COL333/671: Introduction to AI

Semester I, 2024-25

Learning – III: Structured Neural Network

Rohan Paul

Outline

- Last Class
  - Basics of Neural Networks
- This Class
  - Structuring Neural Networks
- Reference Material
  - Please follow the notes as the primary reference on this topic. Additional reading from AIMA book Ch. 18 (18.2, 18.6 and 18.7) and DL book Ch 6 sections 6.1 – 6.5 (except 6.4).

Acknowledgement

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Parag, Emma Brunskill, Alexander Amini, Dan Klein, Anca Dragan, Nicholas Roy and others.

Standard Feed Forward Networks

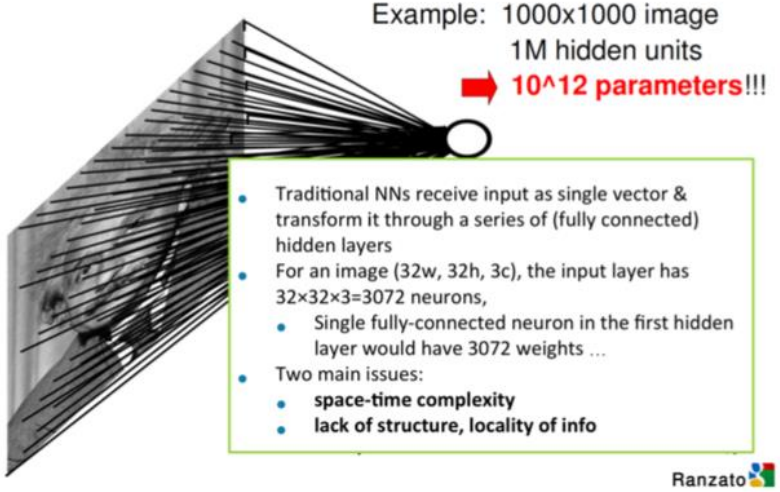
- We already know about standard feedforward networks
  - A deep sequence of fully connected perceptron networks
  - Data as input and a task dependent output. Training using a task-dependent loss.
  - They are a general class of neural network models
- Usually there is structure in data
  - Taking advantage of this structure in the neural network can make learning better and more efficient.
- Architectures is suited to data types/tasks
  - Images -> CNNs
  - Time series -> RNNs
  - Unsupervised -> auto encoders.
  - Many more …..

Just as algorithms are iteratively designed for a problem, machine learning engineers design an architecture for a data type and a task! Designing a network is a bit of an art!

Image data possesses structure

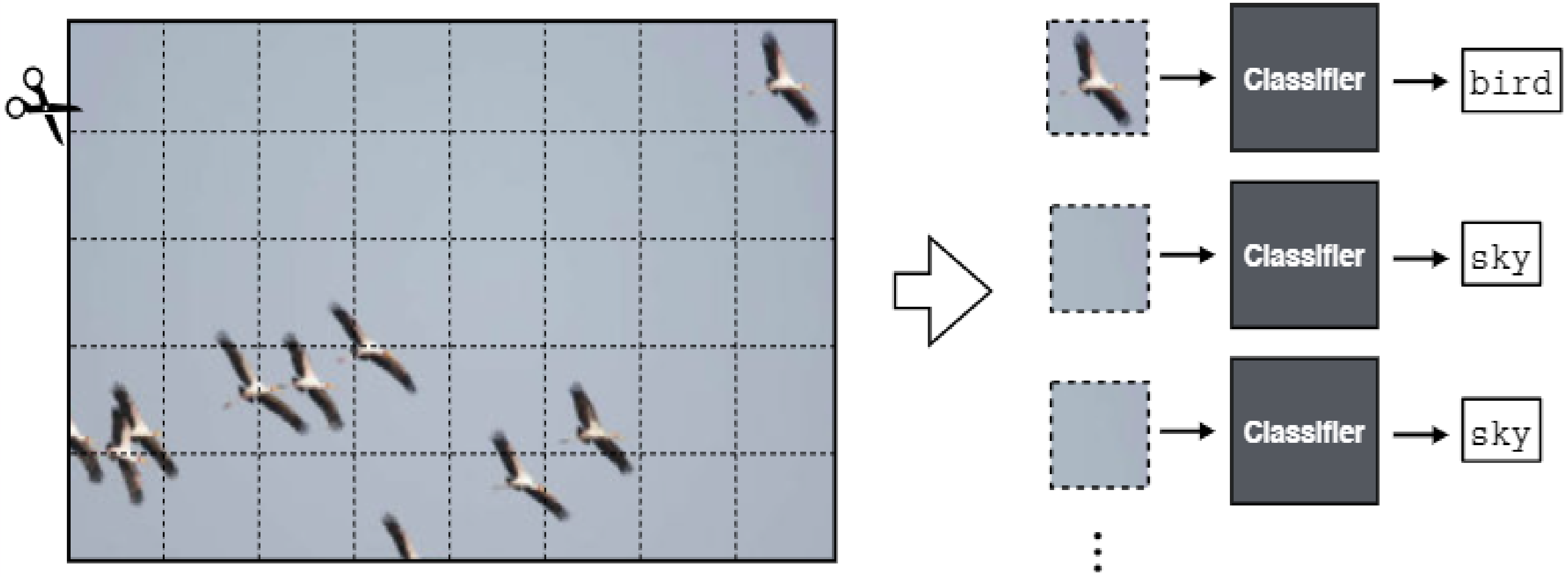
- Adjacency matters
- Spatial Invariance exists
- We can process images using standard fully connected networks say for classification etc.
- Is a more efficient architecture possible taking advantage of spatial invariance?

Does it matter where the bird is for classifying this image as bird or no-bird?



A fully connected network for image classification will have "lots" of parameters.

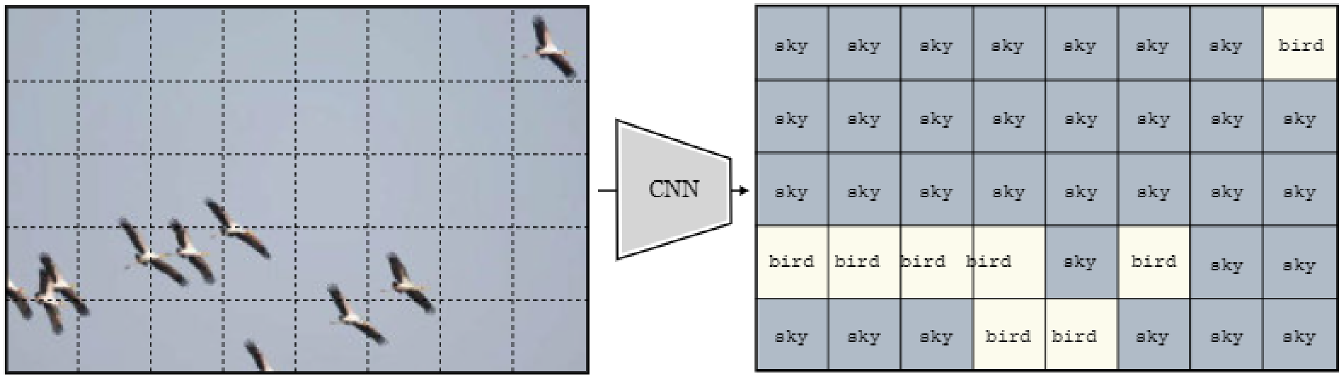
Convolutional neural nets – exploiting spatial structure

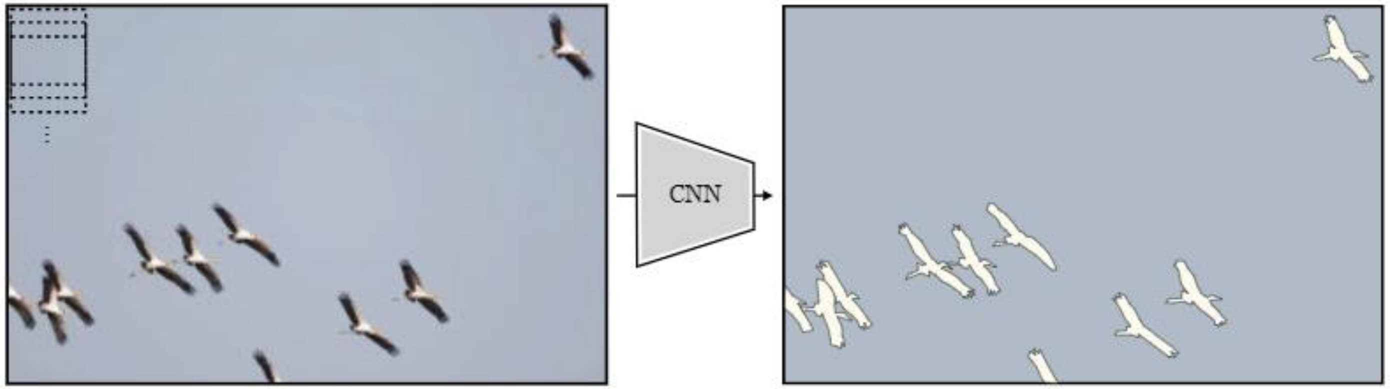


Key idea: divide the image into little patches, and then process each patch independently and identically.

Convolutional neural nets – exploiting spatial structure

Note: patches can be overlapping too. That way the input and output are of the same size. This achieves a task called semantic segmentation.





Convolution Layers

$$\mathrm{x}_{\mathrm{out}} = \mathrm{w} \star \mathrm{x}_{\mathrm{in}} + b \qquad / \quad \texttt{conv}$$

where w is the kernel and b is the bias; $\theta = [\mathrm{w}, b]$ are the parameters of this layer.

$$x_{\mathrm{out}}[n,m] = b + \sum_{k_1,k_2=-K}^{K} w[k_1,k_2]\, x_{\mathrm{in}}[n+k_1, m+k_2] \qquad / \quad \texttt{conv} \quad \text{(expanded)}$$

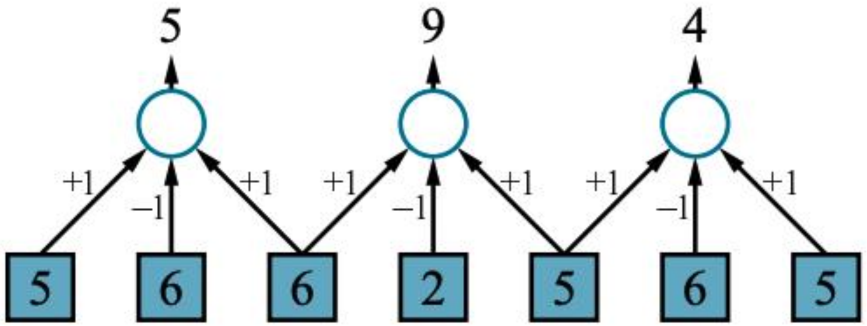
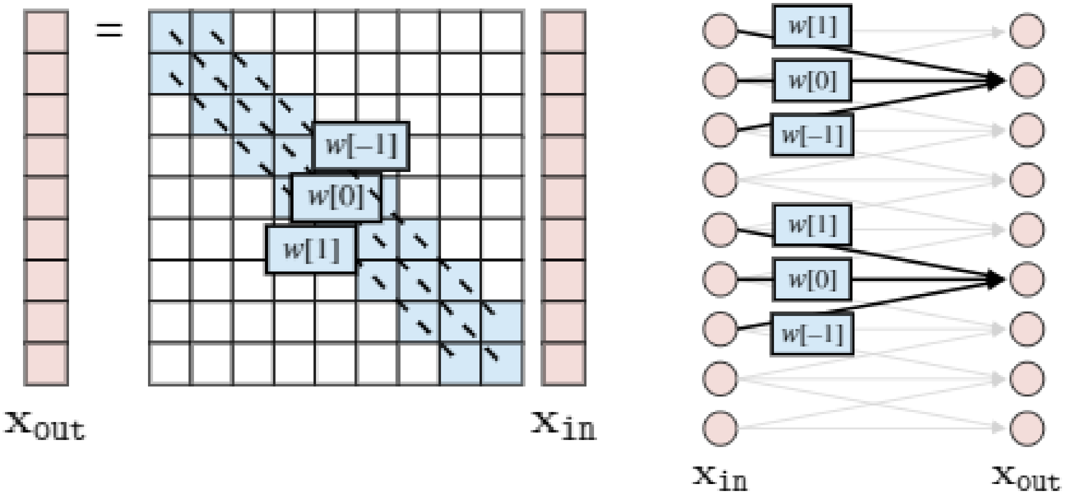


Figure 21.4 An example of a one-dimensional convolution operation with a kernel of size $l=3$ and a stride $s=2$. The peak response is centered on the darker (lower intensity) input pixel. The results would usually be fed through a nonlinear activation function (not shown) before going to the next hidden layer.

Equivalent views



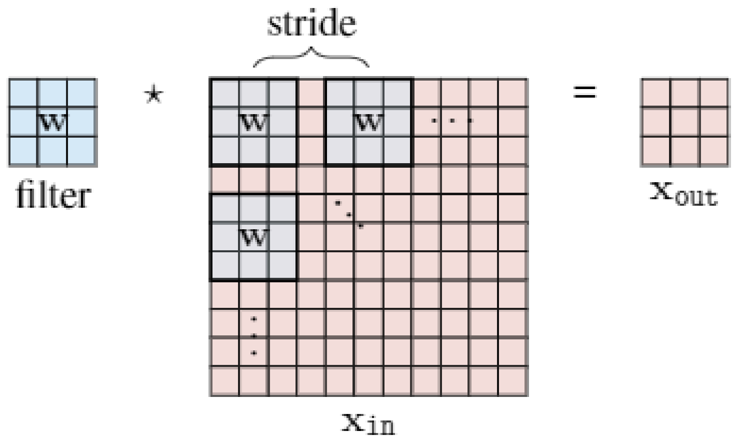
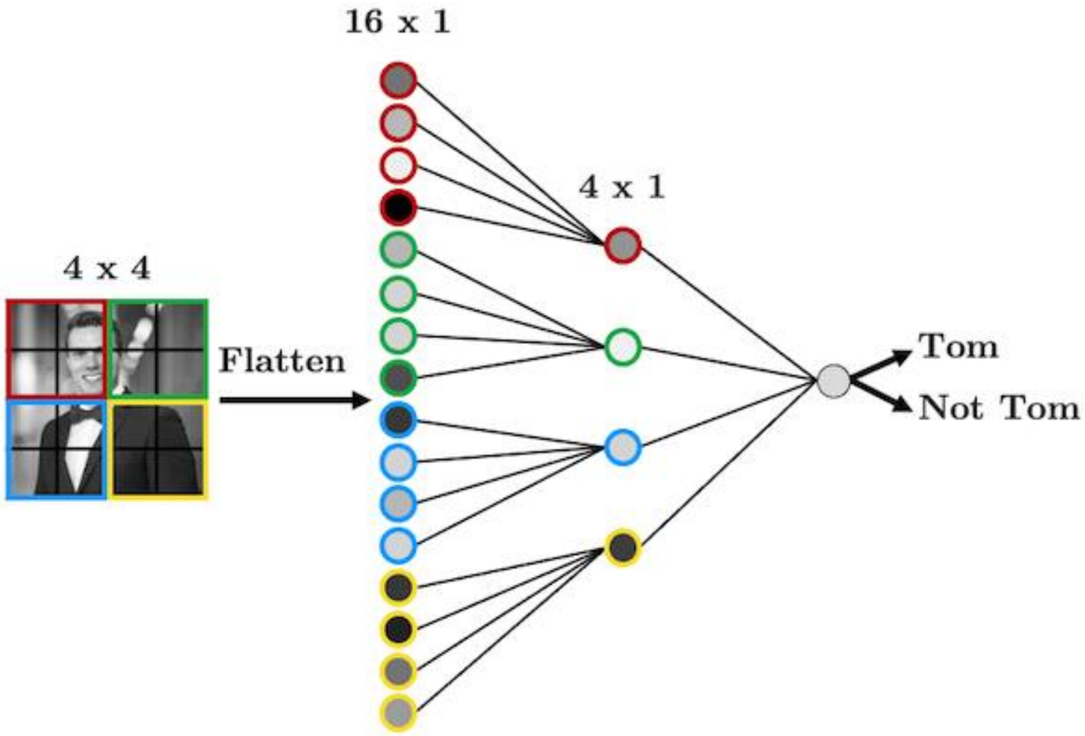
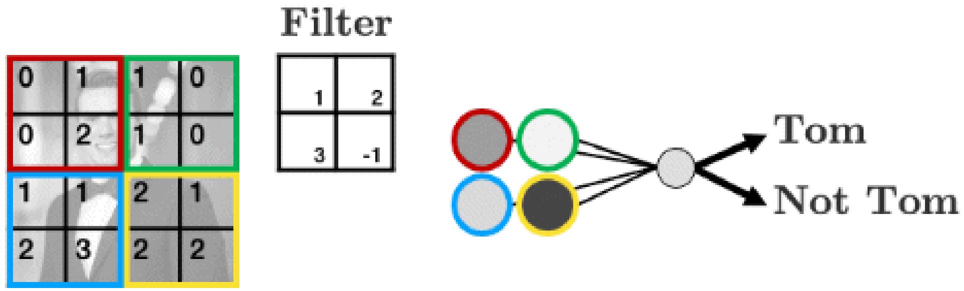
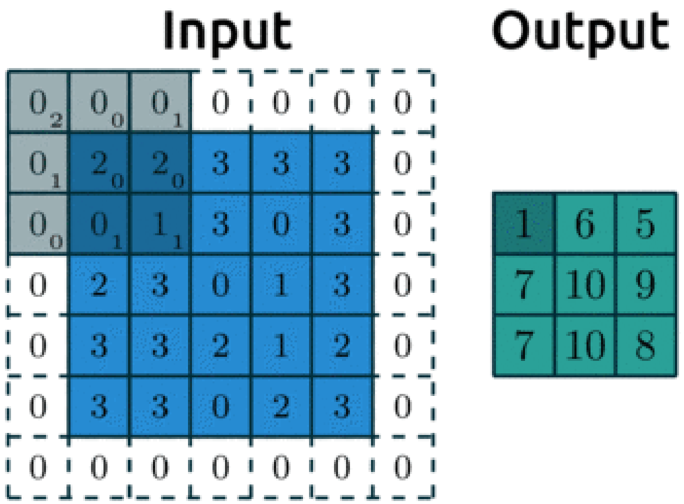


Figure: AIMA Fourth Edition (Chapter: Deep Learning)

Convolution Layers



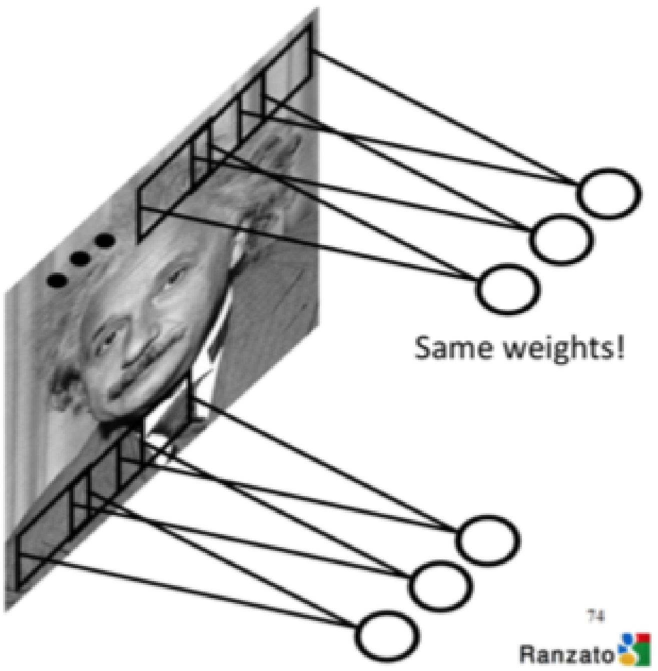




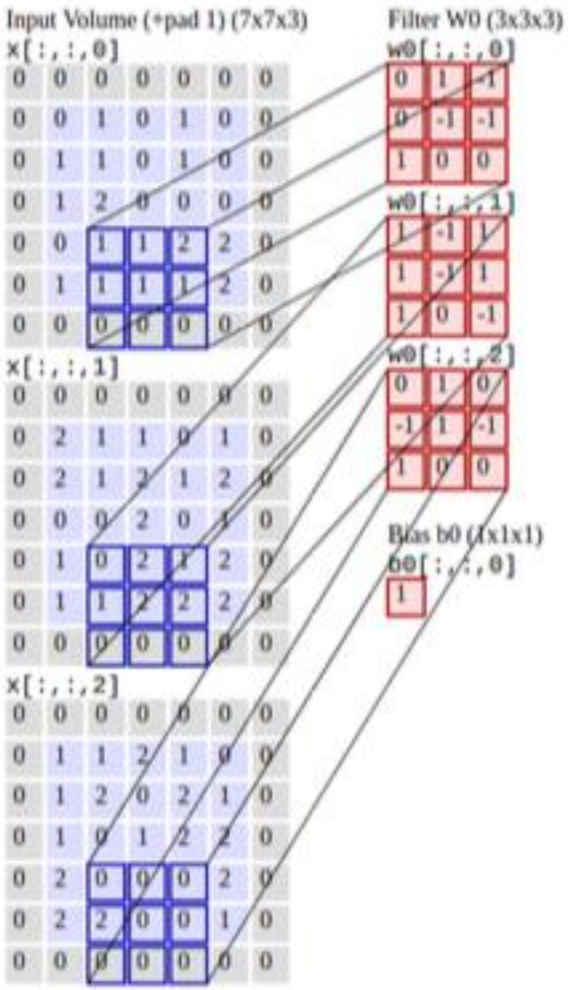
Another reference:
https://cs231n.github.io/convolutional-networks/

Convolutional NN

Feature Maps



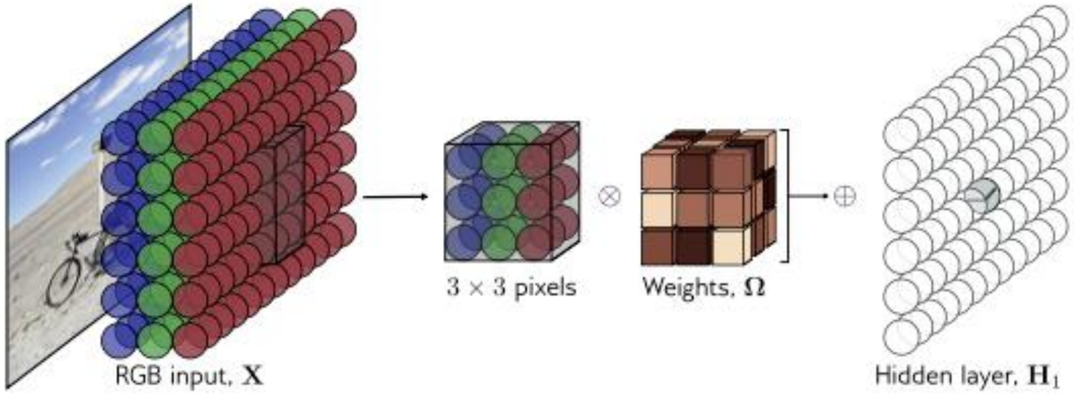
- The map from the input layer to the hidden layer is therefore a feature map: all nodes detect the same feature in different parts
- The map is defined by the shared weights and bias
- The shared map is the result of the application of a convolutional filter (defined by weights and bias), also known as convolution with learned kernels



Multiple Inputs and Filter Banks

Multi-channel input

$$\mathbf{x}_{\text{out}} = \sum_{c} \mathbf{w}[c,:,:] \star \mathbf{x}_{\text{in}}[c,:,:] + b[c]$$

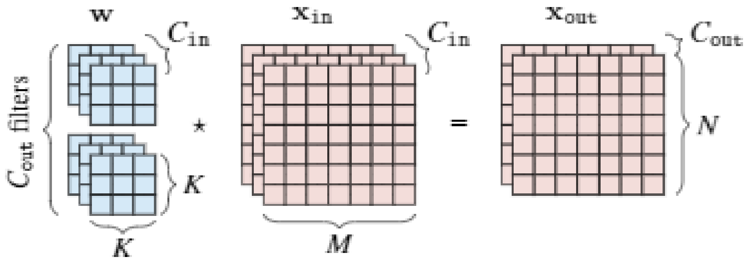


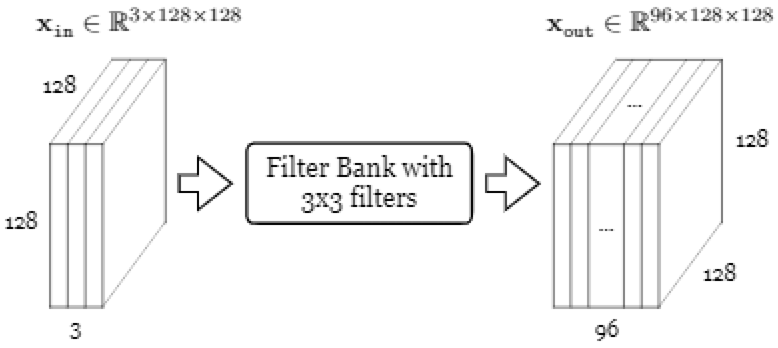
Multi-channel output

$$\mathbf{x}_{\text{out}}[0,:,:] = \mathbf{w}[0,:,:] \star \mathbf{x}_{\text{in}} + b[0]$$

$$\vdots$$

$$\mathbf{x}_{\text{out}}[C,:,:] = \mathbf{w}[C-1,:,:] \star \mathbf{x}_{\text{in}} + b[C-1]$$





Convolution Layers and Receptive Fields

Receptive Fields – part of the input that the neuron focuses on.

Deeper layers have larger receptive fields.

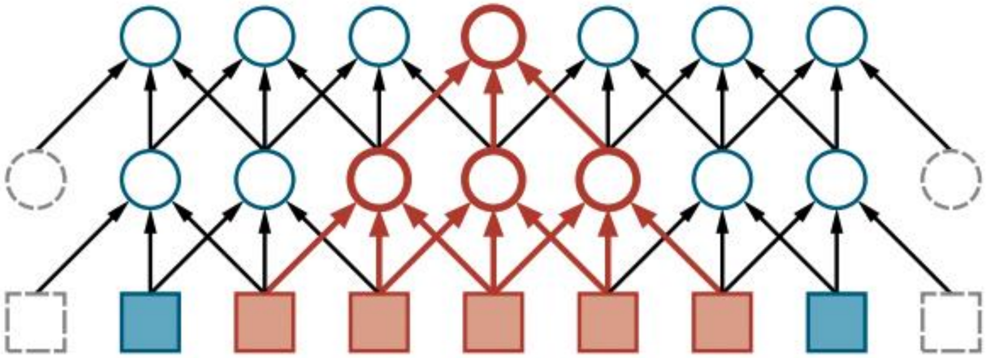


Figure 21.5 The first two layers of a CNN for a 1D image with a kernel size $l = 3$ and a stride $s = 1$. Padding is added at the left and right ends in order to keep the hidden layers the same size as the input. Shown in red is the receptive field of a unit in the second hidden layer. Generally speaking, the deeper the unit, the larger the receptive field.

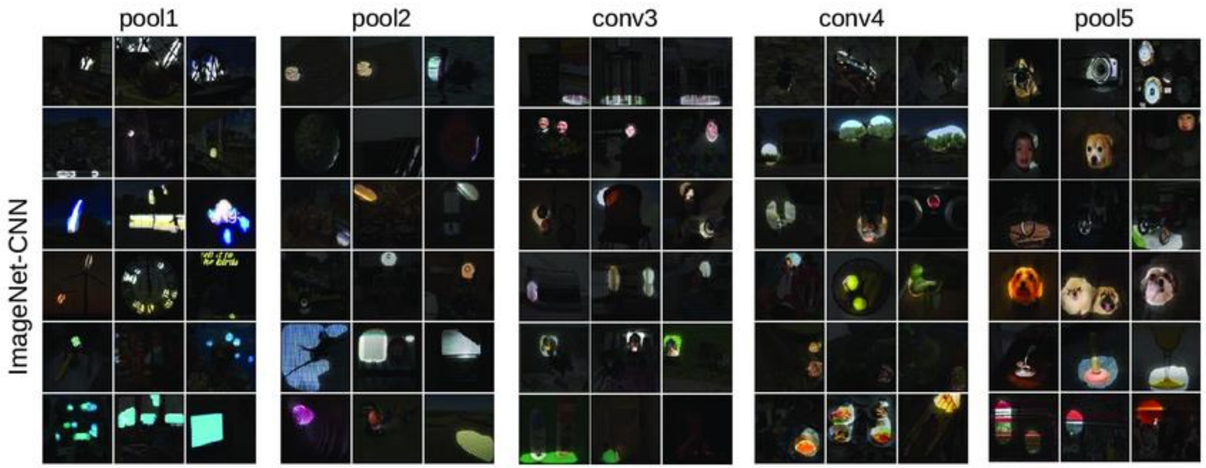
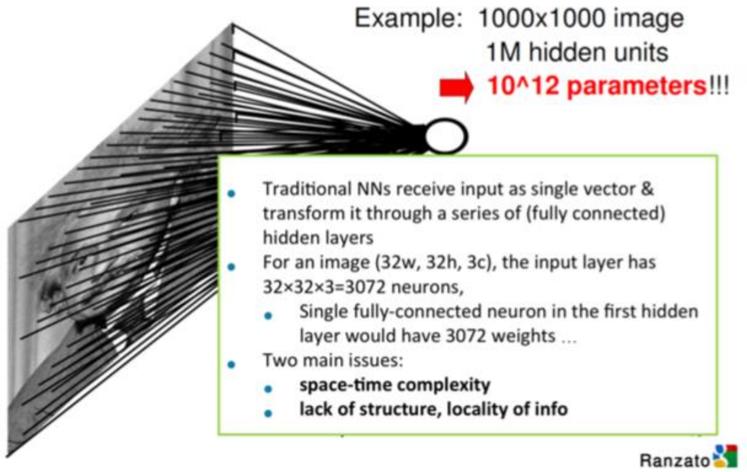


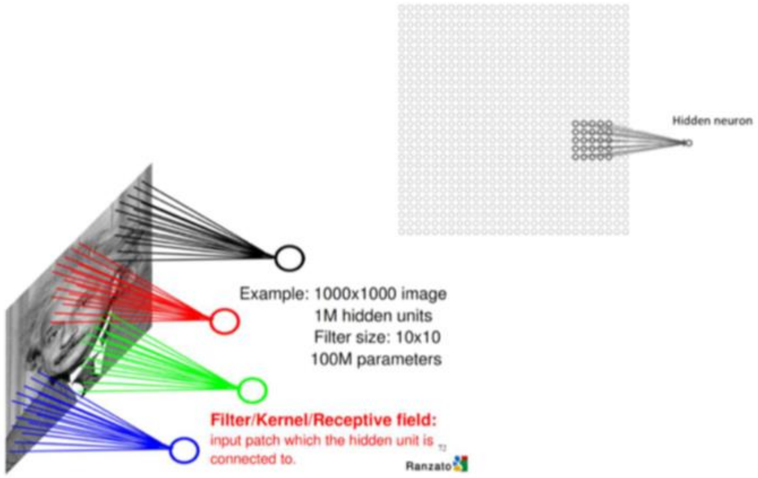
Figure: AIMA Fourth Edition (Chapter Deep Learning)

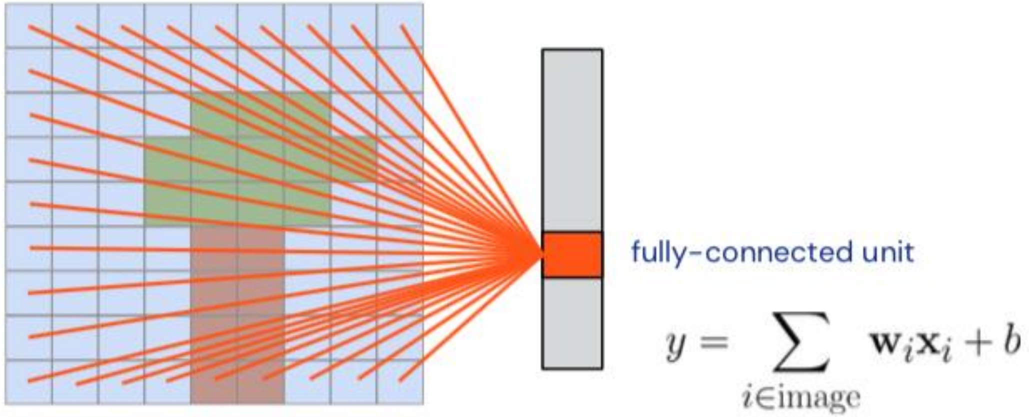
Locality of Information: Receptive Fields

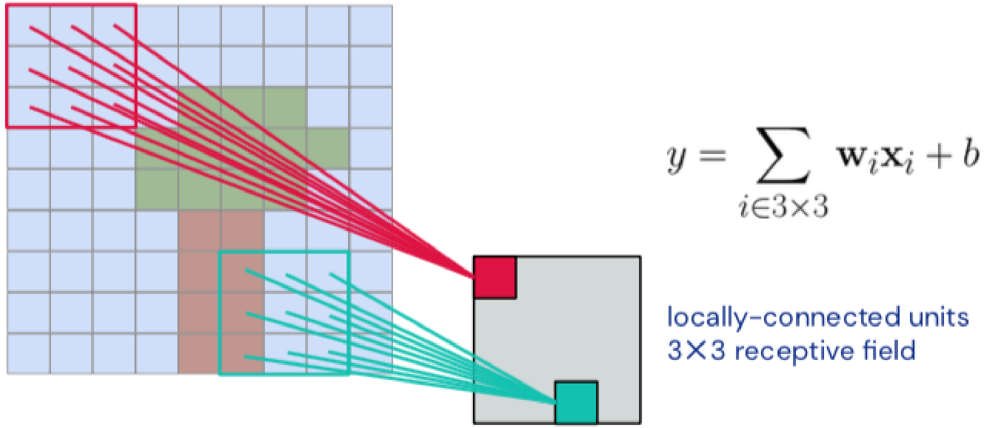
Fully connected network.



Convolutional NN





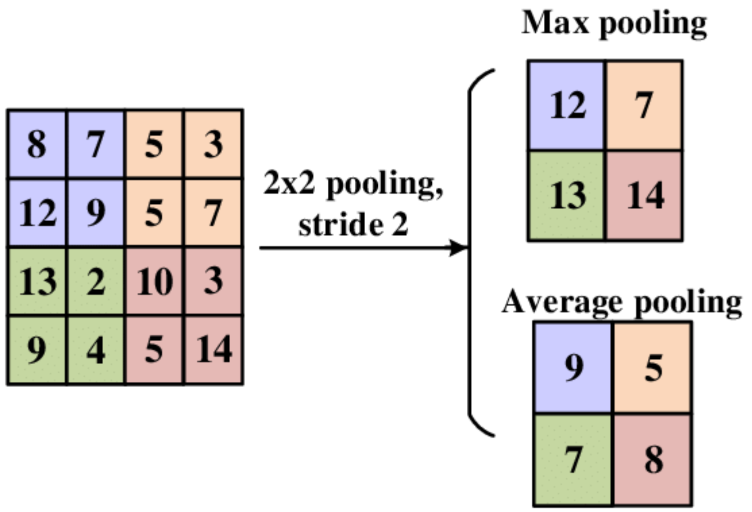


Pooling layers

- Pooling layers are down sampling layers that summarize the information in a patch using some aggregate statistic.
  - Mean pooling
  - Max pooling
- Reduce the resolution of the input, removing the high-frequency information from the signal.
- CNN layers produce outputs that are equivariant to translations in their input.
- Pooling is a way to convert equivariance into invariance.

$$x_{\mathrm{out}}[i] = \max_{i \in \mathcal{N}(i)} x_{\mathrm{in}}[i] \qquad / \quad \texttt{max pooling}$$

$$x_{\mathrm{out}}[i] = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}(i)} x_{\mathrm{in}}[i] \qquad / \quad \texttt{mean pooling}$$

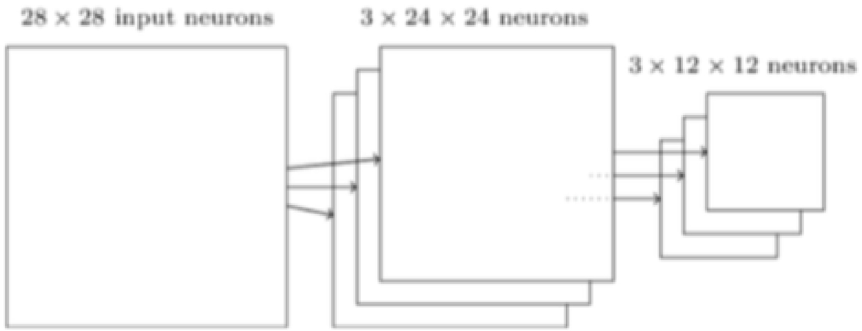


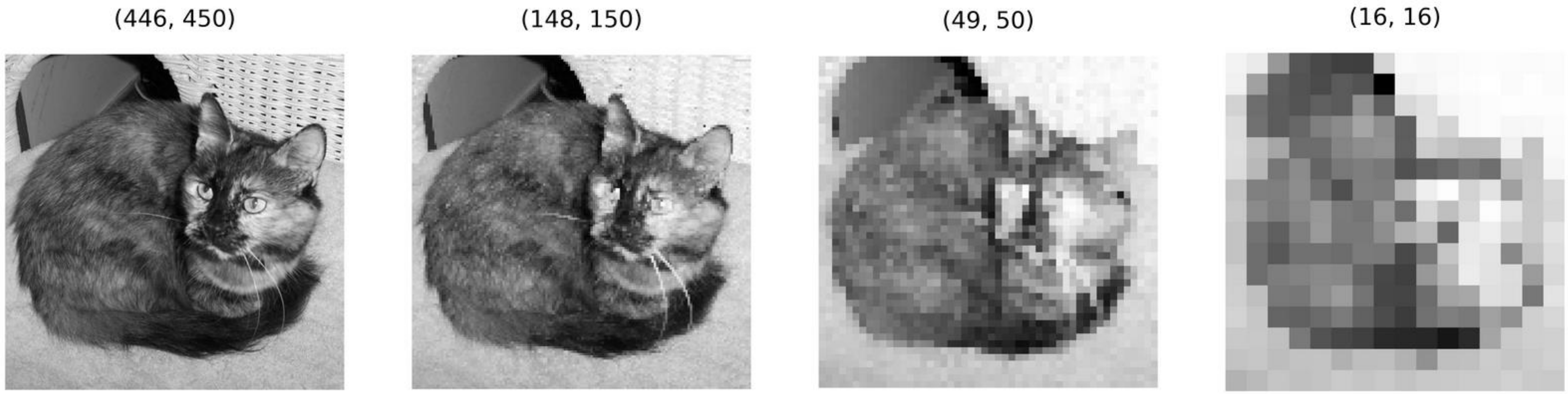
Pooling layers

Pooling layers are usually used immediately after convolutional layers.

Pooling layers simplify / subsample / compress the information in the output from convolutional layer

A pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map



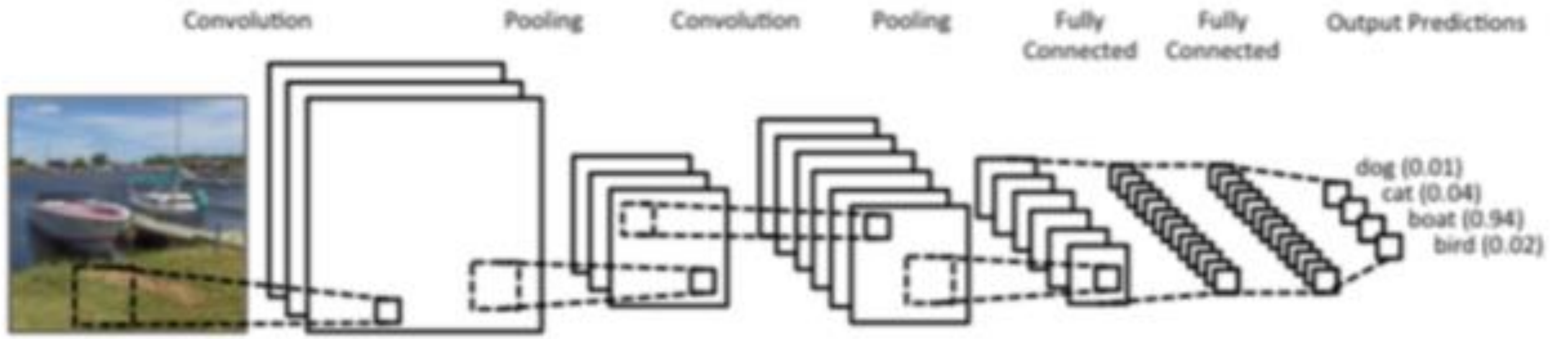


Pooling layers aggregate the data and lower the resolution.

Pooling layers - intuition

- Example
  - Suppose we run a convolution filter wherever there is a vertical edge in the input image.
  - If we run Max pooling filter, then it will coarsen the map resulting in a large response anywhere “near” where there was a vertical edge in the input image.
  - If we use Max-pooling with a large enough neighborhood N, the output will be invariant to the location of the edge in the input image.
  - One extreme of pooling is global average or max pooling that is over the entire image.
- It is common to have convolution-pooling-convolution-pooling layers in sequence

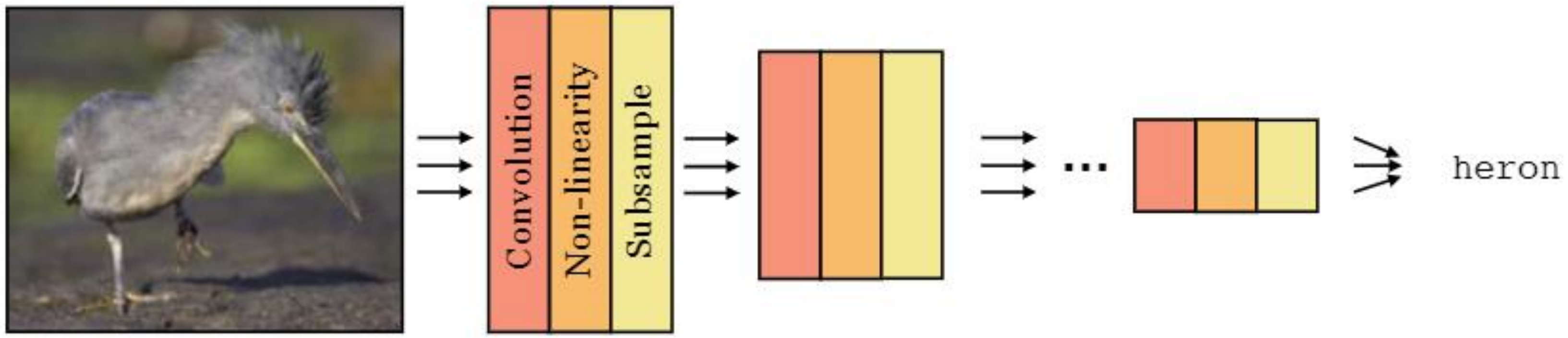
Typical classification pipeline



- Consider local structure and common extraction of features
- Not fully connected. Locality of processing
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks
- Compress to extract salient features & favor generalization

https://poloclub.github.io/cnn-explainer/

A Simple CNN Classifier



$$z_1[c,:,:] = w[c,:,:] \star x + b[c] \qquad / \quad \texttt{conv}: [M \times N] \to [C \times M \times N]$$

$$h[c,n,m] = \max(z_1[c,n,m], 0) \qquad / \quad \texttt{relu}: [C \times M \times N] \to [C \times M \times N]$$

$$z_2[c] = \frac{1}{NM} \sum_{n,m} h[c,n,m] \qquad / \quad \texttt{gap}: [C \times M \times N] \to [C]$$

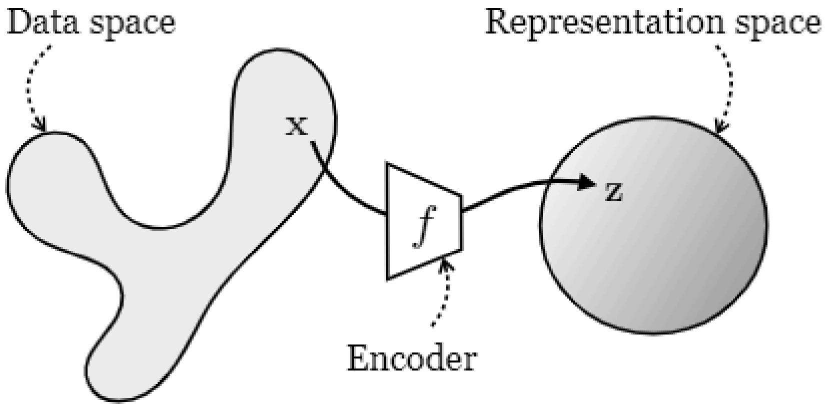
$$z_3 = \mathbf{W} z_2 + c \qquad / \quad \texttt{fc}: [C] \to [K]$$

$$y[k] = \frac{e^{-\tau z_3[k]}}{\sum_{l=1}^{K} e^{-\tau z_3[l]}} \qquad / \quad \texttt{softmax}: [K] \to [K]$$

Note that these equations apply for all $c \in \{0, \ldots, C-1\}$, $n \in \{0, \ldots, N-1\}$ and $m \in \{0, \ldots, M-1\}$.

Learning Representations

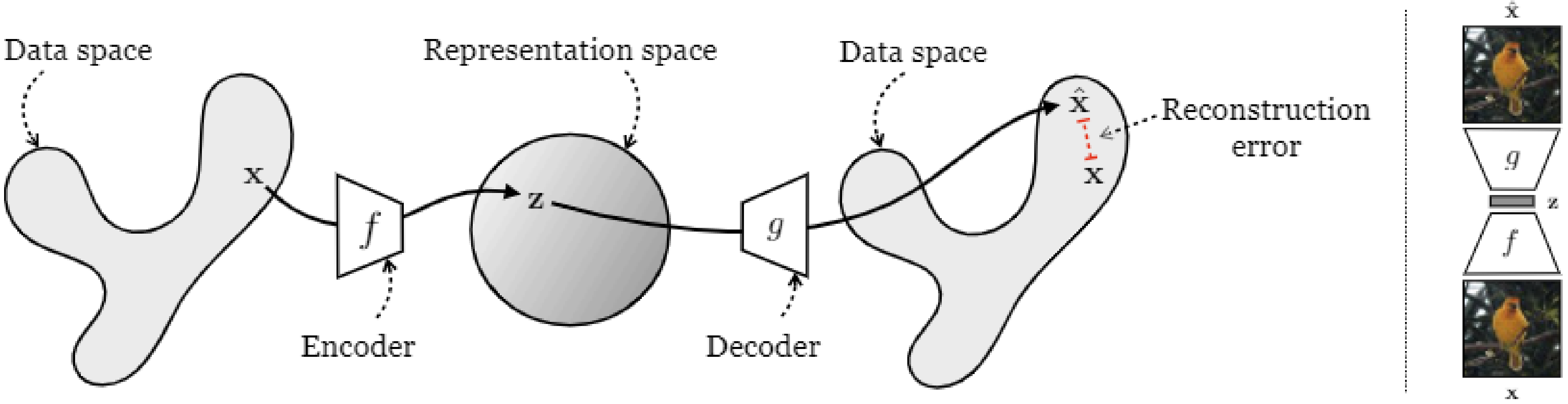
- Task
  - Given data points x, we want to learn a ”good” and “compact” representation of the data in an *unsupervised* manner
  - Obtain Features that capture the necessary aspects of the data but not the noise aspect.
- Utility
  - We would like to extract good features from data for example for a recognition task.
- Think – as acquiring a latent embedding/representation of the data
  - Consider modeling images of hand-written digits. The latent representation z can capture the pen strokes, colors etc. Variables that will impact the generation of the data.



Abstractly, we want an encoder of the raw data to an embedding that is compact and representative of the data.

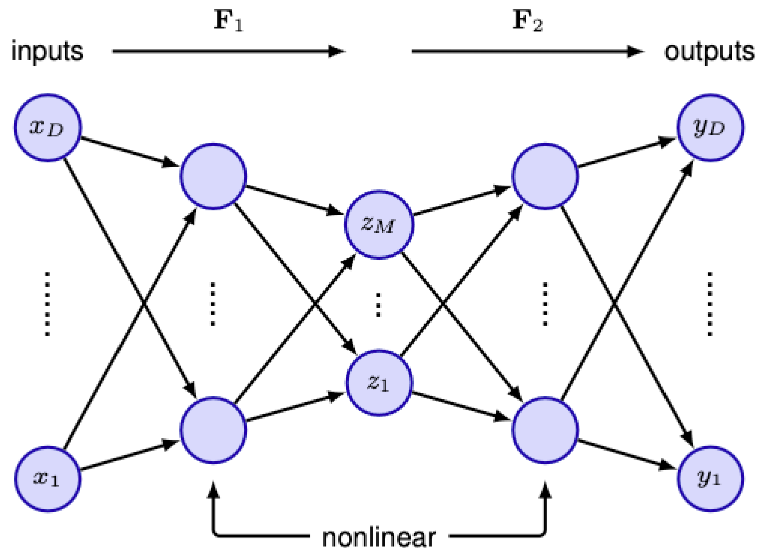
Auto-encoders

- Learn a function that maps the data back to itself
- Via a low-dimensional representation bottleneck. Dimensionality reduction is inherent in it.
- Minimizes the reconstruction loss.

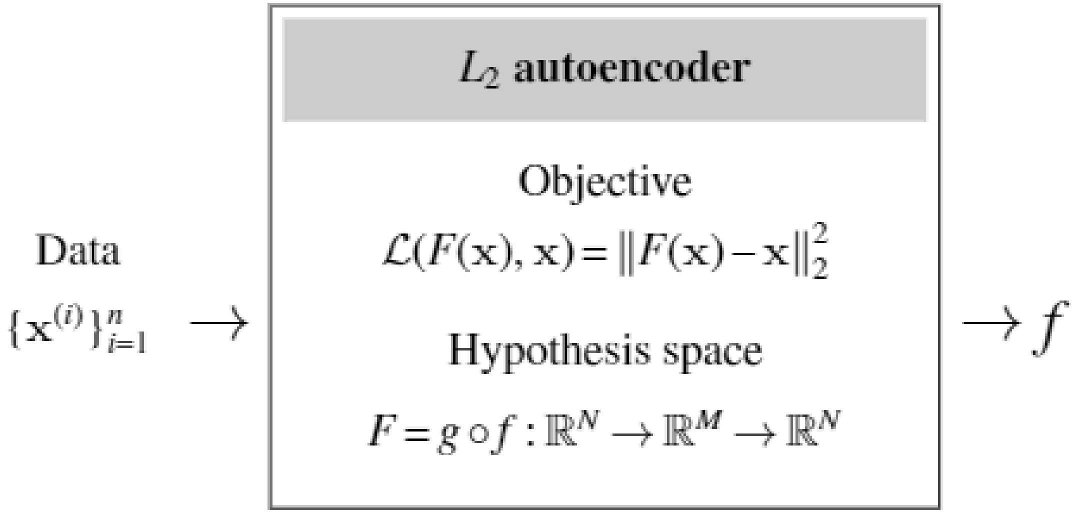


$$f^*, g^* = \arg\min_{f,g} \mathbb{E}_{\mathbf{x}} \|g(f(\mathbf{x})) - \mathbf{x}\|_2^2$$

Auto-encoders



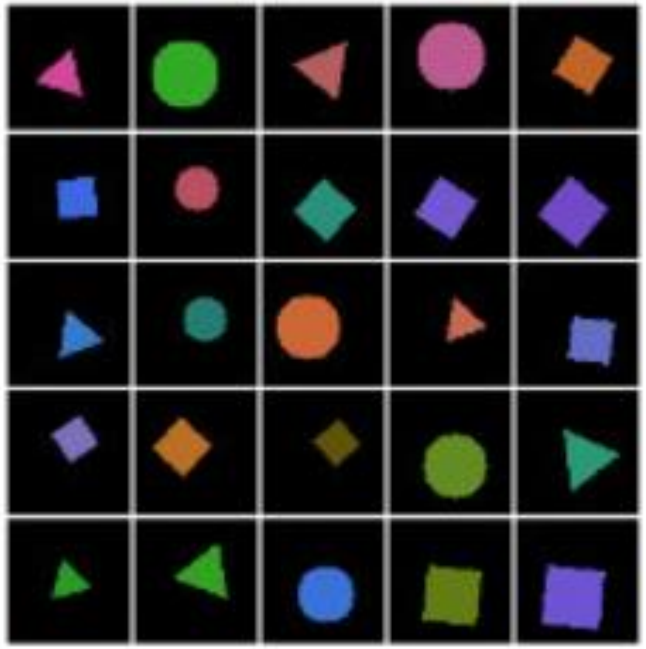
Both the encoder and the decoder are neural networks.



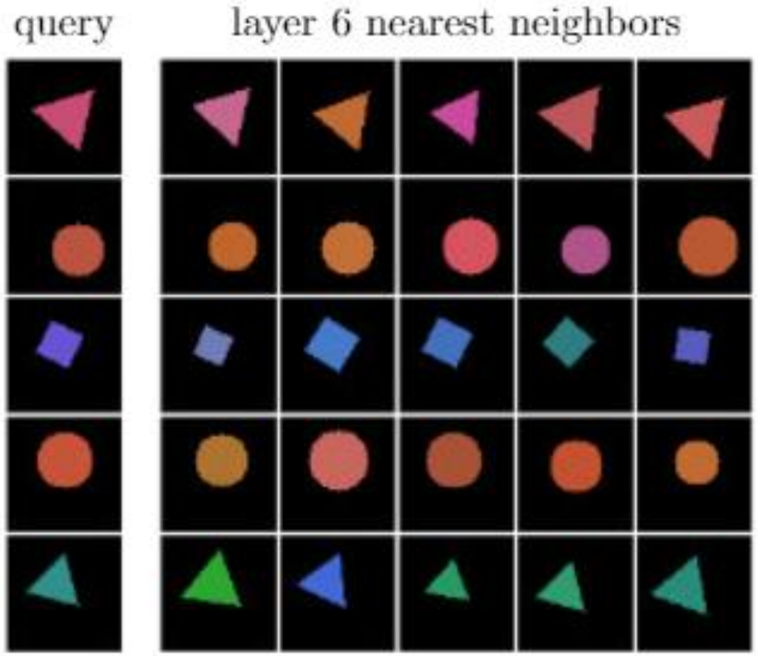
Can view the autoencoder as learning the “best mapping” from data to itself with a bottleneck.

Example of using the representation

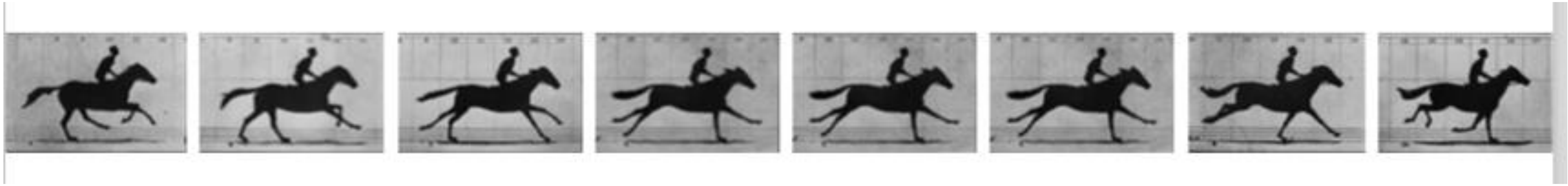
Unlabeled samples from the data set.

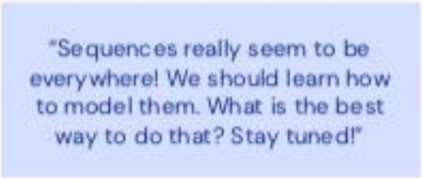


Nearest neighbor look up using the autoencoder features.

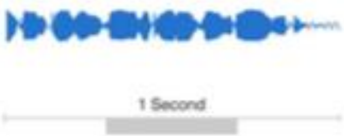


Neural Networks for Sequences



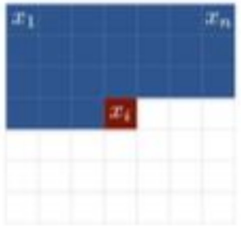


Words, letters



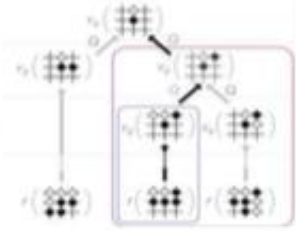
Speech

Videos



Images

Programs



Decision making

Collection of elements where elements can be repeated, order matters and can be of variable or infinite length.

Modeling Sequences

	Supervised learning	Sequence modelling
Data	$\{x, y\}_i$	$\{x\}_i$
Model	$y \approx f_\theta(x)$	$p(x) \approx f_\theta(x)$
Loss	$\mathcal{L}(\theta) = \sum_{i=1}^{N} l(f_\theta(x_i), y_i)$	$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(f_\theta(x_i))$
Optimisation	$\theta^* = \arg\min_\theta \mathcal{L}(\theta)$	$\theta^* = \arg\max_\theta \mathcal{L}(\theta)$

Modeling the conditional distribution

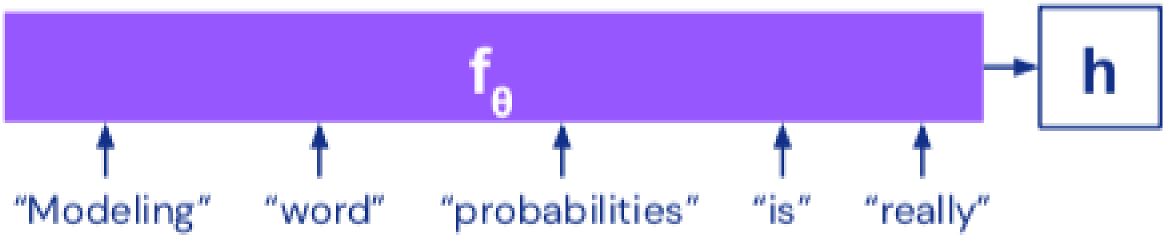
The chain rule

Computing the joint p(**x**) from conditionals

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1})$$

Modeling	$p(x_1)$
Modeling **word**	$p(x_2 \| x_1)$
Modeling word **probabilities**	$p(x_3 \| x_2, x_1)$
Modeling word probabilities **is**	$p(x_4 \| x_3, x_2, x_1)$
Modeling word probabilities is **really**	$p(x_5 \| x_4, x_3, x_2, x_1)$
Modeling word probabilities is really **difficult**	$p(x_6 \| x_5, x_4, x_3, x_2, x_1)$

Vectorizing the conditional likelihood

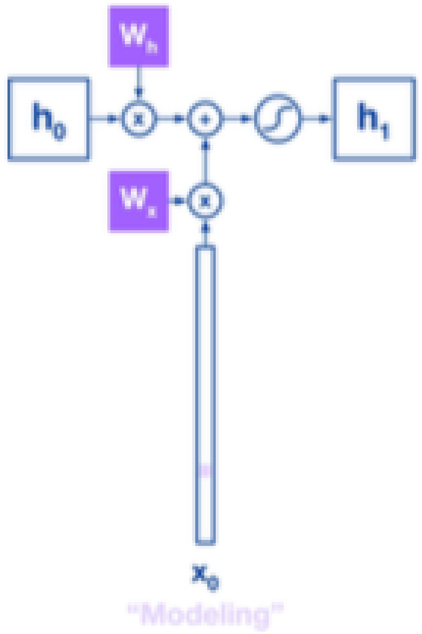


Desirable properties for f_θ:

- Order matters
- Variable length
- Learnable (differentiable)
- Individual changes can have large effects (non-linear/deep)

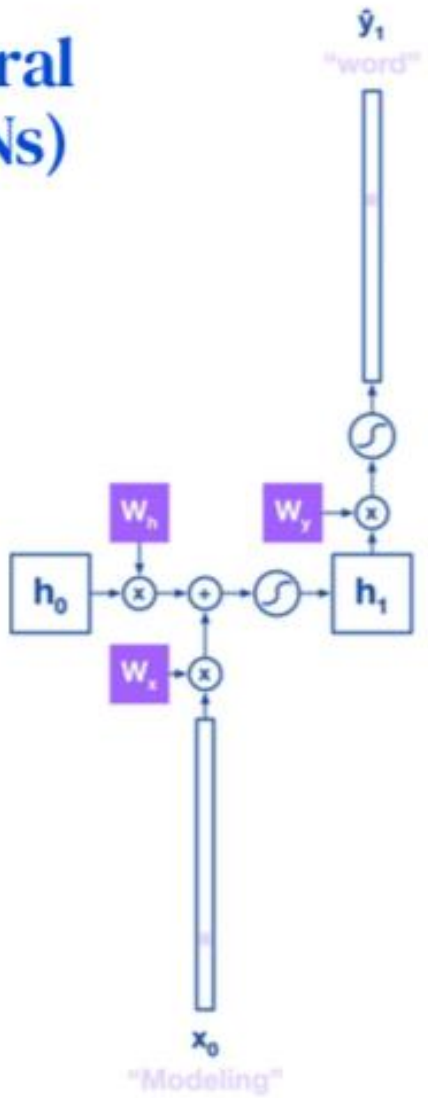
Recurrent Neural Networks (RNNs)

Persistent state variable **h** stores information from the context observed so far.



$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

Recurrent Neural Networks (RNNs)

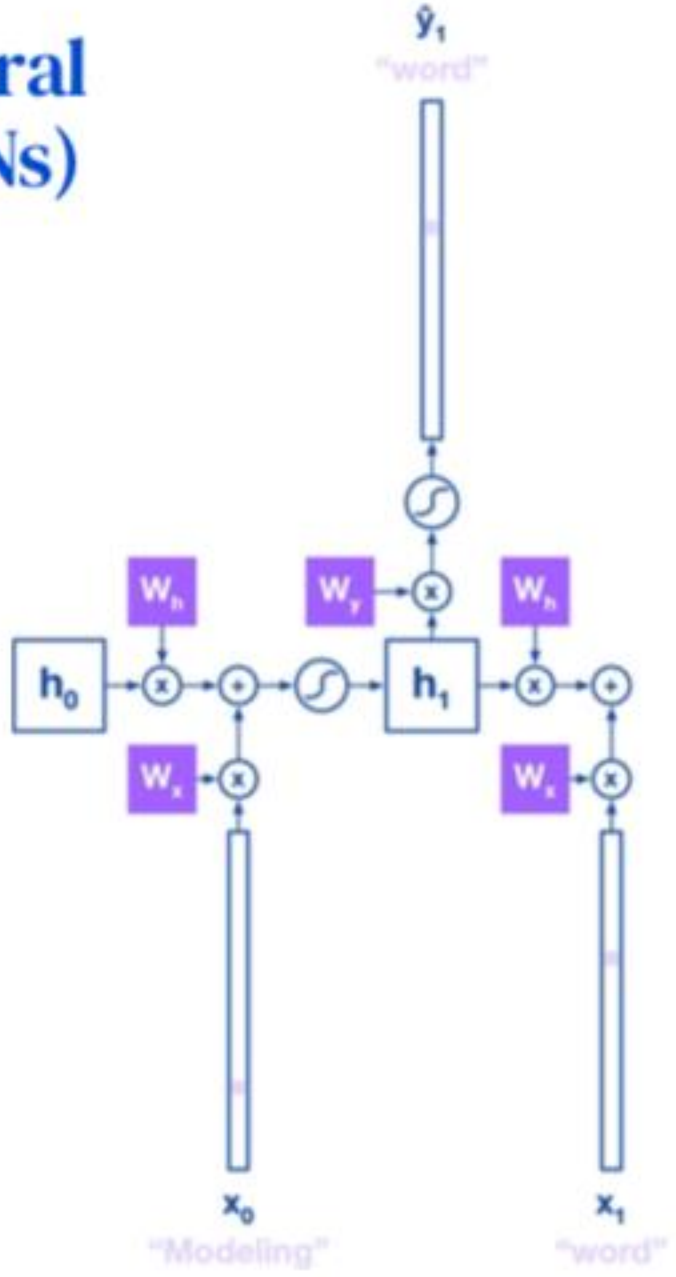


RNNs predict the target **y** (the next word) from the state **h.**

$$p(\mathbf{y_{t+1}}) = softmax(\mathbf{W}_y\mathbf{h}_t)$$

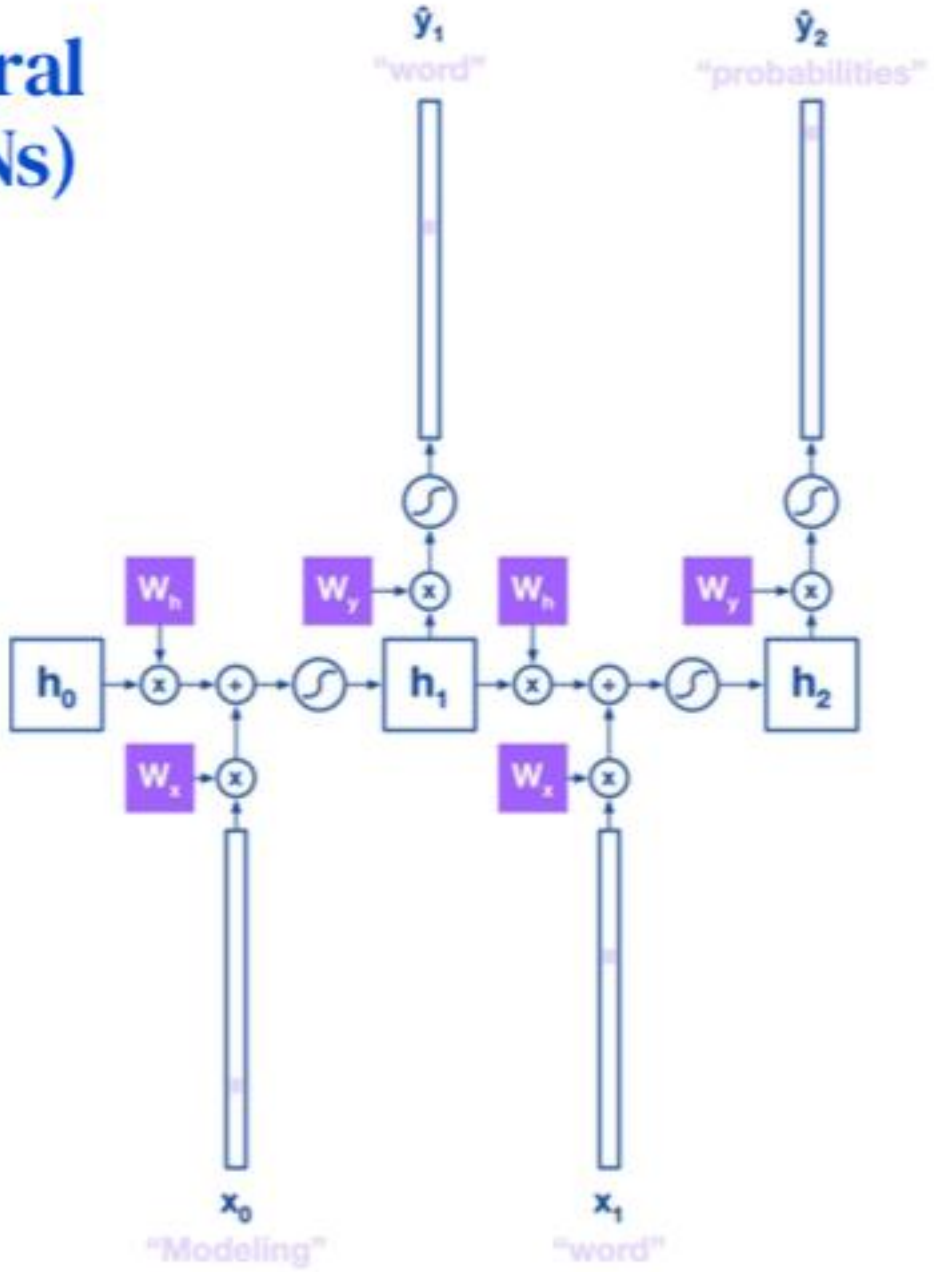
Softmax ensures we obtain a distribution over all possible words.

Recurrent Neural Networks (RNNs)



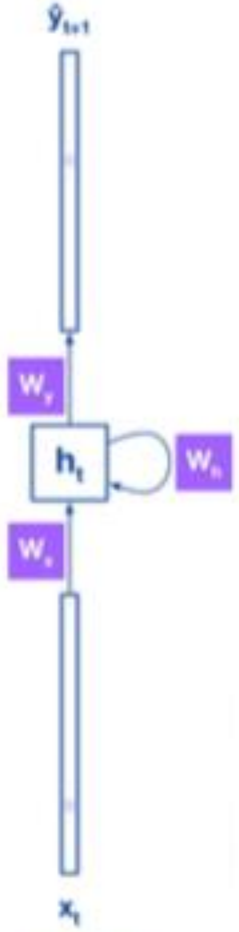
Input next word in sentence x_1

Recurrent Neural Networks (RNNs)

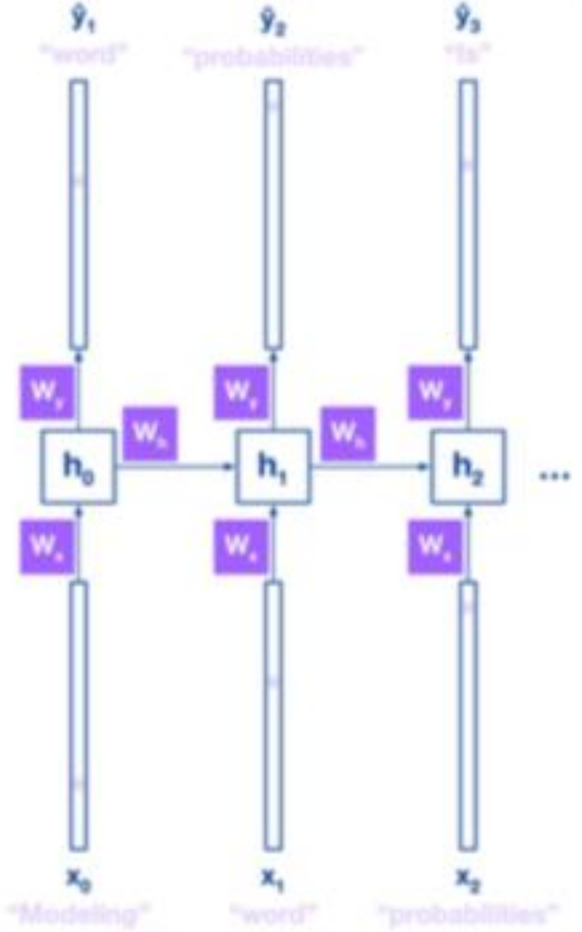


Recurrent Neural Networks (RNNs)

Weights are shared over time steps



RNN



RNN rolled out over time

Loss: Cross Entropy

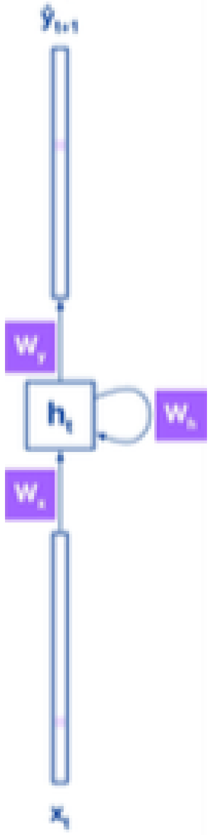
Next word prediction is essentially a classification task where the number of classes is the size of the vocabulary.

As such we use the cross-entropy loss:

For one word: $$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

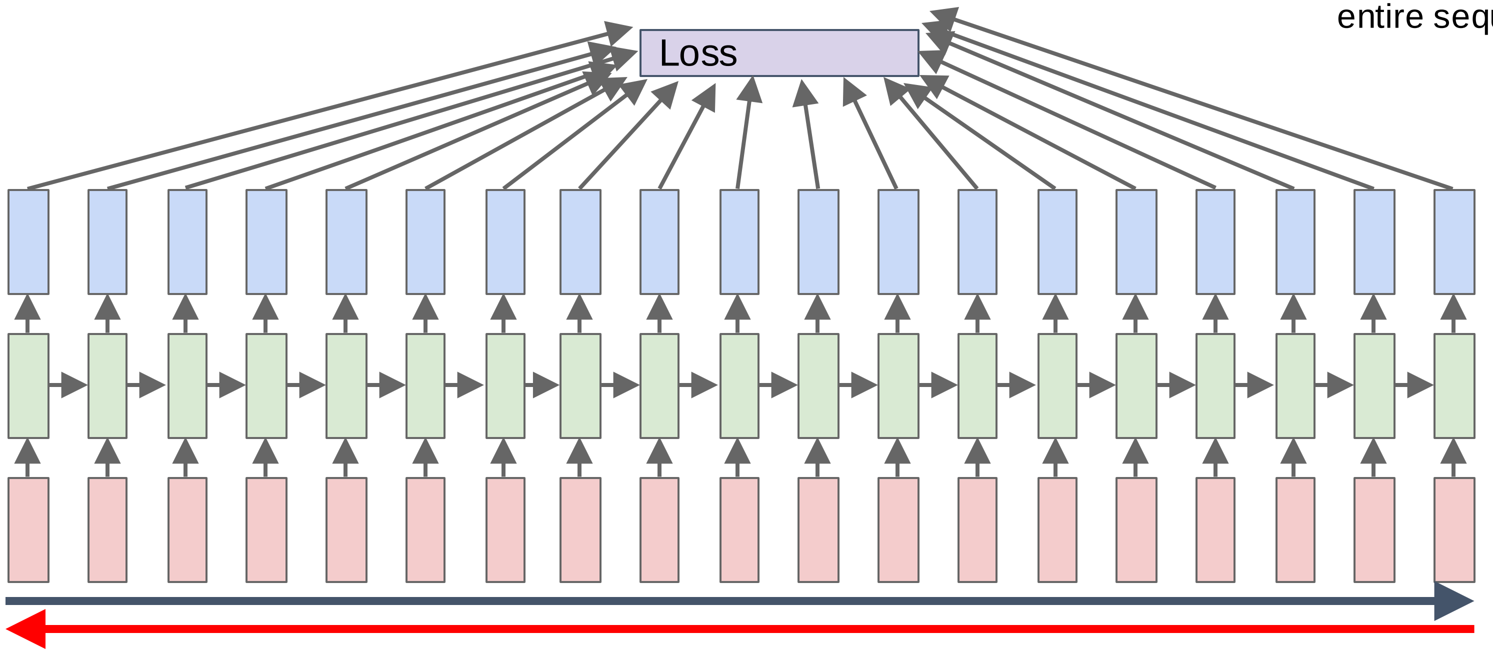
For the sentence: $$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{t=1}^{T} \mathbf{y}_t \log \hat{\mathbf{y}}_t$$

With parameters $\theta = \{\mathbf{W}_y, \mathbf{W}_x, \mathbf{W}_h\}$



Backprop through Time

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

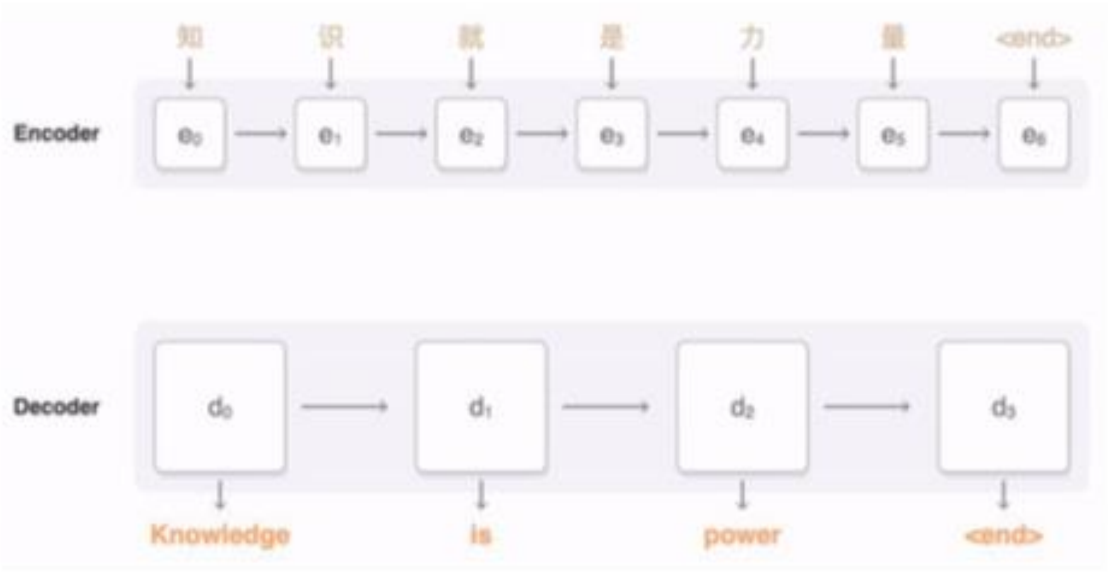


RNNs can have long or short dependencies. When there are long dependencies, gradients have trouble back-propagating through.

Other models such as LSTMs and beyond address that problem.

Applications

Google Neural Machine Translation



Wu et al, 2016

(Kalchbrenner et al, 2013; Sutskever et al, 2014; Cho et al, 2014; Bhadanau et al, 2014; ...)

Applications

$p(\text{language}_1 \mid \text{language}_2) \rightarrow p(\text{language}_1 \mid \text{image})$

Human: A brown dog laying in a red wicker bed.

Best Model: A small dog is sitting on a chair.

Initial Model: A large brown dog laying on top of a couch.