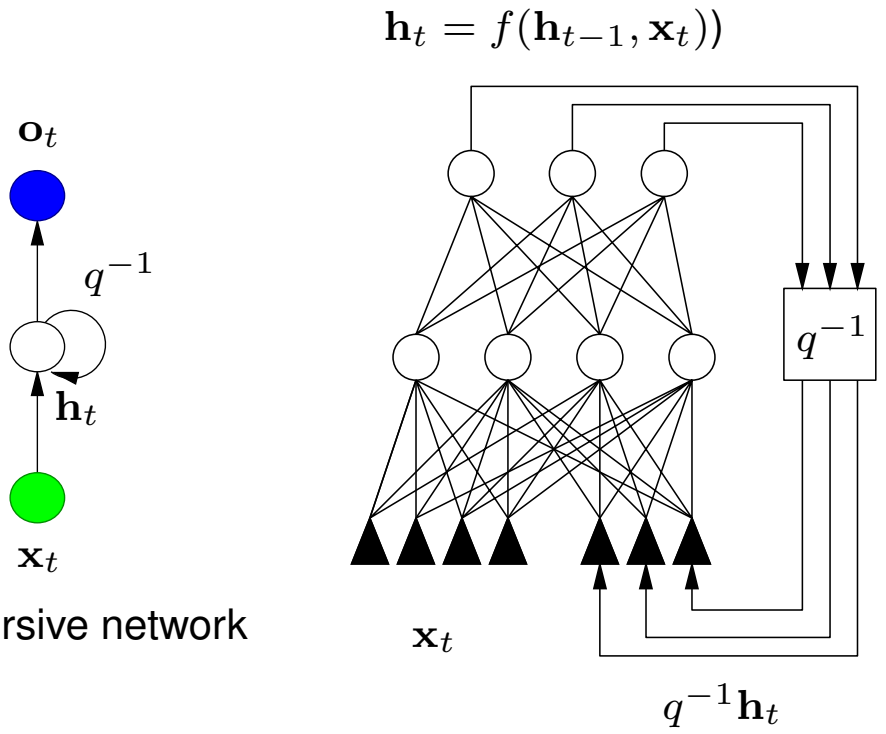
Graphical Description

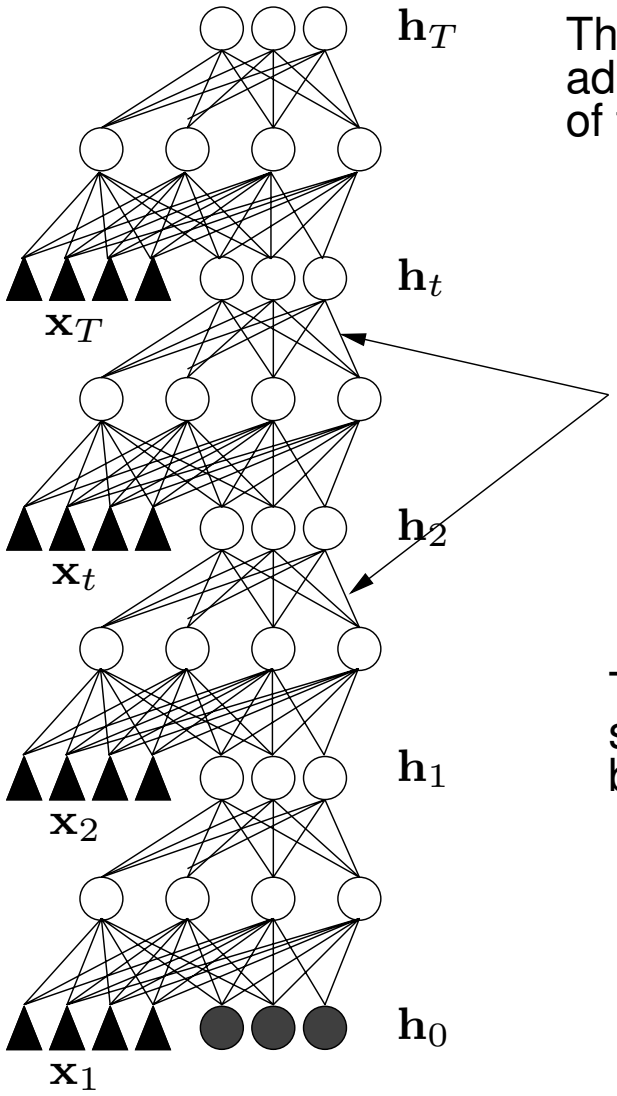
Graphical representation of the state transition function using the time shift operator q^{-1}:



Recursive network

$q^{-1}\mathbf{h}_t = \mathbf{h}_{t-1}$ (unitary time delay)

Time Unfolding



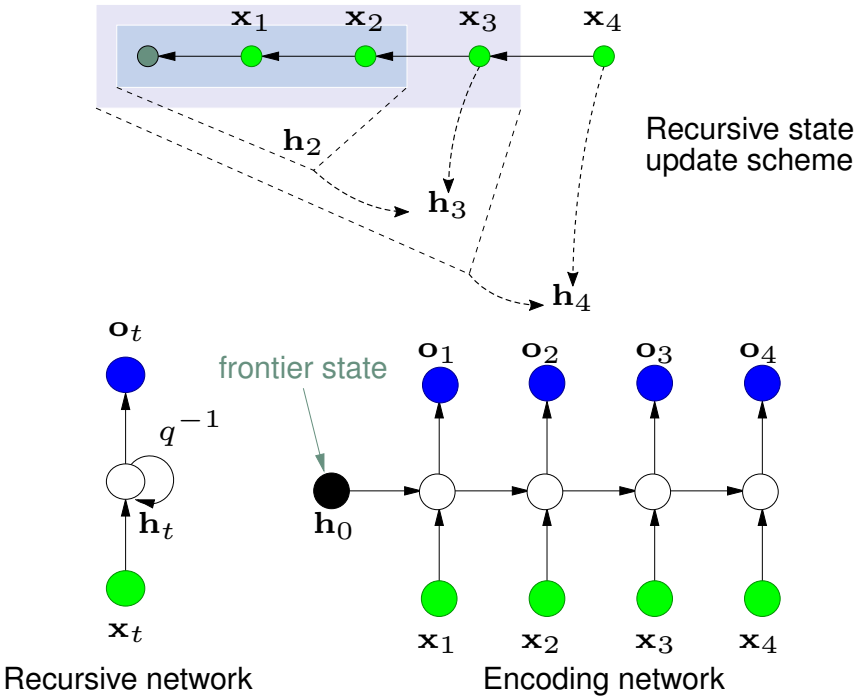
The last state is an adaptive encoding of the whole sequence

Weights are shared (replicated)

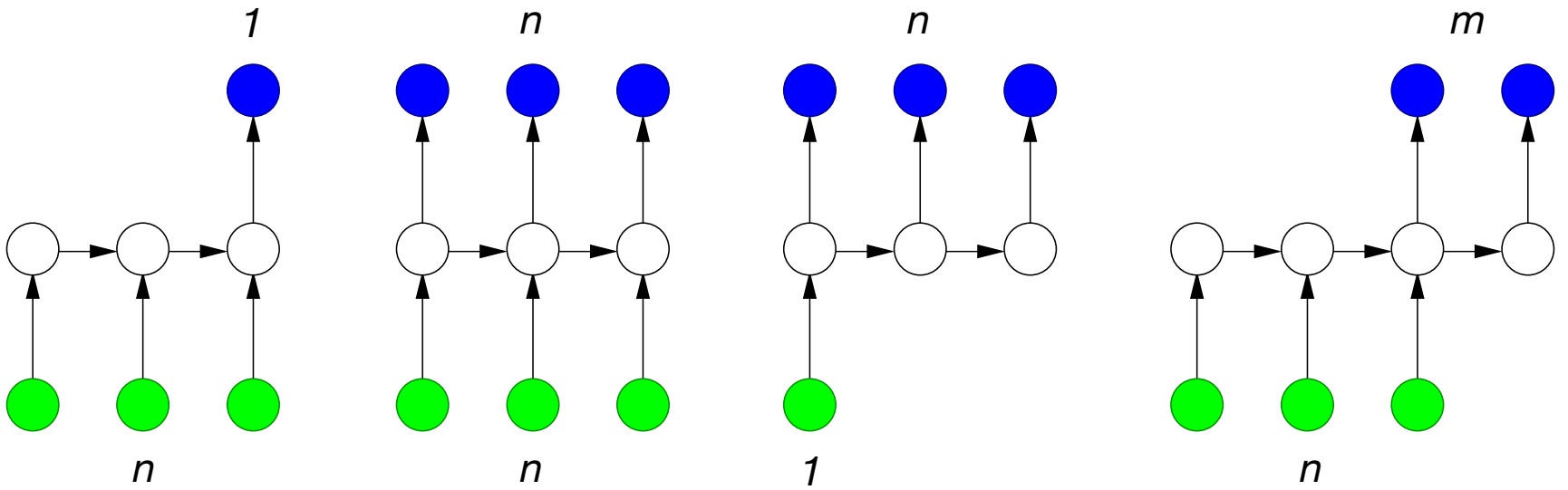
The unfolded network has a feedforward structure: e.g., gradient can be computed by Backpropagation (through time)

Encoding Networks

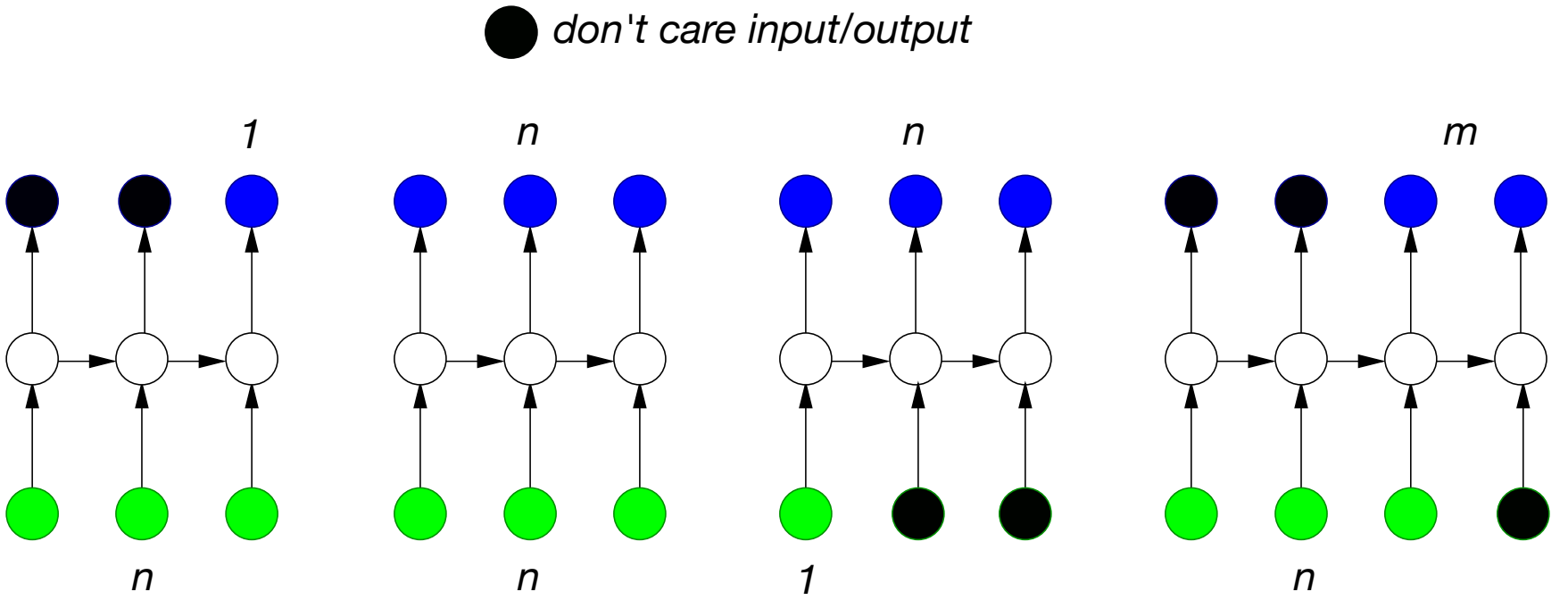
Given a sequence $\mathbf{s} \in \mathcal{X}^*$ and a recursive transduction τ, the *encoding network* associated to $\mathbf{s}$ and τ is formed by unrolling (*time unfolding*) the recursive network of τ through the input sequence $\mathbf{s}$



Examples of Sequential Transductions



A Unifying Perspective as IO-transductions



Model Types

How to implement $f(\cdot)$ and $g(\cdot)$?

Two general families of models:

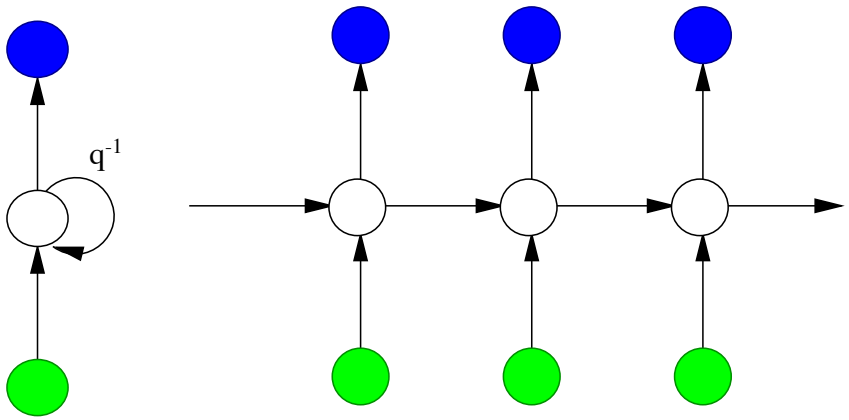
- Linear
  - Kalman Filter
  - Hidden Markov Models
  - Linear Dynamical Systems
    - Linear Autoencoders for Sequences
  - ...
- Nonlinear
  - Recurrent Neural Networks
  - ...

Shallow Recurrent Neural Networks

Non-linear Dynamical System

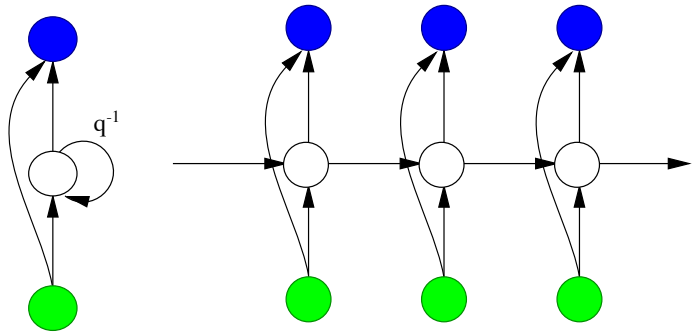
$$\mathbf{h}_t = f(\mathbf{A}\,\mathbf{x}_t + \mathbf{B}\,\mathbf{h}_{t-1}),$$
$$\mathbf{o}_t = g(\mathbf{C}\,\mathbf{h}_t),$$

where $f()$ and $g()$ are non-linear functions (e.g. $tanh()$), and $\mathbf{h}_0 = \mathbf{0}$ (or can be learned jointly with the other parameters)



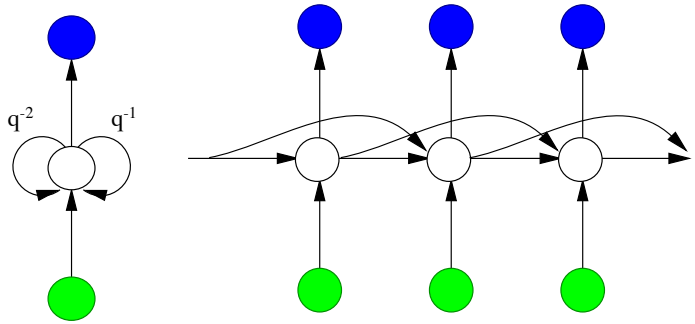
Some Additional Architectural Features for RNN

- using short-cut connections



$$\mathbf{o}_t = g(\mathbf{C}\,\mathbf{h}_t + \mathbf{D}\,\mathbf{x}_t)$$

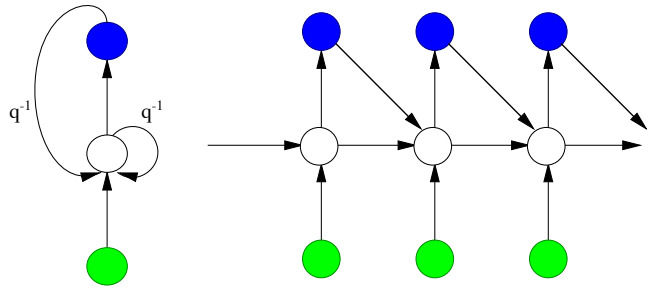
- using higher-order states (e.g., 2^{nd}-order)



$$\mathbf{h}_t = f(\mathbf{A}\,\mathbf{x}_t + \mathbf{B}^1\,\mathbf{h}_{t-1} + \mathbf{B}^2\,\mathbf{h}_{t-2})$$

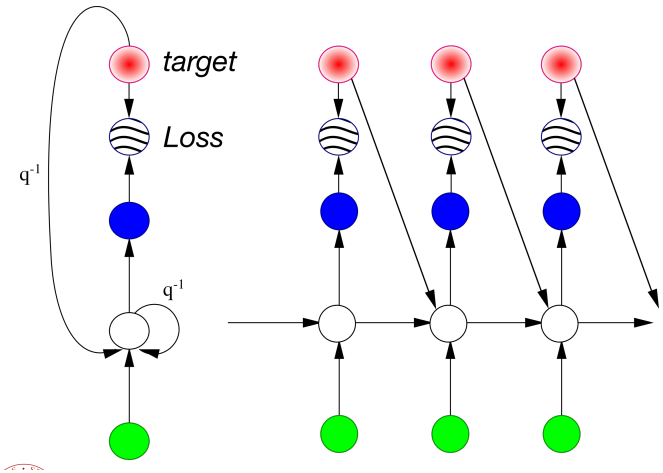
Some Additional Architectural Features for RNN

- using the output to convey contextual information



$$\mathbf{h}_t = f(\mathbf{A}\,\mathbf{x}_t + \mathbf{B}^i\,\mathbf{h}_{t-1} + \mathbf{B}^o\,\mathbf{o}_{t-1})$$

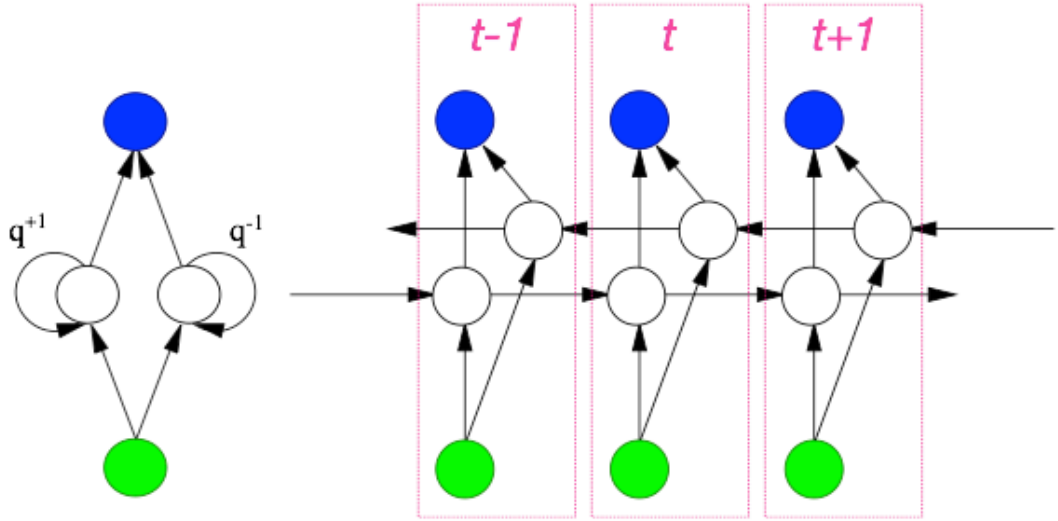
- training can be modified so to *force* the target signal (Teacher Forcing)



Some Additional Architectural Features for RNN

Bidirectional RNN (BRNN)

- when the sequence is not temporal or off-line processing (also in prediction)

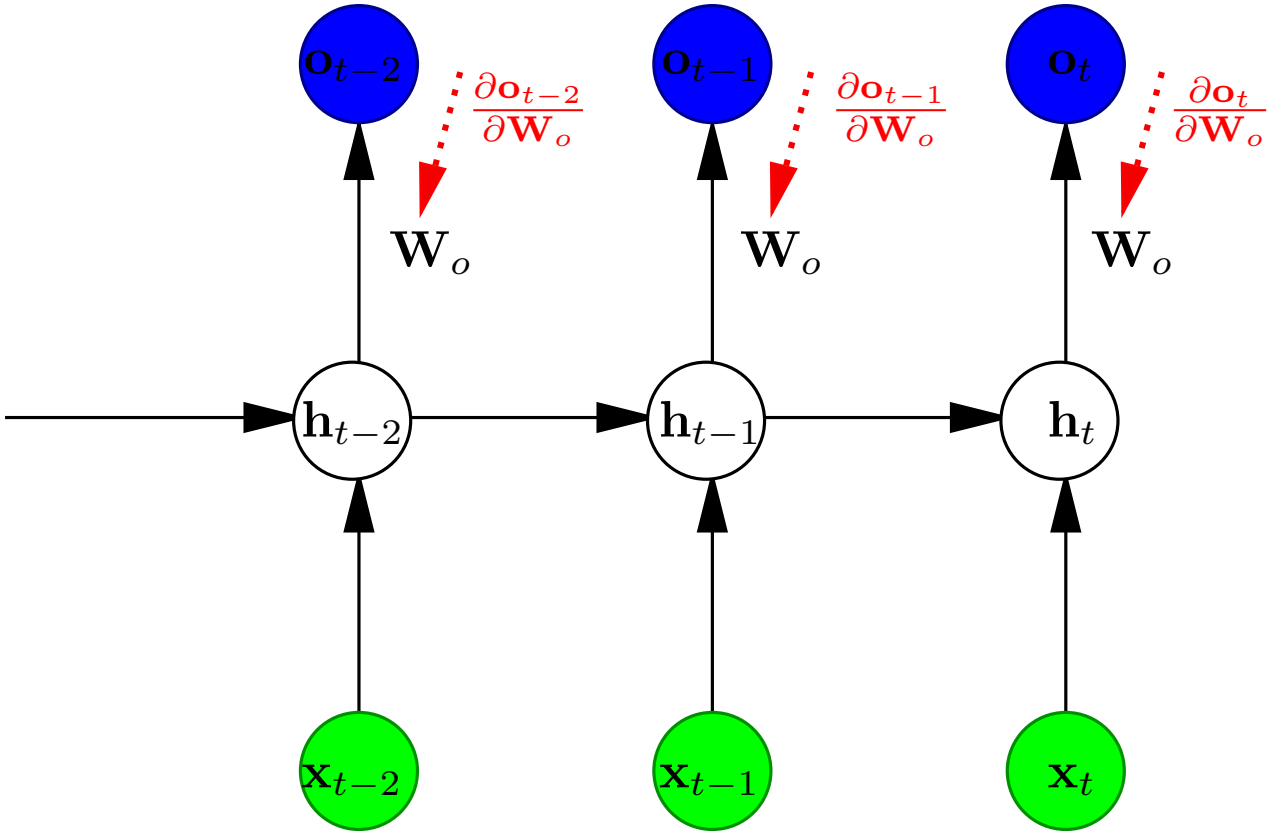


$$\mathbf{o}_t = g(\mathbf{C}^p\ \mathbf{h}_t^p + \mathbf{C}^f\ \mathbf{h}_t^f)$$

$$\mathbf{h}_t^p = f(\mathbf{A}^p\ \mathbf{x}_t + \mathbf{B}^p\ \mathbf{h}_{t-1}) \qquad \mathbf{h}_t^f = f(\mathbf{A}^f\ \mathbf{x}_t + \mathbf{B}^f\ \mathbf{h}_{t+1})$$

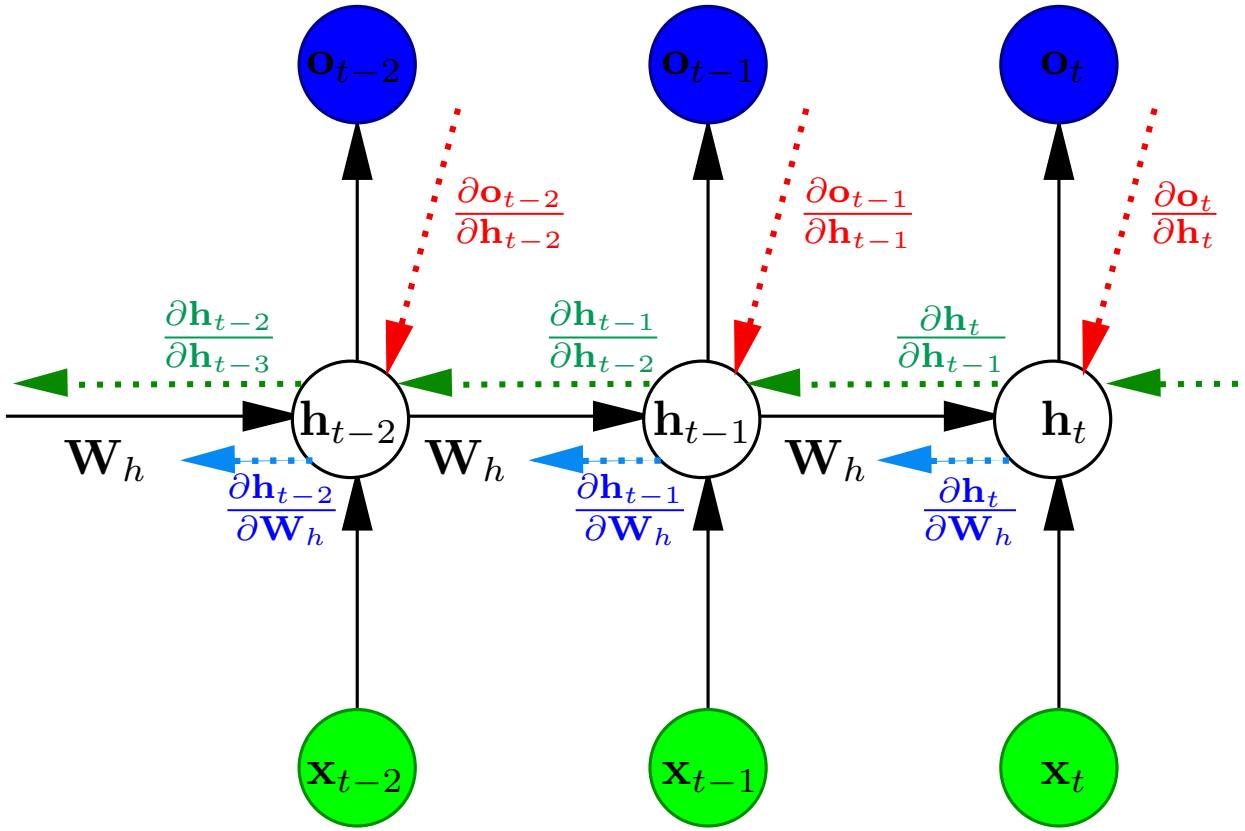
e.g., used in bioinformatics

Back-Propagation Through Time



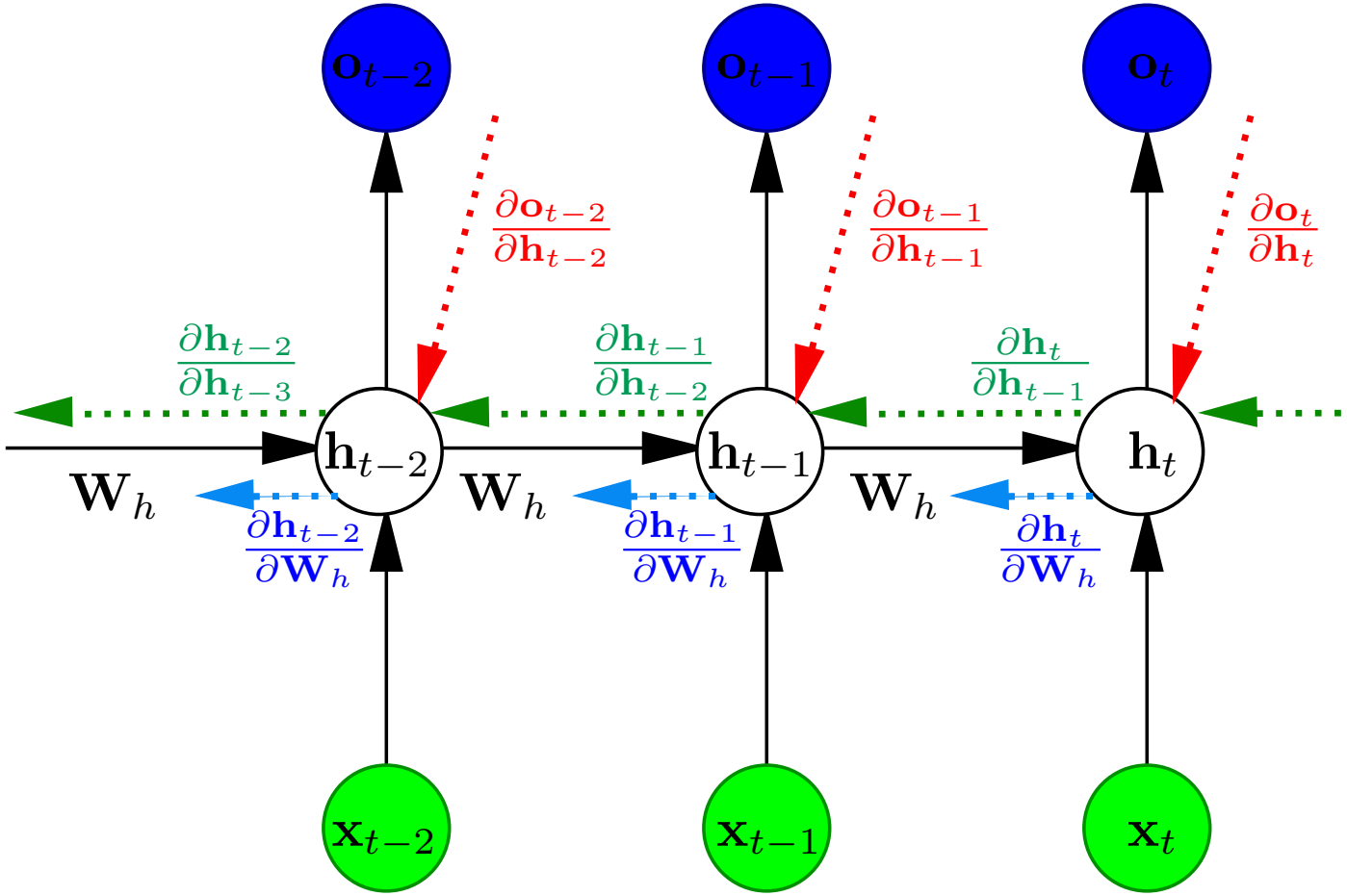
$$\frac{\partial \mathbf{J}}{\partial \mathbf{W}_o} = \cdots + \frac{\partial \mathbf{J}_{t-2}}{\partial \mathbf{o}_{t-2}} \frac{\partial \mathbf{o}_{t-2}}{\partial \mathbf{W}_o} + \frac{\partial \mathbf{J}_{t-1}}{\partial \mathbf{o}_{t-1}} \frac{\partial \mathbf{o}_{t-1}}{\partial \mathbf{W}_o} + \frac{\partial \mathbf{J}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_o}$$

Back-Propagation Through Time



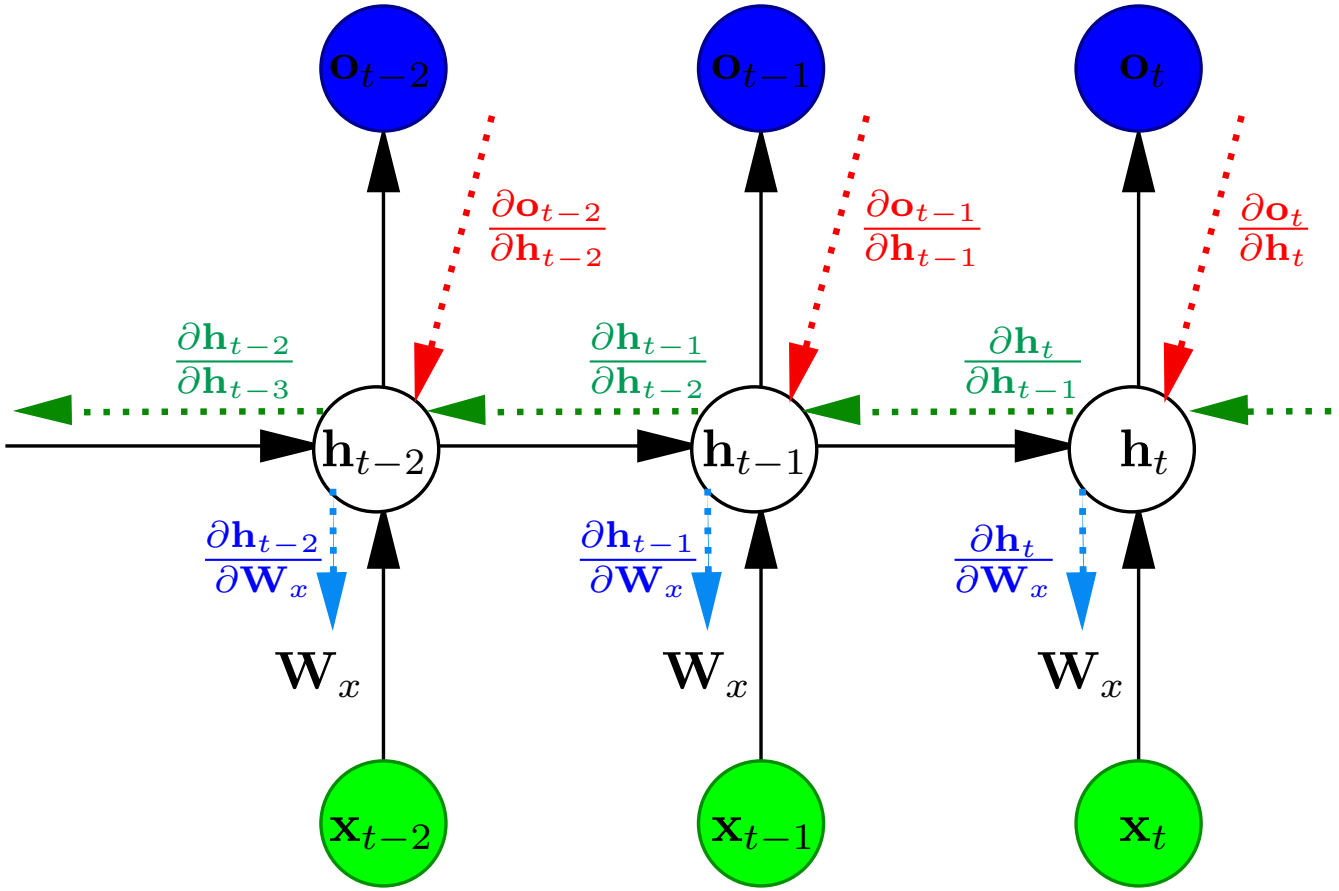
$$\frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_h} = \sum_{t'=1}^{t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t'}} \frac{\partial \mathbf{h}_{t'}}{\partial \mathbf{W}_h}, \text{ where } \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t'}} = \prod_{j=t'+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}$$

Back-Propagation Through Time



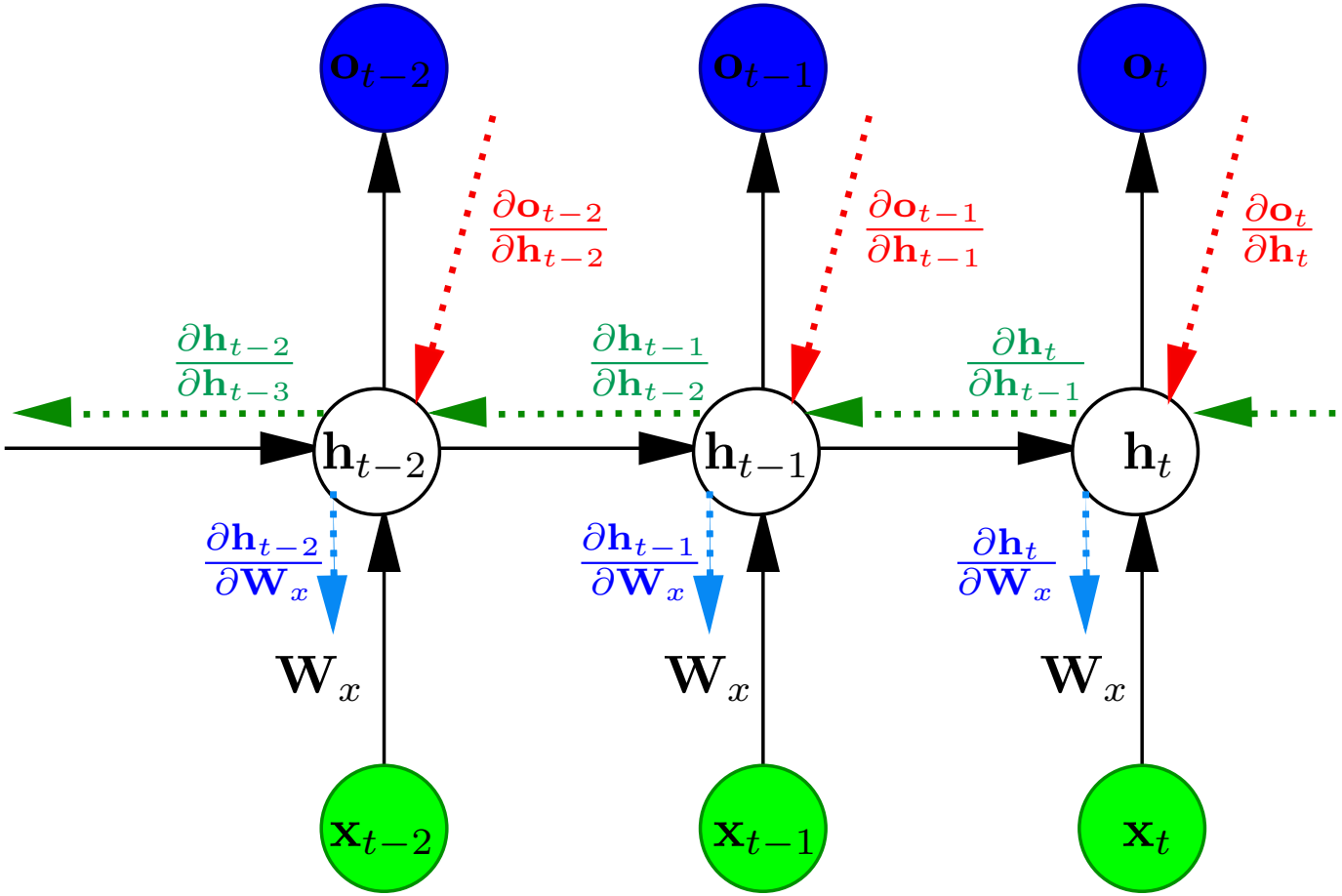
$$\frac{\partial \mathbf{J}}{\partial \mathbf{W}_h} = \cdots + \frac{\partial \mathbf{J}_{t-2}}{\partial \mathbf{o}_{t-2}} \frac{\partial \mathbf{o}_{t-2}}{\partial \mathbf{W}_h} + \frac{\partial \mathbf{J}_{t-1}}{\partial \mathbf{o}_{t-1}} \frac{\partial \mathbf{o}_{t-1}}{\partial \mathbf{W}_h} + \frac{\partial \mathbf{J}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_h}$$

Back-Propagation Through Time



$$\frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_x} = \sum_{t'=1}^{t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t'}} \frac{\partial \mathbf{h}_{t'}}{\partial \mathbf{W}_x}$$

Back-Propagation Through Time



$$\frac{\partial \mathbf{J}}{\partial \mathbf{W}_x} = \cdots + \frac{\partial \mathbf{J}_{t-2}}{\partial \mathbf{o}_{t-2}} \frac{\partial \mathbf{o}_{t-2}}{\partial \mathbf{W}_x} + \frac{\partial \mathbf{J}_{t-1}}{\partial \mathbf{o}_{t-1}} \frac{\partial \mathbf{o}_{t-1}}{\partial \mathbf{W}_x} + \frac{\partial \mathbf{J}_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_x}$$

Vanishing/Exploding Gradients: how to cope with them

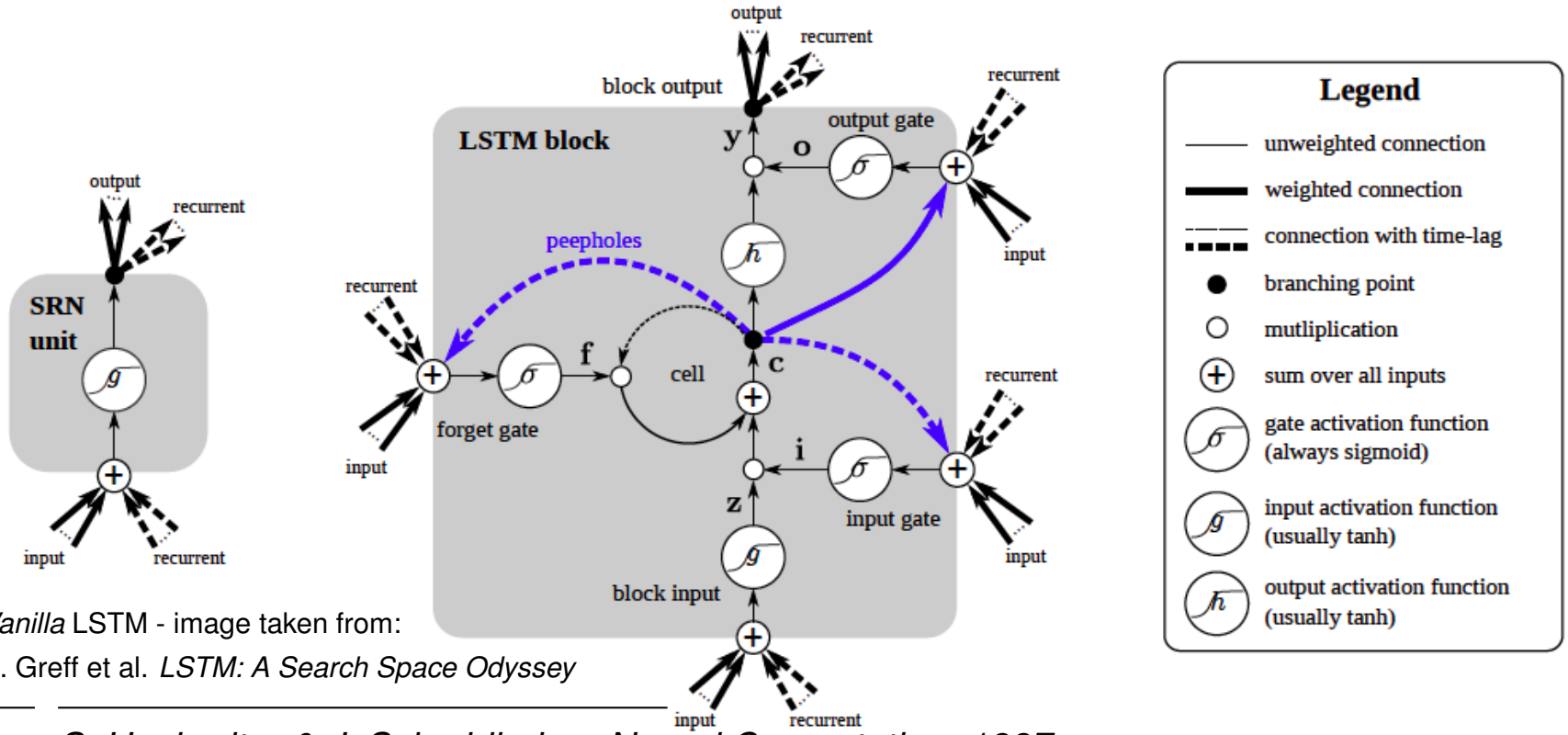
Here are some approaches to try to reduce the vanishing/exploding gradients problem

- Architectural
  - Long Short-Term Memory units
  - Reservoir Computing: Echo State Networks and Liquid State Machines
- Algorithmic
  - *Clipping gradients (avoids exploding gradients)*
  - *Hessian Free Optimization*
  - Smart Initialization: pre-training techniques

Long Short-Term Memory

Extension of RNN that can deal with long-term temporal dependencies

Mechanism that allows the networks to "remember" relevant information for a long period of time



Vanilla LSTM - image taken from:

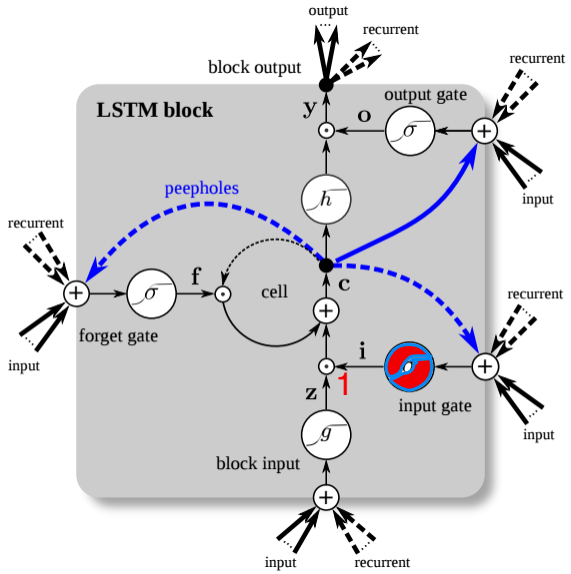
K. Greff et al. *LSTM: A Search Space Odyssey*

S. Hochreiter & J. Schmidhuber, Neural Computation, 1997

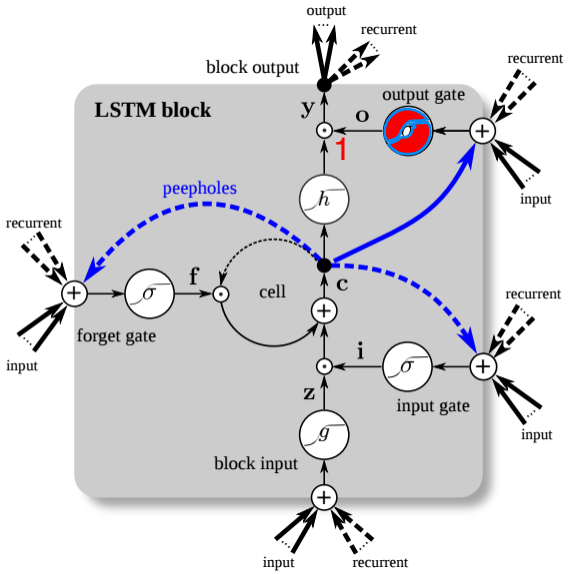
Long Short-Term Memory (*Vanilla* LSTM)

- Exploits a **linear** memory cell (*state*) that integrates input information through time
  - memory obtained by self-loop
  - gradient not down-sized by Jacobian of sigmoidal function $\Rightarrow$ no vanishing gradient
- 3 gate units (with sigmoid: soft version of a 0/1 switch) control the information flow via multiplicative connections
  - input gate "on": let input to flow in the memory cell
  - output gate "on": let the current value stored in the memory cell to be read in output
  - forget gate "off": let the current value stored in the memory cell to be reset to 0

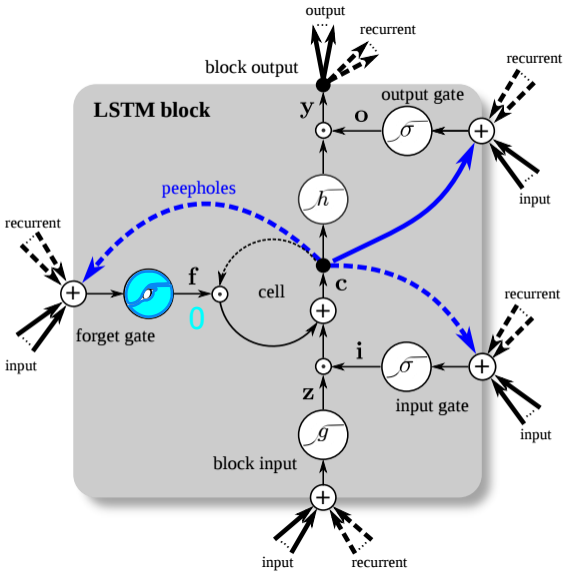
input gate on



output gate on

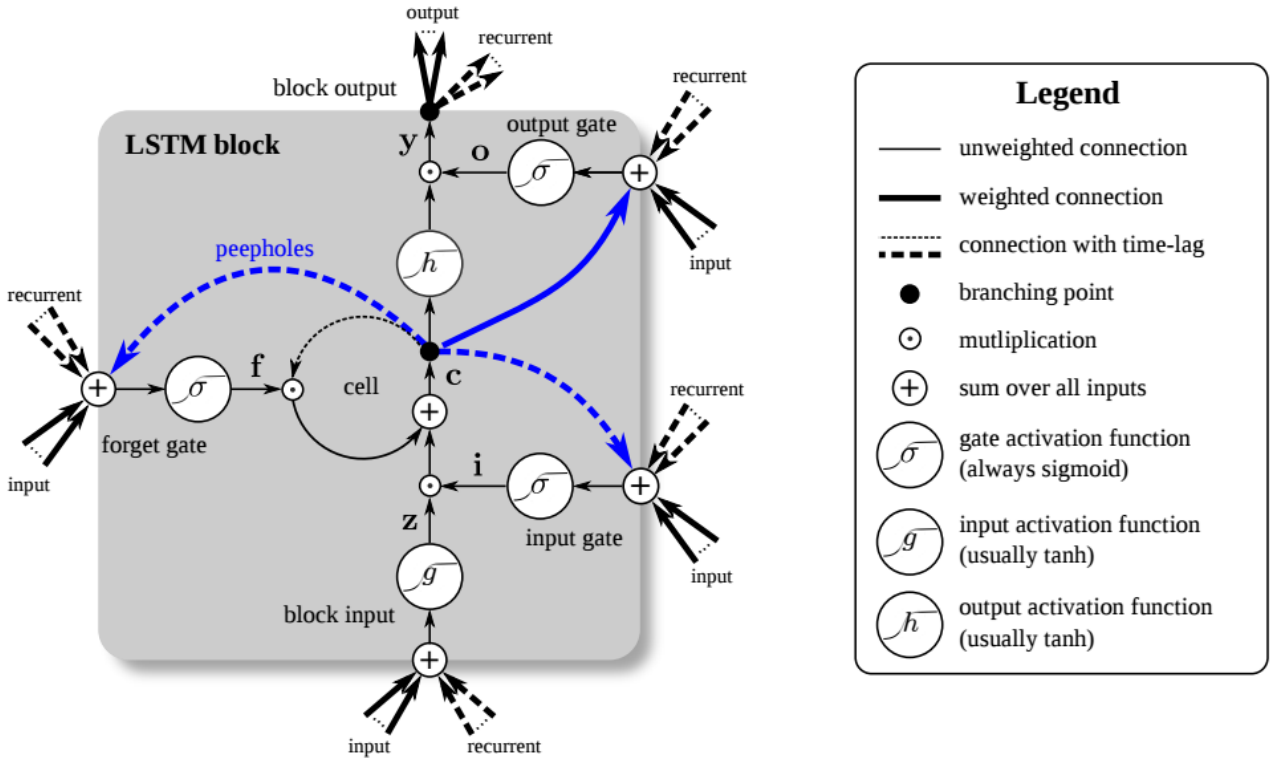


forget gate off



Long Short-Term Memory (*Vanilla* LSTM)

- peepholes connections allow to directly control all gates to allow for easier learning of precise timings
- full back-propagation through time (BPTT) training introduced only in 2005



Simplifying LSTM: Gated Recurrent Units

Is it possibile to simplify LSTM units ?

- Gated Recurrent Units (GRU) do that by using a single gating unit that simultaneously controls the forgetting factor and the decision to update the state unit

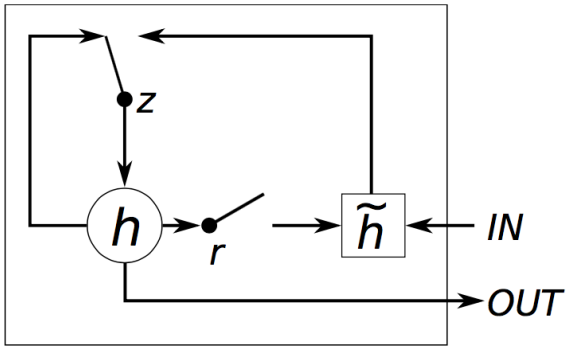


image taken from Chung et al., NIPS Workshop on Deep Learning, 2014

$$[\mathbf{h}_t]_i = z_i[\mathbf{h}_{t-1}]_i + (1 - z_i)\underbrace{\sigma\left(\mathbf{A}_i\mathbf{x}_t + \mathbf{B}_i(\mathbf{r} \odot \mathbf{h}_{t-1})\right)}_{[\tilde{\mathbf{h}}_t]_i}$$

- the *update* gate *z* selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$
- the *reset* gate *r* decides whether the previous hidden state is ignored

Cho et al., *EMNLP*, 2014

Reservoir Computing: Echo State Nets and Liquid State Machines

Idea: fix the input-to-hidden and hidden-to-hidden connections at random values and only learn the output units connections

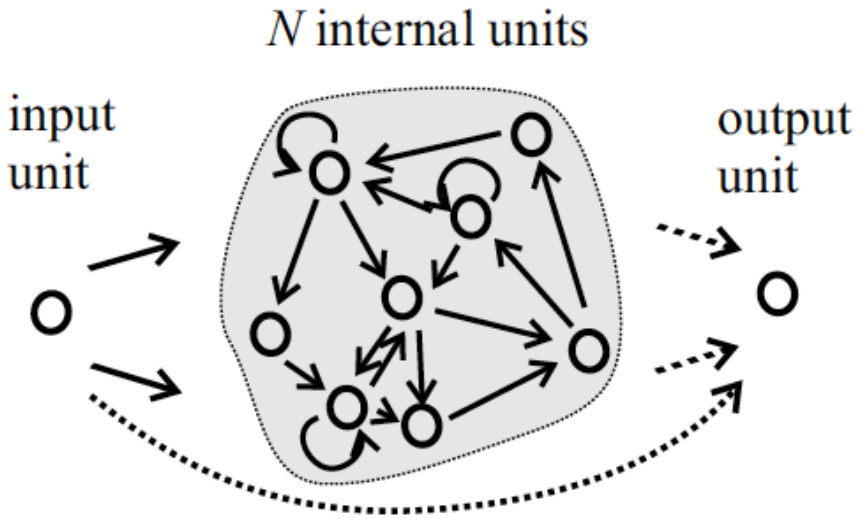


image taken from:
H. Jaeger, *NIPS 2002*

- Echo State Networks (ESN): standard recurrent neurons (+ leaky integrators)
- Liquid State Machines (LSM): spiking integrate-and-fire neurons and dynamic synaptic connection models

---

Jaeger, Technical Report GMD Report 148, 2001

Maass et al., Neural Computation, 2002

Hessian Free Optimization

More efficient gradient descent optimization can reduce the vanishing gradient problem

- 1^{st} order gradient descent (black arrows) in RNN (i.e. Stochastic Gradient Descent) is not very effective because of regions of pathological curvature in the objective function $J(\mathbf{W})$

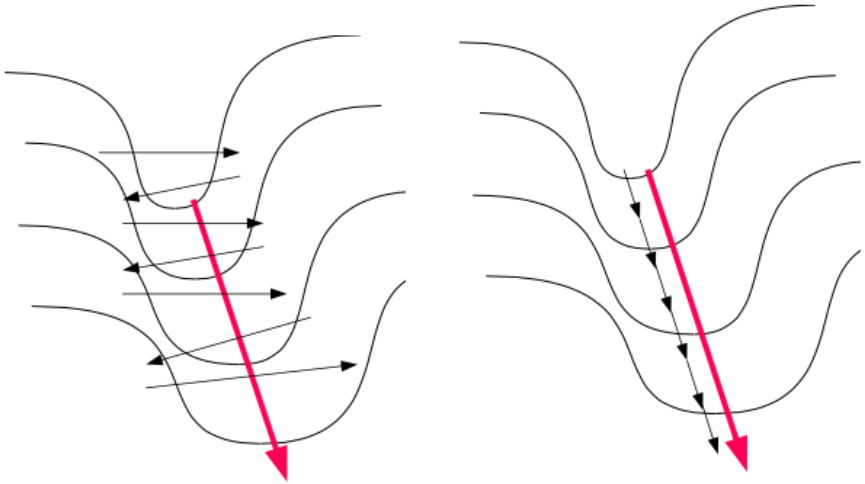


image taken from: Martens *ICML 2010*

- 2^{nd} order methods (red arrows), based on information about curvature, can do better, but are computationally expensive (computation of Hessian Matrix)

Hessian Free Optimization

Newton's Method locally approximates $J(\mathbf{W})$ up to second order

$$J(\mathbf{W}+\boldsymbol{\delta}) \approx J(\mathbf{W}) + \underbrace{\nabla J(\mathbf{W})^{\mathsf{T}}\boldsymbol{\delta}}_{1^{st}-order} + \underbrace{\boldsymbol{\delta}^{\mathsf{T}}\mathbf{H}(\mathbf{W})\boldsymbol{\delta}}_{2^{nd}-order}$$

where $\mathbf{H}$ is the Hessian matrix (in this context, dubbed *Curvature Matrix*)

$$\mathbf{H}(\mathbf{W}) = \begin{bmatrix} \frac{\partial^2 J(\mathbf{W})}{\partial w_1 \partial w_1} & \frac{\partial^2 J(\mathbf{W})}{\partial w_1 \partial w_2} & \frac{\partial^2 J(\mathbf{W})}{\partial w_1 \partial w_3} & .. & \frac{\partial^2 J(\mathbf{W})}{\partial w_1 \partial w_n} \\ \frac{\partial^2 J(\mathbf{W})}{\partial w_2 \partial w_1} & \frac{\partial^2 J(\mathbf{W})}{\partial w_2 \partial w_2} & \frac{\partial^2 J(\mathbf{W})}{\partial w_2 \partial w_3} & .. & \frac{\partial^2 J(\mathbf{W})}{\partial w_2 \partial w_n} \\ \frac{\partial^2 J(\mathbf{W})}{\partial w_3 \partial w_1} & \frac{\partial^2 J(\mathbf{W})}{\partial w_3 \partial w_2} & \frac{\partial^2 J(\mathbf{W})}{\partial w_3 \partial w_3} & .. & \frac{\partial^2 J(\mathbf{W})}{\partial w_3 \partial w_n} \\ .. & .. & .. & .. & .. \\ \frac{\partial^2 J(\mathbf{W})}{\partial w_n \partial w_1} & \frac{\partial^2 J(\mathbf{W})}{\partial w_n \partial w_2} & \frac{\partial^2 J(\mathbf{W})}{\partial w_n \partial w_3} & .. & \frac{\partial^2 J(\mathbf{W})}{\partial w_n \partial w_n} \end{bmatrix}$$

- $\mathbf{H}_{ij}$ specifies how much the gradient in direction i changes as we move in direction j
- 1^{st}-order methods are not effective when off-diagonal entries are large: steepest descent for one weight is *disturbed* by the simultaneous changes to all the other weights

Hessian Free Optimization

Newton's Method suggests to take a step in the direction

$$\boldsymbol{\delta}^\star = \arg\min_{\boldsymbol{\delta}} J(\mathbf{W} + \boldsymbol{\delta})$$

- when $\mathbf{H}$ is positive definite $\boldsymbol{\delta}^\star$ exists:
  $\frac{\partial J(\mathbf{W})}{\partial \boldsymbol{\delta}} = \mathbf{H}(\mathbf{W})\boldsymbol{\delta} + \nabla J(\mathbf{W}) = \mathbf{0} \Rightarrow \boldsymbol{\delta}^\star = -\mathbf{H}^{-1}\nabla J(\mathbf{W})$
- when $\mathbf{H}$ is indefinite, it is re-conditioned, i.e. it is substituted by $(\mathbf{H} + \lambda\mathbf{I})$, where $\lambda \geq 0$ and $\mathbf{I}$ is the identity matrix (this is usually done anyway to avoid numerical problems)

Computing $\mathbf{H}$ and $\mathbf{H}^{-1}$ ($O(n^3)$) is unfeasible even for medium-size networks

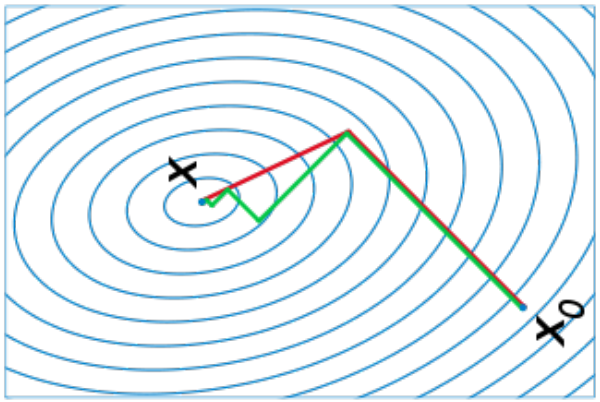
Hessian Free Optimization exploits the Conjugate Gradient approach (plus other "things"):

- $J(\mathbf{W} + \boldsymbol{\delta})$ approximated by $\frac{1}{2}\boldsymbol{\delta}^\mathsf{T}\mathbf{H}(\mathbf{W})\boldsymbol{\delta} - \nabla J(\mathbf{W})^\mathsf{T}\boldsymbol{\delta}$
- only requires access to matrix-vectors products with the curvature matrix
- has a fixed-size storage overhead of a few n-dimensional vectors
- holds optimality guarantees (speed of convergence), no more than n steps

Hessian Free Optimization

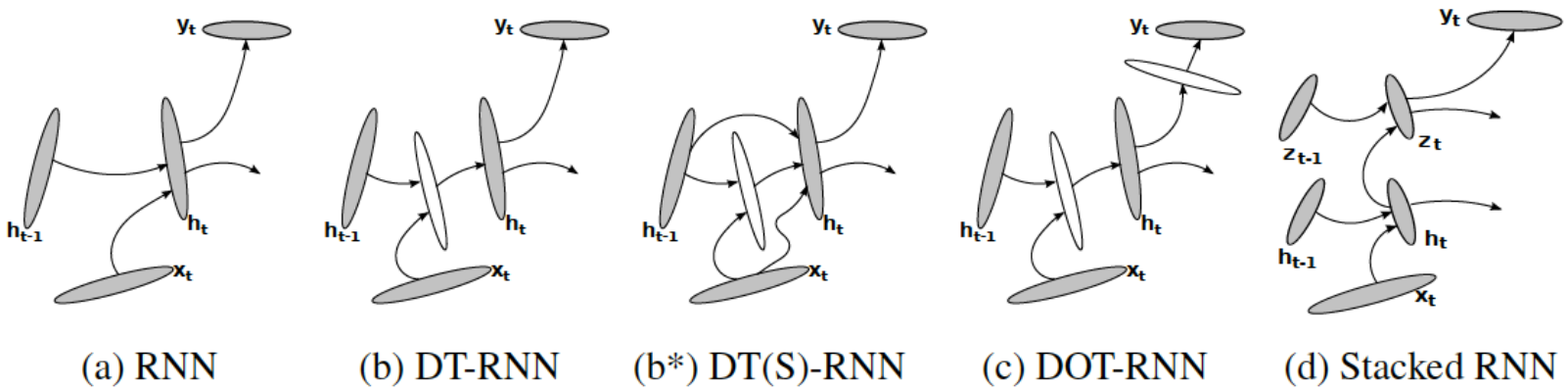
Intuition on how Conjugate Gradient works

- since $J(\mathbf{W} + \boldsymbol{\delta})$ is approximated, there is no hope to get the (local) minimum in one step
- an alternative is to perform multiple update steps, each of which finds the minimum along one direction
- however, avoid to undo previous work by choosing a new “conjugate” direction, i.e. which does not change the gradients in the previous directions



- the second direction in red has zero gradient with respect to the first direction in red (steepest descent is show in green)

How to Construct Deep Recurrent Neural Networks



(a) RNN (b) DT-RNN (b*) DT(S)-RNN (c) DOT-RNN (d) Stacked RNN

Figure 2: Illustrations of four different recurrent neural networks (RNN). (a) A conventional RNN. (b) Deep Transition (DT) RNN. (b*) DT-RNN with shortcut connections (c) Deep Transition, Deep Output (DOT) RNN. (d) Stacked RNN

image taken from: Pascanu et al., arXiv:1312.6026v5

Empirical experiments on polyphonic music prediction and language modeling showed that RNN benefits from having a deeper architecture

Pascanu et al., arXiv:1312.6026v5, 2014

Recent Deep Architectures

- Novel deep RNN architectures/mechanisms have been designed for specific applications
- Examples of new emerging features
  - Encoder-Decoder
  - Alignment/Attention
  - Adaptive Computation Time
  - “External” memory
  - Pointers
- image to text applications

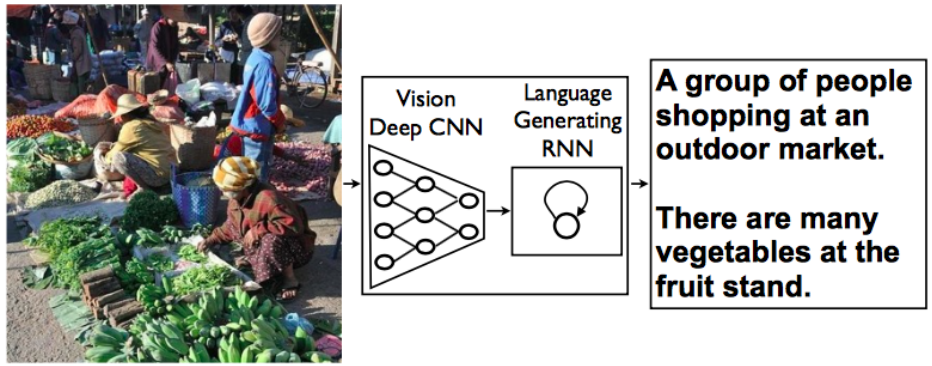
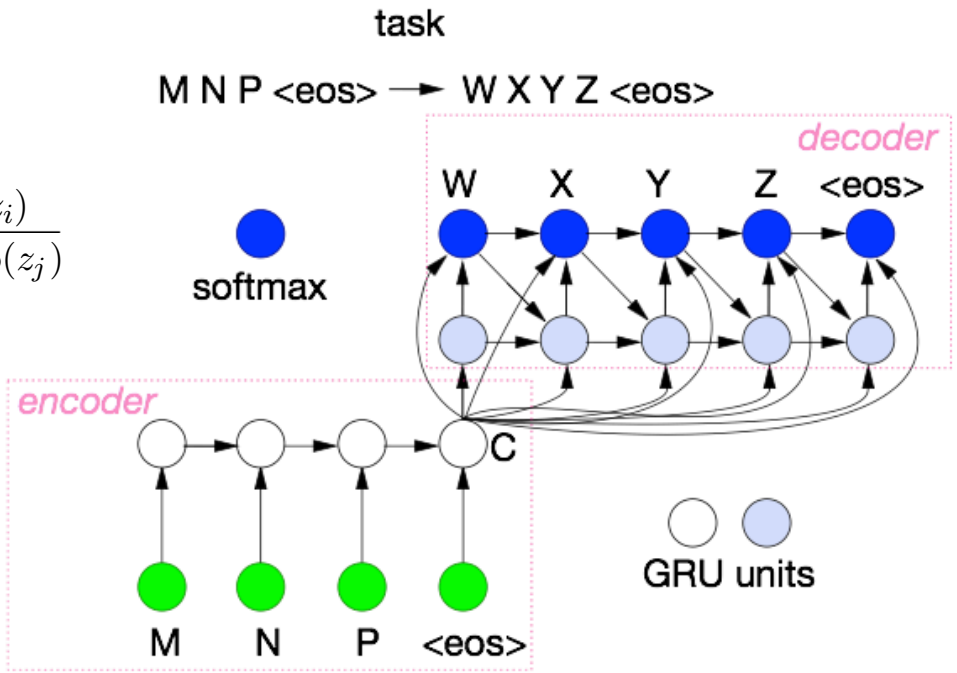


image taken from: Vinyals et al., *IEEE TPAMI*, 2016

Encoding-Decoding Networks

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



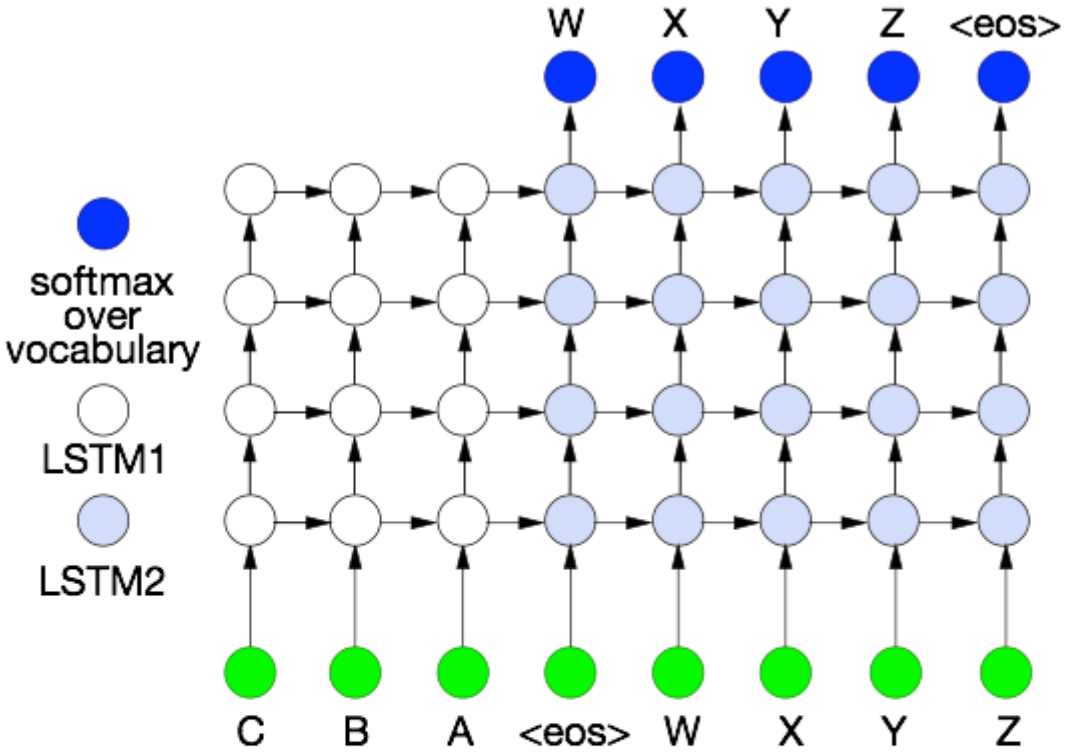
End-to-end learning of $(\max_\theta \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(\mathbf{s}_n^d | \mathbf{s}_n^i))$

$$P(\mathbf{o}_t | \mathbf{o}_{t-1}, \mathbf{o}_{t-2}, \ldots, \mathbf{o}_1, \mathbf{C}) \quad = \quad g(\mathbf{h}_t, \mathbf{o}_{t-1}, \mathbf{C})$$

where $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{o}_{t-1}, \mathbf{C})$

Cho et al., *EMNLP 2014*

Sequence to Sequence Networks



task

A B C <eos> → W X Y Z <eos>

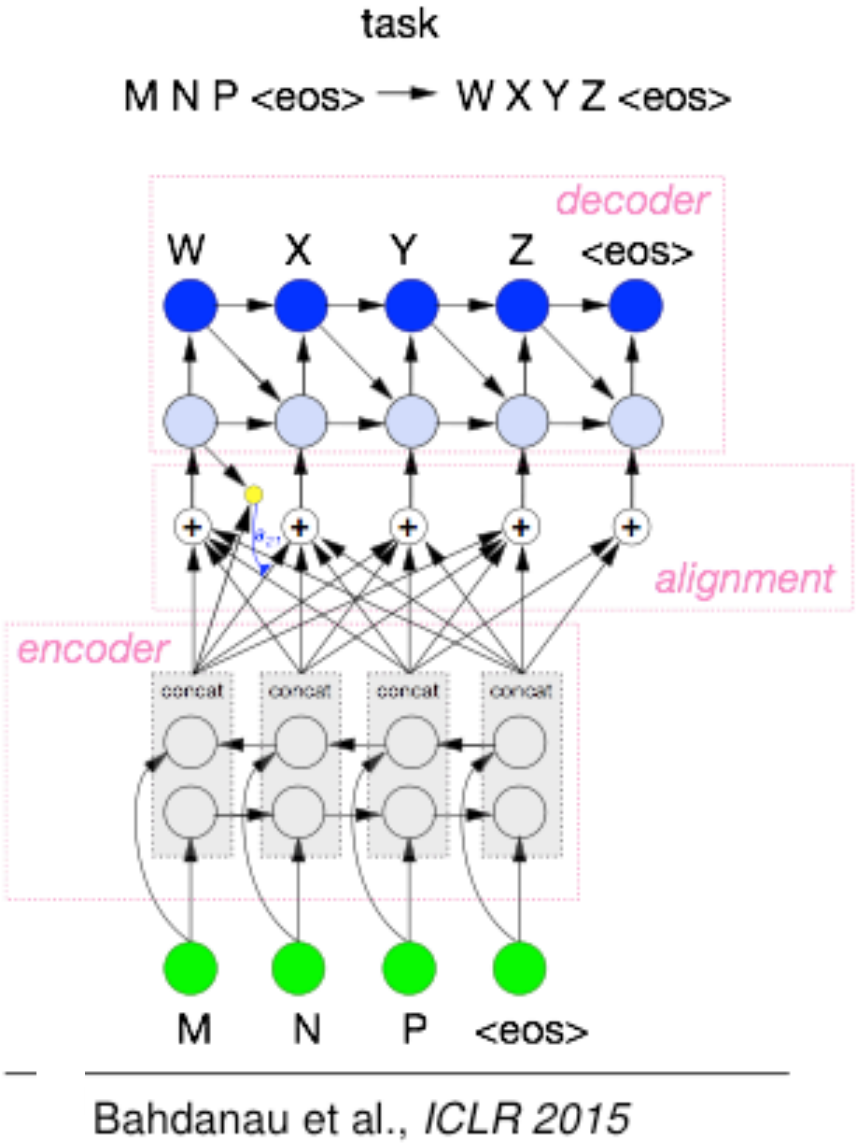
End-to-end learning of

$$P(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_{T'} | \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T'} P(\mathbf{o}_t | \mathbf{v}, \mathbf{o}_{t-1}, \mathbf{o}_{t-2}, ..., \mathbf{o}_1)$$

where $\mathbf{v} = \mathbf{h}_{LSTM1}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$

Sutskever et al., *NIPS 2014*

Encoding-Decoding Networks with Alignment



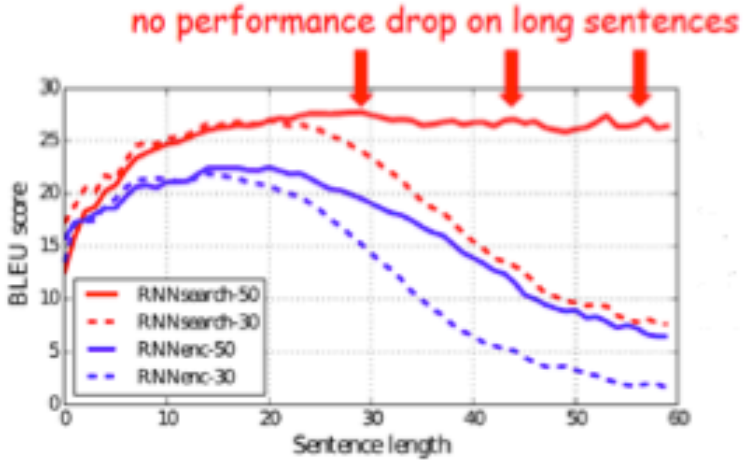
$$P(\mathbf{o}_t|\mathbf{o}_{t-1},\ldots,\mathbf{o}_1,\mathbf{s}^i) = g(\mathbf{o}_{t-1},\mathbf{h}_t^{dec},\mathbf{c}_t)$$

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}^{dec},\mathbf{o}_{t-1},\mathbf{c}_t)$$

$$\mathbf{c}_t = \sum_{j=1}^{T} \alpha_{ij}\mathbf{h}_j^{enc}$$

$$\alpha_{ij} = softmax_j(e_{ij})$$

$$e_{ij} = a(\mathbf{h}_{i-1}^{dec},\mathbf{h}_j^{enc})$$



One example of Image-Text Alignment (Karpathy & Fei-Fei, IEEE TPAMI, 2017)

Task

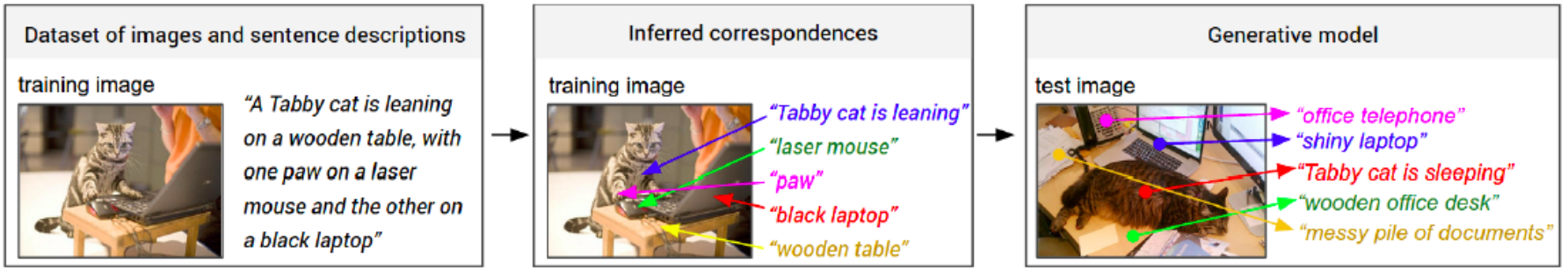
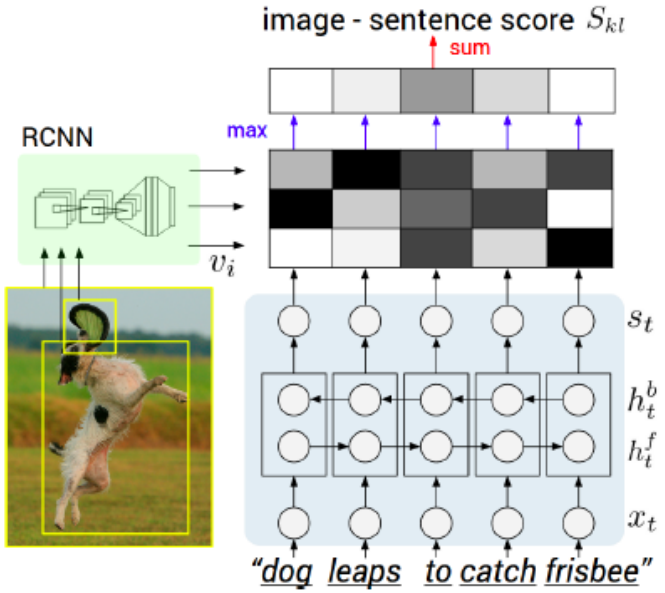


Figure 2. Overview of our approach. A dataset of images and their sentence descriptions is the input to our model (left). Our model first infers the correspondences (middle, Section 3.1) and then learns to generate novel descriptions (right, Section 3.2).



Multimodal Recurrent Neural Network architecture that uses the inferred alignments to learn to generate novel descriptions of image regions