DS 4400

Machine Learning and Data Mining I

Alina Oprea
Associate Professor, CCIS
Northeastern University

November 13 2018

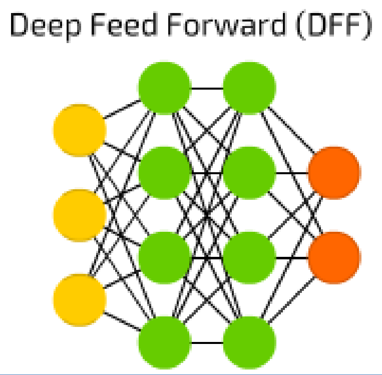
Review

- To train neural networks, need to decide first on architecture
  - Number of layers, number of hidden units, connections between neurons, activation functions
- Randomly initialize parameters
- For each training example, use forward propagation to compute prediction
- Use backpropagation to propagate the error from last layer back into the network
- Stochastic Gradient Descent with mini-batch update is the default optimization method

Neural Network Architectures

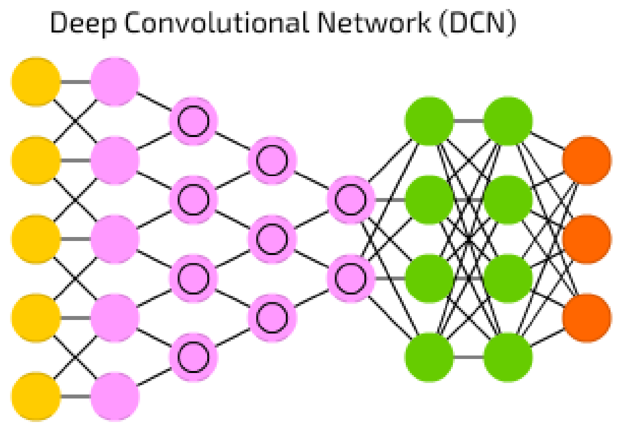
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



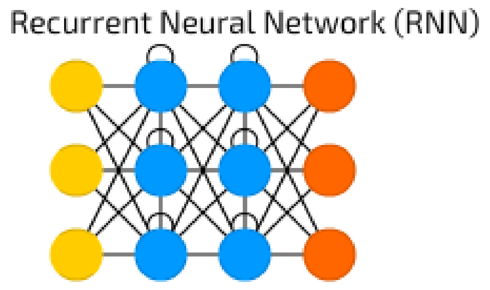
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations



Recurrent Networks

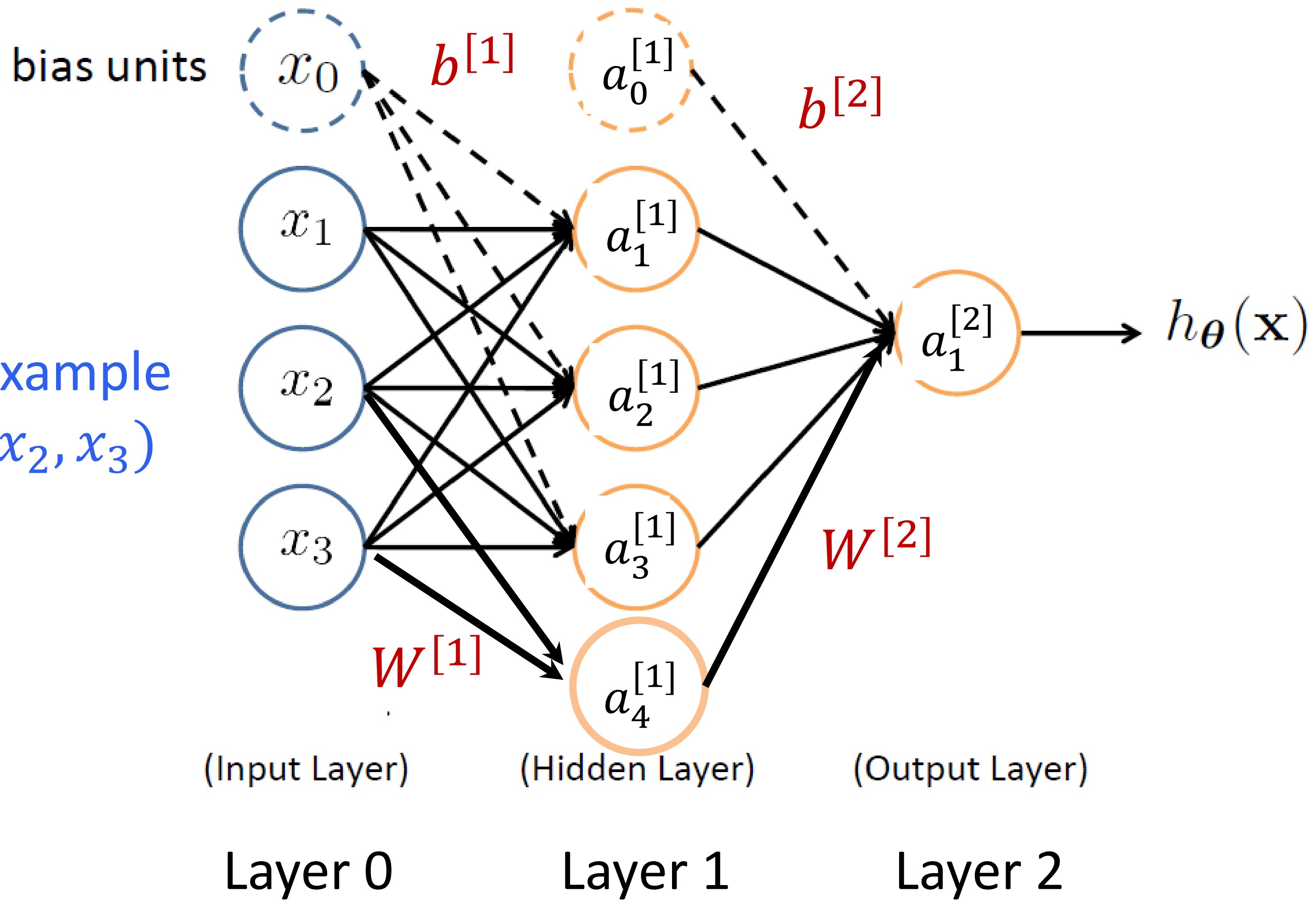
- Keep hidden state
- Have cycles in computational graph



Feed-Forward Neural Network

Training example

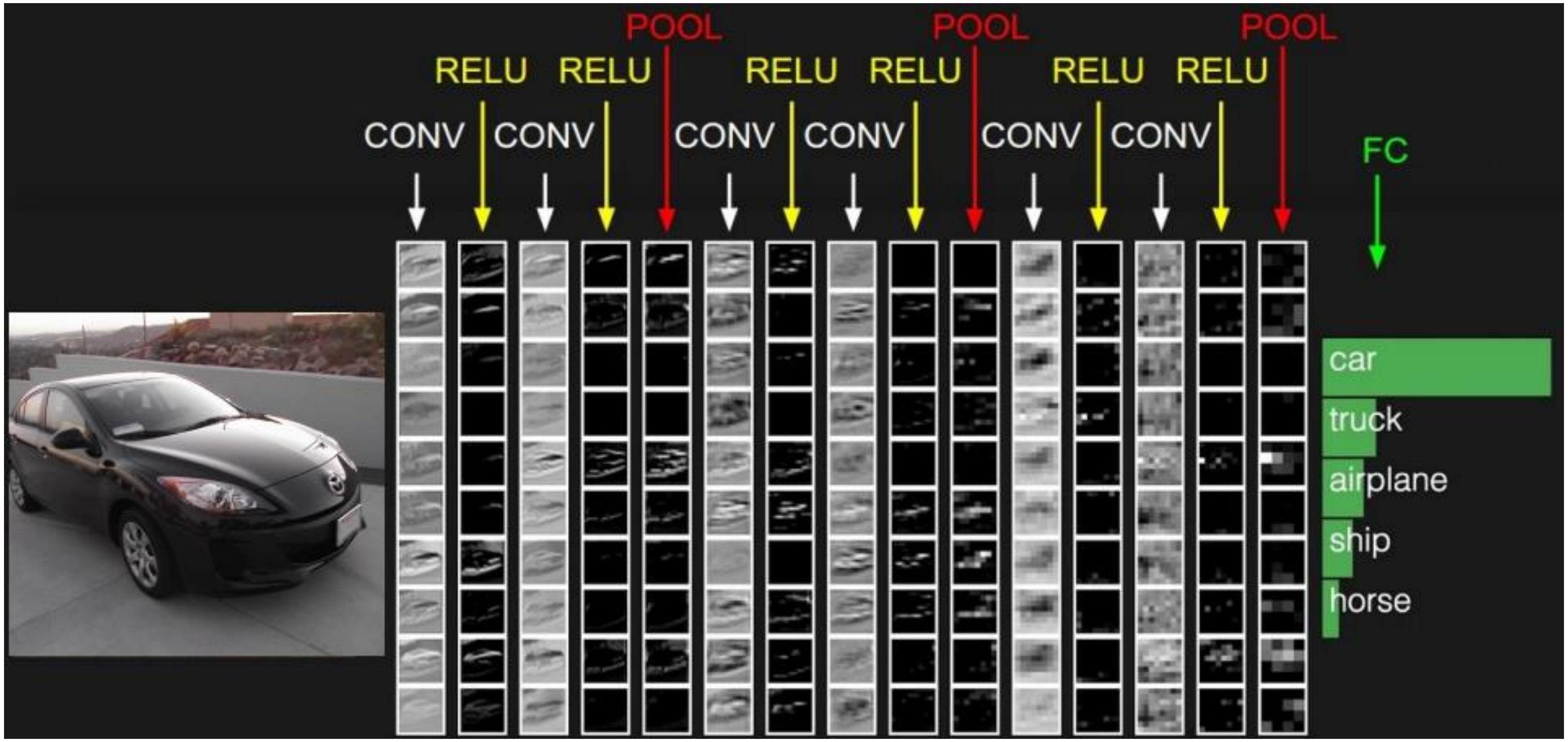
$x = (x_1, x_2, x_3)$



No cycles

$\theta = (b^{[1]}, W^{[1]}, b^{[2]}, W^{[2]})$

Convolutional Nets



Mini-batch Gradient Descent

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - For all layers ℓ (starting backwards)
    - For all batches b of size B with training examples $x^{(ib)}, y^{(ib)}$

$$-\, W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}^{(ib)}, y^{(ib)})}{\partial W^{[\ell]}}$$

$$-\, b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}^{(ib)}, y^{(ib)})}{\partial b^{[\ell]}}$$

Training NN with Backpropagation

Given training set $(x_1, y_1), \ldots, (x_N, y_N)$
Initialize all parameters $W^{[\ell]}, b^{[\ell]}$ randomly, for all layers ℓ
Loop

```
    Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$                    (Used to accumulate gradient)
    For each training instance $(x^{(i)}, y^{(i)})$
        Set $\mathbf{a}^{(1)} = \mathbf{x}_i$
        Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation          EPOCH
        Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y^{(i)}$
        Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
        Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
```

    Update weights via gradient step

- $W_{ij}^{[\ell]} = W_{ij}^{[\ell]} - \alpha \frac{\Delta_{ij}^{[\ell]}}{N}$
- Similar for $b_{ij}^{[\ell]}$

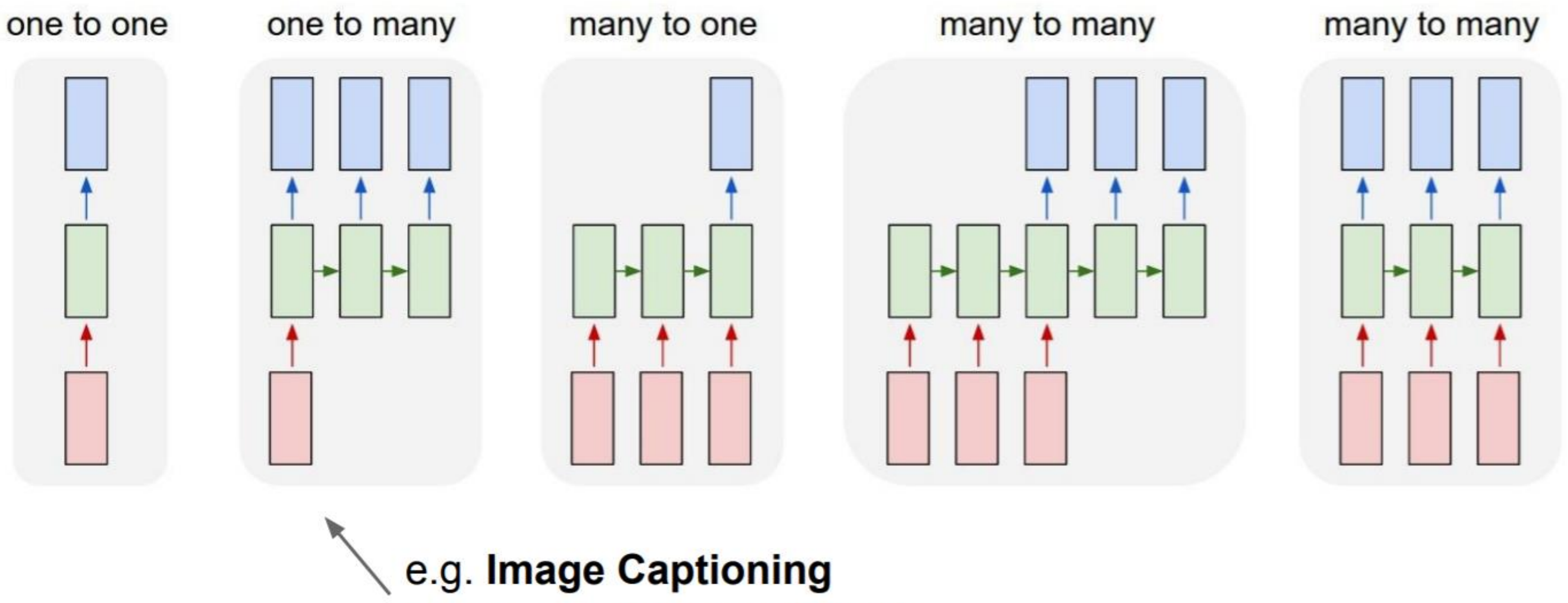
Until weights converge or maximum number of epochs is reached

Outline

- Recurrent Neural Networks (RNNs)
  - One-to-one, one-to-many, many-to-one, many-to-many
  - Blog by Andrej Karpathy
    - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Unsupervised learning
- Dimensionality reduction
  - PCA

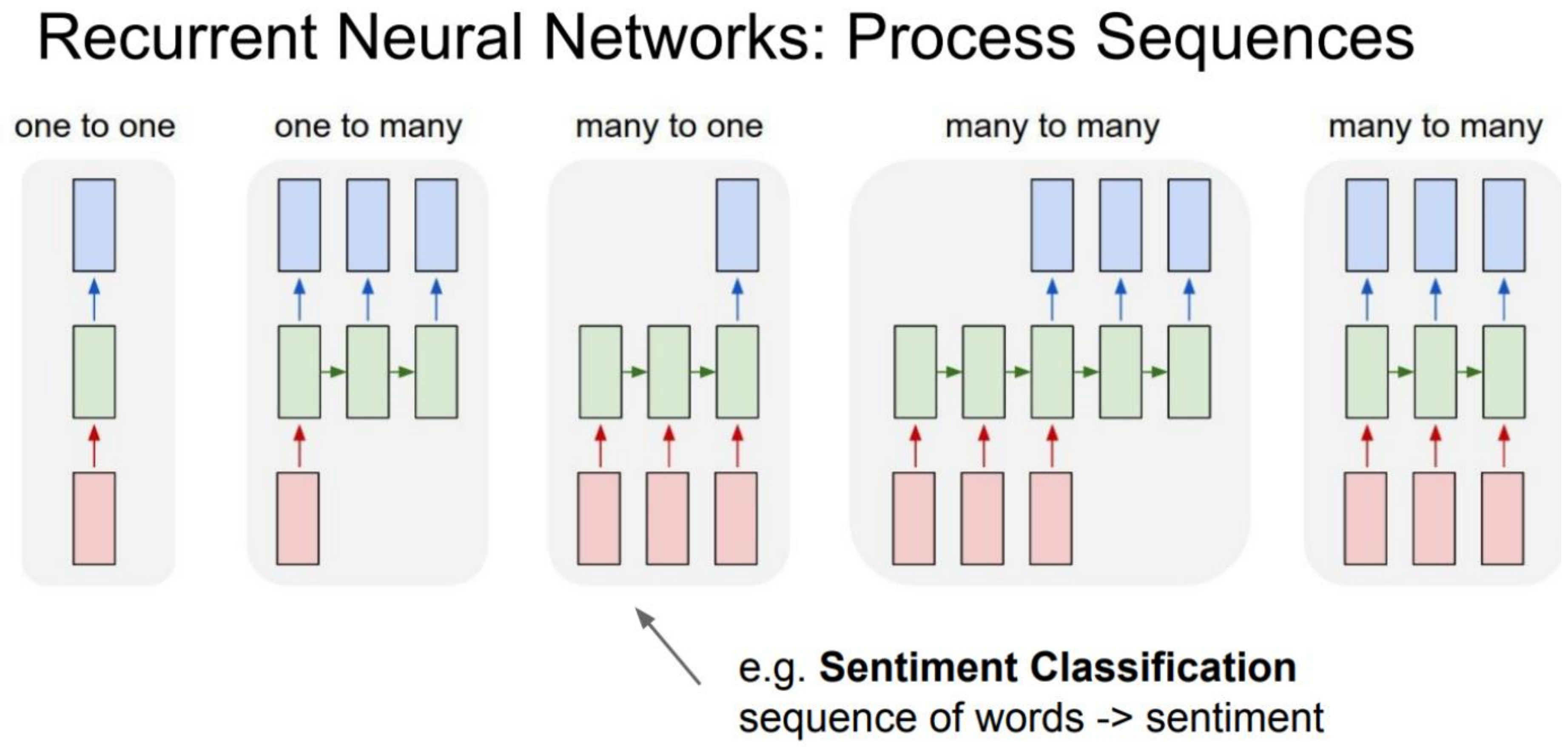
RNN Architectures

Recurrent Neural Networks: Process Sequences

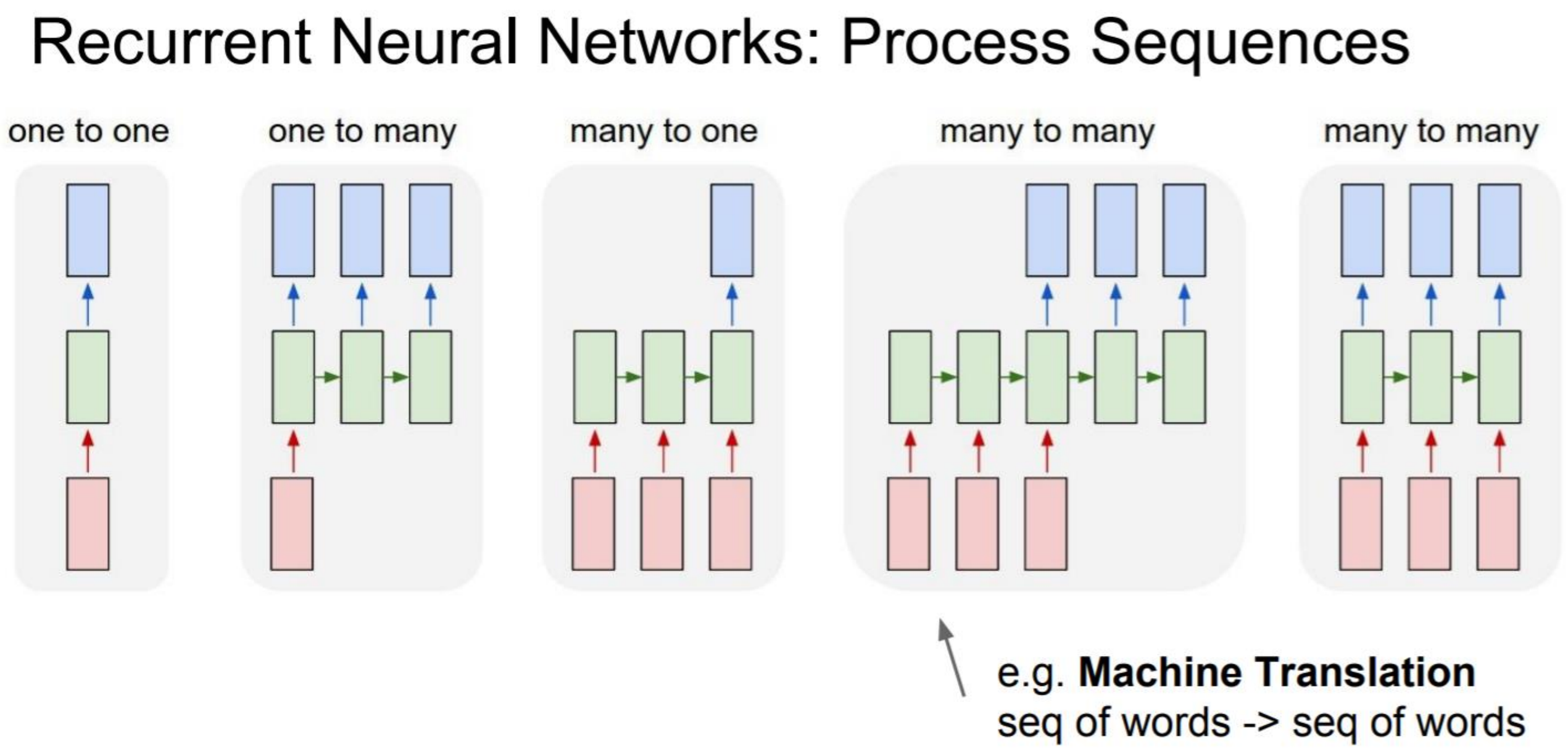


e.g. **Image Captioning**
image -> sequence of words

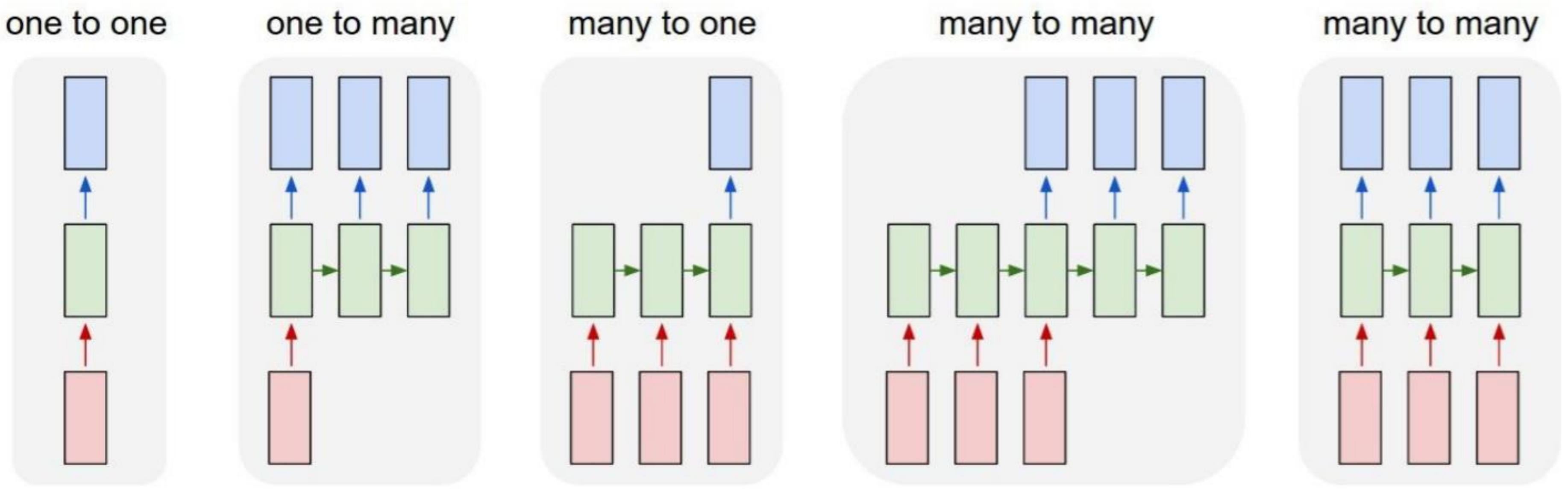
RNN Architectures


Recurrent Neural Networks: Process Sequences
one to one
one to many
many to one
many to many
many to many
e.g. **Sentiment Classification**
sequence of words -> sentiment

RNN Architectures


Recurrent Neural Networks: Process Sequences
one to one
one to many
many to one
many to many
many to many
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Neural Networks: Process Sequences



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

Recurrent Neural Network

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

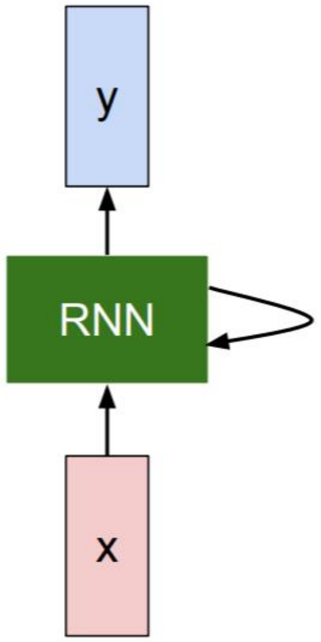
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

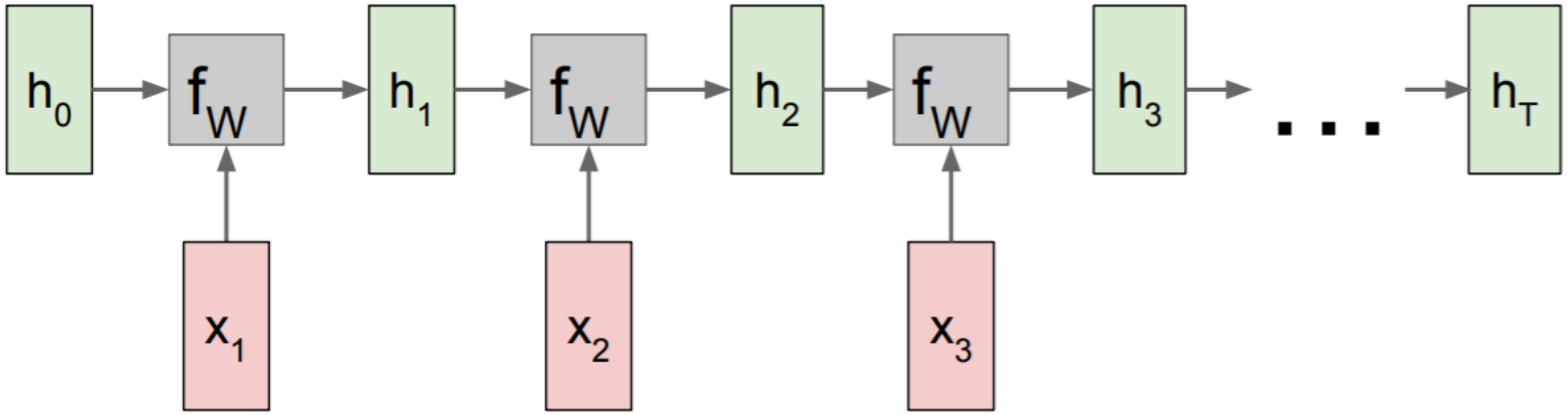
input vector at some time step

some function with parameters W



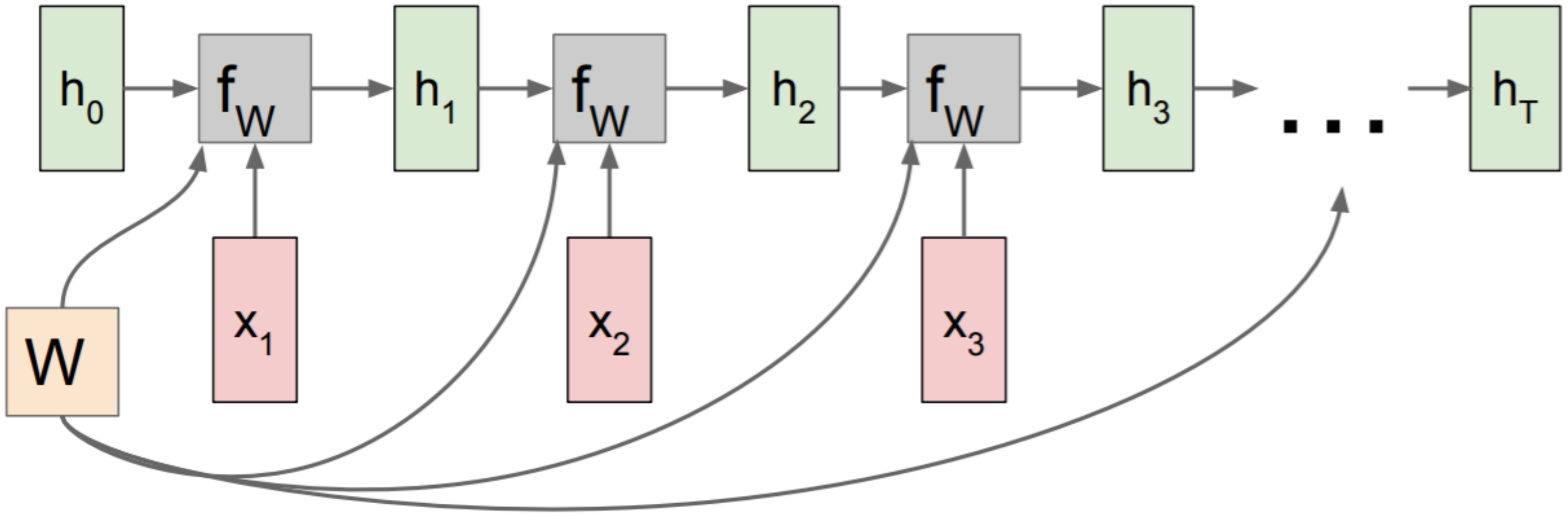
Notice: the same function and the same set of parameters are used at every time step.

RNN: Computational Graph

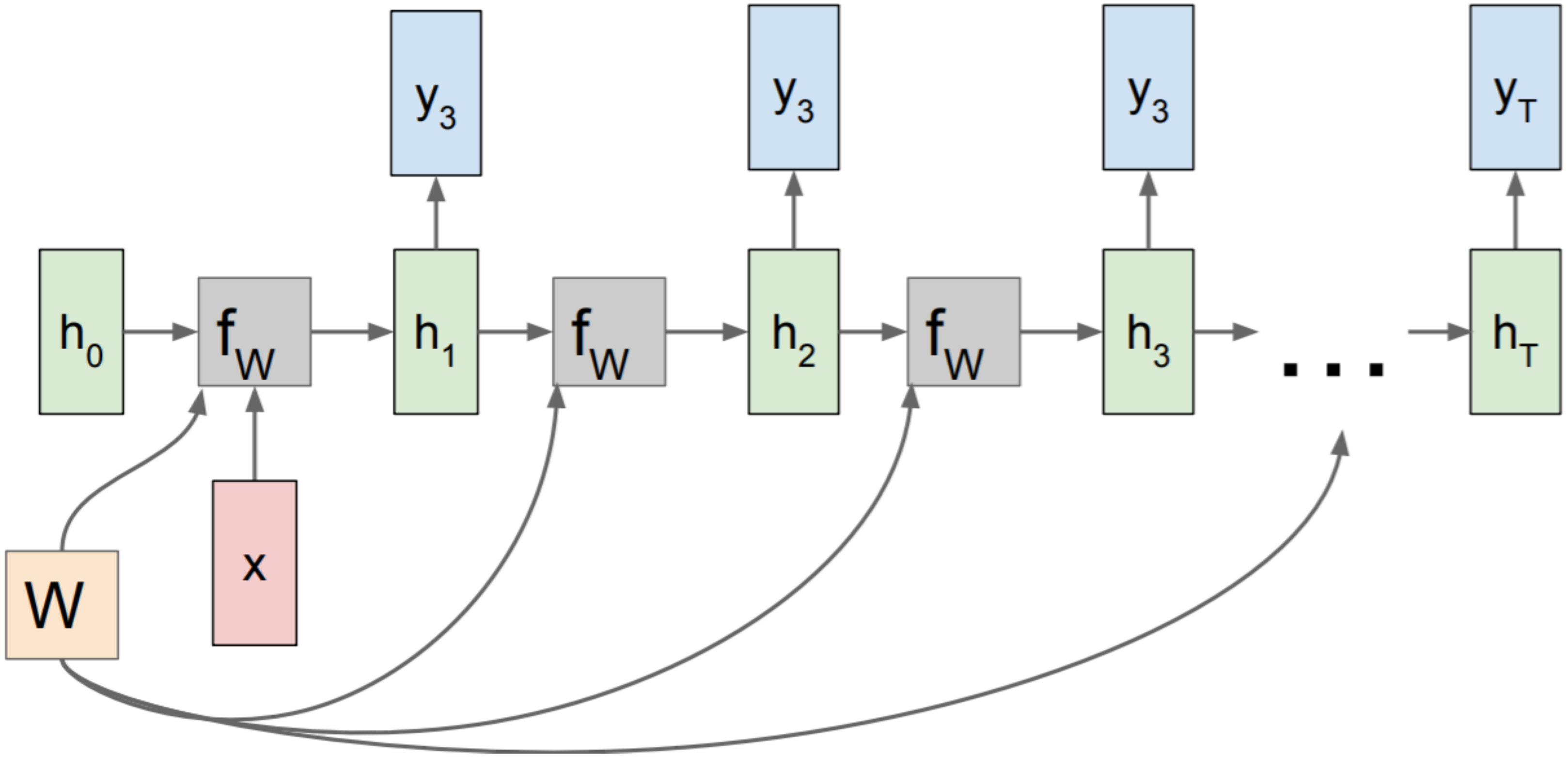

$$\begin{tikzcd}
h_0 \arrow[r] & f_W \arrow[r] & h_1 \arrow[r] & f_W \arrow[r] & h_2 \arrow[r] & f_W \arrow[r] & h_3 \arrow[r] & \ldots \arrow[r] & h_T \\
 & x_1 \arrow[u] & & x_2 \arrow[u] & & x_3 \arrow[u] & & &
\end{tikzcd}$$

RNN: Computational Graph

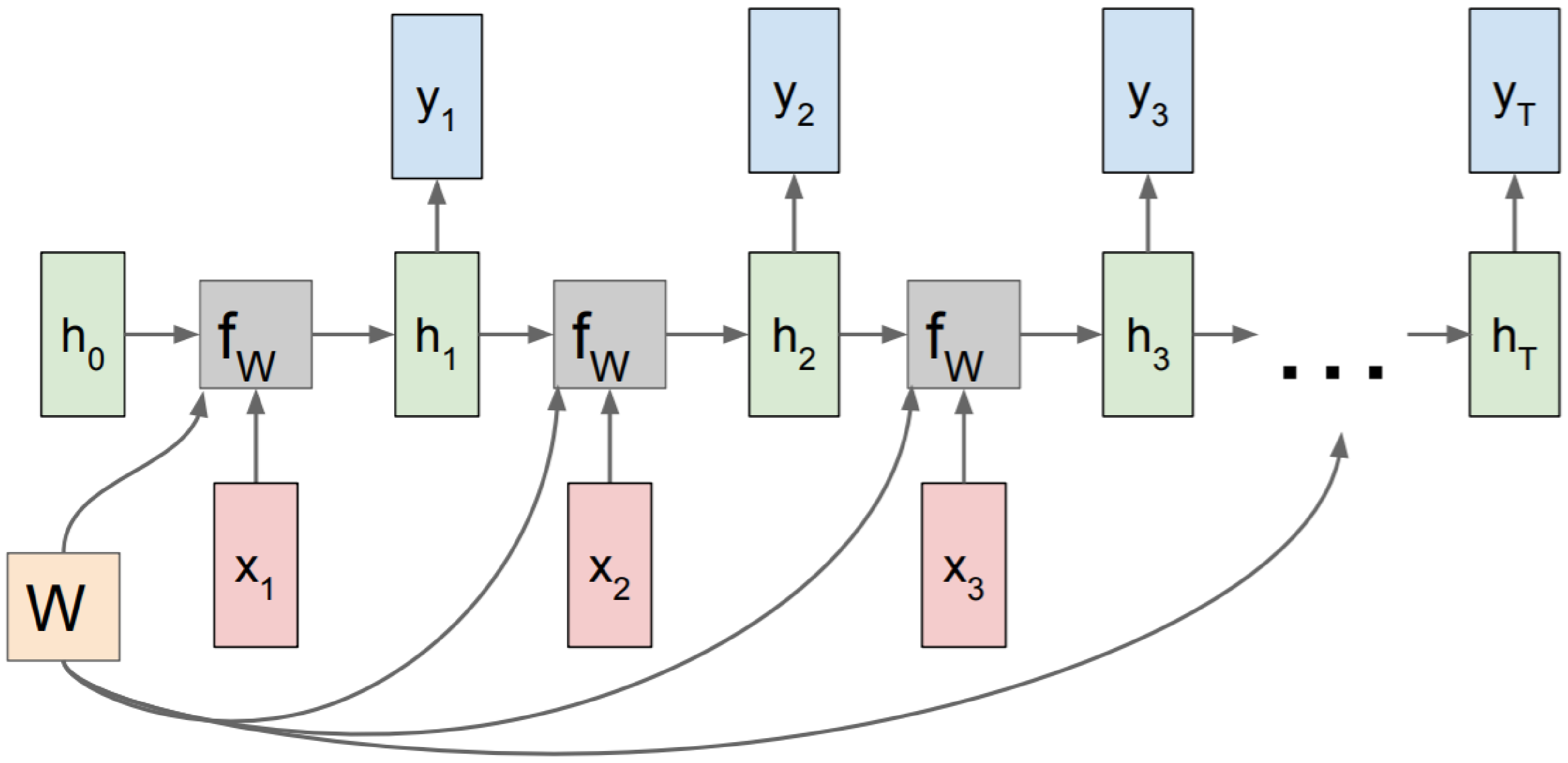
Re-use the same weight matrix at every time-step



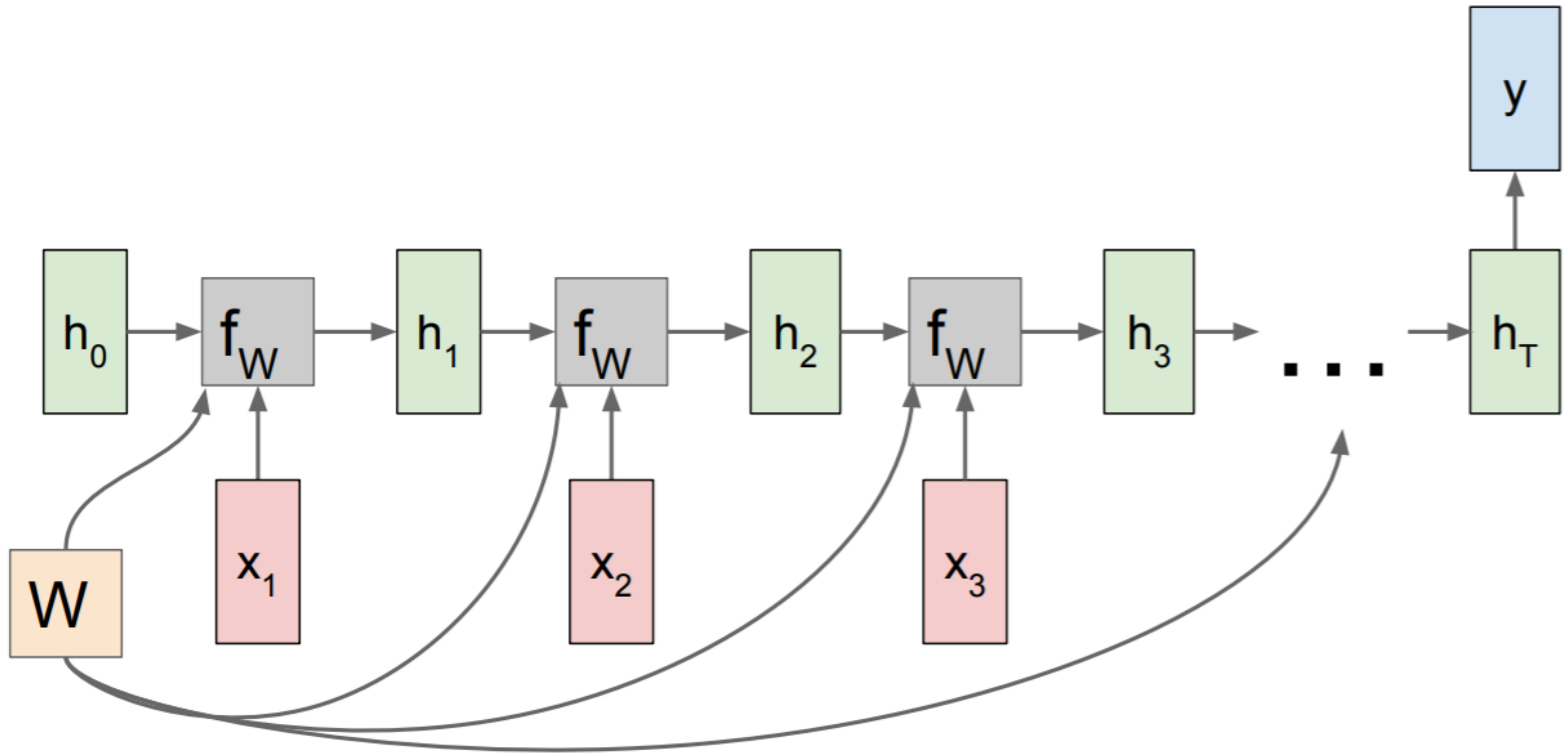
One-to-Many



Many-to-Many



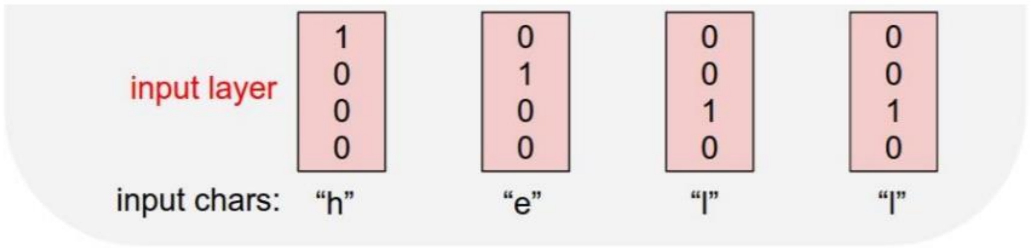
Many-to-One



Example: Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



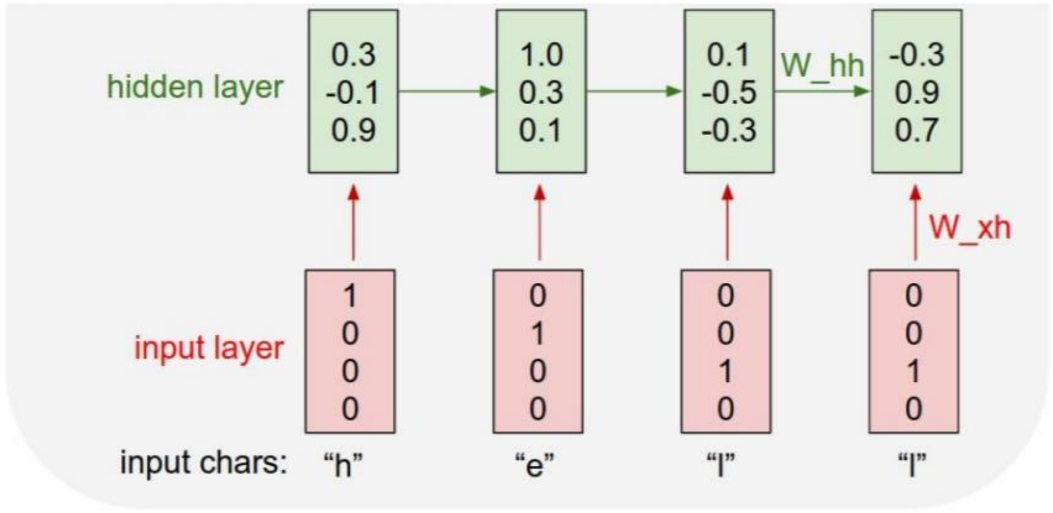
Example: Language Model

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”**

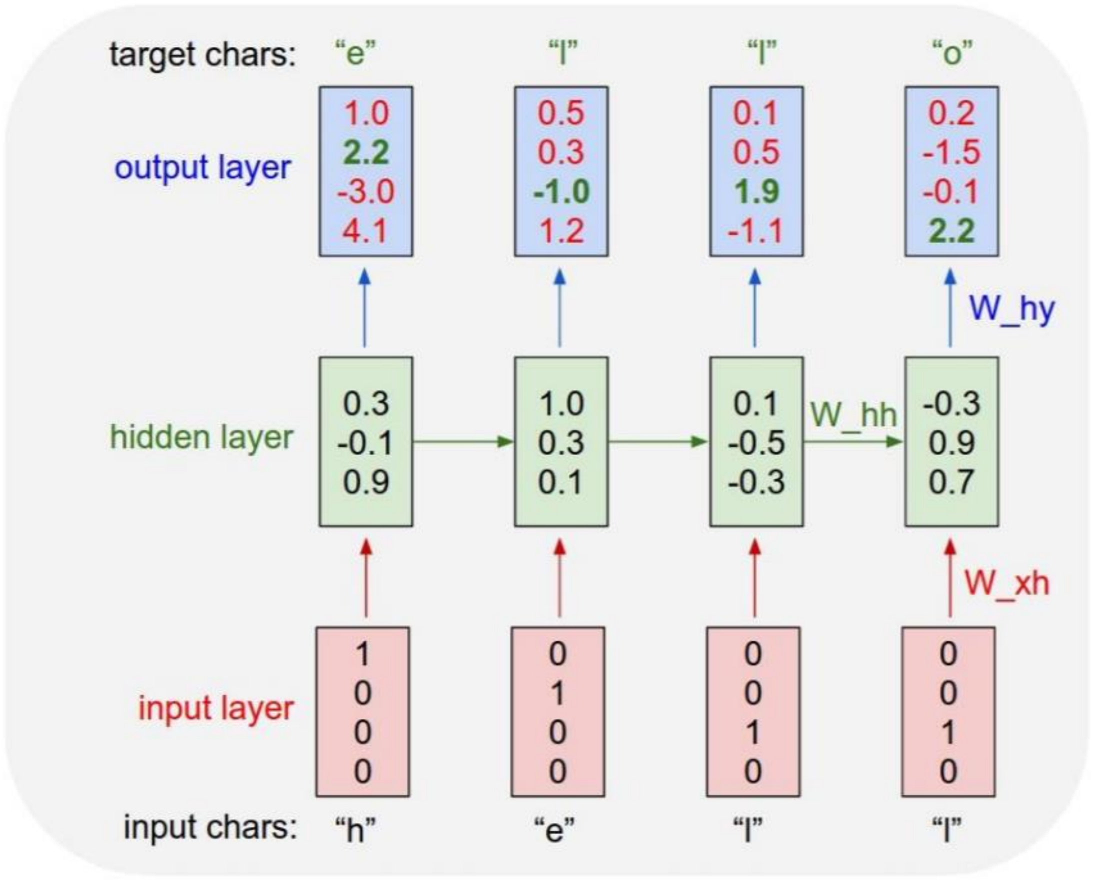


Example: Language Model

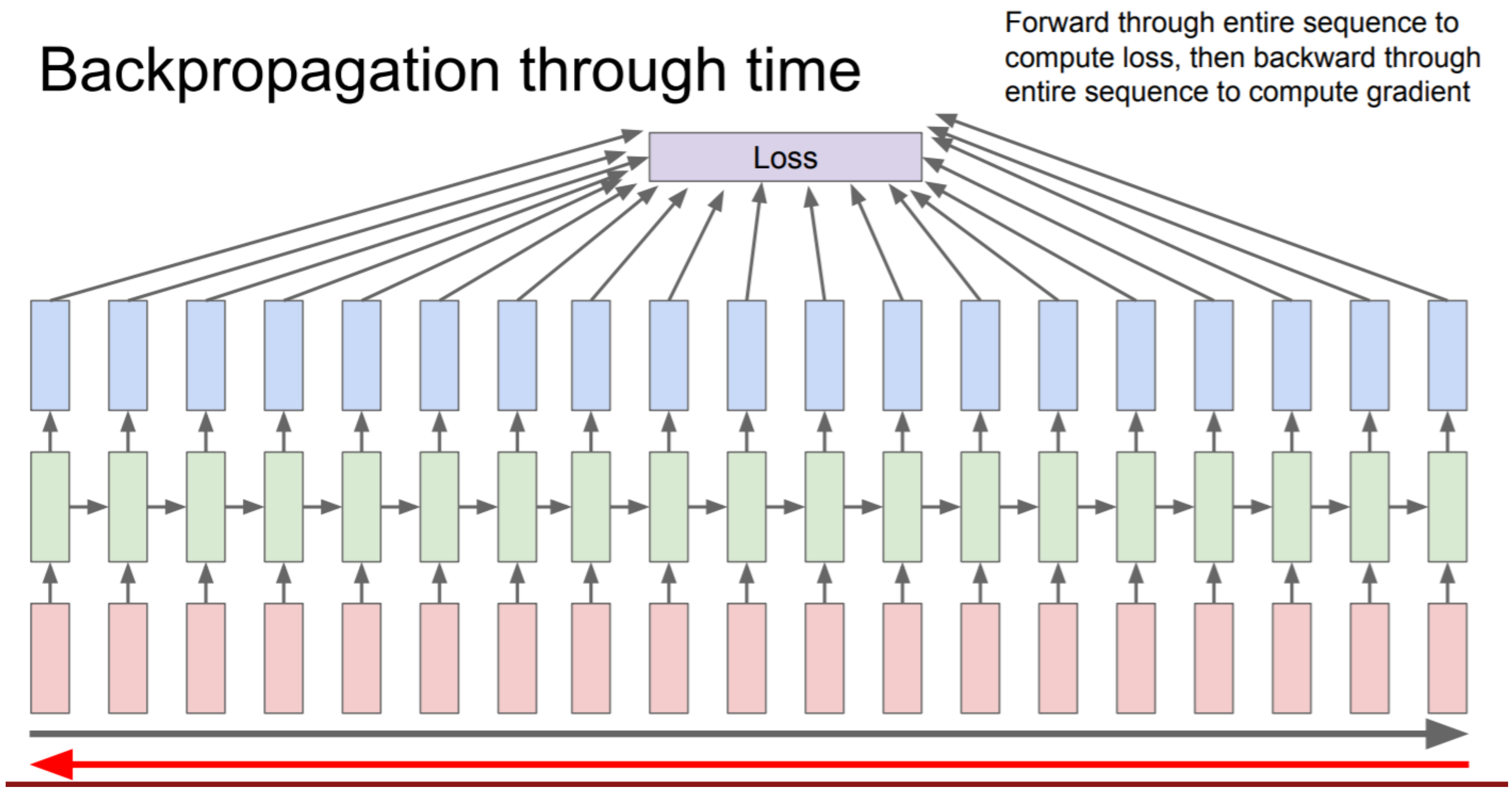
Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**

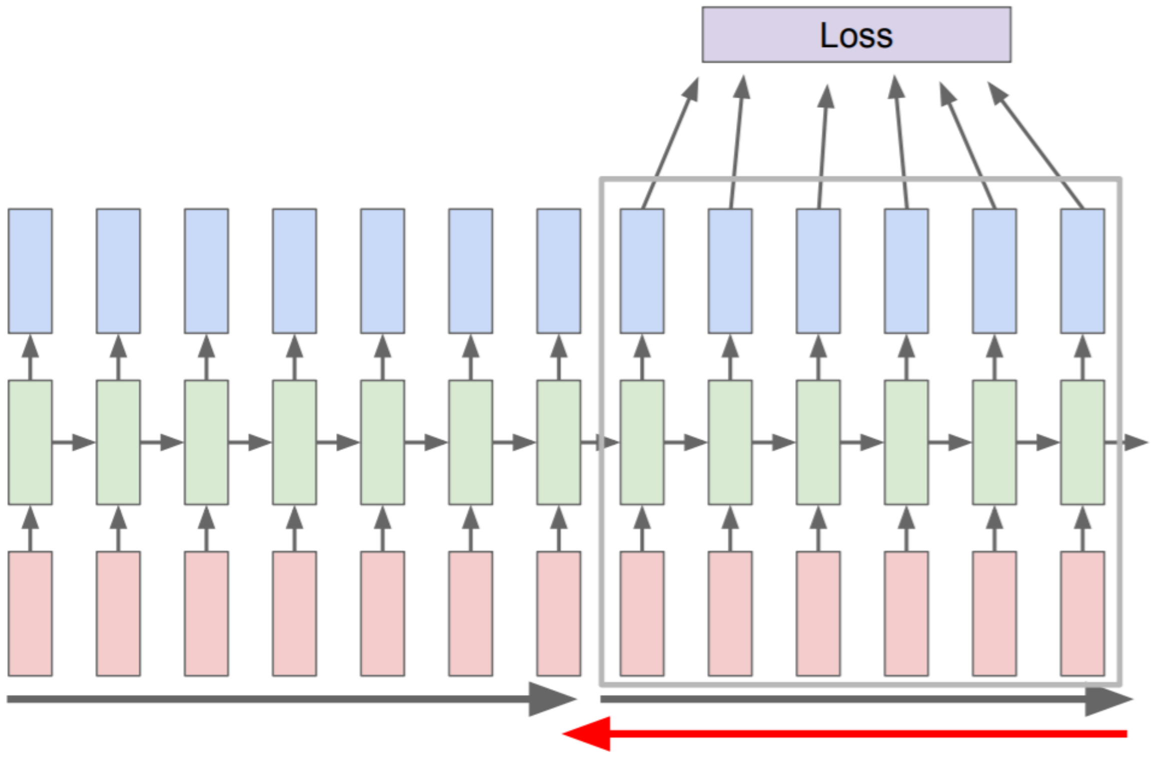


Training RNNs


Backpropagation through time
Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient
Loss

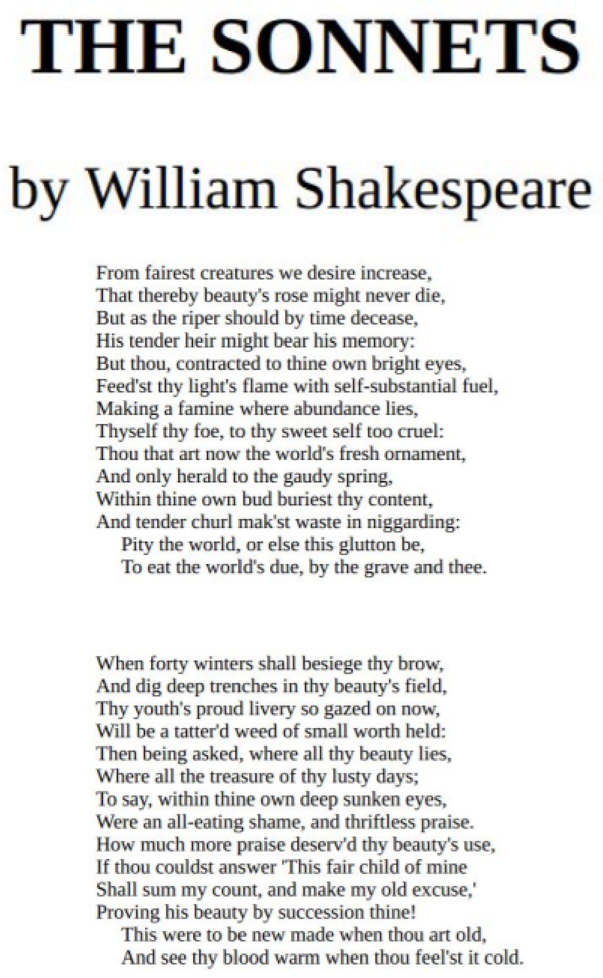
Training RNNs

Truncated Backpropagation through time



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

Writing poetry

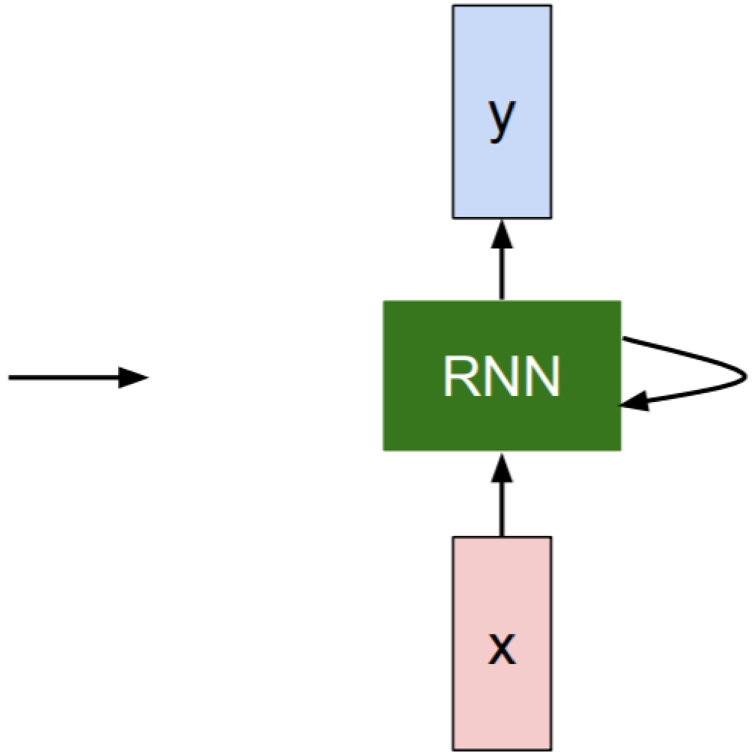


THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.



Writing poetry

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

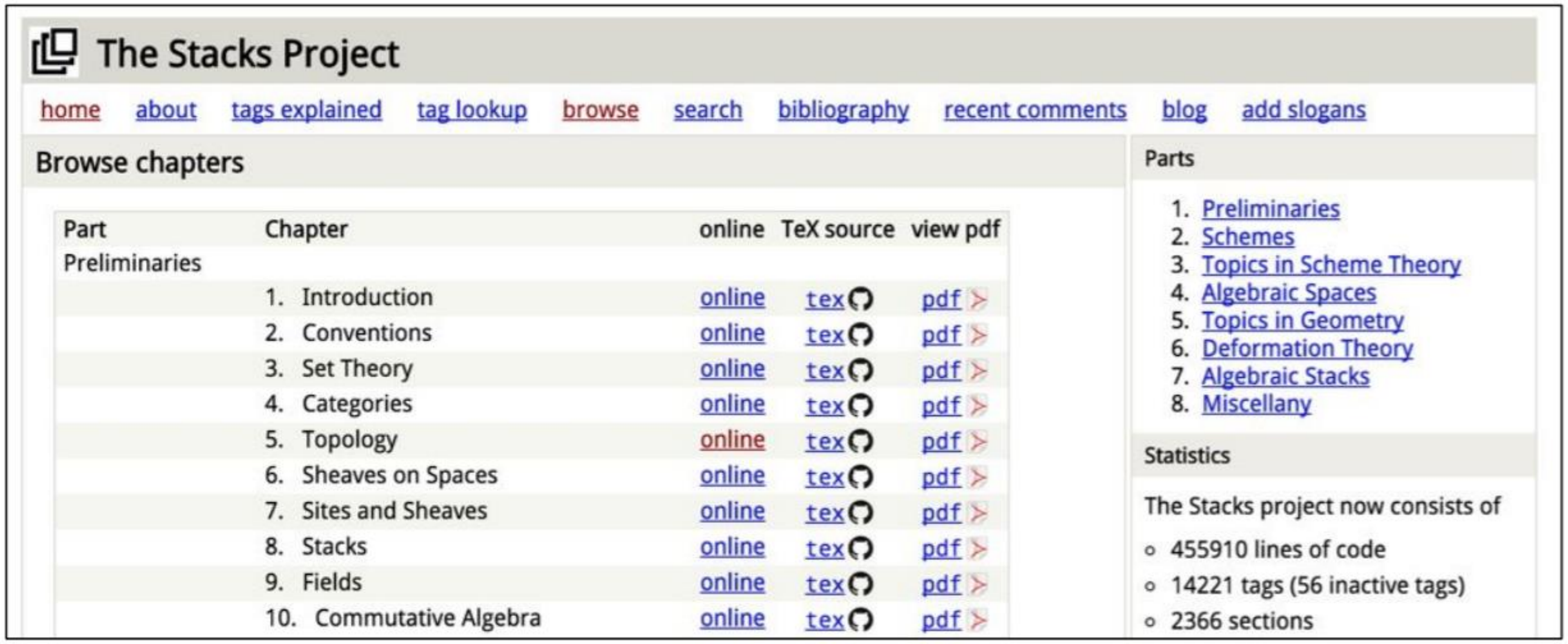
```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

↓ 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.
```

Writing geometry proofs

The Stacks Project: open source algebraic geometry textbook



The Stacks Project

home about tags explained tag lookup browse search bibliography recent comments blog add slogans

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Parts

1. Preliminaries
2. Schemes
3. Topics in Scheme Theory
4. Algebraic Spaces
5. Topics in Geometry
6. Deformation Theory
7. Algebraic Stacks
8. Miscellany

Statistics

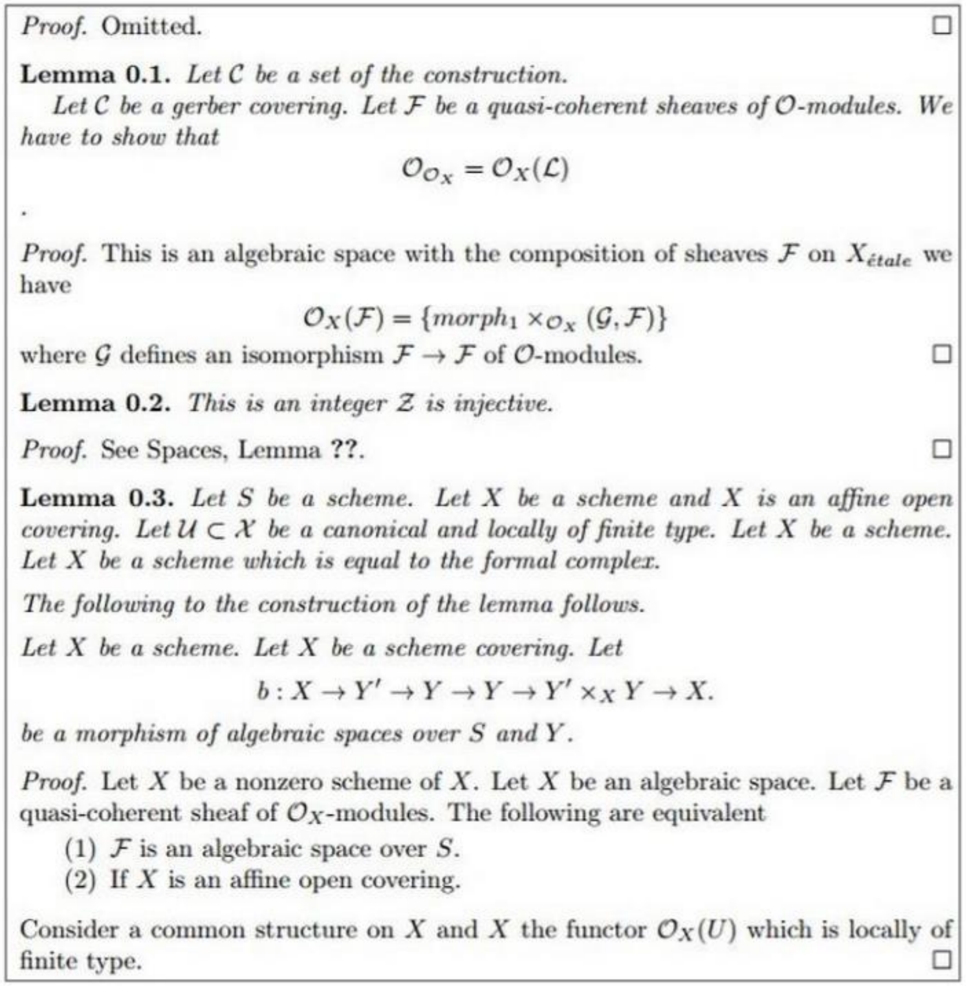
The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

http://stacks.math.columbia.edu/

Writing geometry proofs



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

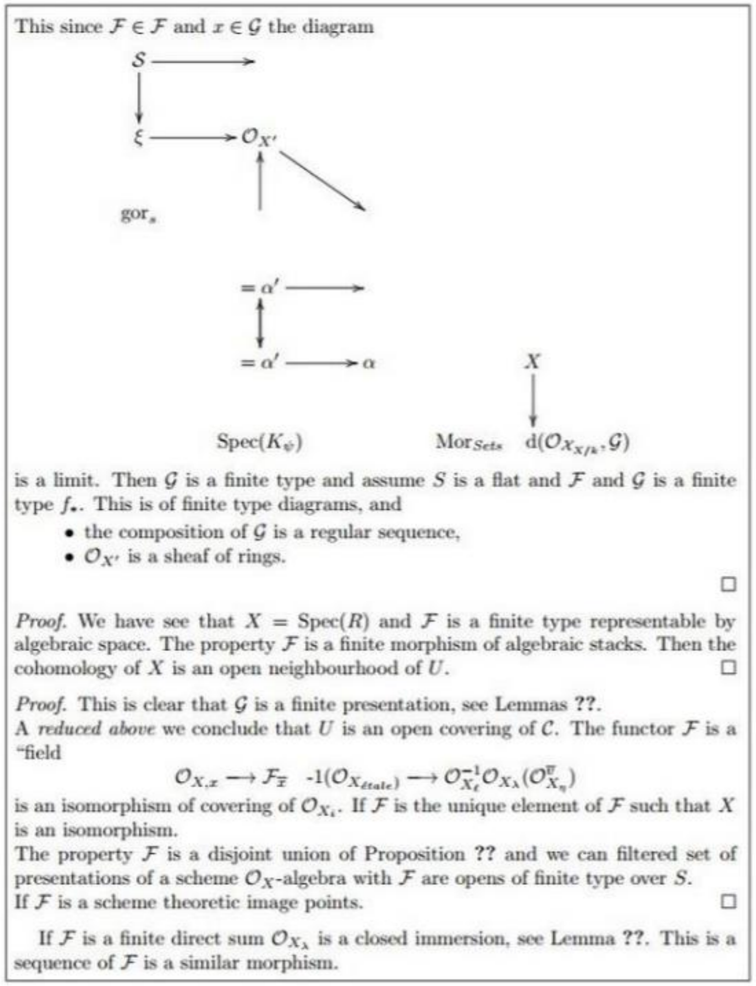
$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

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. □



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

$$\begin{tikzcd}
\mathcal{S} \arrow[r] \arrow[d] & {} & {} & {} \\
\xi \arrow[r] & \mathcal{O}_{X'} \arrow[dr] & {} & {} \\
\mathrm{gor}_* & {} \arrow[u] & {} & {} \\
{} & =\alpha' \arrow[r] \arrow[d, leftrightarrow] & {} & {} \\
{} & =\alpha' \arrow[r] & \alpha & X \arrow[d] \\
{} & \mathrm{Spec}(K_\psi) & \mathrm{Mor}_{Sets} & \mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G})
\end{tikzcd}$$

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.

□

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. □

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

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

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. □

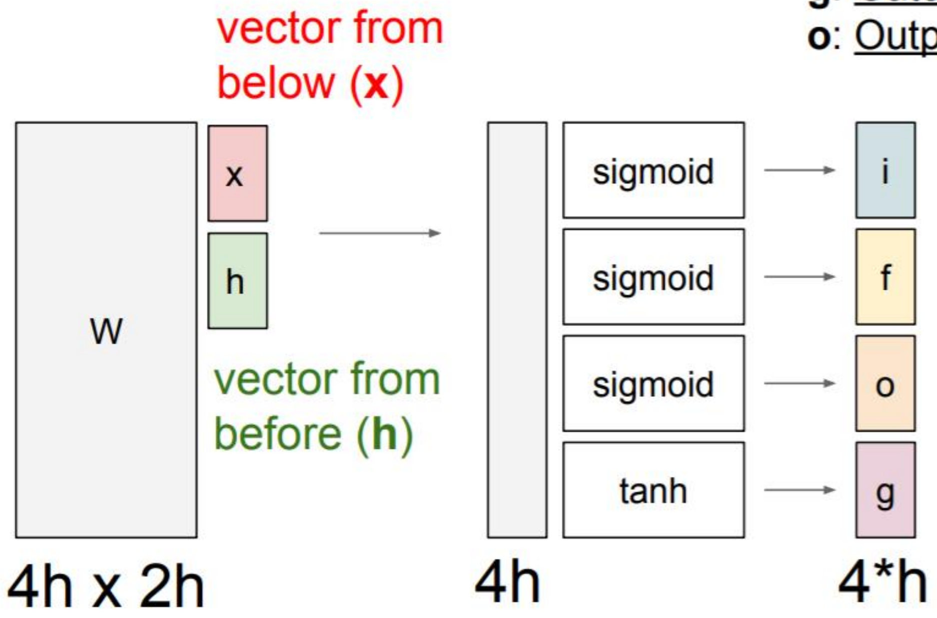
If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

Example RNN: LSTM

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

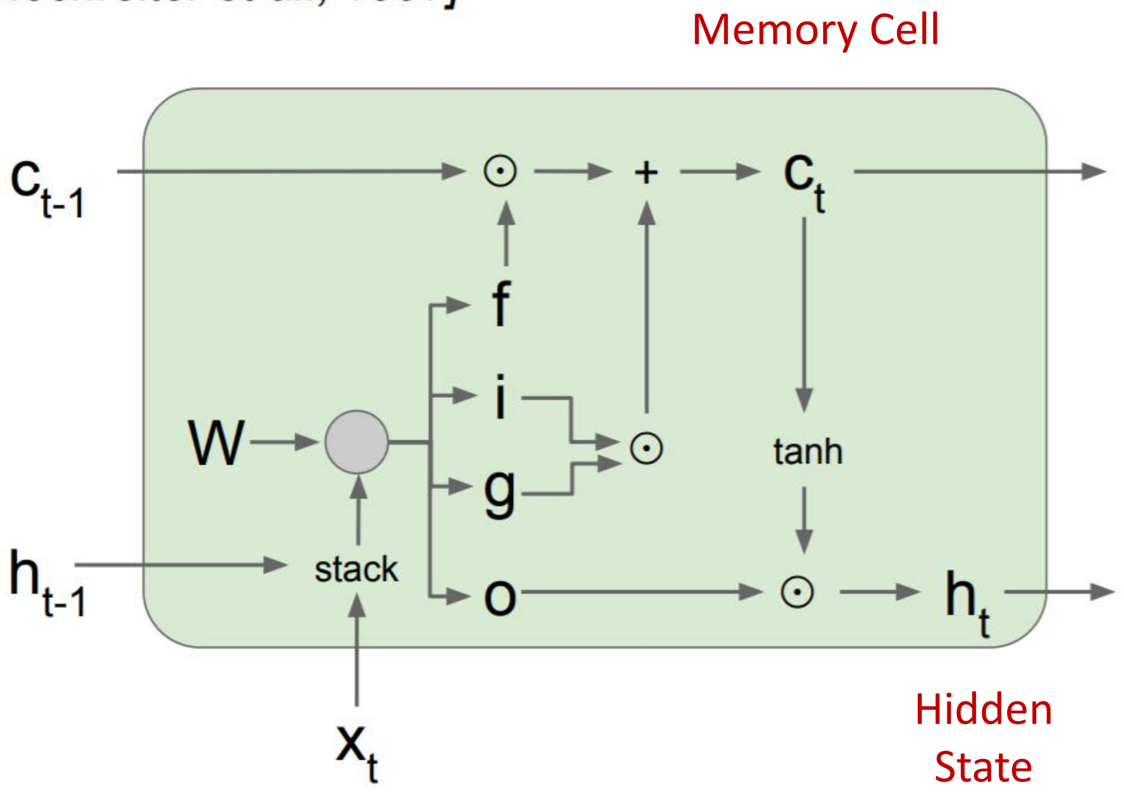
$$h_t = o \odot \tanh(c_t)$$

Capture long-term dependencies by using "memory cells"

LSTM

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Summary RNNs

- RNNS maintain state and have flexible design
  - One-to-many, many-to-one, many-to-many
- Applicable to sequential data
- LSTM is one example RNN architecture
- Better and simpler architectures are a topic of active research

Unsupervised Learning

- Supervised learning used labeled data pairs $(\mathbf{x}, \mathrm{y})$ to learn a function $f : \mathrm{X} \rightarrow \mathrm{Y}$
  - But, what if we don't have labels?

- No labels = **unsupervised learning**
- Only some points are labeled = **semi-supervised learning**
  - Labels may be expensive to obtain, so we only get a few

Unsupervised Learning

- Different learning tasks
- Dimensionality reduction
  - Project the data to lower dimensional space
  - Example: PCA (Principal Component Analysis)
- Feature learning
  - Find feature representations
  - Example: Autoencoders
- Clustering
  - Group similar data points into clusters
  - Example: k-means, hierarchical clustering

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Standard metrics for evaluation

Unsupervised Learning

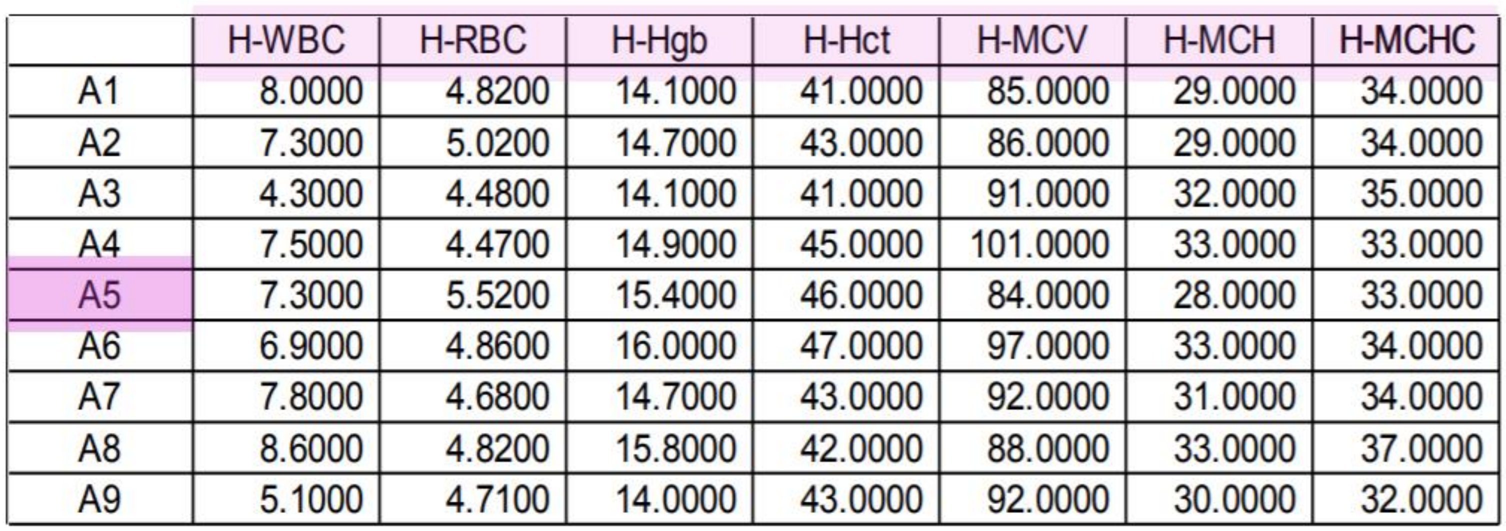
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Difficult to evaluate

How Can we Visualize High-Dimensional Data?



	H-WBC	H-RBC	H-Hgb	H-Hct	H-MCV	H-MCH	H-MCHC
A1	8.0000	4.8200	14.1000	41.0000	85.0000	29.0000	34.0000
A2	7.3000	5.0200	14.7000	43.0000	86.0000	29.0000	34.0000
A3	4.3000	4.4800	14.1000	41.0000	91.0000	32.0000	35.0000
A4	7.5000	4.4700	14.9000	45.0000	101.0000	33.0000	33.0000
A5	7.3000	5.5200	15.4000	46.0000	84.0000	28.0000	33.0000
A6	6.9000	4.8600	16.0000	47.0000	97.0000	33.0000	34.0000
A7	7.8000	4.6800	14.7000	43.0000	92.0000	31.0000	34.0000
A8	8.6000	4.8200	15.8000	42.0000	88.0000	33.0000	37.0000
A9	5.1000	4.7100	14.0000	43.0000	92.0000	30.0000	32.0000

Instances

Features

Difficult to see the correlations between the features...

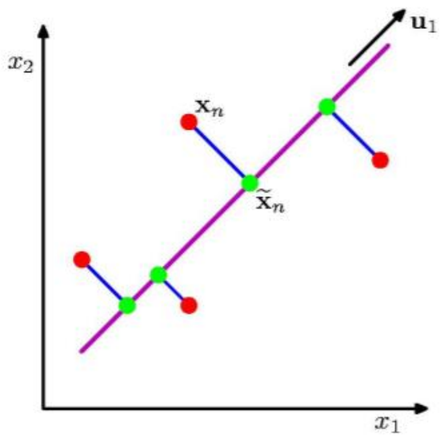
Data Visualization

- Is there a representation better than the raw features?
    - Is it really necessary to show all the 53 dimensions?
    - … what if there are strong correlations between the features?

Could we find the *smallest* subspace of the 53-D space that keeps the *most information* about the original data?

One solution: **Principal Component Analysis**

Principal Component Analysis



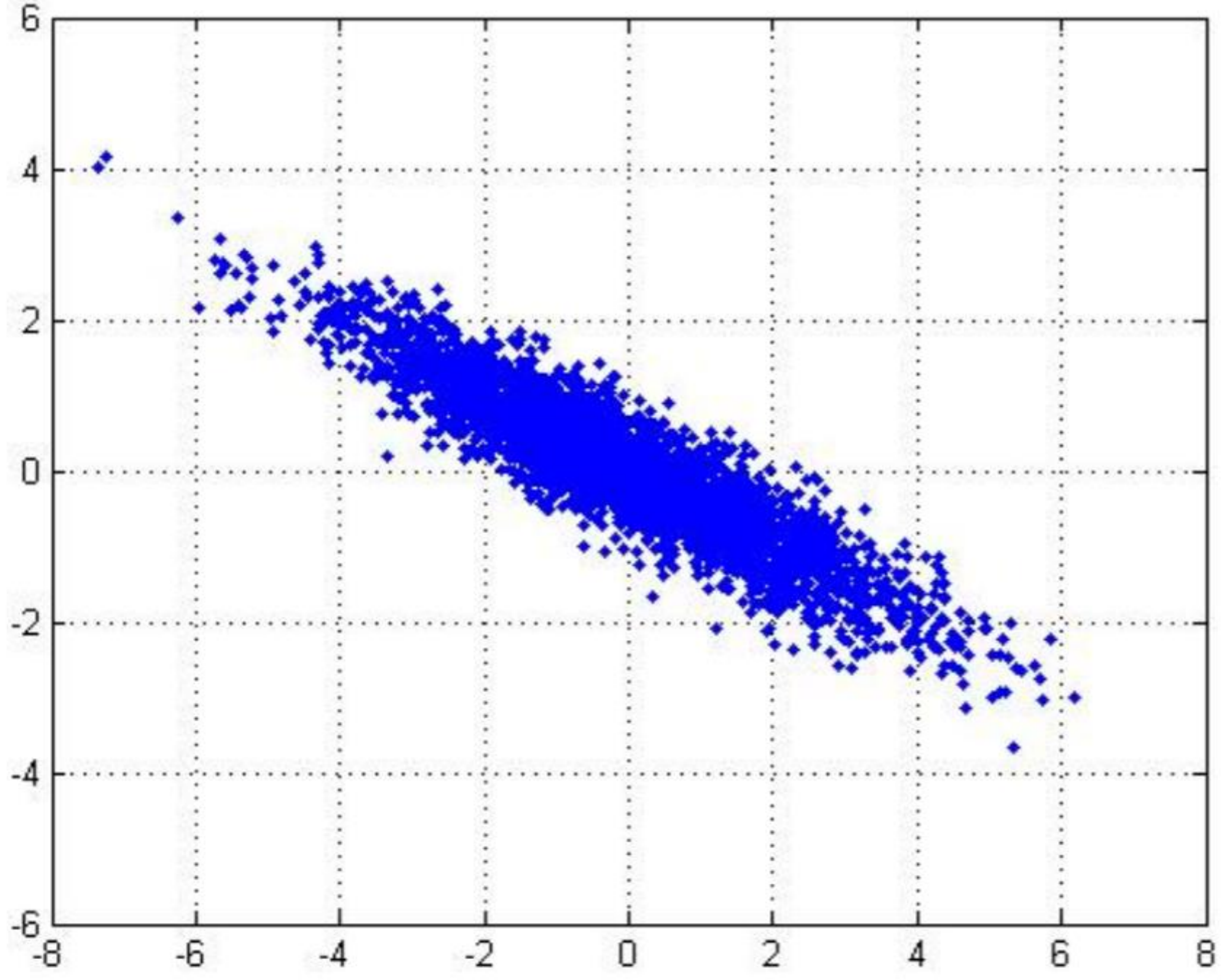
Orthogonal projection of data onto lower-dimension linear space that...

- maximizes variance of projected data (purple line)
- minimizes mean squared distance between data point and projections (sum of blue lines)

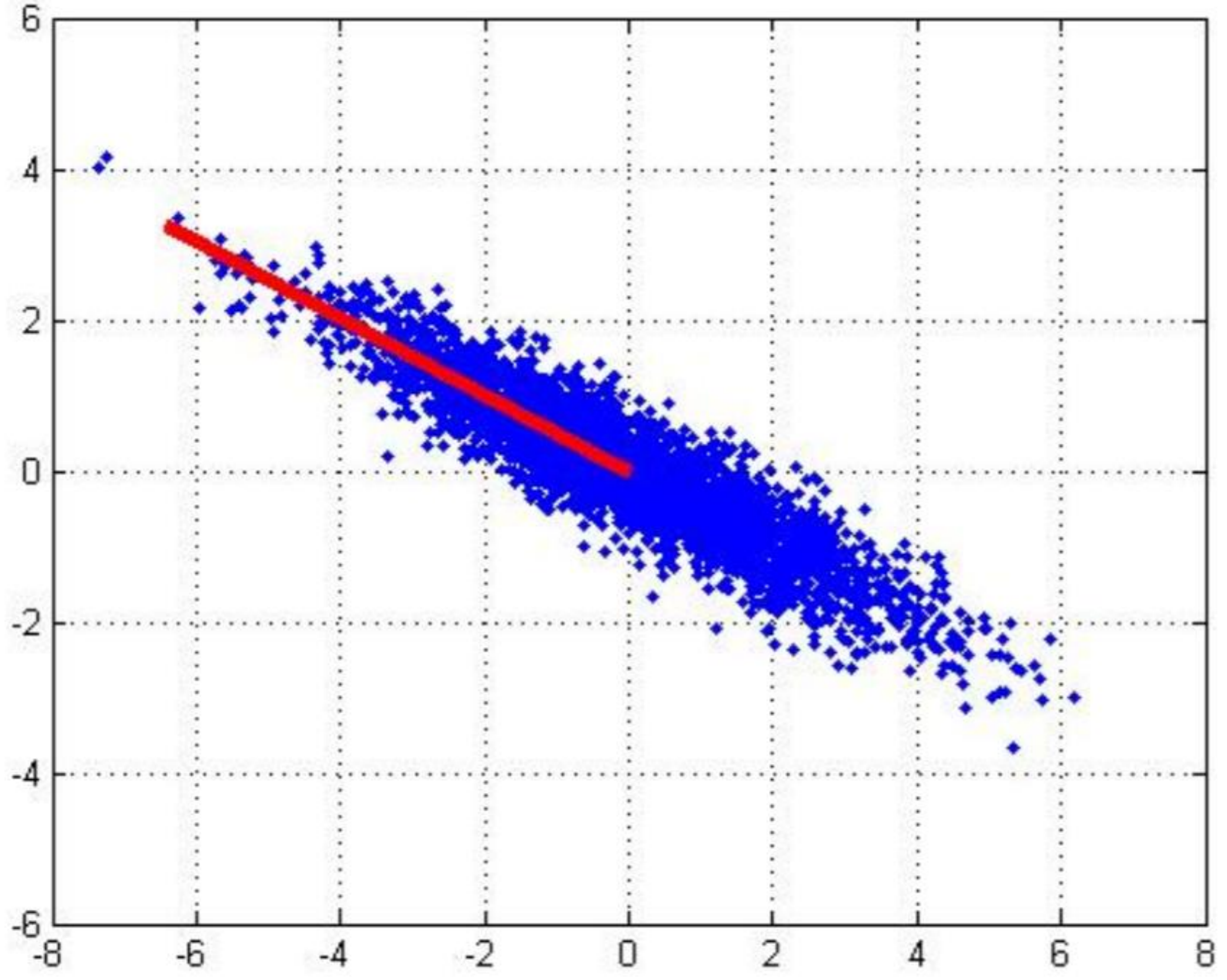
The Principal Components

- **Vectors** originating from the center of mass
- Principal component #1 points in the direction of the **largest variance**
- Each subsequent principal component…
  - is **orthogonal** to the previous ones, and
  - points in the directions of the **largest variance of the residual subspace**

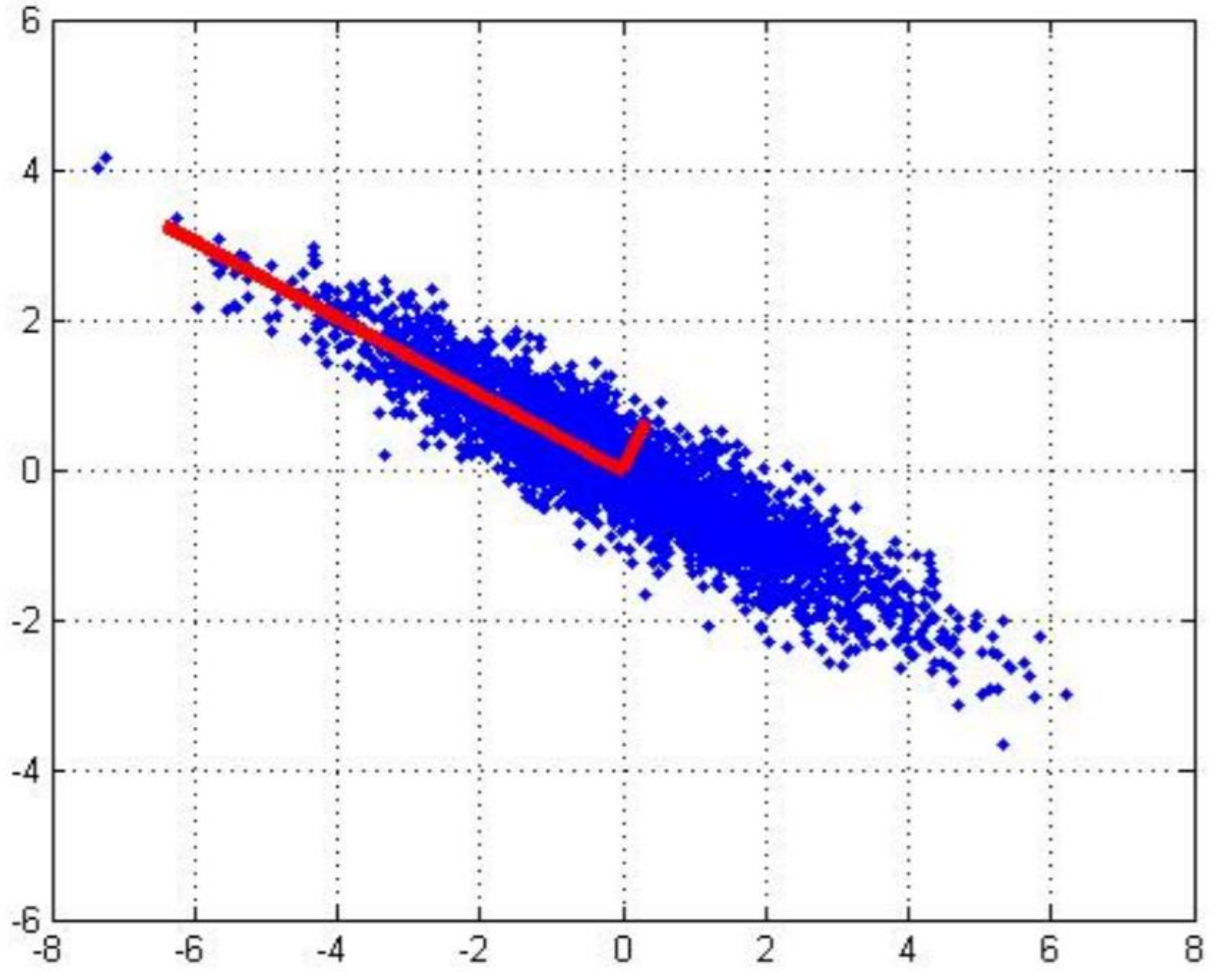
2D Gaussian Data



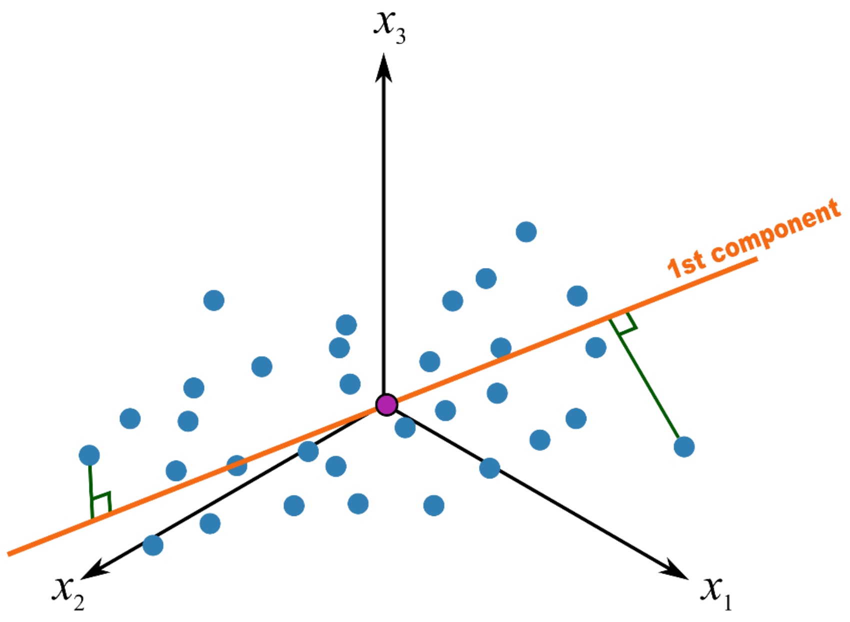
1st PCA Axis

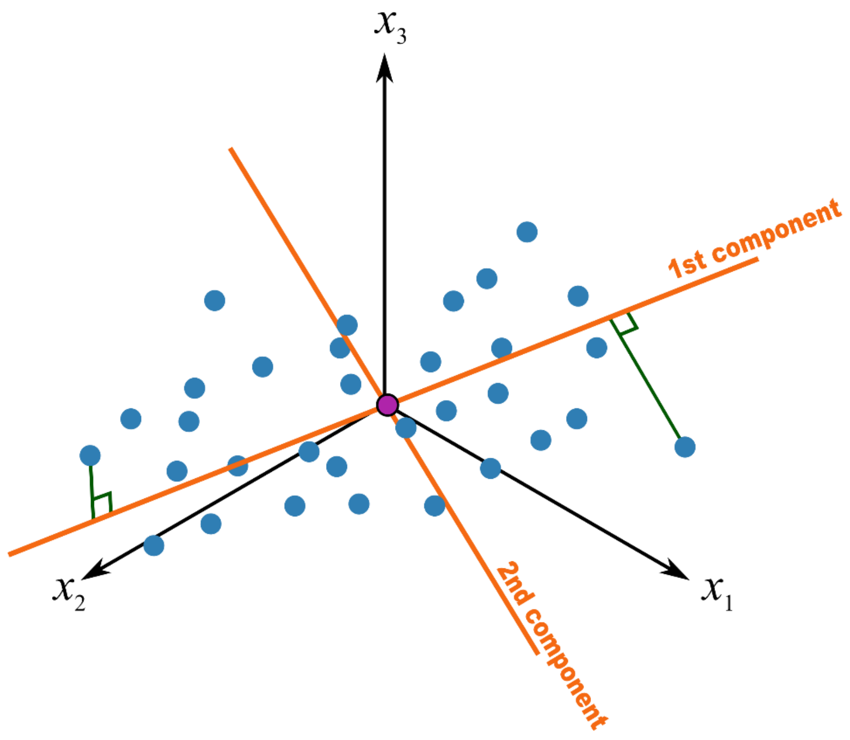


2nd PCA Axis



PCA Example



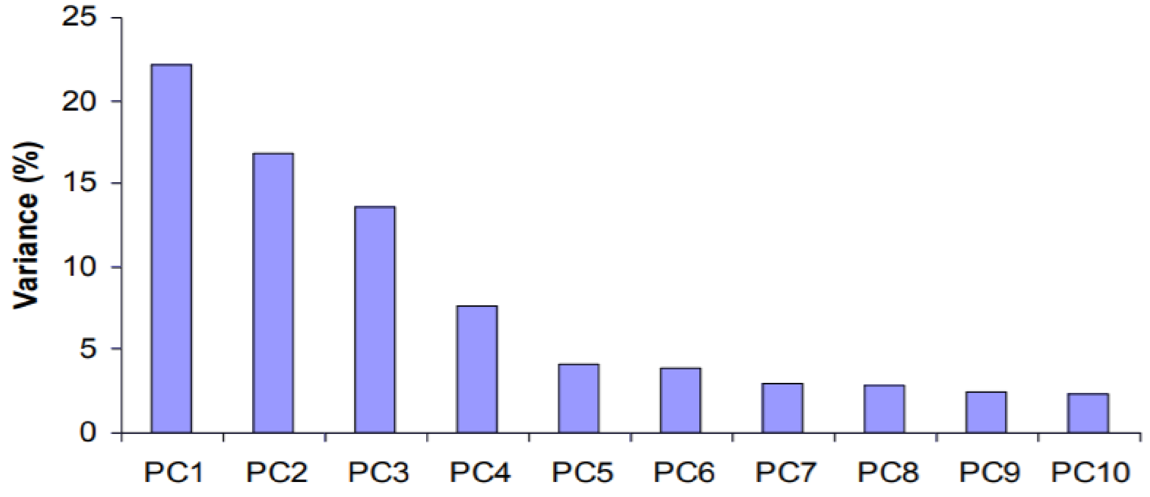


Eigenvectors

- Let A be a matrix and consider $Ax = \lambda x$
- Terminology
  - Eigenvalue of matrix: λ
  - Eigenvector: x for which $Ax = \lambda x$
- Connection to PCA
  - First principal component is largest eigenvector of covariance matrix $\Sigma = X^T X$
  - Second principal component is orthogonal to first and second eigenvector of Σ

Dimensionality Reduction

Can *ignore* the components of lesser significance



You do lose some information, but if the eigenvalues are small, you don't lose much

- choose only the first k eigenvectors, based on their eigenvalues
- final data set has only k dimensions

PCA Algorithm

- Given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, compute covariance matrix $\boldsymbol{\Sigma}$
  - X is the n x d data matrix
  - Compute data mean (average over all rows of X)
  - Subtract mean from each row of X (centering the data)
  - Compute covariance matrix $\boldsymbol{\Sigma} = \mathrm{X}^\mathsf{T}\mathrm{X}$ ($\boldsymbol{\Sigma}$ is d x d)
- **PCA** basis vectors are given by the eigenvectors of $\boldsymbol{\Sigma}$
  - $\mathrm{Q}, \Lambda =$ numpy.linalg.eig(Σ)
  - $\{\mathbf{q}_i, \lambda_i\}_{i=1..n}$ are the eigenvectors/eigenvalues of $\boldsymbol{\Sigma}$
    ... $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$
- Larger eigenvalue $\Rightarrow$ more important eigenvectors

PCA

$$X = \begin{bmatrix} 0\,1\,0\,1\,1\,0\,0\,1\ldots \\ 1\,1\,0\,1\,1\,1\,0\,0\ldots \\ 0\,0\,1\,1\,1\,0\,0\,0\ldots \\ \vdots \\ 1\,0\,1\,0\,1\,0\,0\,0\ldots \end{bmatrix}$$

X has d columns

Q is the eigenvectors of Σ;
columns are ordered by importance!

Q is d x d

$$Q = \begin{bmatrix} 0.34 & 0.23 & -0.30 & -0.23 & \ldots \\ 0.04 & 0.13 & -0.40 & 0.21 & \ldots \\ -0.64 & 0.93 & 0.61 & 0.28 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ -0.20 & -0.83 & 0.78 & -0.93 & \ldots \end{bmatrix}$$

14

PCA

- Each column of Q gives weights for a linear combination of the original features

$$Q = \left[\begin{array}{ccc} \boxed{\begin{array}{r} 0.34 \\ 0.04 \\ -0.64 \\ \vdots \\ -0.20 \end{array}} & \begin{array}{r} 0.23 \\ 0.13 \\ 0.93 \\ \vdots \\ -0.83 \end{array} & \xcancel{\begin{array}{rrc} -0.30 & -0.23 & \ldots \\ -0.40 & 0.21 & \ldots \\ 0.61 & 0.28 & \ldots \\ \vdots & \vdots & \ddots \\ 0.78 & -0.93 & \ldots \end{array}} \end{array}\right]$$

$$\Uparrow$$

= 0.34 feature1 + 0.04 feature2 – 0.64 feature3 + ...

PCA

- We can apply these formulas to get the new representation for each instance $\mathbf{x}$

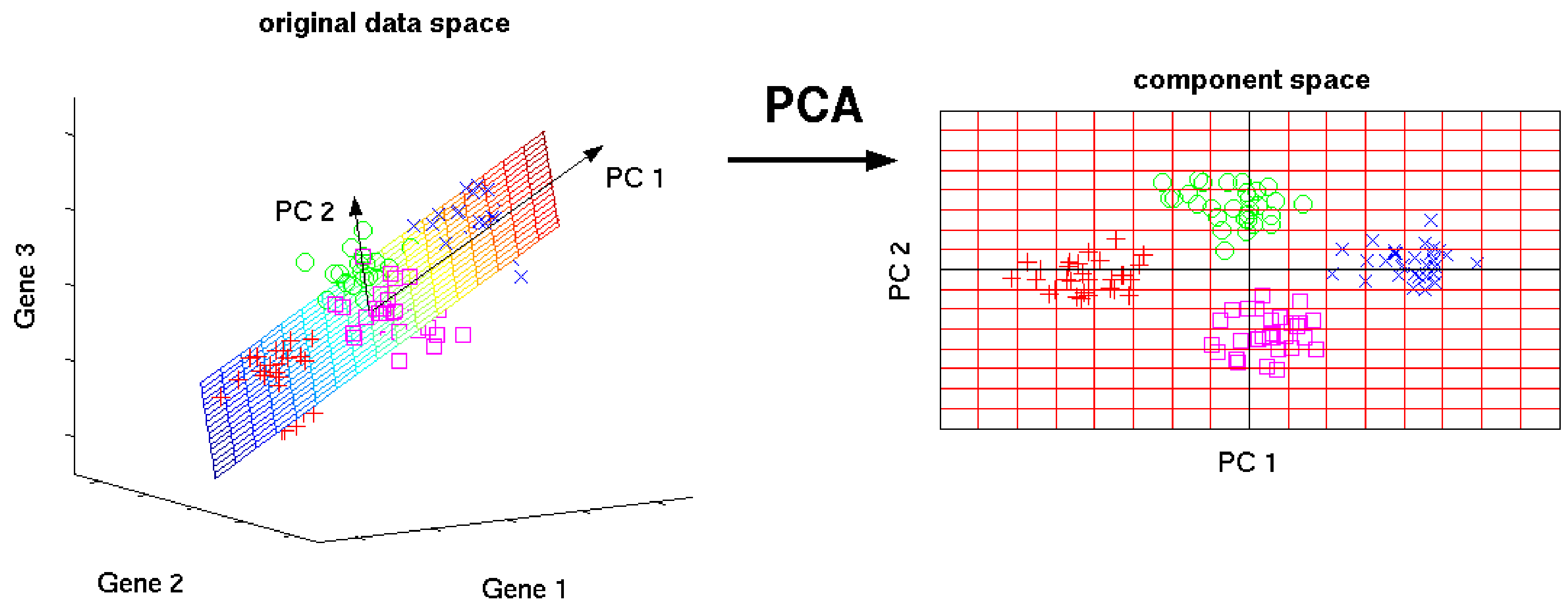
$$X = \begin{bmatrix} 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \ldots \\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \ldots \\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0 \ldots \\ \vdots \\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0 \ldots \end{bmatrix} \mathbf{x}_3 \quad \hat{Q} = \begin{bmatrix} 0.34 & 0.23 \\ 0.04 & 0.13 \\ -0.64 & 0.93 \\ \vdots & \vdots \\ -0.20 & -0.83 \end{bmatrix}$$

- The new 2D representation for $\mathbf{x}_3$ is given by:

$$\hat{x}_{31} = 0.34(0) + 0.04(0) - 0.64(1) + \ldots$$
$$\hat{x}_{32} = 0.23(0) + 0.13(0) + 0.93(1) + \ldots$$

- The re-projected data matrix is given by $\hat{\mathrm{X}} = \mathrm{X}\hat{\mathrm{Q}}$

Visualizing data



Summary: PCA

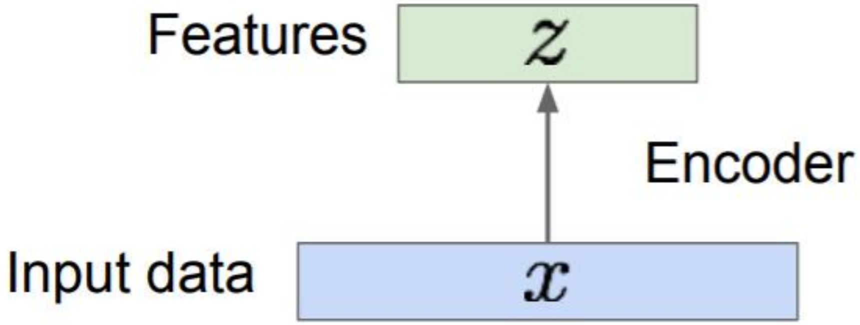
- PCA creates a lower-dimensional feature representation
  - Linear transformation
- Can be used for visualization
- Can be used with supervised on unsupervised learning
  - Very common to use classification after PCA transformation
- Main drawback
  - No interpretability of resulting features

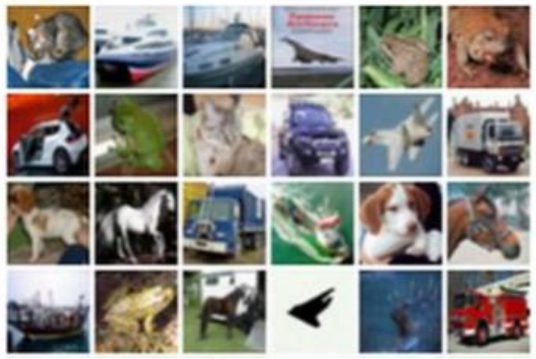
Unsupervised Learning

- Different learning tasks
- Dimensionality reduction
  - Project the data to lower dimensional space
  - Example: PCA (Principal Component Analysis)
- Feature learning
  - Find feature representations
  - Example: Autoencoders
- Clustering
  - Group similar data points into clusters
  - Example: k-means, hierarchical clustering

Autoencoders

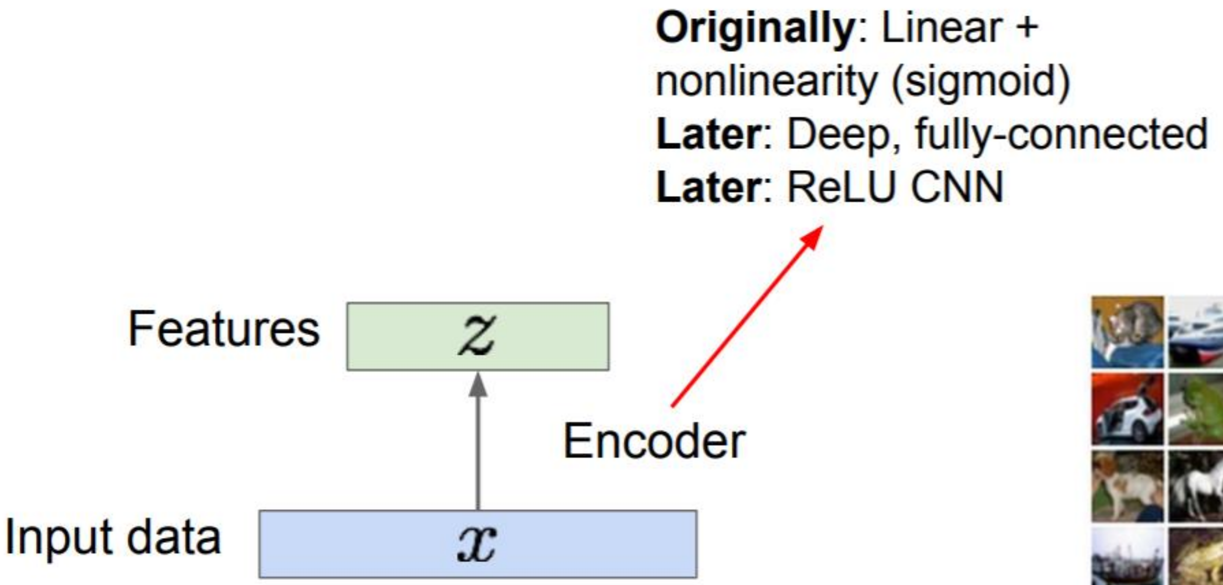
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

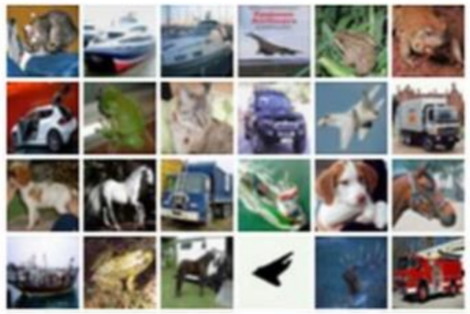




Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data


Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN
Features
z
Encoder
Input data
x



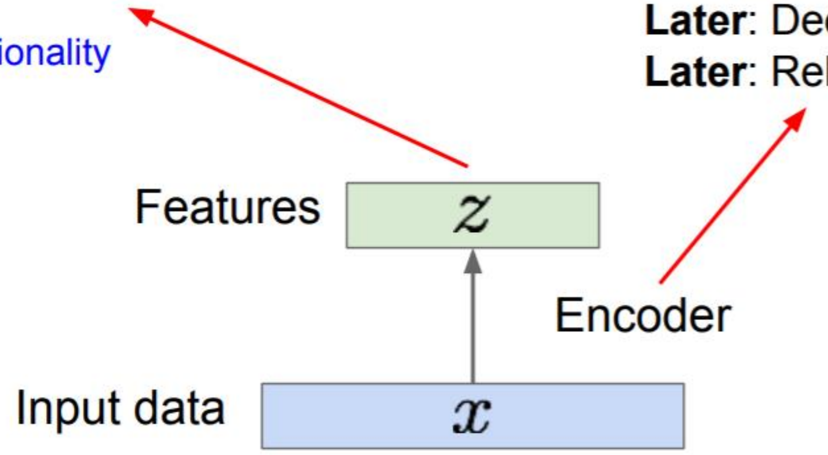
Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x** (dimensionality reduction)

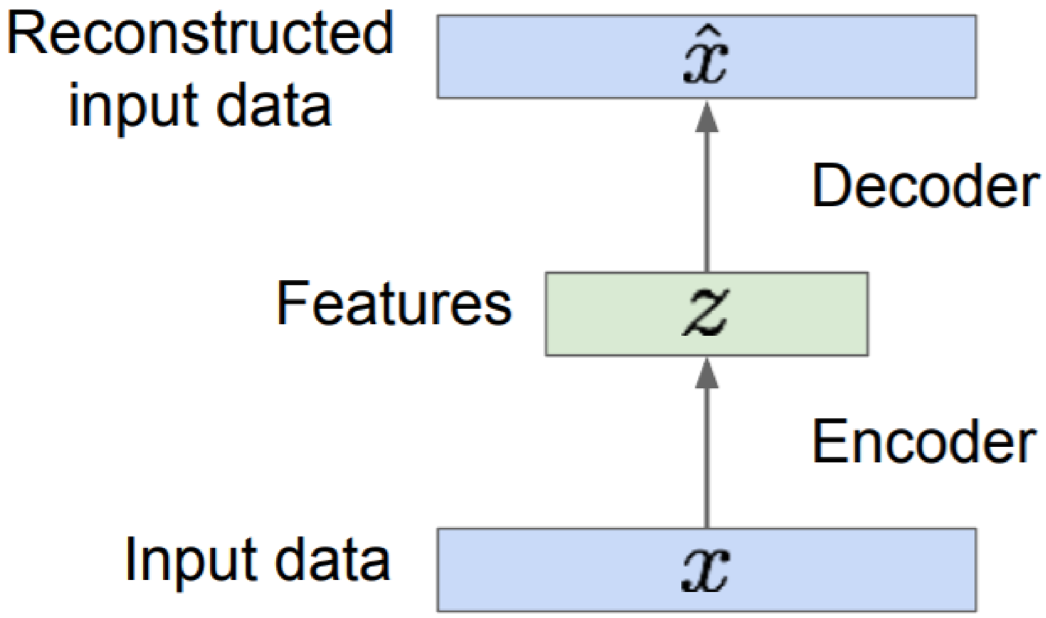
Q: Why dimensionality reduction?

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN

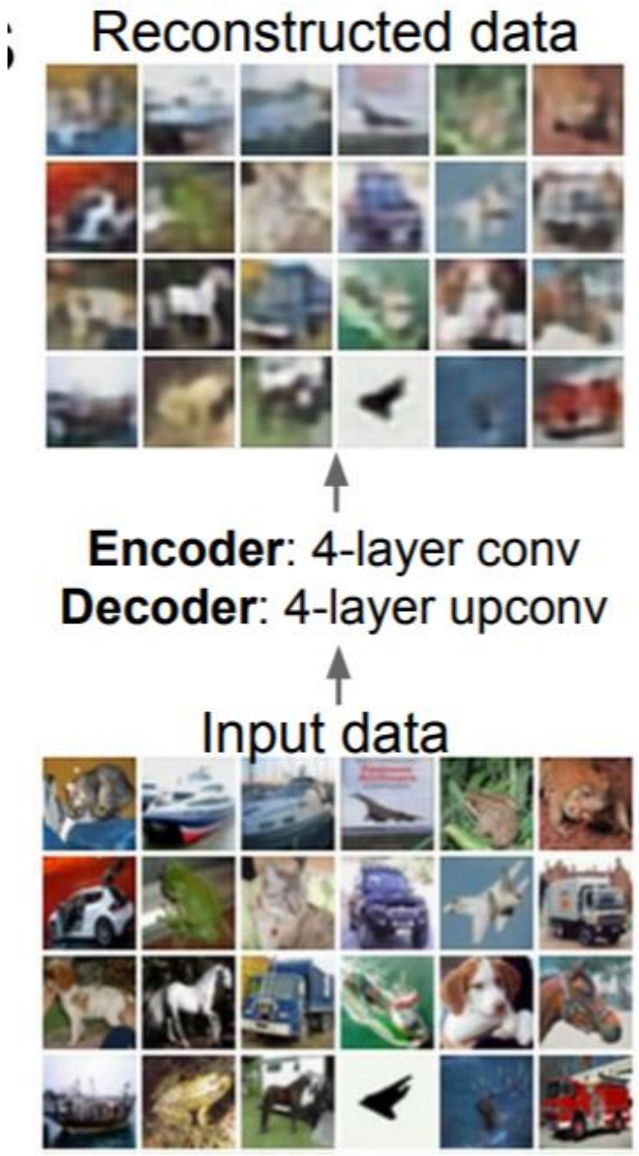


Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself



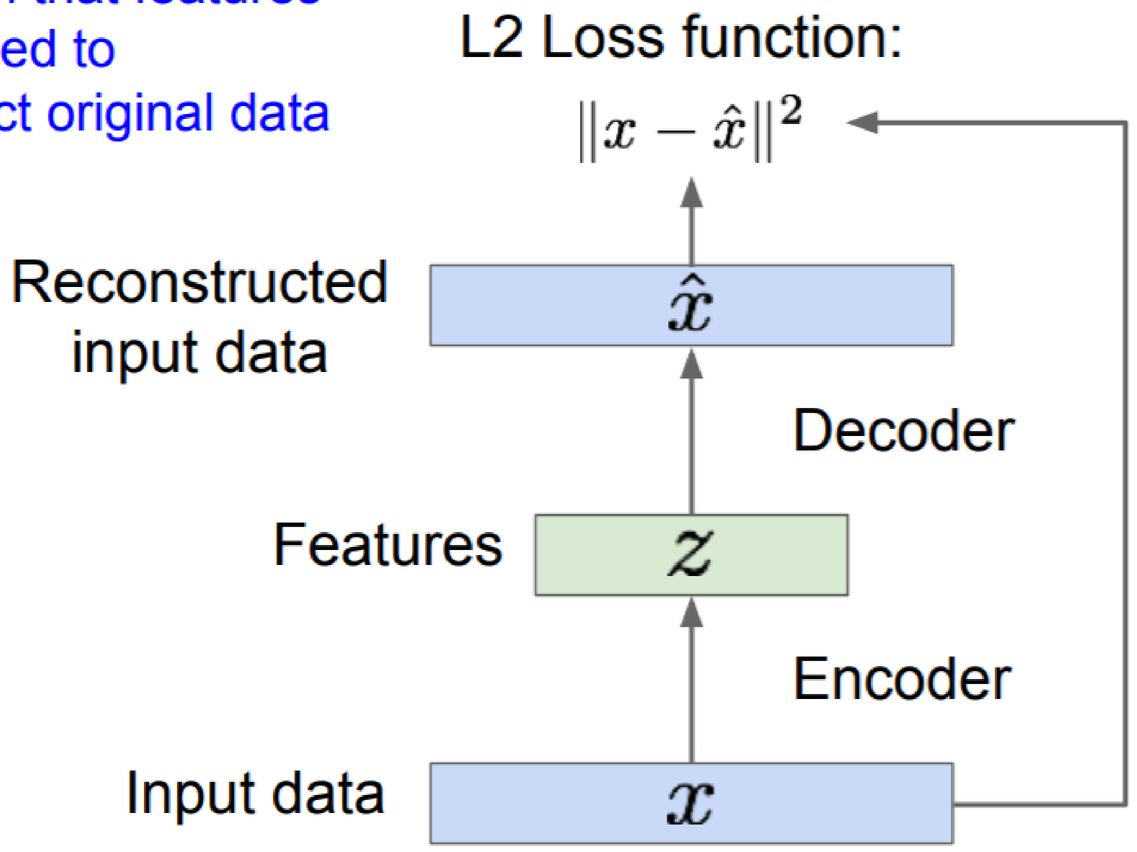
Autoencoders



Training Autoencoders

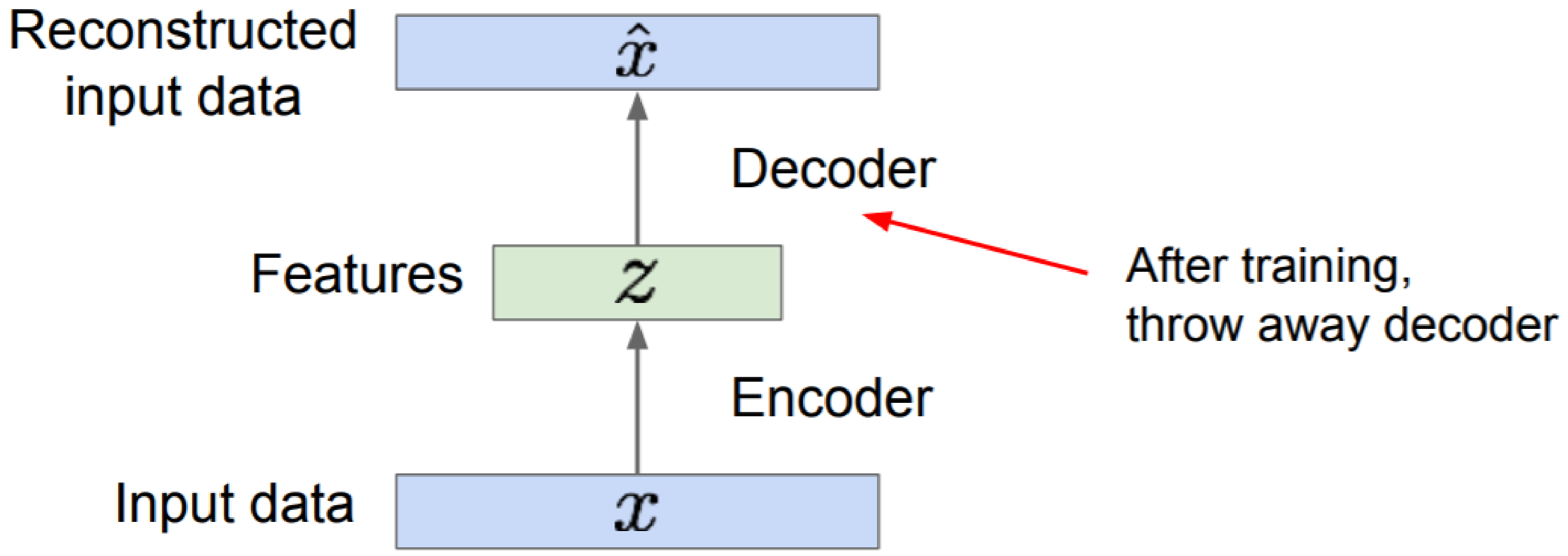
Doesn't use labels!

Train such that features can be used to reconstruct original data

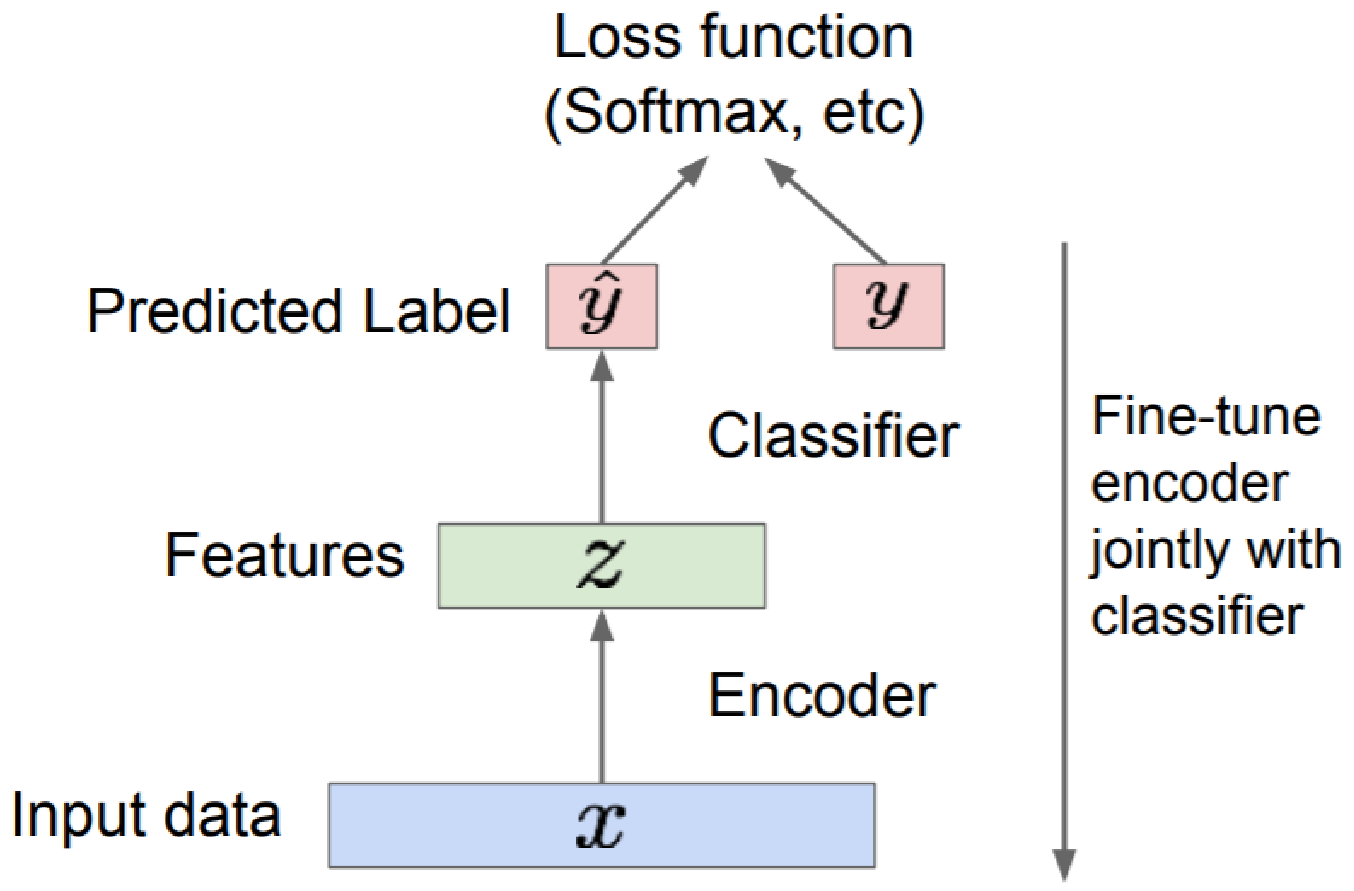


Decoders

- Decoders are only useful in training
- Reconstruct original data



Using Features for Classification



Encoder can be used to initialize a **supervised** model

Summary Autoencoders

- Autoencoders can be used to learn new features
- Applicable to sparse data
- Minimize reconstruction error by using encoder and decoder
  – Unsupervised (no labels on data)
- Encoder can be used as standalone with classification model

Acknowledgements

- Slides made using resources from:
  - Yann LeCun
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
  - Fei-Fei Li
  - Andrej Karpathy
- Thanks!