Imperial College London

EE462/EE9CS728/EE9SO25

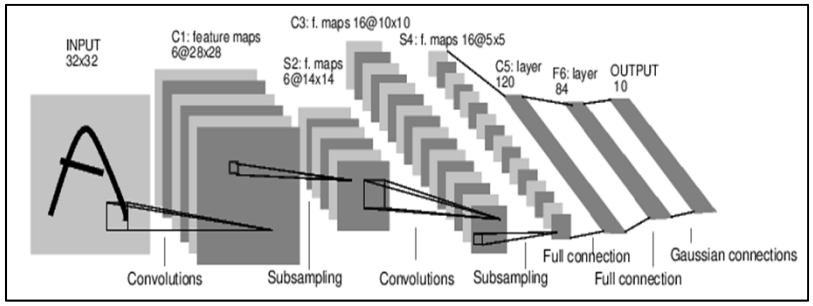
Recurrent Neural Networks (RNN)

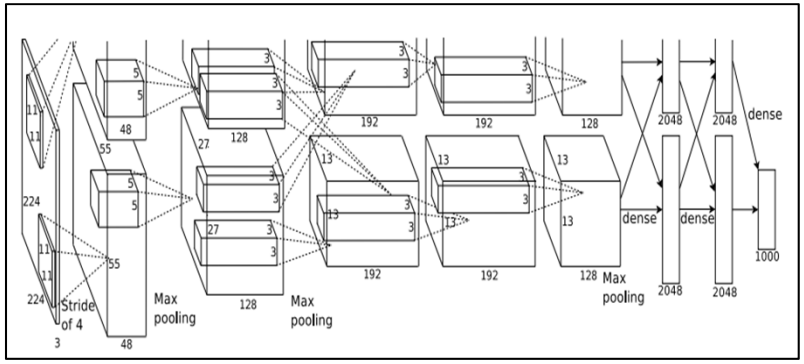
Tae-Kyun (T-K) Kim

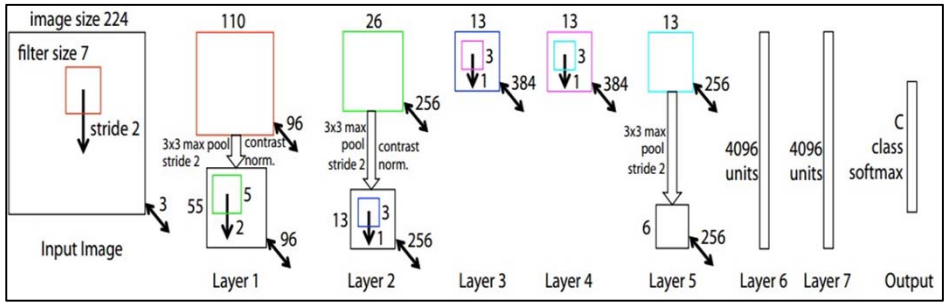
Senior Lecturer

https://labicvl.github.io/

Last Time: ConvNets

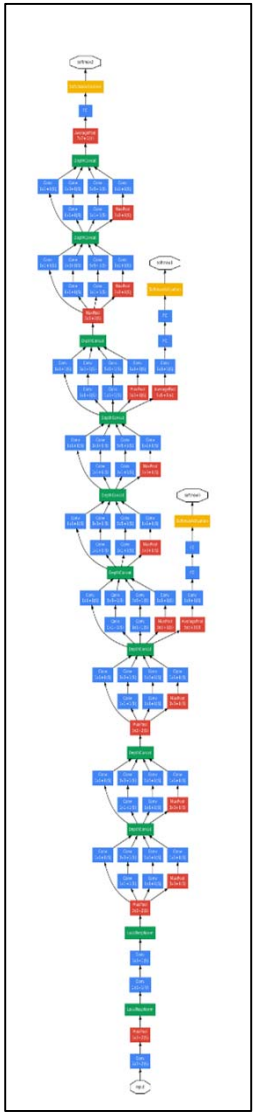






D	E
16 weight layers	19 weight layers
e)	
conv3-64 conv3-64	conv3-64 conv3-64
conv3-128 conv3-128	conv3-128 conv3-128
conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**

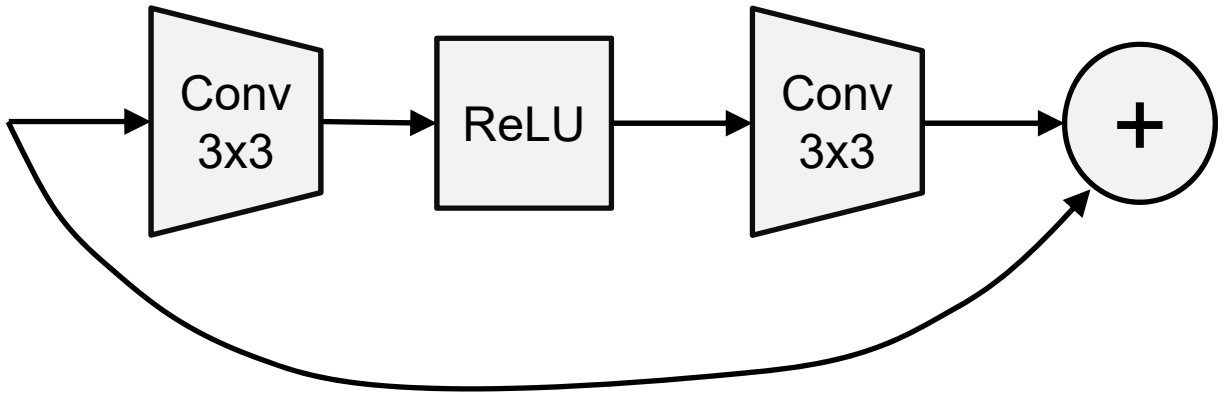
maxpool
FC-4096
FC-4096
FC-1000
soft-max



Neural Network Structure

Standard Neural Networks are Directed Acyclic Graphs (DAGs), and they have a topological ordering.

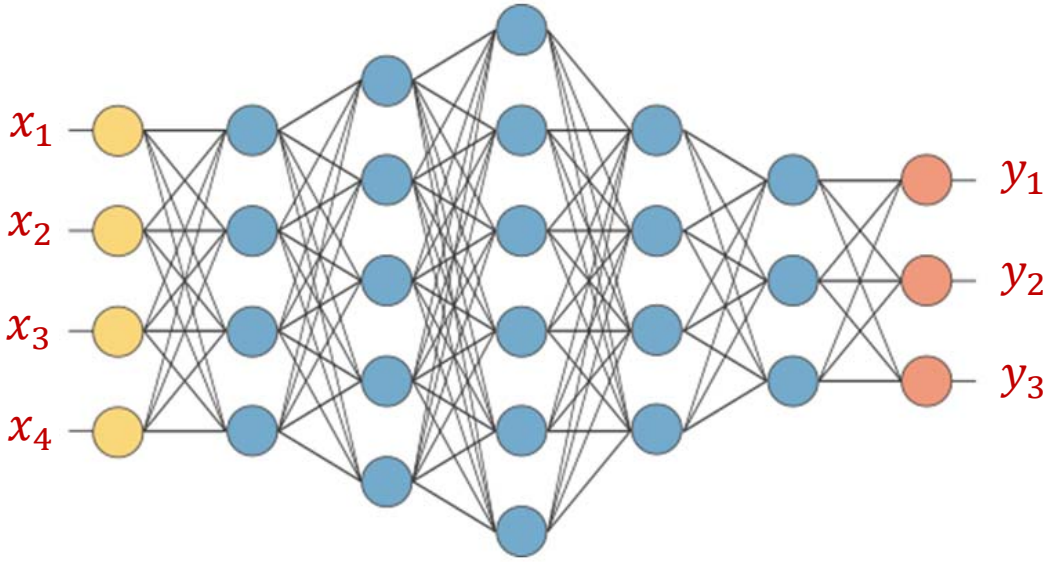
- The topological ordering is used for activation propagation, and for gradient back-propagation.
- They process one input instance at a time.



Conv: Convolutional Neural Network

ReLU: Activation function (Rectified Linear Unit). If the input has a value below zero, the output will be zero, when the input rises above, the output is a linear relationship with the input variable of the form $f(x) = x$.

Fully Connected Network

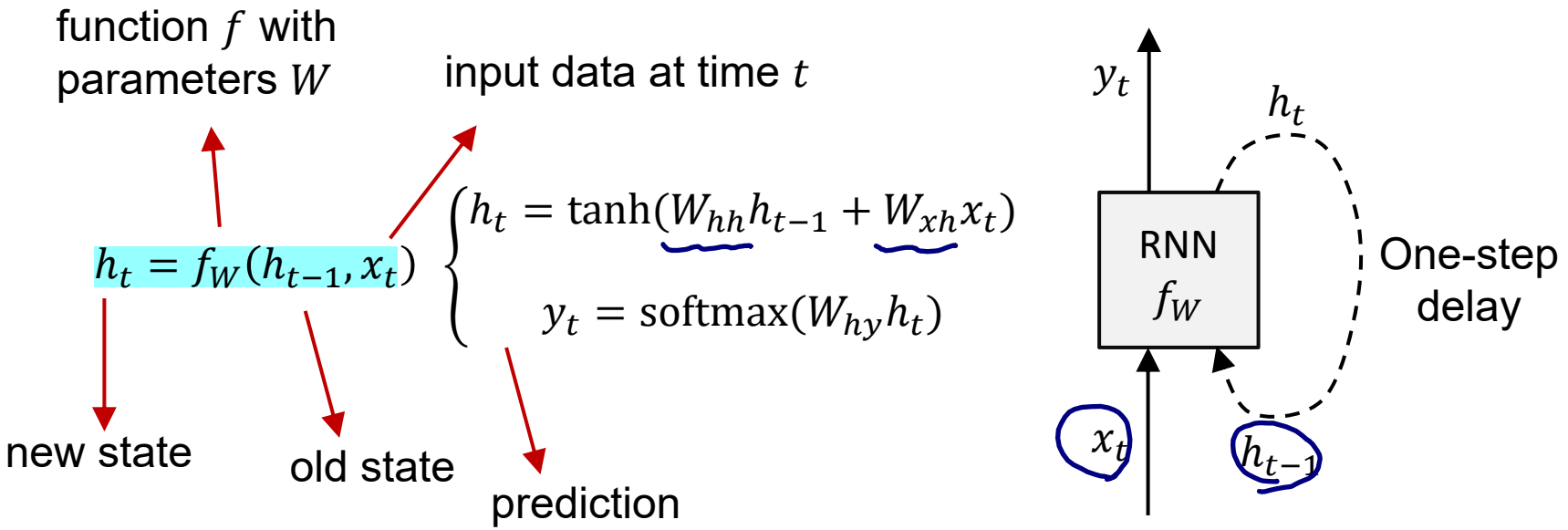


Given $x_1, x_2, \ldots, x_n$;

- If n is very large and growing, this network would become too large.
- Because, in a fully connected layer, each neuron is connected to every neuron in the previous layer, and each connection has it's own weight.
- It's also very expensive in terms of memory (weights) and computation (connections).
- We will now (with Recurrent Neural Networks) input one x_i at a time and re-use the same edge weights.

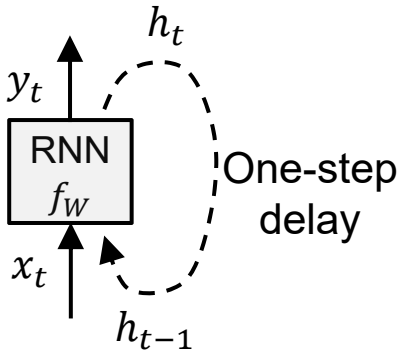
Recurrent Neural Networks

- Recurrent networks introduce cycles and a notion of time.
- They are designed to process a sequence of data $x_1, x_2, \ldots, x_n$ and can produce a sequence of outputs $y_1, y_2, \ldots, y_m$.
- We can process the sequence of data $x_1, x_2, \ldots, x_n$ by applying a recurrence formula at every time step:

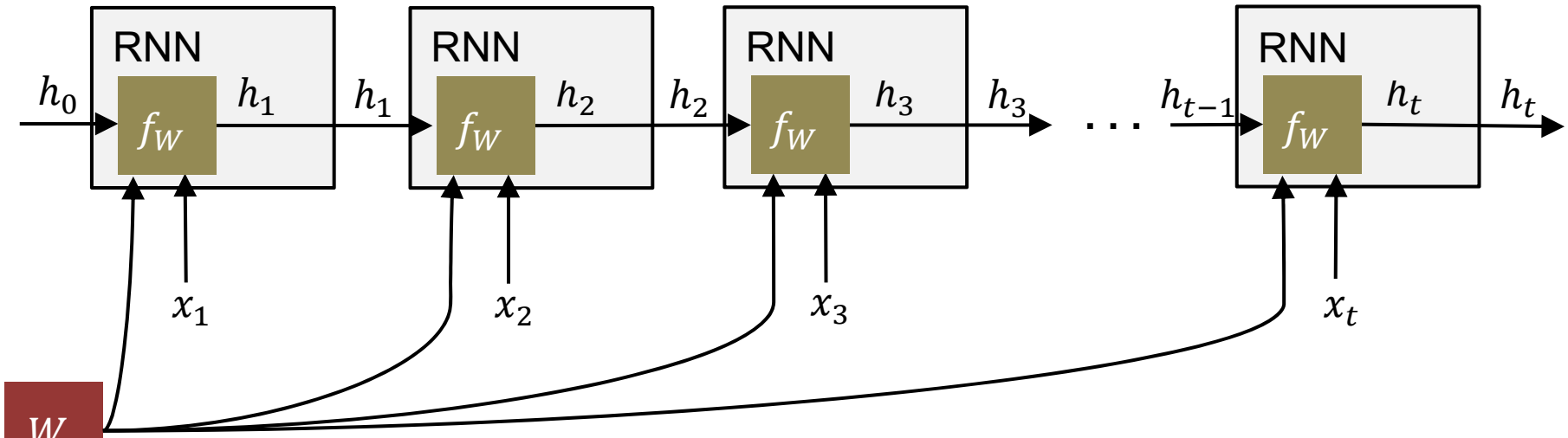


Recurrent Neural Networks

$$h_t = f_W(h_{t-1}, x_t) \quad \begin{cases} h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t = \text{softmax}(W_{hy}h_t) \end{cases}$$



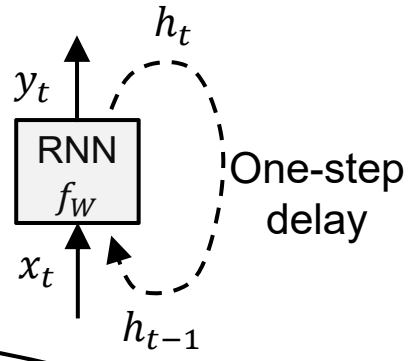
RNNs can be unrolled across multiple time steps:



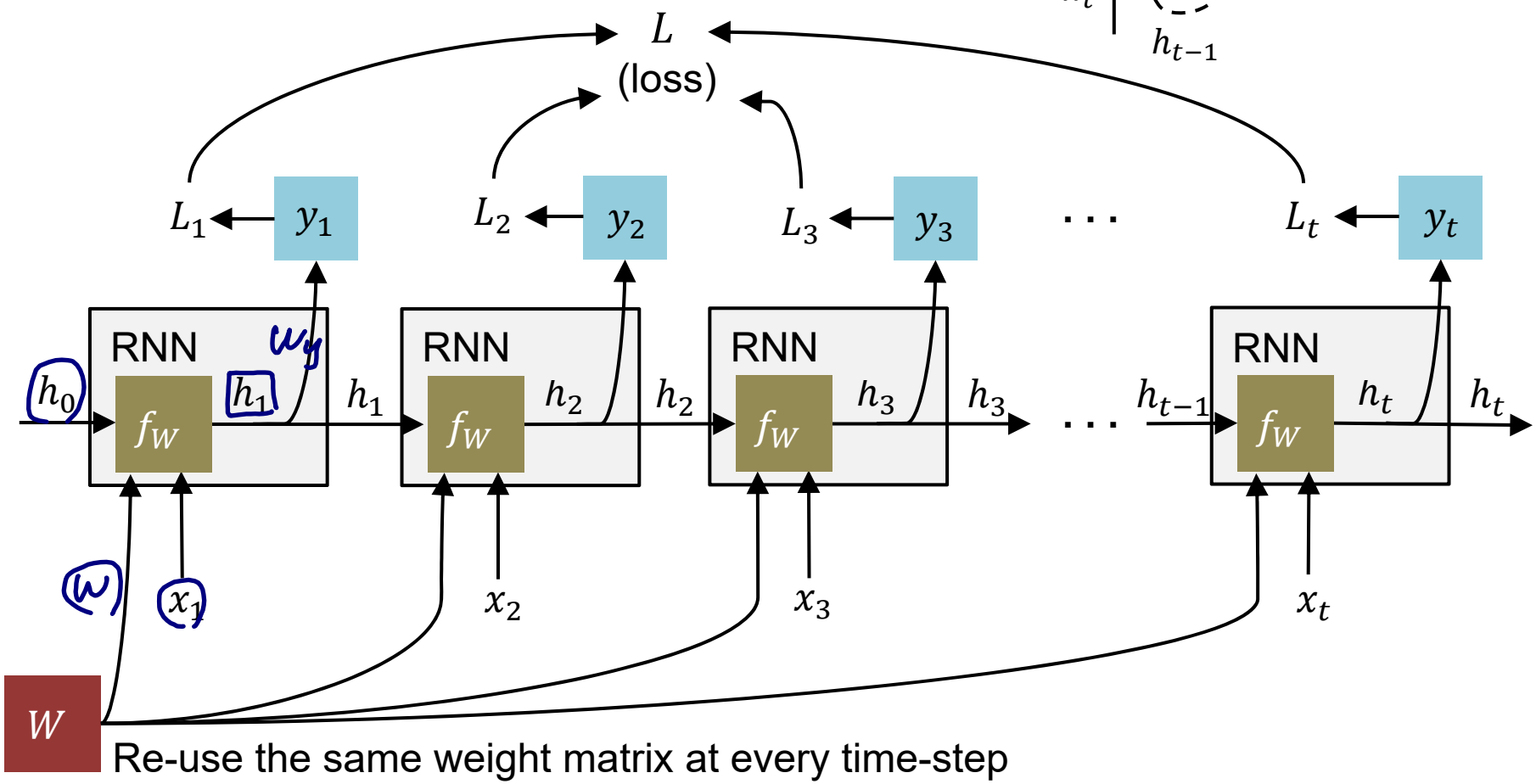
Re-use the same weight matrix at every time-step

Recurrent Neural Networks

$$h_t = f_W(h_{t-1}, x_t) \quad \begin{cases} h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t = \text{softmax}(W_{hy}h_t) \end{cases}$$



RNNs can be unrolled across multiple time steps:

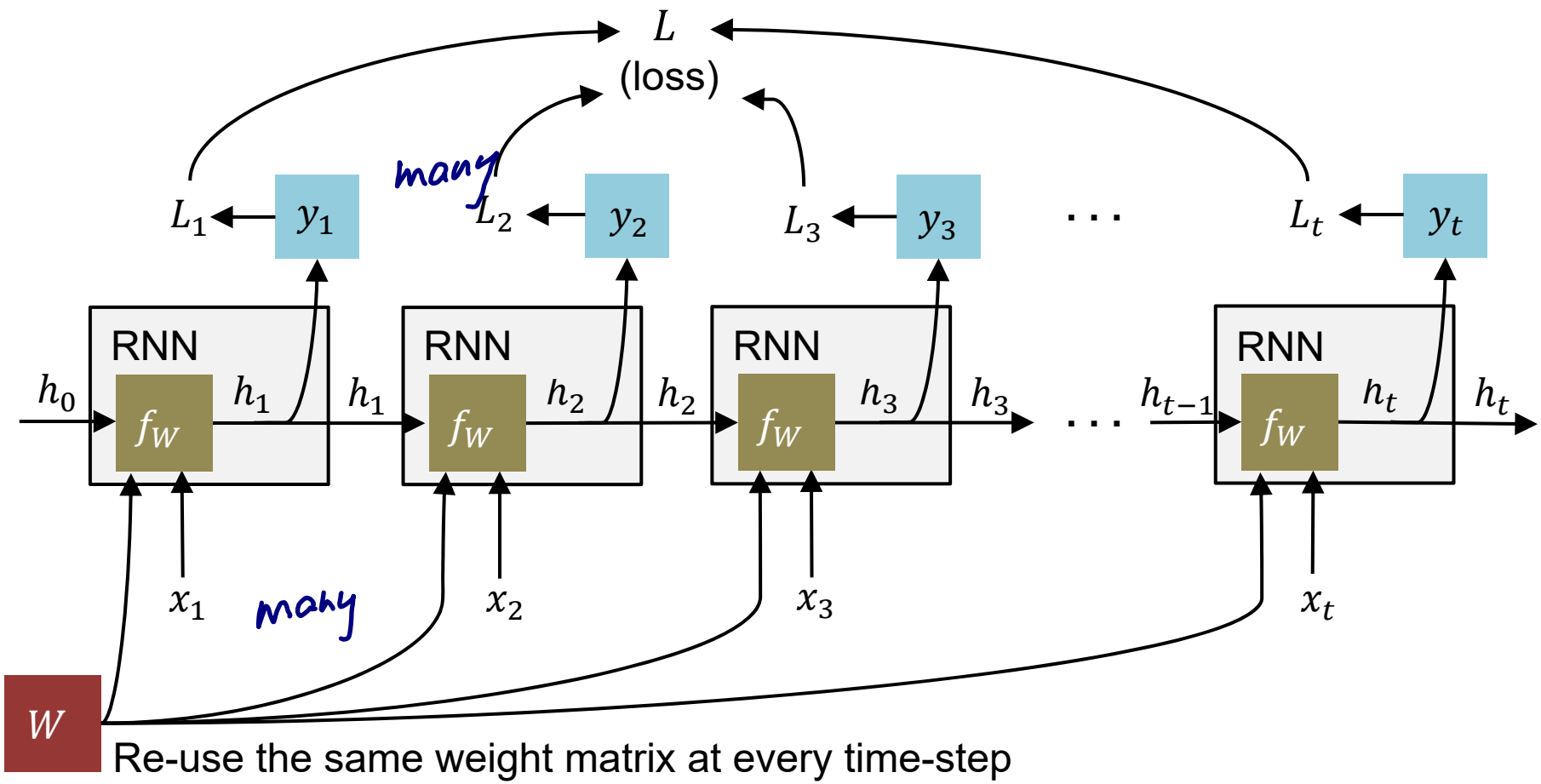


Re-use the same weight matrix at every time-step

Recurrent Neural Networks

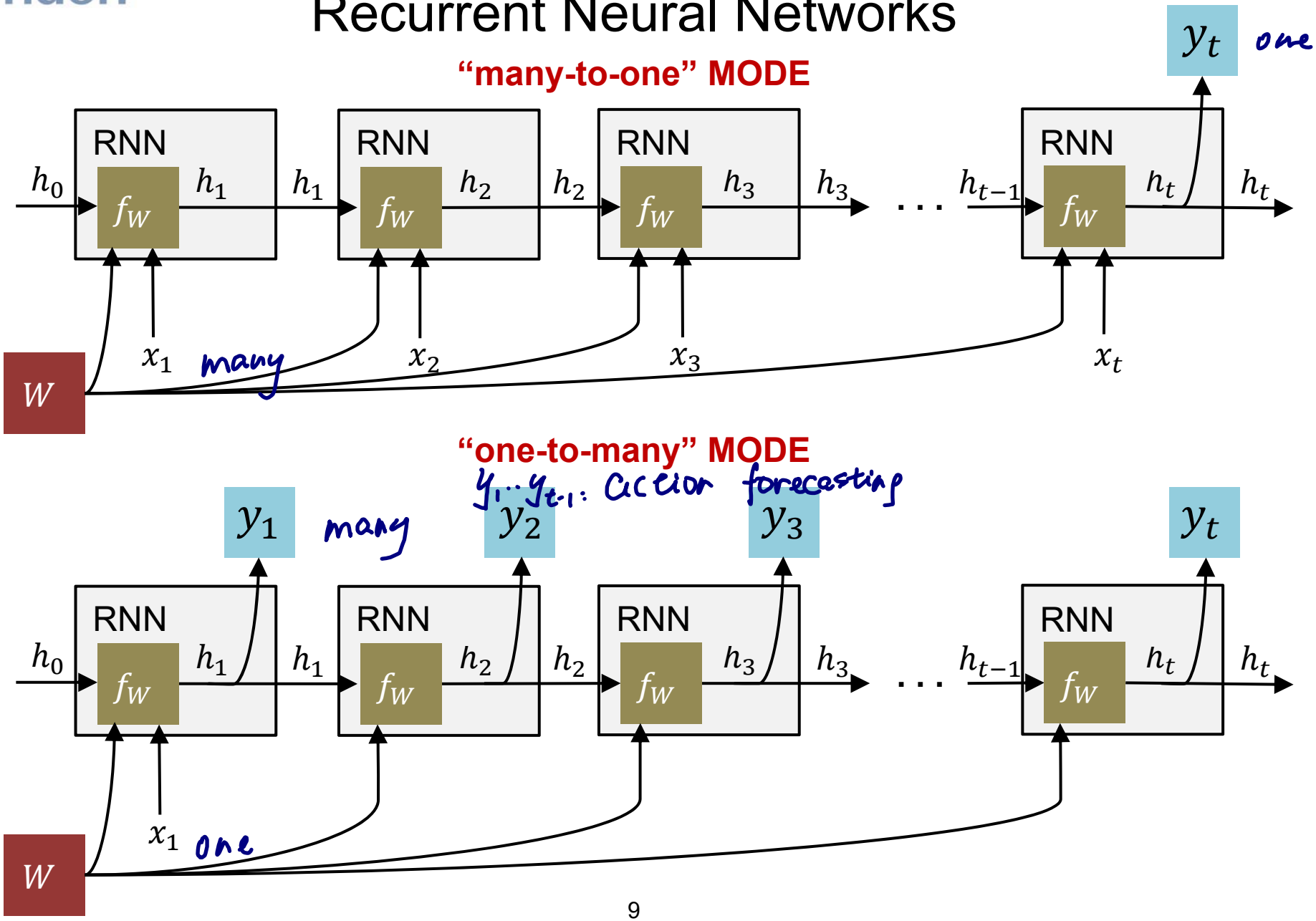
According to the input sequence and output, this is named as:

"many-to-many" MODE



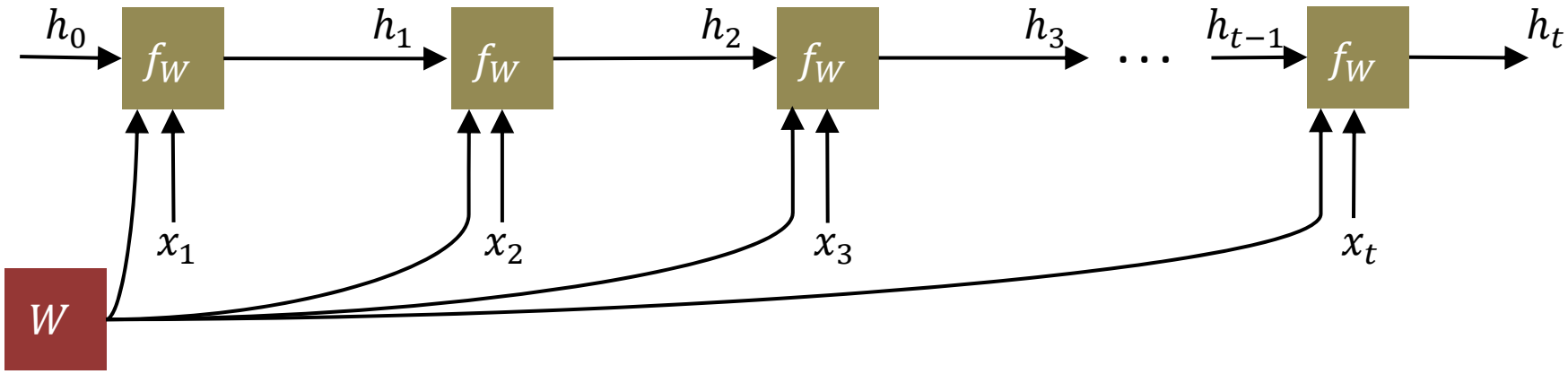
Re-use the same weight matrix at every time-step

Recurrent Neural Networks



Same Weights at Every Time Step: WHY?

Why do we use the same weight matrix at every time step? That is, why do we share parameters across different parts of a model?



Parameter sharing makes it possible to extend and apply the model to examples of different forms (length in our case) and generalize across them.

If we had separate parameters for each data of the time index:

- We could NOT generalize to sequence lengths not seen during training.
- We also could NOT share statistical strength across different sequence lengths and across different positions in time.

BackPropogation Through Time (BPTT)

Recap the eqs. of our RNN: $h_t = f_W(h_{t-1}, x_t)$ $\begin{cases} h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t = \text{softmax}(W_{hy}h_t) \end{cases}$

,and we switch W_{hh} with U, and W_{xh} with W, and W_{hy} with V:

$h_t = f_W(h_{t-1}, x_t)$ $\begin{cases} h_t = \tanh(Uh_{t-1} + Wx_t) \\ y_t = \text{softmax}(Vh_t) \end{cases}$

We also define our loss (cross entropy):

$$L_t(\bar{y}_t, y_t) = -\bar{y}_t \log y_t$$
$$L(\bar{y}, y) = \sum_t L_t(\bar{y}_t, y_t)$$
$$L(\bar{y}, y) = -\sum_t \bar{y}_t \log y_t$$

where,
$\bar{y}_t$ is the gt data (word) at time step t, and y_t is our prediction.

We typically treat the full sequence (sentence) as one training example, so the total error is just the sum of the errors at each time step (word).

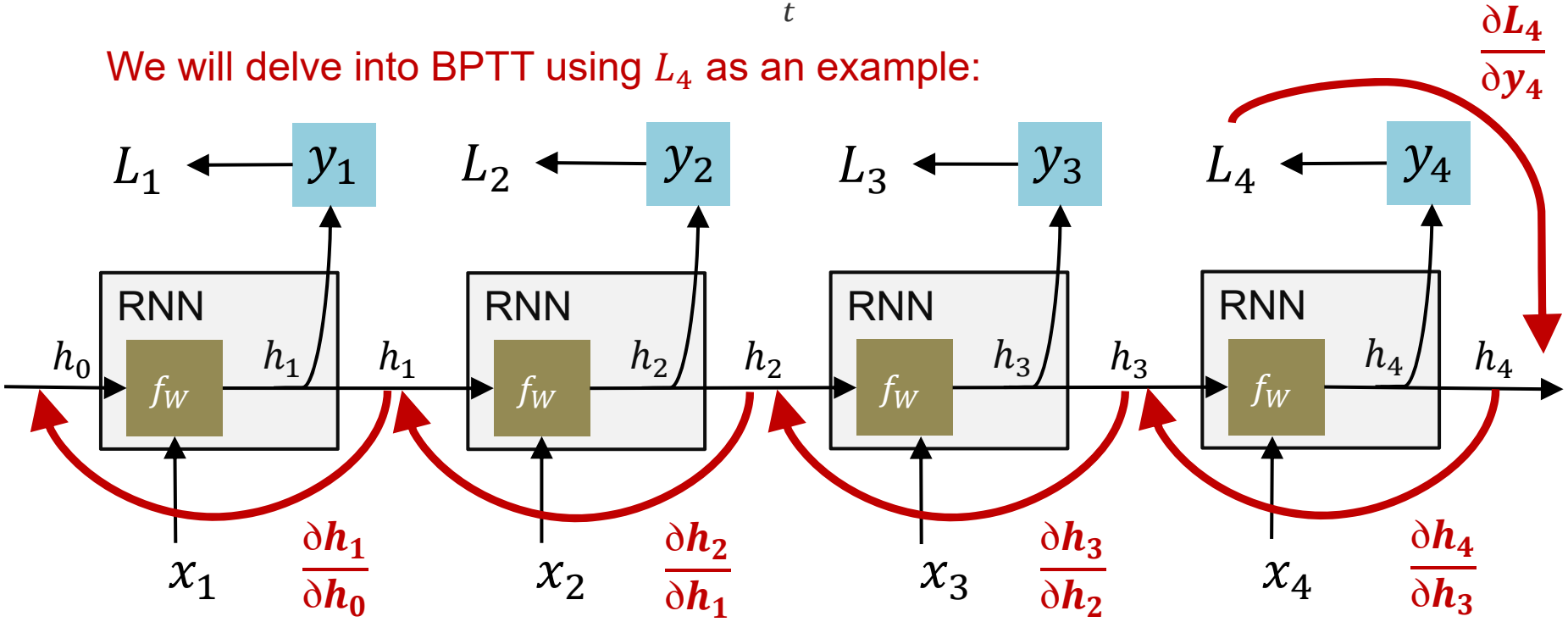
BackPropogation Through Time (BPTT)

- Our goal is to calculate the gradients of the loss (error) wrt. our parameters U, V, and W, and then learn good parameters using SGD.

- Just like we sum up the losses (errors), we also sum up the gradients at each time step for one training example:

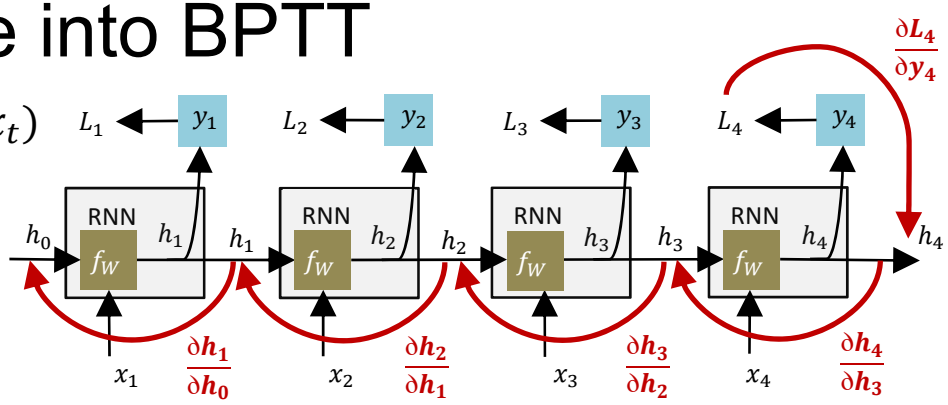
$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial W}$$

We will delve into BPTT using L_4 as an example:



Delve into BPTT

$$h_t = f_W(h_{t-1}, x_t) \begin{cases} h_t = \tanh(Uh_{t-1} + Wx_t) \\ y_t = \text{softmax}(Vh_t) \end{cases}$$



Let' s first calculate $\frac{\partial L_4}{\partial V}$: $V = W_{hy}$

$$\frac{\partial L_4}{\partial V} = \frac{\partial L_4}{\partial y_4}\frac{\partial y_4}{\partial V} = \frac{\partial L_4}{\partial y_4}\frac{\partial y_4}{\partial (Vh_4)}\frac{\partial (Vh_4)}{\partial V} = \ldots = (\bar{y}_4 - y_4) \otimes h_4$$

vector outer prod.

- $\frac{\partial L_4}{\partial V}$ only depends on the values at the current time step $\bar{y}_4, y_4, h_4$.
- As long as we have these, the gradient for V is a simple matrix multiplication.

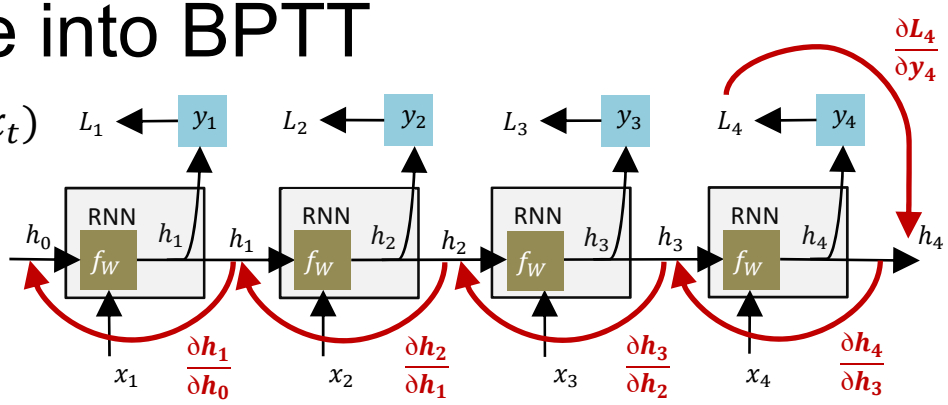
But the story is different for $\frac{\partial L_4}{\partial W}$ (and for $\frac{\partial L_4}{\partial U}$):

$$\frac{\partial L_4}{\partial W} = \frac{\partial L_4}{\partial y_4}\frac{\partial y_4}{\partial h_4}\frac{\partial h_4}{\partial W}$$ (continued, next slide…)

Delve into BPTT

$$h_t = f_W(h_{t-1}, x_t) \begin{cases} h_t = \tanh(Uh_{t-1} + Wx_t) \\ y_t = \text{softmax}(Vh_t) \end{cases}$$

$$\frac{\partial L_4}{\partial W} = \frac{\partial L_4}{\partial y_4}\frac{\partial y_4}{\partial h_4}\frac{\partial h_4}{\partial W}$$



- Note that $h_4 = \tanh(Wh_3 + Ux_4)$ depends on h_3, which depends on W and h_2, and so on.
- So when we take the derivative with respect to W, we can't simply treat h_3 as a constant! We need to apply the chain rule again, and what we really have is this:

$$\frac{\partial L_4}{\partial W} = \sum_{t=0}^{4} \frac{\partial L_4}{\partial y_4}\frac{\partial y_4}{\partial h_4}\frac{\partial h_4}{\partial h_t}\frac{\partial h_t}{\partial W}.$$

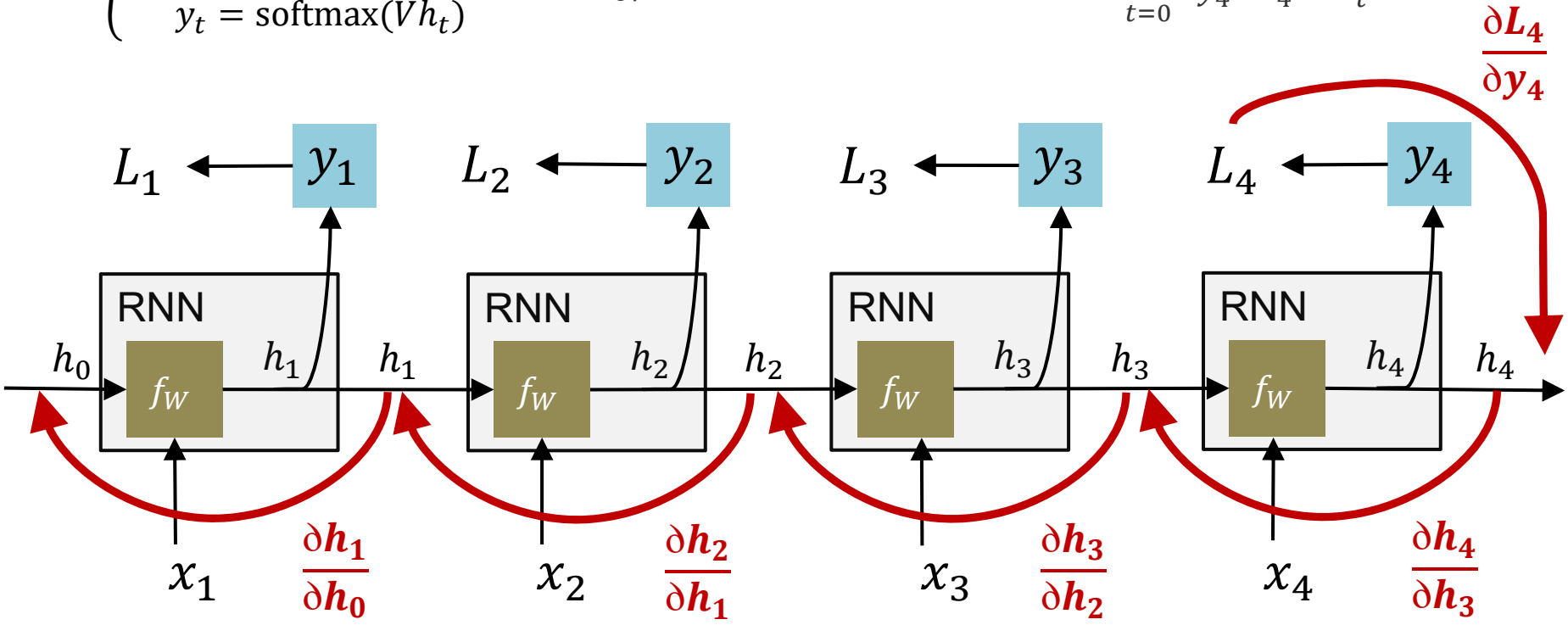
- We sum up the contributions of each time step to the gradient.
- In other words, since W is used in every step up to the output we care about, we need to backpropagate gradients from $t = 4$ through the network all the way to $t = 0$.

BPTT: Wrap up

$$\begin{cases} h_t = \tanh(Uh_{t-1} + Wx_t) \\ y_t = \text{softmax}(Vh_t) \end{cases}$$

$$\frac{\partial L_4}{\partial V} = (\bar{y}_4 - y_4) \otimes h_4$$

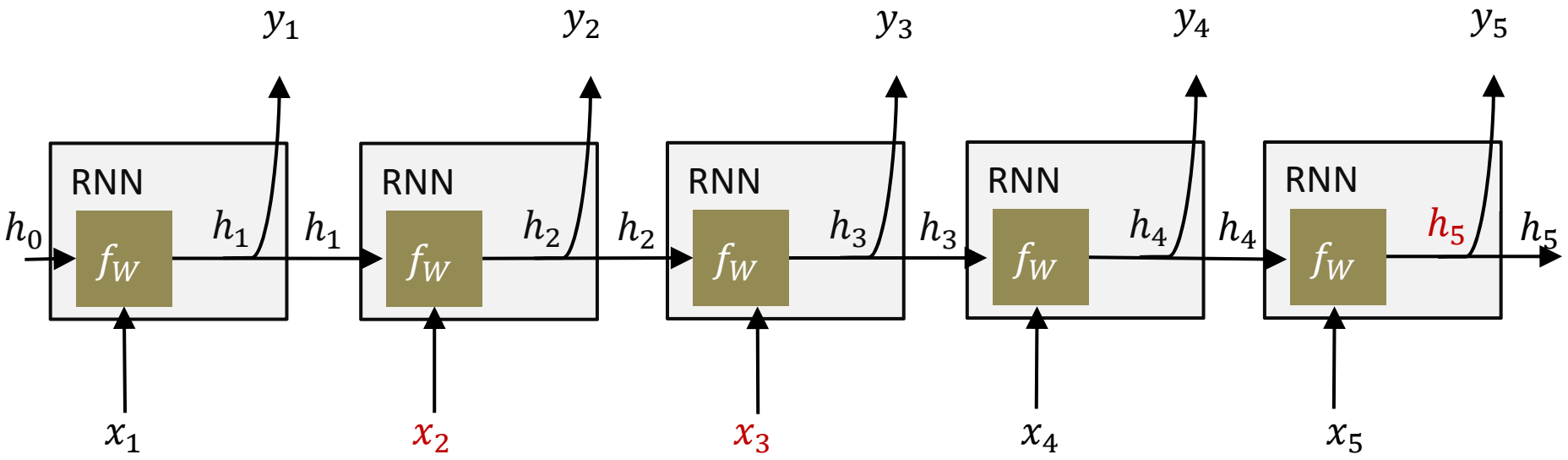
$$\frac{\partial L_4}{\partial W} = \sum_{t=0}^{4} \frac{\partial L_4}{\partial y_4} \frac{\partial y_4}{\partial h_4} \frac{\partial h_4}{\partial h_t} \frac{\partial h_t}{\partial W}$$



This is exactly the same as the standard backpropagation algorithm that we use in deep Feedforward Neural Networks. The key difference is that we sum up the gradients for W at each time step. In a traditional NN we don't share parameters across layers, so we don't need to sum anything.

Recurrent Neural Networks

An equivalent framing of RNNs is to think of a string of text as streaming in over time. Regardless of how many words I have seen in a given document, I want to make as good an estimate as possible about whatever outcome is of interest at that moment.

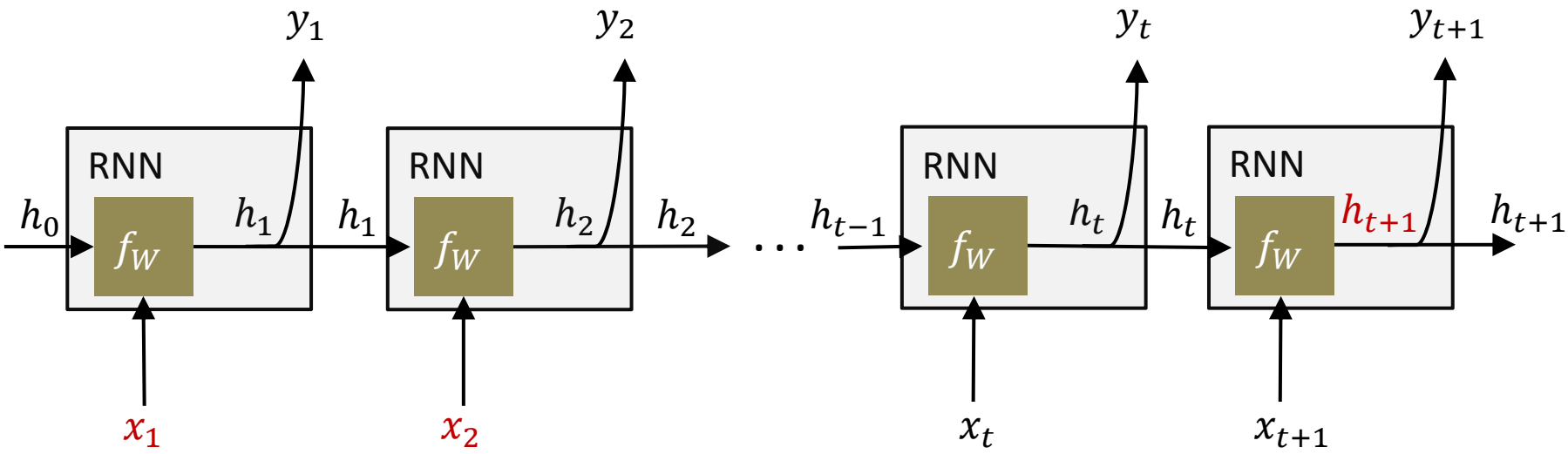


Because of the state in the model, words that occur early in the sequence can still have an influence on later outputs.

For instance; the effect of x_2 and x_3 on h_5.

Recurrent Neural Networks

However, using a basic dense layer as the RNN unit, makes it so that long range effects are hard to pass on.



For instance; the effect of x_1 and x_2 on h_{t+1} is hard to pass.

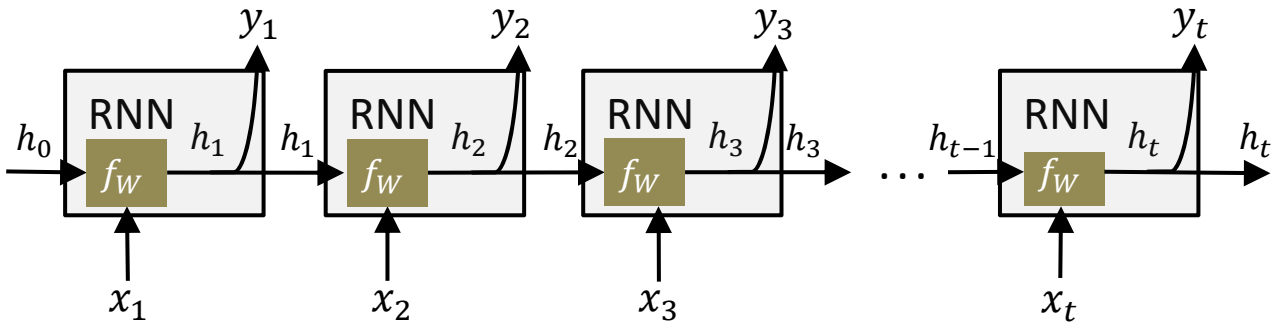
Long short-term memory was original proposed way back in 1997 in order to alleviate this problem.

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

Their specific idea that has had surprising staying power.

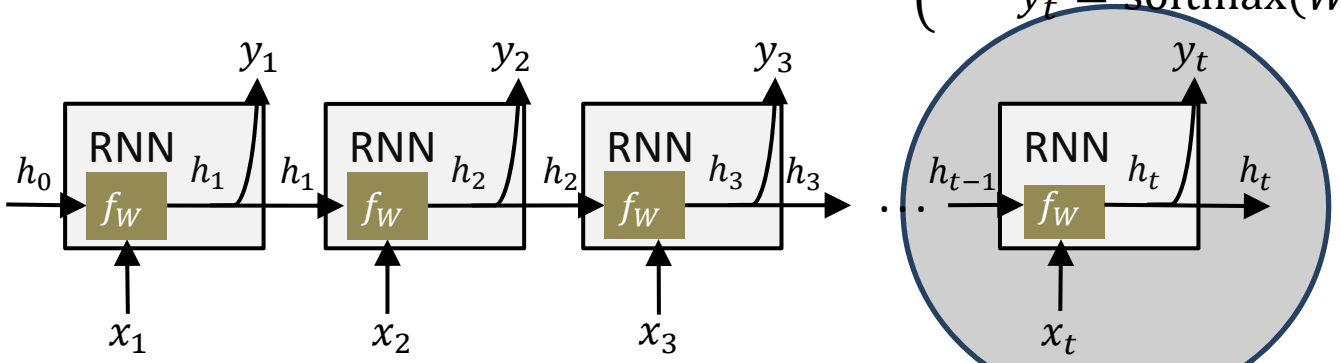
Long Short Term Memory (LSTM)

Reawake RNN representation: $h_t = f_W(h_{t-1}, x_t)$ $\begin{cases} h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t = \text{softmax}(W_{hy}h_t) \end{cases}$

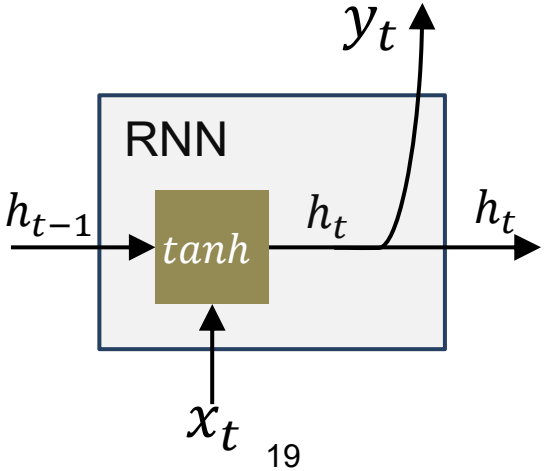


Long Short Term Memory (LSTM)

Reawake RNN representation: $h_t = f_W(h_{t-1}, x_t)$ $\begin{cases} h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t = \text{softmax}(W_{hy}h_t) \end{cases}$

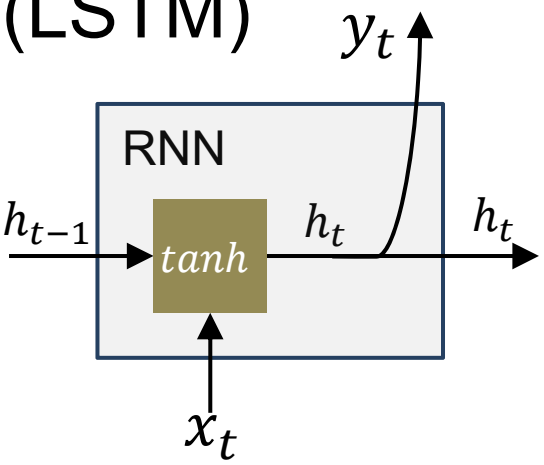


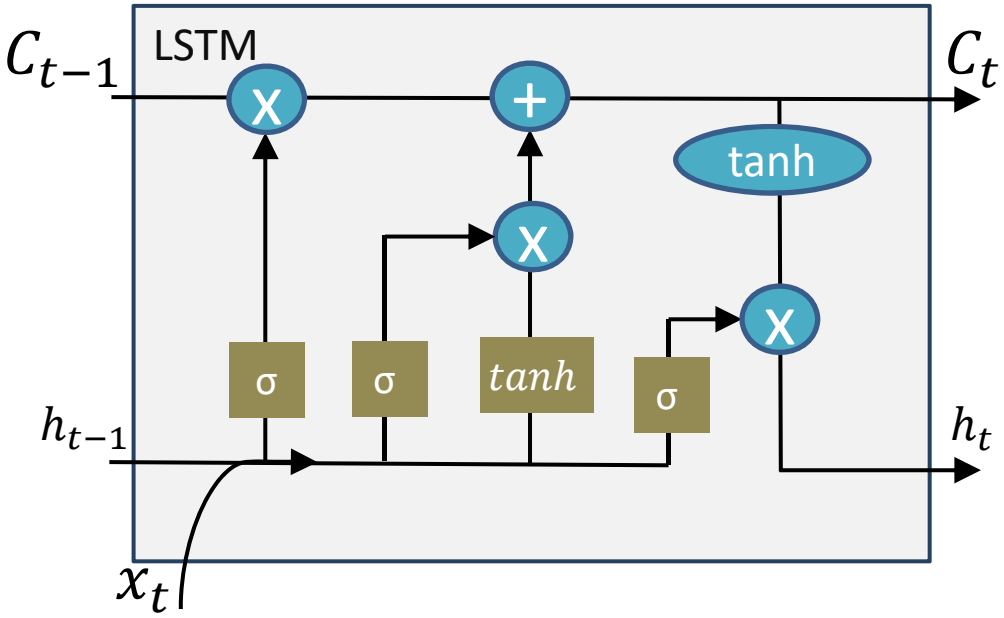
Take one layer (let's take the last one) and plug the formulas in:



Long Short Term Memory (LSTM)

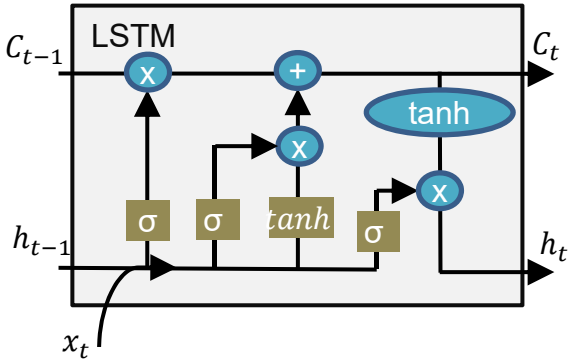
An LSTM layer is basically the same as a simple RNN layer, it just has two separate self-loops and several independent weight functions to serve slightly different purposes:





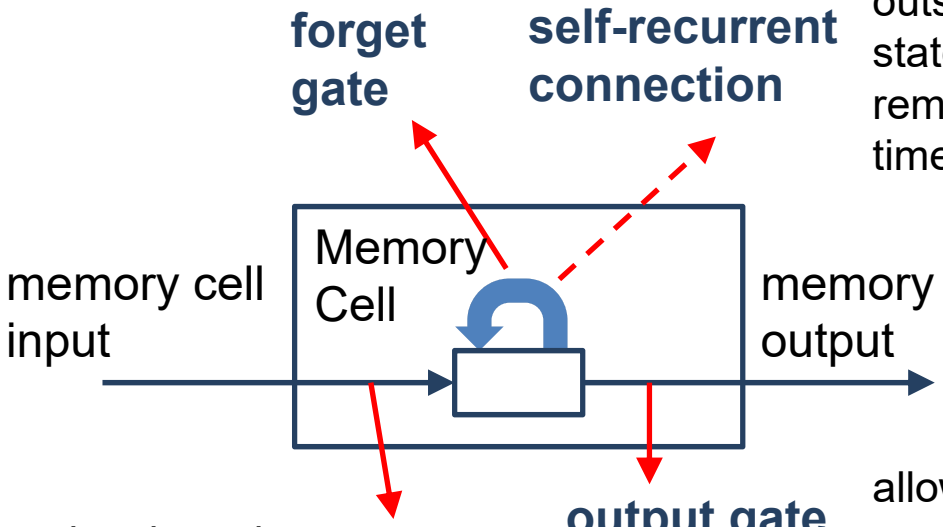
LSTM – Memory Cell

We will now delve into LSTMs designing a memory cell using logistics and linear units with multiplicative operations.



modulate the memory cell's self-recurrent connection, allowing the cell to remember or forget its previous state, as needed.

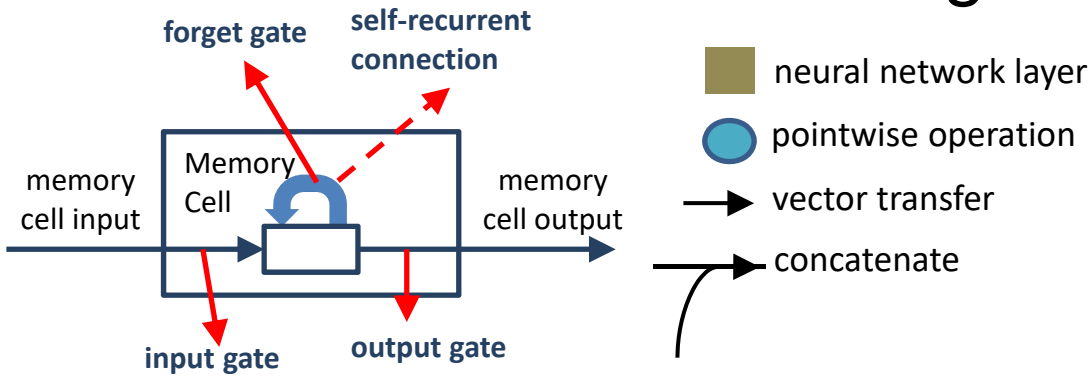
ensures that, barring any outside interference, the state of a memory cell can remain constant from one time step to another.

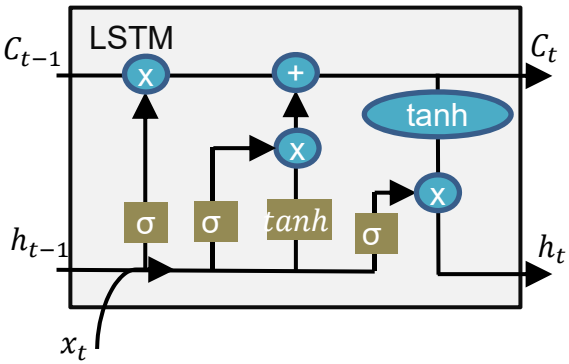


allow incoming signal to alter the state of the memory cell or block it.

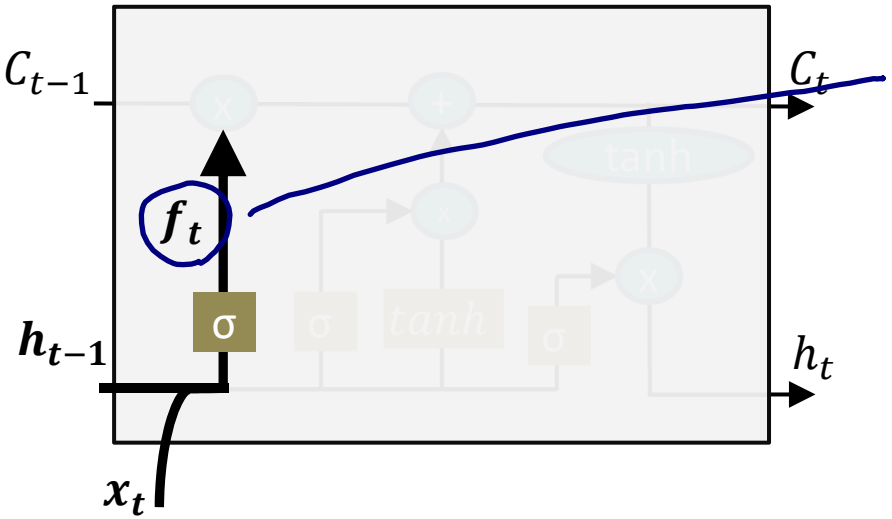
allow the state of the memory cell to have an effect on other neurons or prevent it.

LSTM – Forget Gate





LSTM: Forget Gate

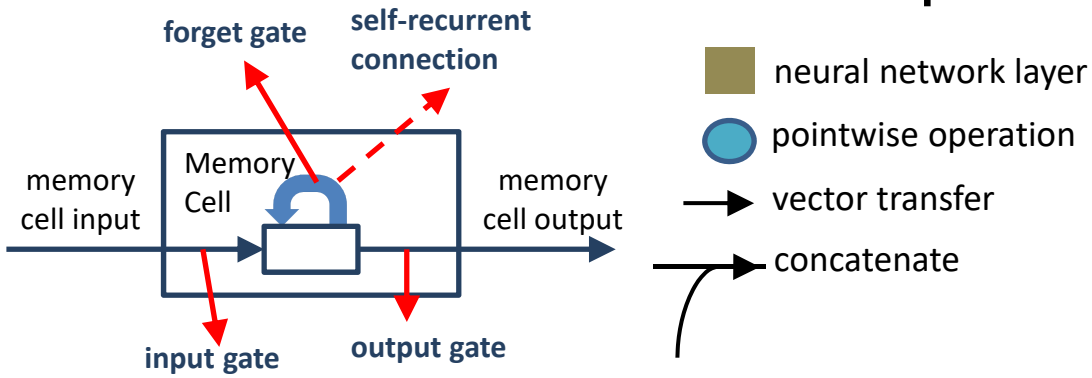


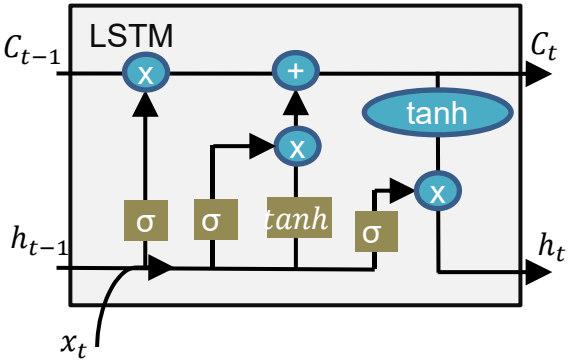
Forget gate layer looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.

A 1 represents "completely keep this" while a 0 represents "completely get rid of this".

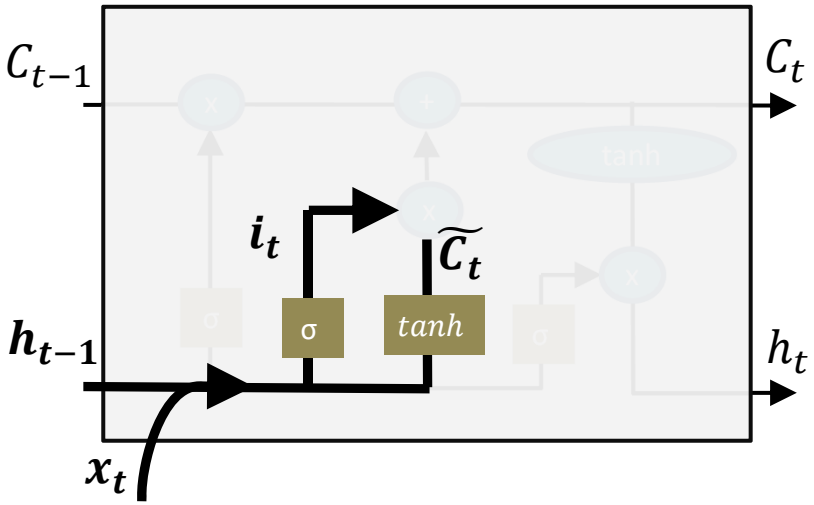
$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f)$$

LSTM – Input Gate





LSTM: Input Gate



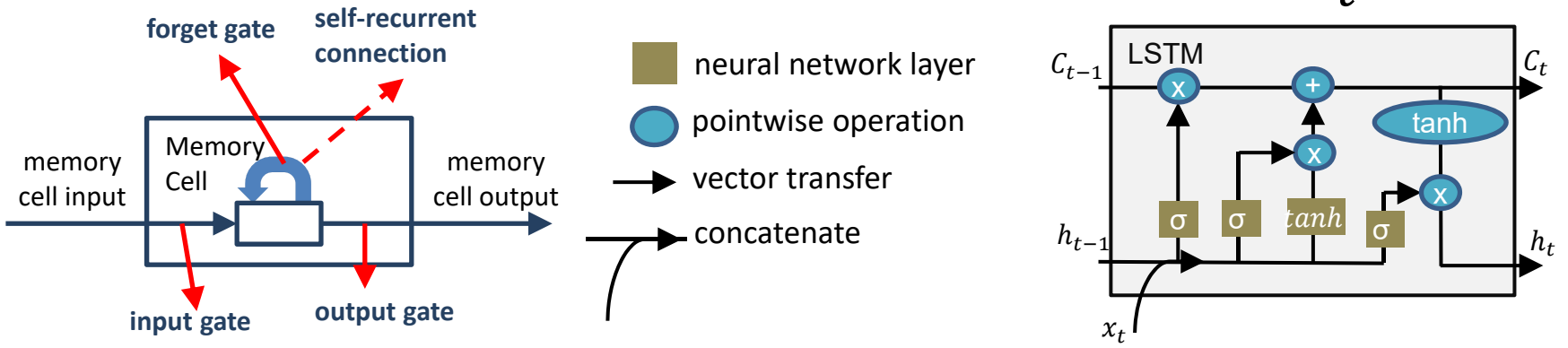
Input gate layer (sigmoid layer) decides which values we'll update.

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

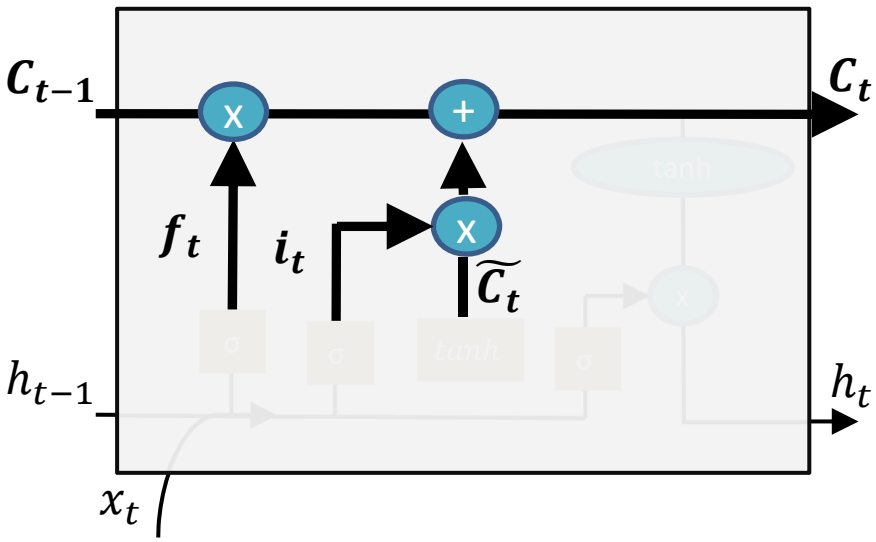
A **tanh layer** creates a vector of new candidate values, that could be added to the state.

$$\widetilde{C}_t = tanh(W_c . [h_{t-1}, x_t] + b_c)$$

LSTM – Calculate New State Value C_t



So far, we have determined f_t, i_t, and $\widetilde{C}_t$. We can now calculate C_t.

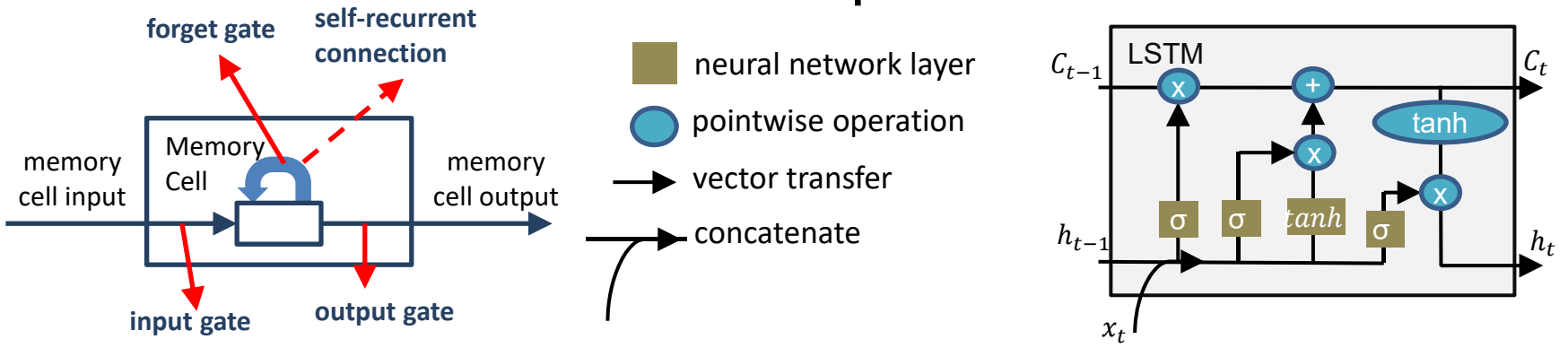


$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t$$

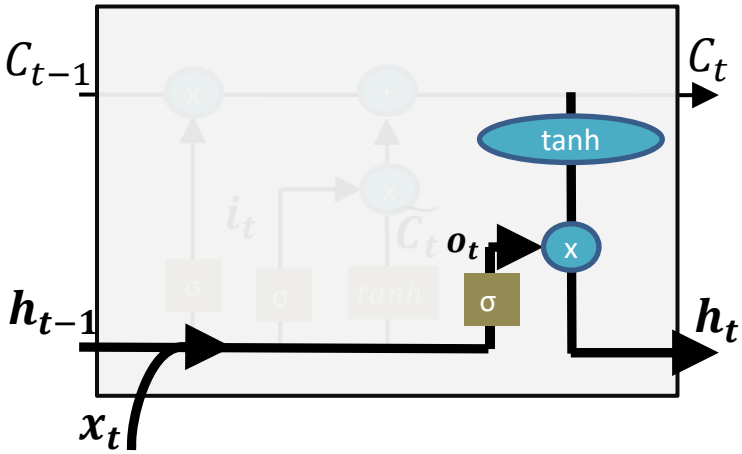
We multiply the old state f_t by forgetting the things we decided to forget earlier.

Then, we add the new candidate values, scaled by how much we decided to update each state value.

LSTM – Output Gate



LSTM: Output Gate



First, run a sigmoid layer which decides what parts of the cell state we're going to output:

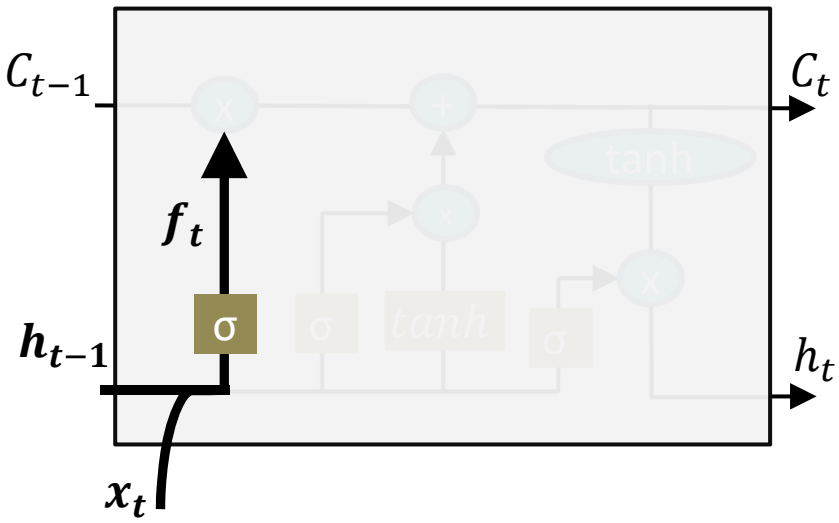
$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to:

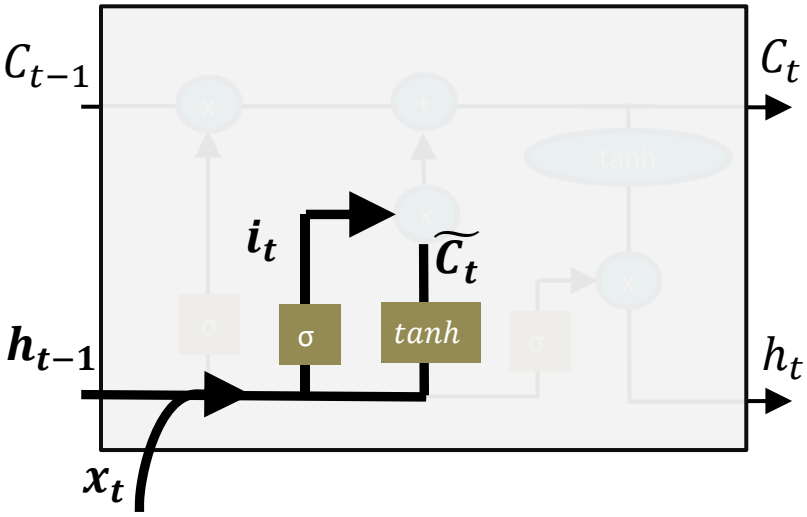
$$h_t = o_t * \tanh(C_t)$$

LSTM: Wrap Up

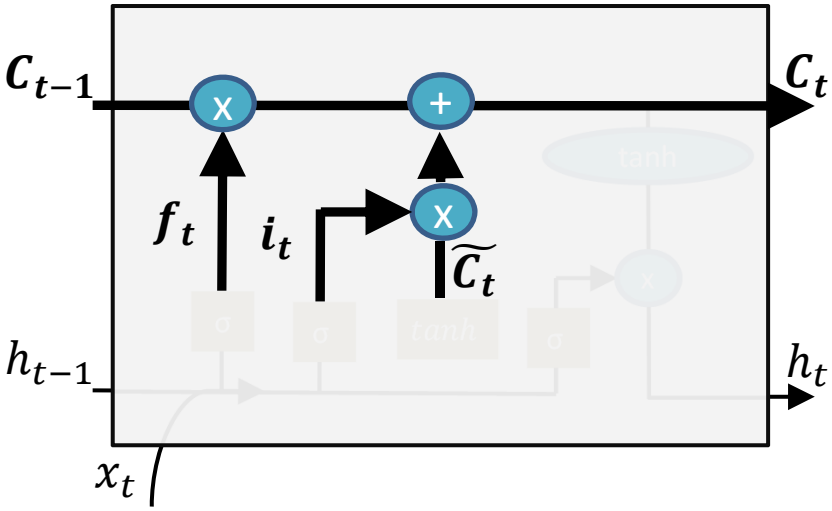
LSTM: Forget Gate



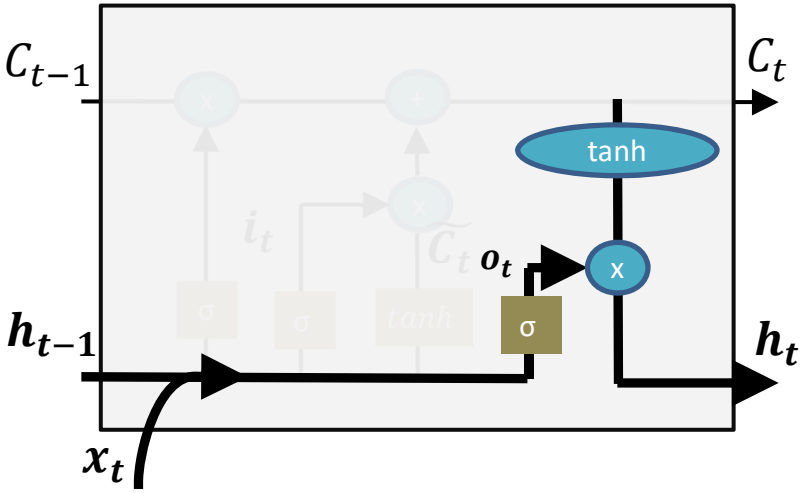
LSTM: Input Gate



Calculate C_t



LSTM: Output Gate



Hierarchical RNN for Action Rec.

Problem: Skeleton Based Action Recognition

- Human action analysis from skeletal data consists of two key aspects, namely pose and motion. While pose is the structural arrangement of key body joints, motion information relates to the temporal movements of the joints.

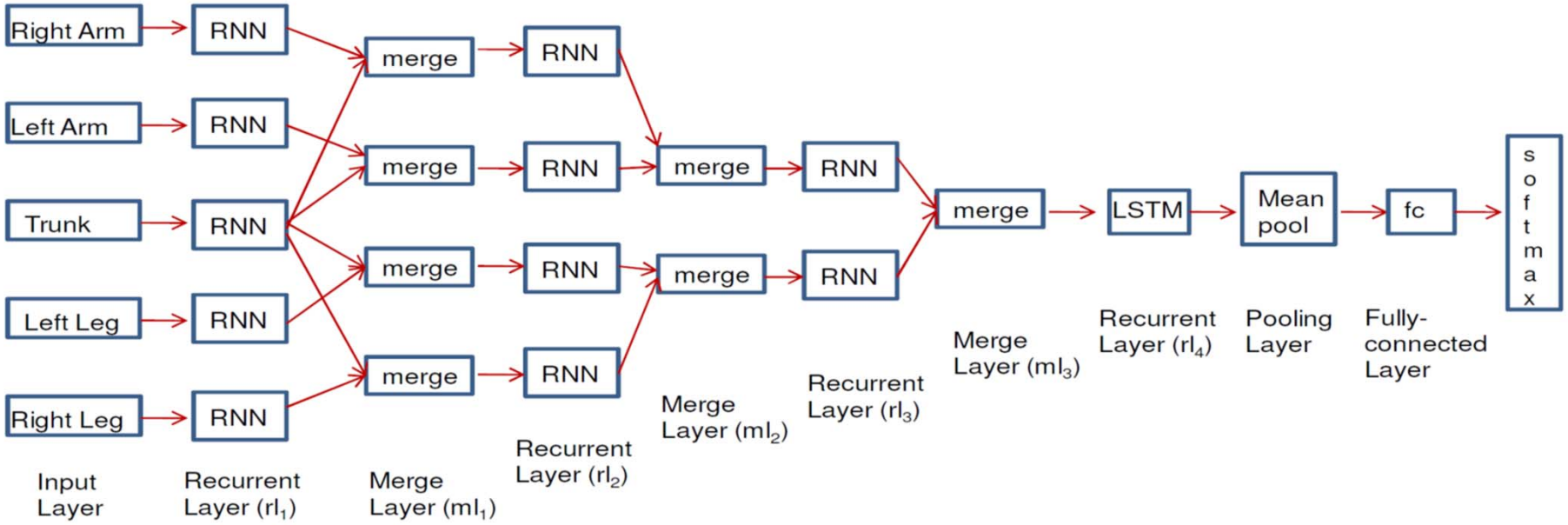
- The human body is composed of five main parts: right and left arms, right and left legs, trunk.

- The proposed method utilizes Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) to learn the temporal dependency between joints' positions.

- The proposed architecture uses a hierarchical scheme for aggregating the learned responses of various RNN units. The proposed network consists of ten layers:

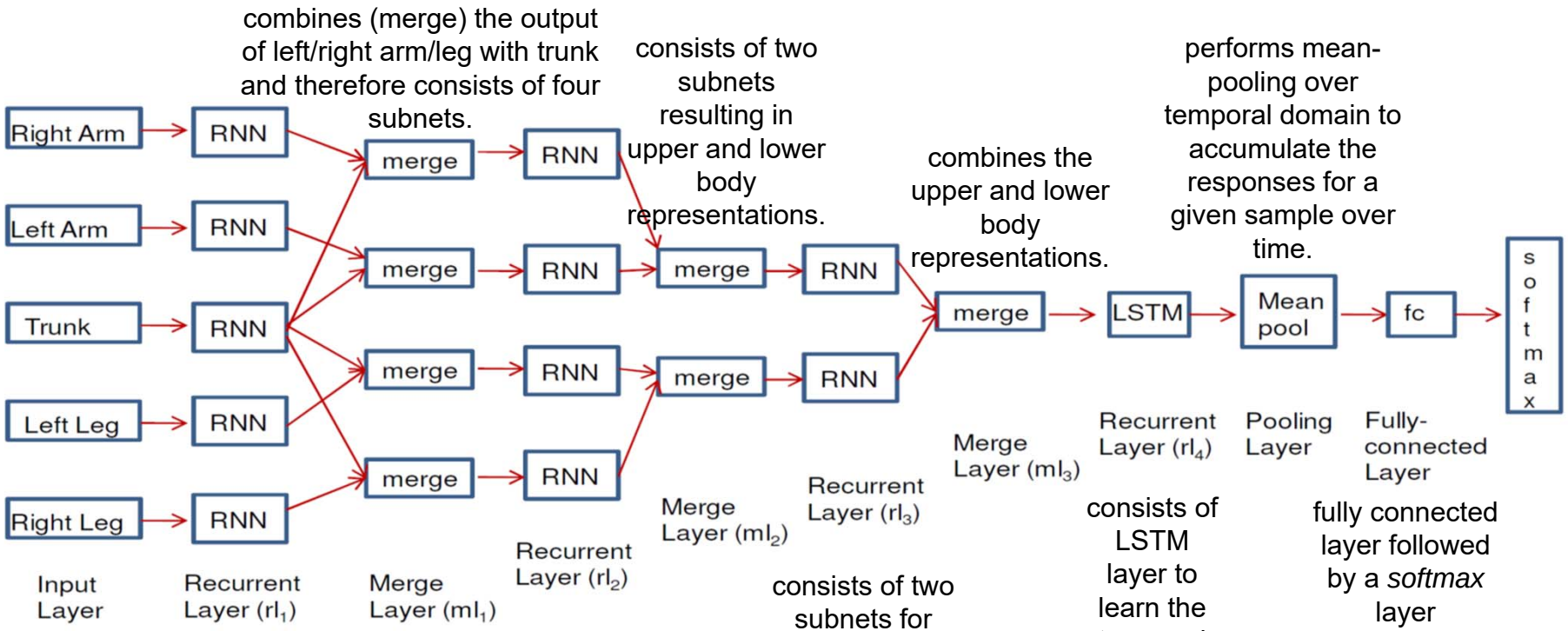
Hierarchical RNN for Action Rec.

Approach: Hierarchical RNN



Hierarchical RNN for Action Rec.

Approach: Hierarchical RNN



combines (merge) the output of left/right arm/leg with trunk and therefore consists of four subnets.

consists of two subnets resulting in upper and lower body representations.

combines the upper and lower body representations.

performs mean-pooling over temporal domain to accumulate the responses for a given sample over time.

consists of recurrent layer with five subnets corresponding to the five main body parts

consists of four subnets for capturing the temporal dependencies corresponding to the four subnets of ml_1

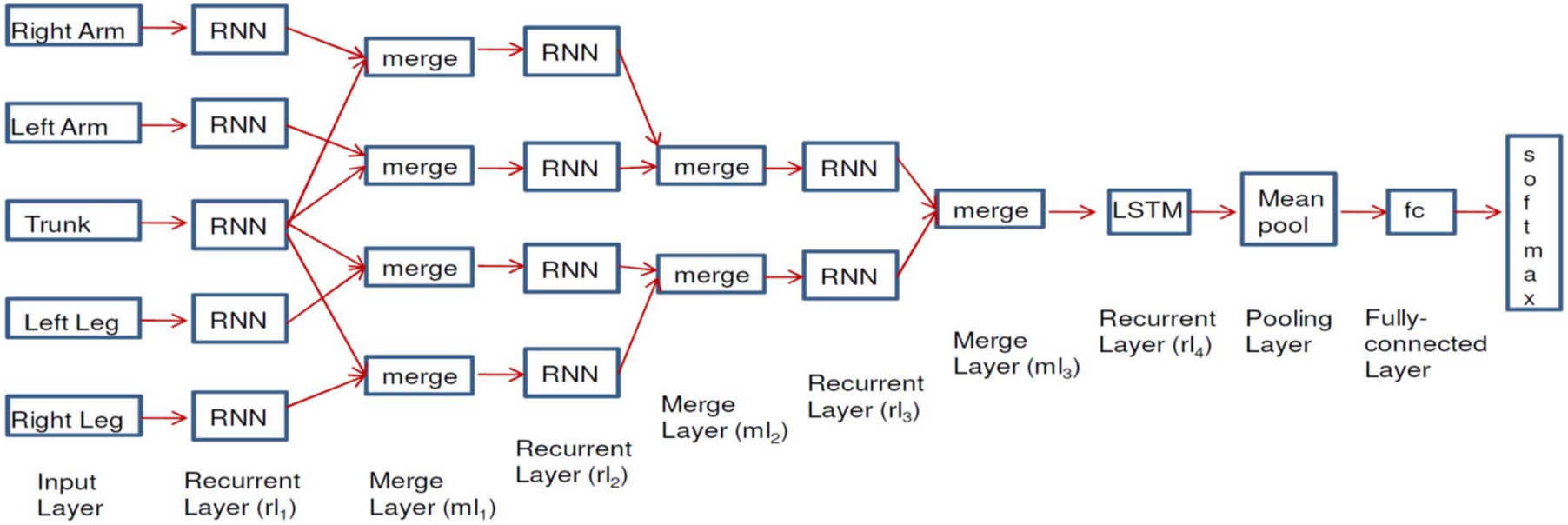
consists of two subnets for capturing the temporal dependencies of lower and upper body movements of ml_2.

consists of LSTM layer to learn the temporal behavior of full body.

fully connected layer followed by a *softmax* layer

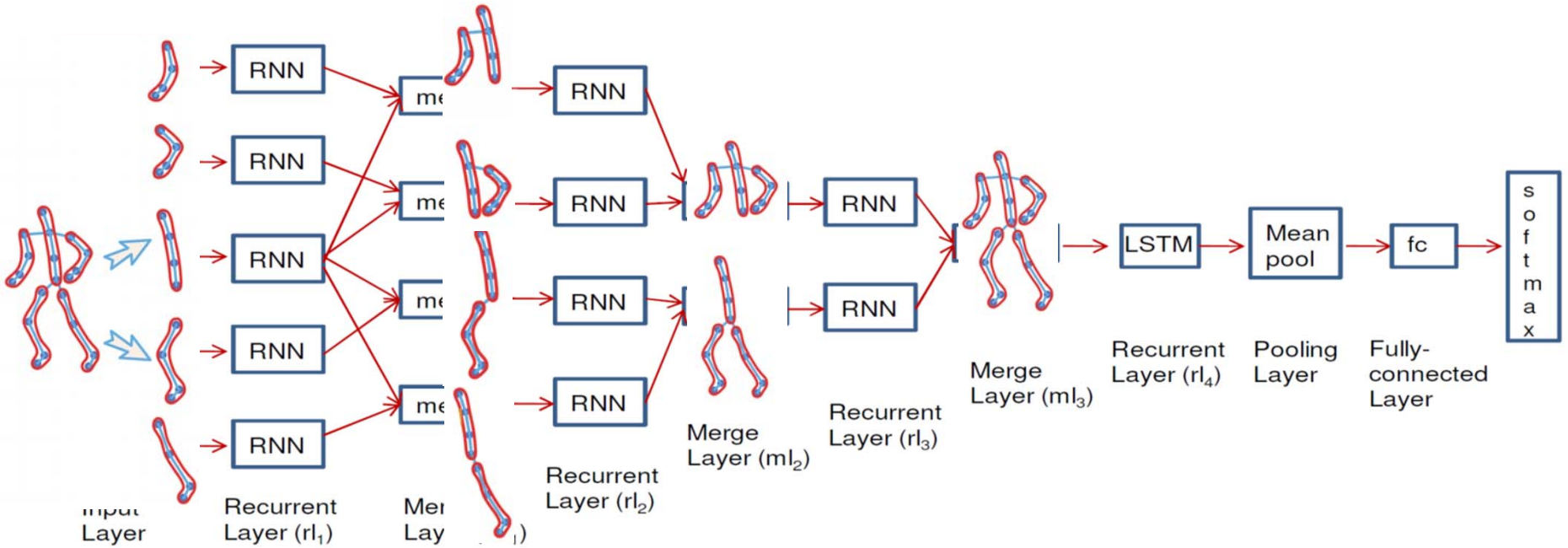
Hierarchical RNN for Action Rec.

Approach: Hierarchical RNN



Hierarchical RNN for Action Rec.

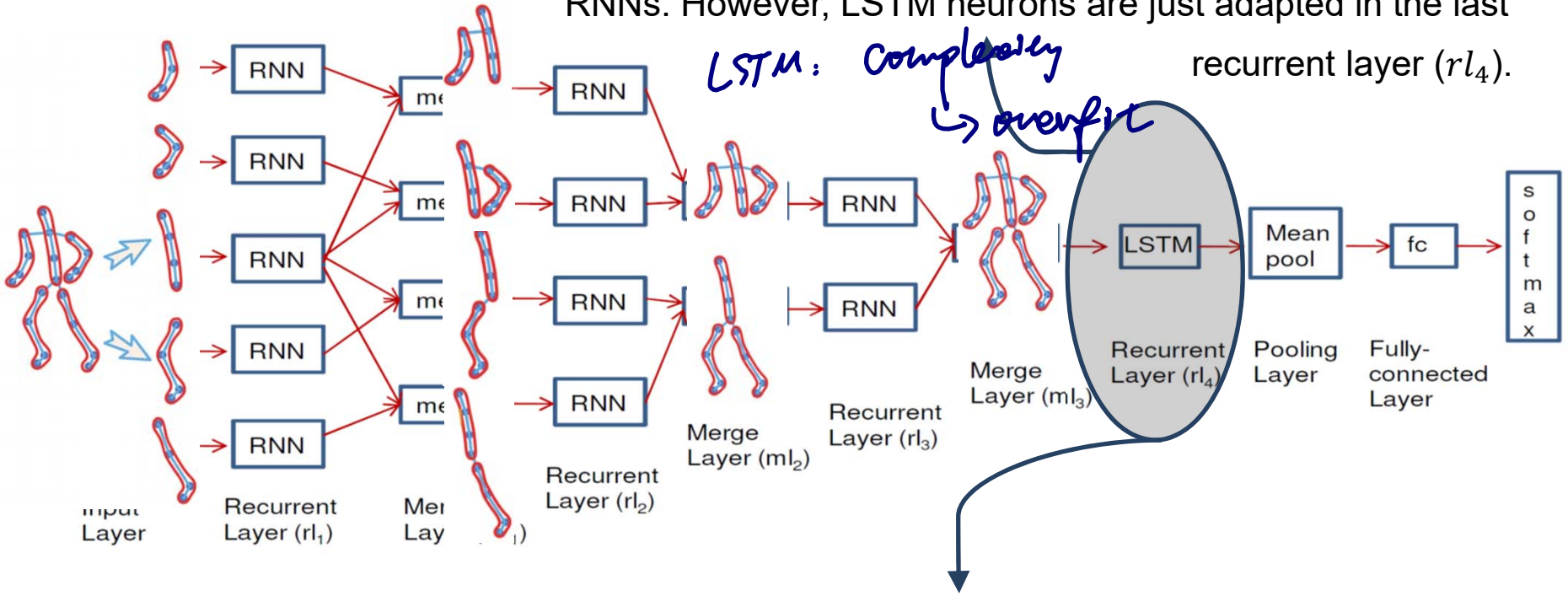
Approach: Hierarchical RNN



Hierarchical RNN for Action Rec.

Approach: Hierarchical RNN

LSTM architecture can effectively overcome the vanishing gradient problem while training RNNs. However, LSTM neurons are just adapted in the last recurrent layer (rl_4).



The first three RNN layers all use the $\tanh$ activation function. This is a trade-off between improving the representation ability and avoiding overfitting. Generally, the number of weights in a LSTM block is several times more than that in a $\tanh$ neuron. It is very easy to overfit the network with limited training sequences.

Hierarchical RNN for Action Rec.

Dataset: MSR Action3D

- This dataset contains twenty actions: high arm wave, horizontal arm wave, hammer, hand catch, forward punch, high throw, draw x, draw tick, draw circle, hand clap, two hand wave, side-boxing, bend, forward kick, side kick, jogging, tennis swing, tennis serve, golf swing, pickup and throw.
- Each action is performed by 7 subjects for 3 times.
- The subjects are facing the camera during the performance.
- The depth maps are captured at about 15 frames per second by a depth camera that acquires the depth through structure infra-red light.
- Each frame in a sequence contains 20 skeleton joints.
- All together, the dataset has 23797 frames of depth maps for the 4020 action samples.

Hierarchical RNN for Action Rec.

Dataset: MSR Action3D

Tennis serve

Draw tick

Hierarchical RNN for Action Rec.

Dataset: Berkeley MHAD (Multi-modal Human Action Db.)

This dataset contains;
- 11 actions,
- each of which is performed by 7 male and 5 female subjects,
- in the range 23-30 years of age except for one elderly subject.

All subjects;
- performed 5 repetitions of each action,
- yielding about 660 action sequences,
- which correspond to about 82 minutes of total recording time.

A T-pose is recorded for each subject which can be used for;
- the skeleton extraction,
- the background data (with and without the chair used in some of the activities)

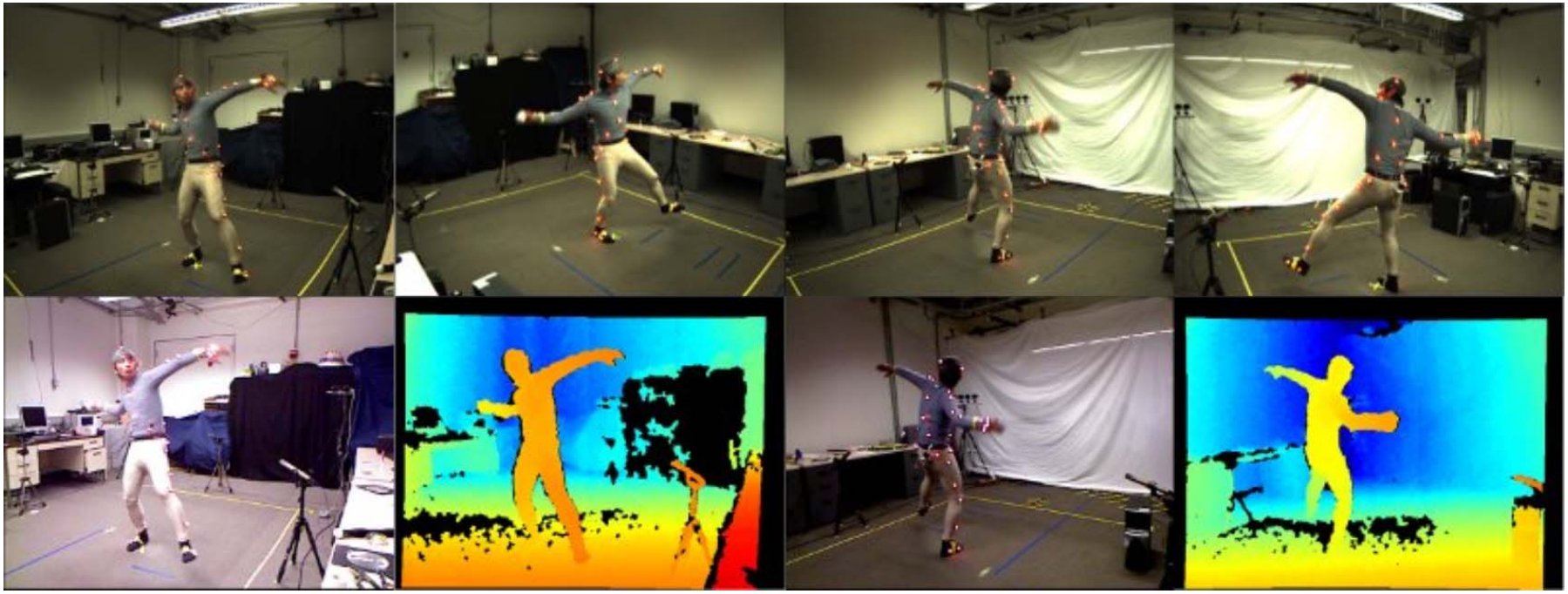
Hierarchical RNN for Action Rec.

Dataset: Berkeley MHAD (Multi-modal Human Action Db.)

- All actions (coupled with corresponding point clouds).
- Actions (from left to right): jumping, jumping jacks, bending, punching, waving two hands, waving one hand, clapping, throwing, sit down/stand up, sit down, stand up.

Hierarchical RNN for Action Rec.

Dataset: Berkeley MHAD (Multi-modal Human Action Db.)



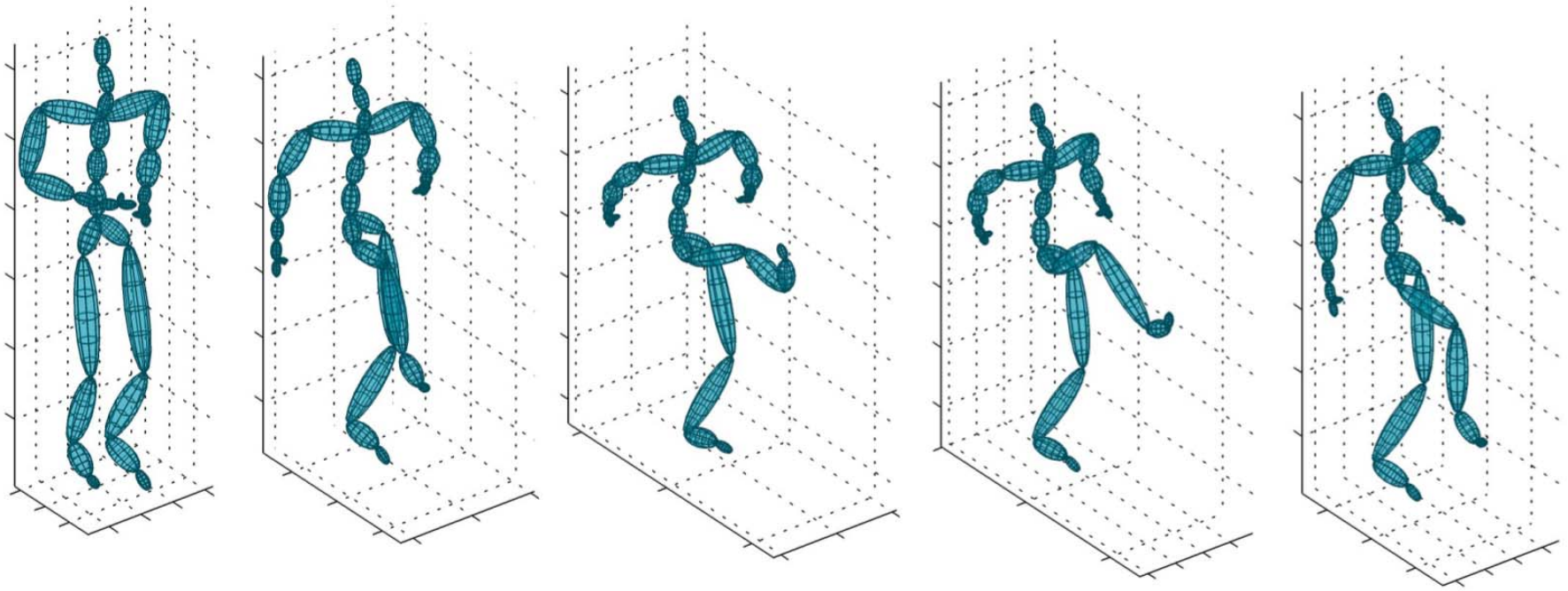
Throwing action is displayed from the reference camera in each camera cluster as well as from the two Kinect cameras.

Hierarchical RNN for Action Rec.

Dataset: Motion Capture Dataset HDM05

It is captured;

- by an optical marker-based technology with the frequency of 120 Hz,
- which contains 2337 sequences for 130 actions,
- performed by 5 non-professional actors, and 31 joints in each frame.



Kicking action

Hierarchical RNN for Action Rec.

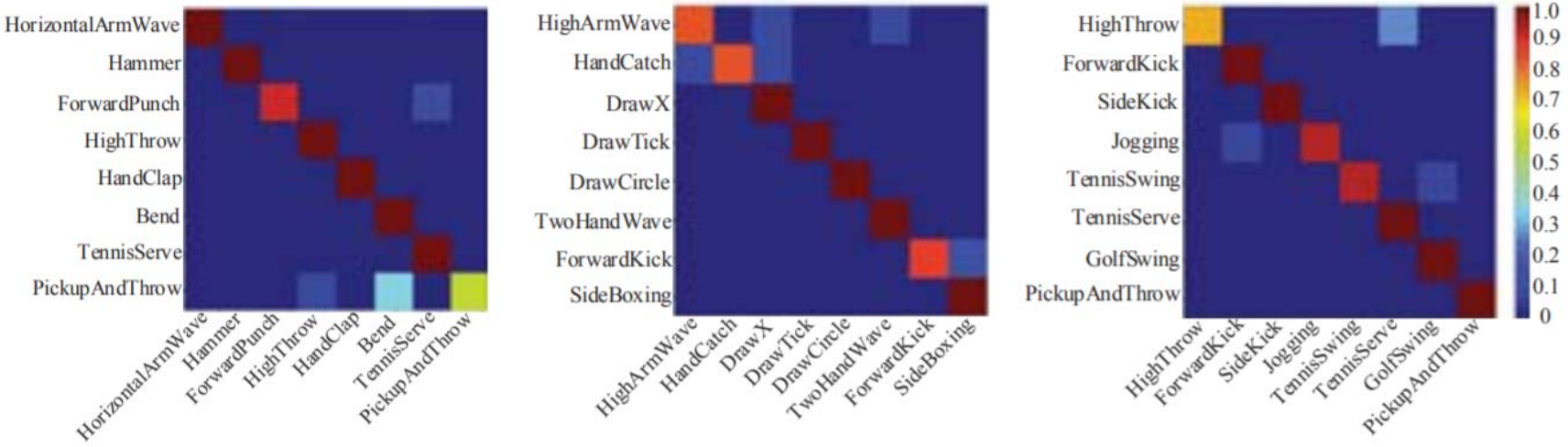
Results on MSR Action3D

Results on MSR Action3D

Method	AS1	AS2	AS3	Ave.
Li *et al.*, 2010	72.9	71.9	79.2	74.7
Chen *et al.*, 2013	**96.2**	83.2	92.0	90.47
Gowayyed *et al.*, 2013	92.39	90.18	91.43	91.26
Vemulapalli *et al.*, 2014	95.29	83.87	**98.22**	92.46
HBRNN-L	93.33	**94.64**	95.50	**94.49**

Action Set 1 (AS1)	Action Set 2 (AS2)	Action Set 3 (AS3)
Horizontal arm wave	High arm wave	High throw
Hammer	Hand catch	Forward kick
Forward punch	Draw x	Side kick
High throw	Draw tick	Jogging
Hand clap	Draw circle	Tennis swing
Bend	Two hand wave	Tennis serve
Tennis serve	Forward kick	Golf swing
Pickup & throw	Side boxing	Pickup & throw

Confusion matrices on MSR Action3D



Hierarchical RNN for Action Rec.

Results on Berkeley MHAD and HDM05

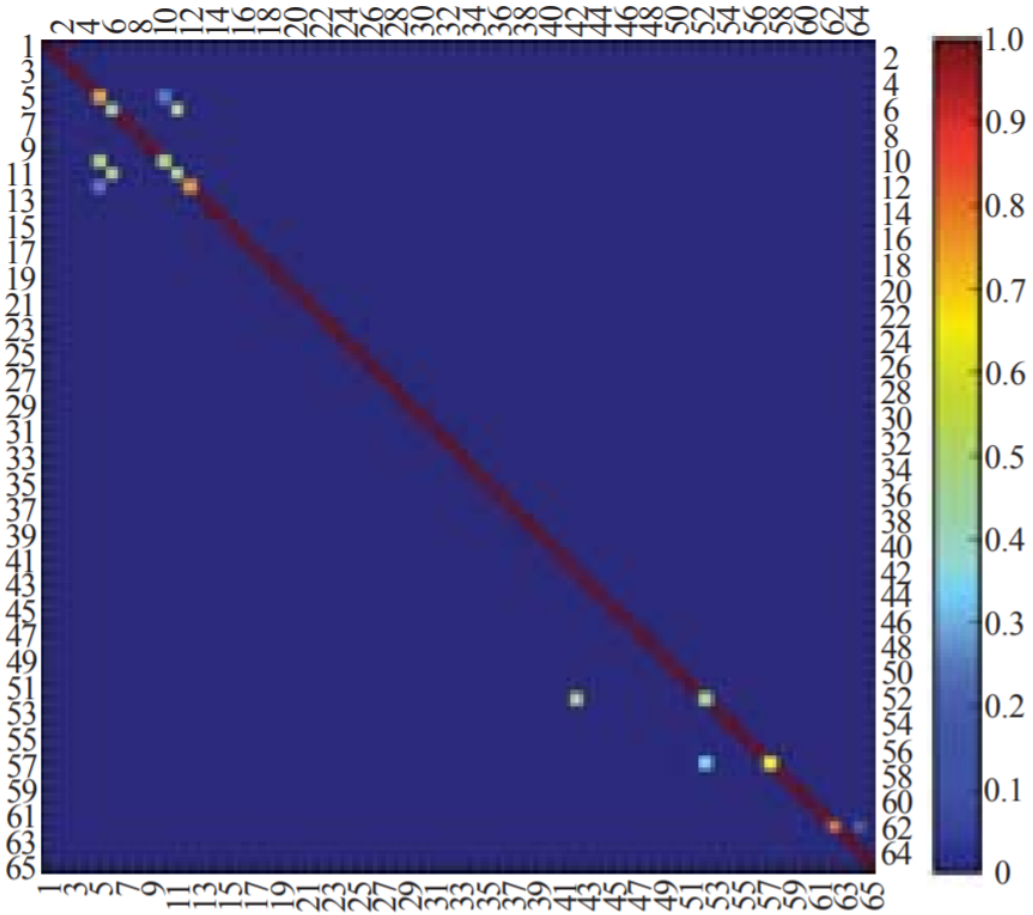
Results on Berkeley MHAD

Method	Acc.(%)
Ofli *et al.*, 2014	95.37
Vantigodi *et al.*, 2013	96.06
Vantigodi *et al.*, 2014	97.58
Kapsouras *et al.*, 2014	98.18
Chaudhry *et al.*, 2013	99.27
Chaudhry *et al.*, 2013	100
HBRNN-L	**100**

Results on HDM05

Method	Ave.(%)	Std.
Cho and Chen, 2013	95.59	0.76
HBRNN-L	**96.92**	**0.50**

Confusion matrix on HDM05 dataset



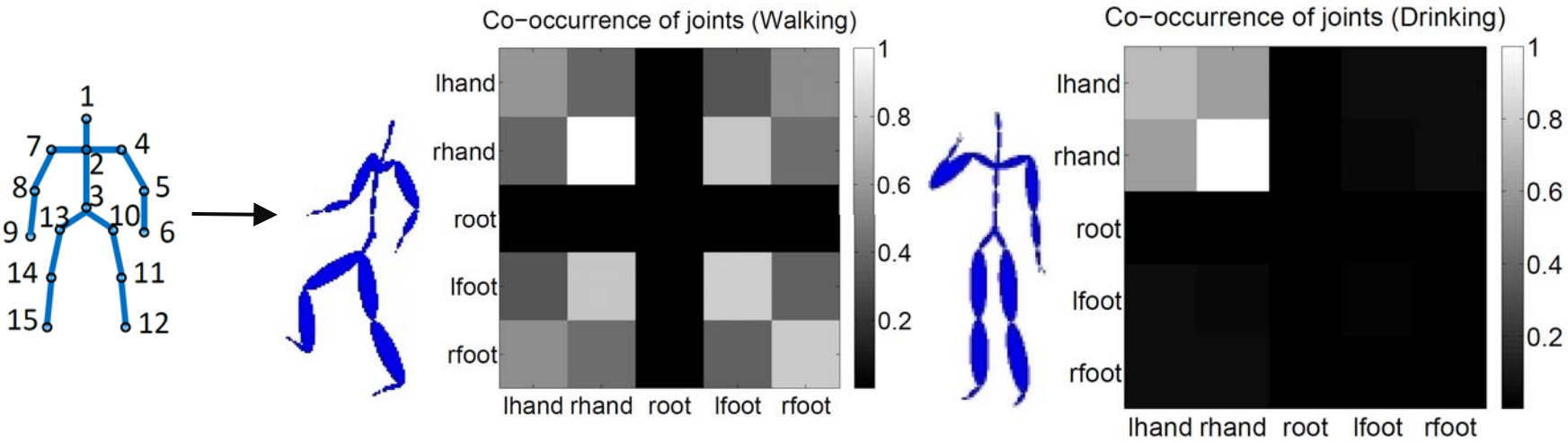
Co-occurrence Feature Learning for Action Rec.

Problem: Skeleton Based Action Recognition

- RNNs with LSTM can learn feature representations and model long-term temporal dependencies automatically.
- Considering this, an end-to-end fully connected deep LSTM network is engineered for skeleton based action recognition.
- Co-occurrences of joints intrinsically characterize human actions.
- Inspired by this, skeleton is taken as input at each time slot.
- The co-occurrence features of the skeleton joints are learnt by a novel regularization scheme.
- A new dropout algorithm is used for training. It simultaneously operates on the gates, cells, and output responses of the LSTM neurons.

Co-occurrence Feature Learning for Action Rec.

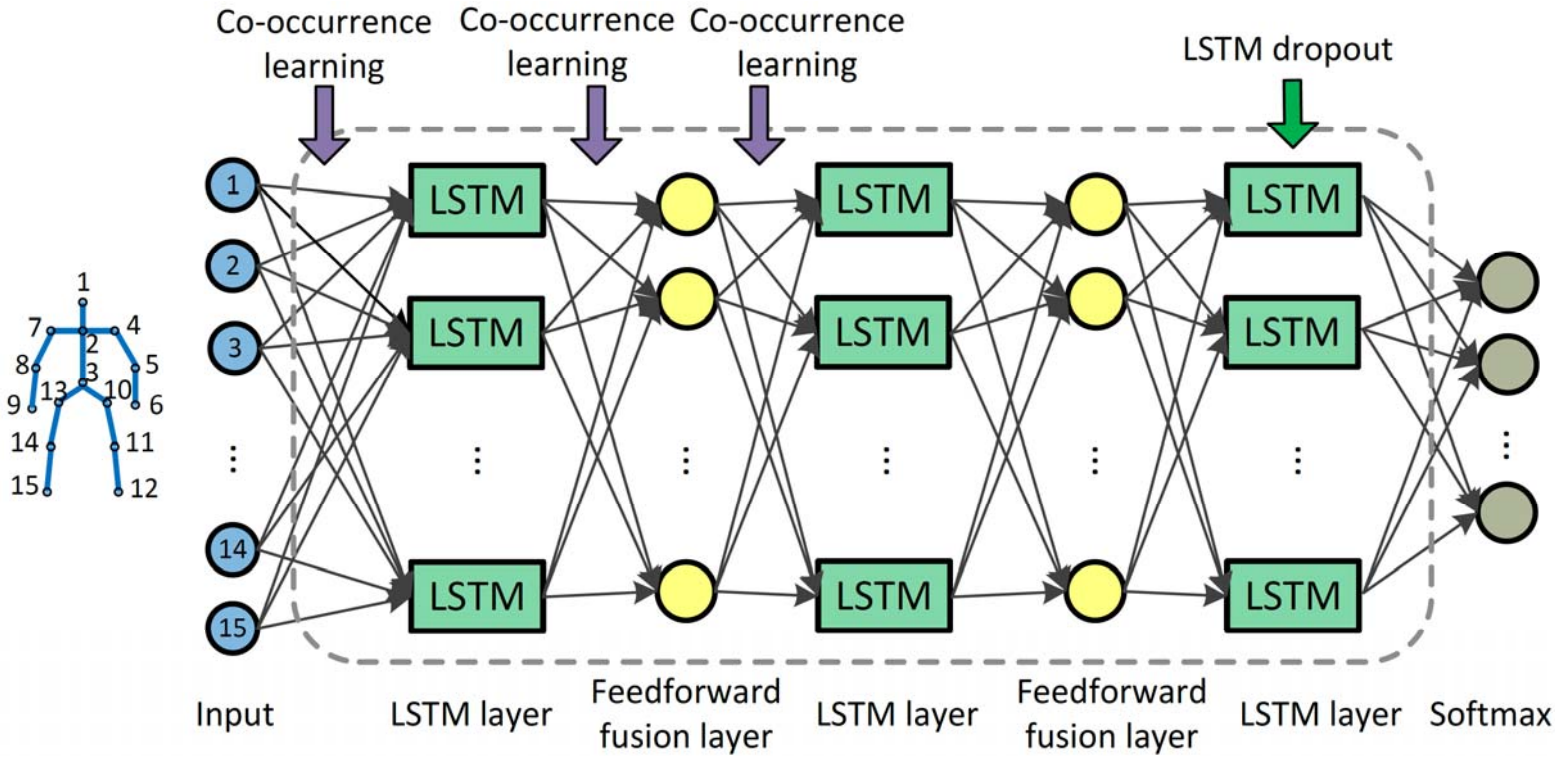
Approach: Regularized Deep LSTM



- Co-occurrences of joints intrinsically characterize human actions.
- For walking, the joints from hands and feet have high correlations but they all have low correlations with the joint of root.
- The sets of correlated joints for walking and drinking are very different, indicating the discriminative subset of joints varies for different types of actions.

Co-occurrence Feature Learning for Action Rec.

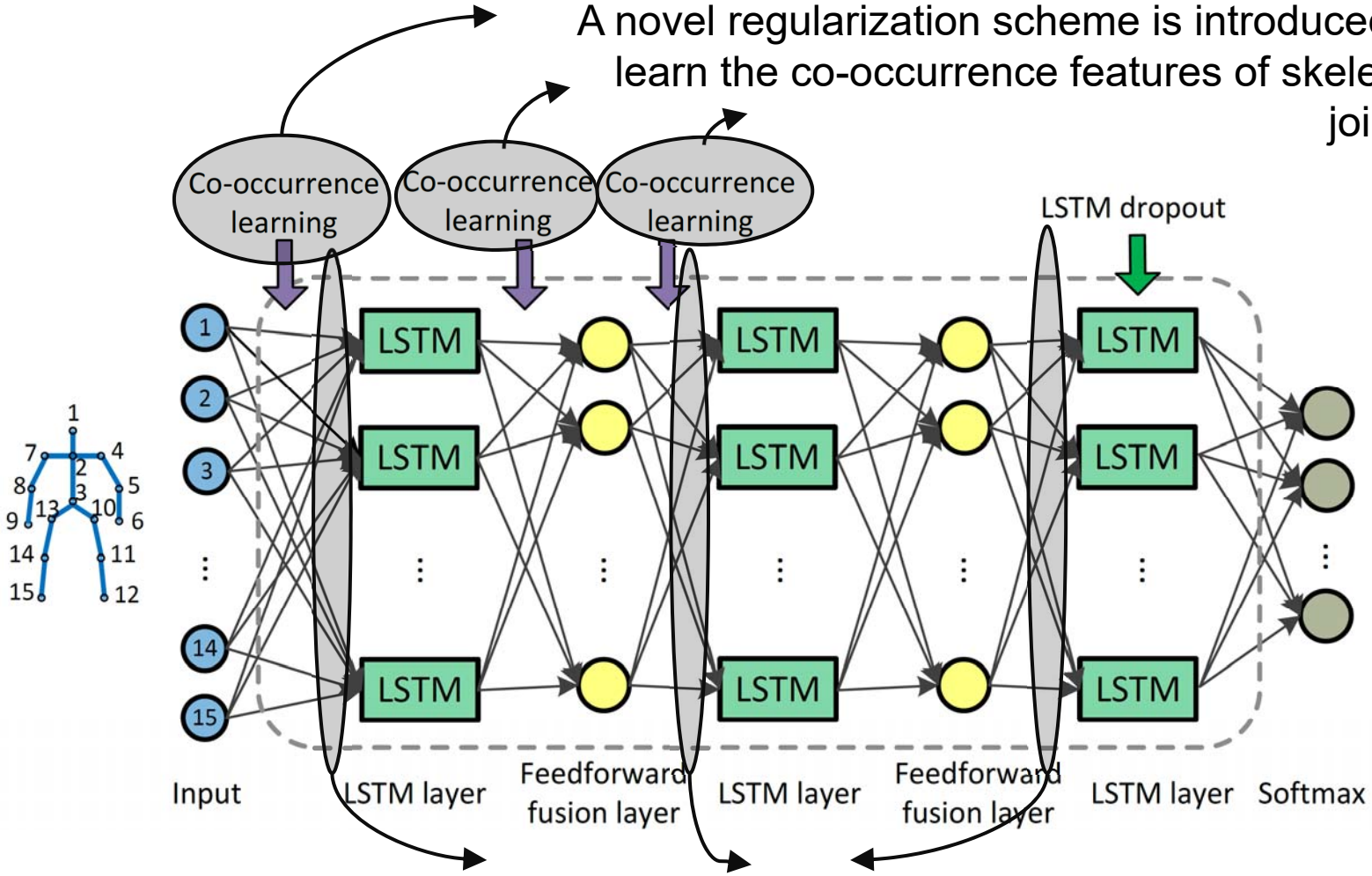
Approach: Regularized Deep LSTM



Co-occurrence Feature Learning for Action Rec.

Approach: Regularized Deep LSTM

A novel regularization scheme is introduced to learn the co-occurrence features of skeleton joints.



Skeleton data are taken as input at each time slot.

Co-occurrence Feature Learning for Action Rec.

Approach: Regularized Deep LSTM

- The network is expected to explore different co-occurrences for different types of actions.
- The network is expected to explore the conjunctions of discriminative joints.
- Fully connected network is designed to allow each neuron being connected to any joints (for the first layer) or responses of the previous layer (for the second or higher layer) to automatically explore the co-occurrences.
- The output responses are also referred to as “features” which are the input of the next layer.
- The neurons in the same layer are divided into K groups to allow different groups to focus on exploration of different conjunctions of discriminative joints.

Co-occurrence Feature Learning for Action Rec.

Approach: Regularized Deep LSTM

In the design, the co-occurrence regularization is incorporated into the loss function:

$$\min_{\boldsymbol{W}_{x\beta}} \boldsymbol{\mathcal{L}} + \lambda_1 \sum_{\beta \in S} ||\boldsymbol{W}_{x\beta}||_1 + \lambda_1 \sum_{\beta \in S} \sum_{k=1}^{K} \left\| \boldsymbol{W}_{x\beta,k}^T \right\|_{2,1}$$

- $\boldsymbol{\mathcal{L}}$ is the maximum likelihood loss of the LSTM. The other two terms are used for the co-occurrence regularization.
- $\boldsymbol{W}_{x\beta} \in R^{NxJ}$ is the connection weight matrix from inputs to the units associated with $\beta \in S$, (N is the number of neurons, J is the dimension of inputs.
- The N neurons are partitioned into K groups and the number of neurons in a group is $L = [N/K]$, with [.] representing the rounding up operation.

Co-occurrence Feature Learning for Action Rec.

Approach: Regularized Deep LSTM

In the design, the co-occurrence regularization is incorporated into the loss function:

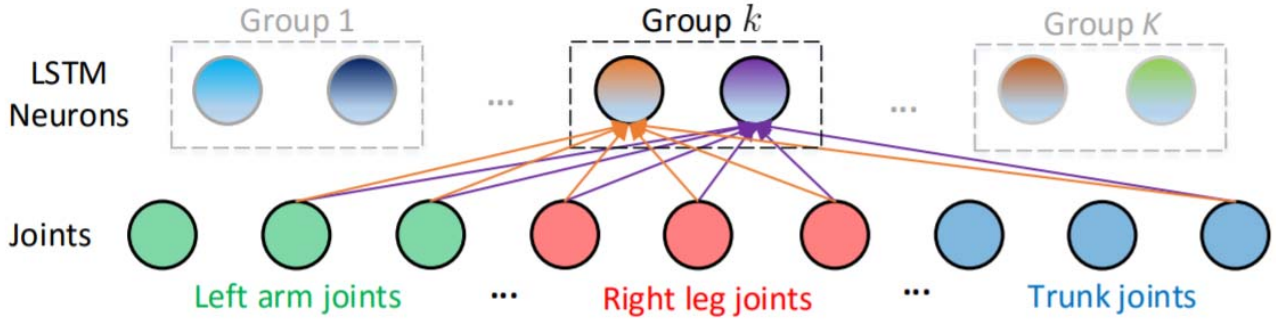
$$\min_{\boldsymbol{W}_{x\beta}} \boldsymbol{\mathcal{L}} + \lambda_1 \sum_{\beta \in S} ||\boldsymbol{W}_{x\beta}||_1 + \lambda_1 \sum_{\beta \in S} \sum_{k=1}^{K} \left\| \boldsymbol{W}_{x\beta,k}^T \right\|_{2,1}$$

- S denotes the set of internal units which are directly connected to the input of a neuron.
- In the third term, for each group of units, a structural l_{21} norm is used to drive the units to select a conjunction of descriptive inputs (joints/features).
- The l_1 norm constraint (the second term) helps to learn discriminative joints/features.

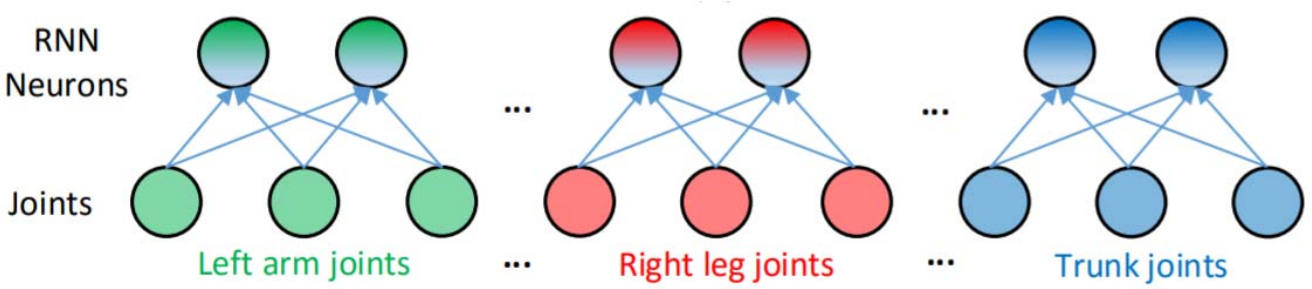
Co-occurrence Feature Learning for Action Rec.

Approach: Regularized Deep LSTM

Illustration of the connections between joints and neurons in the first layer:



(a) Co-occurrence connections automatically learned for Group k (proposed).



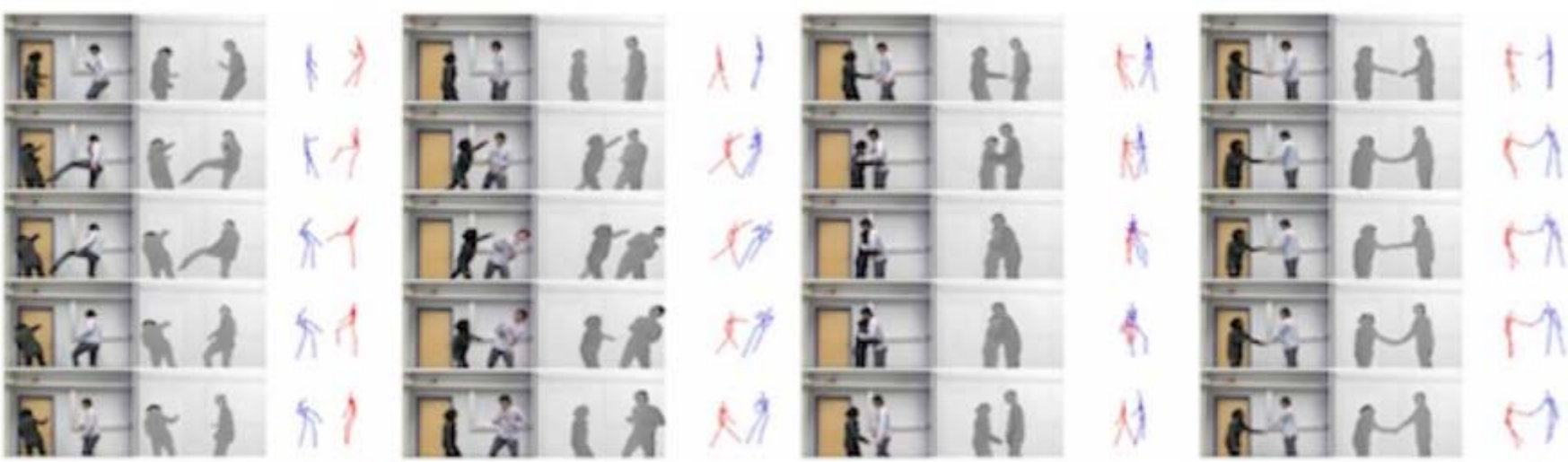
(b) Part-based subnet connections (Du, Wang, and Wang 2015), where the co-occurrences of joints across different parts are prohibited.

Co-occurrence Feature Learning for Action Rec.

Dataset: SBU Kinect Interaction Dataset

The SBU Kinect Interaction Dataset;

- is a Kinect captured human activity recognition dataset,
- depicting two person interaction,
- which contains 230 sequences of 8 classes (6,614 frames),
- with subject independent 5-fold cross validation.

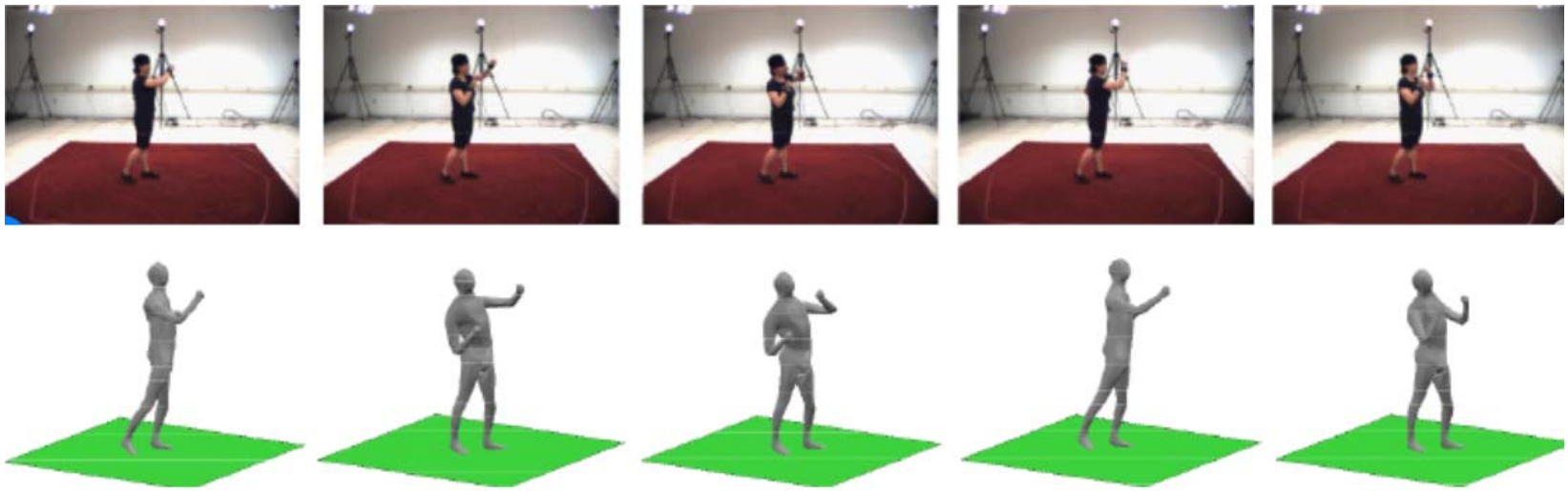


kicking punching hugging shaking hands

Co-occurrence Feature Learning for Action Rec.

Dataset: CMU Motion Capture Dataset

- This dataset is created by a Vicon mocap system consisting of 12 infrared MX-40 cameras, each records at 120 Hz with images of 4 MP resolution.
- Motions are captured in a working volume of approximately 3m x 8m. The capture subject wears 41 markers and a stylish black garment.
- Over 100 subjects, 109 action classes, 2605 sequences.



Co-occurrence Feature Learning for Action Rec.

Results

Results on SBU

Methods	Acc.(%)
Raw skeleton (Yun et al. 2012)	49.7
Joint feature (Yun et al. 2012)	80.3
Raw skeleton (Ji, Ye, and Cheng 2014)	79.4
Joint feature (Ji, Ye, and Cheng 2014)	86.9
Hierarchical RNN (Du, Wang, and Wang 2015)	80.35
Deep LSTM	86.03
Deep LSTM + Co-occurrence	89.44
Deep LSTM + Simple Dropout	89.70
Deep LSTM + In-depth Dropout	90.10
Deep LSTM+Co-occurrence+In-depth Dropout	**90.41**

Results on CMU

Methods	CMU subset	CMU
Hierarchical RNN (Du, Wang, and Wang 2015)	83.13	75.02
Deep LSTM	86.00	79.53
Deep LSTM + Co-occurrence	87.20	79.60
Deep LSTM + Simple Dropout	87.80	80.59
Deep LSTM + In-depth Dropout	88.25	80.99
Deep LSTM+ Co-occurrence+In-depth Dropout	**88.40**	**81.04**