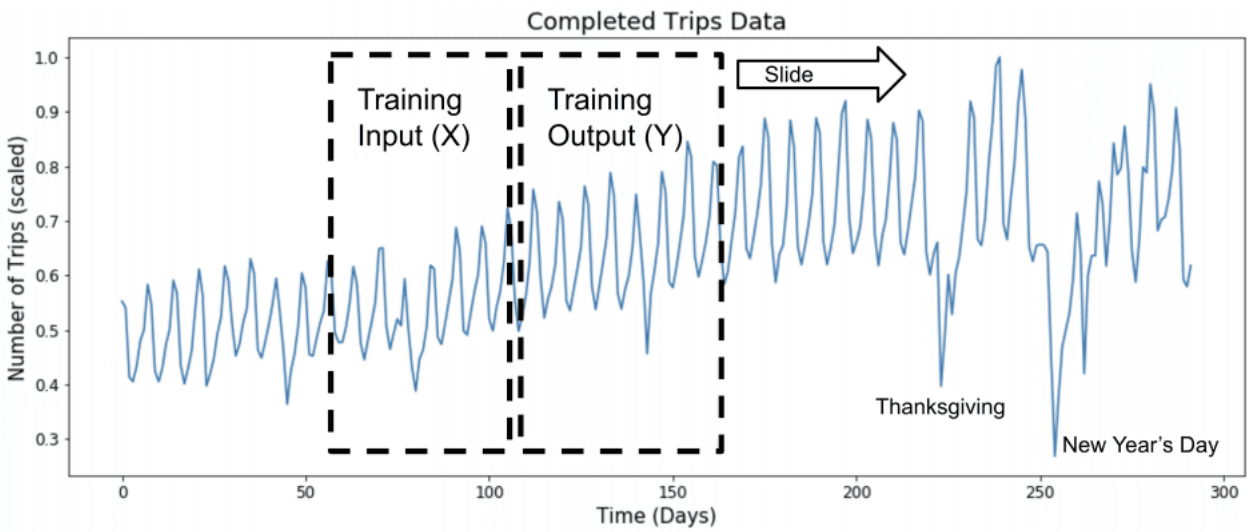
- Sekvensinlärning
- Återkopplade nät (recurrent networks)
  - INTE = dubbelriktat kopplade nät i emergent
- Long Short-Term Memory (LSTM)
- Praktiska råd
  - Transfer learning för CNN
  - Förberedelse av data för LSTM

Sekvensinlärning

- T.ex. följa börsen, förutsäga nästa dags aktiekurser
  - Variabel (oändlig) längd på input
  - Target (label) finns bara periodiskt (t.ex. i början av varje dag)

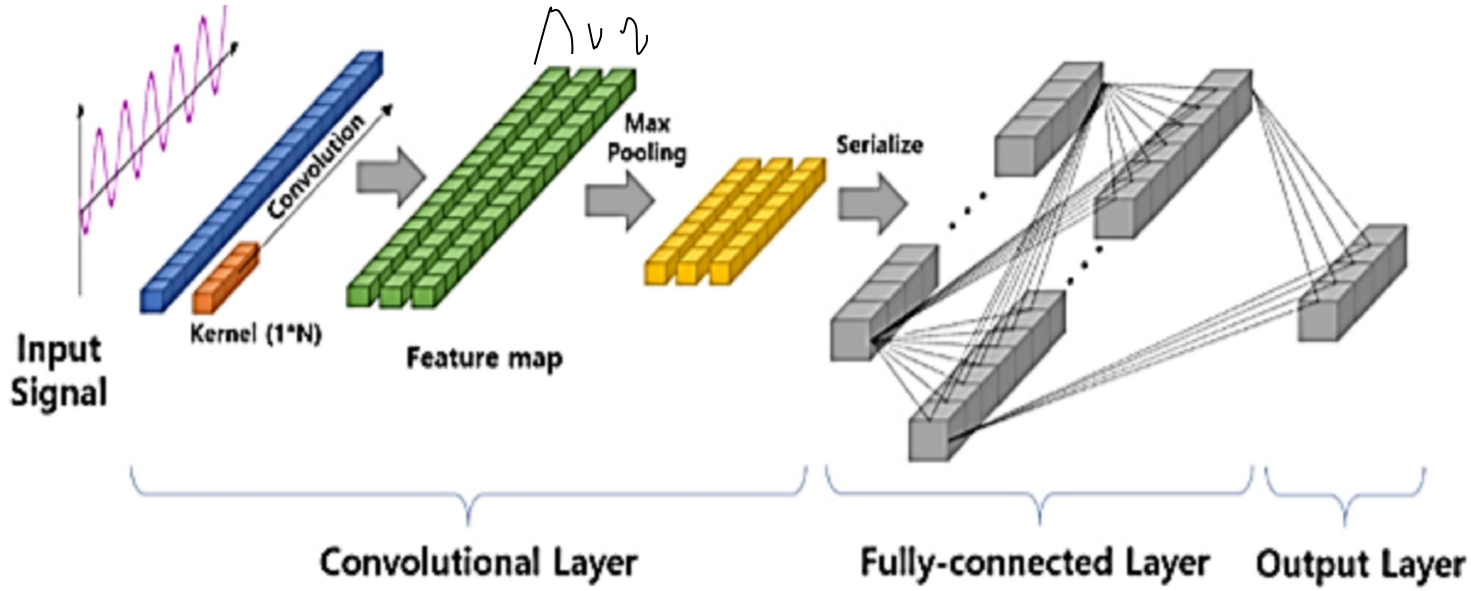
Använd 1D CNN

- Fönsterstorlek, t.ex. 50 tidssteg
- Dra fönster över datasekvensen



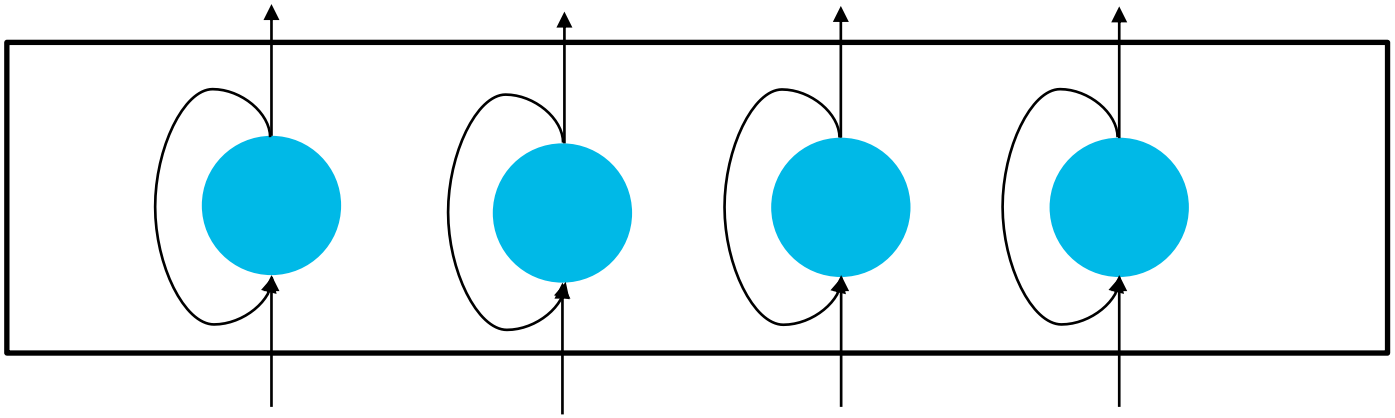
1D CNN

- Första lagret tittar på *beroenden inom fönstret*
  - Nästa lager kombinerar dessa inom närliggande fönster, osv.



För beroenden med längre spann

- T.ex. läsa kapitel där tolkning av slutet beror på vad som stod i början
- Recurrent Neural Network (RNN)

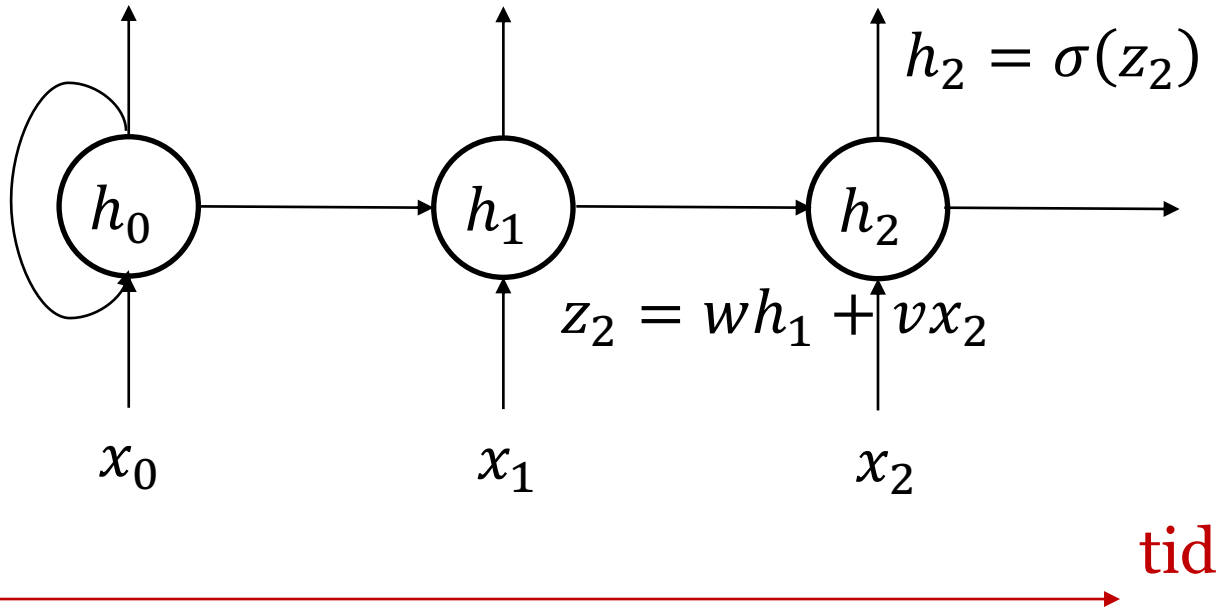


Kommer ihåg sin aktivering sen föregående input

En nod i ett Simple RNN (SRN)

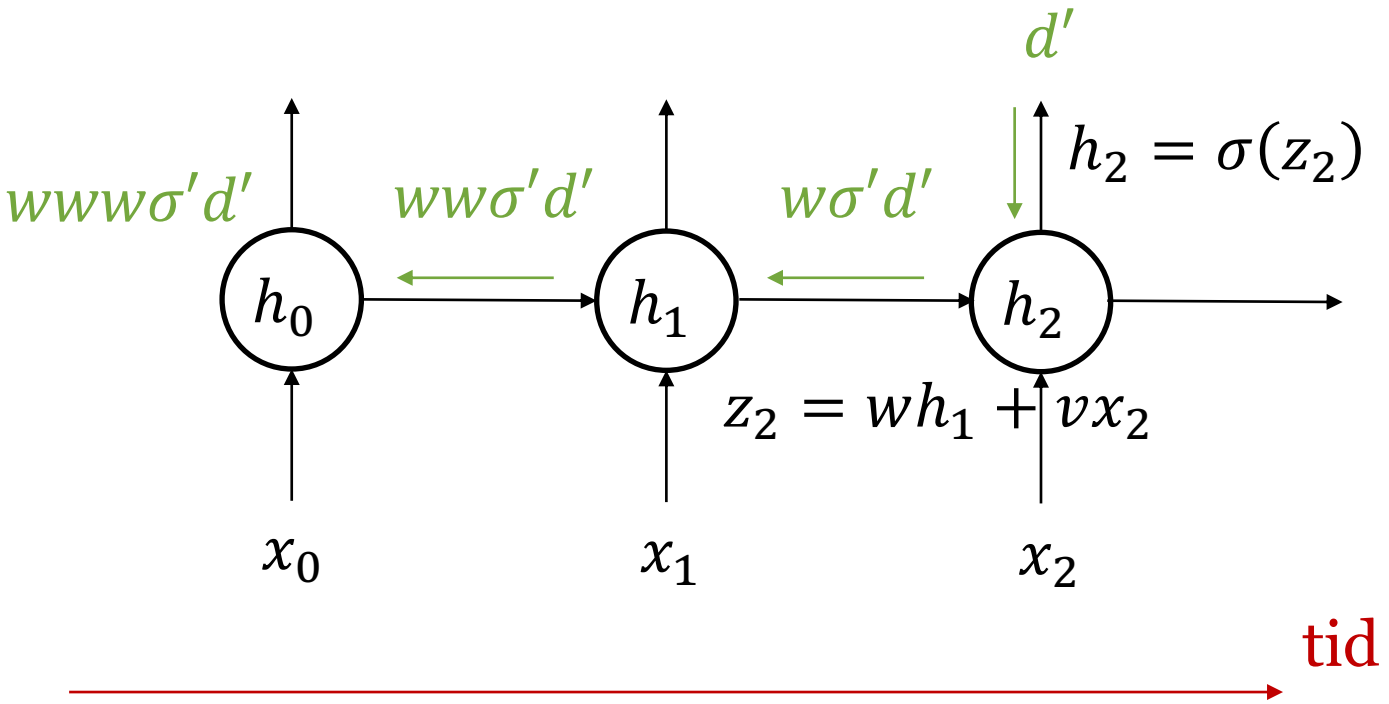
Aktivering av **en nod** över tid

Varje tidssteg: kombinera minne h_{t-1} med ny info x_t



En nod i ett Simple RNN (SRN)

Gradientflödet i **en nod** över tid:



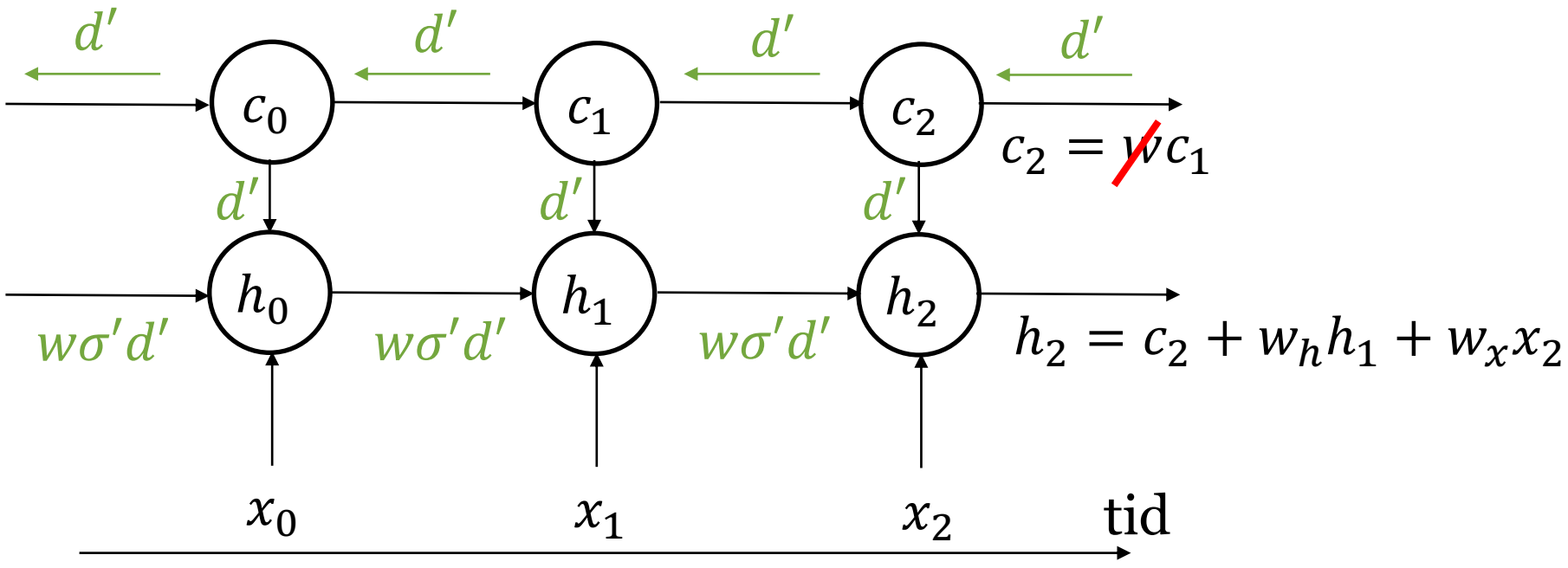
Problemet med långa sekvenser

- När man tränar ett RNN multipliceras återkopplingens gradienter med samma viktmatris, gång på gång, t.ex.
  - $\Delta w = h * 0.9 * 0.9 * 0.9 * \cdots * \sigma' d'$
    - För varje tidssteg bakåt blir gradienten mindre … *vanishing gradient*
  - När får man *exploding gradients*?

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

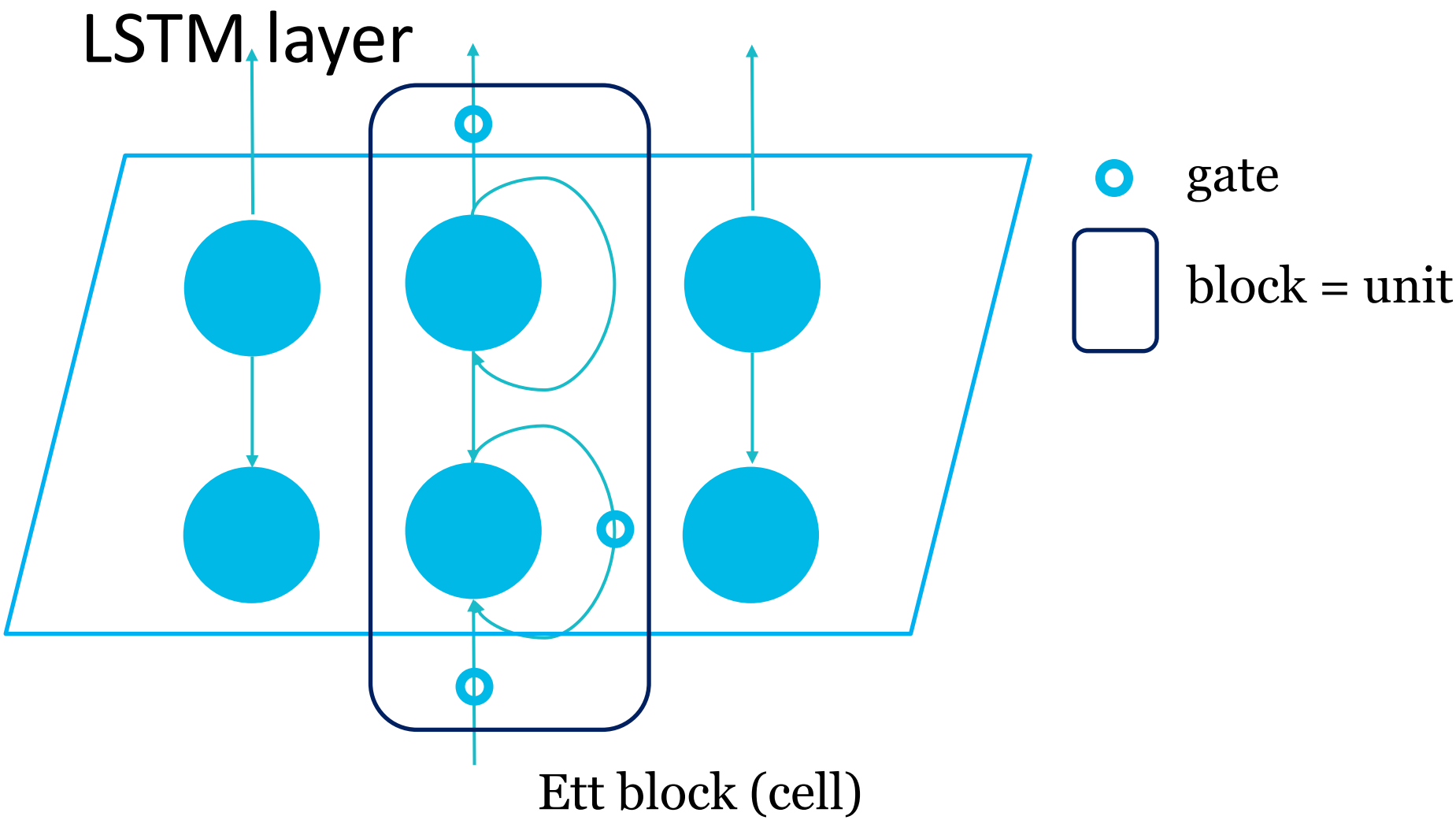
Idéskiss: dela upp minnesfunktionen

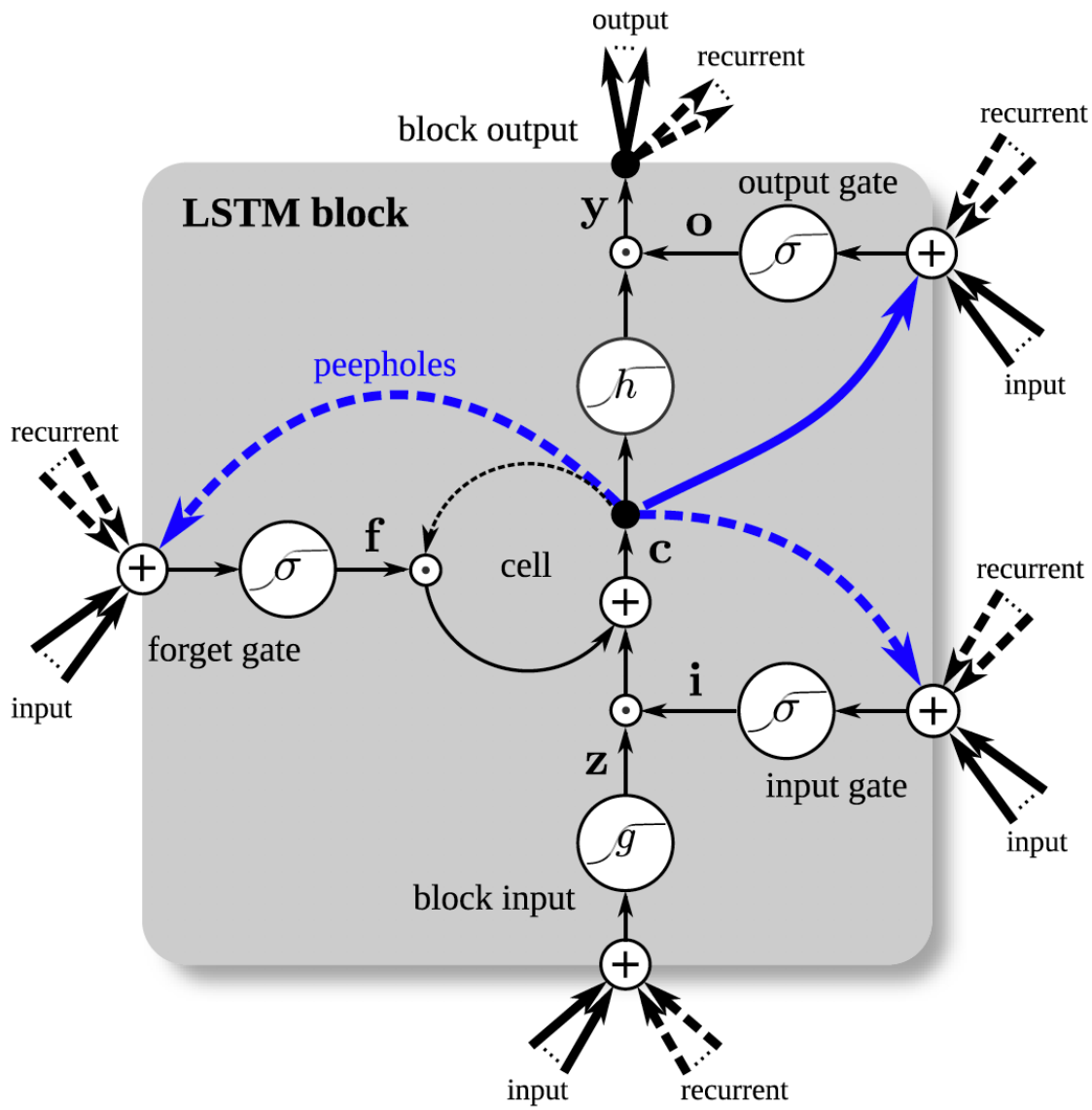
c bevarar minnet helt (ingen decay, $w = 1$), identitet
h kombinerar minnet med ny input, alltså diff mot c

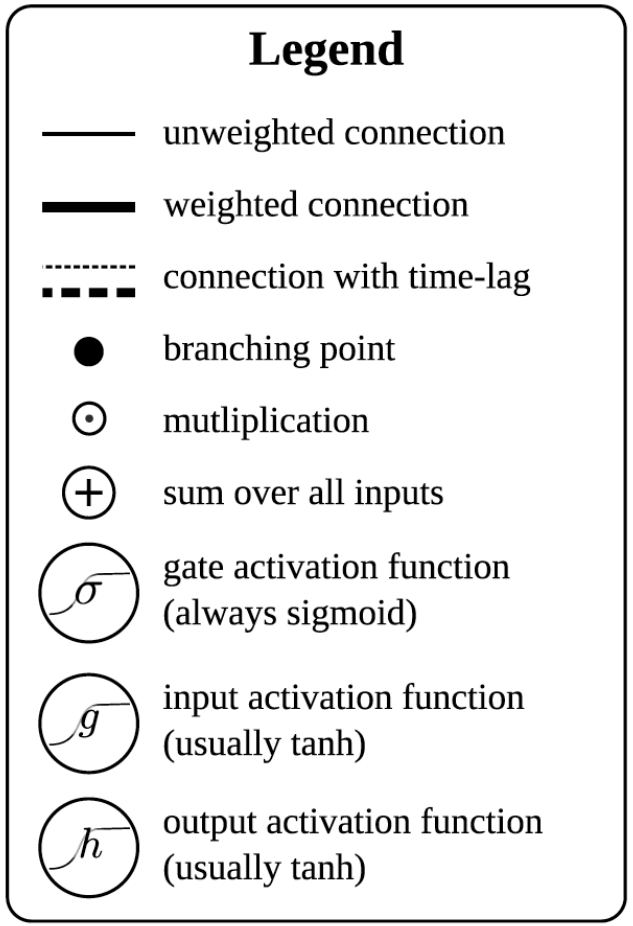


Long Short-Term Memory (LSTM)

- Själva tidssteget (minnet bakåt i tiden) ska gå utan förlust av information
  - Gradient alltid = 1
  - Lagrade informationen tynar aldrig bort
- För att införa dynamik: gates som öppnas och stängs
  - *Punktvis* uppdatering / utläsning av relevanta minnesbitar (celler)
  - *En punkt* = en nod + gates = en LSTM cell


LSTM layer
gate
block = unit
Ett block (cell)

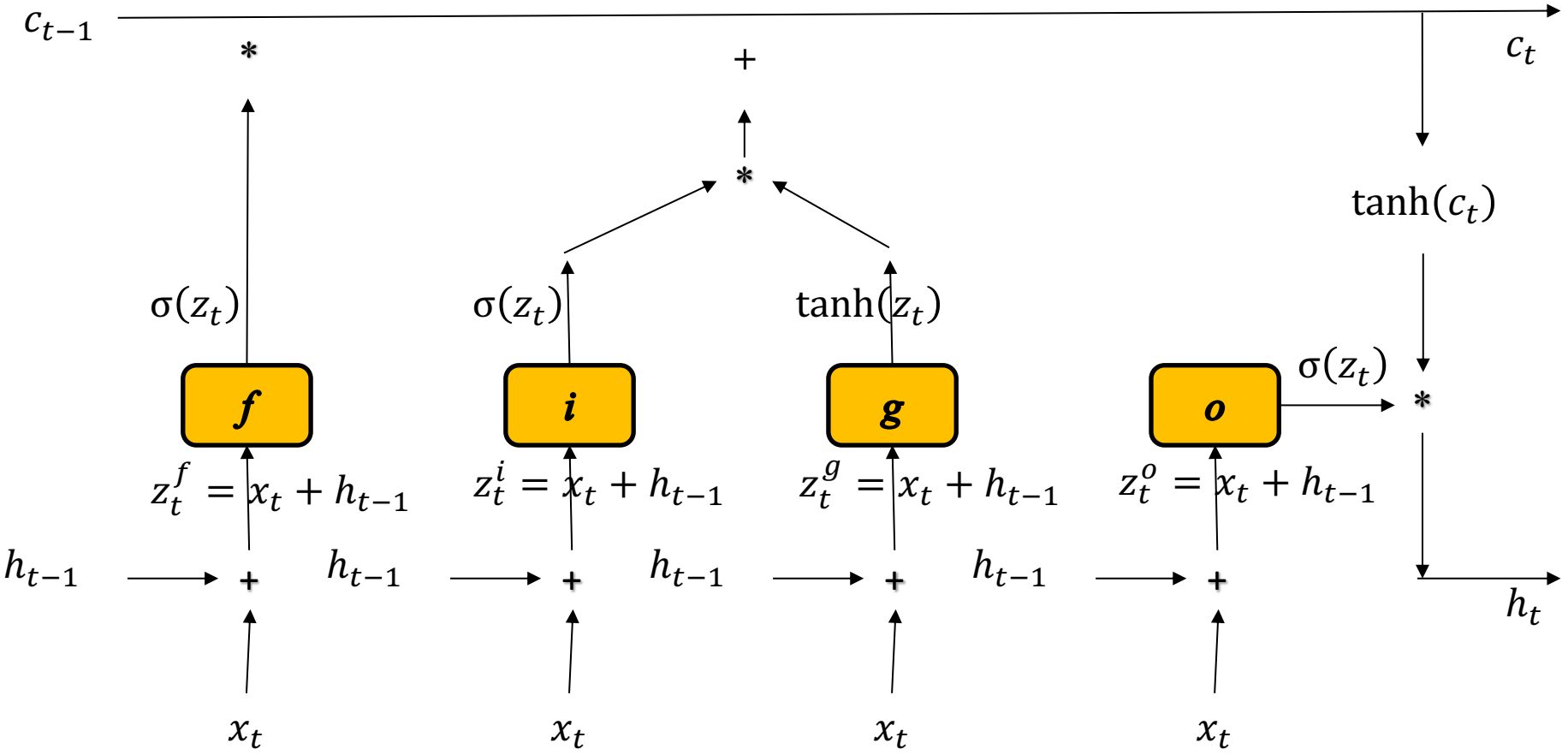




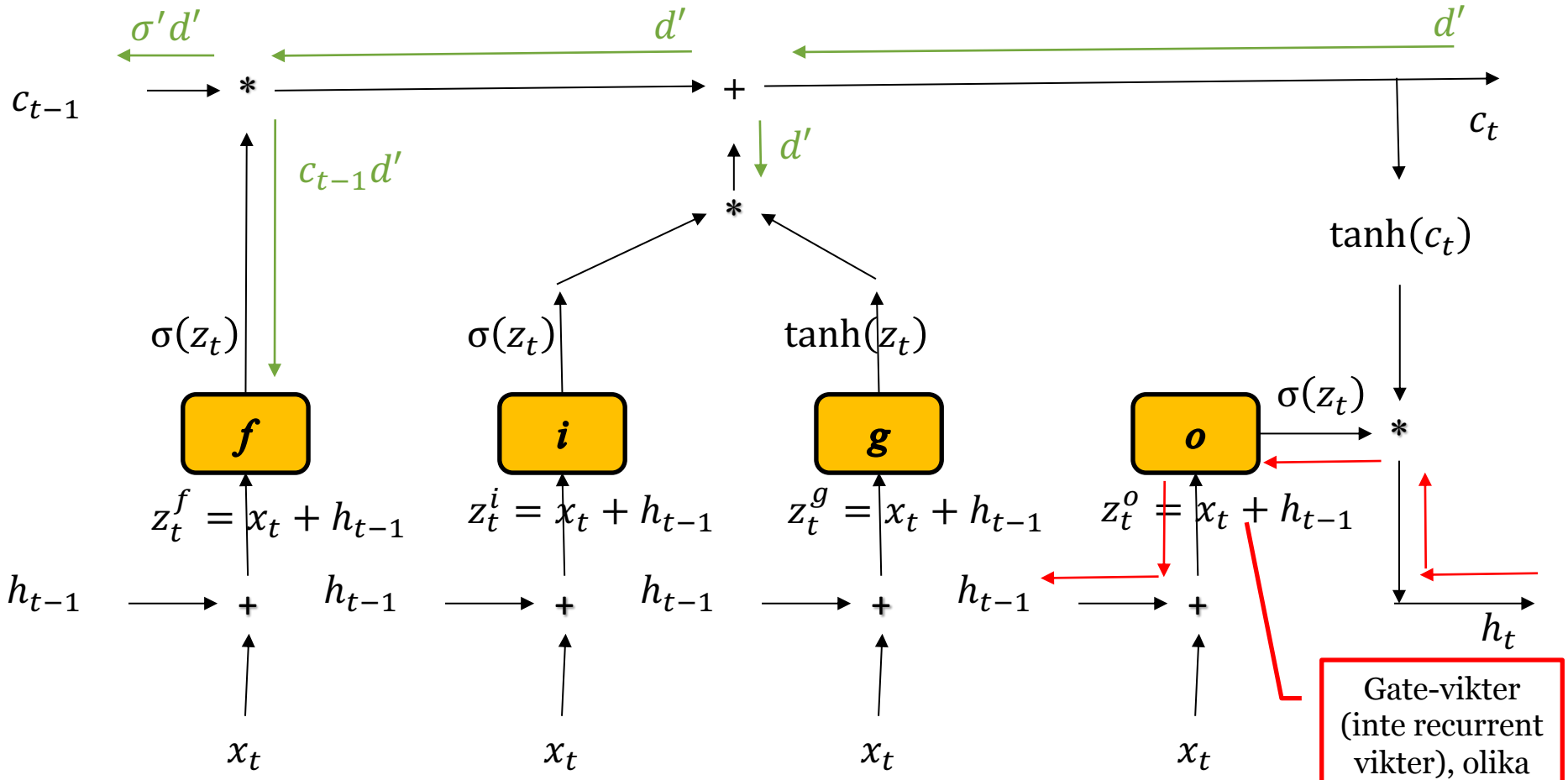
LINKÖPINGS UNIVERSITET

Greff et al. 2017: LSTM: A Search Space Odyssey

Bra gradientflöde

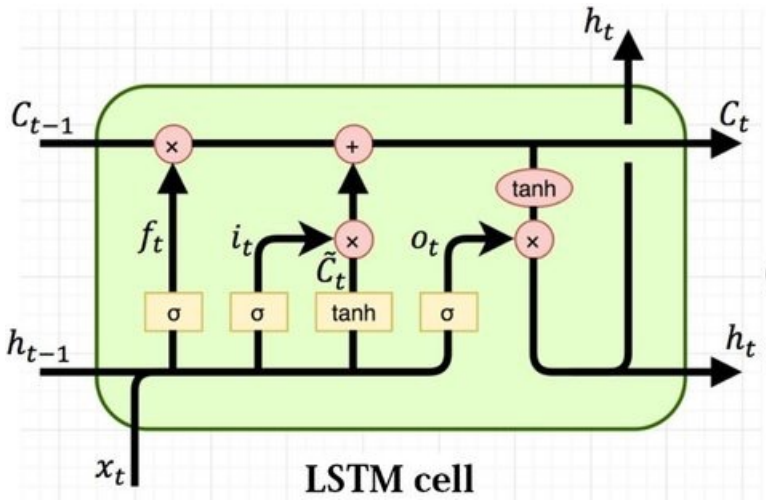


Bra gradientflöde



Gate-vikter (inte recurrent vikter), olika vid olika tidssteg

c: cell state (intern) h: hidden state (exponerad)



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

- Fritt gradientflöde för *C* vid bakåtpropagering
  - Endast add och mult. med gate:s (som ju är olika för varje tidssteg bakåt)

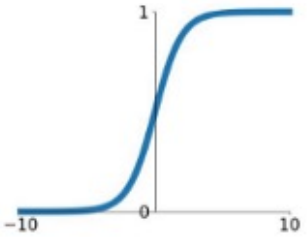
Förbereda data för LSTM

Hur dela upp långa sekvenser av data?

- Mata en datapunkt i taget (online learning)
- Mata batches, normalisera inom varje batch
  - Zero-centered, unit variance
  - (Vill träffa σ och tanh på rätt ställe)

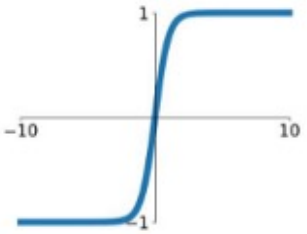
Sigmoid

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



tanh

$$\tanh(x)$$



Ange batch_size i LSTM (i Keras)

- Data.shape: (*total_series, timesteps, features*)
- LSTM
  - input_shape: (*timesteps, features*)
  - batch_input_shape: (*number_of_series_in_batch, timesteps, features*)

Om man inte vill batch:a

Formen på batch

En lång serie (t.ex. aktie A över tid)

Bach 1	Batch 2	Batch 3	Batch 4	Batch 5
Serie del1	del2	del3	del4	del5

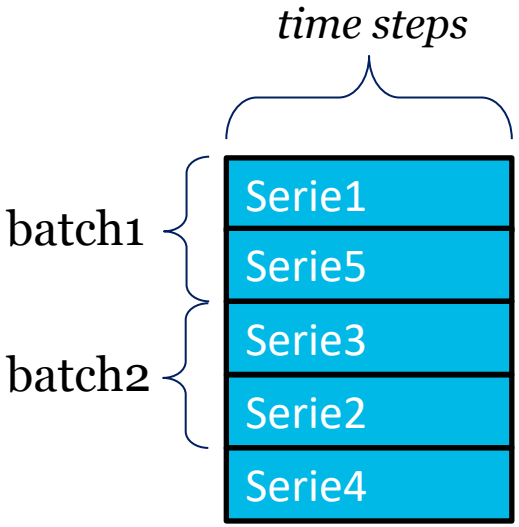
Serie del1
-”- del2
-”- del3
-”- del4
-”- del5

Forma om input-tensorn, manuellt,
t.ex. från shape (1, ts, feat) till (batches, ts/batches, feat)

En del = en batch

- Sätt stateful=True i LSTM-lagret, så LSTM:ets hidden state fortsätter mellan delarna
- Sätt shuffle=False i fit-metoden

Flera oberoende serier



Flera slumpvis valda serier = en batch

Som Keras fungerar kommer:
- data kommer automatiskt blandas före varje epok
- LSTM rullas ut i tiden med *time_steps* antal steg (djupet i tiden = *time_steps*)
- backprop startas efter att *time_steps* datapunkter har setts för alla serier inom batch:en

Time steps bör inte göras alltför stort

- Keras kommer att köra igenom alla tidssteg i alla delar/serier i batchen, innan vikterna uppdateras
  - Strategin ger stabilitet i viktuppdateringen
  - Men gör det svårare för Backprop through time, även om man löst recurrent-problemet (dvs. samma problem som med djupa nät)

Transfer learning

Använd förtränat nät till ny uppgift

Ersätt sista lagren

- Ersätt sista lagret i ett redan tränat CNN


```
base_model =
tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                  include_top=False,
                                  weights='imagenet')
```

Läs mer på:
https://www.tensorflow.org/tutorials/images/transfer_learning

Frys modellen

- Stäng av viktuppdatering i alla gamla lager

base_model.trainable = False

- Fixerar även *mean* och *variance* i alla BN lager!
  - It has long been debated whether the moving statistics of the BatchNormalization layer should stay frozen or adapt to the new data. Historically, bn.trainable = False would only stop backprop but would not prevent the training-time statistics update. After extensive testing, we have found that it is *usually* better to freeze the moving statistics in fine-tuning use cases. **Starting in TensorFlow 2.0, setting bn.trainable = False will *also* force the layer to run in inference mode** [i.e. **no** stats update]. **(**https://keras.io/getting_started/faq/#whats-the-difference-between-the-training-argument-in-call-and-the-trainable-attribute **)**

Bygg vidare på basmodellen

- Lägg till ett eget topp-lager (t.ex. eget nytt sista utlager)
- Ladda och förbered egna data

Träna på nya data

- Använd valideringsdata för att regelbundet under träningen kolla generaliseringsförmåga (och indirekt, overfit)
  - Valideringsdata
    - 10-15% av data
    - T.ex. efter varje träningsepok körs en valideringsepok
      - För att kolla overfit, välja bland hyperpar.
  - Testdata
    - 10% av data
    - Används en enda gång för att testa vald modell

Fine-tuning

Finjustering (efterträning)

- Samträna några av topp-lagren eller hela bas-nätet plus nya utlagret på nya data
  - Använd väldigt liten learning rate (t.ex. 1e-5)
  - BN-lager bör köras i inference mode (training=False)
  - Stor risk för overfitting när alla vikter tränas!
- Läs mer på: https://www.tensorflow.org/guide/keras/transfer_learning

rita.kovordanyi@liu.se

www.liu.se

LINKÖPINGS UNIVERSITET