Lecture 5: Recurrent Neural Networks I: Basics

Lan Xu
SIST, ShanghaiTech
Fall, 2023

Outline

- CNN applications in dense prediction
- Recurrent Neural Networks
  - Sequence modeling, Autoregressive models
  - (Vanilla) RNN models
- Backpropagation through time
  - Computational graph
- Example: language modeling
  - Neural language models

Acknowledgement: Feifei Li et al's cs231n notes

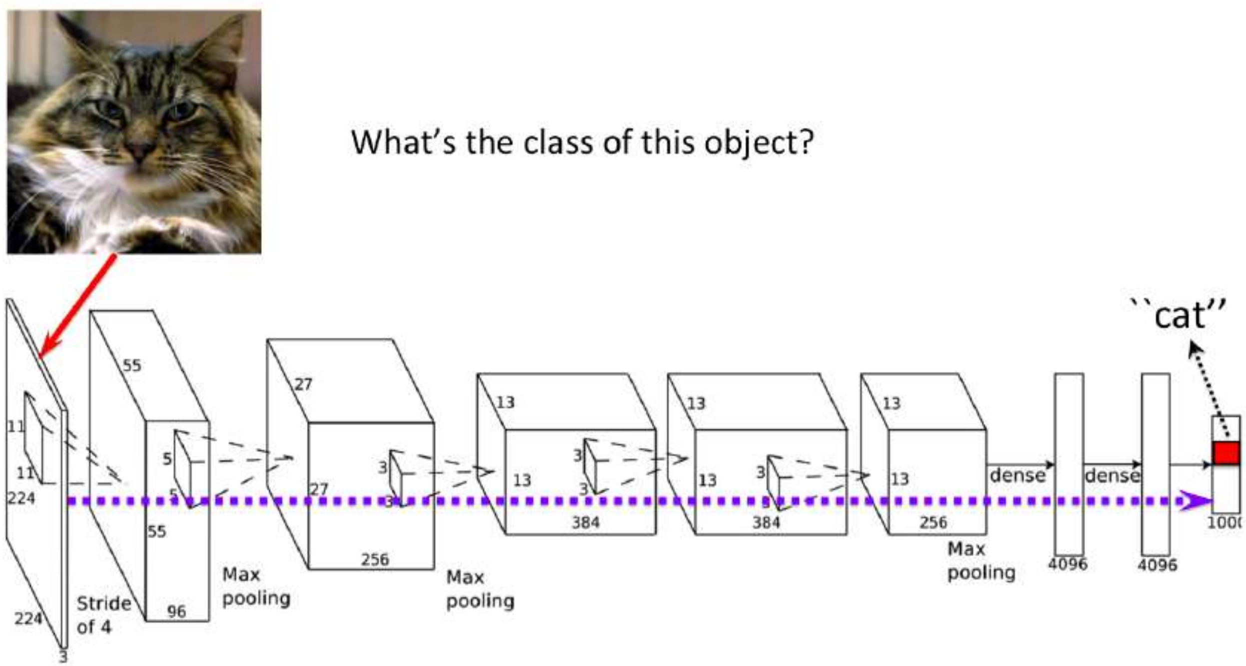
Review

- In general, our goal is to learn a mapping from a signal to a 'semantically meaningful' representation.
  - Output can have many different forms:

red panda (*Ailurus fulgens*)

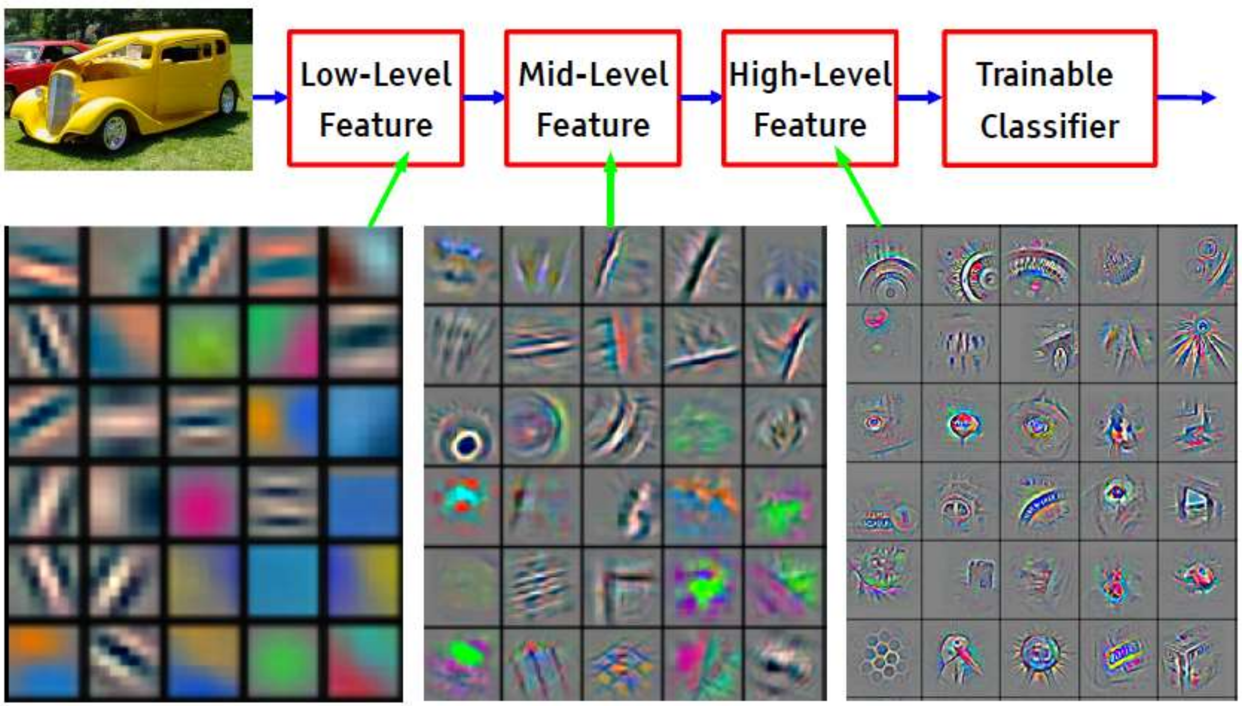
Previously on CNNs

- CNNs for image/object classification
  - AlexNet, VGGNet, GoogLeNet, ResNet, DenseNet
  - Trend towards deeper networks with more flexible network architecture
  - Better representation and more effective learning strategies



More on classification

- Why it works well for image classification?
  - Built-in translation and small deformation invariance
  - Hierarchical feature learning – shared representation
  - End-to-end training for the target problem



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

More on classification

- Is image classification a solved problem?
  - "(Super-)Human level" performance on some benchmarks
    - Face identification
    - ImageNet 1000 classes
- But compared to human vision…
  - Limitations in learning
    - We can learn new classes using one or two examples
    - We can also handle label noises
    - We can generalize to unfamiliar scenes
  - Limitation in prediction
    - We can also predict the uncertainty
    - We can easily handle adversarial examples
    - We are much more efficient in power consumption

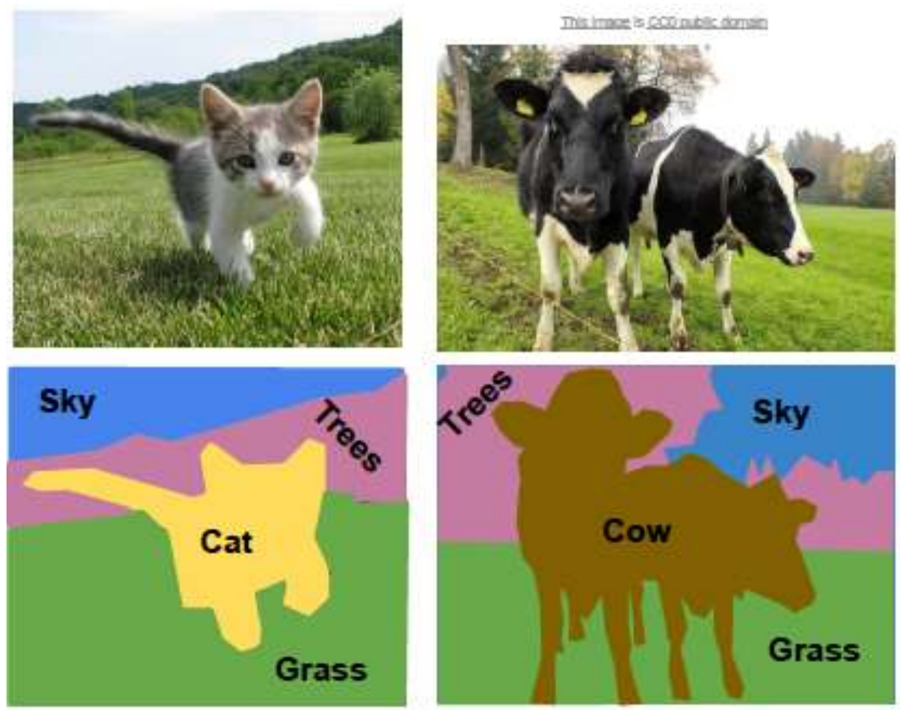
Outline

- What is semantic segmentation?
- Network architecture for semantic segmentation
  - Main idea for dense prediction
  - Fully convolutional network
  - Upsampling operators
  - Multiscale context modeling
- Network training losses

Acknowledgement: Feifei Li et al's cs231n notes

Semantic Segmentation

- Problem setup
  - Label each pixel in the image with an object category label
  - Do not differentiate object instances



Key to many applications

- Autonomous robots and cars

- Safety and security

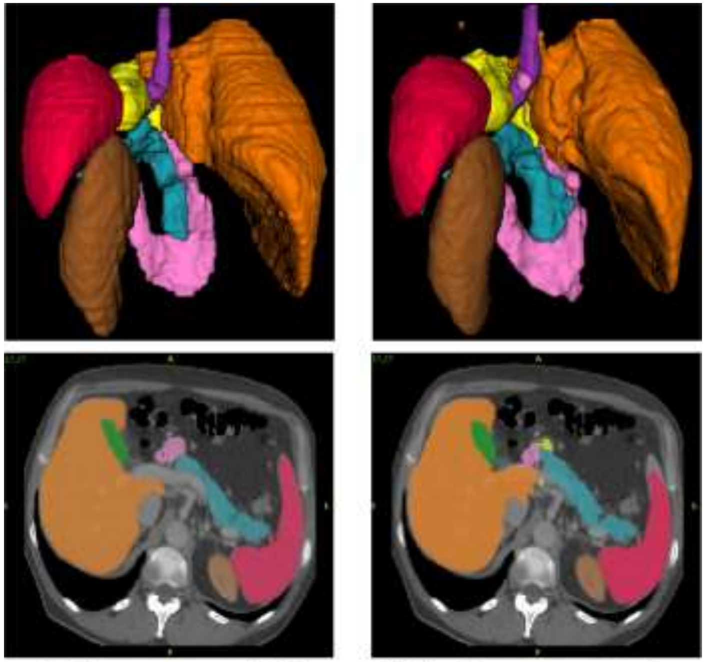
- Medical analysis and health

- etc…

Key to many applications

Autonomous driving
https://youtu.be/qWl9idsCuLQ

Multi-organ abdominal
CT segmentation
https://doi.org/10.1016/j.cmpb.2018.01.025



Reference standard NiftyNet segmentation

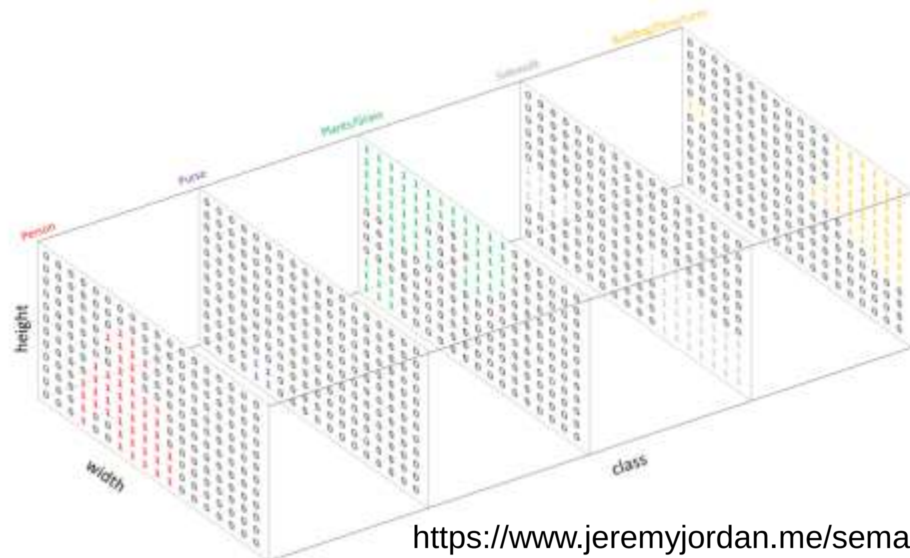
Semantic Segmentation

- Problem formulation
  - Pixel-wise object classification task

Input

Semantic Labels

  - One-hot encoding



https://www.jeremyjordan.me/semantic-segmentation/

Why this is challenging?

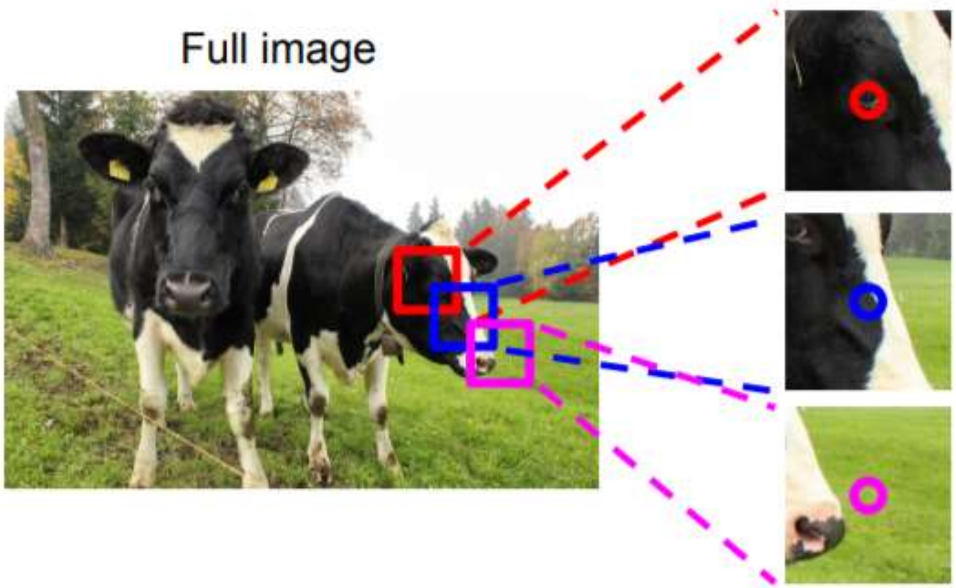
- A naïve approach

Impossible to classify without context

Q: how do we include context?

Why this is challenging?

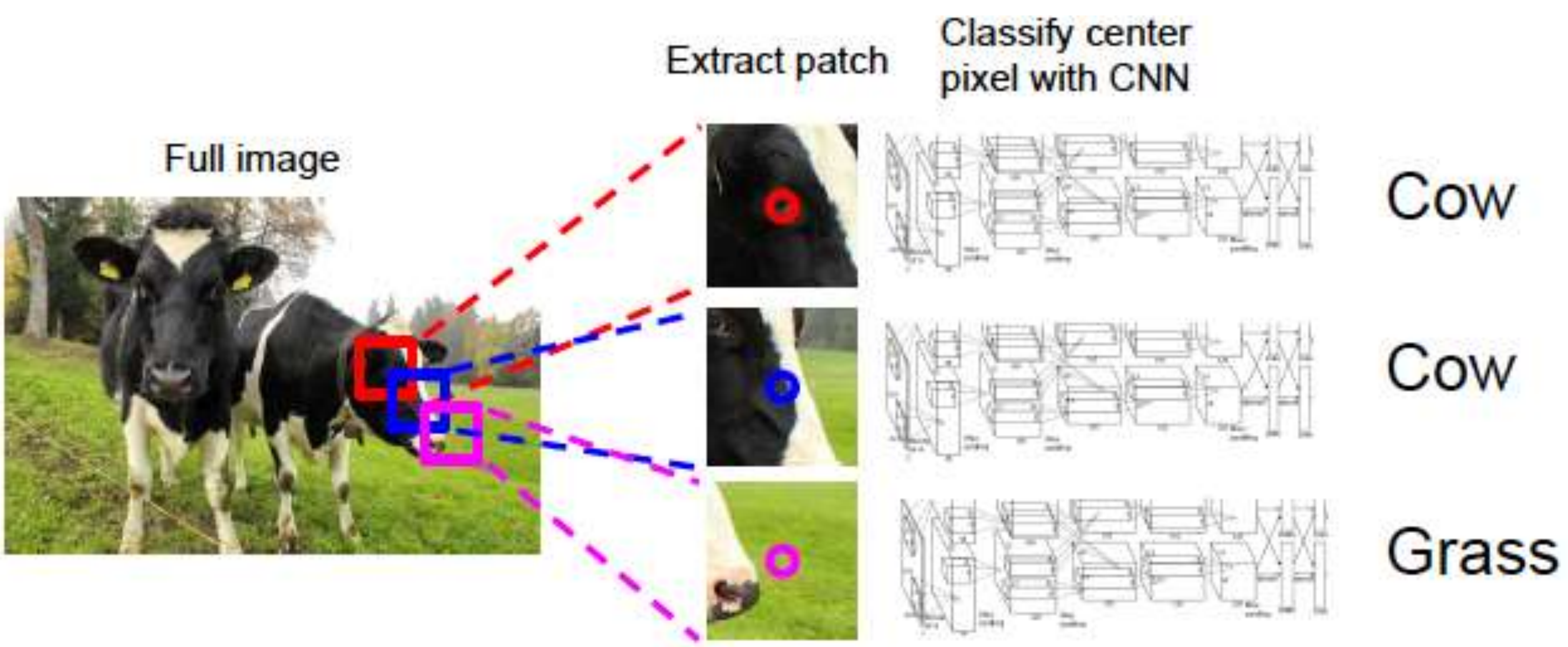
- A naïve approach



Q: how do we model this?

Why this is challenging?

- A naïve approach



Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Network for semantic segmentation

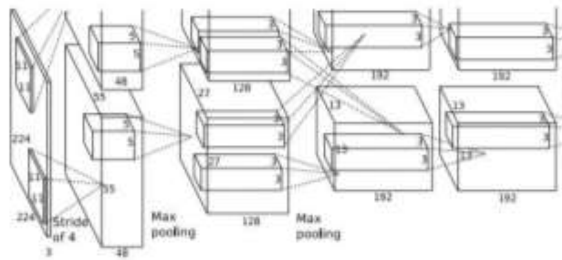
- Main idea for dense prediction
- Fully convolutional network
- Upsampling operators
- Multiscale context modeling

Acknowledgement: Feifei Li et al's cs231n notes

Several Ideas for SS

- First idea

Full image

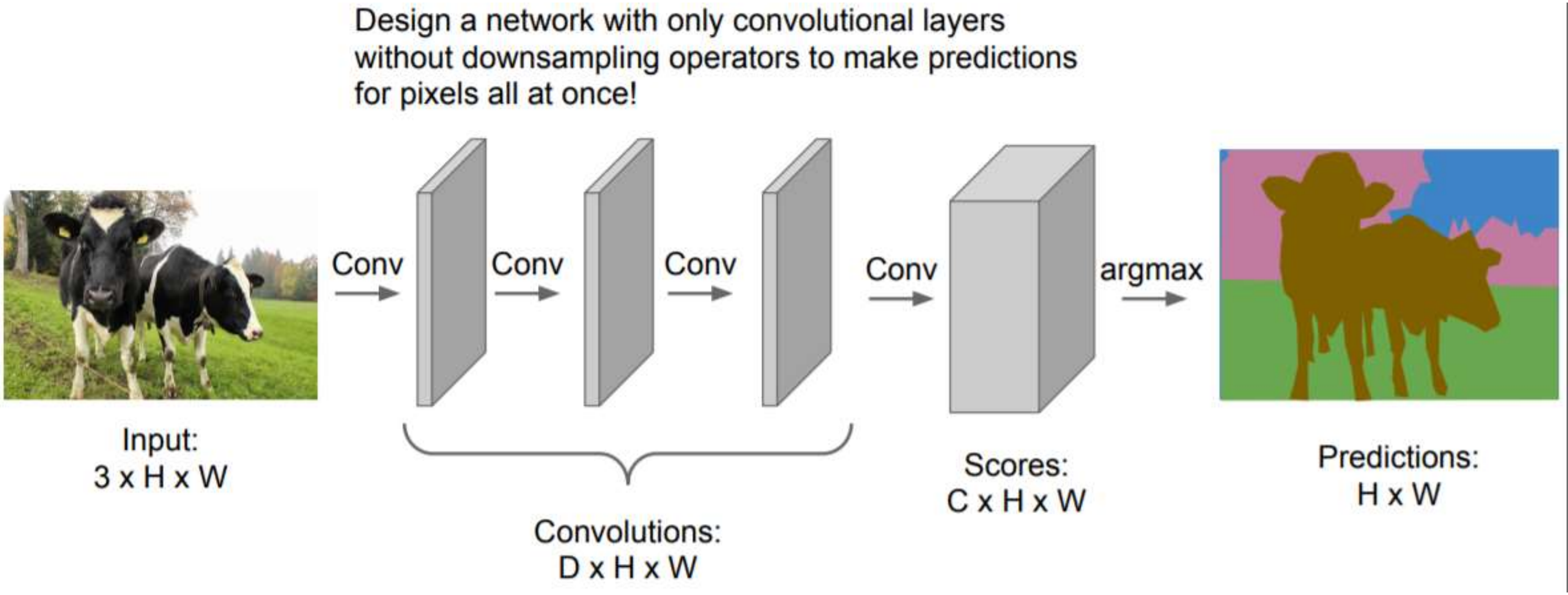


An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

Problem: classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

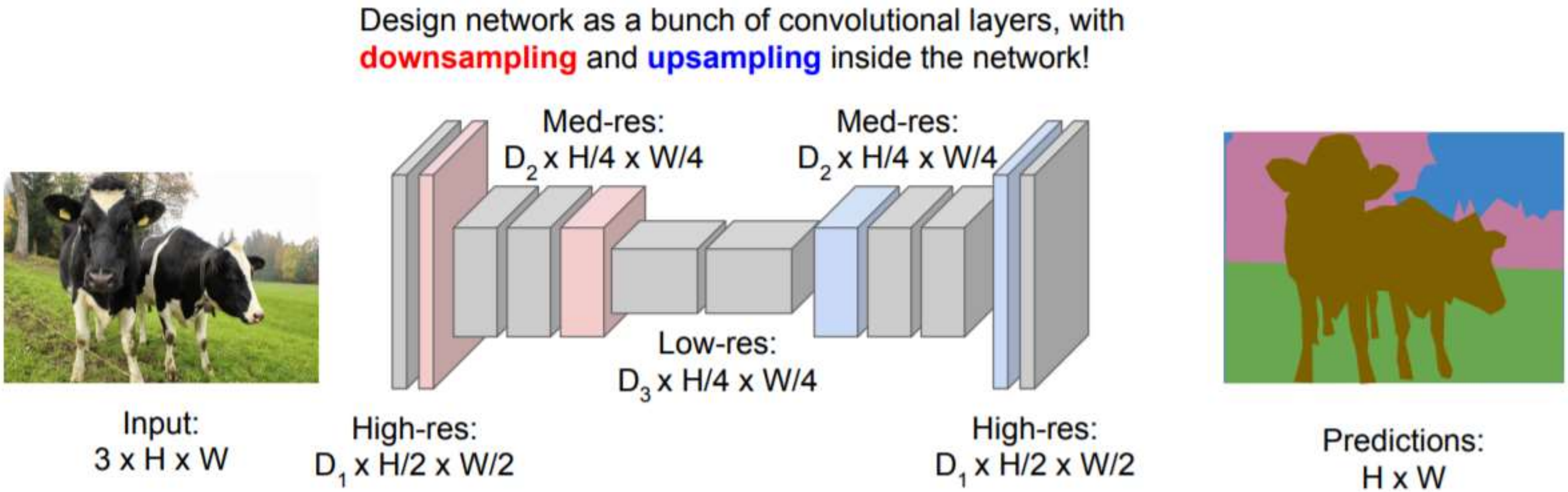
Several Ideas for SS

- Second idea



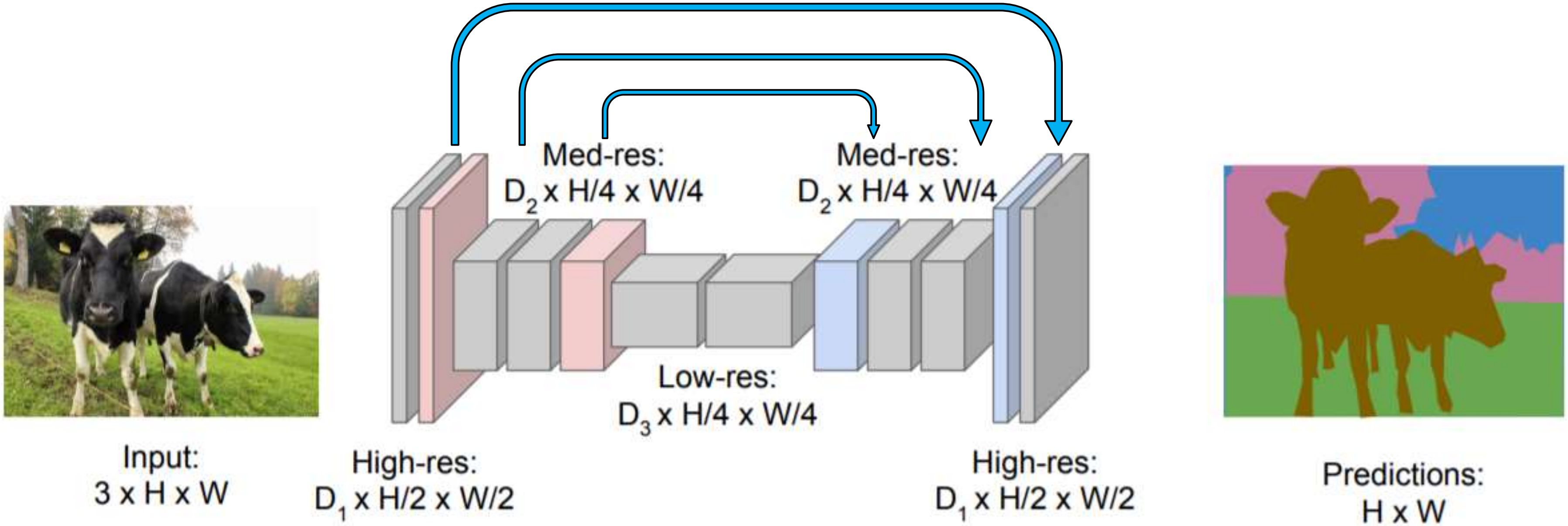
Several Ideas for SS

- Second idea improved



Several Ideas for SS

- Third idea



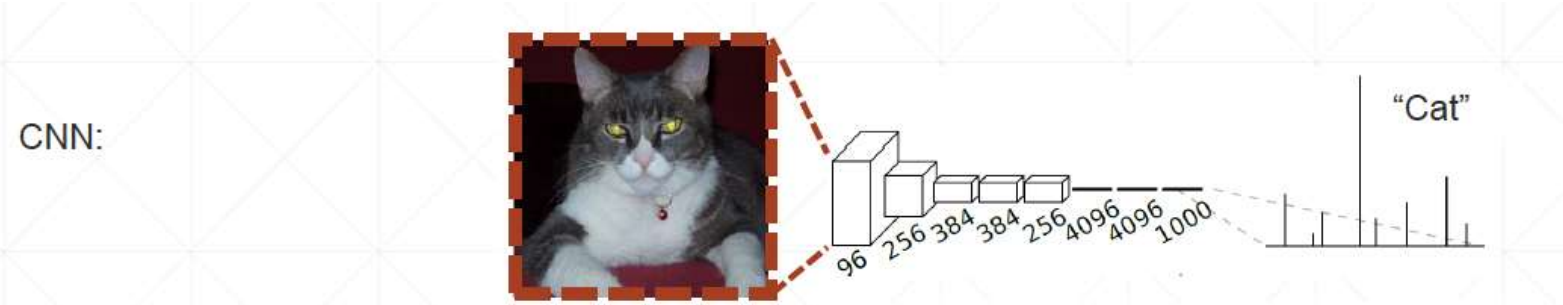
Network for semantic segmentation

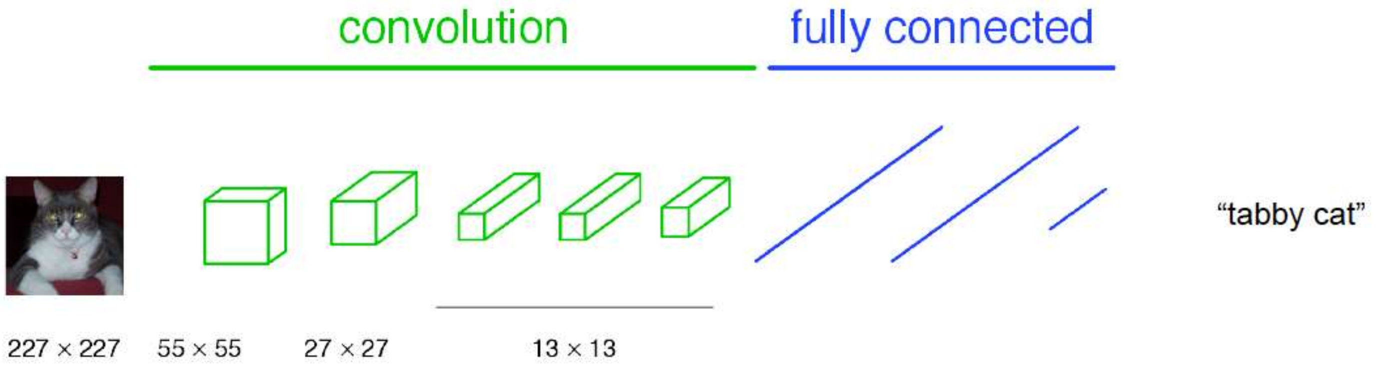
- Main idea for dense predictionedicti1o
- Fully convolutional network
- Upsampling operators
- Multiscale context modeling

Acknowledgement: Feifei Li et al's cs231n notes

Network Design I: Efficiency

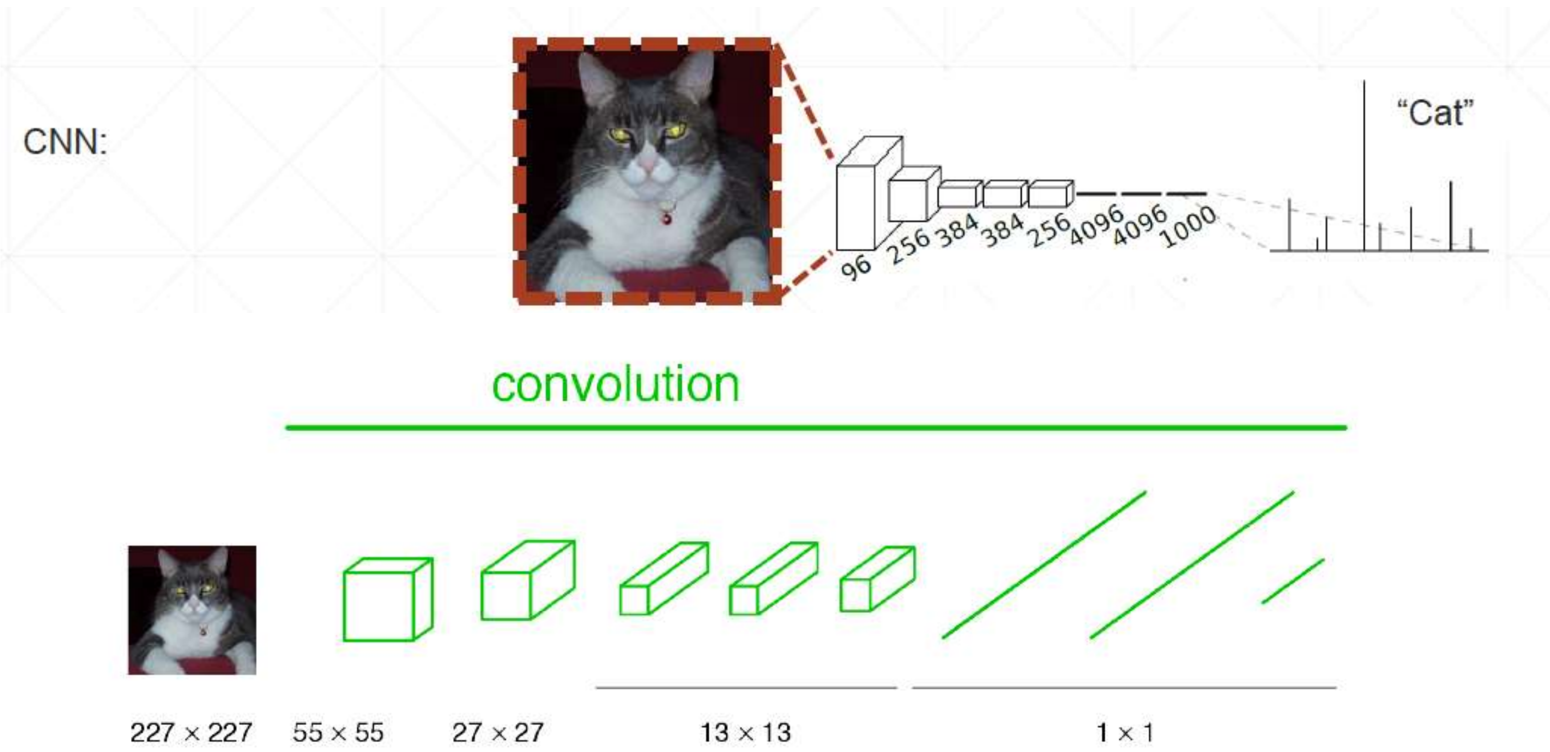
- Starting from a classification network





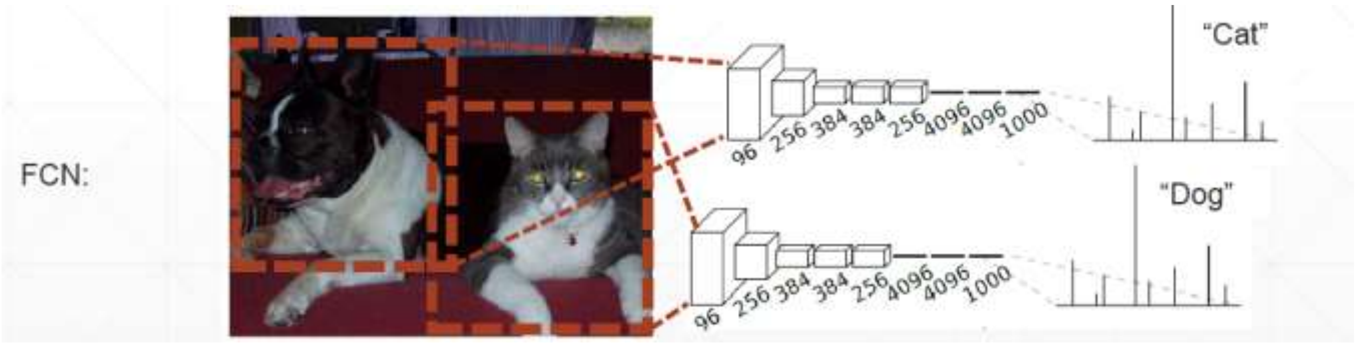
Network Design I: Efficiency

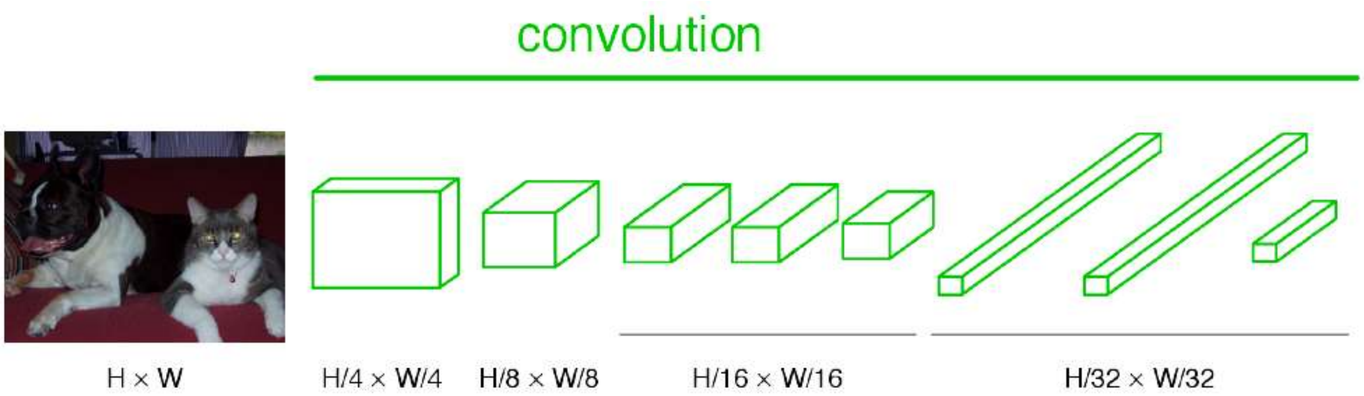
- Interpreting fully connected layers as 1x1 convolution (after reshaping)



Network Design I: Efficiency

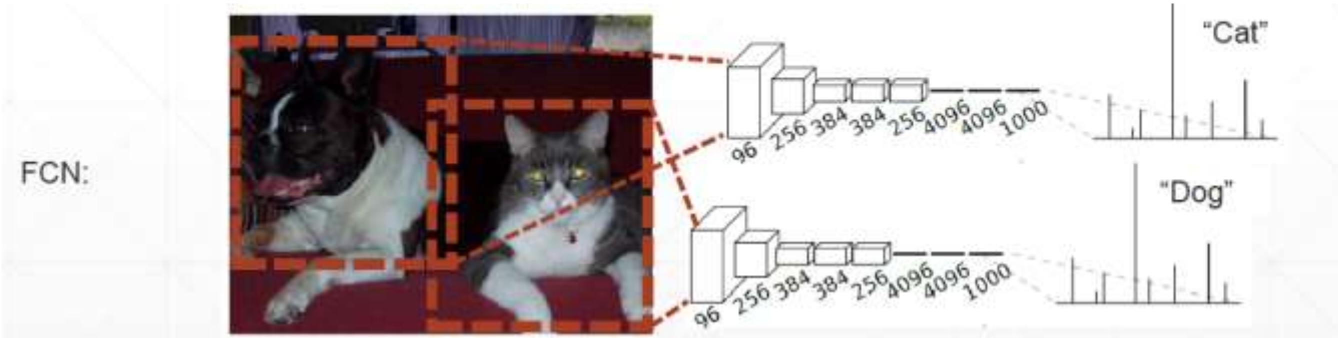
- Extending to a complete image

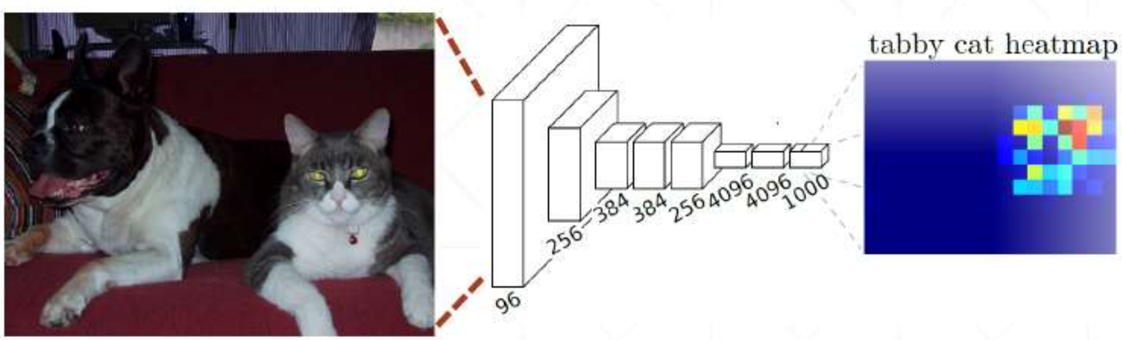




Network Design I: Efficiency

- Extending to a complete image





- Keep kernel sizes and strides
- Replace dense layer with convolution

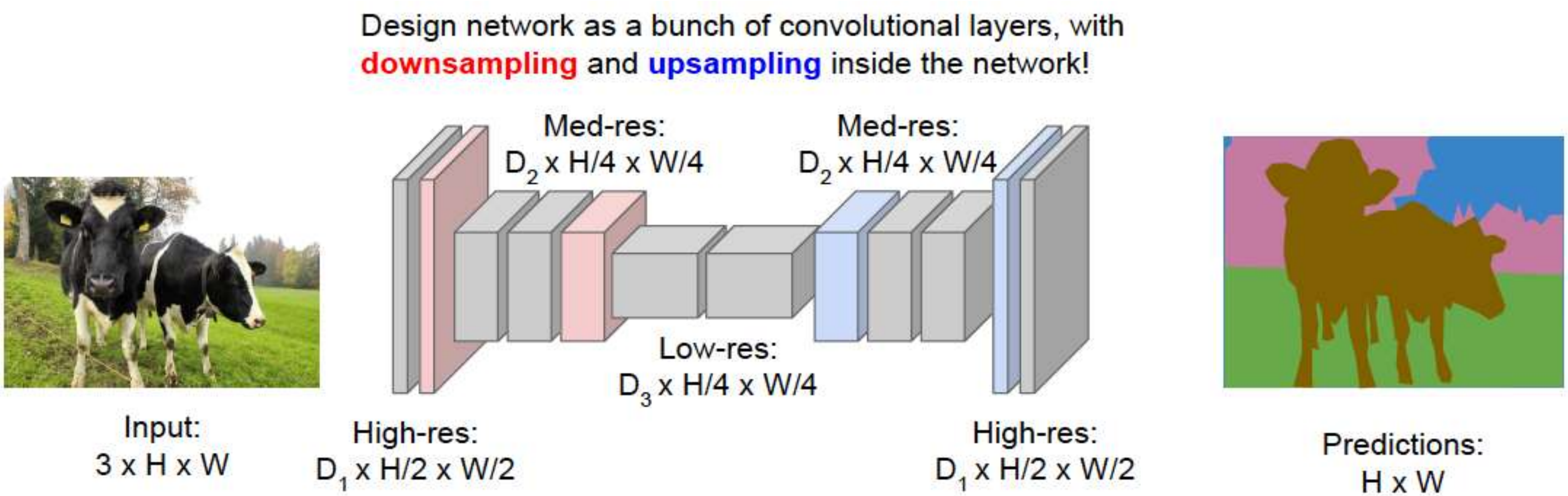
Network for semantic segmentation

- Main idea for dense predictionedicti1o
- Fully convolutional network
- Upsampling operators
- Multiscale context modeling

Acknowledgement: Feifei Li et al's cs231n notes

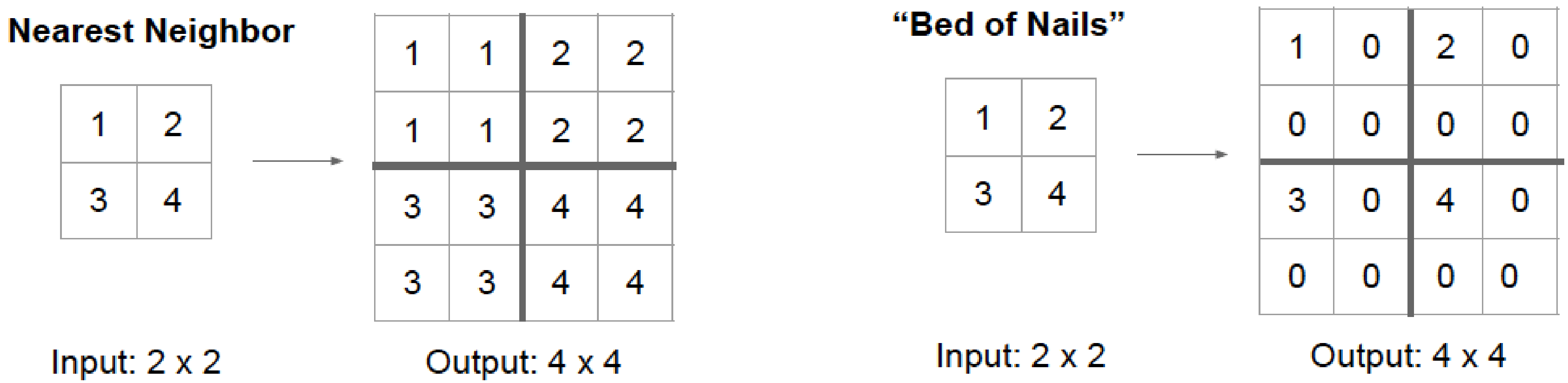
Network Design: Spatial resolution

- General encoder-decoder architecture



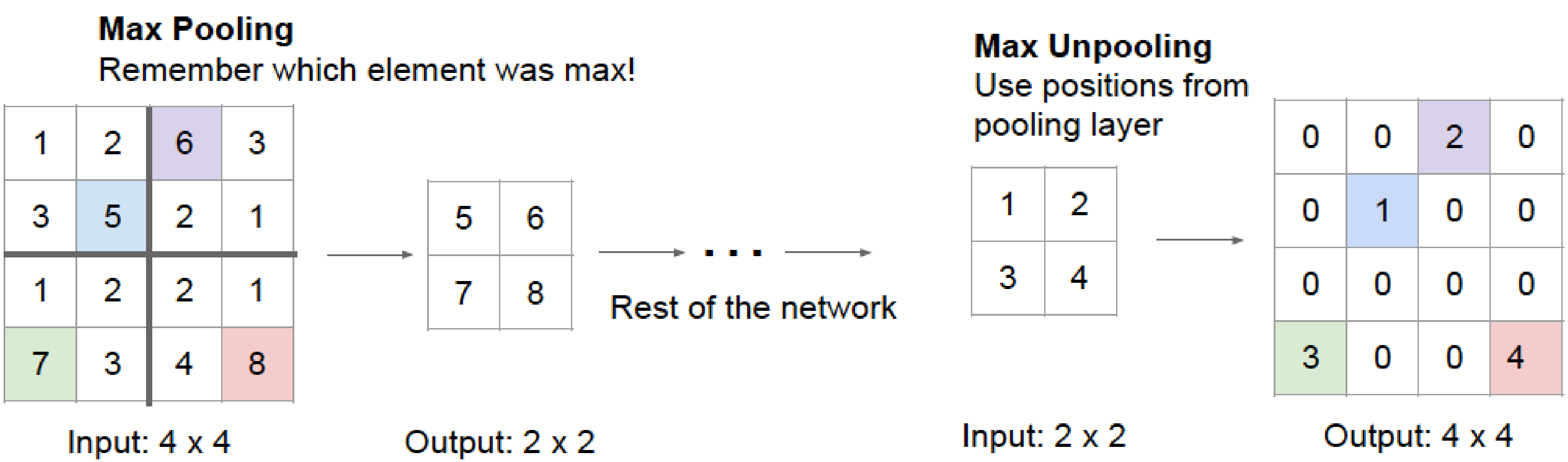
In-Network upsampling

- Unpooling

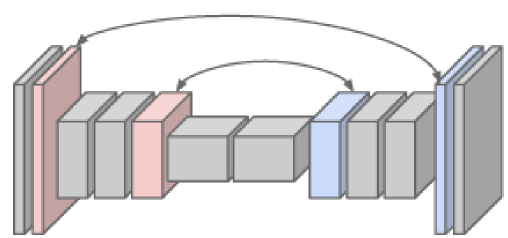


In-Network upsampling

- Max Unpooling



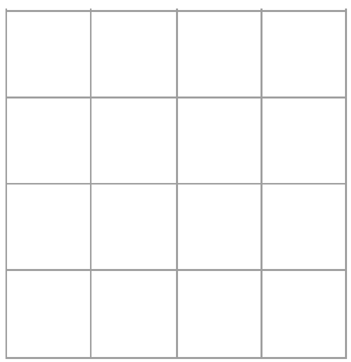
Corresponding pairs of downsampling and upsampling layers



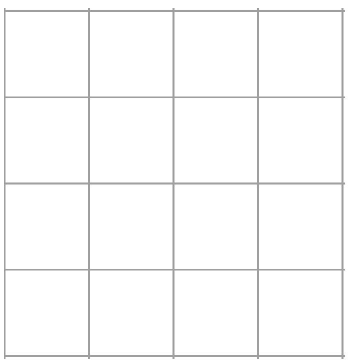
In-Network upsampling

- Learnable Upsampling: Transpose convolution

Recall: Typical 3 x 3 convolution, stride 1 pad 1



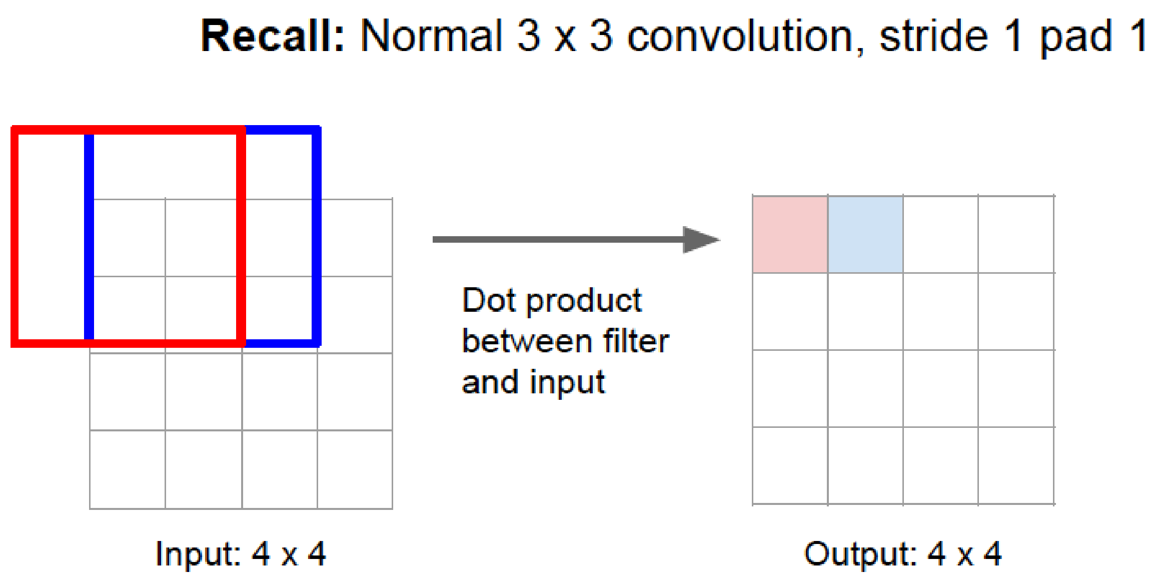
Input: 4 x 4



Output: 4 x 4

In-Network upsampling

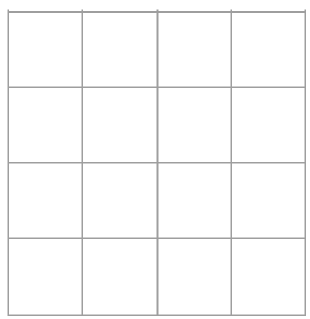
- Learnable Upsampling: Transpose convolution



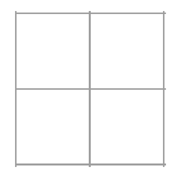
In-Network upsampling

- Learnable Upsampling: Transpose convolution

Recall: Normal 3 x 3 convolution, <u>stride 2</u> pad 1



Input: 4 x 4

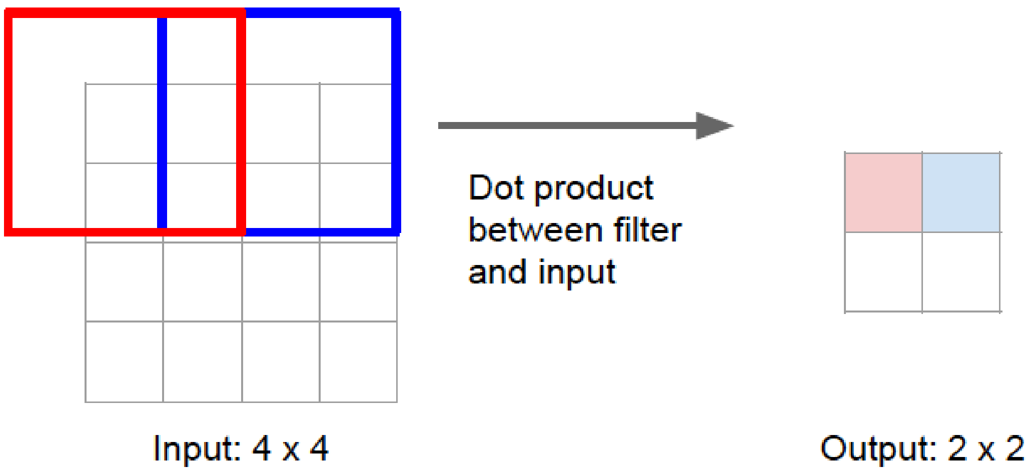


Output: 2 x 2

In-Network upsampling

- Learnable Upsampling: Transpose convolution

Recall: Normal 3 x 3 convolution, <u>stride 2</u> pad 1



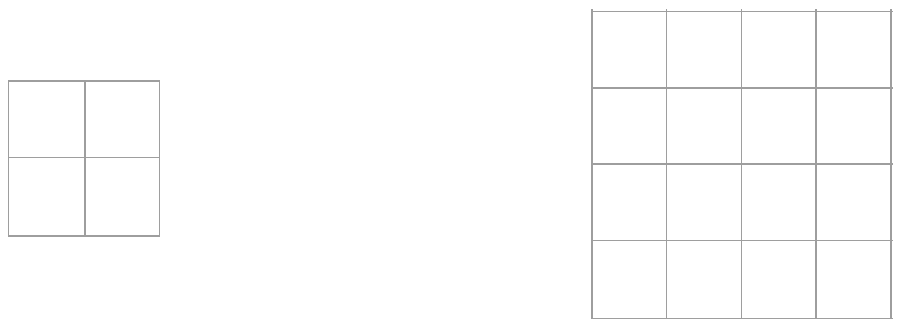
Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

In-Network upsampling

- Learnable Upsampling: Transpose convolution

3 x 3 **transpose** convolution, stride 2 pad 1



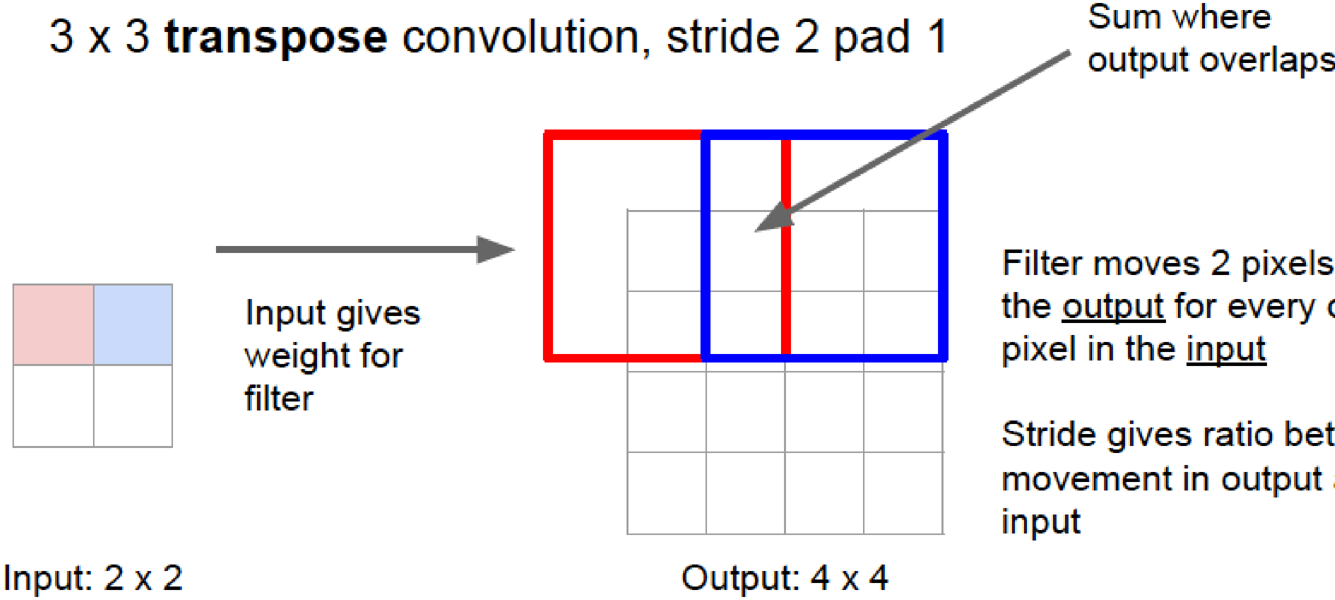
Input: 2 x 2

Output: 4 x 4

In-Network upsampling

- Learnable Upsampling: Transpose convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

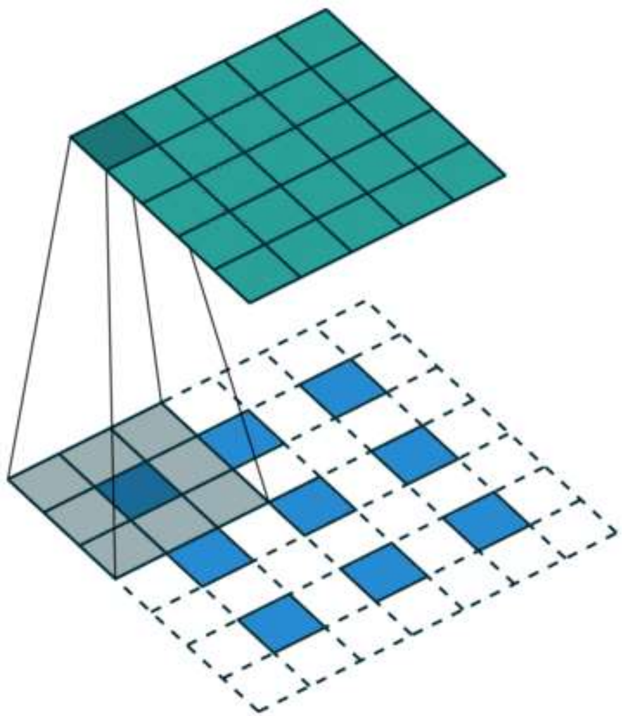


Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

In-Network upsampling

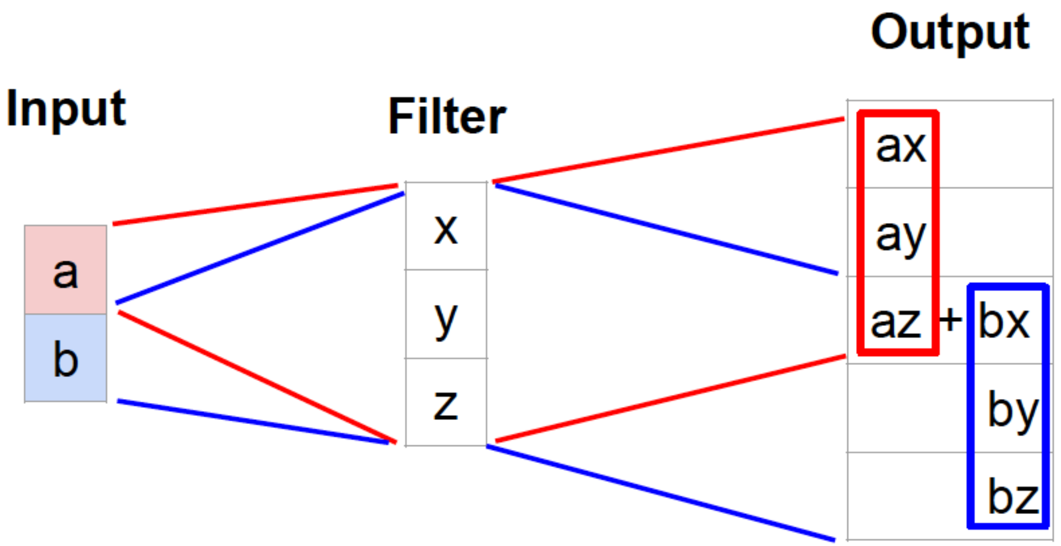
- 2D animation



https://github.com/vdumoulin/conv_arithmetic

In-Network upsampling

- Learnable Upsampling: Transpose convolution
  - 1D example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

In-Network upsampling

- Learnable Upsampling: Transpose convolution
  - 1D example

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

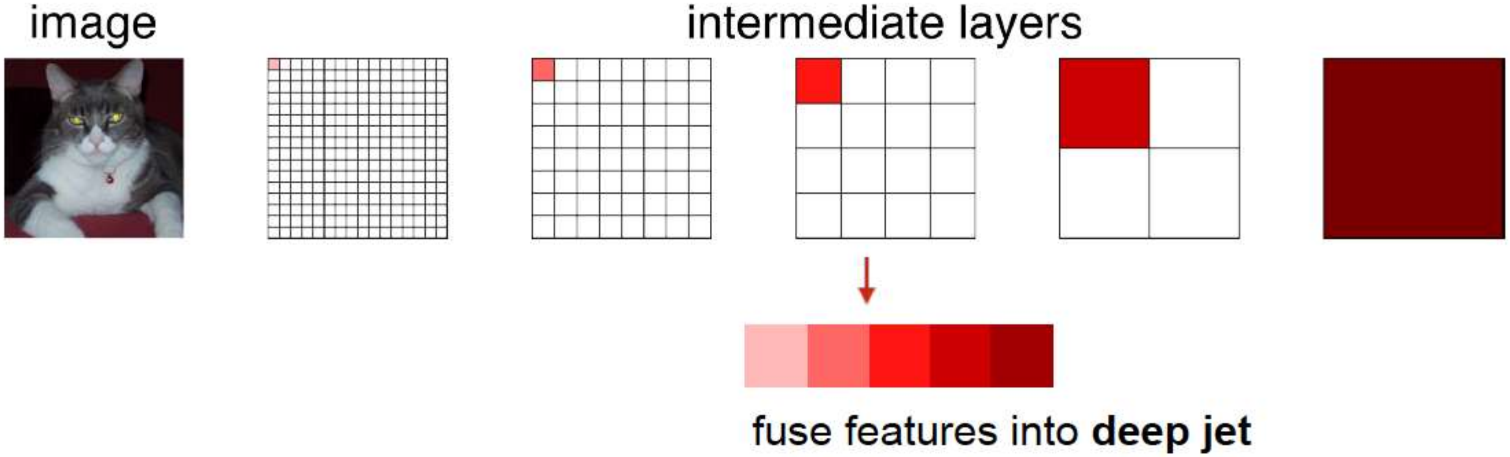
$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, convolution transpose is no longer a normal convolution!

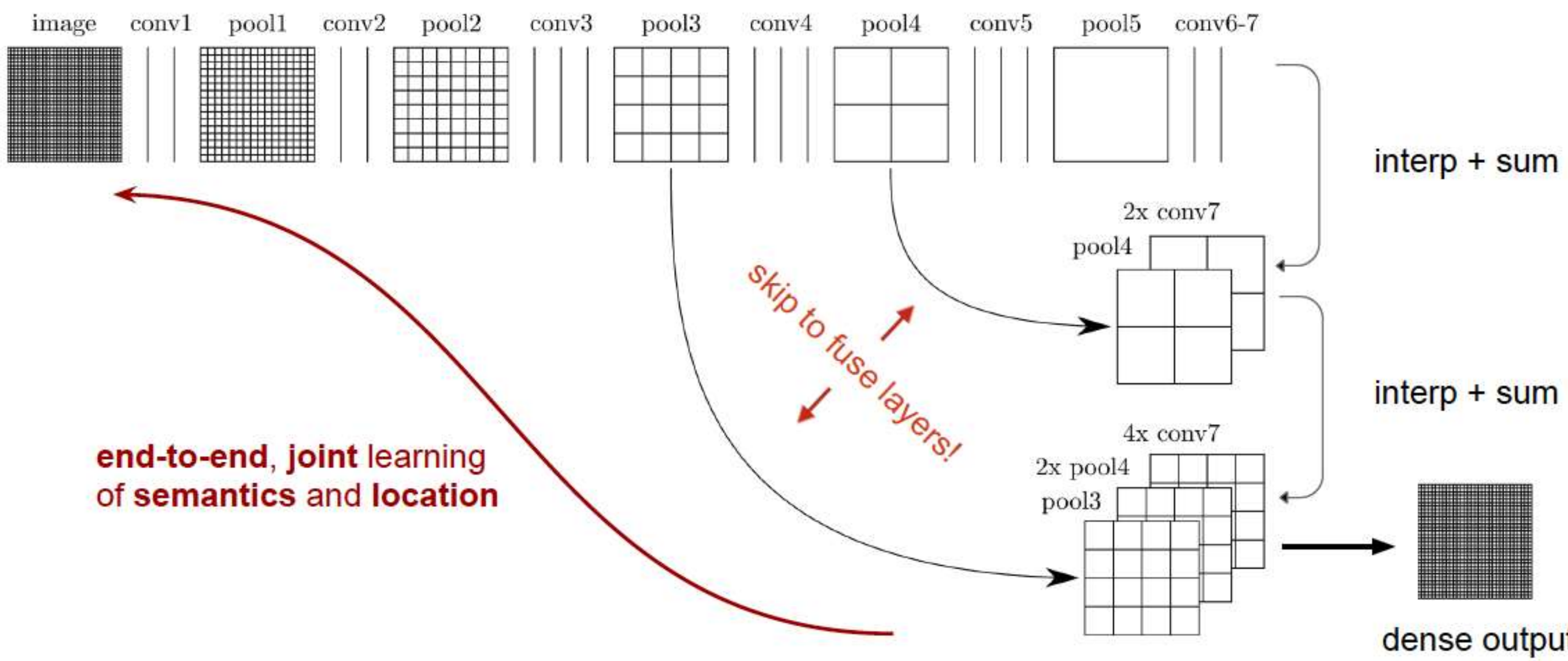
Network Design: Examples

- Fully Convolutional Network [Long et al, CVPR 2015]
  - Upsampling: low-resolution, lack spatial details
  - Combining *where (local, shallow)* with *what (global, deep)*



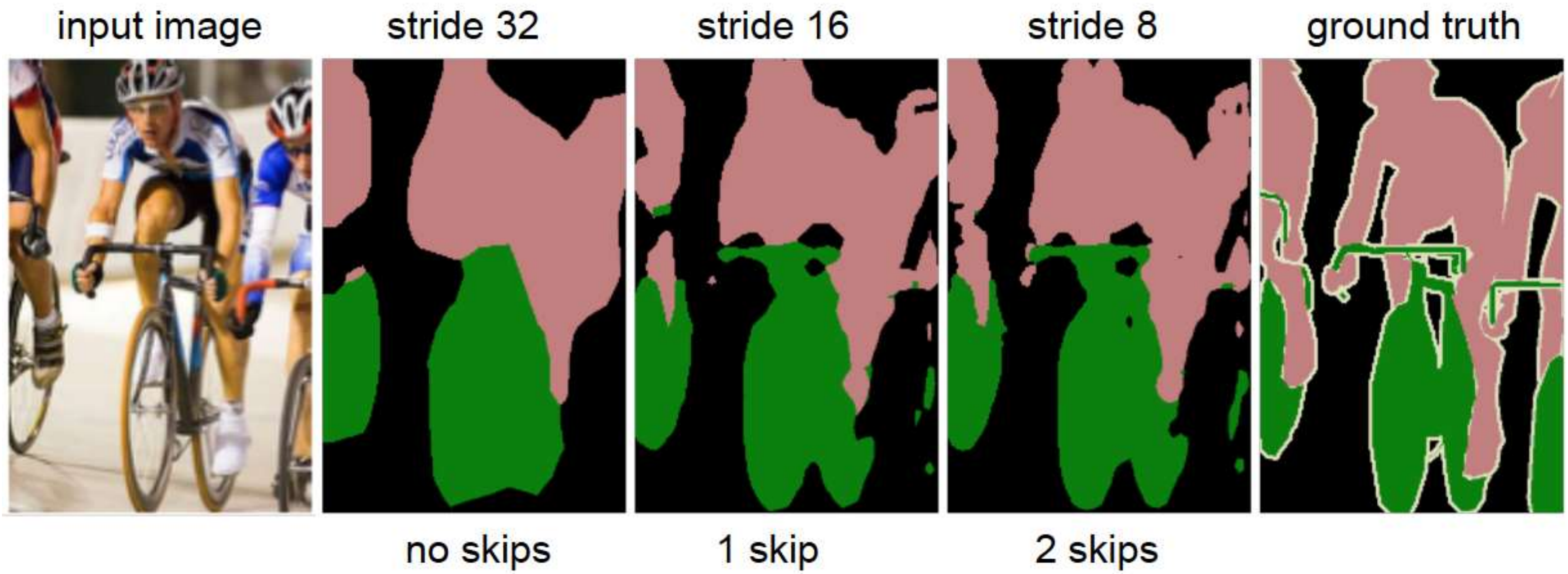
Network Design: Examples

- Fully Convolutional Network [Long et al, CVPR 2015]
  - Upsampling: low-resolution, lack spatial details
  - Introducing ***skip layers***



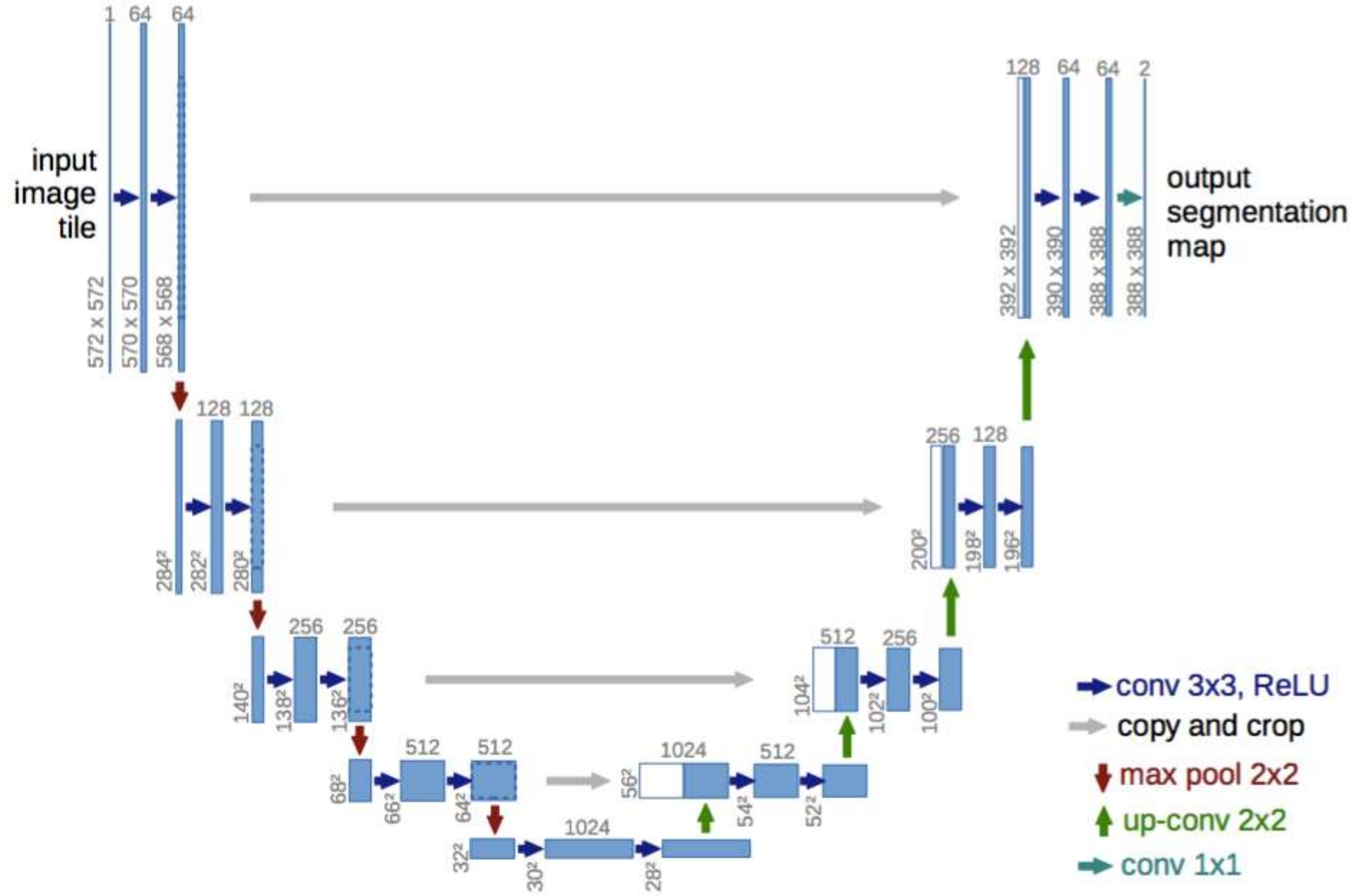
Network Design: Examples

- Fully Convolutional Network [Long et al, CVPR 2015]
  - Upsampling: low-resolution, lack spatial details
  - Skip layer refinement



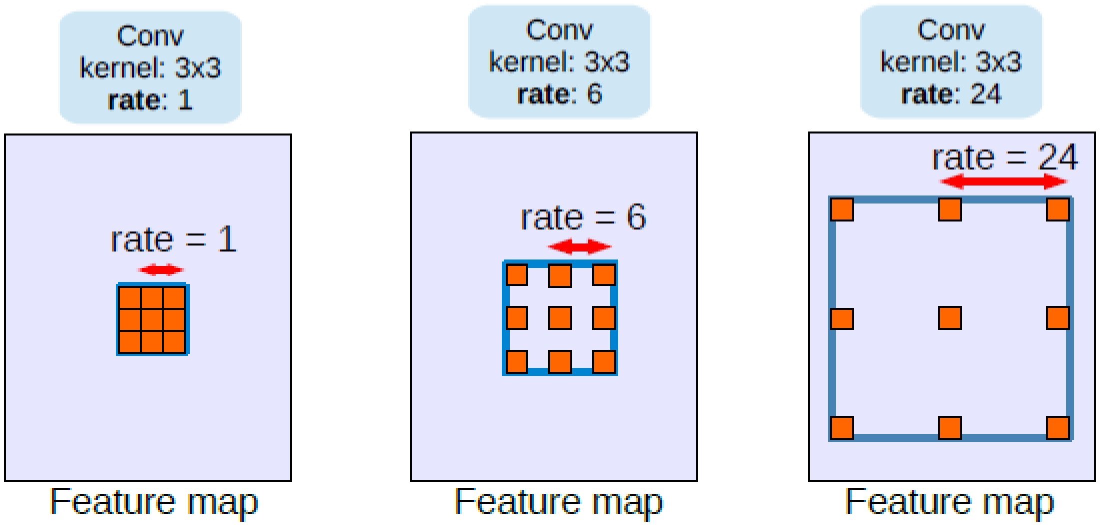
Network Design: Examples

- U-Net [Ronneberger et al, MICCAI 2015]



Network Design: Spatial resolution

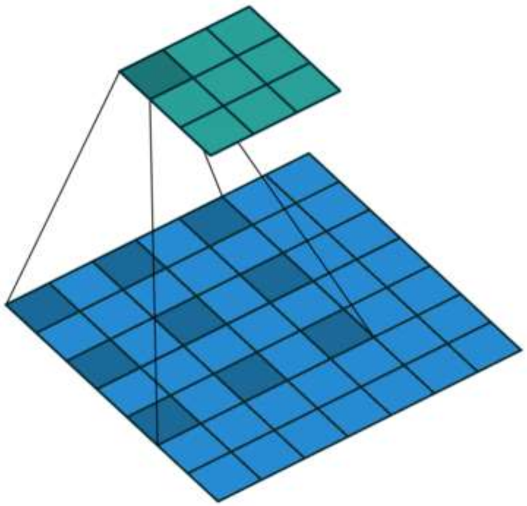
- Dilated Convolutional Network [Yu and Koltun, ICLR 2016]
  - Dense feature map without upsampling
  - ***Dilated (or Atrous) convolution***



$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]w[k].$$

Network Design: Spatial resolution

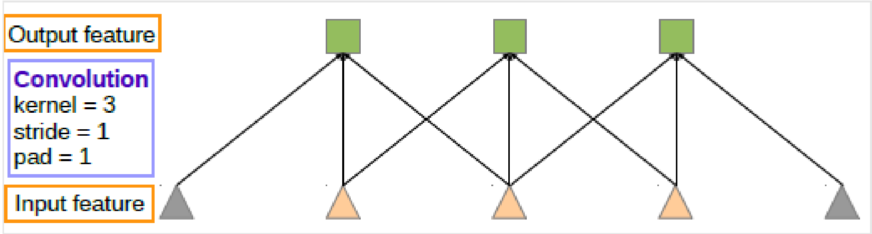
- Dilated Convolutional Network [Yu and Koltun, ICLR 2016]
  - Dense feature map without upsampling
  - ***Dilated (or Atrous) convolution***



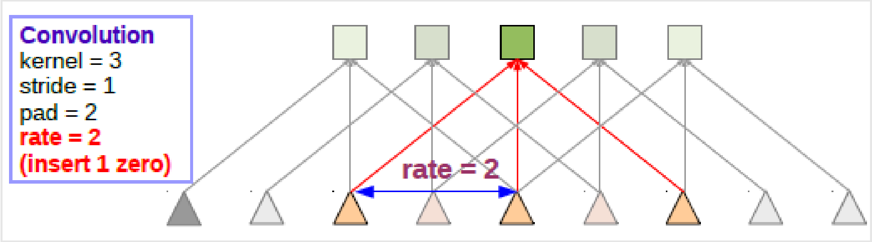
$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]w[k].$$

Network Design: Spatial resolution

- Dilated Convolutional Network [Yu and Koltun, ICLR 2016]
  - Dense feature map without upsampling
  - ***Dilated (or Atrous) convolution***



(a) Sparse feature extraction

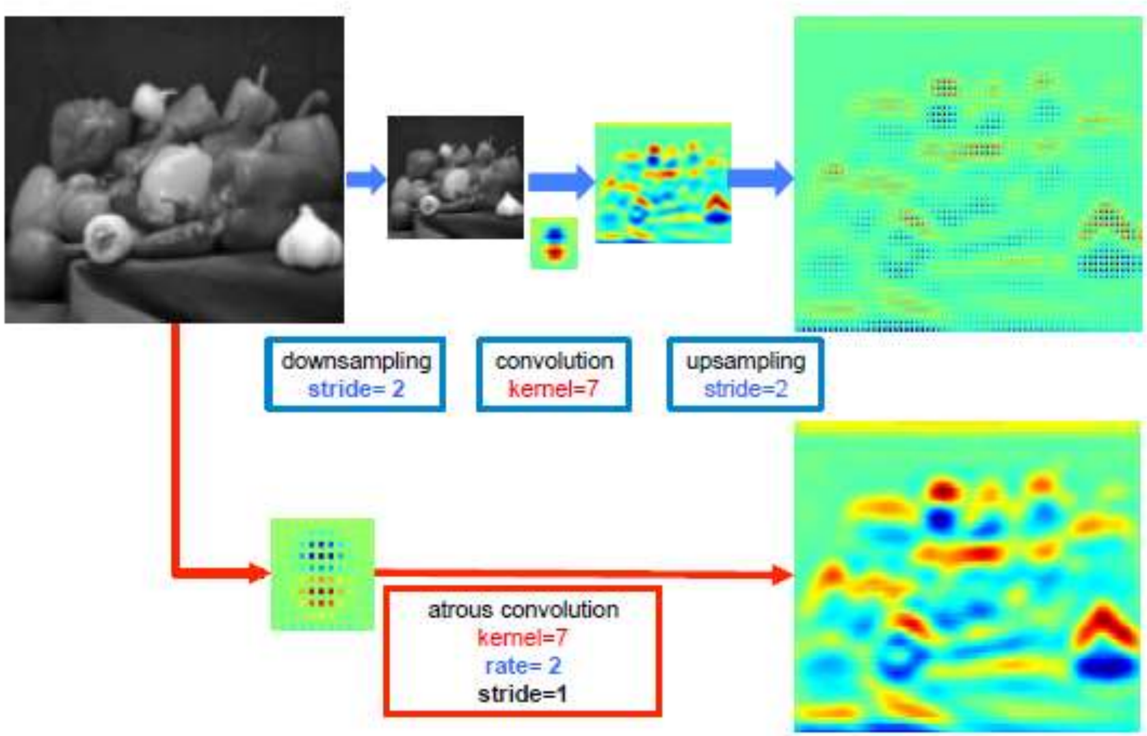


(b) Dense feature extraction

$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]w[k].$$

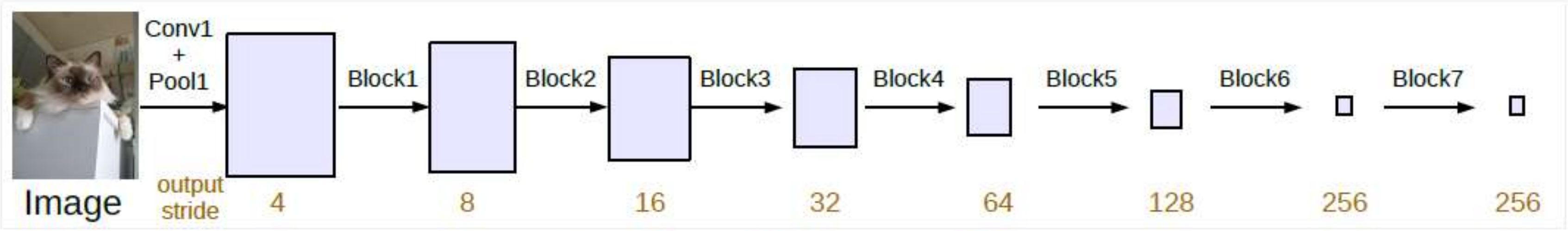
Network Design: Spatial resolution

- Dilated Convolutional Network [Yu and Koltun, ICLR 2016]
  - Dense feature map without upsampling
  - ***Dilated (or Atrous) convolution***

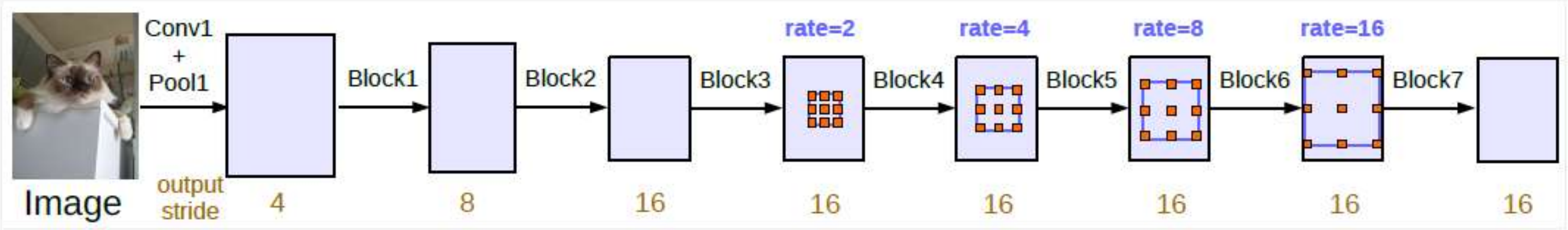


Network Design: Spatial resolution

- Dilated Convolutional Network [Yu and Koltun, ICLR 2016]
  - Dense feature map without upsampling
  - ***Dilated (or Atrous) convolution***



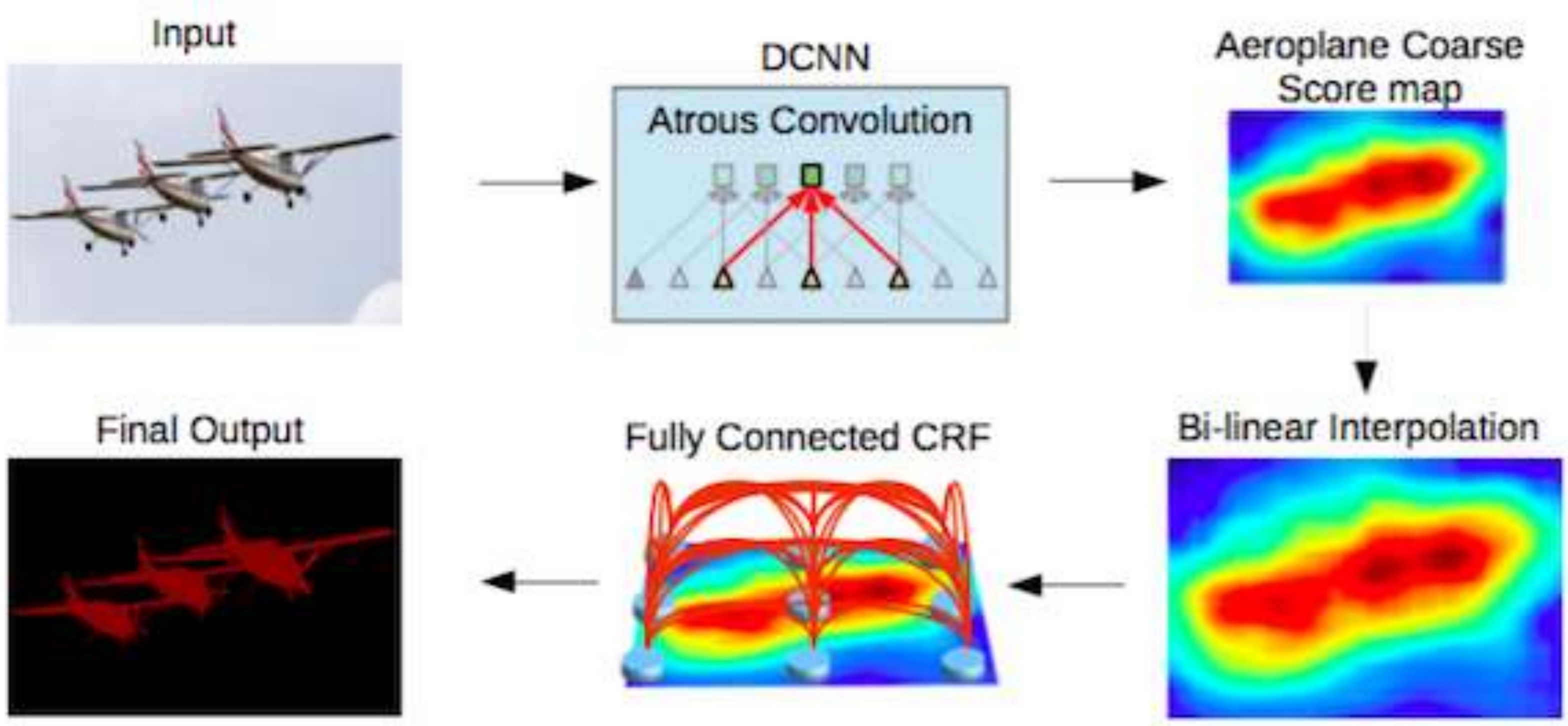
(a) Going deeper without atrous convolution.



(b) Going deeper with atrous convolution. Atrous convolution with $rate > 1$ is applied after block3 when $output\_stride = 16$.

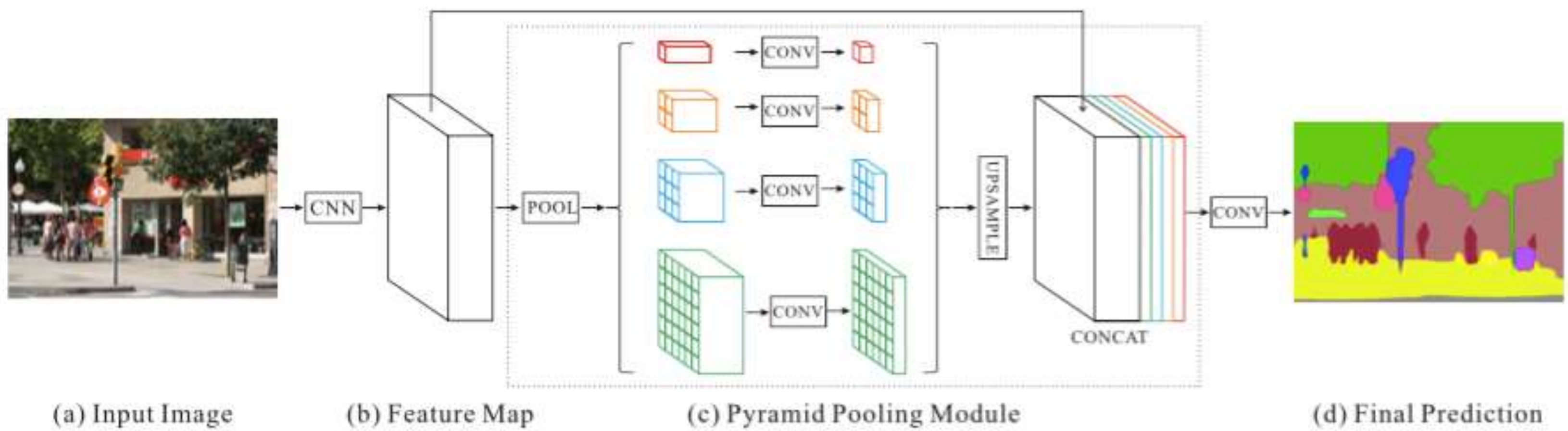
Network Design: Multi-scale context

- DeepLab v1&v2
  - Post-processing with dense CRFs.



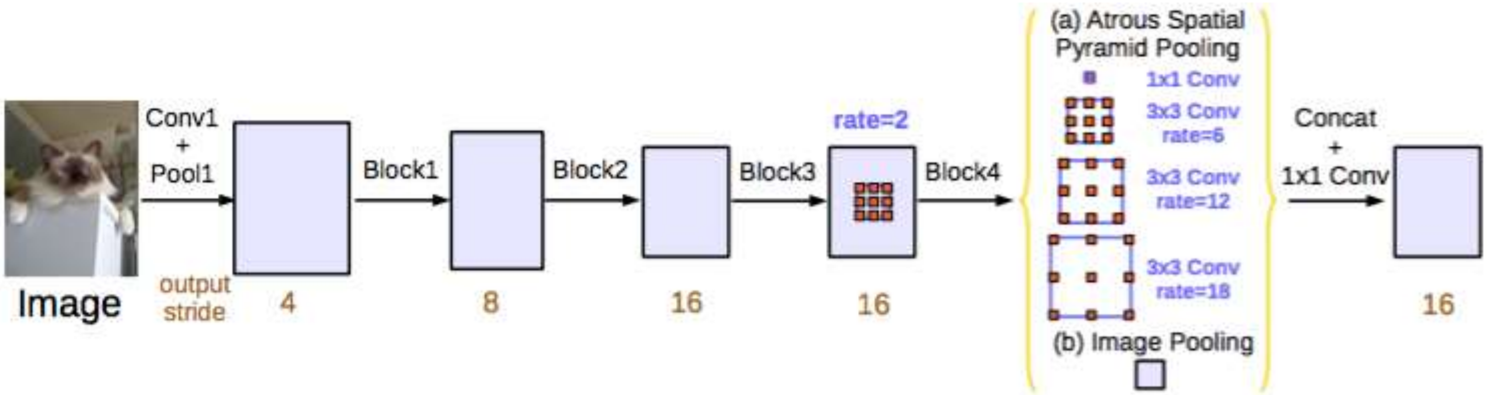
Network Design: Multi-scale context

- PSPNet [Zhao et al CVPR 2017]
  - A pyramid parsing module that carries both local and global context information

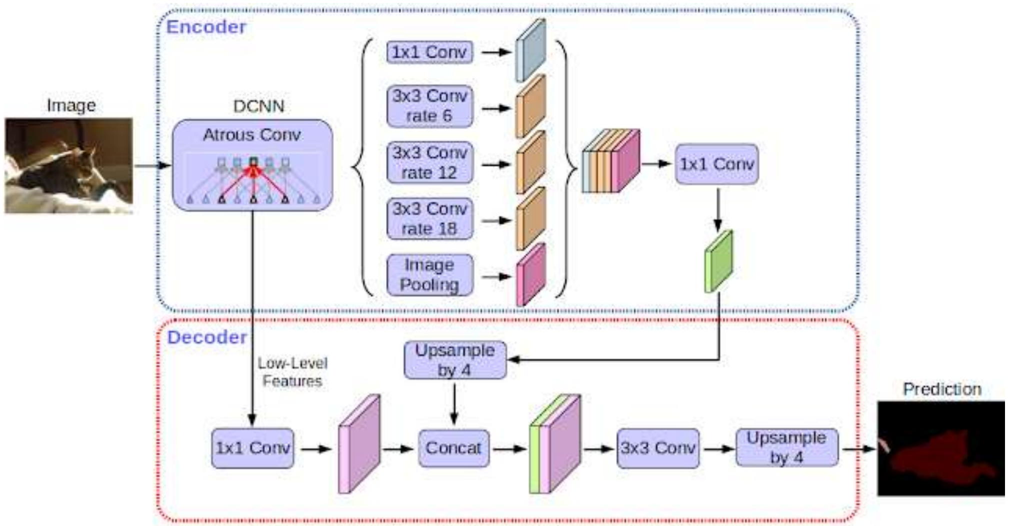


Network Design: Multi-scale context

- DeepLab v3



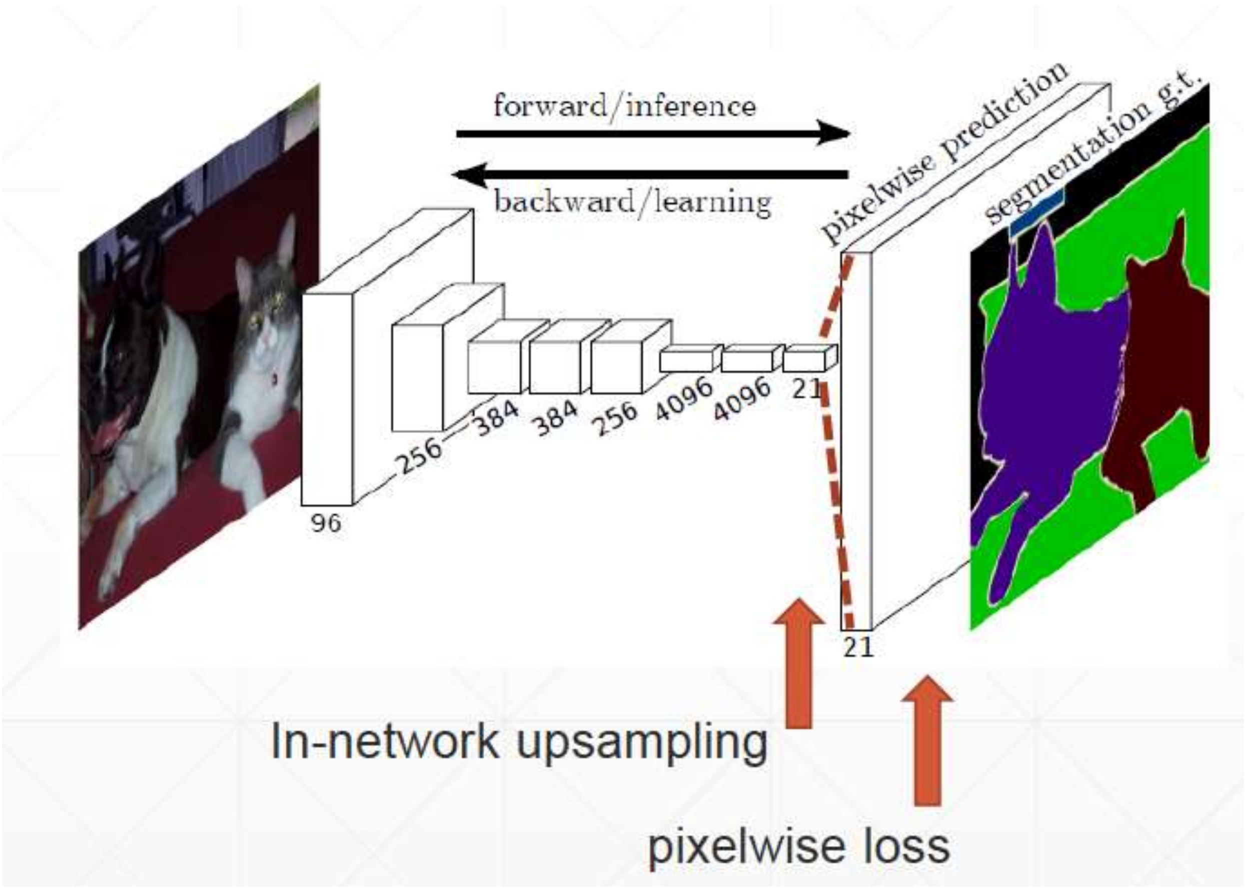
- Deeplab v3+



https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html

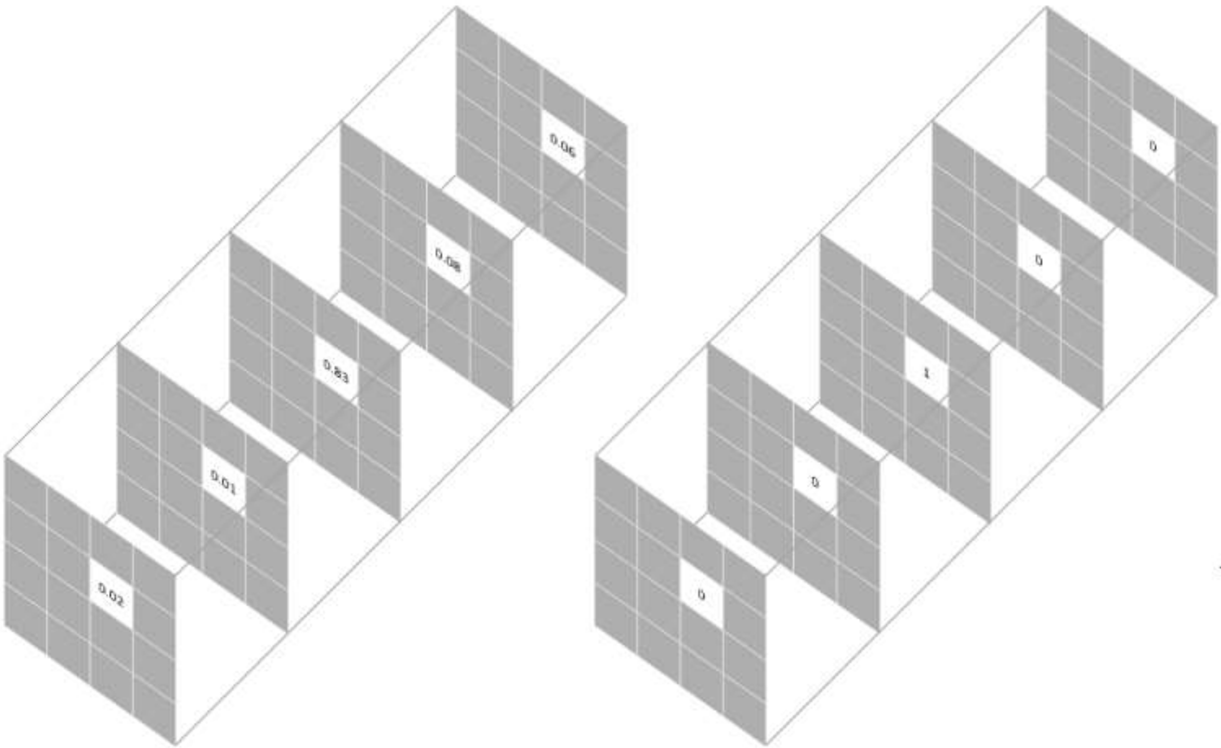
Semantic segmentation: loss function

- Main idea: pixel-wise classification



Semantic segmentation: loss function

- Pixel-wise loss



Prediction for a selected pixel

Target for the corresponding pixel

Pixel-wise loss is calculated as the log loss, summed over all possible classes

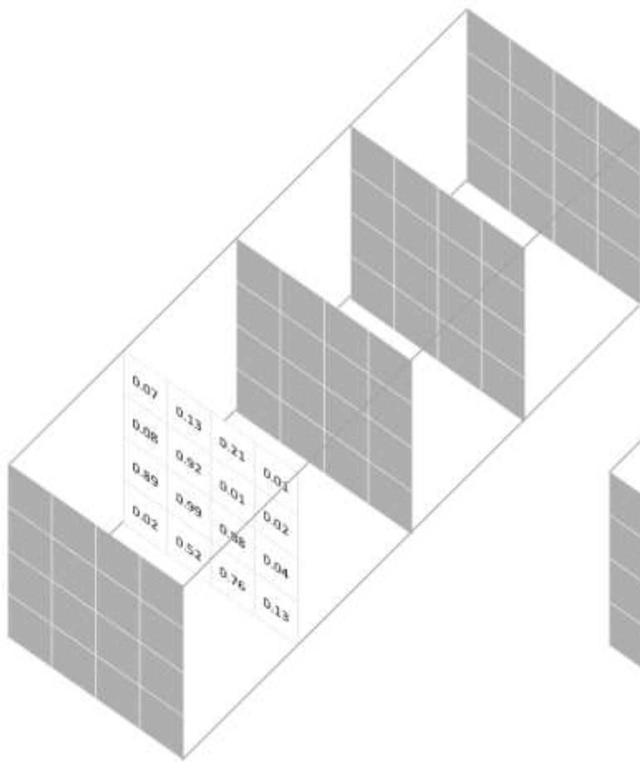
$$-\sum_{classes} y_{true} \log\left(y_{pred}\right)$$

This scoring is repeated over all **pixels** and averaged

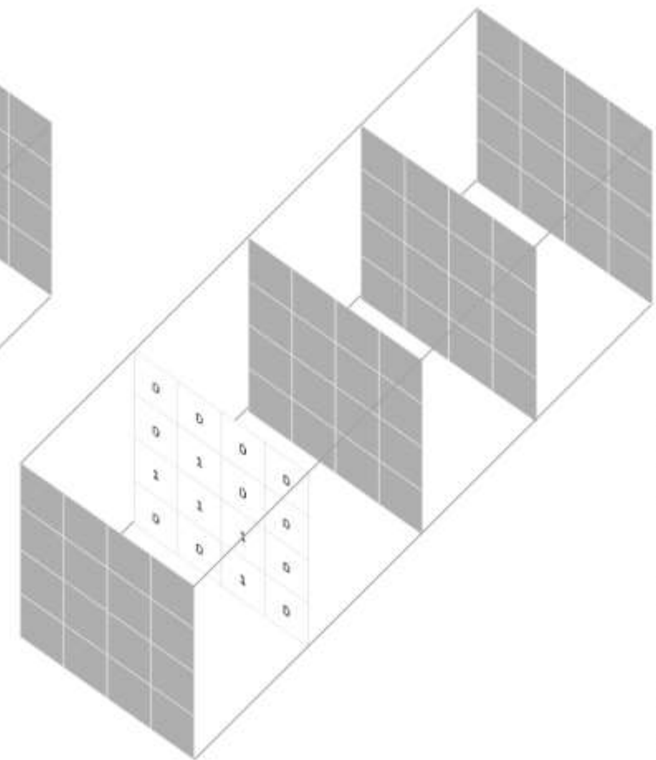
Semantic segmentation: loss function

- Region-based loss

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$



Prediction for a selected class



Target for the corresponding class

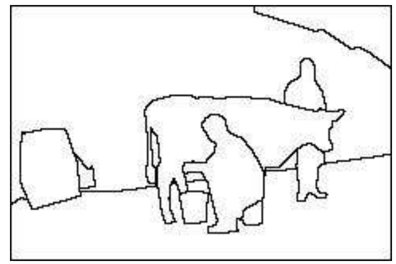
Soft Dice coefficient is calculated for each class mask

$$1 - \frac{2\sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2}$$

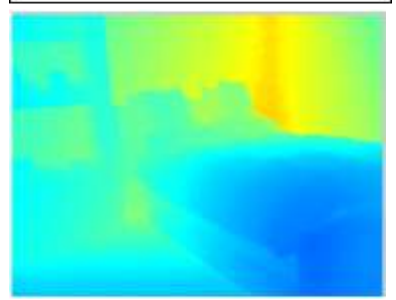
This scoring is repeated over all **classes** and averaged

Semantic Segmentation: Summary

- Pixel-wise annotation of images
  - An instance of scene understanding

Boundary

Depth

- Other research topics (not discussed)
  - *Low-level vision: superresolution, deblurring, inpainting, depth*
  - *Video: optical flow, action and activity recognition and detection*
  - *Volumetric/Multimodality: RGB-D images, medical imaging, etc.*

Outline

- CNN applications in dense prediction
- Recurrent Neural Networks
  - Sequence modeling, Autoregressive models
  - (Vanilla) RNN models
- Backpropagation through time
  - Computational graph
- Example: language modeling
  - Neural language models

Acknowledgement: Feifei Li et al's cs231n notes

Sequence modeling

- Modeling a sequence of tokens
  - Running example: sentences
- Goal: learn/build a good distribution of sentences

- Inputs: a corpus of sentences $\mathbf{s}^{(1)}, \cdots, \mathbf{s}^{(N)}$
- Output: a distribution $p(\mathbf{s})$

- Common approach: maximum likelihood
  - Assume sentences are independent

$$\max \prod_{i=1}^{N} p(\mathbf{s}^{(i)})$$

Sequence modeling

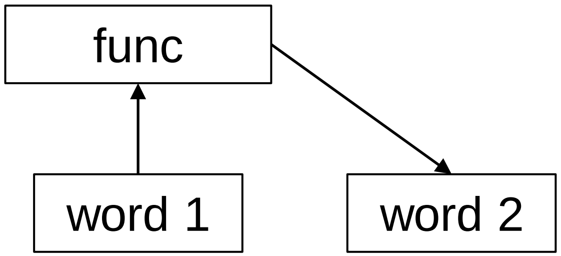
- What is $p(\mathbf{s})$?
- A sentence is a sequence of words $w_1, w_2, \cdots, w_T$.

$$p(\mathbf{s}) = p(w_1, \ldots, w_T) = p(w_1)p(w_2 \mid w_1) \cdots p(w_T \mid w_1, \ldots, w_{T-1}).$$

  - Essentially aim to predict the next word
- Markovian assumption
  - The distribution over the next word depends on the preceding few words. For example,

$$p(w_t \mid w_1, \ldots, w_{t-1}) = p(w_t \mid w_{t-3}, w_{t-2}, w_{t-1}).$$

  - Autoregressive model
    - Memoryless
    - Can be modeled by a parametrized function



Traditional language models

- N-Gram model
  - Autoregressive model: Markov assumption
  - Use a conditional probability table

	cat	and	city	$\cdots$
the fat	0.21	0.003	0.01	
four score	0.0001	0.55	0.0001	$\cdots$
New York	0.002	0.0001	0.48	
$\vdots$		$\vdots$		

  - Estimate the probabilities from the empirical distribution

$$p(w_3 = \mathrm{cat} \mid w_1 = \mathrm{the}, w_2 = \mathrm{fat}) = \frac{\mathrm{count}(\mathrm{the\ fat\ cat})}{\mathrm{count}(\mathrm{the\ fat})}$$

  - The phrases we're counting are called n-grams (where n is the length), so this is an n-gram language model.
    - Note: the above example is considered a 3-gram model, not a 2-gram model!

Traditional language models

- Problems with n-gram language models
  - The number of entries in the conditional probability table is exponential in the context length
  - Data sparsity: most n-grams never appear in the corpus
- Solutions
  - Use a short context (less expressive)
  - Smooth the probabilities (priors)
  - Using an ensemble of n-gram models with different n

Neural language model

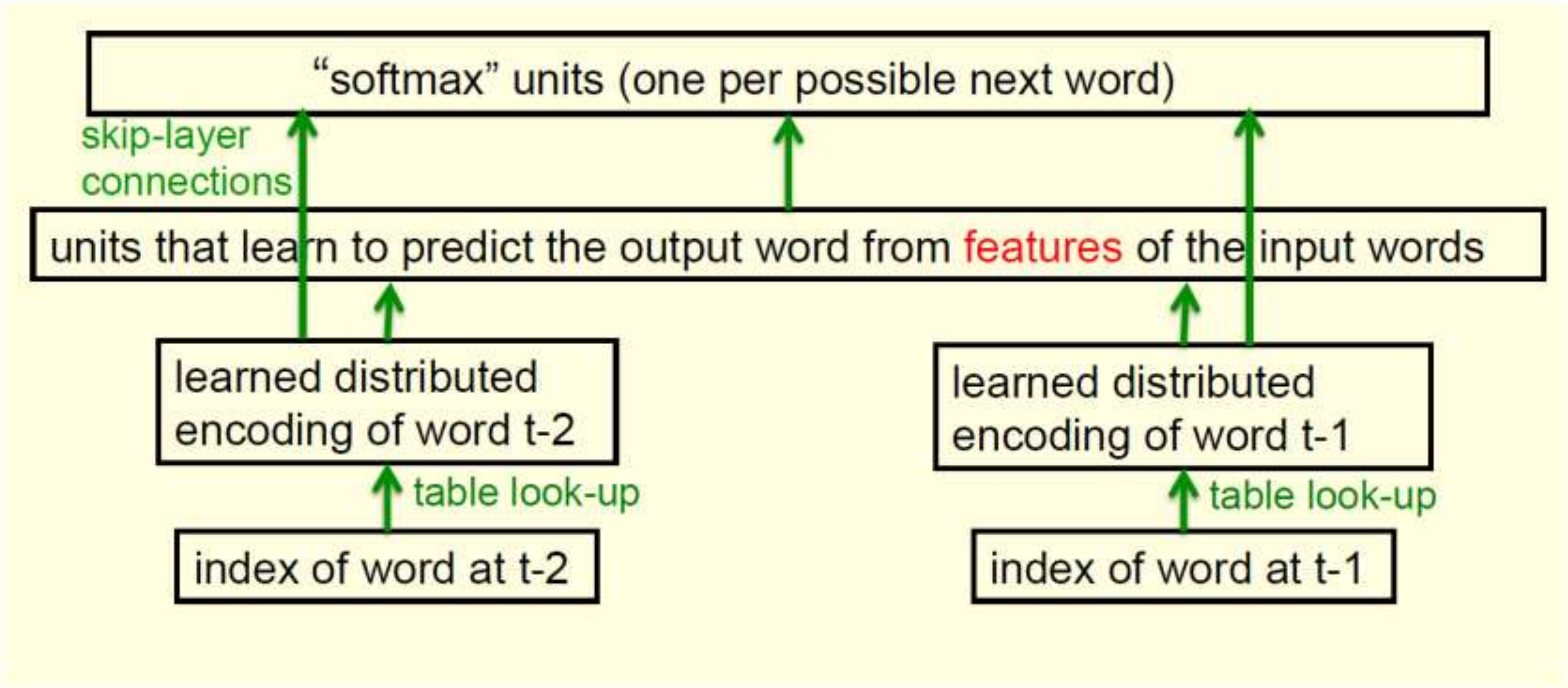
- Predicting the distribution of the next word given the previous K is a multiway classification problem
  - Inputs: previous K words
  - Output/Target: next word
  - Loss: cross-entropy

$$
\begin{aligned}
-\log p(\mathbf{s}) &= -\log \prod_{t=1}^{T} p(w_t \mid w_1, \ldots, w_{t-1}) \\
&= -\sum_{t=1}^{T} \log p(w_t \mid w_1, \ldots, w_{t-1}) \\
&= -\sum_{t=1}^{T} \sum_{v=1}^{V} t_{tv} \log y_{tv},
\end{aligned}
$$

where t_{iv} is the one-hot encoding for the ith word and y_{iv} is the predicted probability for the ith word being index v.

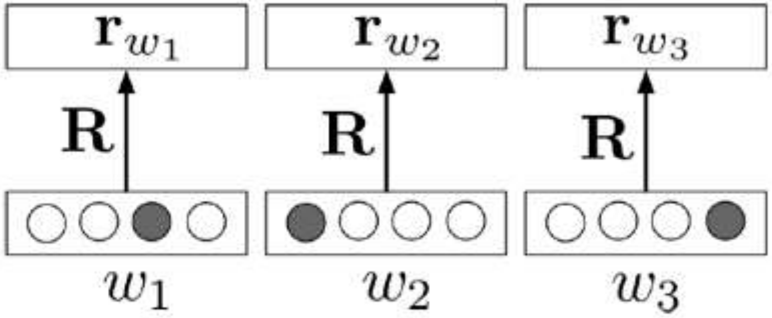
Neural language model

- Model structure (context length = 2)



Neural language model

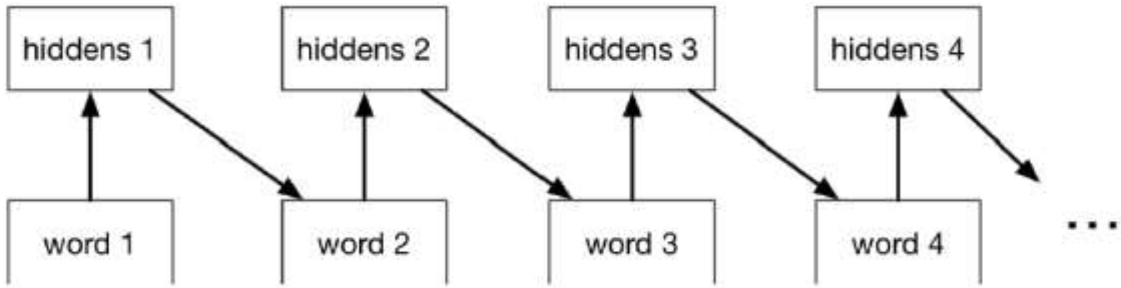
- Word embedding
  - If we use a 1-of-K encoding for the words, the first layer can be thought of as a linear layer with tied weights.



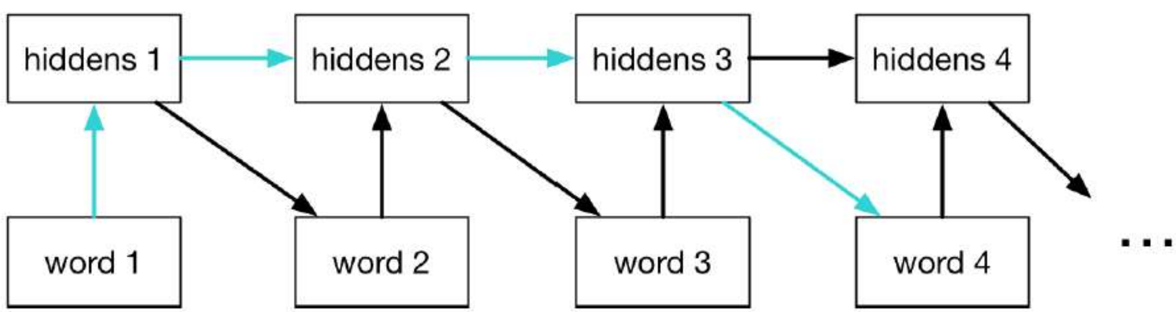
  - The weight matrix basically acts like a lookup table. Each column is the representation of a word, also called an embedding, feature vector, or encoding.
    - "Embedding" emphasizes that it's a location in a high-dimensonal space; words that are closer together are more semantically similar
    - "Feature vector" emphasizes that it's a vector that can be used for making predictions, just like other feature mappigns we've looked at (e.g. polynomials)

Sequence modeling

- Problems?
- Autoregressive models are memoryless
  - Can only use information from their immediate context

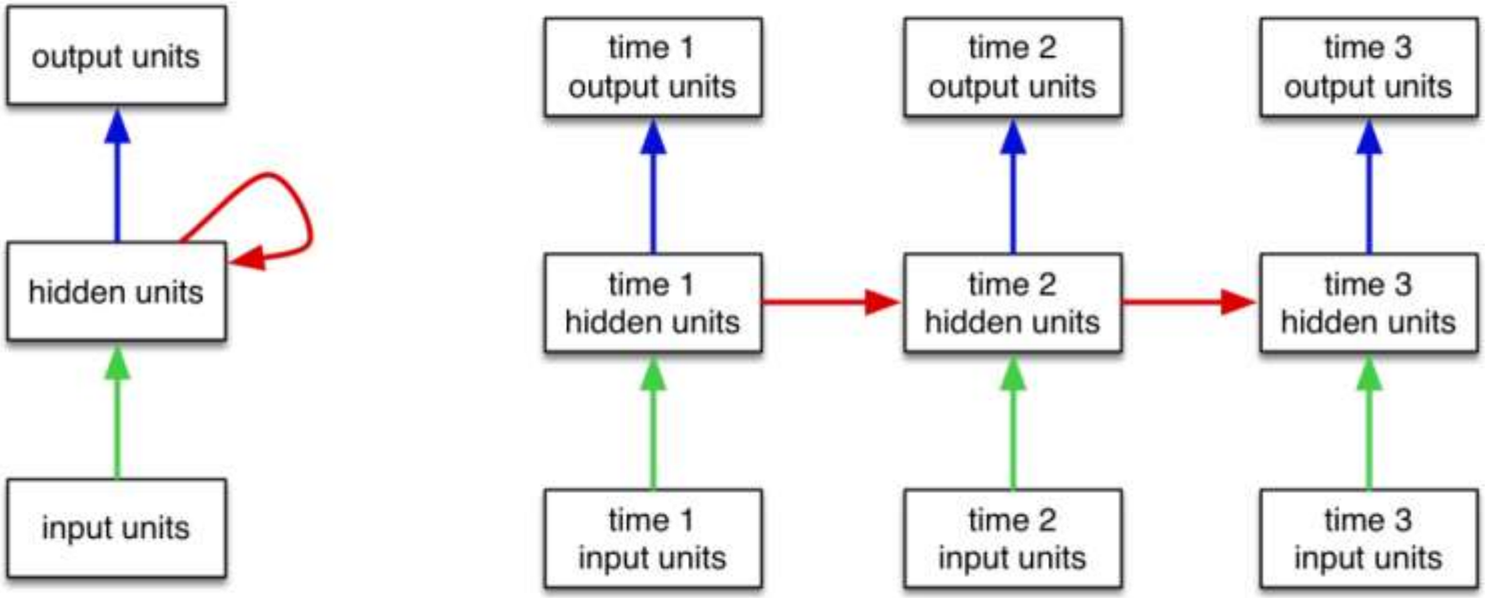


- Adding connections between hidden units
  - Having a memory lets the model use longer-term dependencies



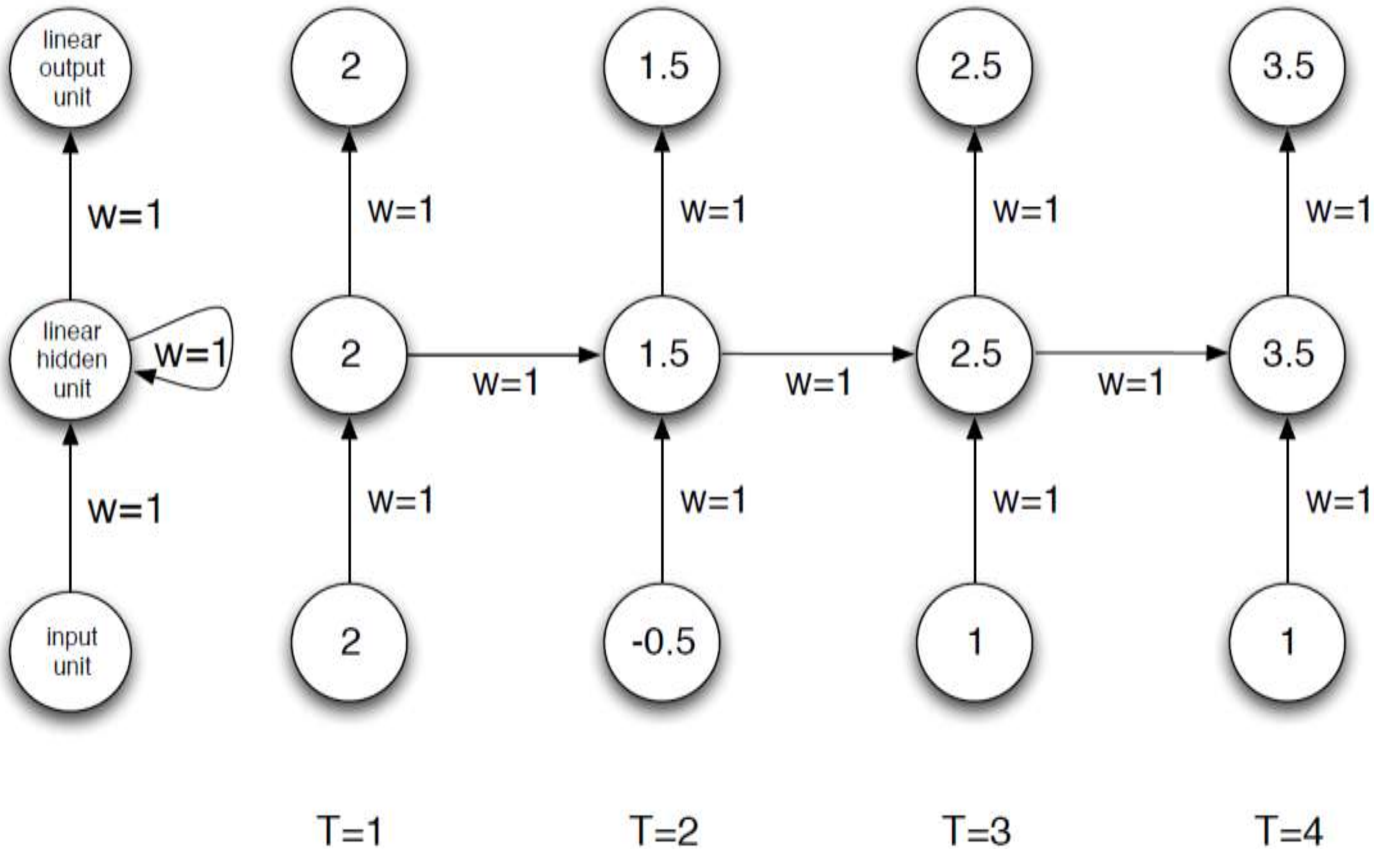
Recurrent Neural Network

- Recurrent Neural Network as a dynamical system with one set of hidden units feeding into themselves
  - The network's graph has self-loops
- The RNN's graph can be unrolled by explicitly representing the units at all time steps
  - The weights and biases are shared



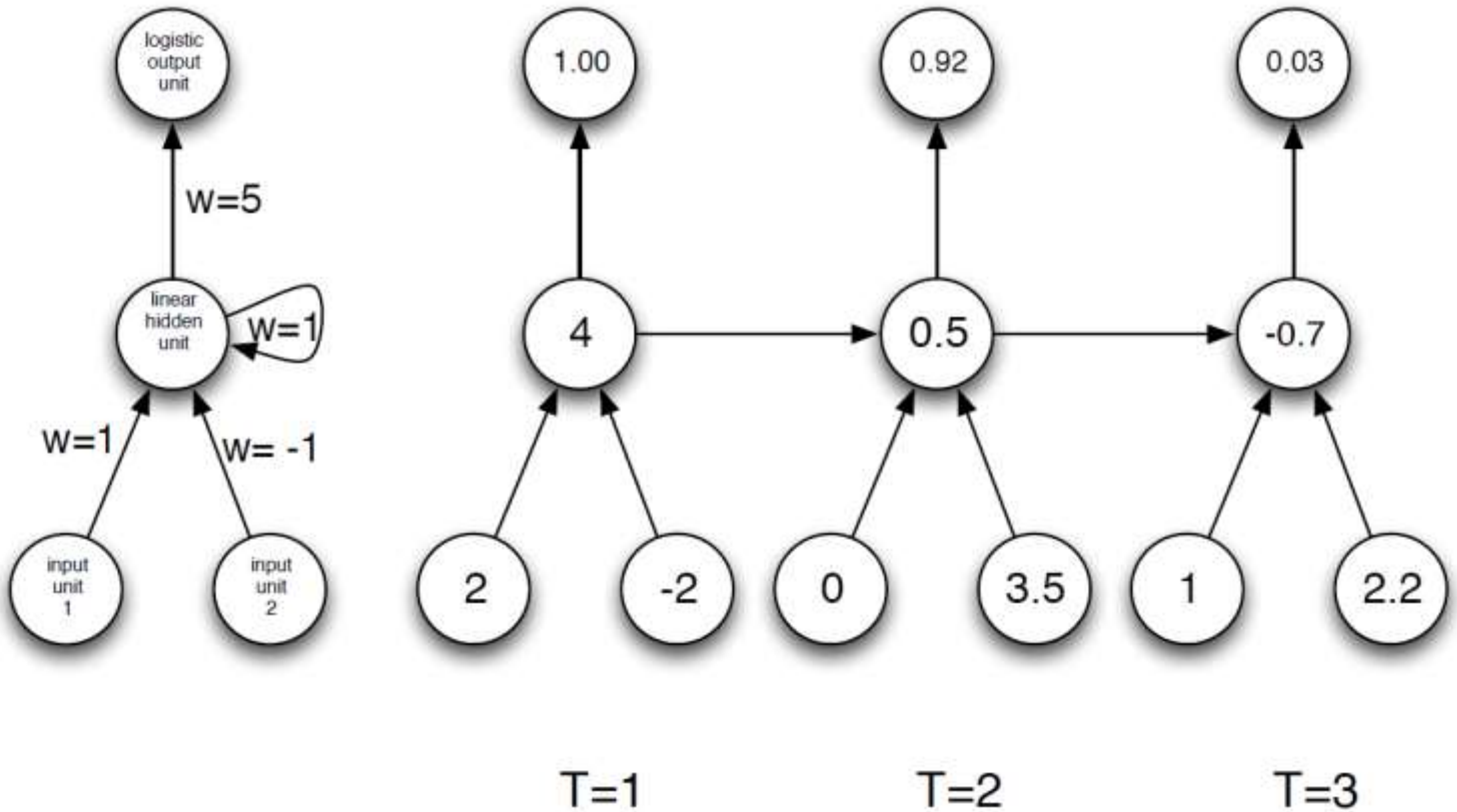
RNN examples

- Summation network



RNN examples

- Summation & comparison network



RNN examples

- Parity-check network
- Problem: determine the parity of a sequence of binary inputs

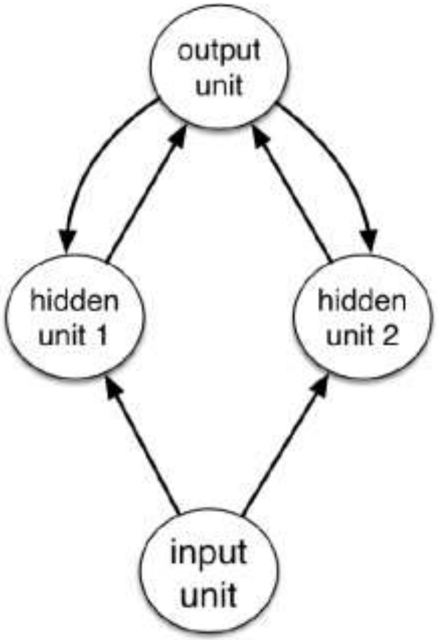
Parity bits: 0 1 1 0 1 1 $\longrightarrow$

Input: 0 1 0 1 1 0 1 0 1 1

  - Each parity bit is the XOR of the input and the previous parity bit
  - Hard to solve with a shallow feed-forward network

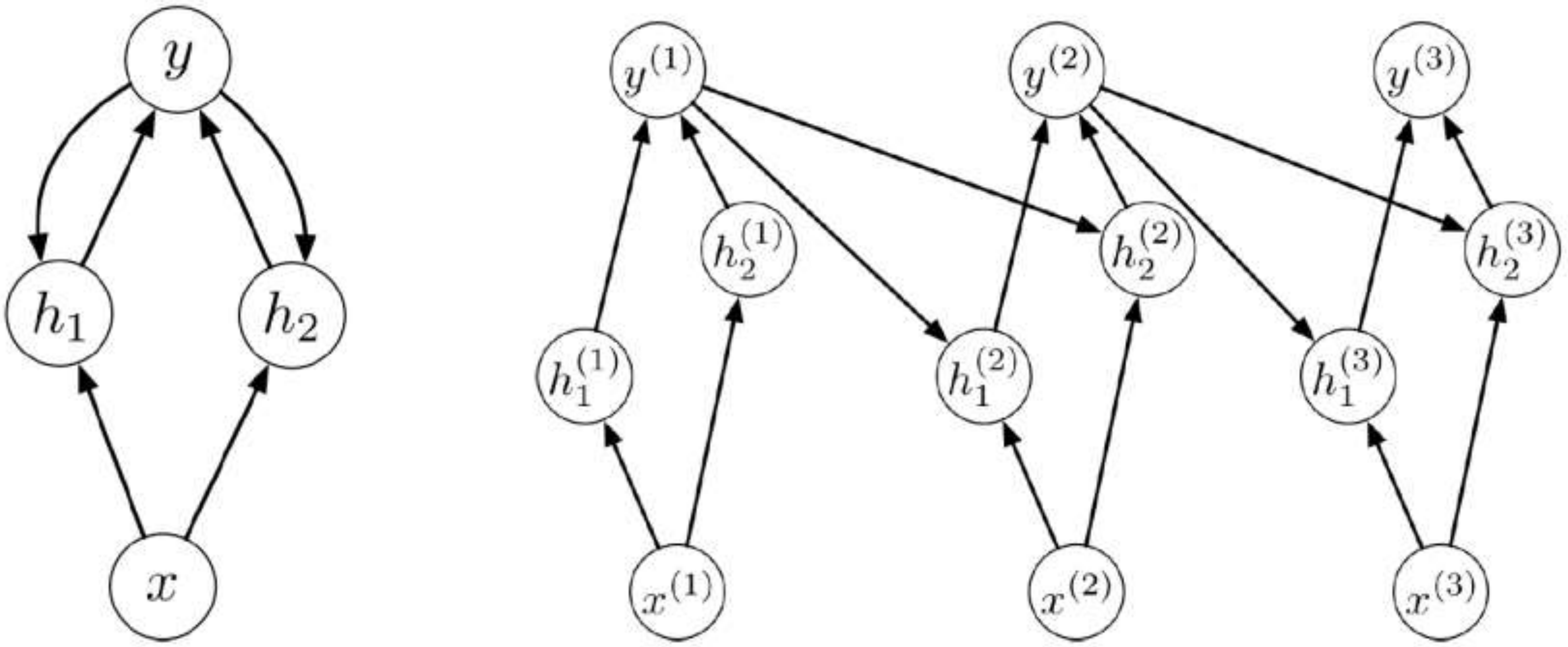
RNN examples

- Parity-check network
- Problem: determine the parity of a sequence of binary inputs
  - Each parity bit is the XOR of the input and the previous parity bit
  - Easy for RNN to solve the task
- Strategy
  - The output units tracks the current parity
  - The hidden units help compute the XOR
  - All hidden and output units are binary threshold units



RNN examples

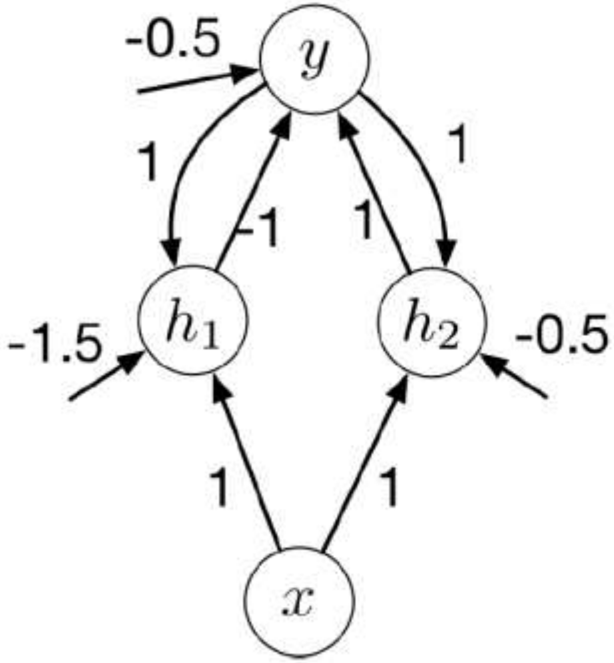
- Parity-check network
  - Unrolling in time



RNN examples

- Parity-check network
  - Use hidden units to compute XOR
  - Pick weights and biases as in the multilayer perceptrons

$y^{(t-1)}$	$x^{(t)}$	$h_1^{(t)}$	$h_2^{(t)}$	$y^{(t)}$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



Recurrent Neural Network

- General formulation

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

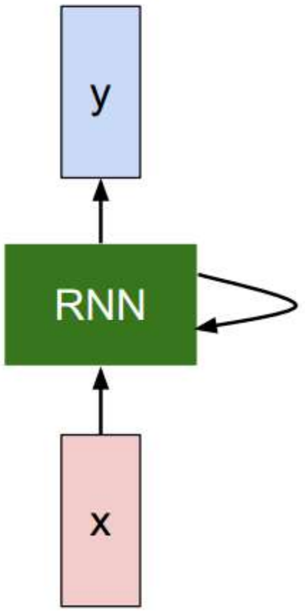
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W



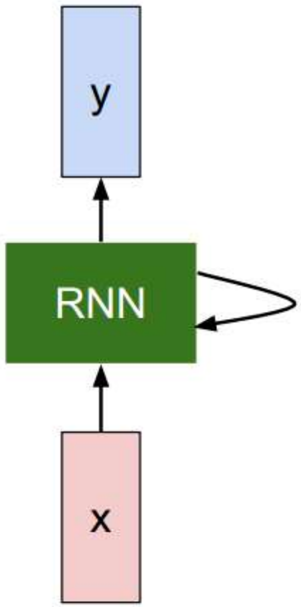
Recurrent Neural Network

- General formulation

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

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

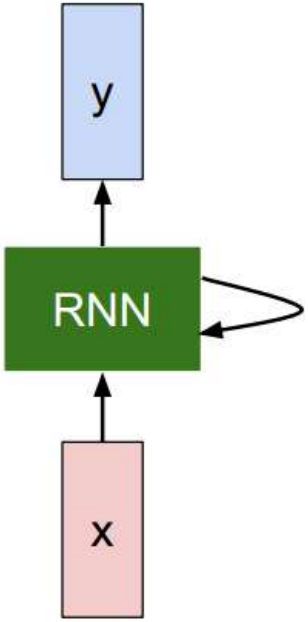
Notice: the same function and the same set of parameters are used at every time step.



(Vanilla)Recurrent Neural Network

- General formulation

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



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

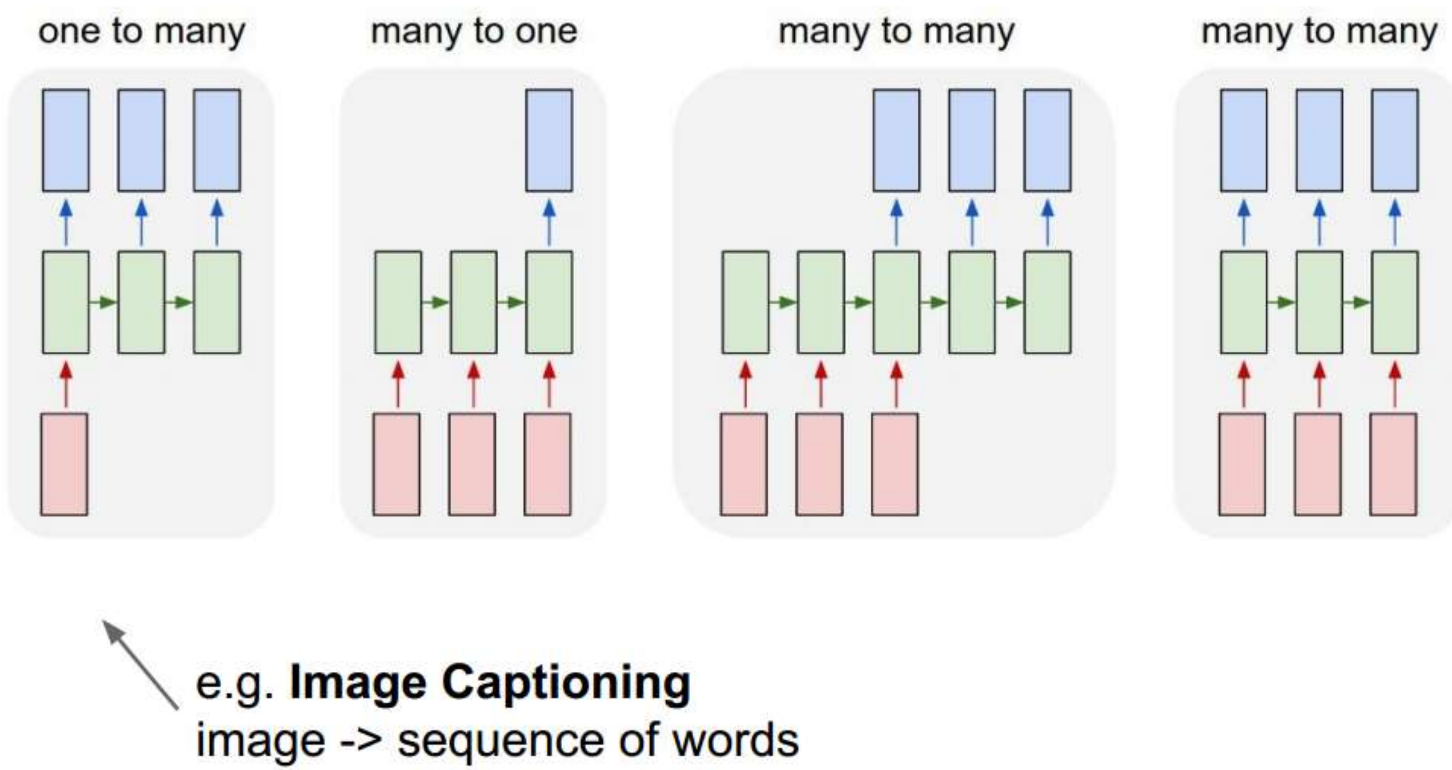
$$\downarrow$$

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

$$y_t = W_{hy}h_t$$

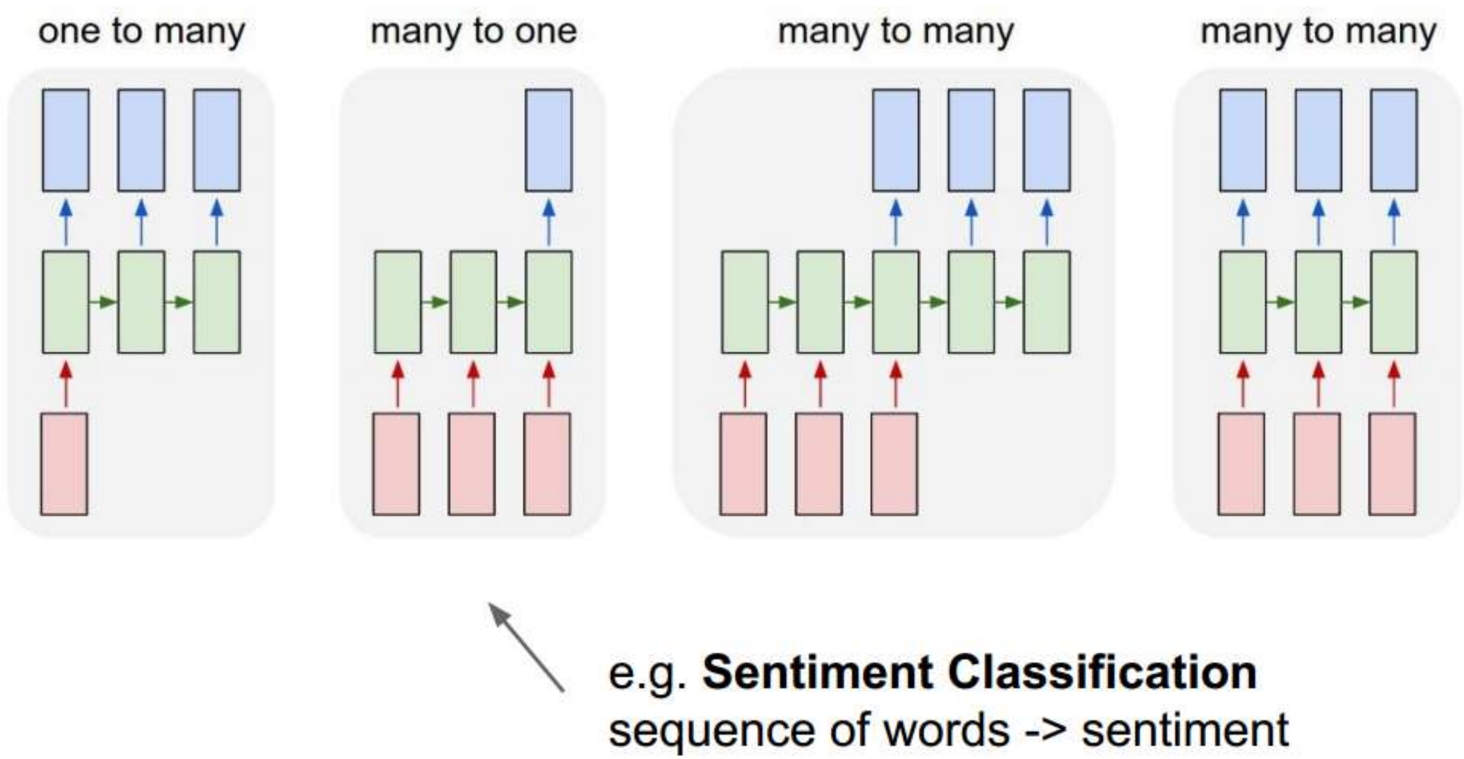
Recurrent Neural Network

- Recurrent Neural Networks: model variants



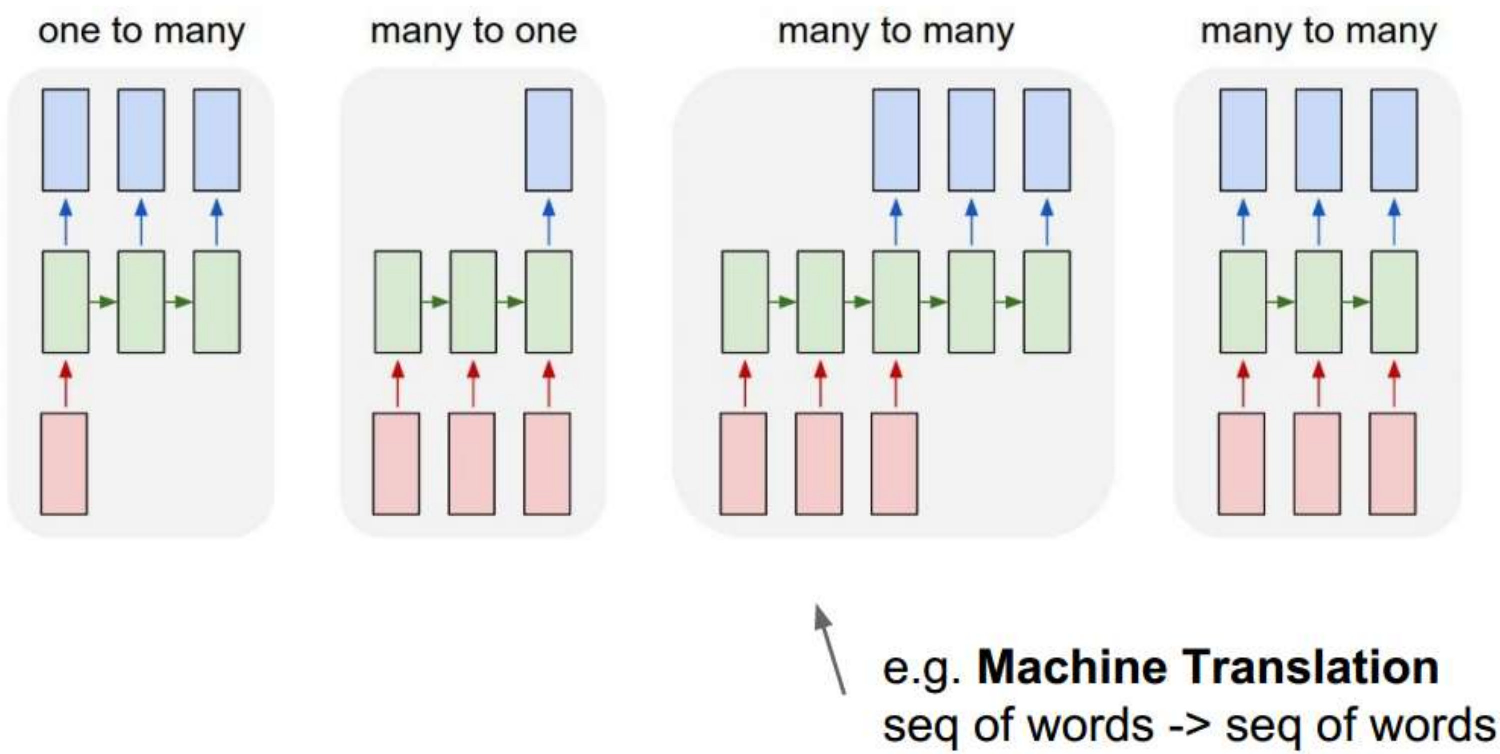
Recurrent Neural Network

- Recurrent Neural Networks: model variants



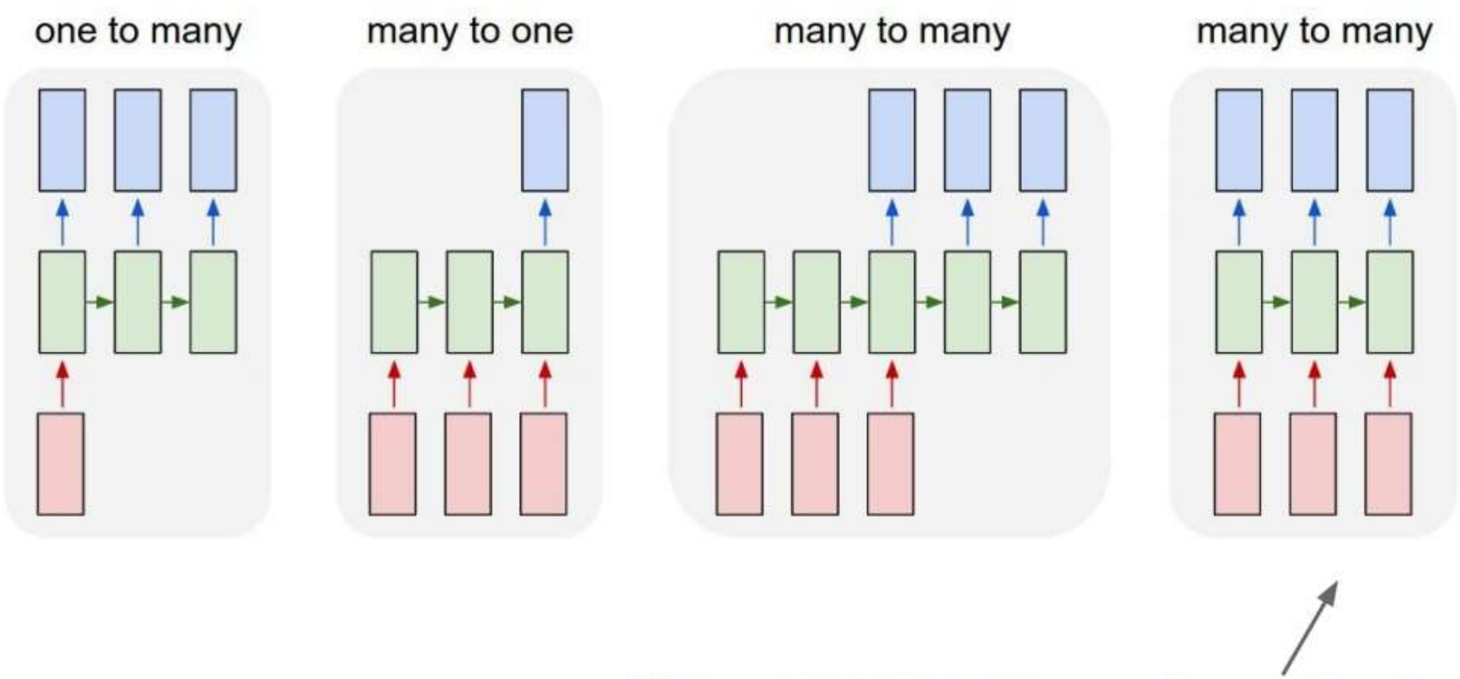
Recurrent Neural Network

- Recurrent Neural Networks: model variants



Recurrent Neural Network

- Recurrent Neural Networks: model variants



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

Recurrent Neural Network

- Sequential Processing of Non-Sequence Data

Classify images by taking a series of “glimpses”

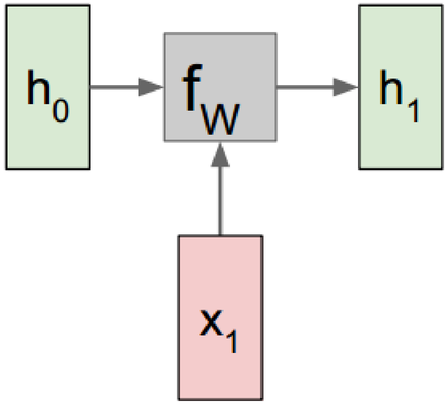
Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

Outline

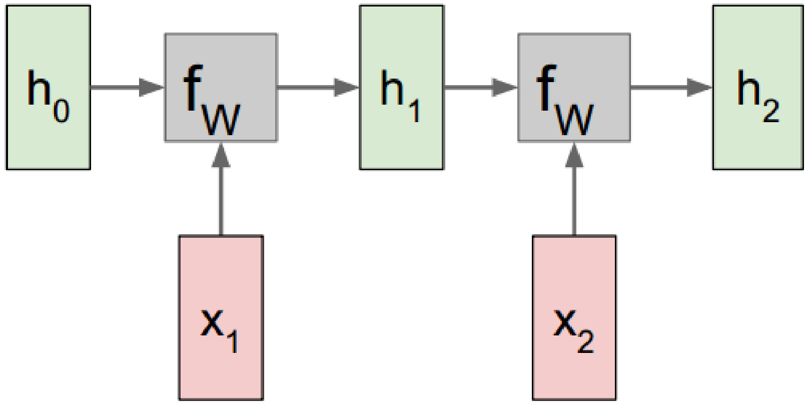
- Recurrent Neural Networks
  - Sequence modeling problem
  - Autoregressive models
  - (Vanilla) RNN models
- Backpropagation through time
  - Computational graph
- Example: language modeling
  - Neural language models

Acknowledgement: Feifei Li et al's cs231n notes

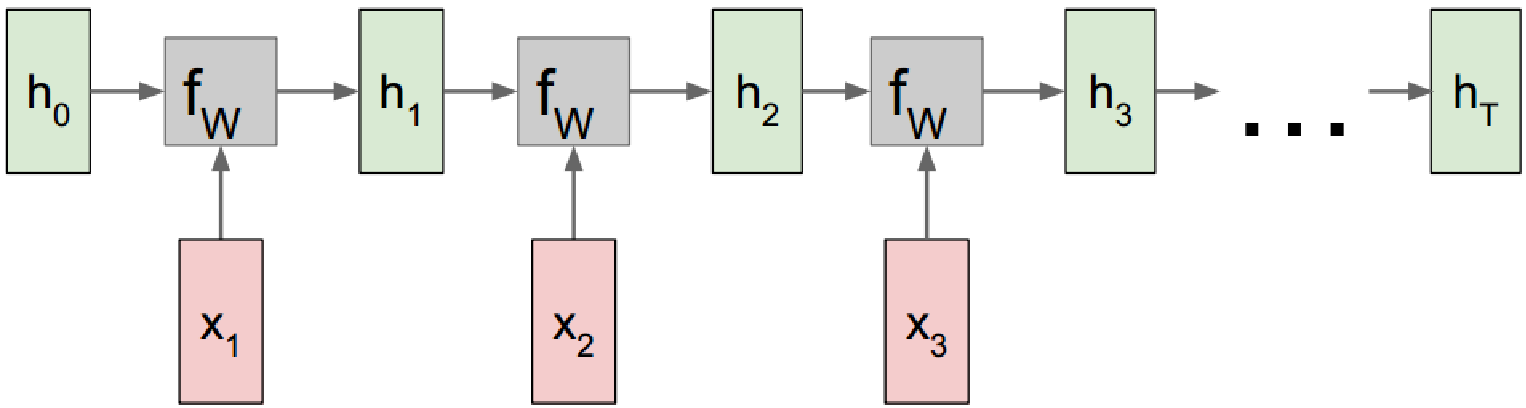
RNN: Computational Graph



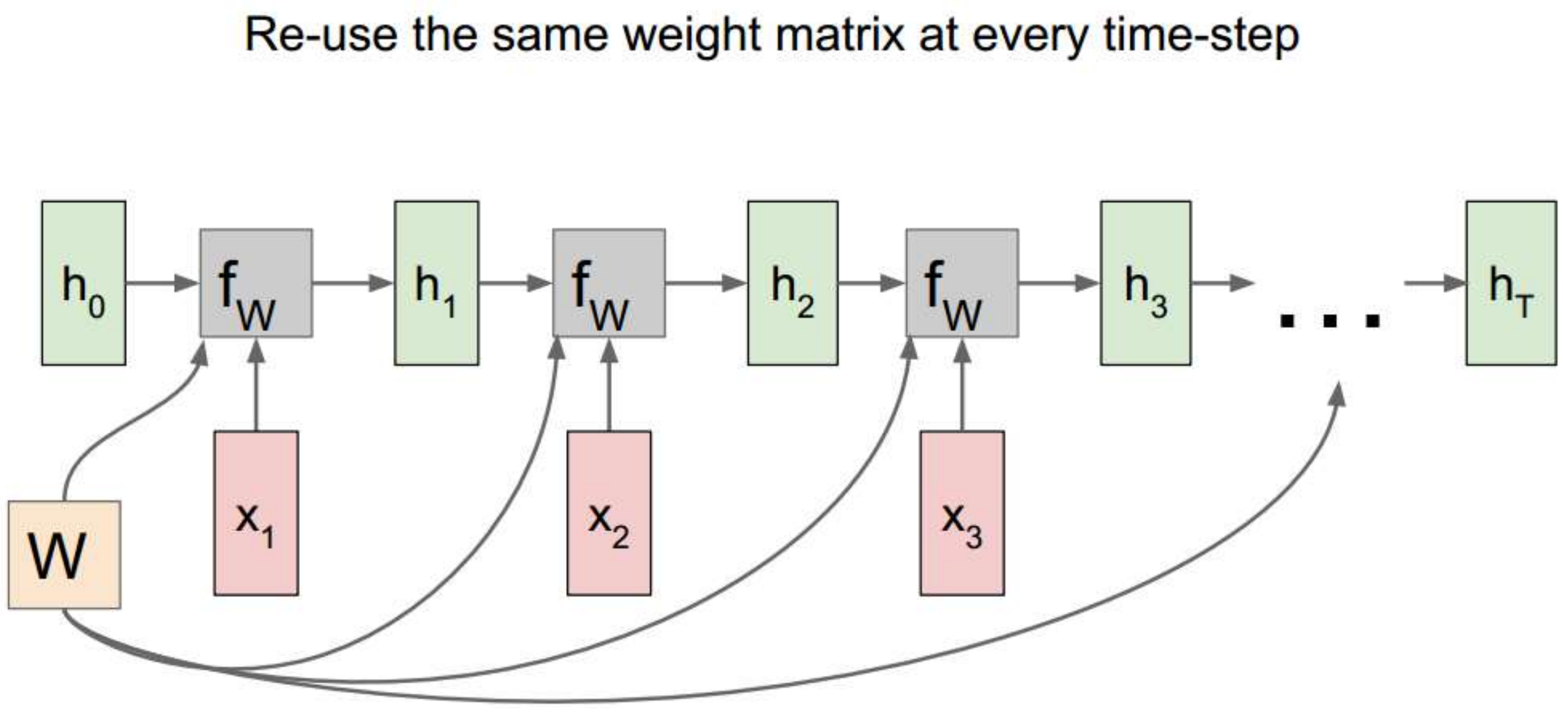
RNN: Computational Graph



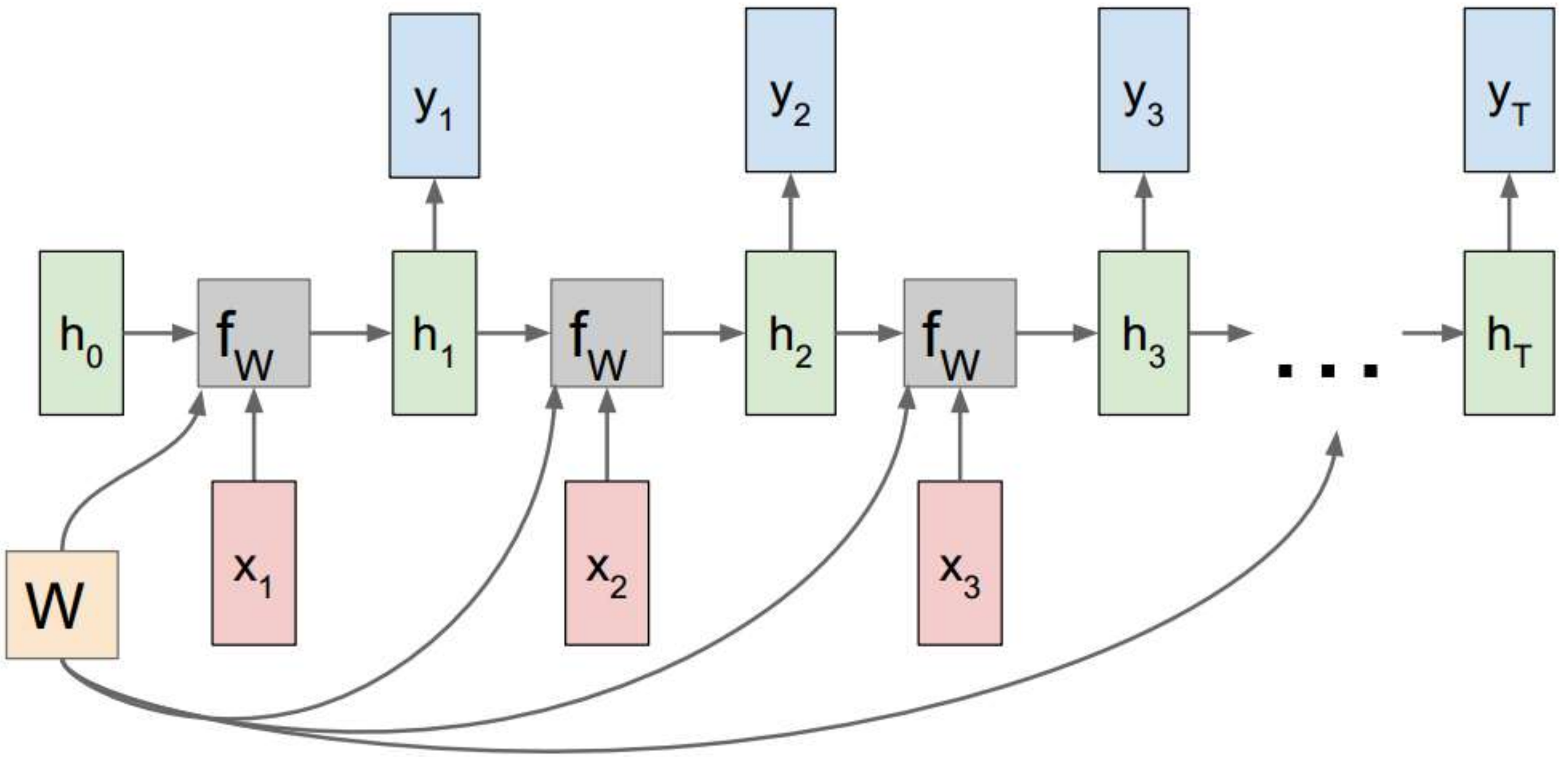
RNN: Computational Graph



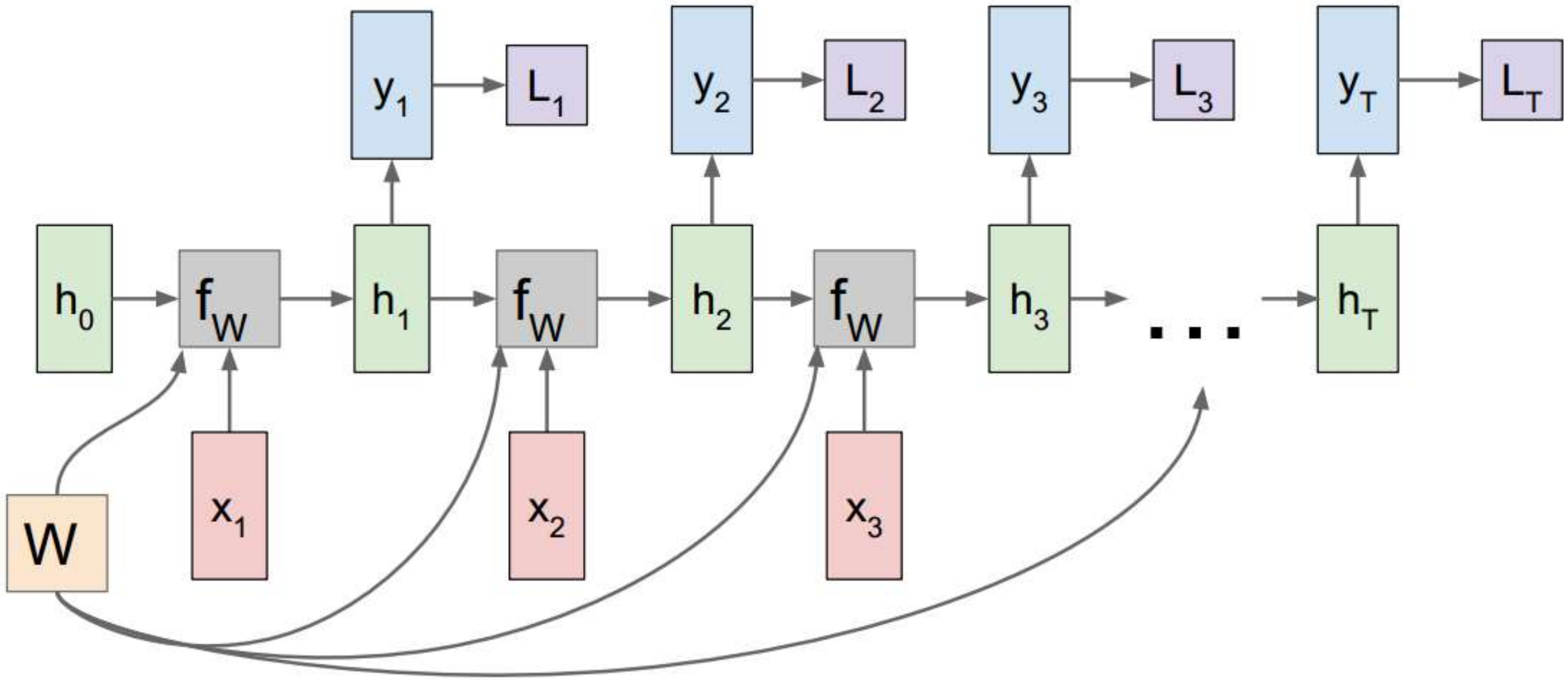
RNN: Computational Graph



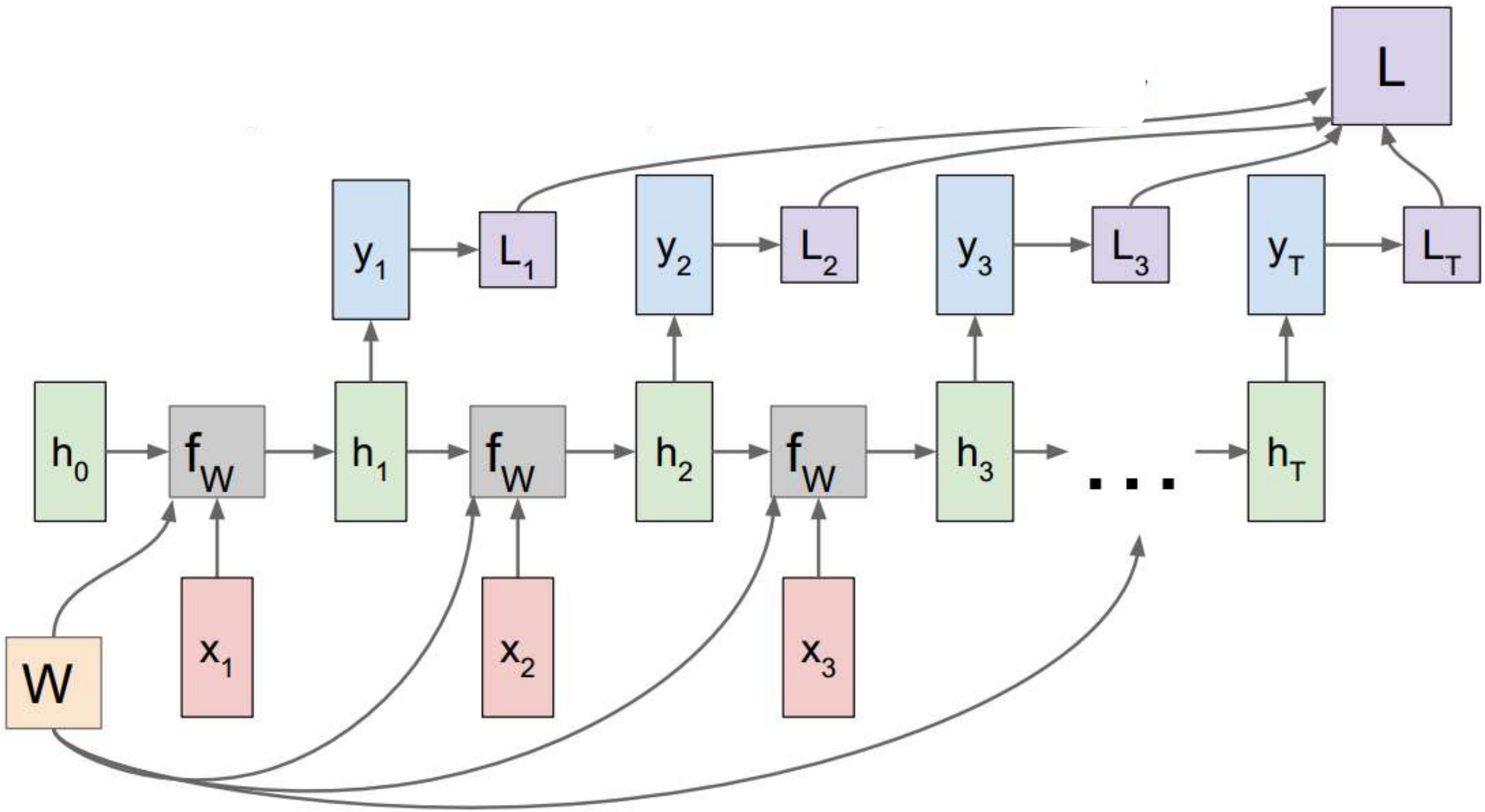
RNN: Computational Graph: Many to Many



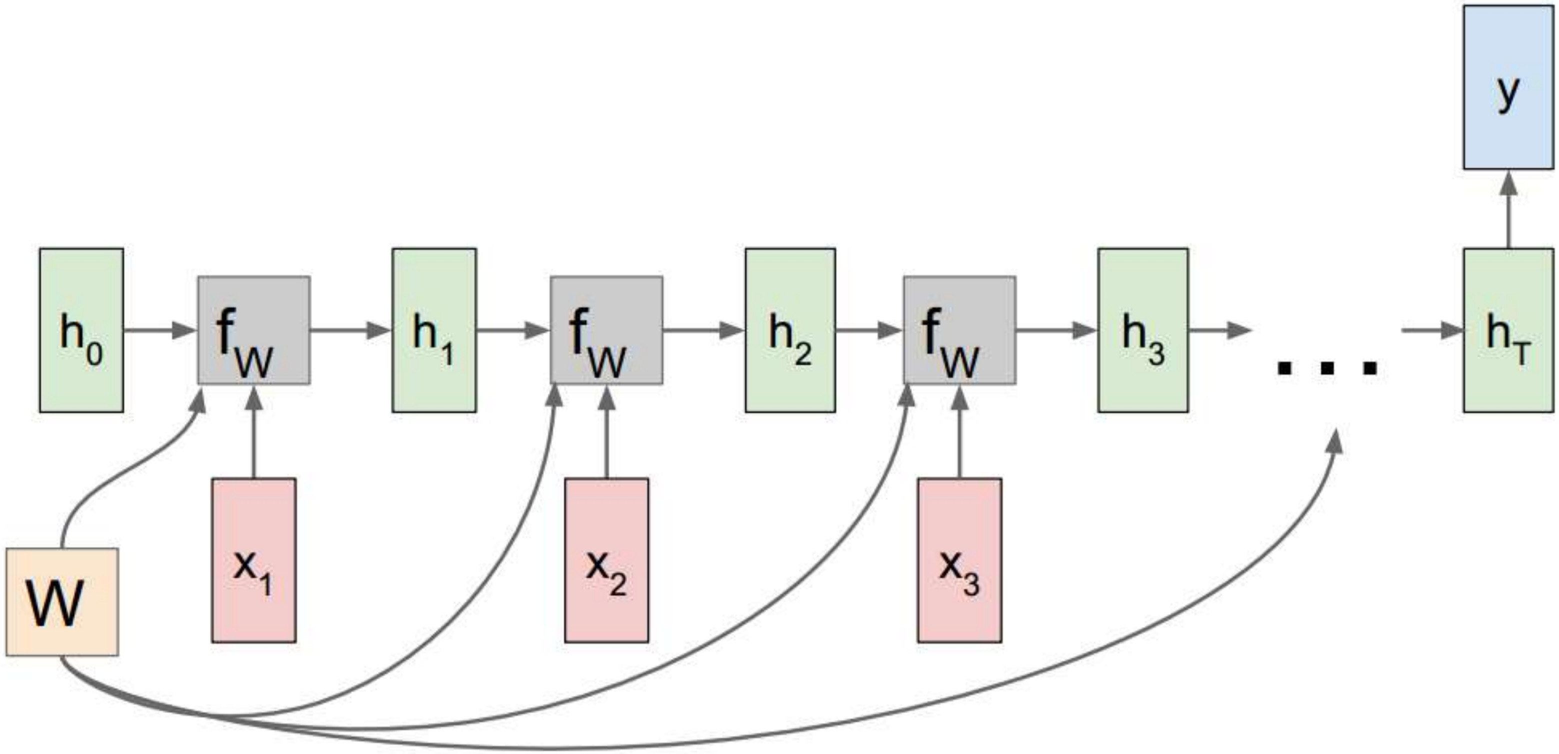
RNN: Computational Graph: Many to Many



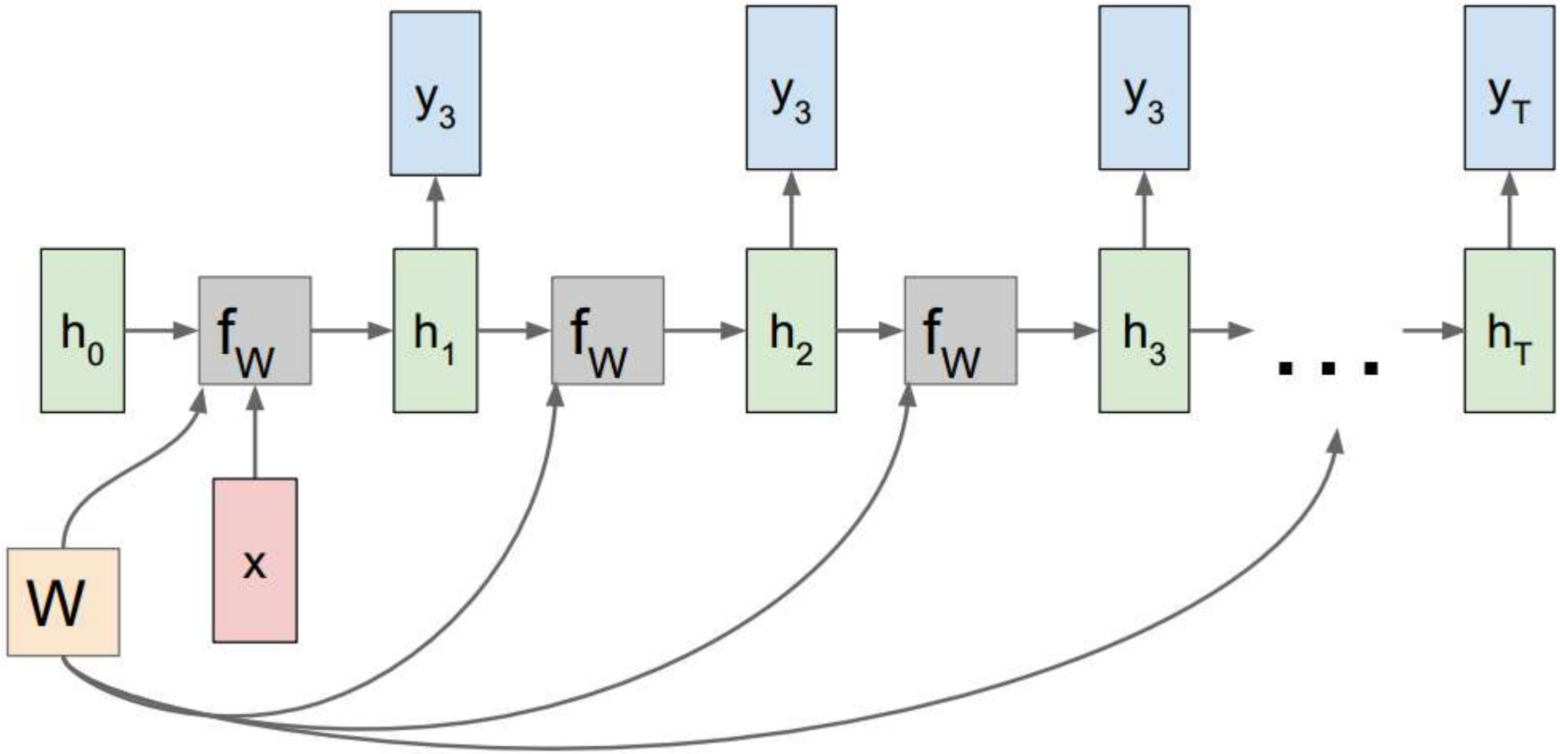
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One



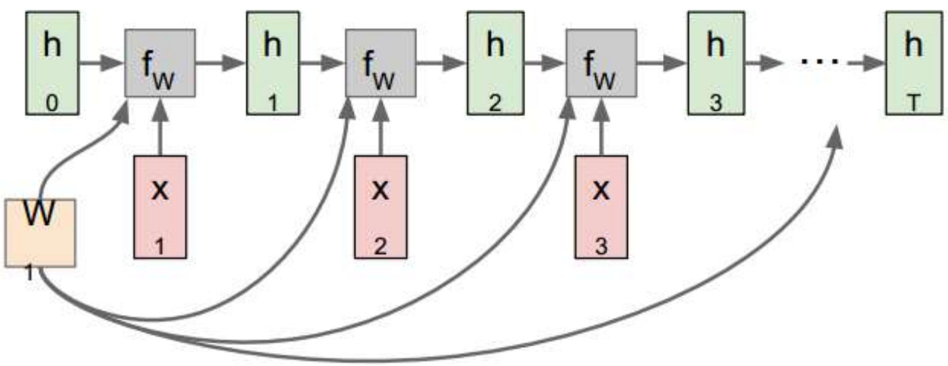
RNN: Computational Graph: One to Many



Sequence to Sequence

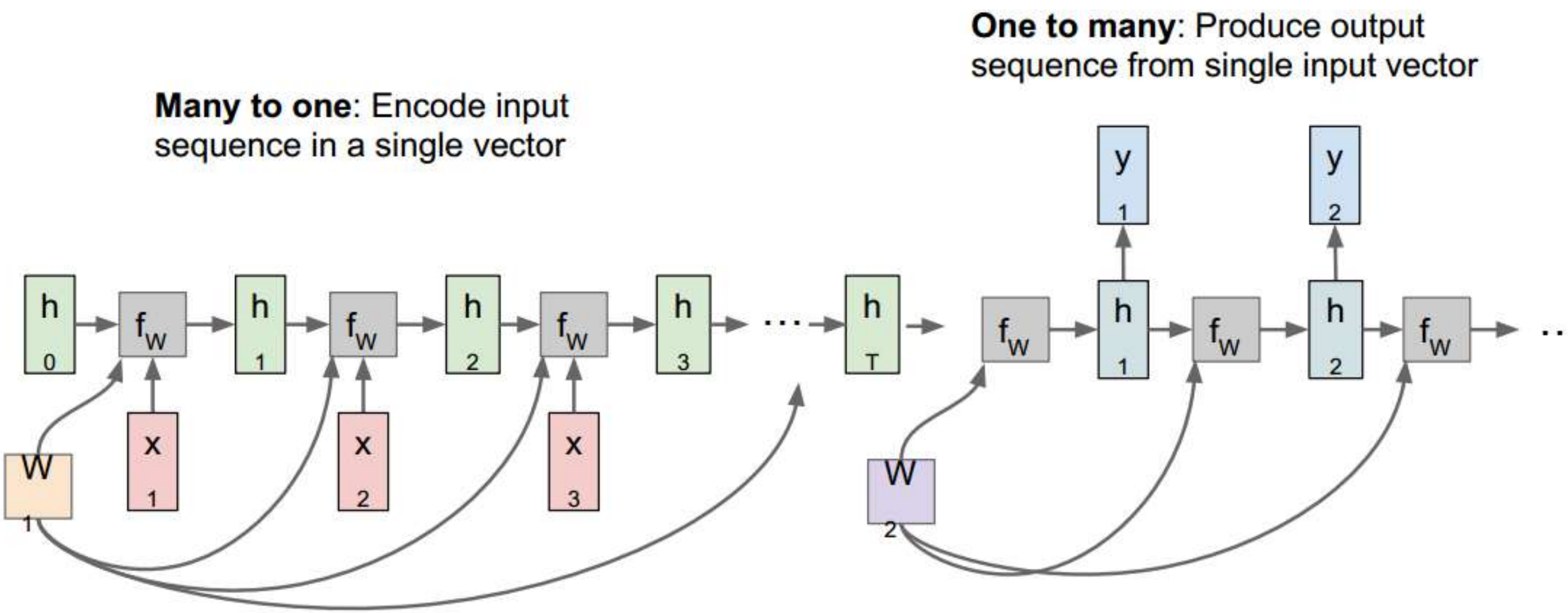
- Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector



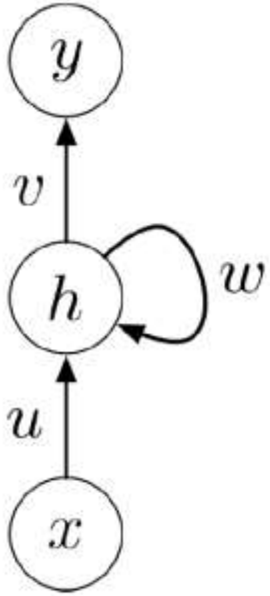
Sequence to Sequence

- Many-to-one + one-to-many



BPTT example

- A simple network
  - Everything is scalar



$$z^{(t)} = ux^{(t)} + wh^{(t-1)}$$

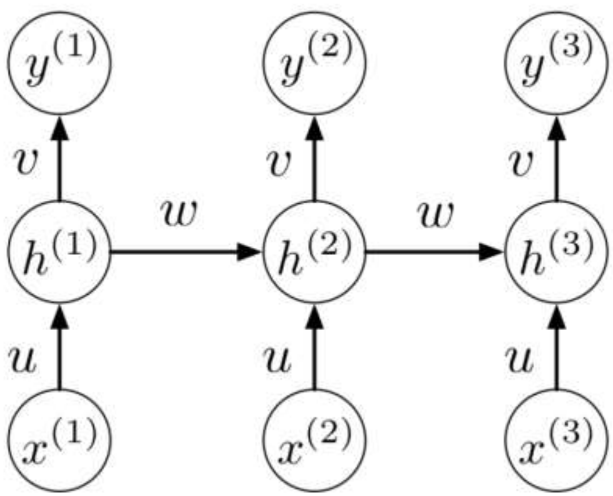
$$h^{(t)} = \phi(z^{(t)})$$

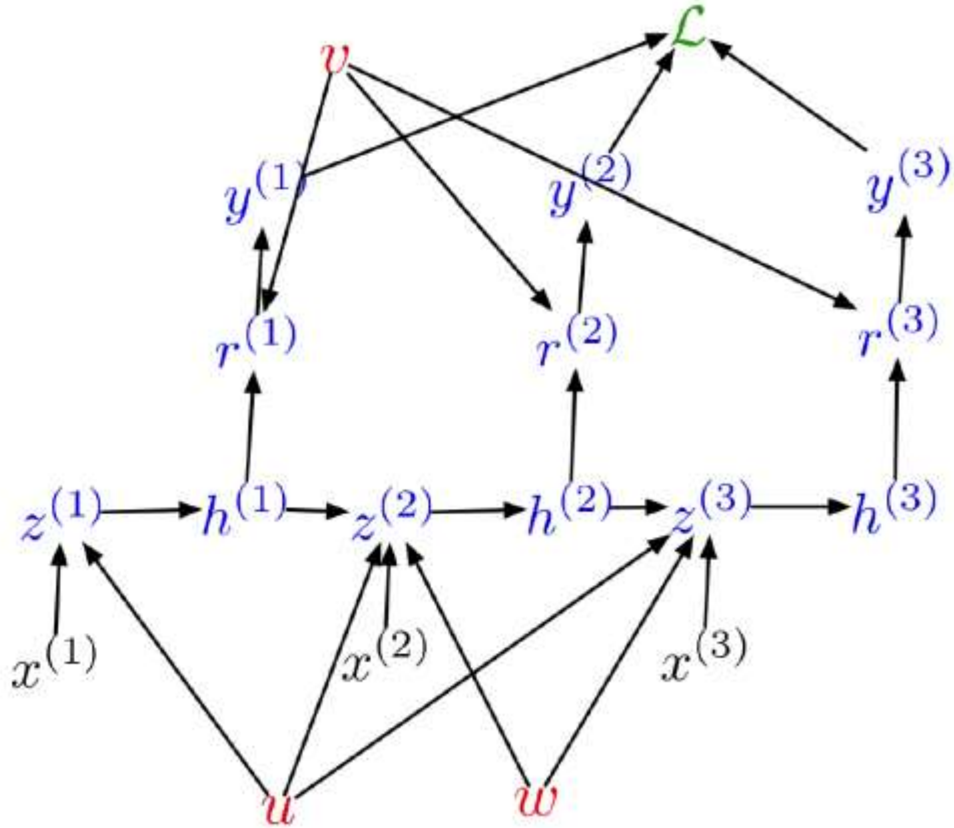
$$r^{(t)} = vh^{(t)}$$

$$y^{(t)} = \phi(r^{(t)}).$$

BPTT example

- A simple network
  - Everything is scalar
  - Unrolled computation graph with shared parameters





Recall: General Backpropagation

- Given a computation graph

Let $v_1, \ldots, v_N$ be a topological ordering of the computation graph (i.e. parents come before children.)

v_N denotes the variable we're trying to compute derivatives of (e.g. loss)

forward pass

For $i = 1, \ldots, N$

Compute v_i as a function of $\mathrm{Pa}(v_i)$

backward pass

$\delta_{v_N} = 1$

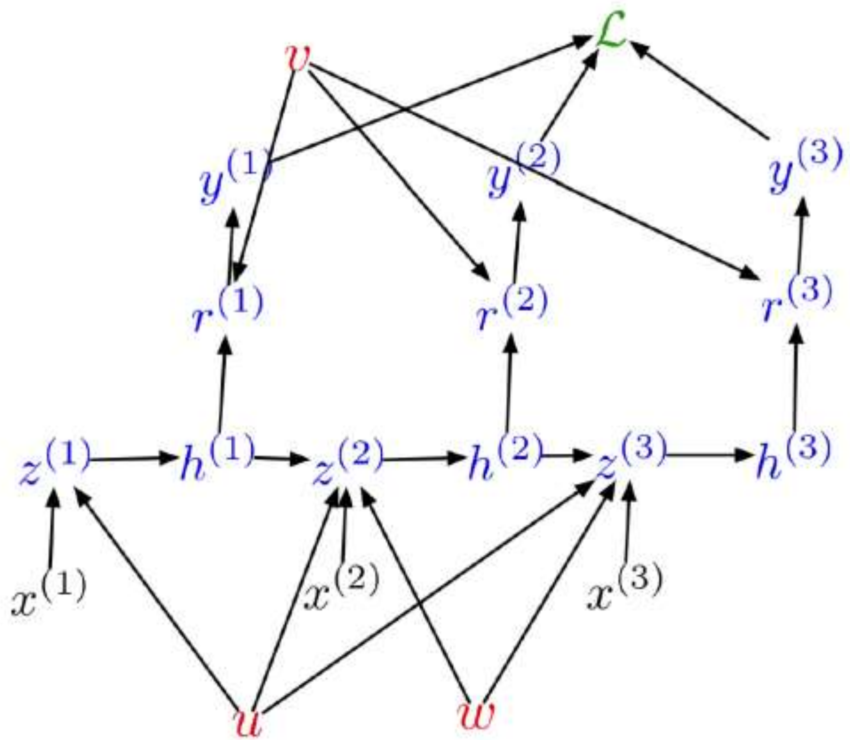
For $i = N-1, \ldots, 1$

$\delta_{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \delta_{v_j} \frac{\partial v_j}{\partial v_i}$

BPTT example

$$z^{(t)} = ux^{(t)} + wh^{(t-1)}$$
$$h^{(t)} = \phi(z^{(t)})$$
$$r^{(t)} = vh^{(t)}$$
$$y^{(t)} = \phi(r^{(t)}).$$

- A simple network
  - Unrolled computation graph with shared parameters



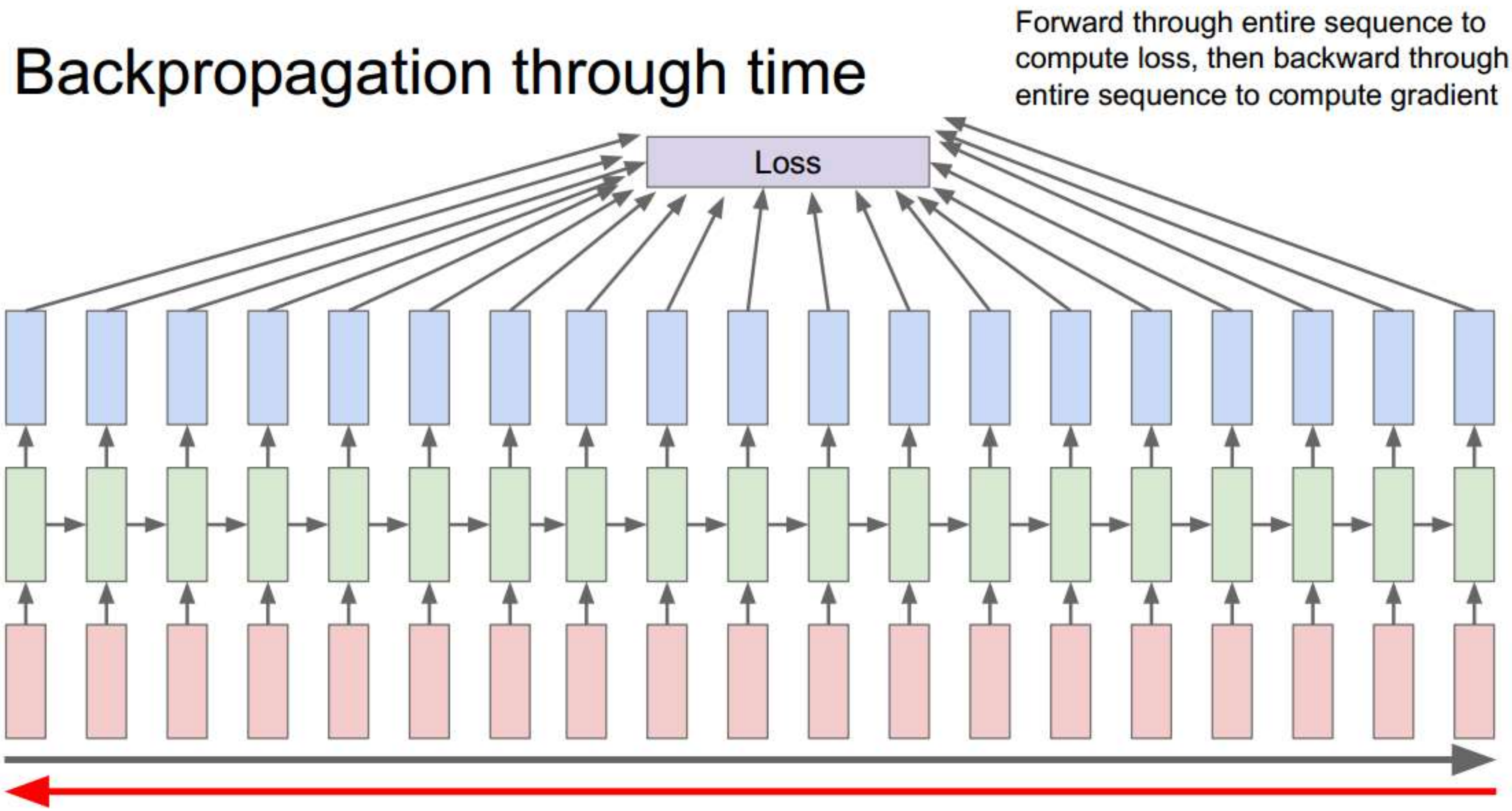
Activations:

$$\overline{\mathcal{L}} = 1$$
$$\overline{y^{(t)}} = \overline{\mathcal{L}}\,\frac{\partial \mathcal{L}}{\partial y^{(t)}}$$
$$\overline{r^{(t)}} = \overline{y^{(t)}}\,\phi'(r^{(t)})$$
$$\overline{h^{(t)}} = \overline{r^{(t)}}\,v + \overline{z^{(t+1)}}\,w$$
$$\overline{z^{(t)}} = \overline{h^{(t)}}\,\phi'(z^{(t)})$$

Parameters:

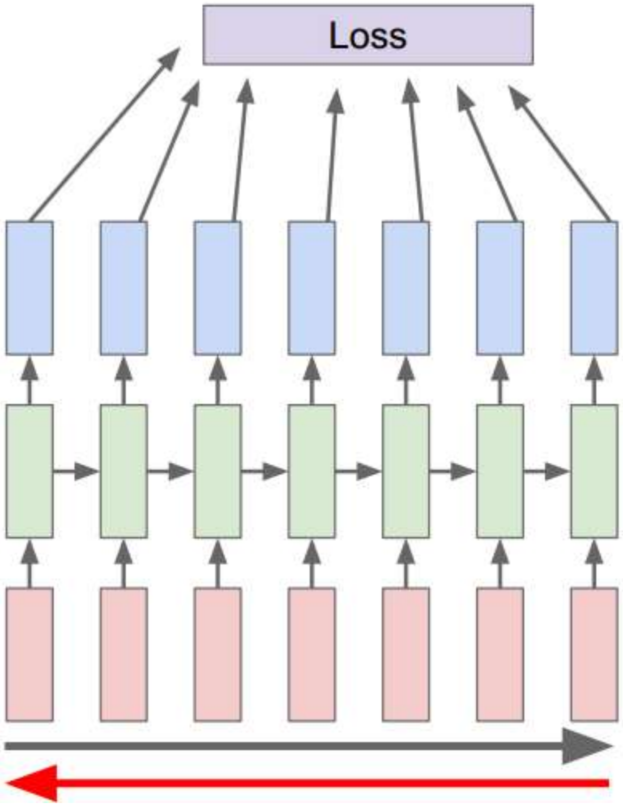
$$\overline{u} = \sum_t \overline{z^{(t)}}\,x^{(t)}$$
$$\overline{v} = \sum_t \overline{r^{(t)}}\,h^{(t)}$$
$$\overline{w} = \sum_t \overline{z^{(t+1)}}\,h^{(t)}$$

Recurrent Neural Network



Recurrent Neural Network

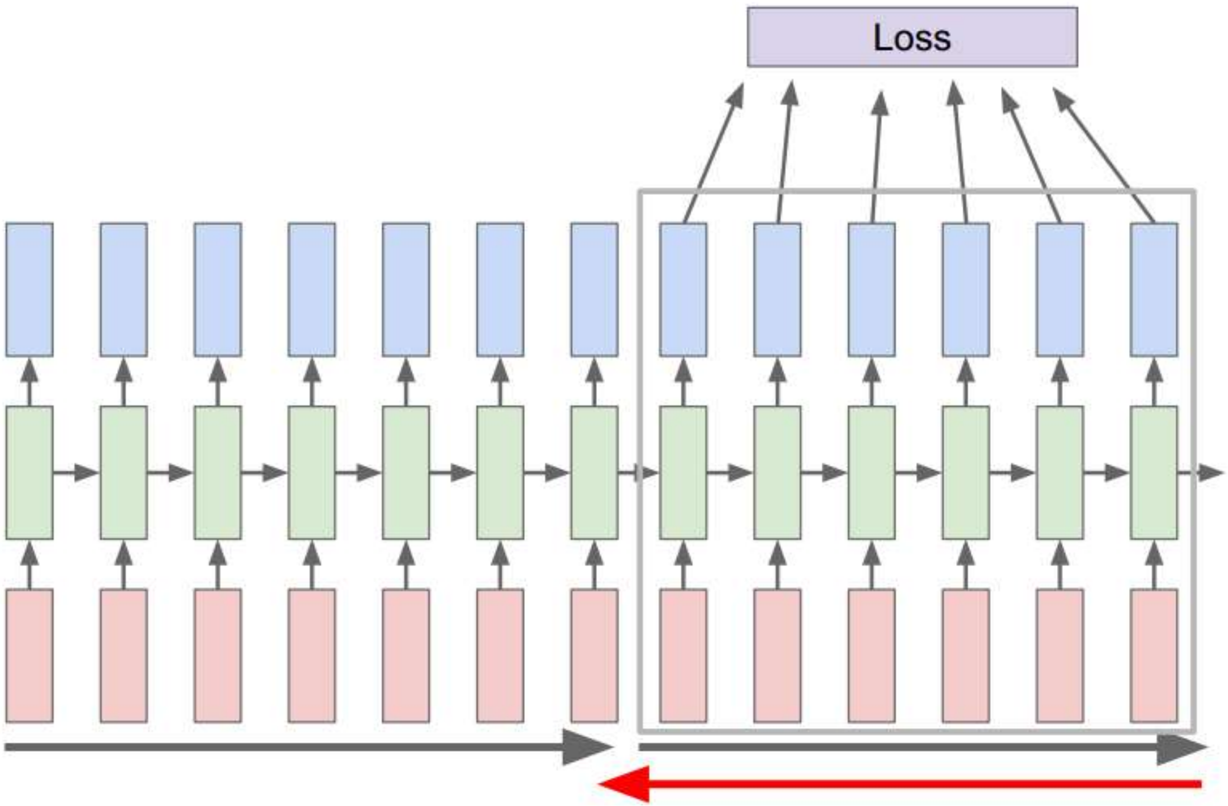
Truncated Backpropagation through time



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

Recurrent Neural Network

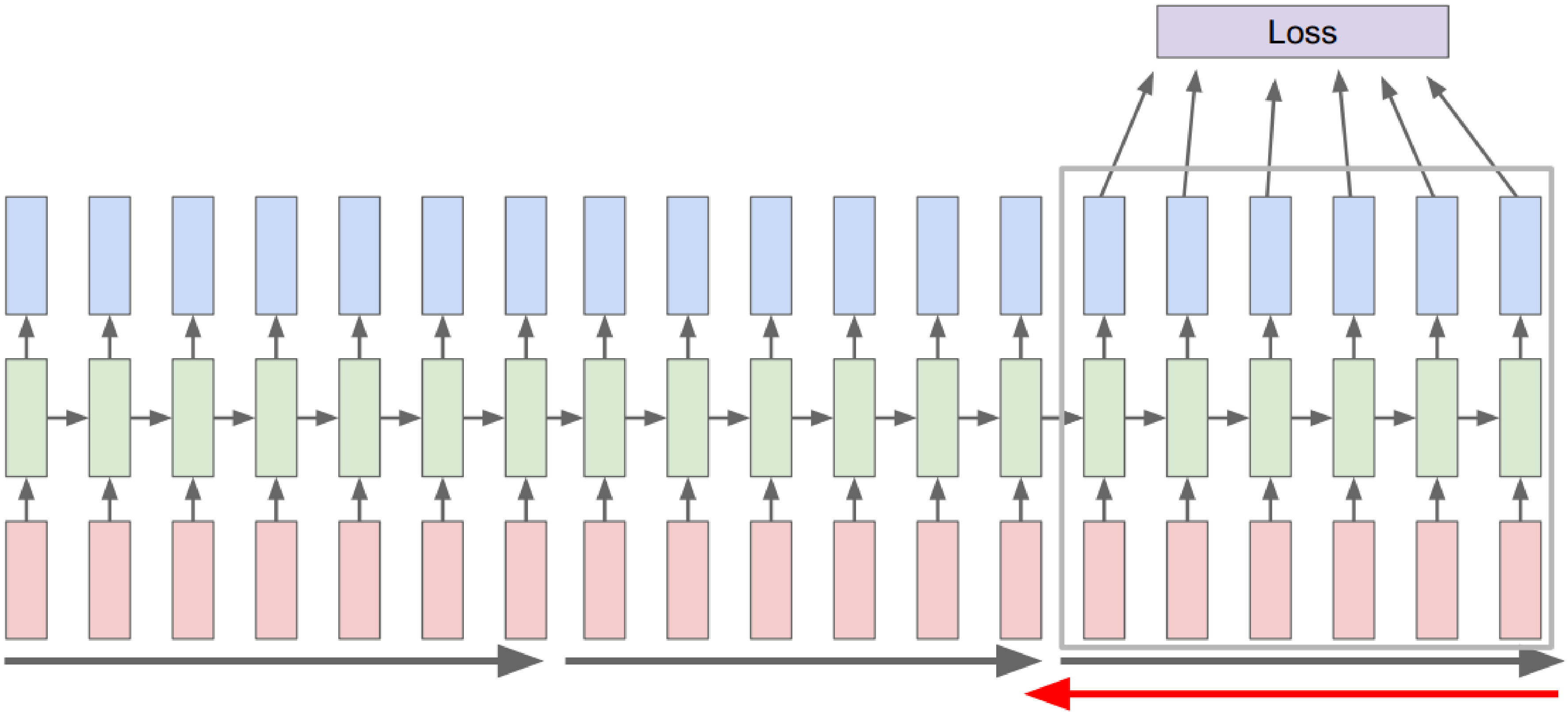
Truncated Backpropagation through time



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

Recurrent Neural Network

Truncated Backpropagation through time



Outline

- Recurrent Neural Networks
  - Sequence modeling problem
  - Autoregressive models
  - (Vanilla) RNN models
- Backpropagation through time
  - Computational graph
- Example: language modeling
  - Neural language models

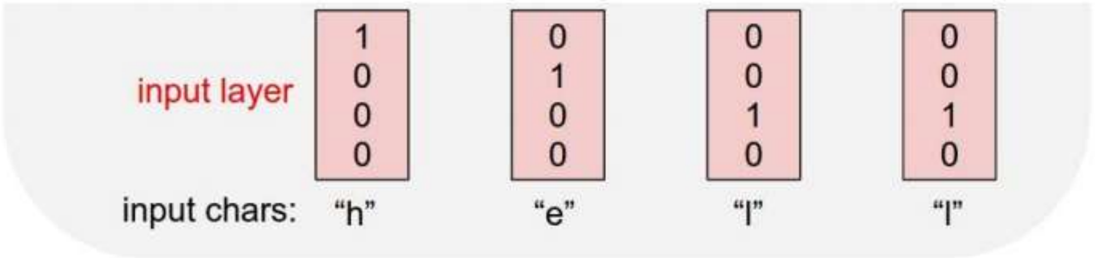
Acknowledgement: Feifei Li et al's cs231n notes

RNN for language modeling

Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



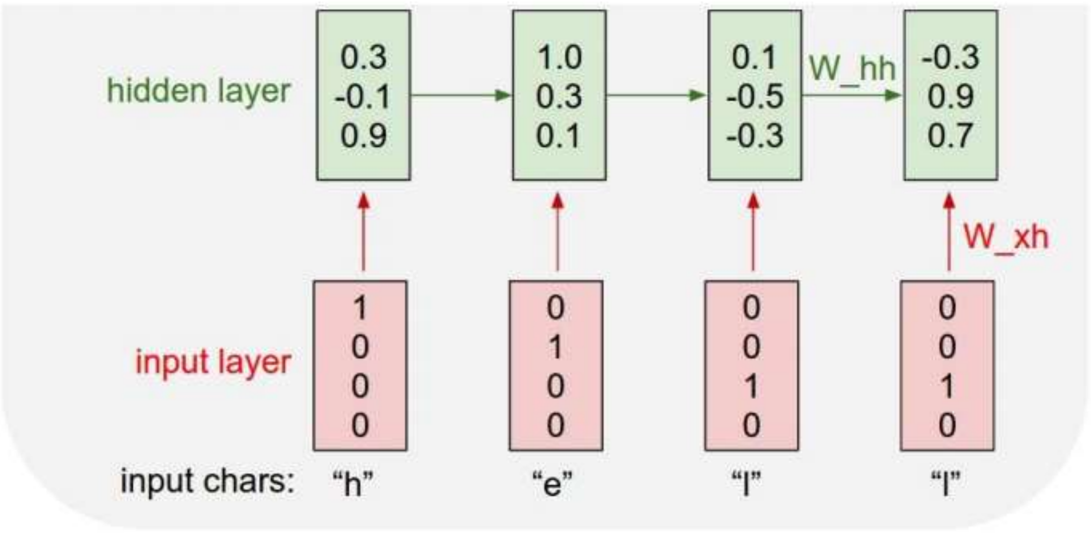
RNN for language modeling

Example:
Character-level
Language Model

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

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

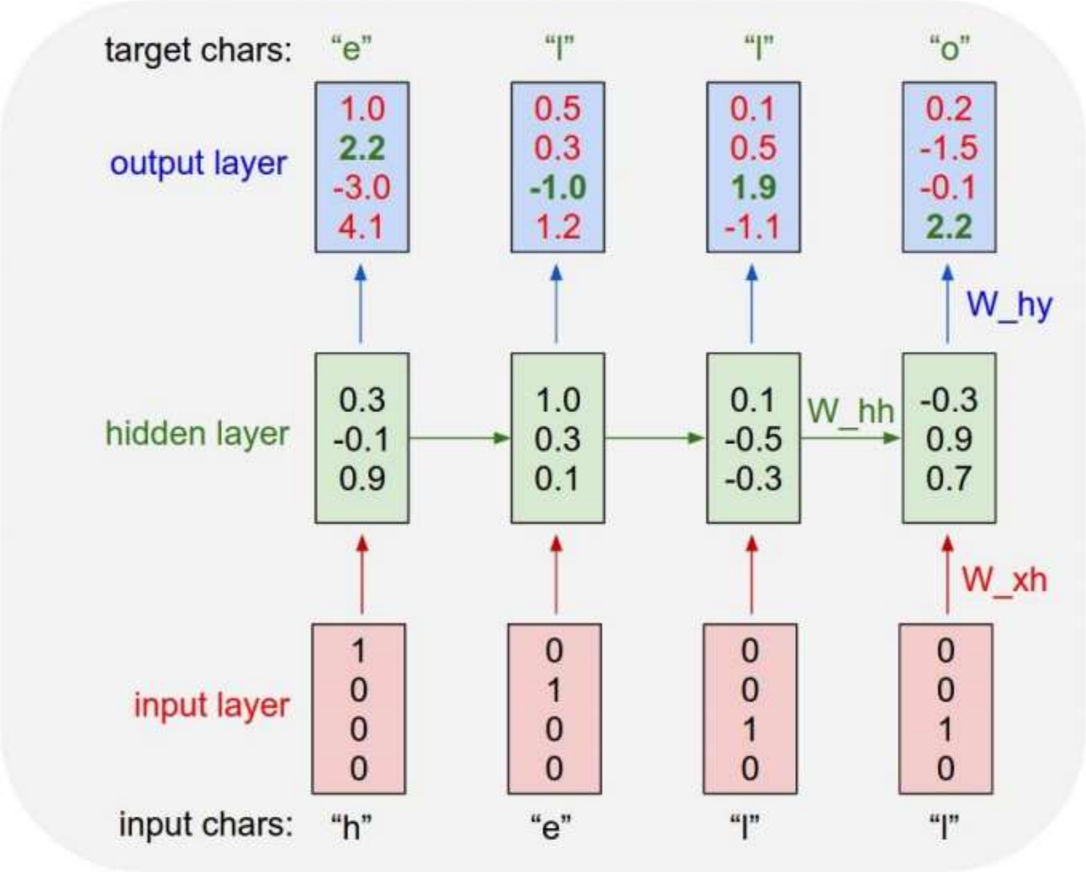


RNN for language modeling

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

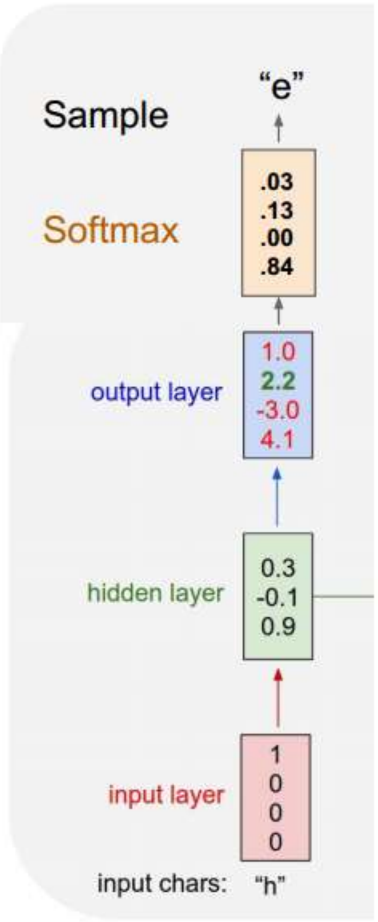


RNN for language modeling

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

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

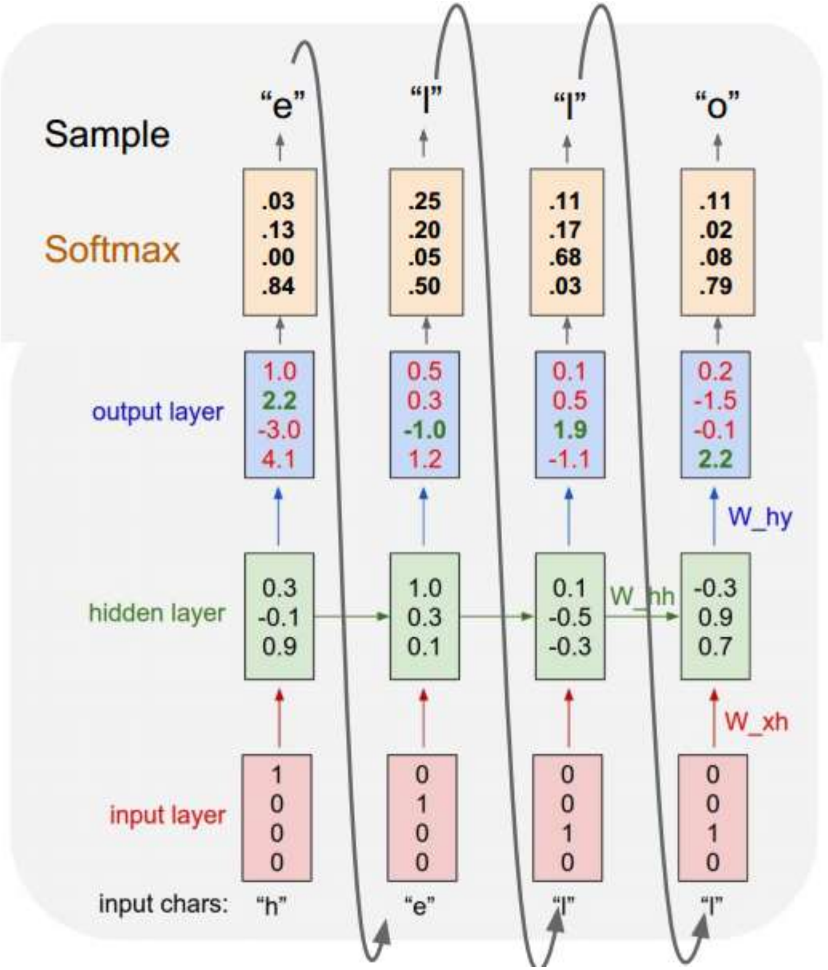


RNN for language modeling

Example:
Character-level Language Model Sampling

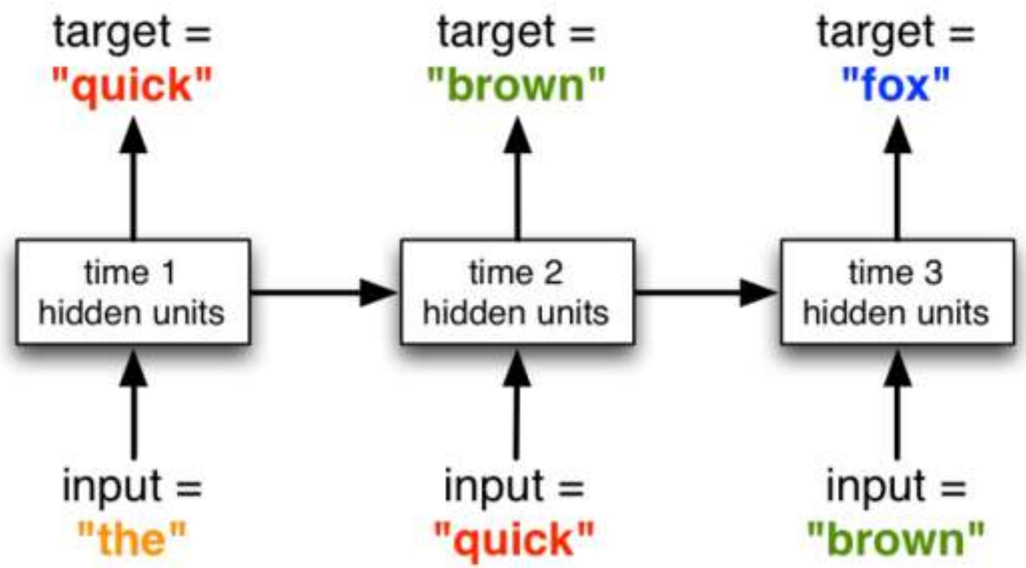
Vocabulary:
[h,e,l,o]

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



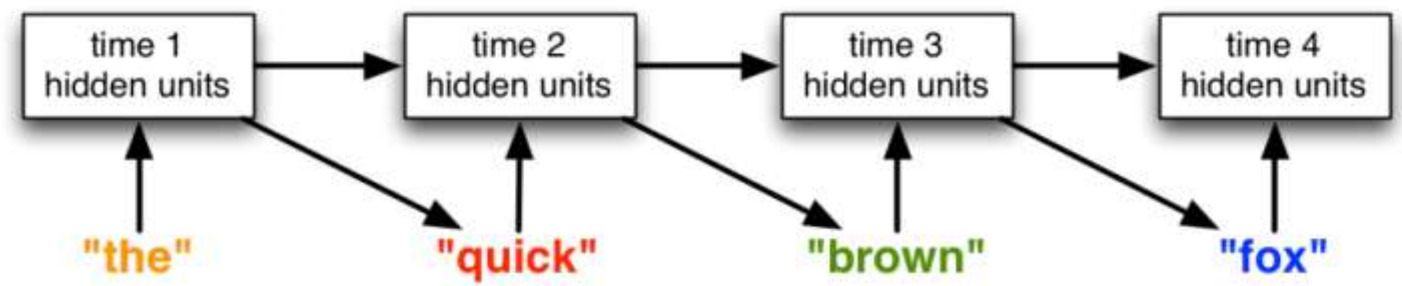
RNN for language modeling

- Modeling at word level
  - Each word is represented as an indicator vector
  - The model predicts a distribution over words



RNN for language modeling

- Generating from a RNN language model
  - The outputs are fed back to the network



- Training time: the inputs are the token from the training set (teacher forcing).

RNN for language modeling

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

RNN for language modeling

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

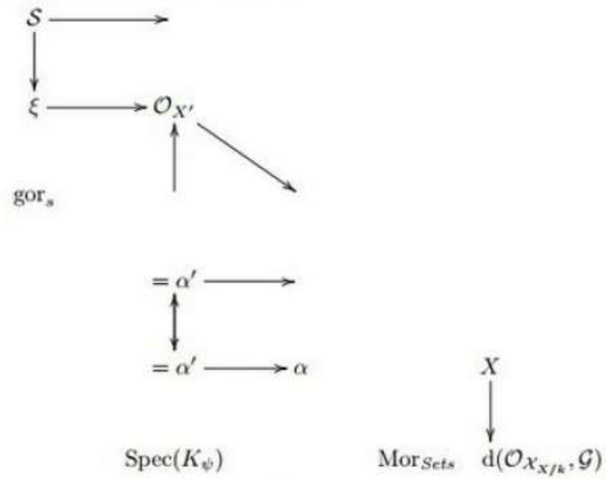
(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

- Generated math from algebraic geometry textbook

RNN for language modeling

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

$\square$

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. $\square$

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad -1(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. $\square$

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

- Generated math from algebraic geometry textbook

RNN for language modeling

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

- Generated C code

RNN for language modeling

```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

- Generated C code

RNN for language modeling

- Some remaining challenges
  - Vocabularies can be very large once you include people, places, etc. It's computationally difficult to predict distributions over millions of words.
  - How do we deal with words we haven't seen before?
  - In some languages (e.g. Chinese), it's hard to determine what should be considered a word.

Summary

- RNN
  - RNN for sequence modeling
  - RNNs allow a lot of flexibility in architecture design
  - Language modeling in NLP

- Next time:
  - LSTM, GRU

- Reading materials:
  - http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/readings/L05%20Distributed%20Representations.pdf
  - http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/readings/L13%20Recurrent%20Neural%20Nets.pdf