数据挖掘与机器学习

1

潘斌
panbin@nankai.edu.cn
范孙楼227

上节回顾

- **决策树**

本节提要

- 神经网络
- BP算法/BP神经网络
- 循环神经网络RNN

人工神经网络

Artificial Neural Networks

- 本章关注：
  - 感知器
    - 感知器法则
    - 梯度下降（delta）法则
  - 多层网络和BP算法
    - 多层网络
    - BP算法

Artificial Neural Networks

- 人工神经网络 （Artificial Neural Network， ANN）
  - –神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应。（Kohonen,1988）

感知器（Perceptron）

感知准则函数

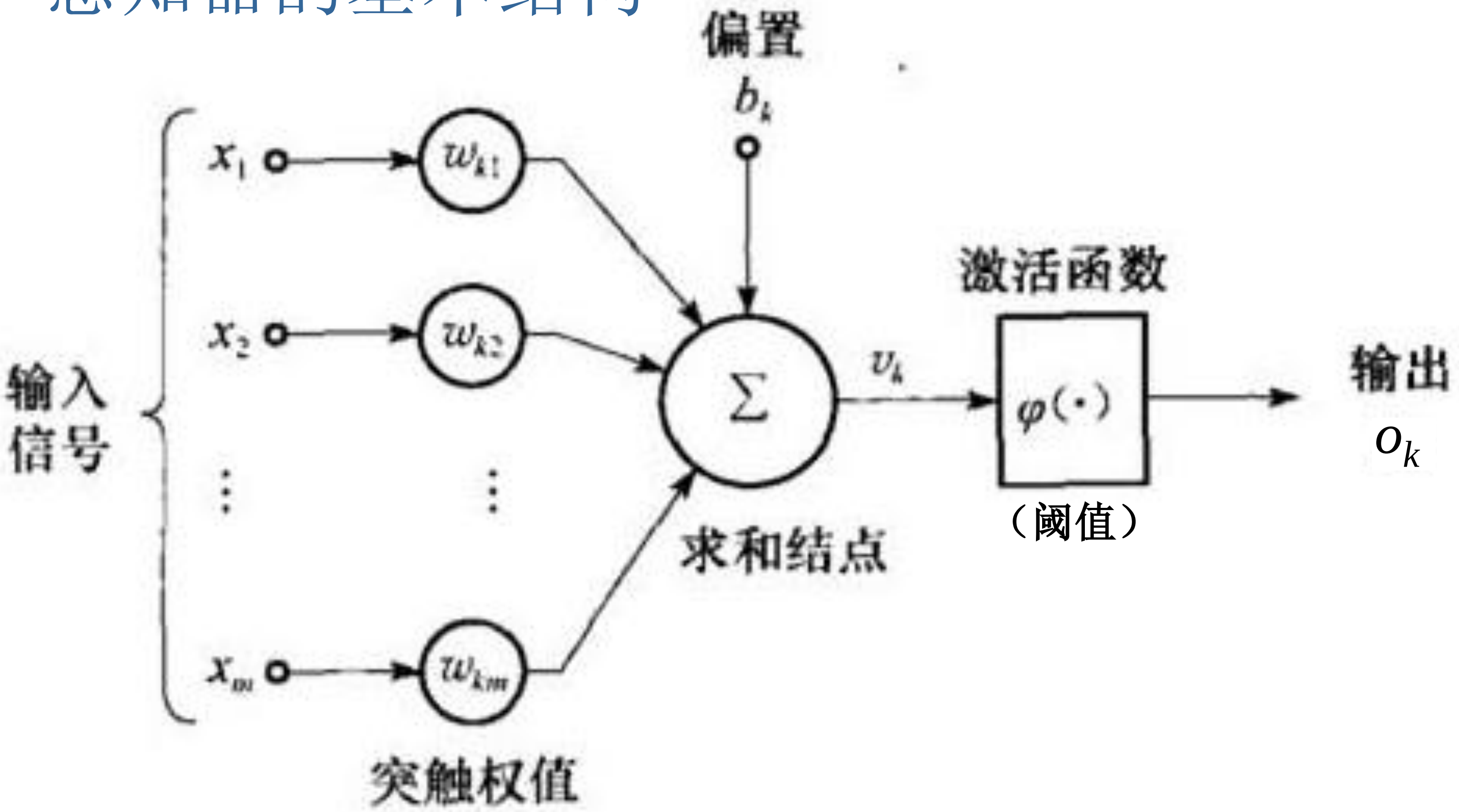
- **感知准则函数**
  - 对于规范化的增广样本集
    - $a^T y_i < 0$——错误分类
  - 定义感知准则函数，作为优化准则函数

$$\min J_p(a) = \sum_{y \in Z_E} (-a^T y)$$

  - 求解向量（或解区）

Artificial Neural Networks

- 感知器的基本结构



Artificial Neural Networks

- 感知器训练方法
  - 单个感知器的学习任务：决定一个权向量，它可以使感知器对于给定的训练样例输出正确的1或-1。
  - 感知器训练的方法有：感知器法则、delta法则，这两种方法是训练多层神经网络的基础。

Artificial Neural Networks

- 感知器法则
- 算法过程：
  - 随机选取权值
  - 反复应用这个感知器到每个训练样例，只要它误分类样例就修改感知器的权值：$w_i \leftarrow w_i + \Delta w_i \quad \Delta w_i = \eta\,(t - o)x_i$
    - 其中：η为学习速率、t为当前样本目标输出、o为感知器输出。
  - 重复这个过程，直到感知器正确分类所有的训练样例。
  - 如果训练样例线性可分，并且使用充分小的η，感知器法则会收敛到一个能正确分类所有训练样例的权向量

Artificial Neural Networks

- 为什么这个更新法则会成功收敛到正确的权值呢？为了得到直观的感觉，考虑一些特例。
  - 假定训练样本已被感知器正确分类。这时，(t-o)是0，这使Δwi为0，所以没有权值被修改。
  - 而如果当目标输出是+1时感知器输出一个-1，这种情况为使感知器输出一个+1而不是-1，权值必须被修改以增大Δwi的值(x恒正) 。
    - 例如，如果xi>0，那么增大wi会使感知器更接近正确分类这个实例。
      - 如果xi=0.8，η=0.1，t=1，并且o= -1，那么权更新就是Δwi =η(t-o)xi=0.1(1-(-1))0.8=0.16。
  - 另一方面，如果t=-1而o=1，那么和正的xi关联的权值会被减小而不是增大。

Artificial Neural Networks

- 例：训练数据集中的正例为$x_1=(3,3)'$, $x_2=(4,3)'$, 反例为$x_3=(1,1)'$, 试用感知器法则求感知器模型$f(x)=sign(w \cdot x)$。$w=(w_{(0)}, w_{(1)}, w_{(2)})'$, $\underline{x}=(1,x_{(1)},x_{(2)})'$,($\eta=1$)
- 解：
  - 取初值$w^{(0)}=0$
  - 对$x_1=(3,3)'$, $w^{(0)} \cdot \underline{x}_1=0$，未能被正确分类
    - 更新$w^{(0)}$， $w^{(1)} = w^{(0)} + \eta(t-o)\underline{x}_1=(2,6,6)'$
  - 对$x_1=(3,3)'$, $x_2=(4,3)'$,显然 $w^{(1)} \cdot \underline{x}_1>0$， $w^{(1)} \cdot \underline{x}_2>0$，被正确分类，无需更新$w^{(1)}$
  - 对$x_3=(1,1)'$, $w^{(1)} \cdot \underline{x}_3>0$， 被误分类
    - 更新$w^{(1)}$， $w^{(2)} = w^{(1)} + \eta(t-o)\underline{x}_3=(0,4,4)'$
  - 如此下去，最终得感知器模型为$f(x)=sign(2x_{(1)}+2x_{(2)}-6)$

正例为$x_1=(3,3)'$, $x_2=(4,3)'$, 反例为$x_3=(1,1)'$

Artificial Neural Networks

迭代次数	误分类点	**w**	**w·<u>x</u>**
0	—	$(0,0,0)'$	0
1	x_1	$(2,6,6)'$	$6x_{(1)}+6x_{(2)}+2$
2	x_3	$(0,4,4)'$	$4x_{(1)}+4x_{(2)}$
3	x_3	$(-2,2,2)'$	$2x_{(1)}+2x_{(2)}-2$
4	x_3	$(-4,0,0)'$	-4
5	x_1	$(-2,6,6)'$	$6x_{(1)}+6x_{(2)}-2$
6	x_3	$(-4,4,4)'$	$4x_{(1)}+4x_{(2)}-4$
7	x_3	$(-6,2,2)'$	$2x_{(1)}+2x_{(2)}-6$
8	无	$(-6,2,2)'$	$2x_{(1)}+2x_{(2)}-6$

Artificial Neural Networks

- 尽管当训练样例线性可分时，感知器法则可以成功地找到一个权向量，但如果样例不是线性可分时它将不能收敛。因此，人们设计了另一个训练法则来克服这个不足，称为delta法则（delta rule）。
- 如果训练样本不是线性可分的，那么delta法则会收敛到目标概念的最佳近似。
- delta法则的关键思想是使用梯度下降（gradient descent）来搜索可能权向量的假设空间，以找到最佳拟合训练样例的权向量。

Artificial Neural Networks

- 考虑简单的线性单元

$$o = w_0 + w_1 x_1 + ... + w_n x_n$$

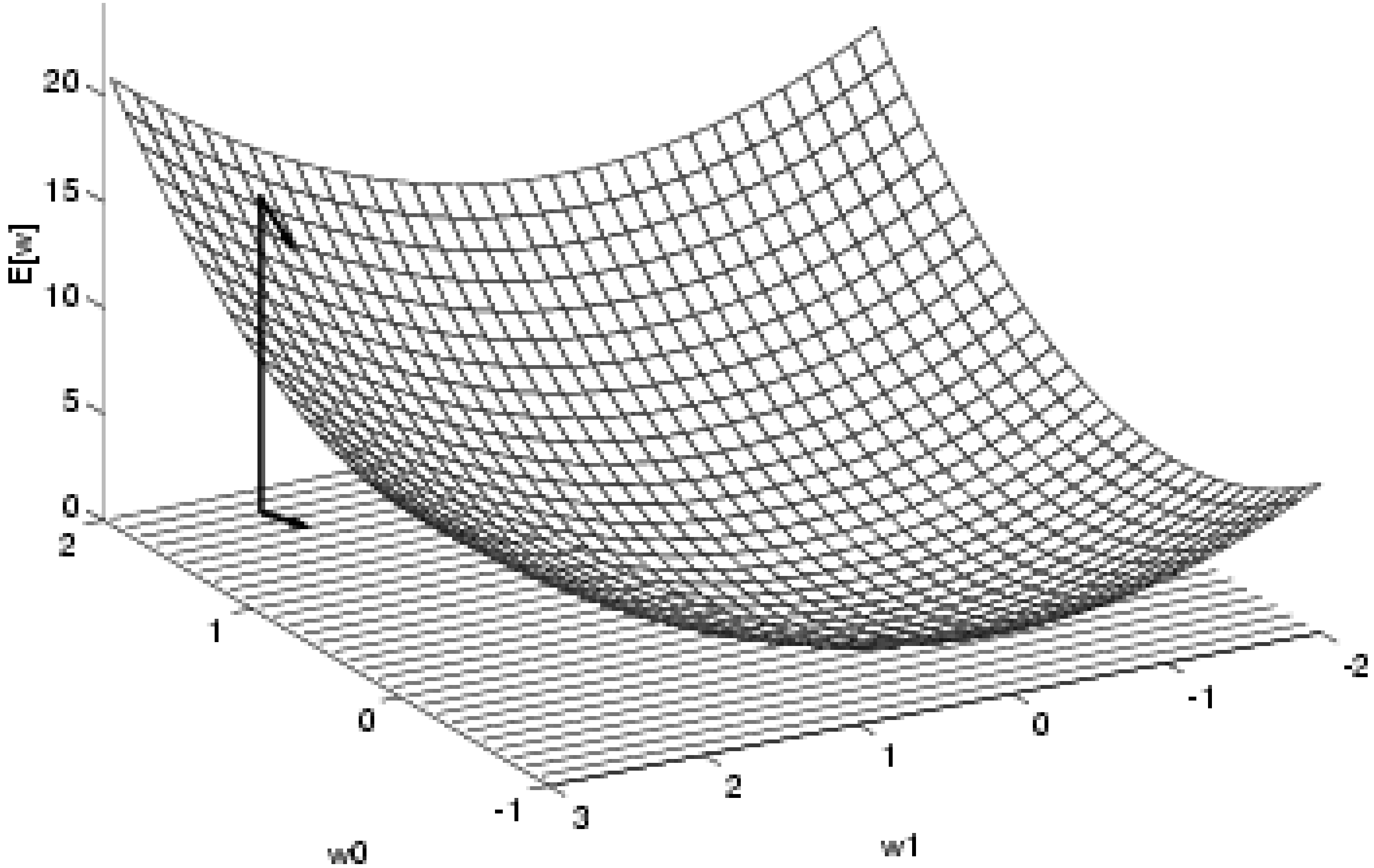
- 为推导权值学习法则，先要确定一个度量标准来衡量假设（权向量）相对于训练样例的训练误差（training error）。一个常用的度量标准为：

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

  - 其中D是训练样例集合，td是训练样例d的目标输出，od是线性单元对训练样例d的输出。

Artificial Neural Networks

- 包含可能权向量的整个假设空间和与它们相关联的E值。



Artificial Neural Networks

- 梯度下降法则的推导
  - 可以通过计算E相对向量的每个分量的导数来得到沿误差曲面最陡峭下降的方向。这个向量导数被称为E对于w 的梯度，记作∇E[w]

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Artificial Neural Networks

$$E[\vec{w}] \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d})
\end{aligned}$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

其中x_{id}表示训练样例d的第i个输入分量，o_d表示感知器输出，t_d表示训练样例的目标值

梯度下降权值更新法则： $\Delta w_i = \eta \sum_{d\in D}(t_d - o_d)x_{id}$

Artificial Neural Networks

- 修改后的训练法则与之前给出的相似，只是在迭代计算每个训练样例时根据下面的公式来更新权值

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

$$\Delta w_i = \eta (t - o) x_i$$

与感知器法则相同否？

其中t，o，和xi分别是目标值、单元输出和第i个训练样例的输入。

训练法则被称为增量法则（delta rule），或有时叫LMS法则（least-mean-square最小均方）、Adaline法则、或Windrow-Hoff法则（以它的发明者命名）。

Artificial Neural Networks

- 两种算法的区别
  - 权值更新
    - 感知器训练法则根据阈值化（thresholded）的感知器输出的误差更新权值
    - 增量法则根据输入的非阈值化（unthresholded）线性组合的误差来更新权
  - 收敛特性
    - 感知器训练法则经过有限次的迭代收敛到一个能理想分类训练数据的假设，但条件是训练样例线性可分。
    - 增量法则渐近收敛到最小误差假设，可能需要无限的时间，但无论训练样例是否线性可分都会收敛。

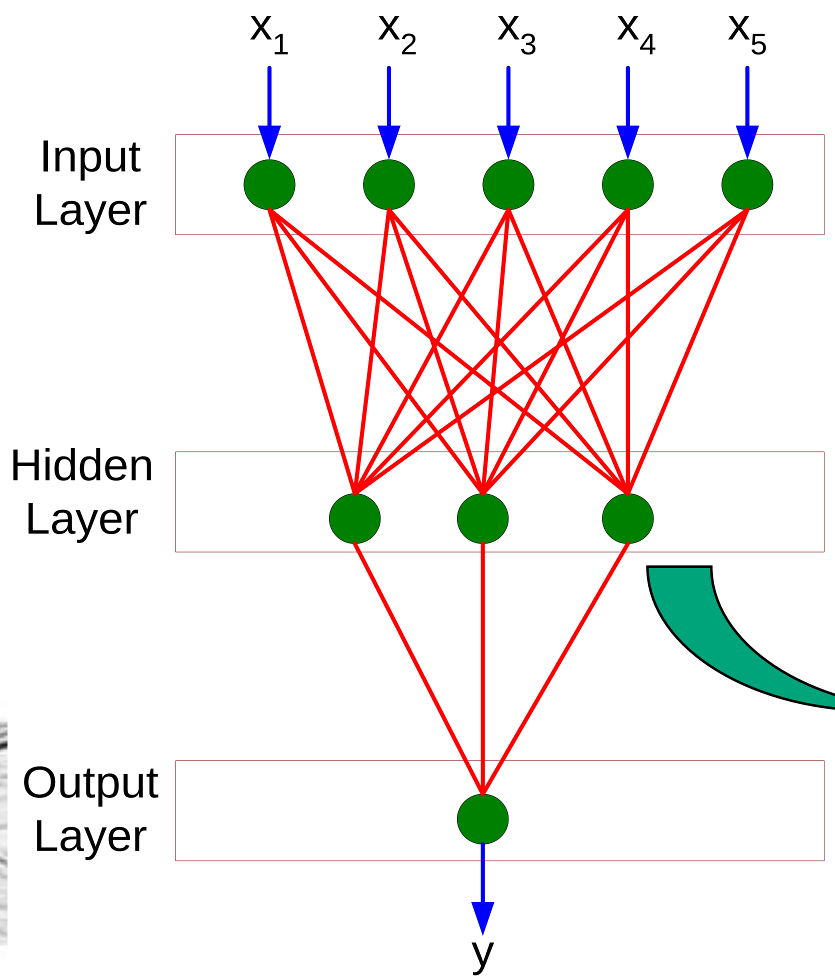
多层网络和反向传播算法

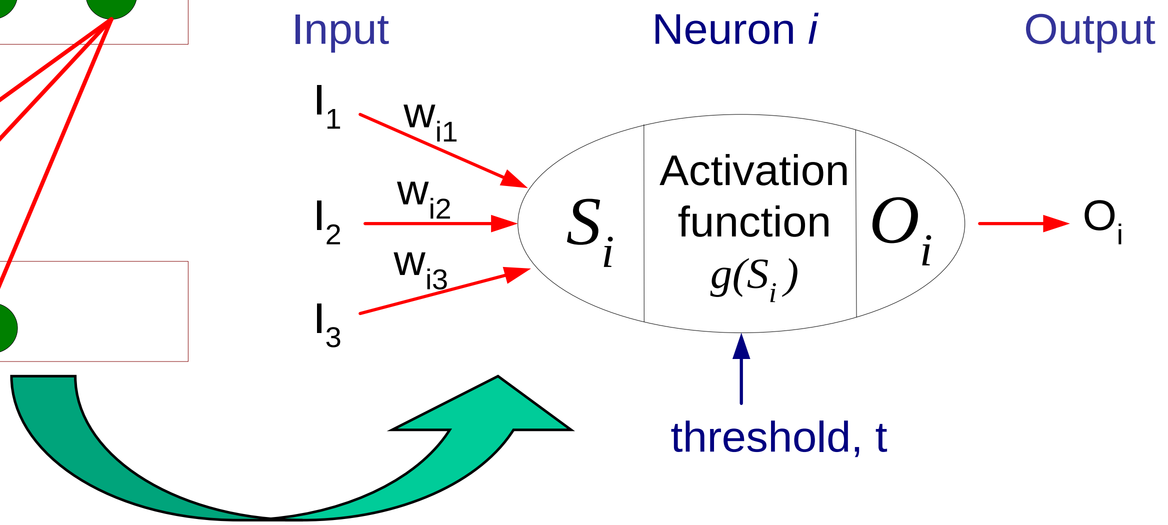
Artificial Neural Networks

- 多层感知机MLP和反向传播算法BP的区别？
- MLP是网络模型，BP是优化算法
- MLP可以很多层，但受限于激活函数，用BP优化时通常只有3层

Artificial Neural Networks

- 多层网络，多层感知器（MLP）





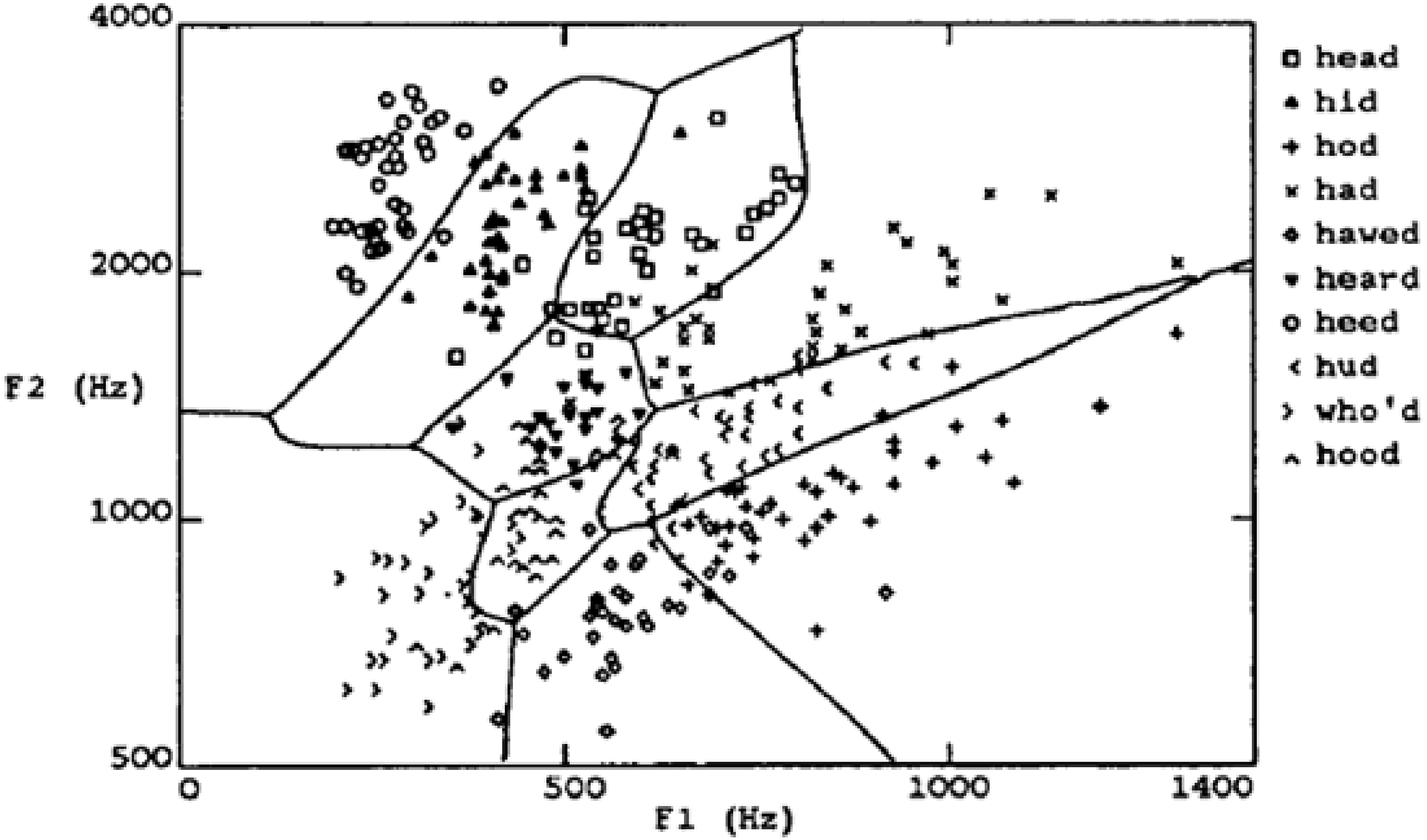
Training ANN means learning the weights of the neurons

Artificial Neural Networks

- 单个感知器仅能表示线性决策面，而多层网络能够表示种类繁多的非线性曲面。
- 核心是可微阈值单元（即神经元）
  - 多个线性单元的连接仍产生线性函数，但我们希望构建表征非线性函数的网络
  - 修改感知器单元确实可以构建非线性函数，但它的不连续阈值使它不可微（主要指感知器法则），不适合梯度下降算法
  - 神经元需要满足的条件
    - 输出是输入的非线性函数
    - 输出是输入的可微函数
  - 解决方案：引入Sigmoid单元，类似于感知器单元，但基于一个平滑的可微阈值函数

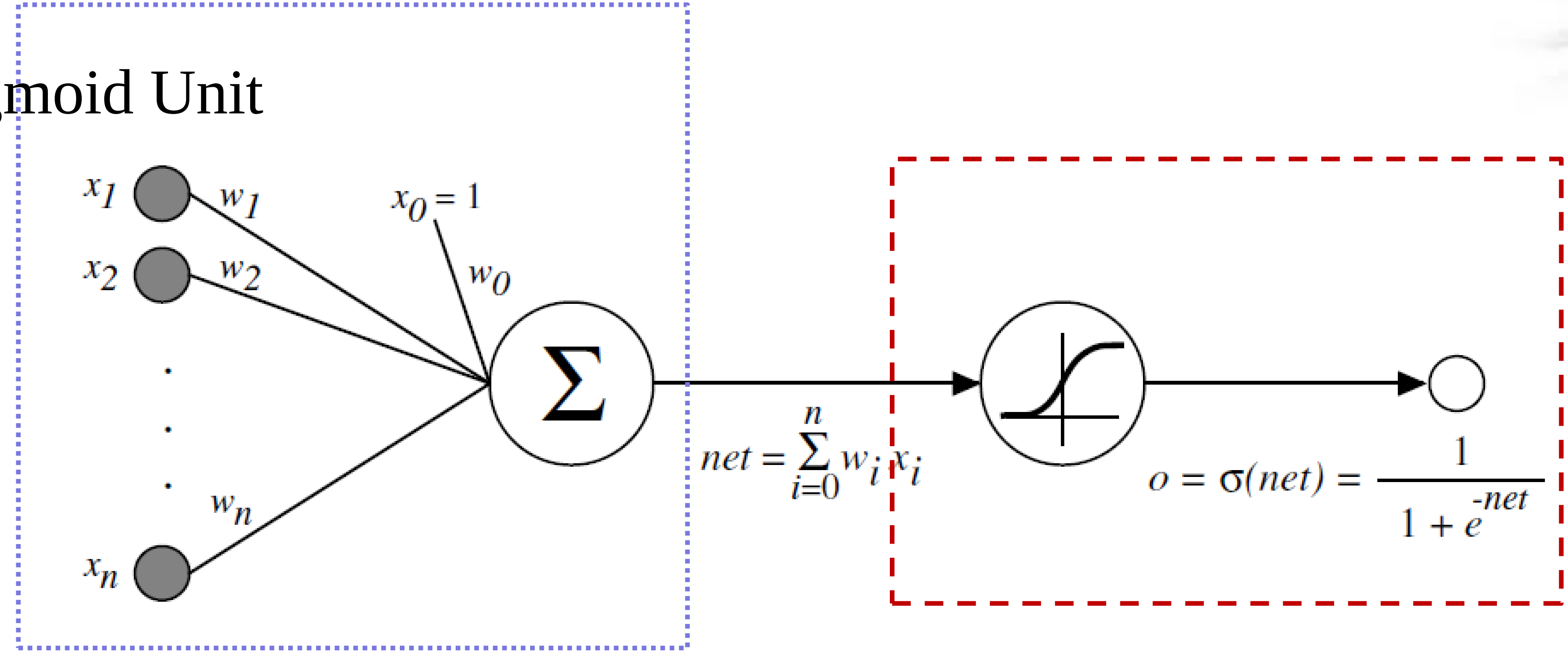
Artificial Neural Networks

- 语音识别任务：区分出现10种元音



Artificial Neural Networks

- Sigmoid Unit

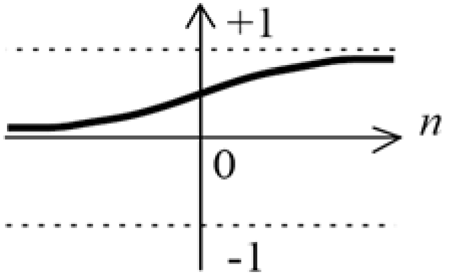


$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

- 非线性、可微

Artificial Neural Networks

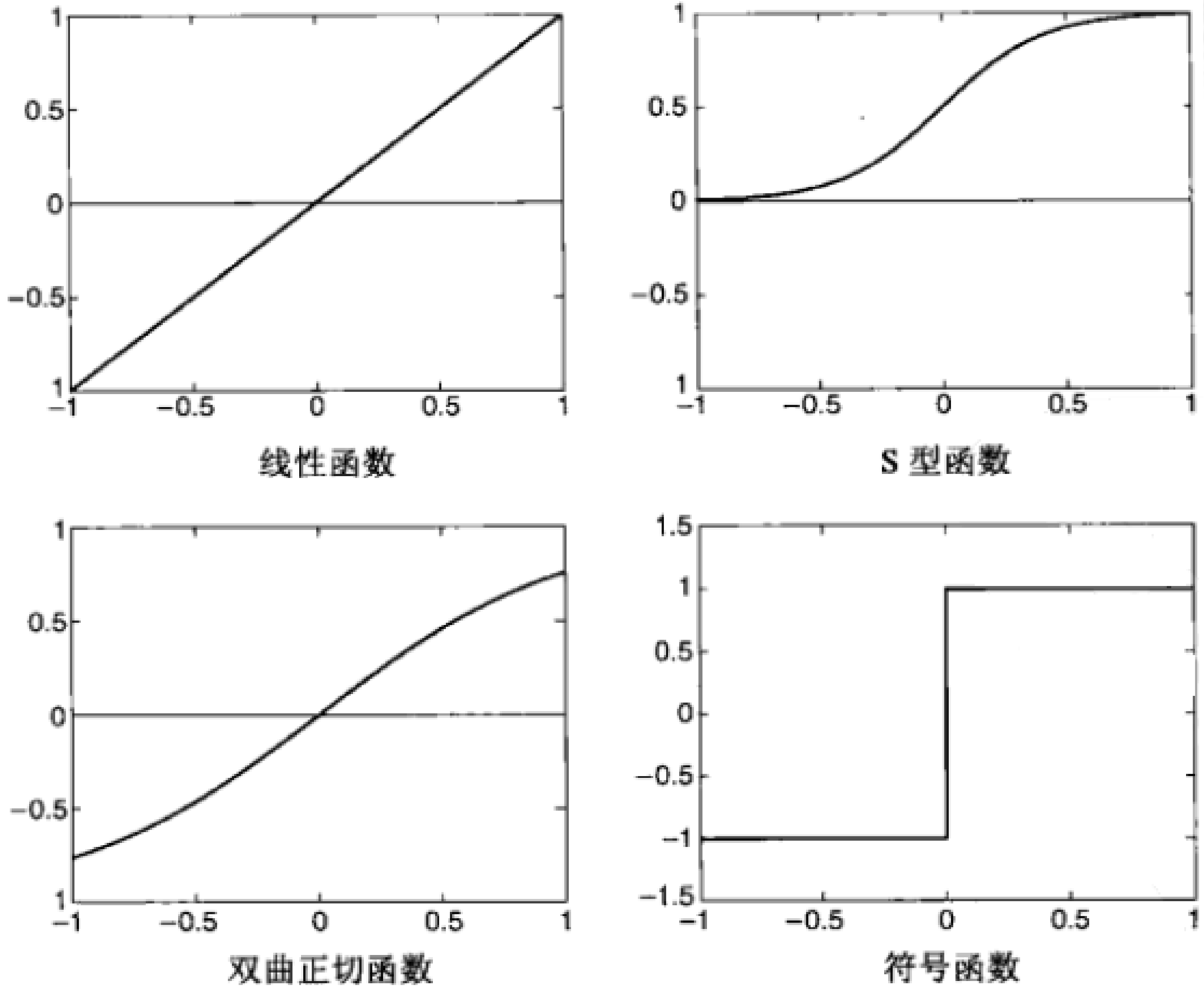


- sigmoid函数
  - 也称logistic函数、sigmoid单元的挤压函数（squashing function）
  - 输出范围是0到1
  - 随输入单调递增
  - 导数很容易以它的输出表示

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

Artificial Neural Networks

- Sigmoid函数被称为激活函数



人工神经网络中激活函数的类型

Artific

- 一个sigmoid单元的梯度下降法则

$$E[\vec{w}] \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right) \\
&= -\sum_d (t_d - o_d)\boxed{\frac{\partial o_d}{\partial net_d}\frac{\partial net_d}{\partial w_i}}
\end{aligned}$$

net_d ： 网络输出

o_d ： 激活函数输出

相比于感知机，这里增加了激活函数

Artificial Neural Networks

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D}(t_d - o_d)o_d(1 - o_d)x_{i,d}$$

$$\Delta w_i = -\eta \frac{\partial E_d}{\partial w_i} = \eta \sum_{d \in D}(t_d - o_d)o_d(1 - o_d)x_{i,d}$$

Artificial Neural Networks

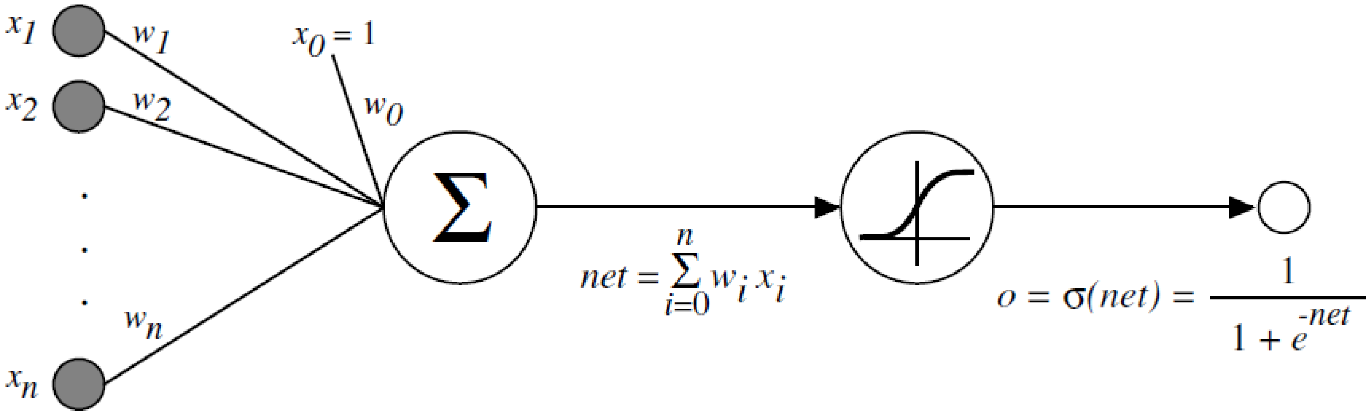
- 反向传播法则的推导
  - 随机梯度下降算法迭代处理训练样例，每次处理一个，对于每个训练样例d，利用关于这个样例的误差E_d的梯度修改权值

$$E_d(\vec{w}) = \frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

Artificial Neural Networks

- 符号说明
  - x_{ji}： 单元j的第i个输入
  - w_{ji}： 与x_{ji}相关联的权值
  - net_j： 单元j的输入的加权和
  - o_j： 单元j计算出的输出
  - t_j： 单元j的目标输出
  - σ： sigmoid函数
  - outputs： 网络最后一层的输出单元的集合
  - Downstream(j)： 单元j的输出所能到达单元的集合

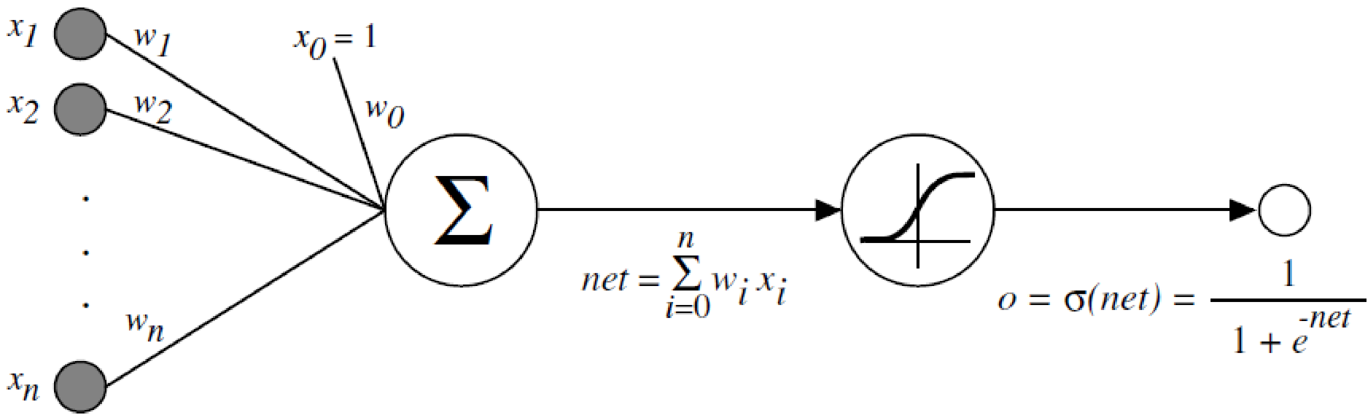


Artificial Neural Networks

– 随机梯度下降法则的推导
  - 注意权值wji仅能通过netj影响网络的其他部分

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$

– 下面我们分情况讨论
  - 单元j是网络的一个输出单元
  - 单元j是网络的一个隐藏单元



Artificial Neural Networks

- 输出单元的权值训练法则
  - netj仅能通过oj影响其余的网络

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2}(t_j - o_j)^2$$

$$= \frac{1}{2} 2(t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j)$$

Artificial Neural Networks

$$o_j = \sigma(net_j)$$

sigmoid函数的导数为σ(netj)(1-σ(netj))

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = o_j(1-o_j)$$

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$= -(t_j - o_j)o_j(1-o_j)$$

Artificial Neural Networks

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j)$$

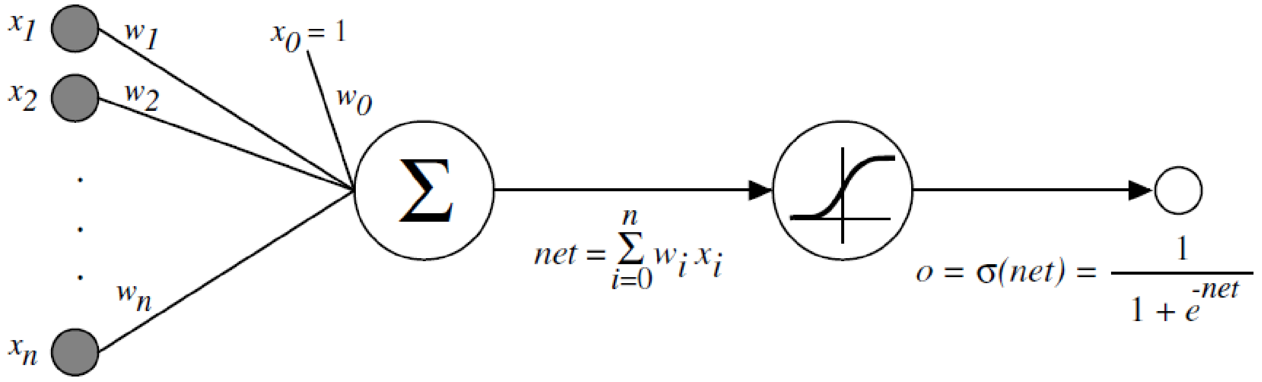
– 最终得到输出单元的随机梯度下降法则：

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta x_{ji} (t_j - o_j) o_j (1 - o_j)$$

Artificial Neural Networks

- 隐藏单元的权值训练法则
  - 对于网络中的内部单元或者说隐藏单元的情况，推导wji必须考虑wji间接地影响网络输出，从而影响Ed。
  - netj只能通过Downstream(j)中的单元影响网络输出（再影响Ed）

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$



*Downstream(j)：单元j的输出所能到达单元的集合

Artificial Neural Networks

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j (1 - o_j)$$

$$= -o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

$$\delta_k = -\frac{\partial E_d}{\partial net_k}$$

链式法则

$$\delta_j = o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

$$\Delta w_{ji} = \eta x_{ji} o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

–隐含单元的随机梯度下降法则

Artifici

delta法则 $\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$ BP算法：

输出单元 $\Delta w_{ki} = \eta x_{ki} (t_k - o_k) o_k (1 - o_k)$

隐藏单元 $\Delta w_{hi} = \eta x_{hi} o_h (1 - o_h) \sum_{k \in Downstream(j)} \delta_k w_{kh}$

- 反向传播算法与delta训练法则很相似，均是依照以下三者的乘积来更新每一个权值w：学习速率η、该权值应用的输入值xji、这个单元输出的误差。
- 区别：反向传播算法与delta法则不同点是误差项被替换成一个更复杂的误差项δ_j。
- 输出单元k的误差项
  - δ_k与delta法则中的(t_k-o_k)相似，但乘上了sigmoid函数的导数$o_k(1-o_k)$。
- 隐藏单元h的误差项
  - 因为训练样例仅对网络的输出提供了目标值t_k，所以缺少直接的目标值来计算隐藏单元的误差值，采取以下的间接方法计算隐藏单元的误差项：对受隐藏单元h影响的每一个单元的误差δ_k进行加权求和，每个误差δ_k权值为w_{kh}，w_{kh}就是从隐藏单元h到输出单元k的权值。这个权值刻画了隐藏单元h对于输出单元k的误差应负责的程度。

Artificial Neural Networks

- BP可以学习任意的无环网络
  - 算法可以简单地推广到任意深度的无环前馈网络
  - 第m层的单元r的δ_r值由更深的第m+1层δ值根据下式计算

$$\delta_r = o_r(1-o_r)\sum_{s\in m+1层} w_{sr}\delta_s$$

  - 将这个算法推广到任何有向无环结构也同样简单，而不论网络中的单元是否被排列在统一的层上，计算任意内部单元的δ的法则是：

$$\delta_r = o_r(1-o_r)\sum_{s\in Downstream(r)} w_{sr}\delta_s$$

    - Downstream(r)是在网络中单元r的直接下游单元的集合，即输入中包括r的输出的所有单元

Artificial Neural Networks

- 本章小结：
  - 人工神经网络为学习实数值和向量值函数提供了一种实际的方法，对于连续值和离散值的属性都可以使用，并且对训练数据中的噪声具有很好的稳健性。
  - 反向传播算法是最常见的网络学习算法
  - 反向传播算法考虑的假设空间是固定连接的有权网络所能表示的所有函数的空间
  - 包含3层单元的前馈网络能够以任意精度逼近任意函数，只要每一层有足够数量的单元。

Artificial Neural Networks

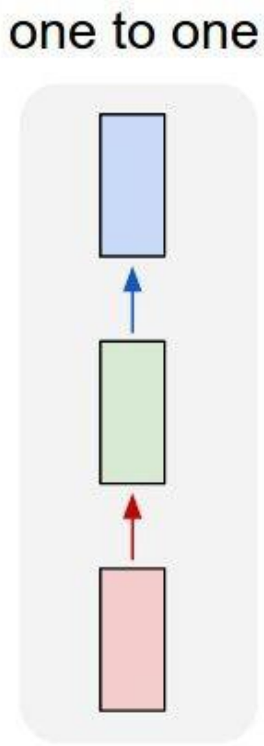
- 反向传播算法使用梯度下降方法搜索可能假设的空间，迭代减小网络的误差以拟合训练数据。
- 梯度下降收敛到训练误差相对网络权值的局部极小值。只要训练误差是假设参数的可微函数，梯度下降可用来搜索很多连续参数构成的假设空间。

实验5：基于BP的信用卡欺诈预测

- 给定信用卡数据，判断是否存在欺诈
- 用BP神经网络算法实现

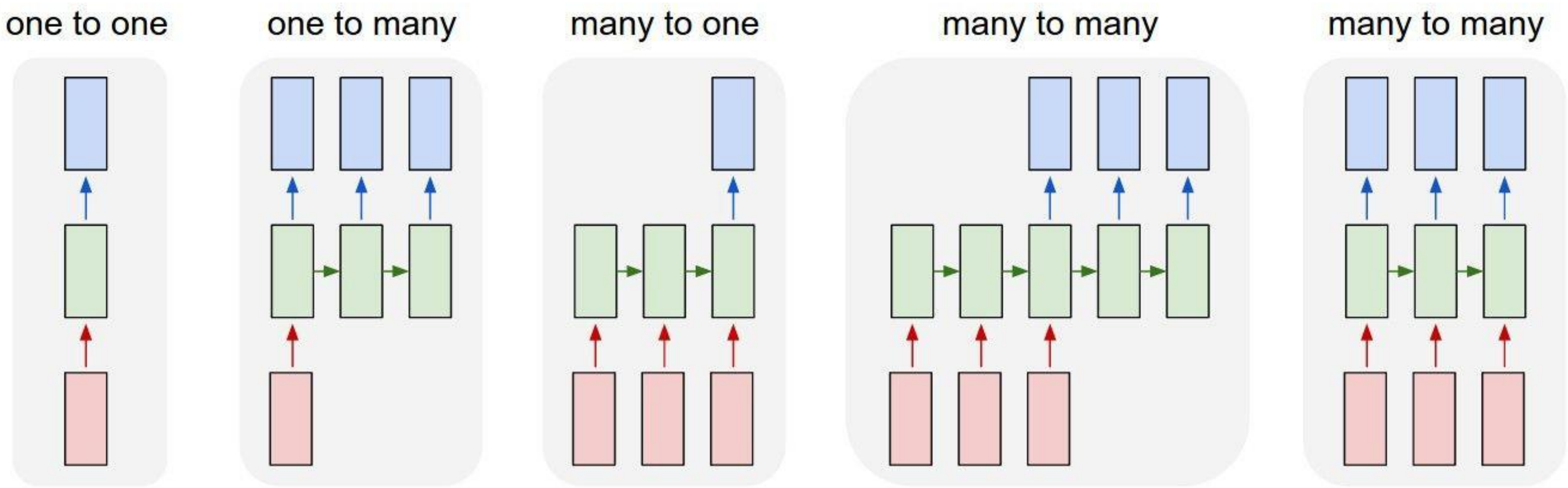
循环神经网络

“Vanilla” Neural Network（普通神经网络）



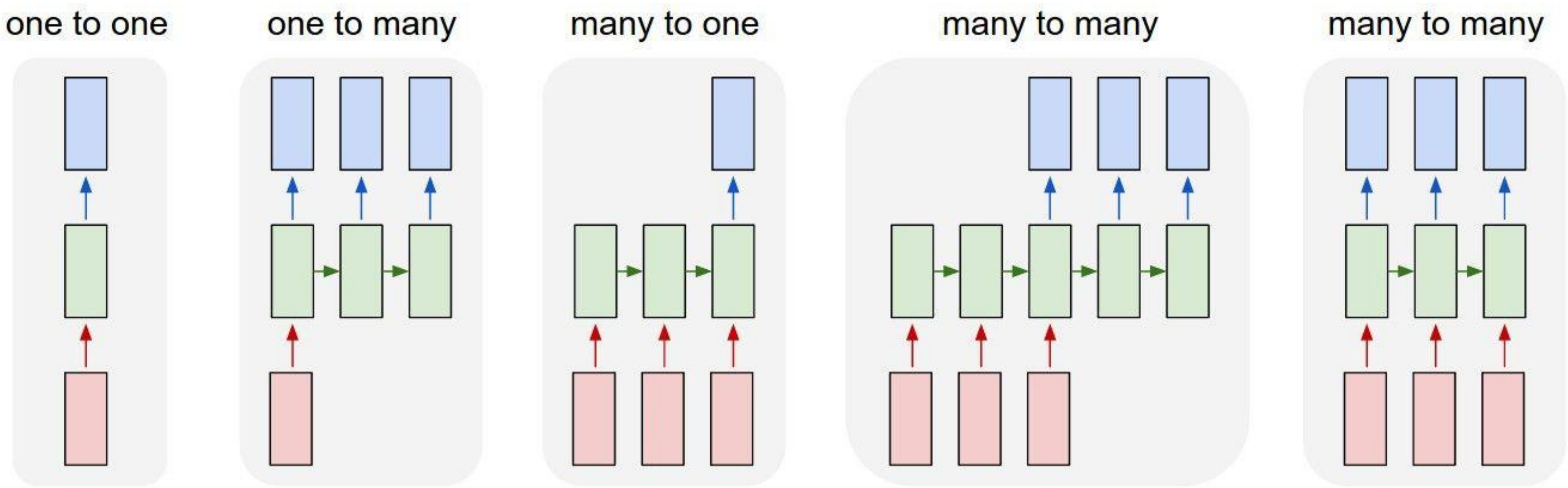
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



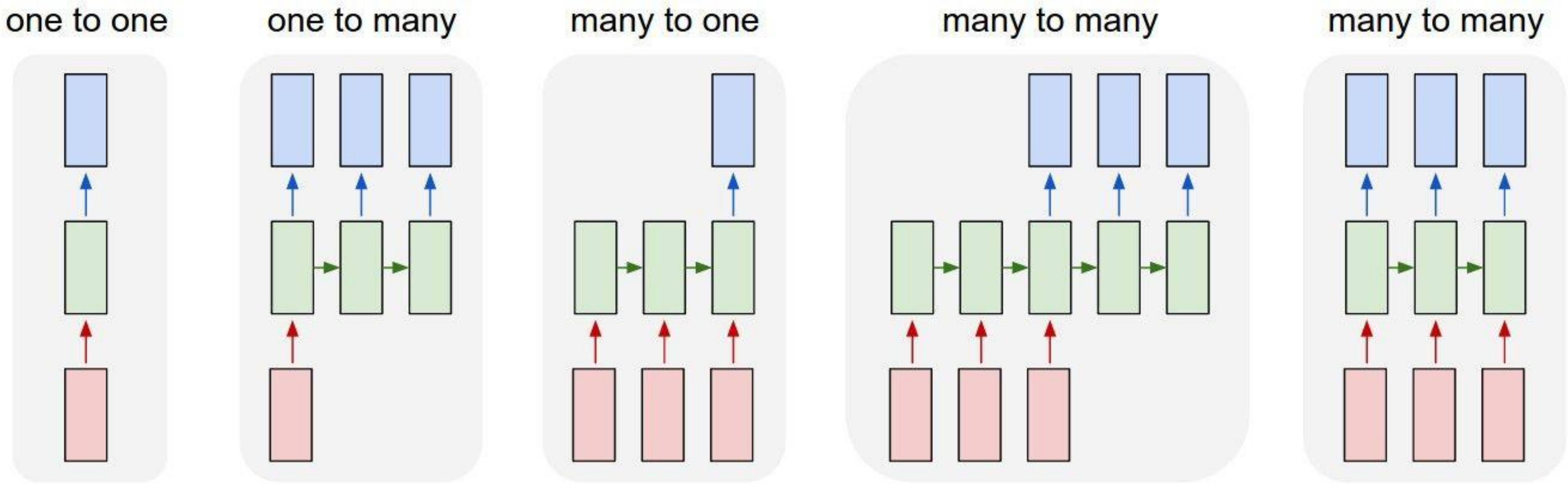
e.g. **Image Captioning（注：输入是一组向量）**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



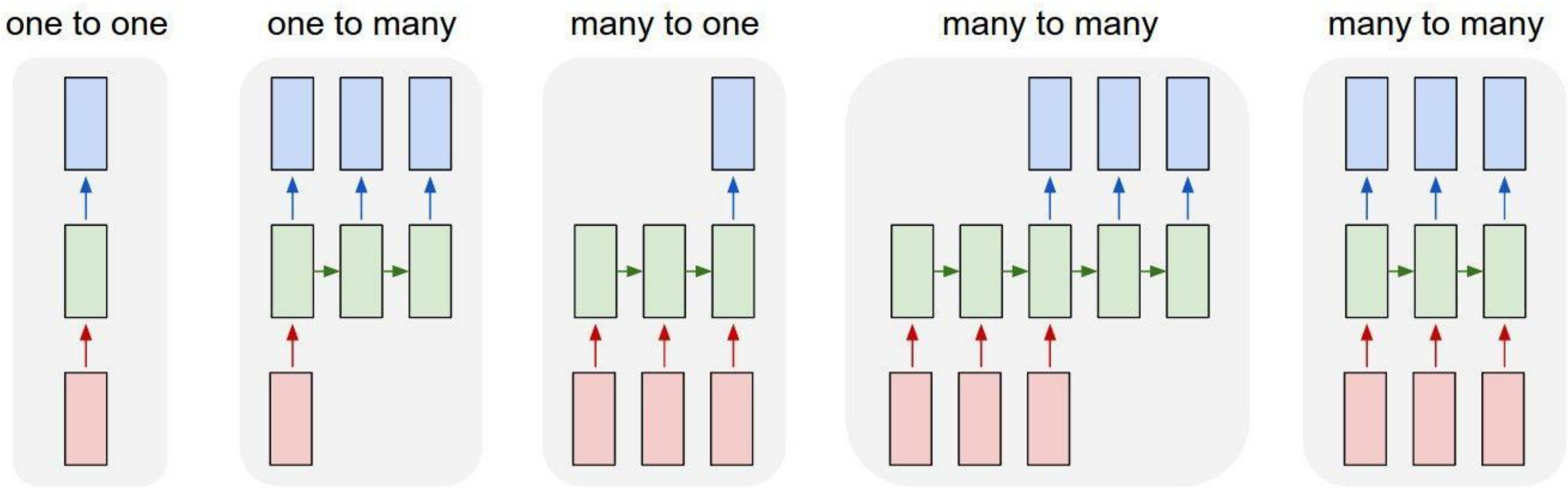
e.g. **Sentiment Classification**
sequence of words -> sentiment（情绪）

Recurrent Neural Networks: Process Sequences



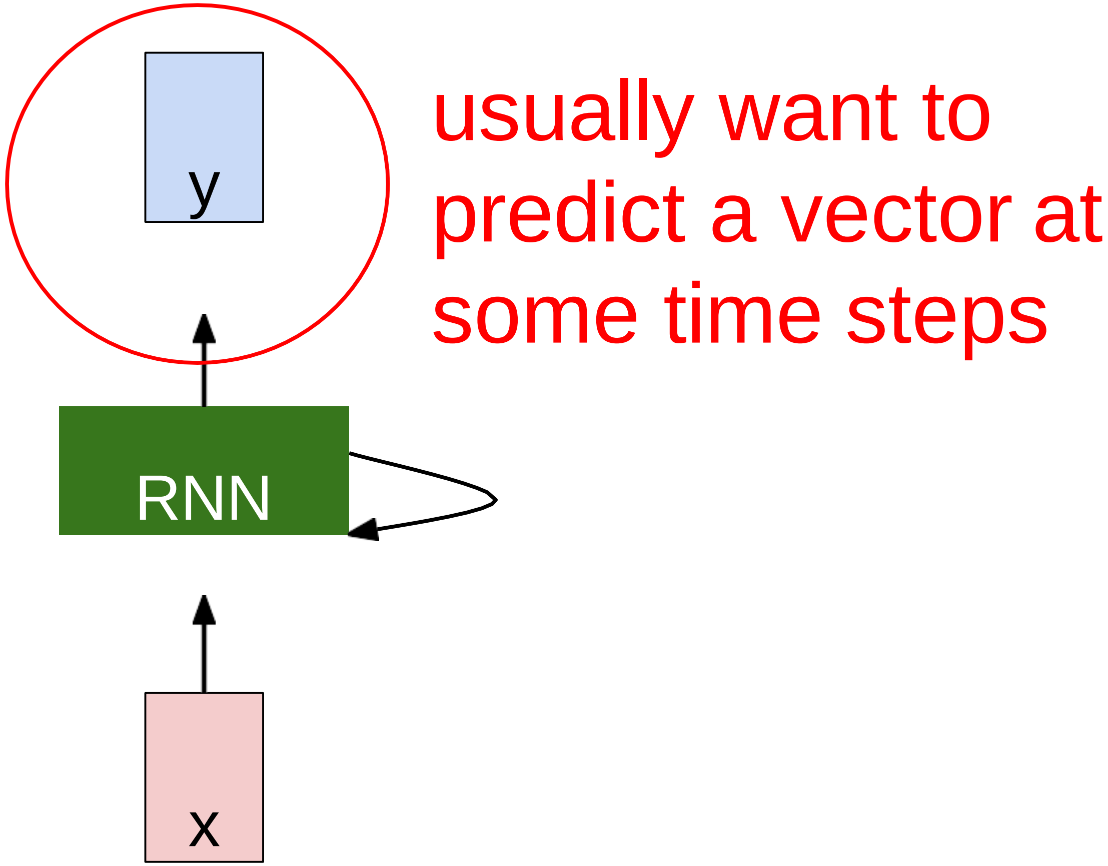
e.g. **台风预测**
序列观测数据->未来时刻台风位置

Recurrent Neural Networks: Process Sequences



e.g. **Video classification on frame level**

Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

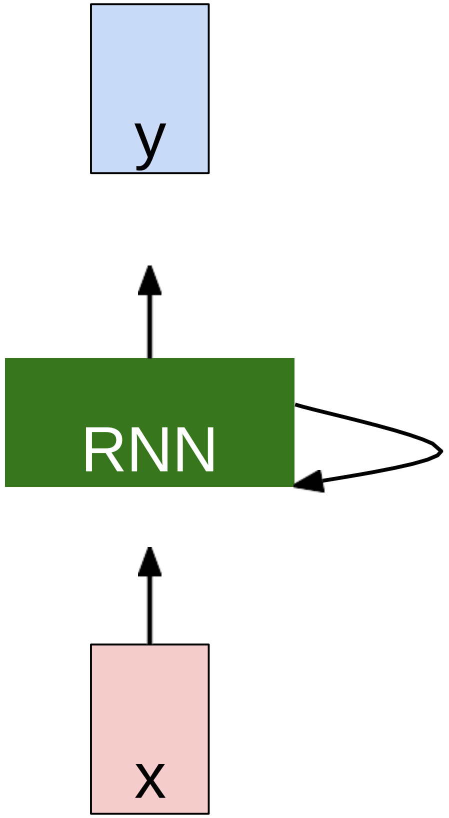
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W

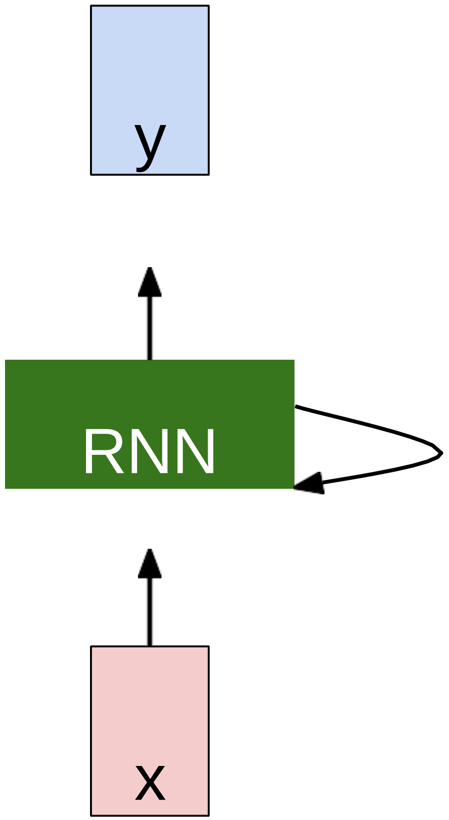


Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

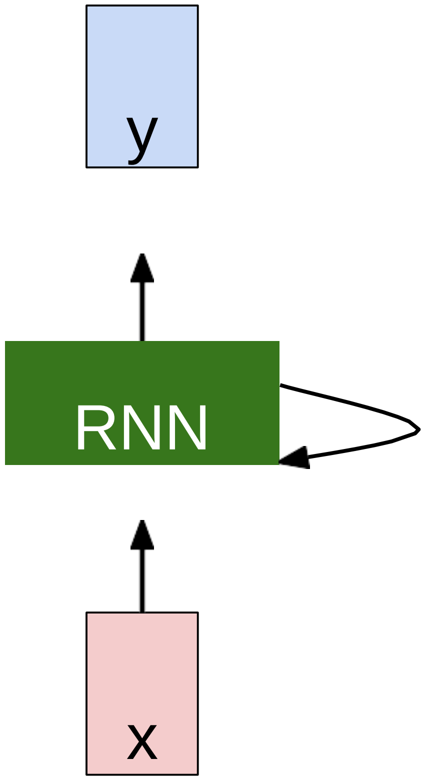
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.



(Vanilla) Recurrent Neural Network

The state consists of a single ***"hidden"*** vector **h**:

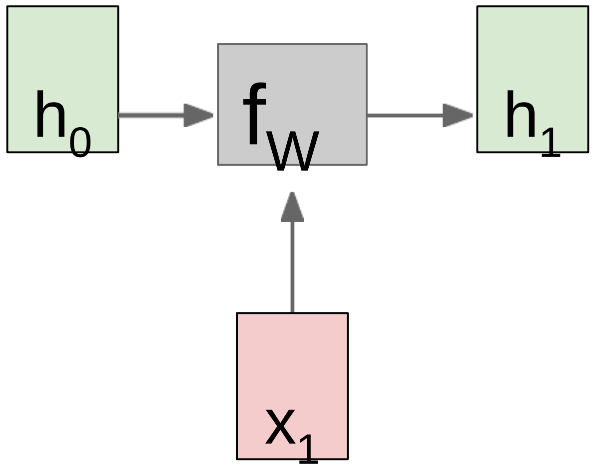


$$h_t = f_W(h_{t-1}, x_t)$$

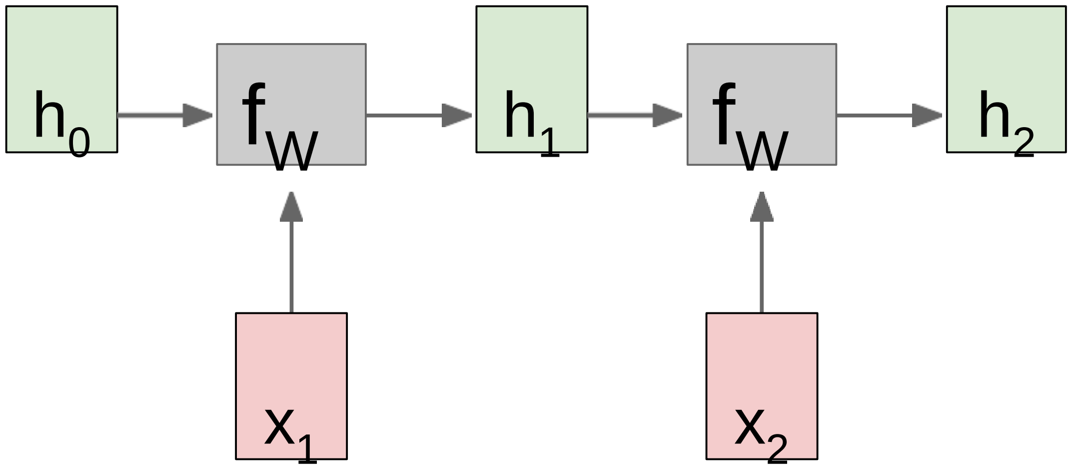
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

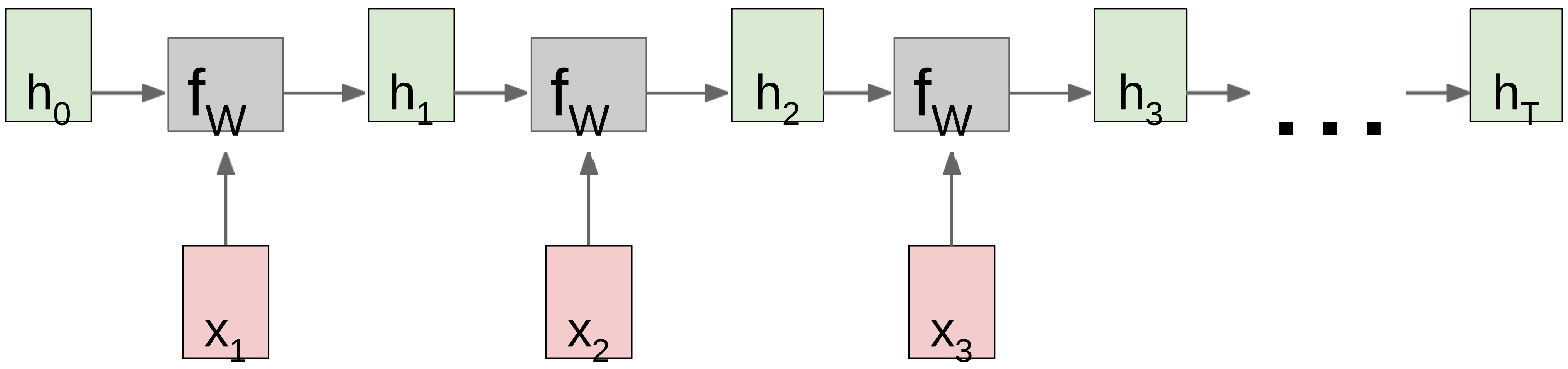
RNN: Computational Graph



RNN: Computational Graph

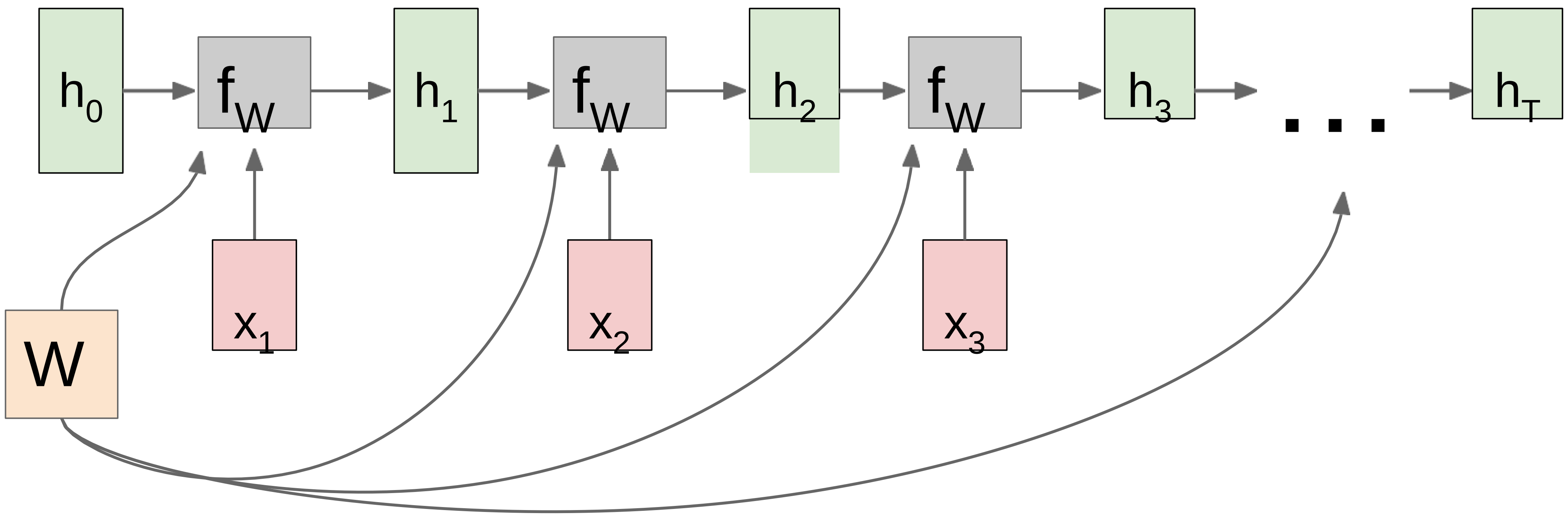


RNN: Computational Graph

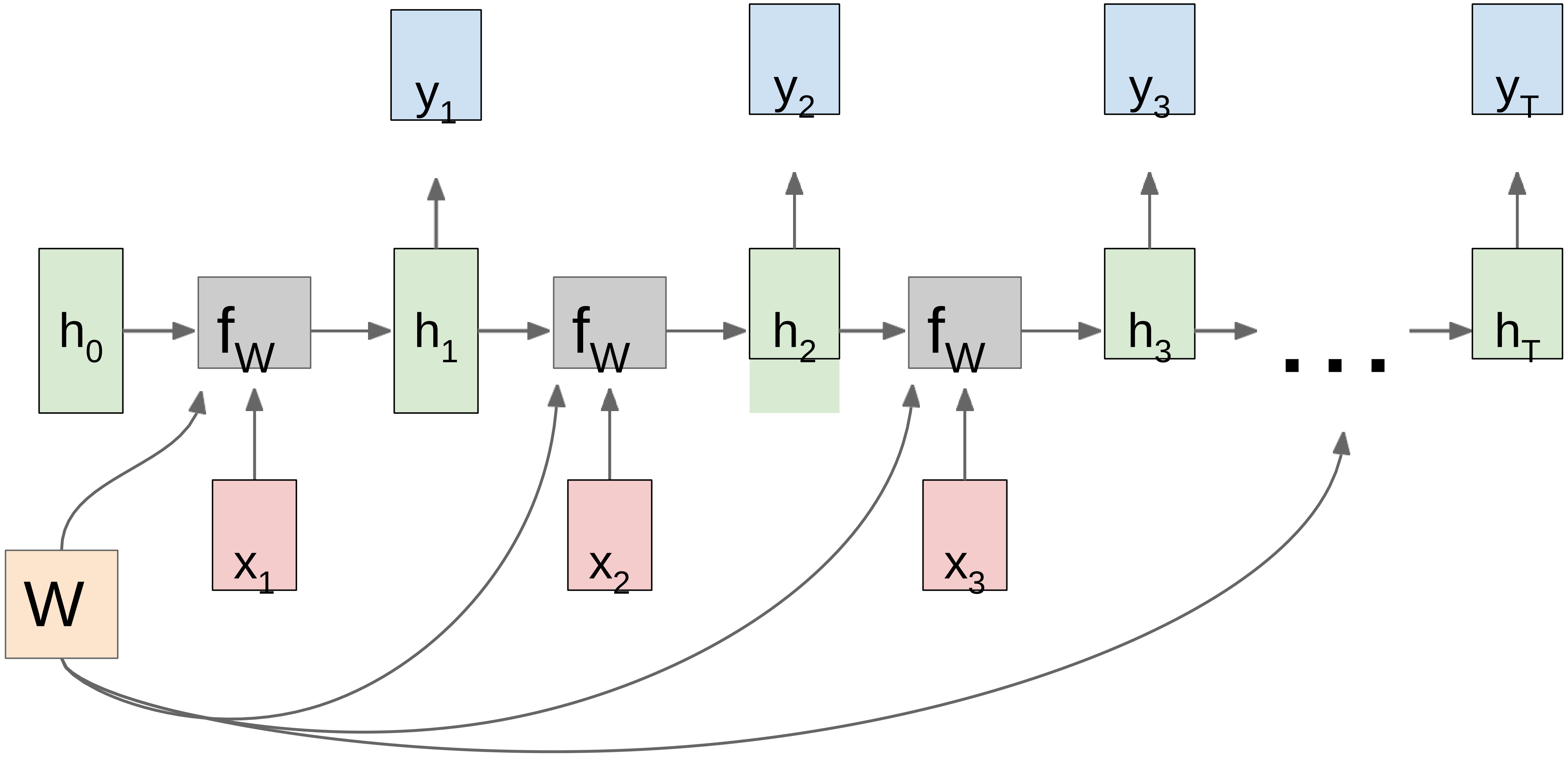


RNN: Computational Graph

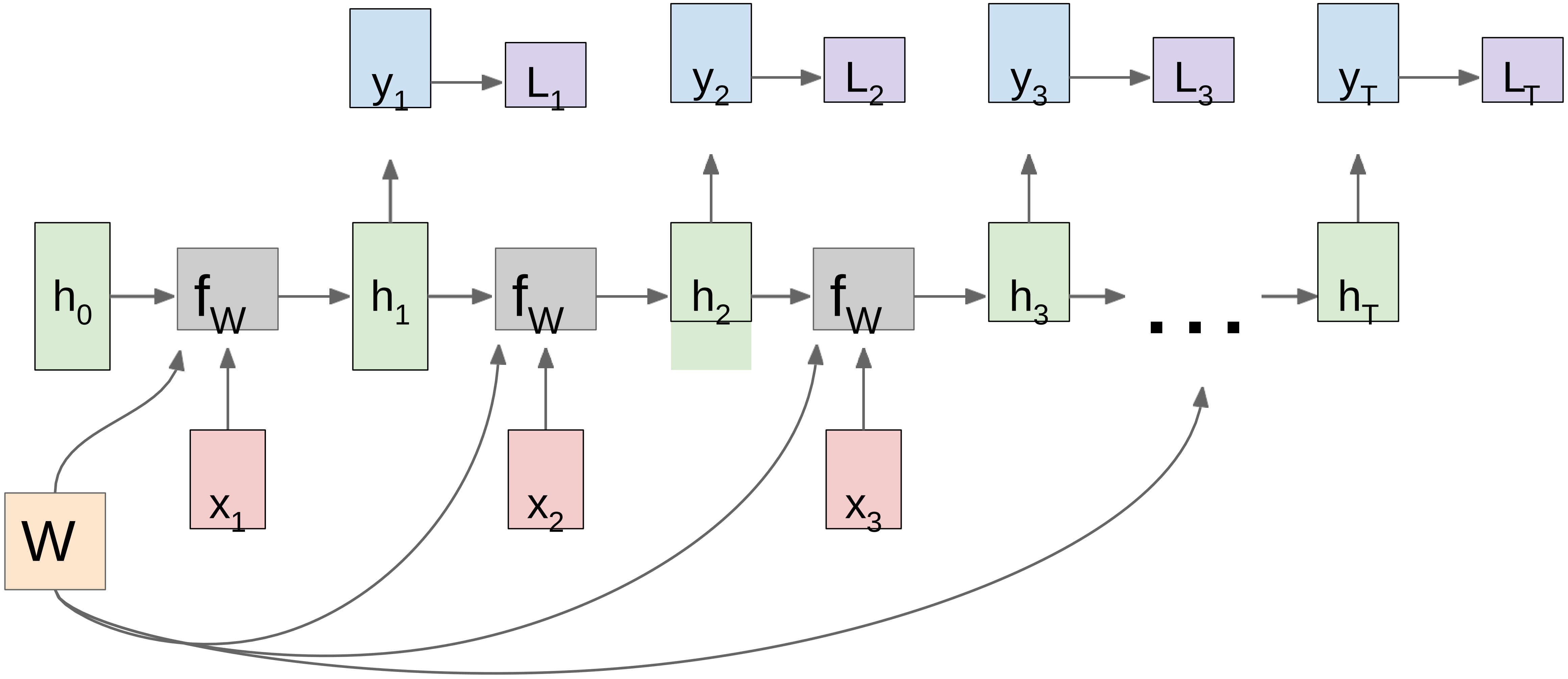
Re-use the same weight matrix at every time-step



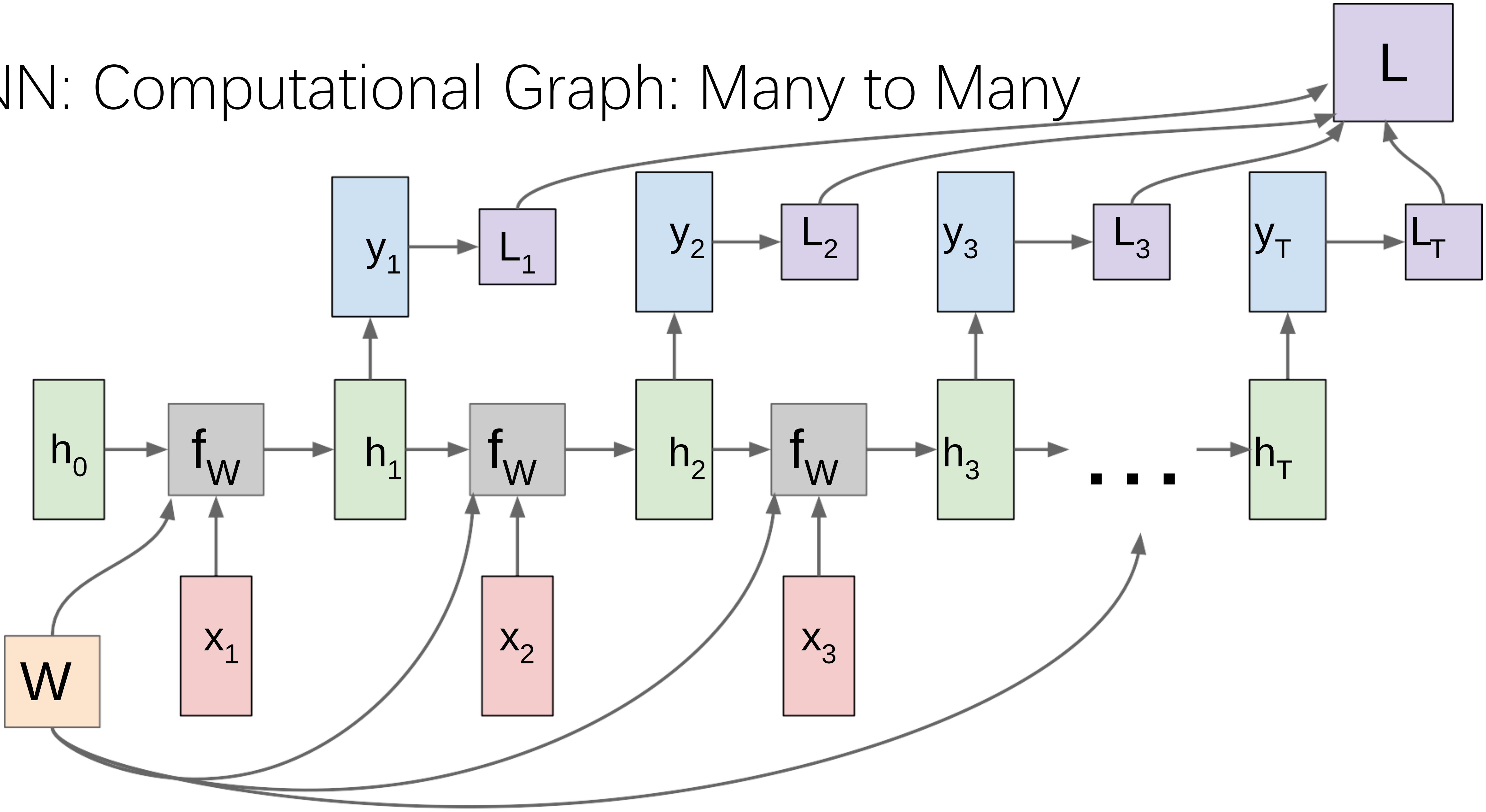
RNN: Computational Graph: Many to Many



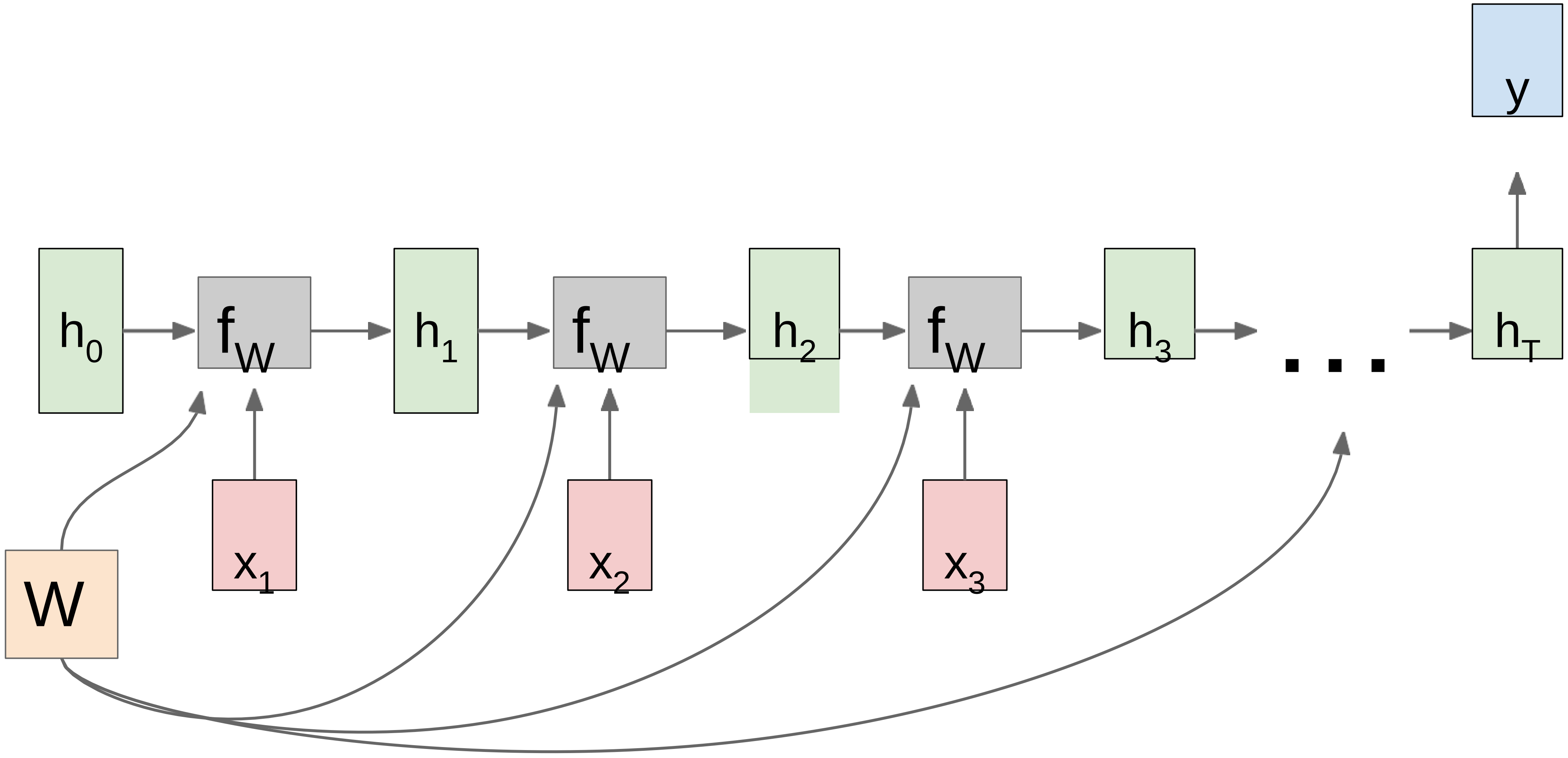
RNN: Computational Graph: Many to Many



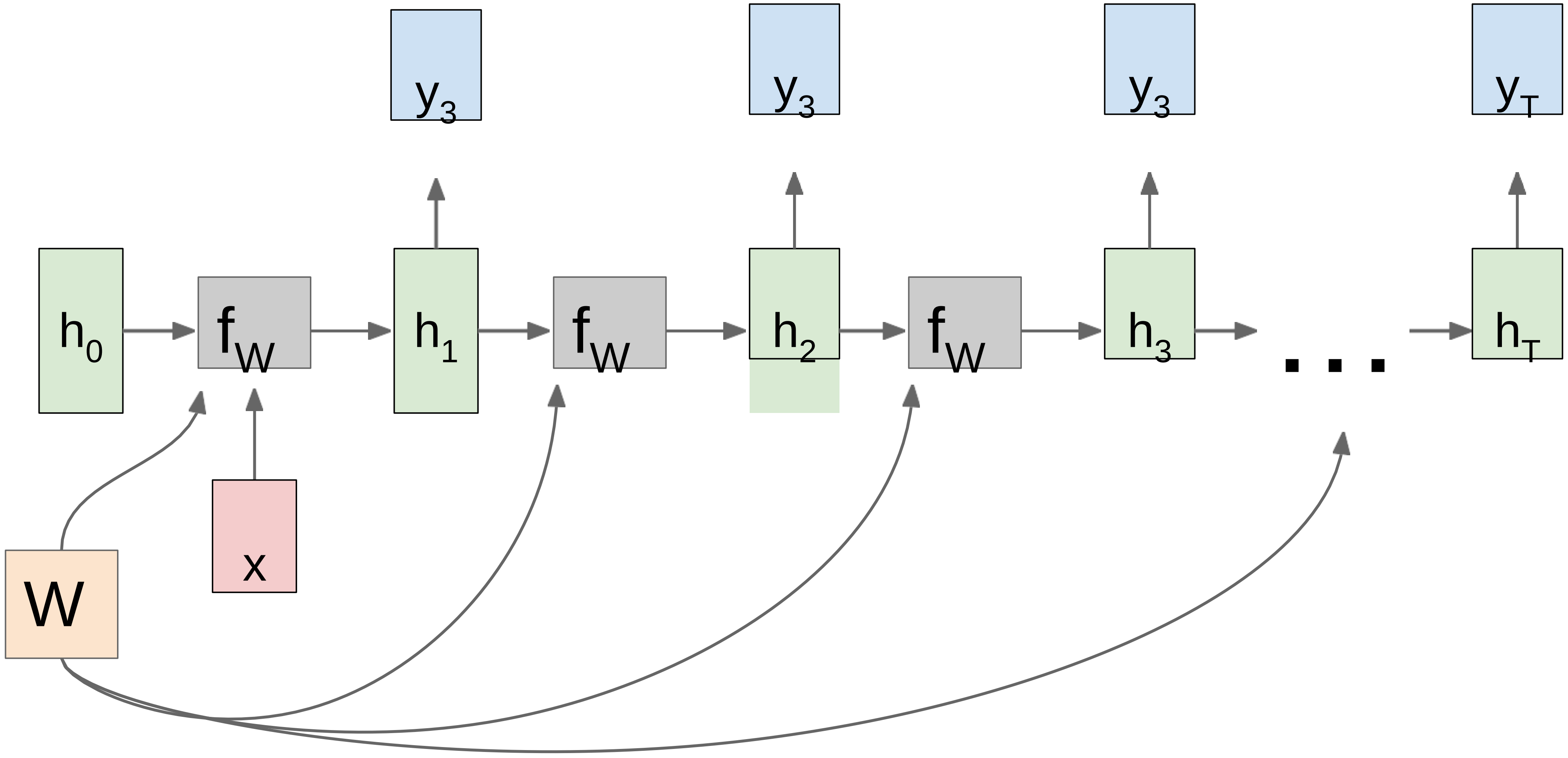
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

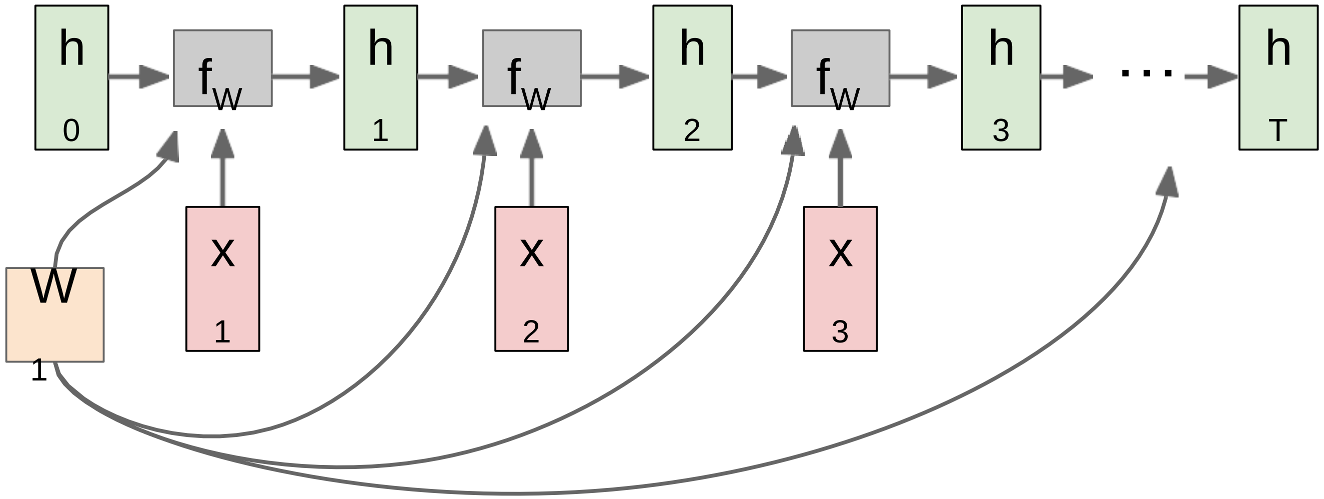


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

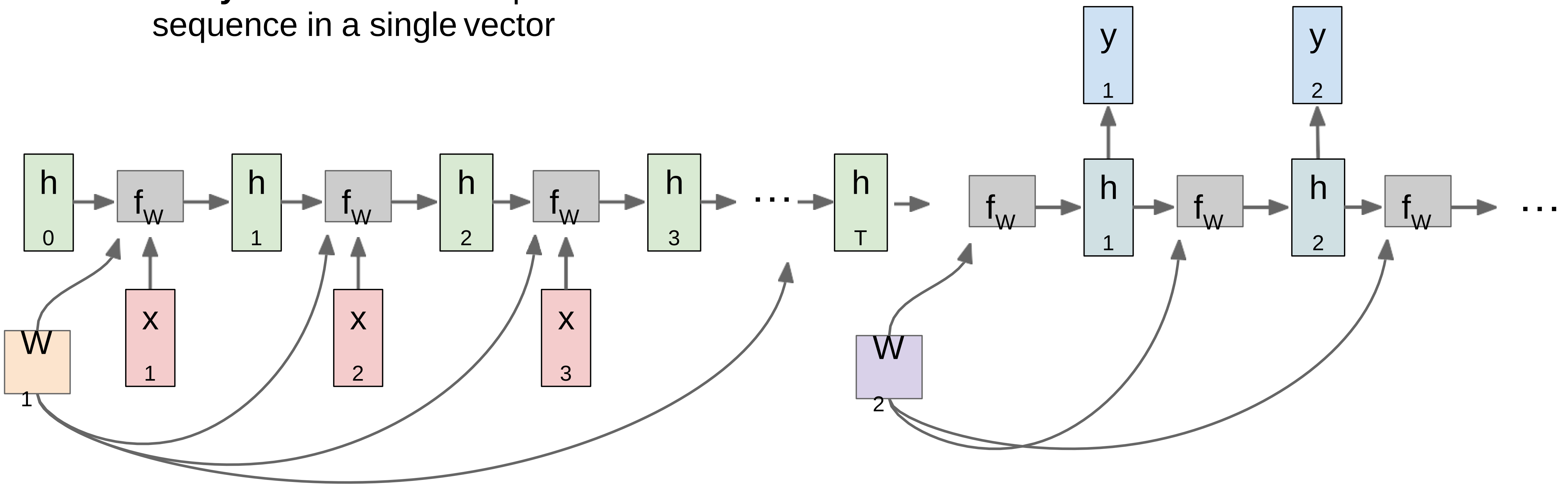
Many to one: Encode input sequence in a single vector



Sequence to Sequence: Many-to-one + one-to-many

One to many: Produce output sequence from single input vector

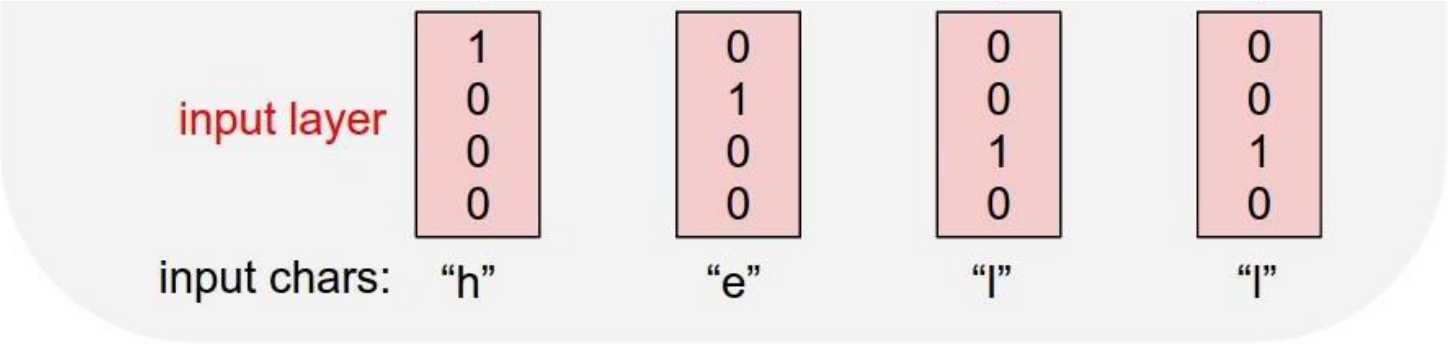
Many to one: Encode input sequence in a single vector



Example: Character-level Language Model （输入法联想）

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

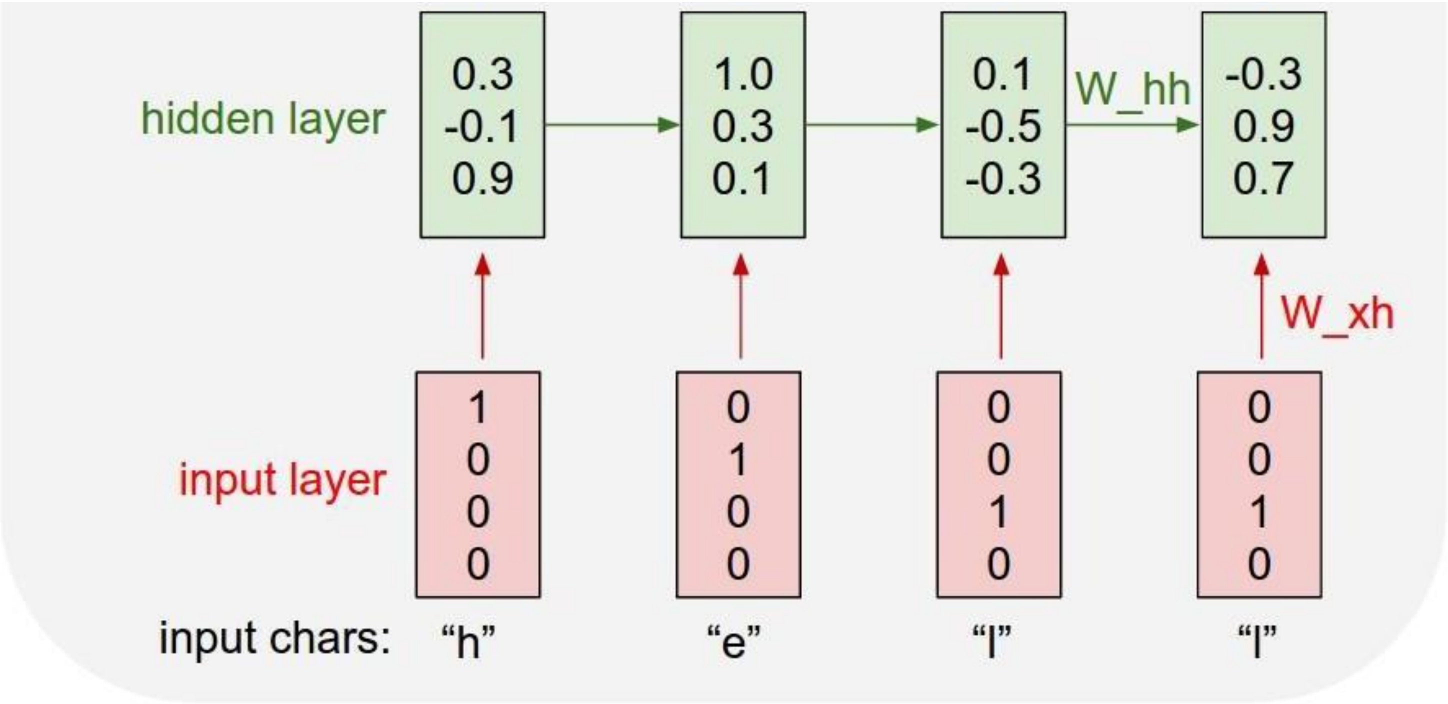


Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

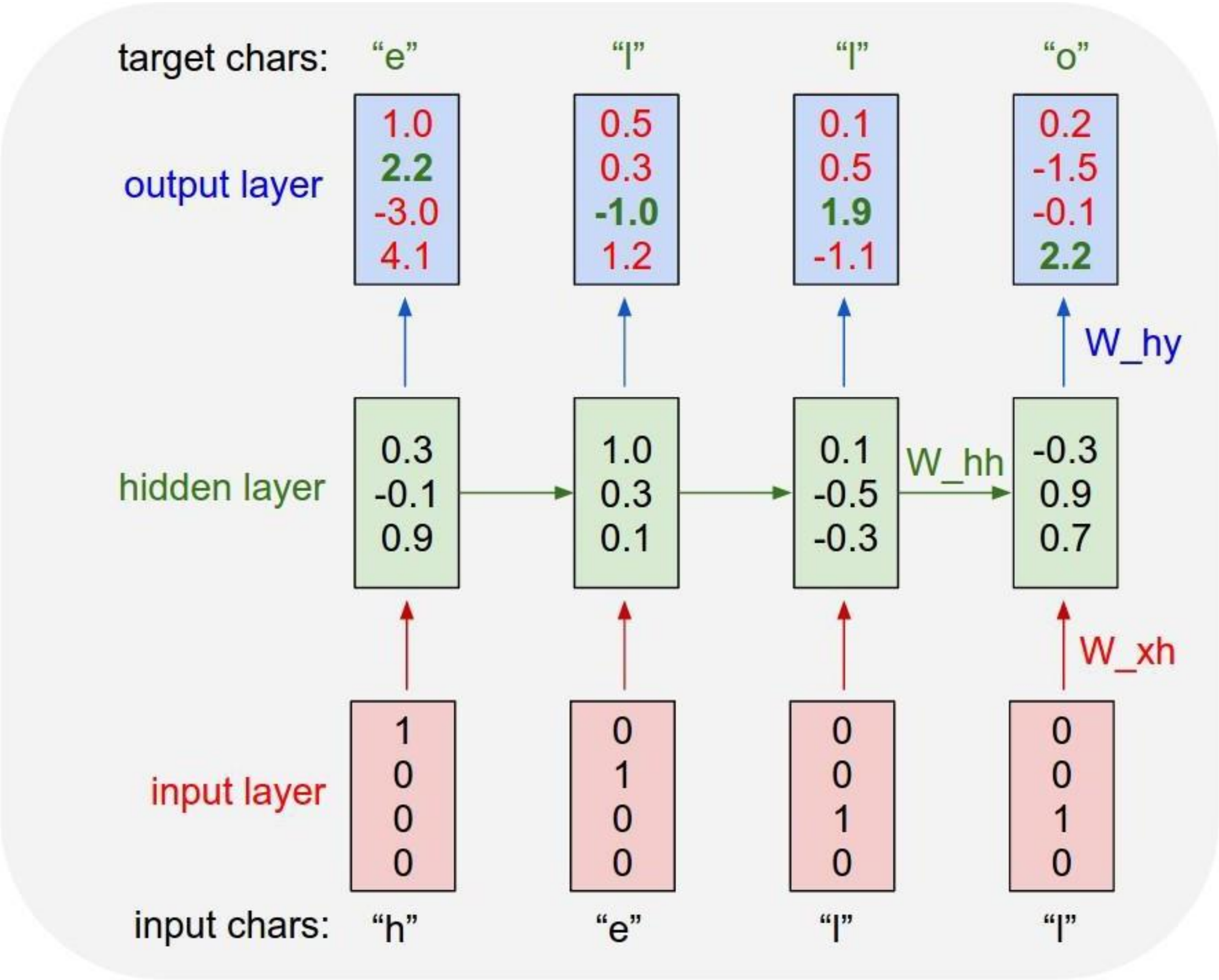
$$\boxed{h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)}$$



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

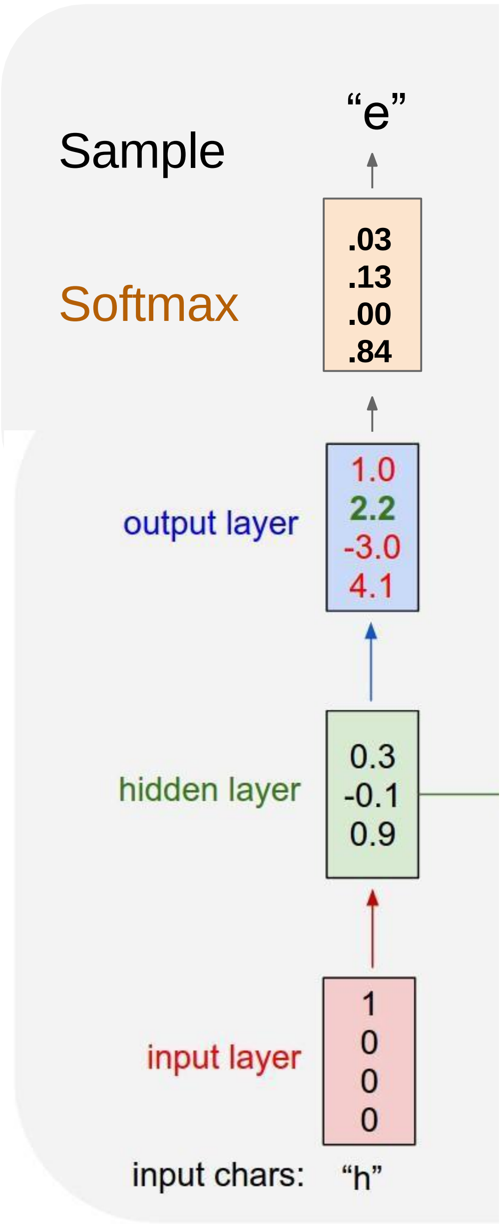
Example training sequence: **“hello”**



Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

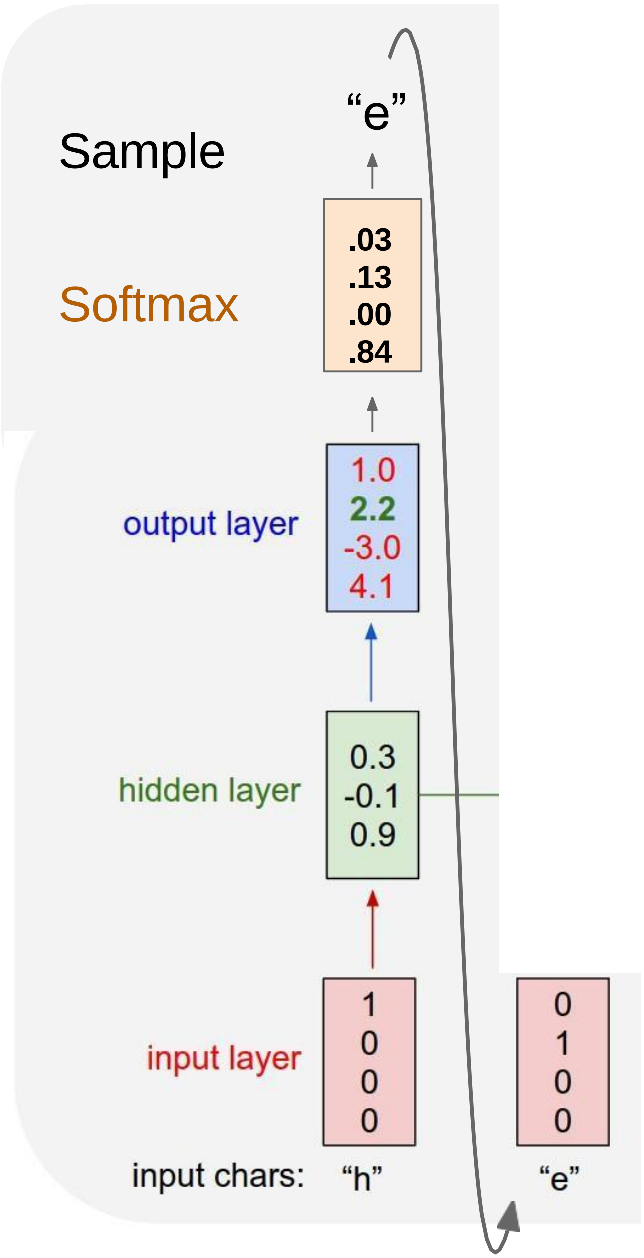
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

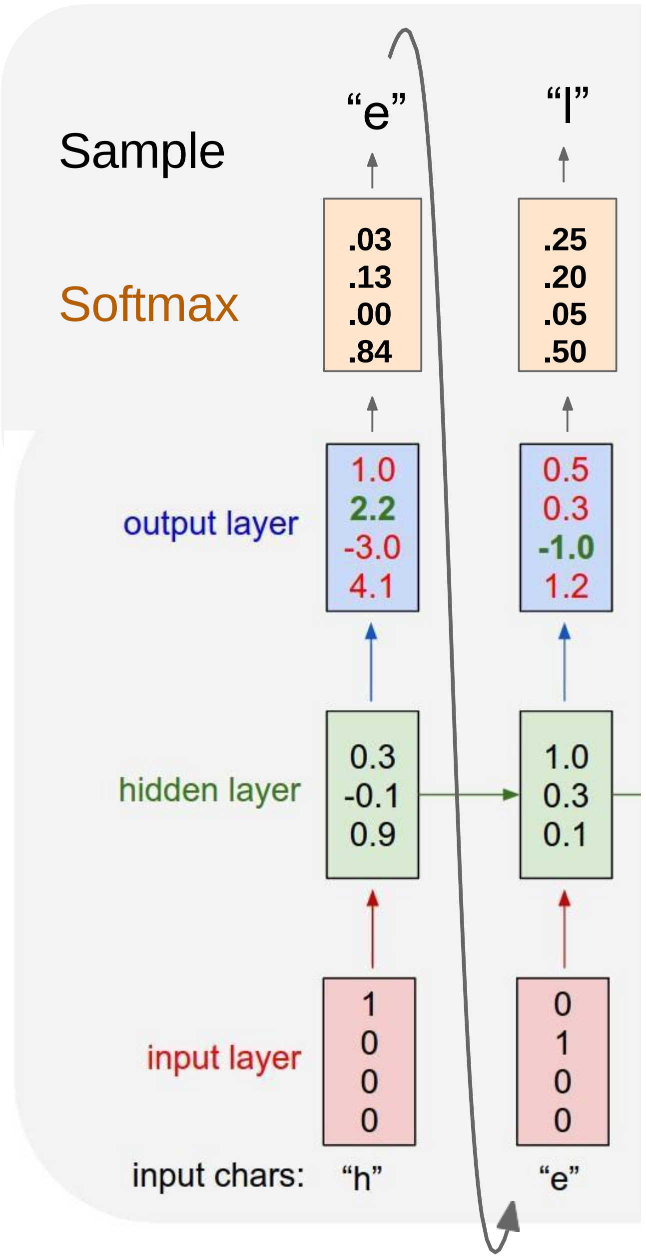
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

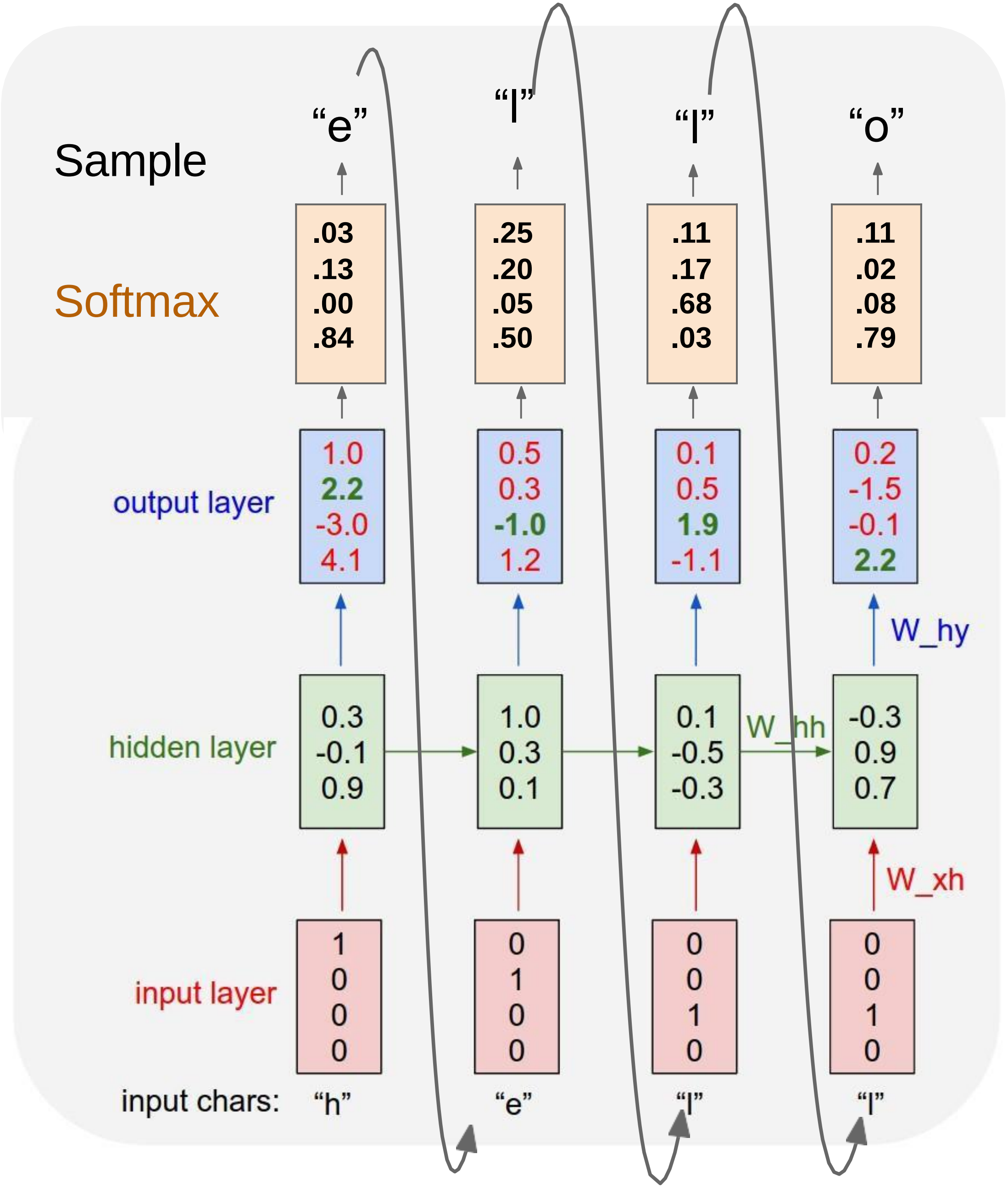
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

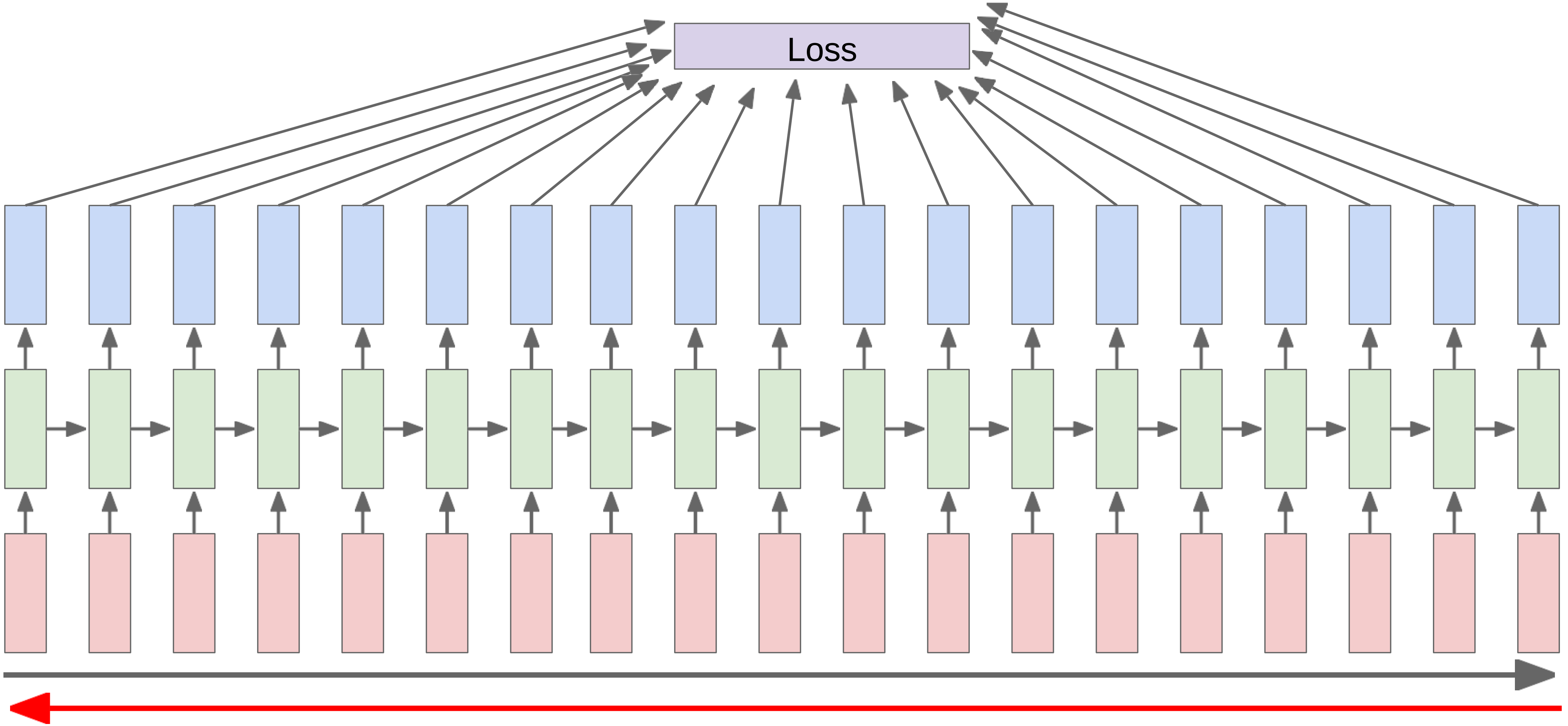
Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

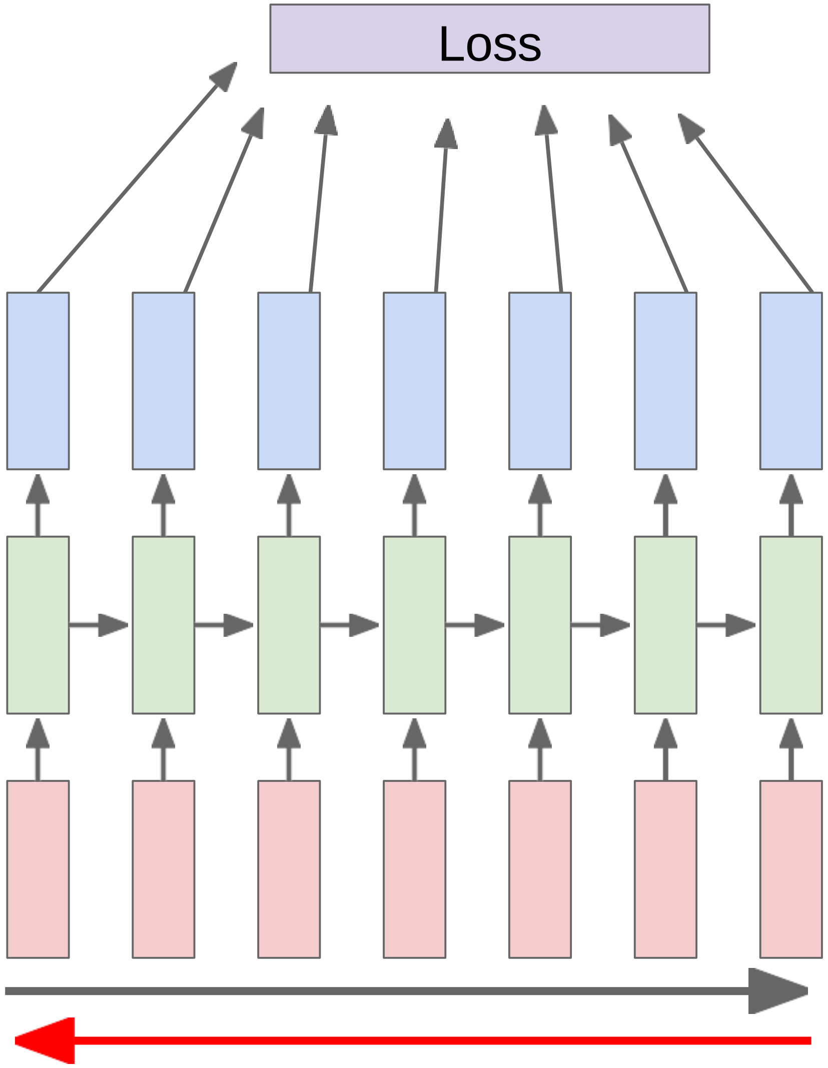


Backpropagation through time （BPTT）

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

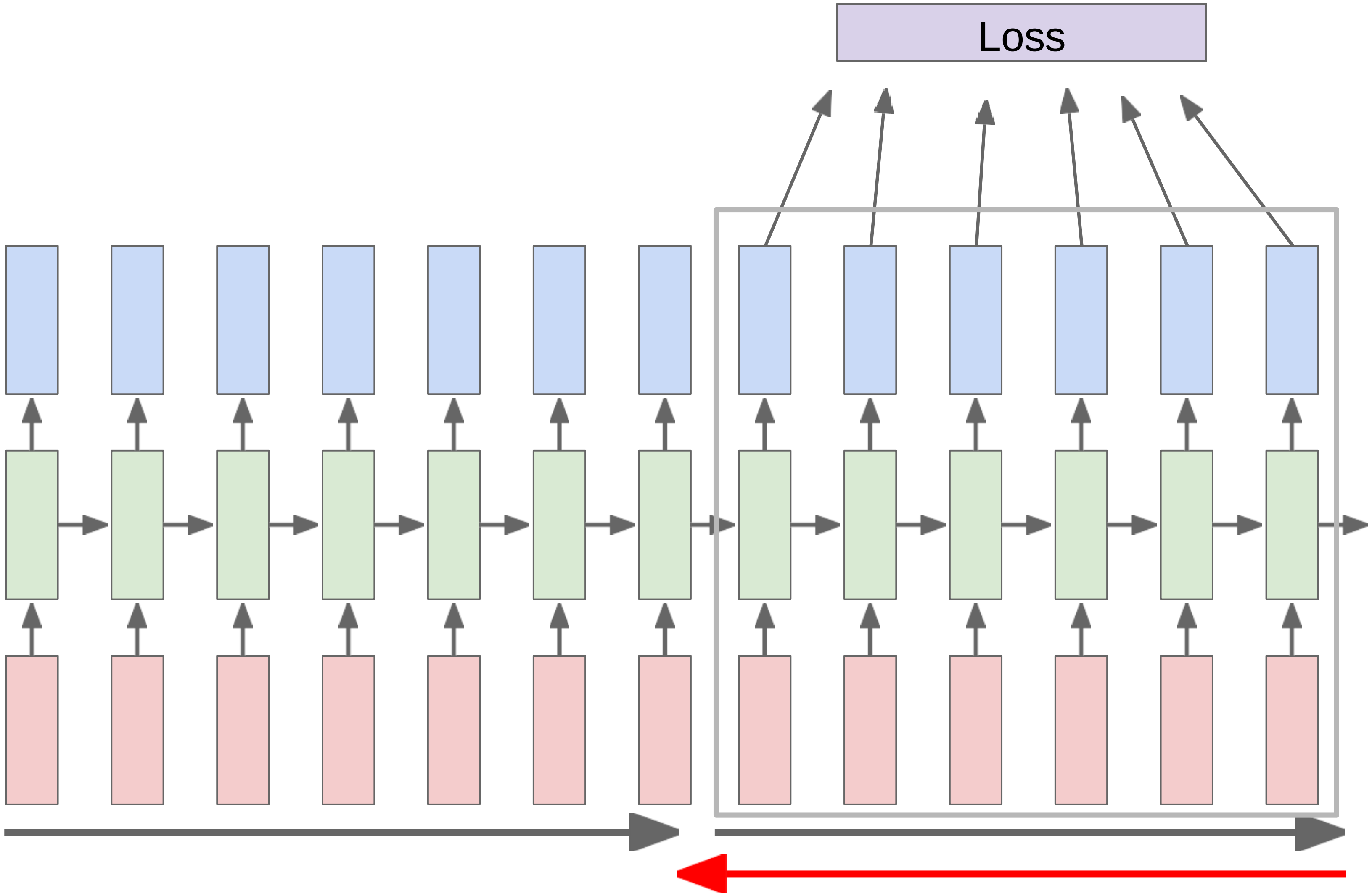


Truncated (缩减的) Backpropagation through time



Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through time

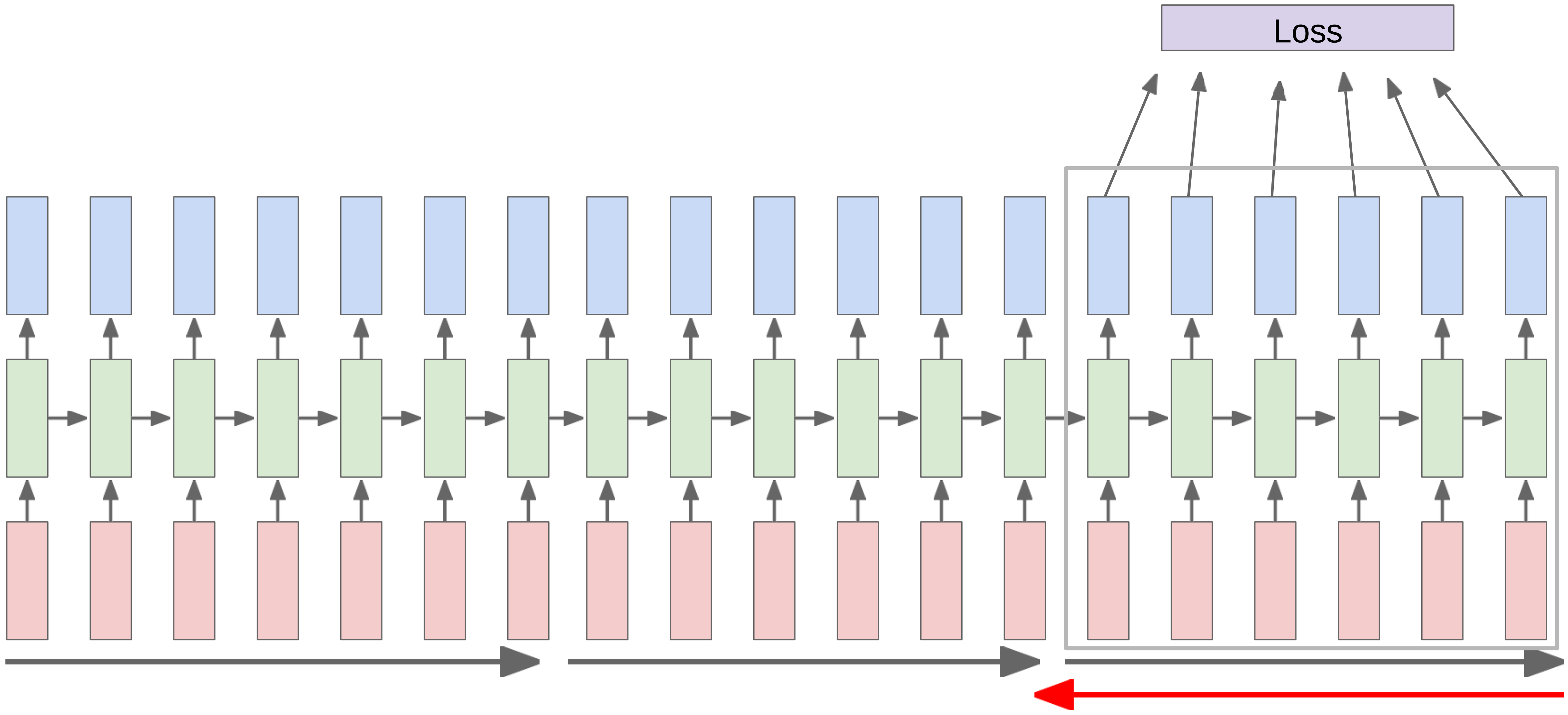


Image Captioning

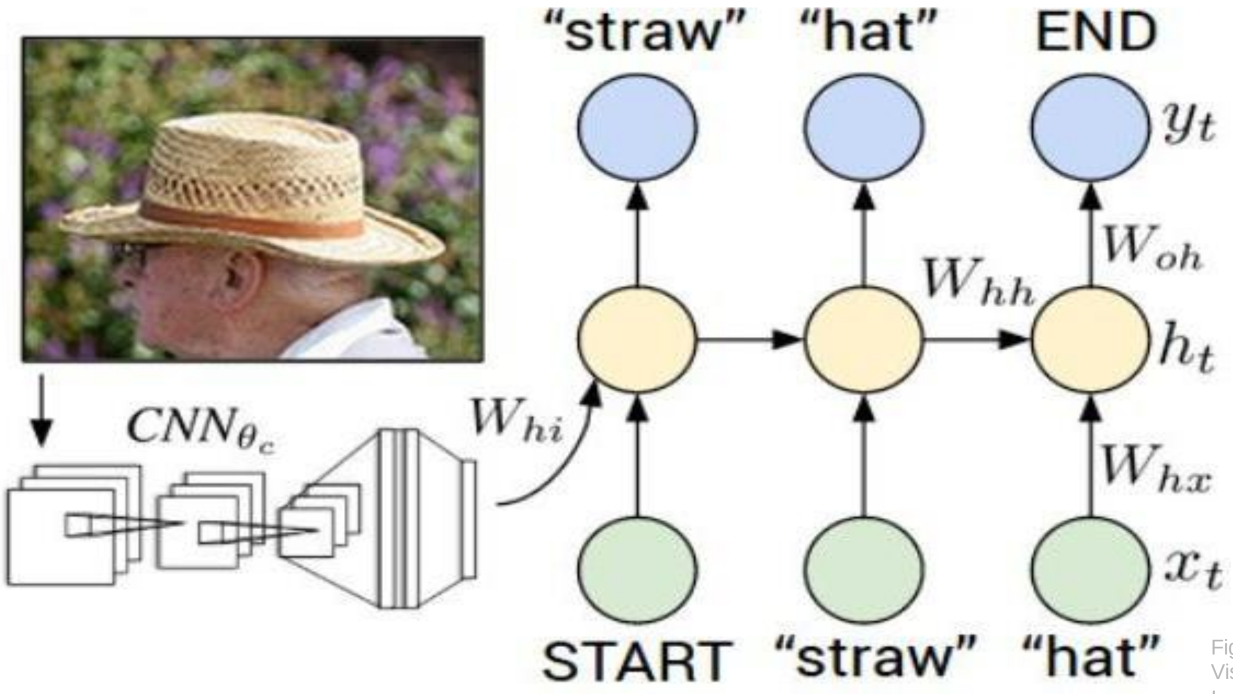
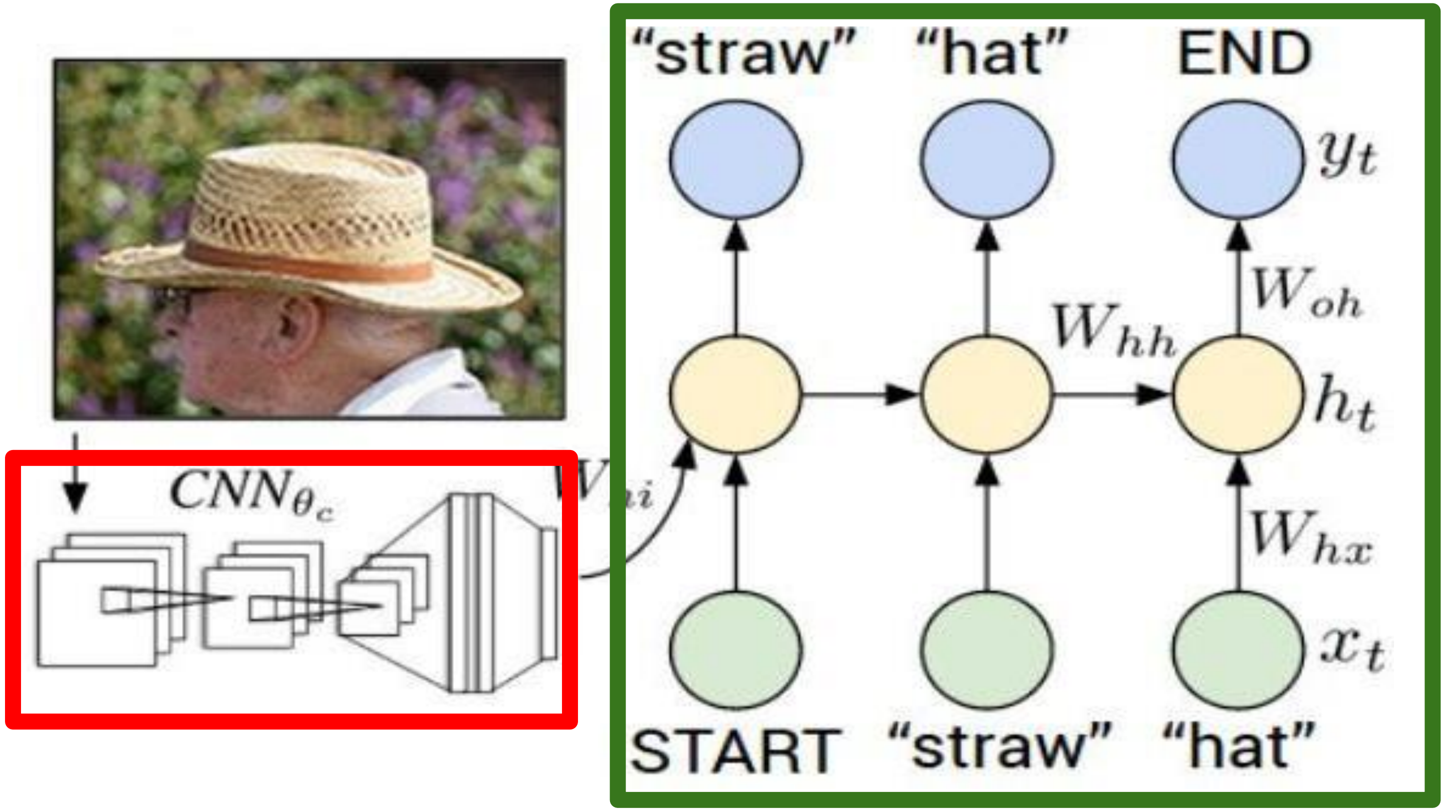


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network



Convolutional Neural Network

test image

[This image](#) is [CC0 public domain](#)

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096
FC-1000
softmax

test image


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

test image

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

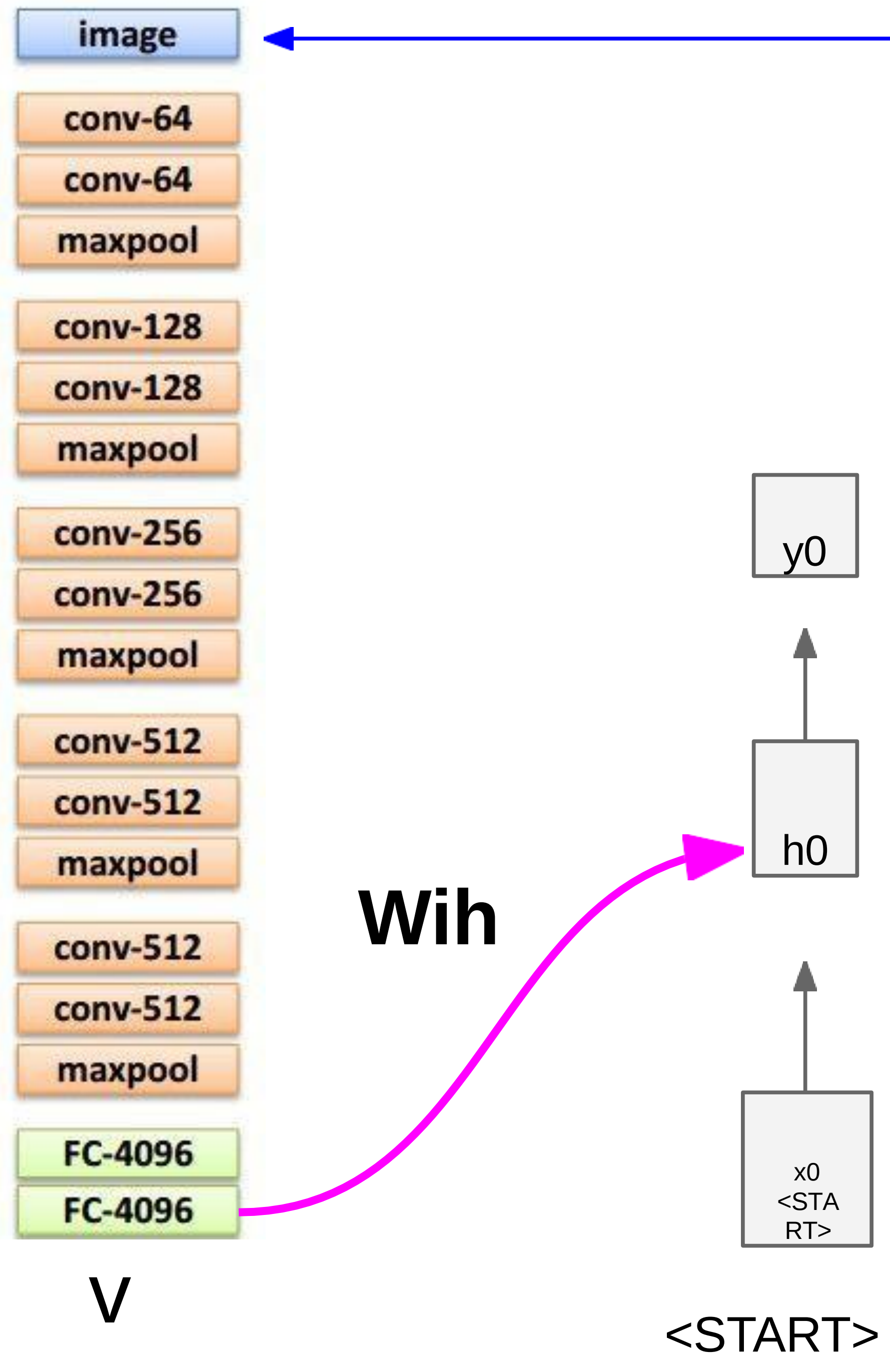
conv-512
conv-512
maxpool

FC-4096
FC-4096

test image

x0
<STA
RT>

<START>



test image

before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h **+ Wih * v**)

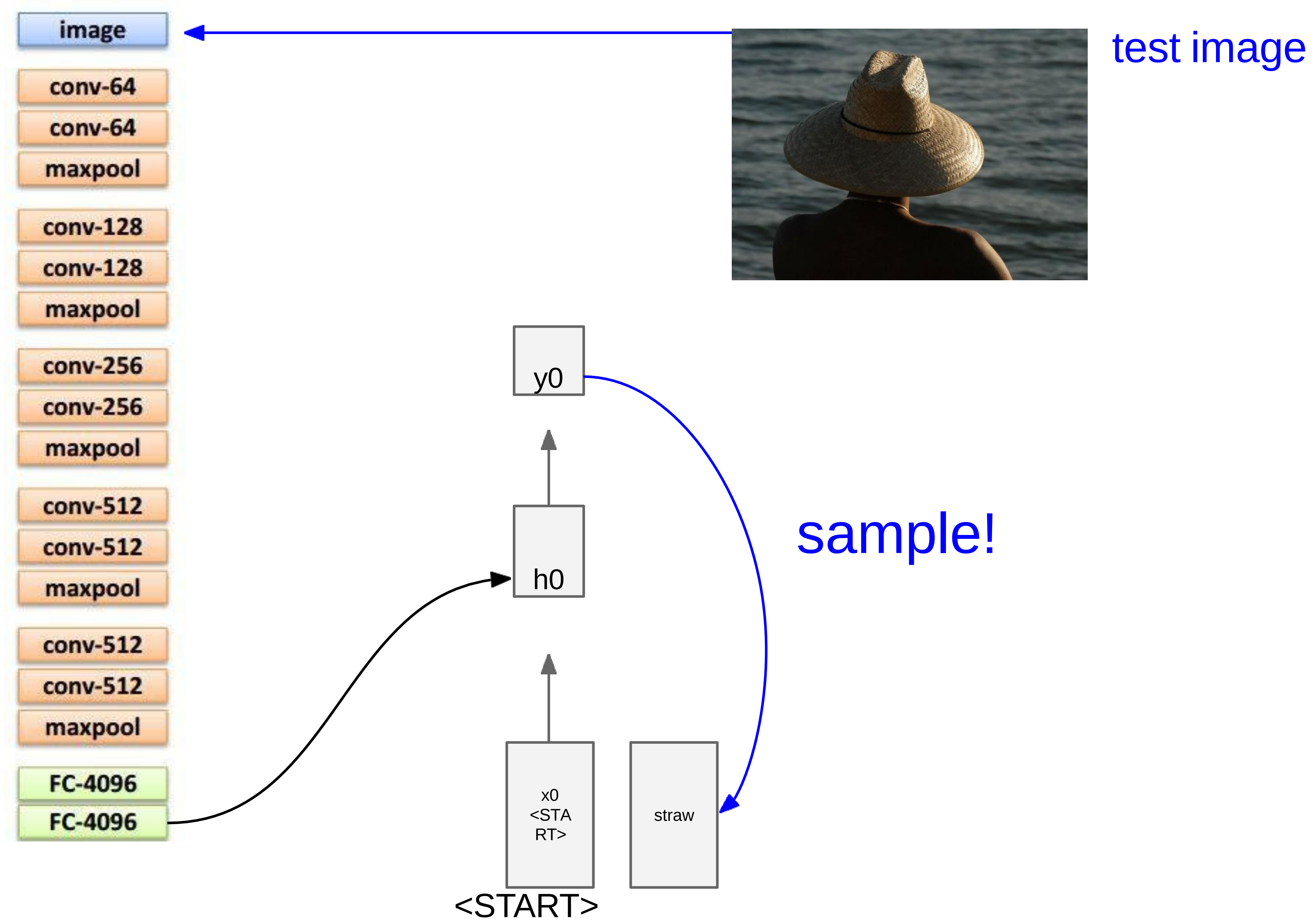


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
sample!
h0
x0 <START>
straw
<START>

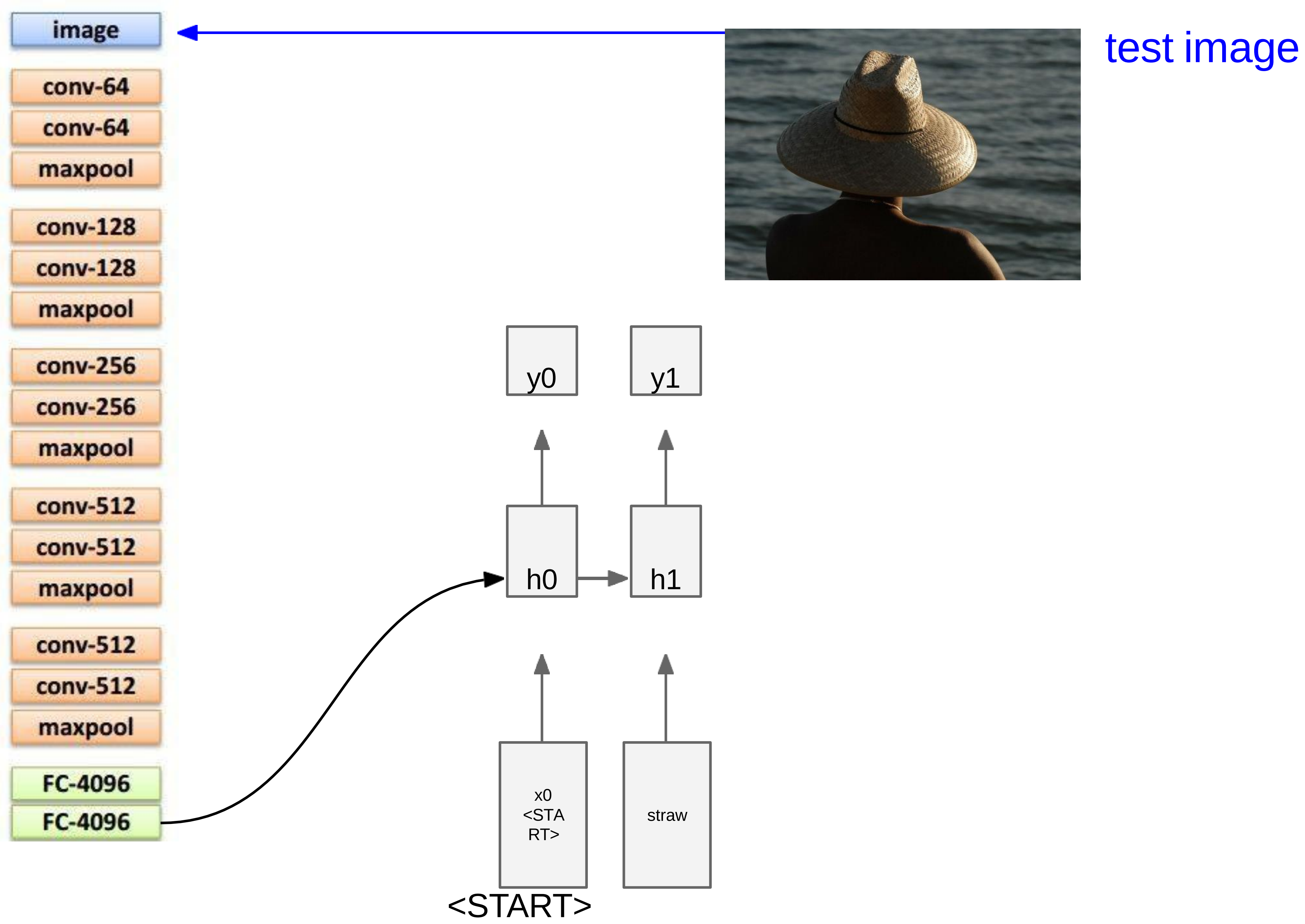


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
x0 <START>
straw
<START>

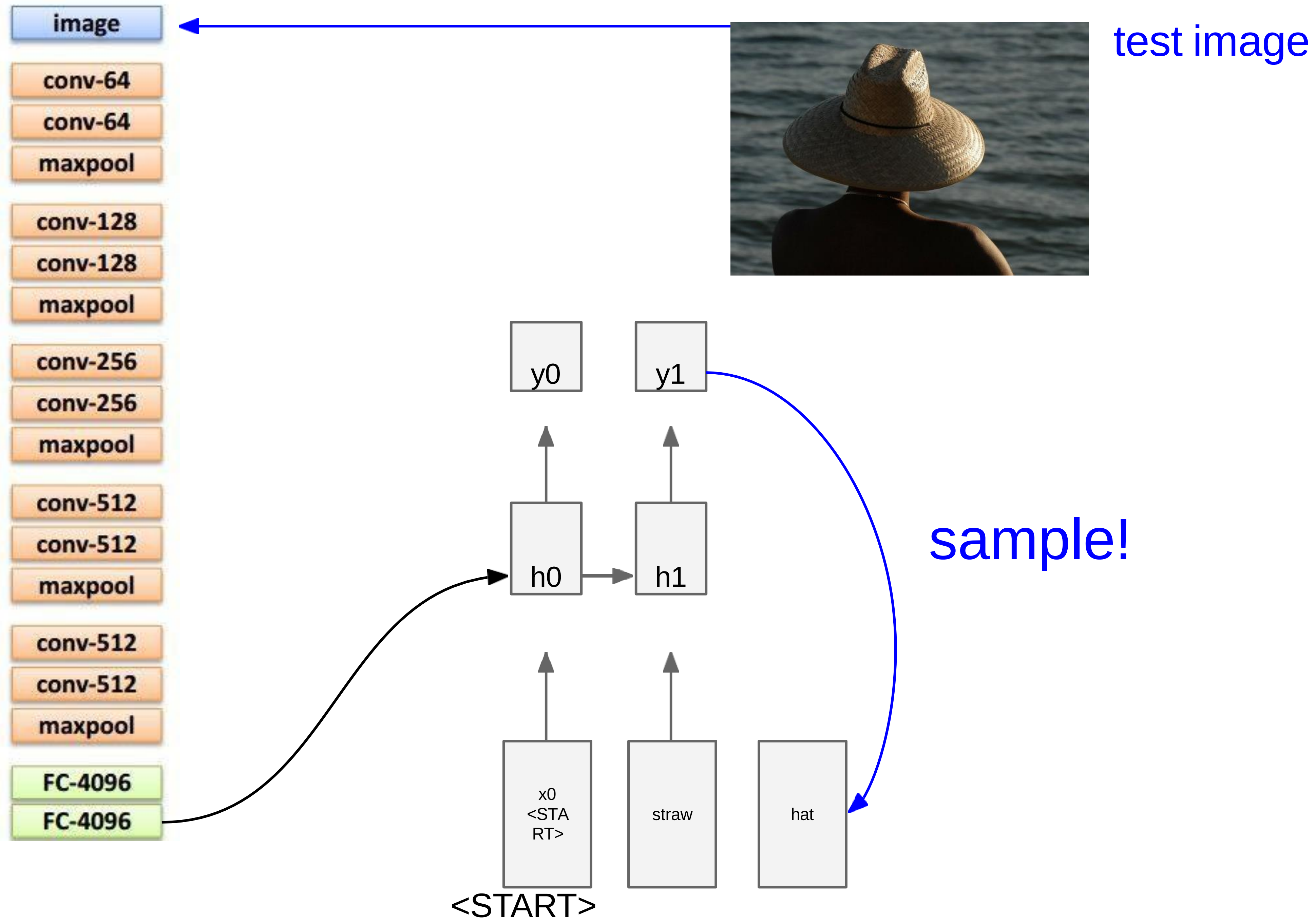


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0
<STA
RT>
straw
hat
<START>

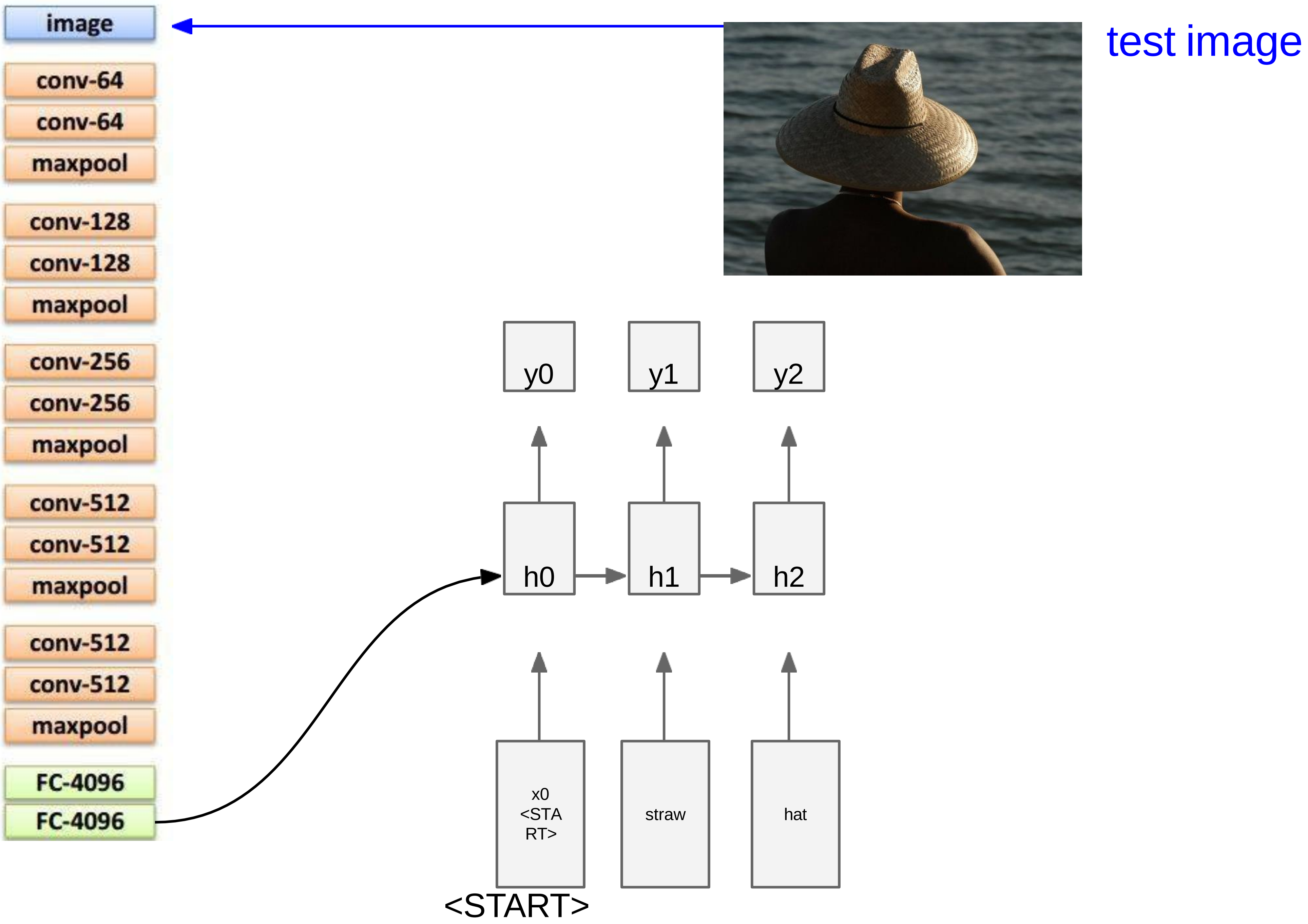


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0 <START>
straw
hat
<START>

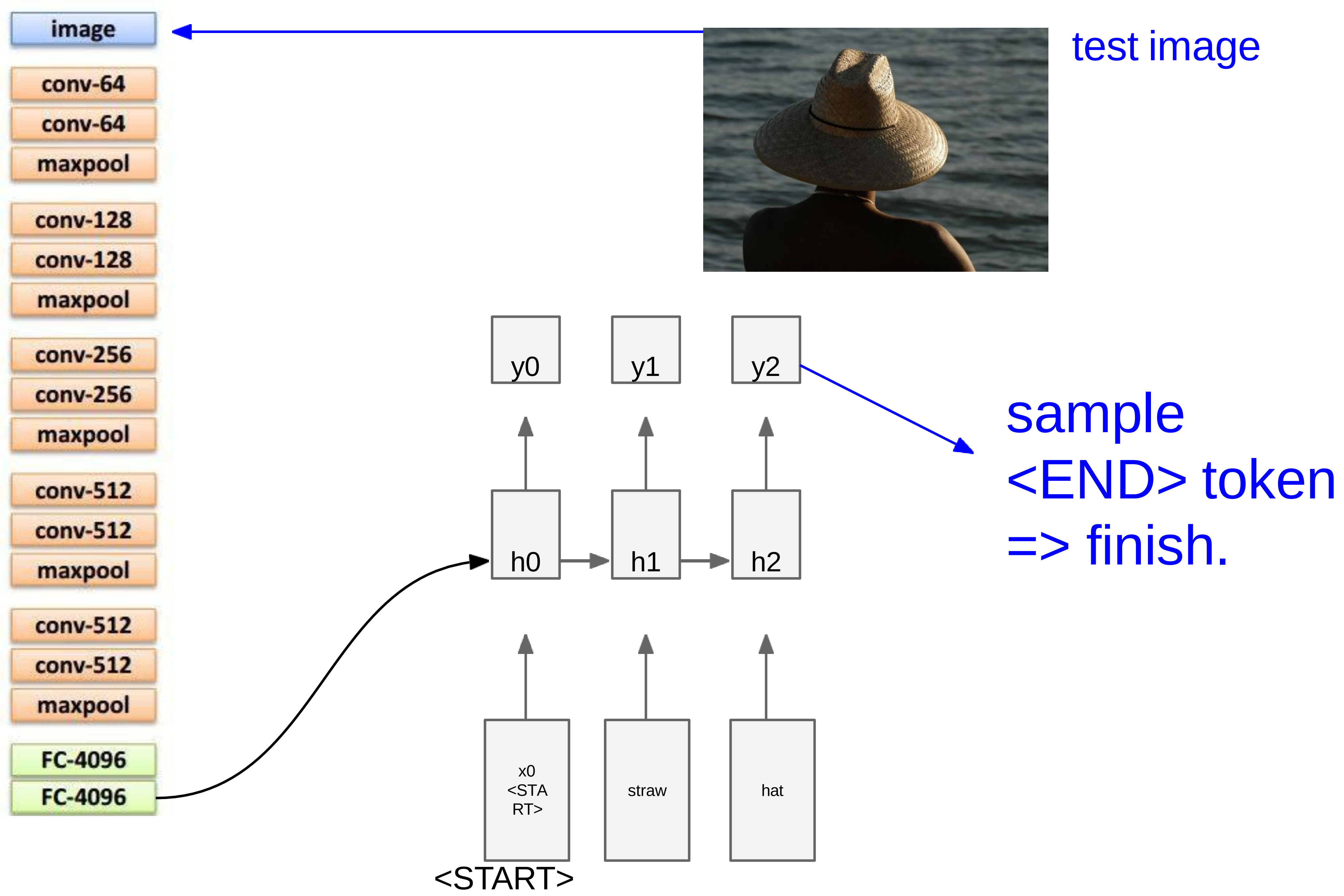


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0 <START>
straw
hat
<START>
sample <END> token => finish.

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

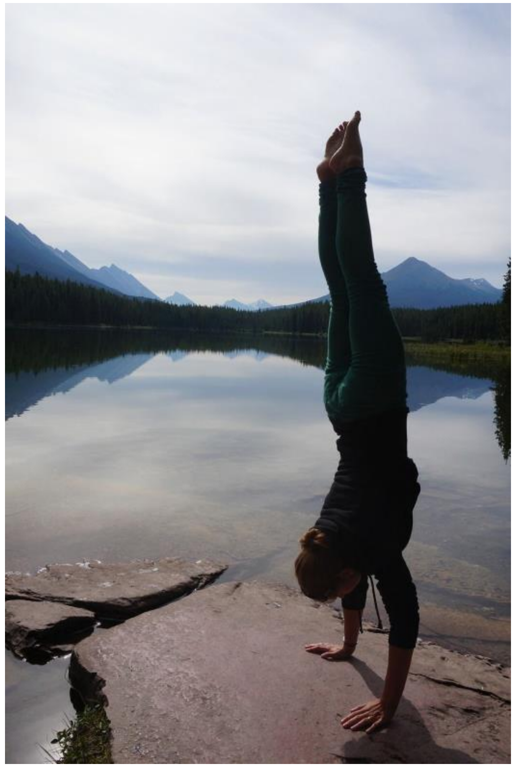
A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

THE END !