Feature Engineering

Matt Golub
Hunter Schafer

UNIVERSITY of WASHINGTON

Finish Up: Images

Contrastive Learning

- **Contrastive learning:** Select pairs of data points in your dataset, apply augmentations to each and train a model to predict which augmented copy is more similar to which original of the pairs. Example: SimCLR

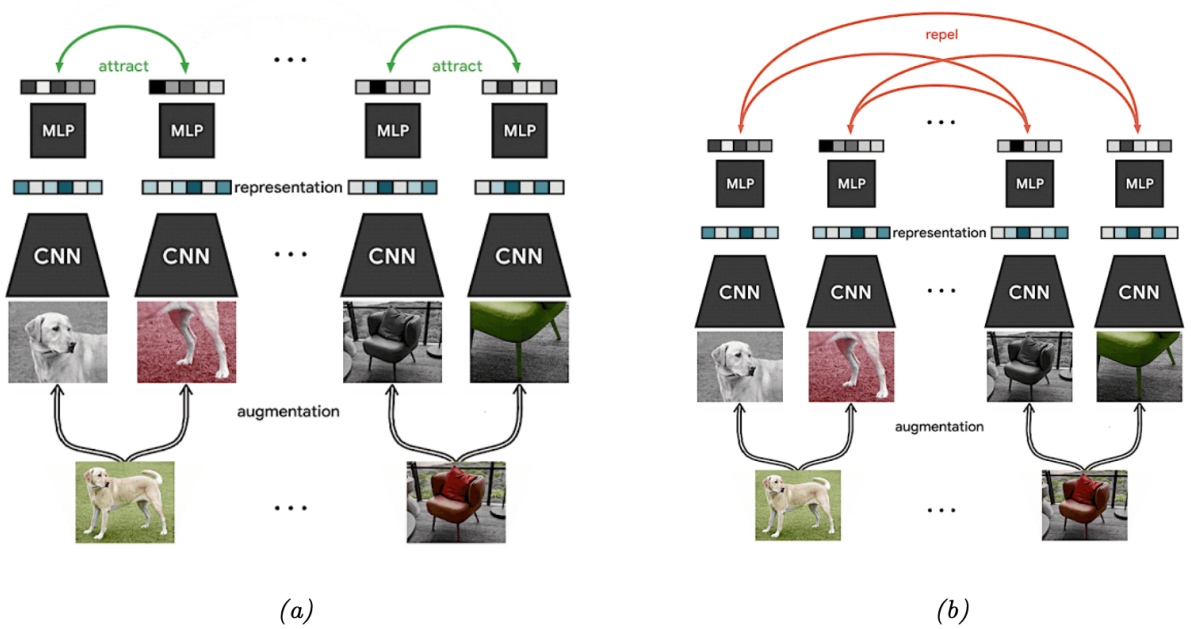


Figure 19.6: Visualization of SimCLR training. Each input image in the minibatch is randomly modified in two different ways (using cropping (followed by resize), flipping, and color distortion), and then fed into a Siamese network. The embeddings (final layer) for each pair derived from the same image is forced to be close, whereas the embeddings for all other pairs are forced to be far. From `https://ai.googleblog.com/2020/04/advancing-self-supervised-and-semi.html`. *Used with kind permission of Ting Chen.*

Image credit Murphy '22

Example: CLIP

- **CLIP**, from OpenAI, considers a mini batch of image-caption pairs from the internet (e.g., Wikipedia), and trains a model to match each caption to the correct image. This enables *zero-shot prediction*: predicting on a new test set without ever seeing a training image.

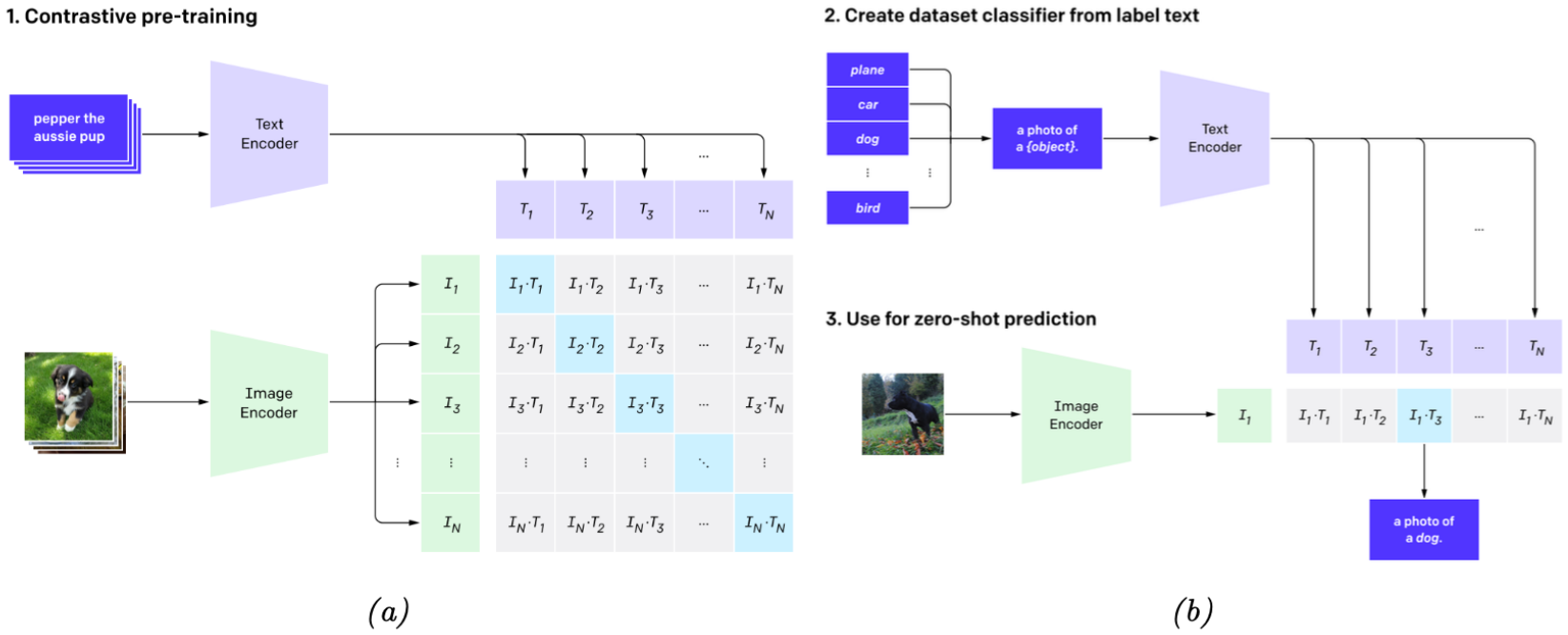


Figure 19.7: Illustration of the CLIP model. From Figure 1 of [Rad+]. Used with kind permission of Alec Radford.

Image credit Murphy '22

Any questions?

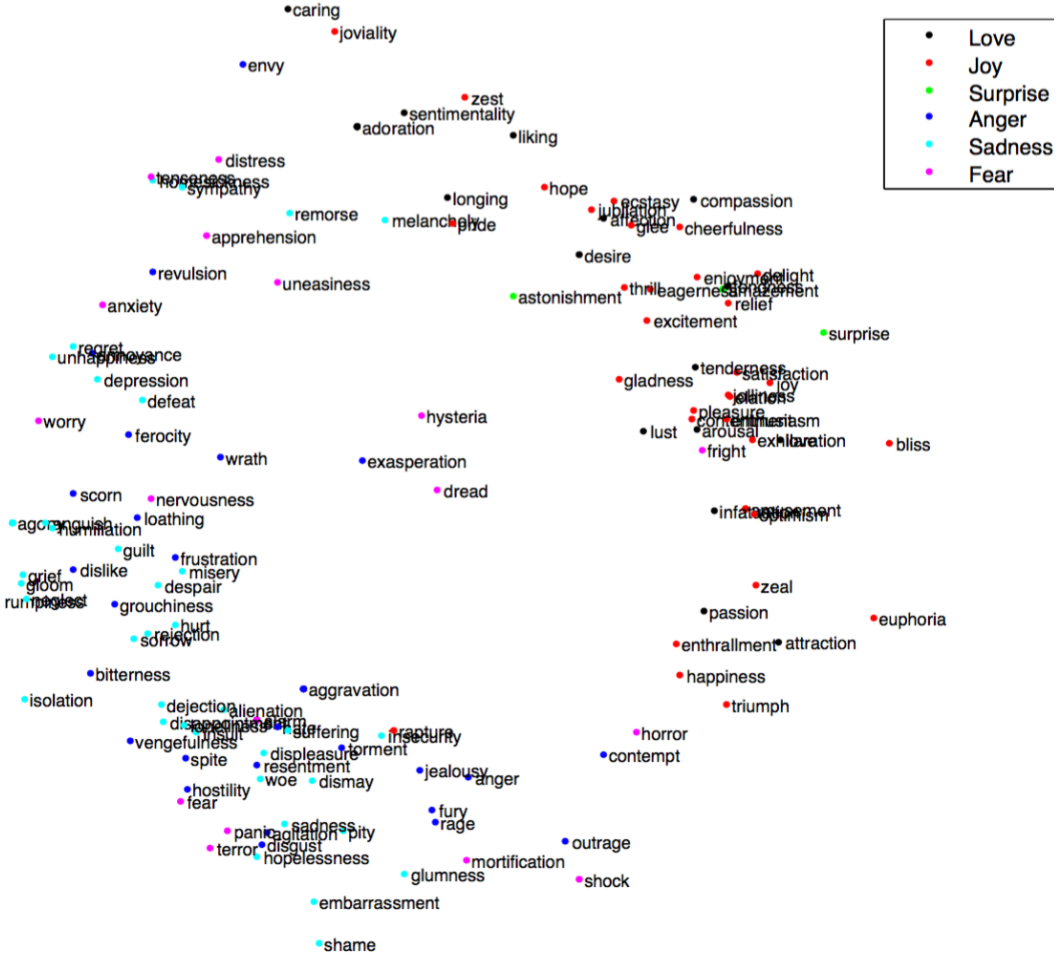
Feature extraction given Text data

Word embeddings

Can we **embed words** into a latent space?

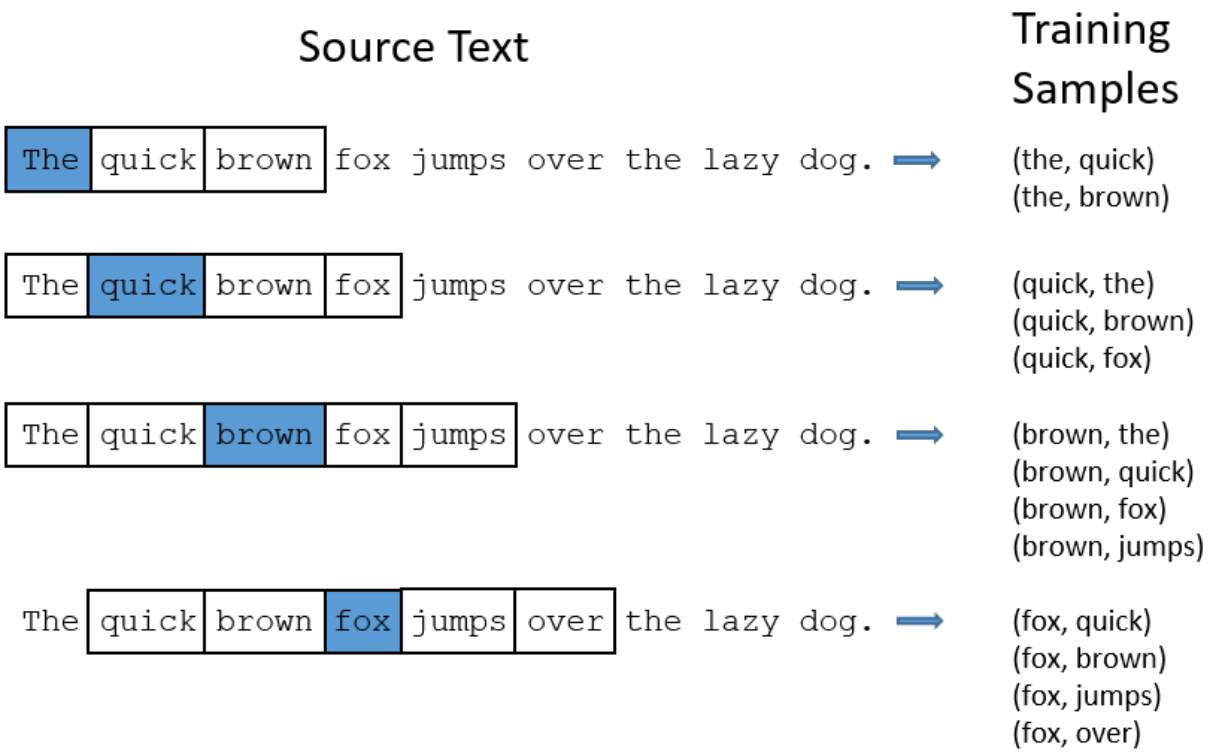
This embedding came from directly querying for relationships.

But we can also infer relationships through natural text (e.g., nytimes)



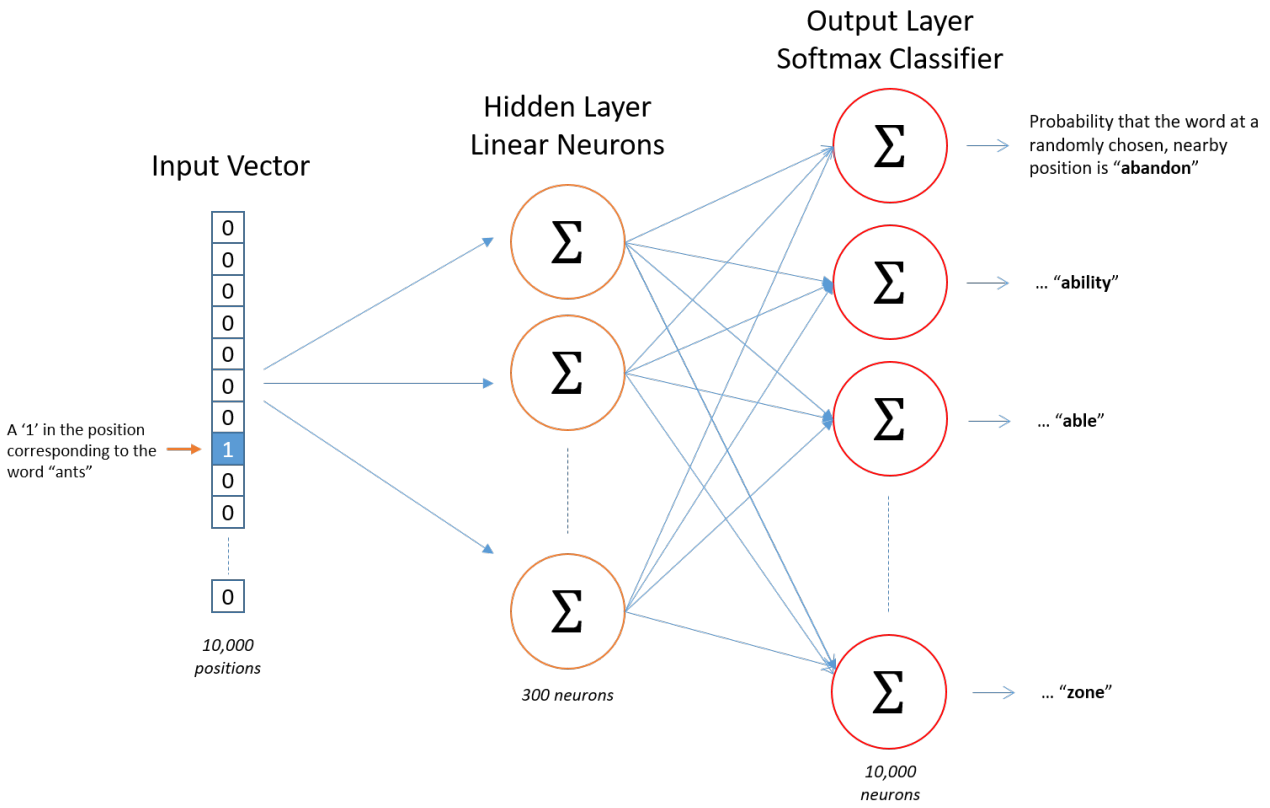
Word embeddings, word2vec

word2vec is a popular unsupervised learning approach that just uses a text corpus



slide: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

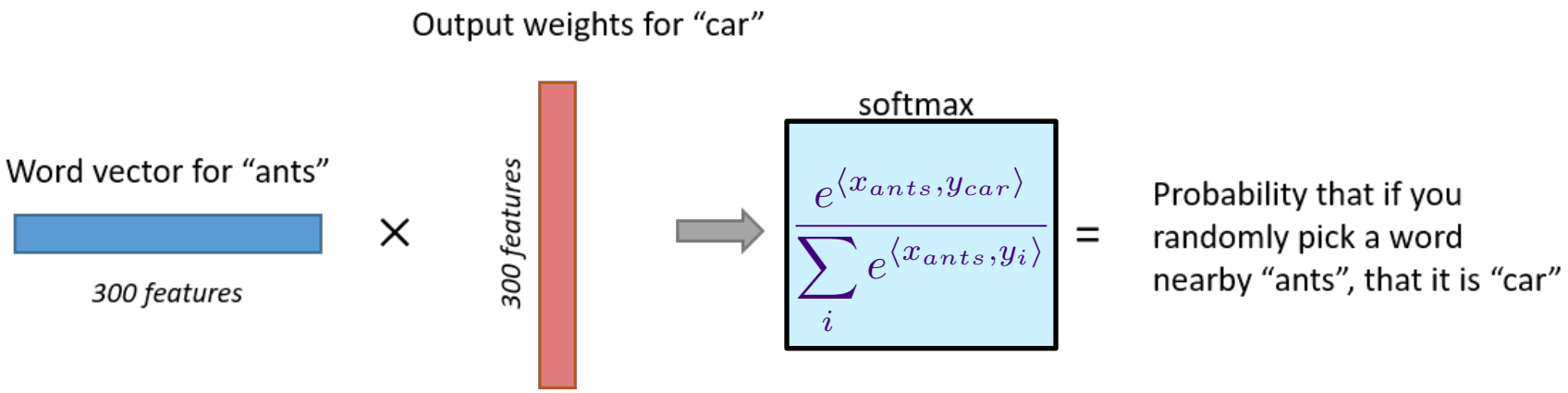
Word embeddings, word2vec



Training neural network to predict co-occuring words. Use first layer weights as embedding, throw out output layer

slide: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

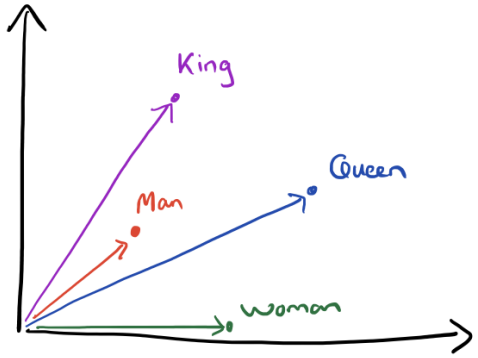
Word embeddings, word2vec



Training neural network to predict co-occuring words. Use first layer weights as embedding, throw out output layer

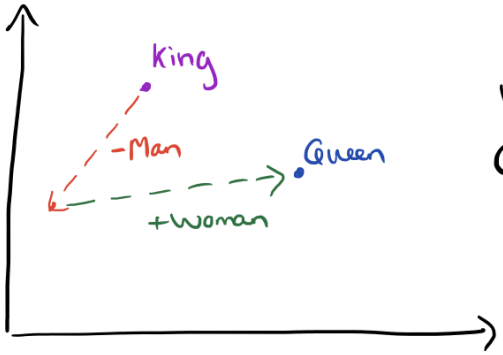
slide: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

word2vec outputs

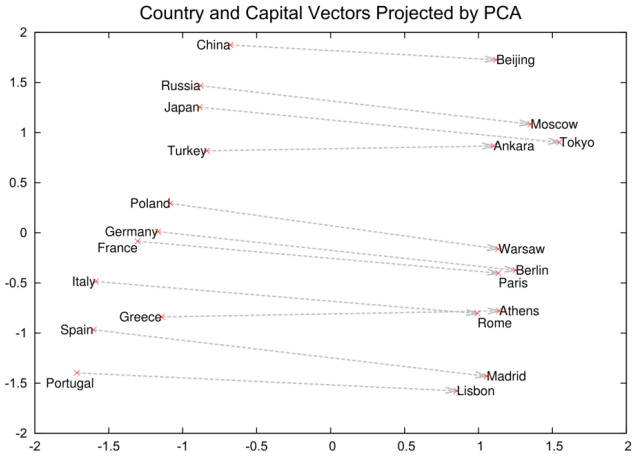


Word Vectors

king - man + woman = queen



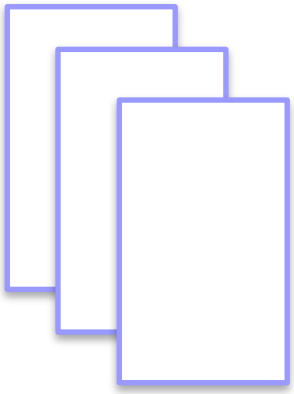
Vector Composition



country - capital

slide: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Bag of Words



n documents/articles with lots of text

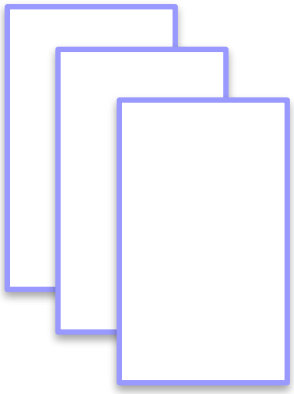
Questions:
- How to get a feature representation of each article?
- How to cluster documents into topics?

Bag of words model:

i*th* document: $x_i \in \mathbb{R}^D$

$x_{i,j} =$ proportion of times jth word occurred in ith document

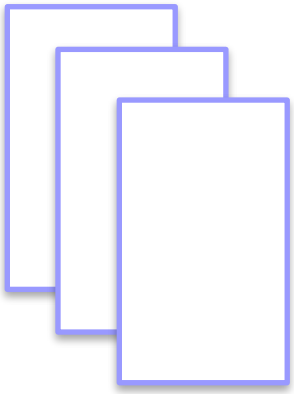
Bag of Words



n documents/articles with lots of text

- **Can we embed each document into a feature space?**

Bag of Words



n documents/articles with lots of text

- **Can we embed each document into a feature space?**

Bag of words model:

*i*th document: $x_i \in \mathbb{R}^D$

$x_{i,j} =$ proportion of times jth word occurred in ith document

Given vectors, run k-means or Gaussian mixture model to find k clusters/topics

Nonnegative matrix factorization (NMF)

$A \in \mathbb{R}^{m \times n}$ $\quad A_{i,j} = \text{frequency of } j\text{th word in document } i$

Nonnegative Matrix factorization:

$$\min_{W \in \mathbb{R}_+^{m \times d}, H \in \mathbb{R}_+^{n \times d}} \|A - WH^T\|_F^2$$

d is number of topics

Also see latent Dirichlet factorization (LDA)

Nonnegative matrix factorization (NMF)

$A \in \mathbb{R}^{m \times n}$ $\quad A_{i,j} = \text{frequency of } j\text{th word in document } i$

Nonnegative Matrix factorization:

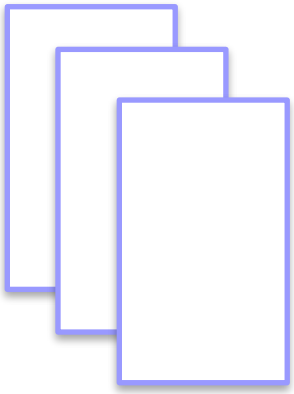
$$\min_{W \in \mathbb{R}_+^{m \times d}, H \in \mathbb{R}_+^{n \times d}} \|A - WH^T\|_F^2$$

d is number of topics

Each column of H represents a cluster of a topic,
Each row W is some weights a combination of topics

Also see latent Dirichlet factorization (LDA)

TF*IDF



n documents/articles with lots of text

How to get a feature representation of each article?

1. For each document *d* compute the proportion of times word *t* occurs out of all words in *d*, i.e. **term frequency**

$$TF_{d,t}$$

2. For each word *t* in your corpus, compute the proportion of documents out of *n* that the word *t* occurs, i.e., **document frequency**

$$DF_t$$

3. Compute score for word *t* in document *d* as $TF_{d,t} \log(\frac{1}{DF_t})$

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

ratebeer

Two Hearted Ale - Input ~2500 natural language reviews

http://www.ratebeer.com/beer/two-hearted-ale/1502/2/1/

3.8 AROMA 8/10 APPEARANCE 4/5 TASTE 8/10 PALATE 3/5 OVERALL 15/20

fonefan (25678) - VestJylland, DENMARK - JAN 18, 2009

Bottle 355ml.

Clear light to medium yellow orange color with a average, frothy, good lacing, fully lasting, off-white head. Aroma is moderate to heavy malty, moderate to heavy hoppy, perfume, grapefruit, orange shell, soap. Flavor is moderate to heavy sweet and bitter with a average to long duration. Body is medium, texture is oily, carbonation is soft. [250908]

4 AROMA 8/10 APPEARANCE 4/5 TASTE 7/10 PALATE 4/5 OVERALL 17/20

Ungstrup (24358) - Oamaru, NEW ZEALAND - MAR 31, 2005

An orange beer with a huge off-white head. The aroma is sweet and very freshly hoppy with notes of hop oils - very powerful aroma. The flavor is sweet and quite hoppy, that gives flavors of oranges, flowers as well as hints of grapefruit. Very refreshing yet with a powerful body.

Reviews for each beer

Bag of Words weighted by TF*IDF

Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

Two Hearted Ale - Weighted Bag of Words:

Reviews for each beer	Bag of Words weighted by TF*IDF	Get 100 nearest neighbors using cosine distance	Non-metric multidimensional scaling	Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

Weighted count vector for the ith beer:

$$z_i \in \mathbb{R}^{400{,}000}$$

Cosine distance:

$$d(z_i, z_j) = 1 - \frac{z_i^T z_j}{||z_i||\,||z_j||}$$

Two Hearted Ale - Nearest Neighbors:

Bear Republic Racer 5
Avery IPA
Stone India Pale Ale (IPA)
Founders Centennial IPA
Smuttynose IPA
Anderson Valley Hop Ottin IPA
AleSmith IPA
BridgePort IPA
Boulder Beer Mojo IPA
Goose Island India Pale Ale
Great Divide Titan IPA
New Holland Mad Hatter Ale
Lagunitas India Pale Ale
Heavy Seas Loose Cannon Hop3
Sweetwater IPA

Reviews for each beer

Bag of Words weighted by TF*IDF

Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

Find an embedding $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ such that

$||x_k - x_i|| < ||x_k - x_j||$ whenever $d(z_k, z_i) < d(z_k, z_j)$

for all 100-nearest neighbors.

(10^7 constraints, 10^5 variables)

distance in 400,000 dimensional "word space"

Solve with hinge loss and stochastic gradient descent.
(20 minutes on my laptop) (d=2,err=6%) (d=3,err=4%)

Could have also used local-linear-embedding, max-volume-unfolding, kernel-PCA, etc.

Reviews for each beer

Bag of Words weighted by TF*IDF

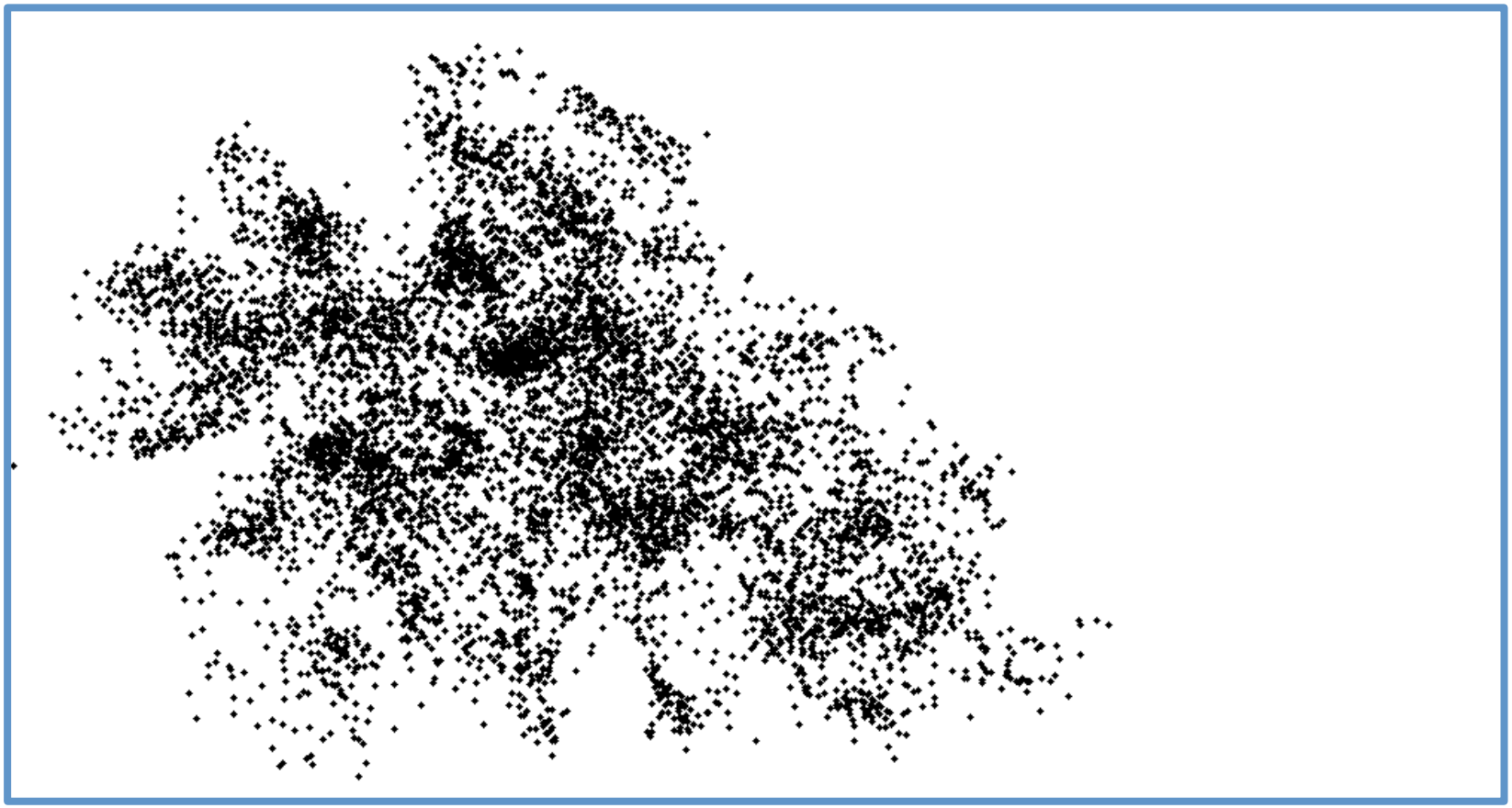
Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$



Reviews for each beer

Bag of Words weighted by TF*IDF

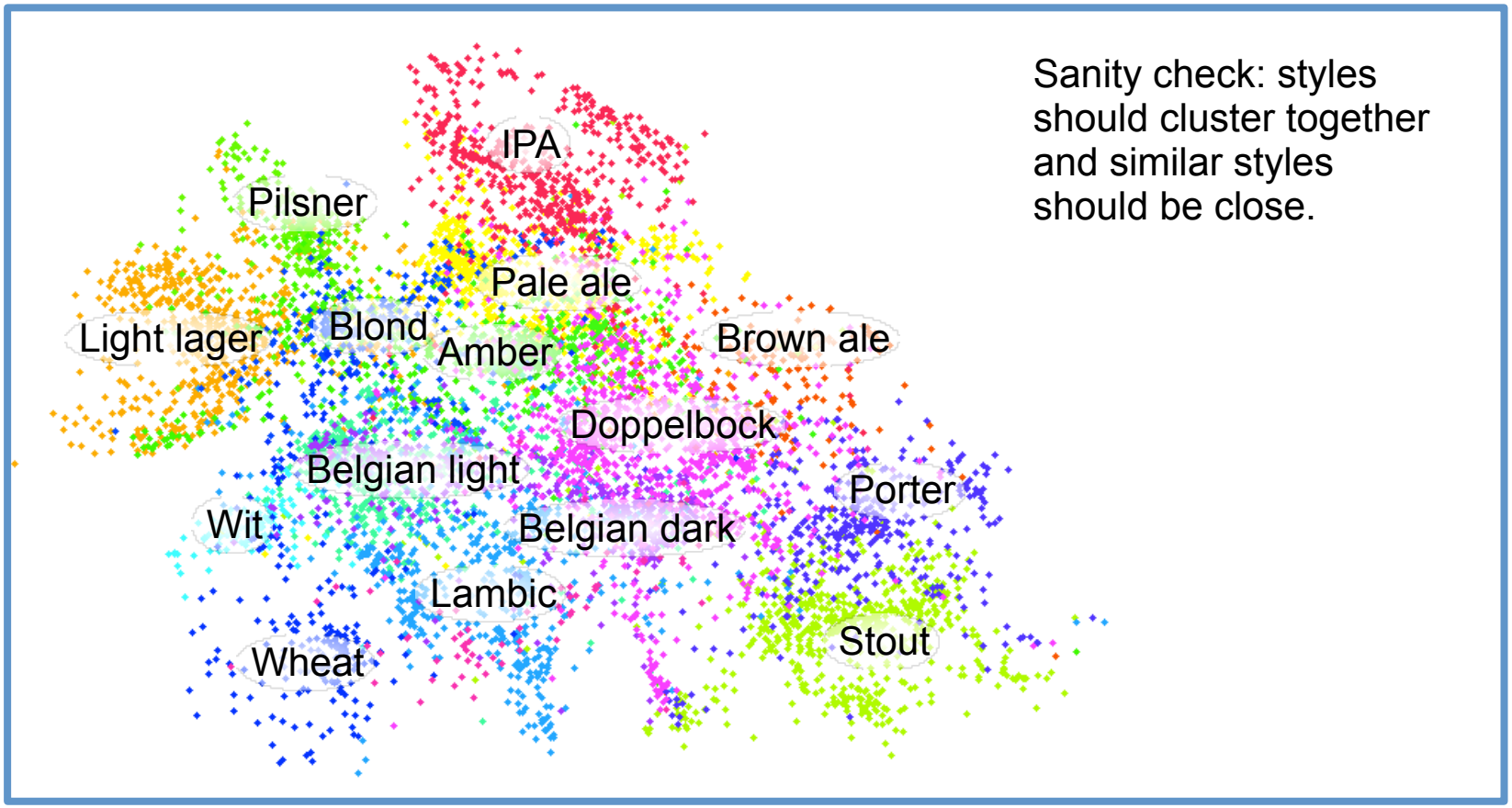
Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$



Reviews for each beer

Bag of Words weighted by TF*IDF

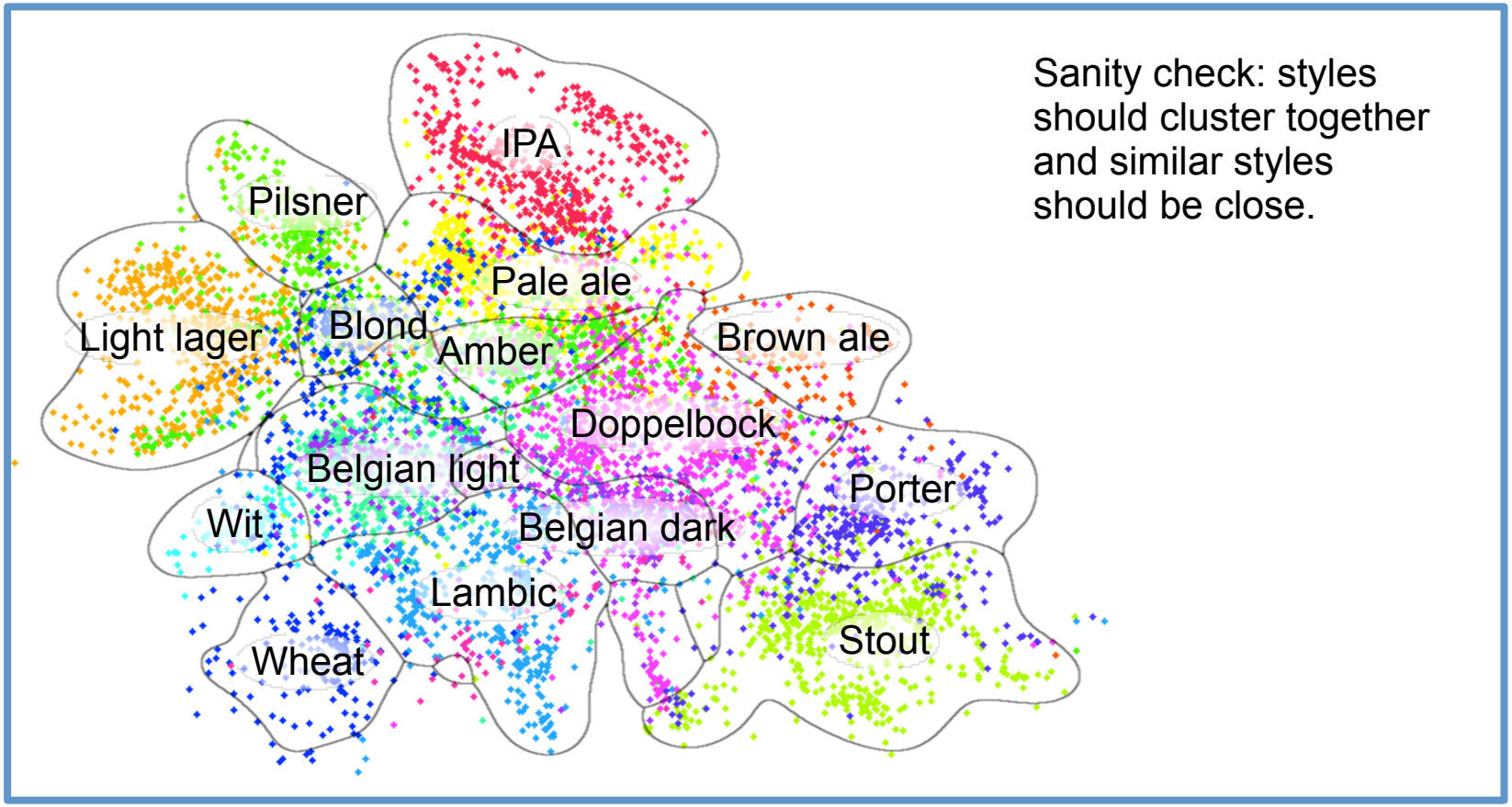
Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$



Reviews for each beer

Bag of Words weighted by TF*IDF

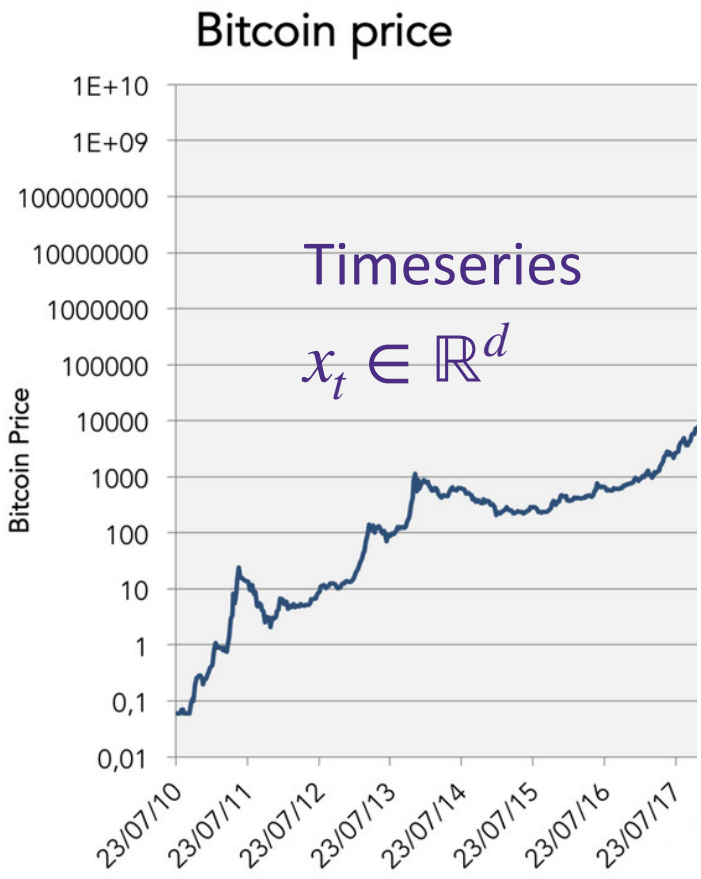
Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

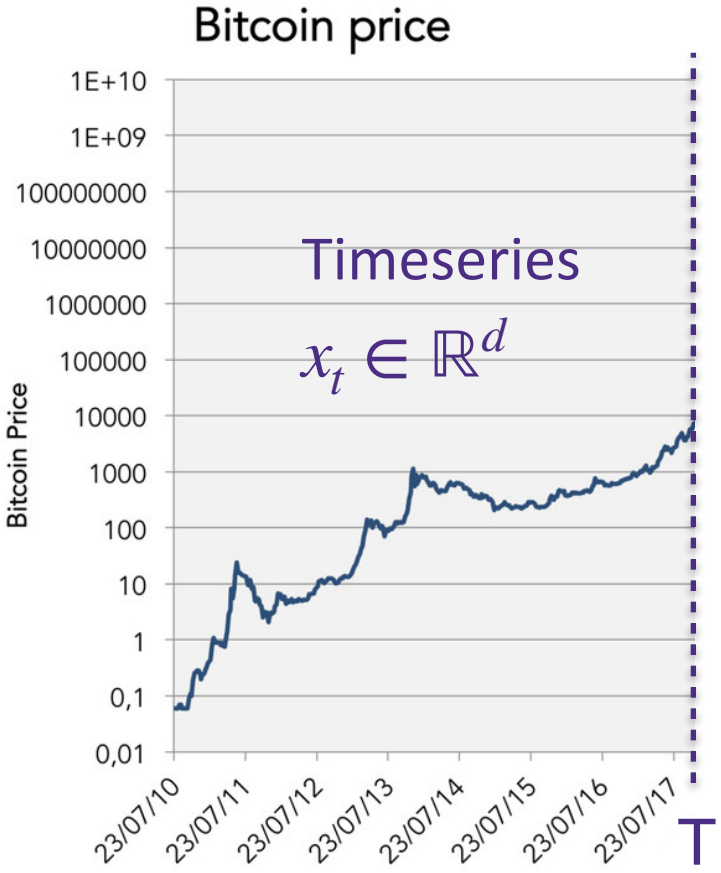
Embedding in d dimensions

Feature extraction given sequential data

Time-dependent data



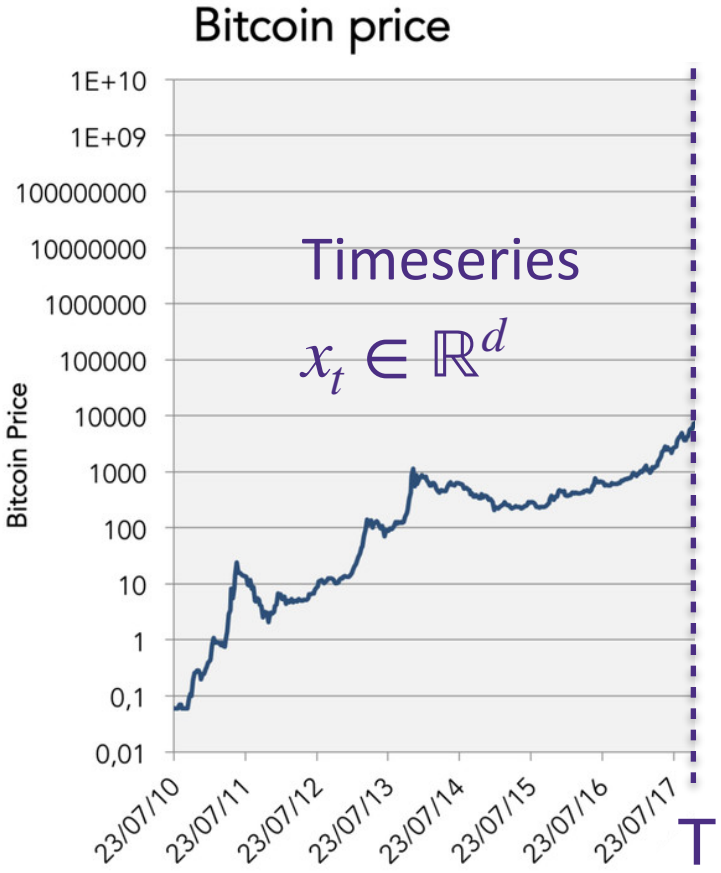
Time-dependent data



To predict x_t for $t > T$ we can learn a model $p_\theta(x_{t+1} \mid x_t, x_{t-1}, \ldots, x_1)$.

How?

Time-dependent data

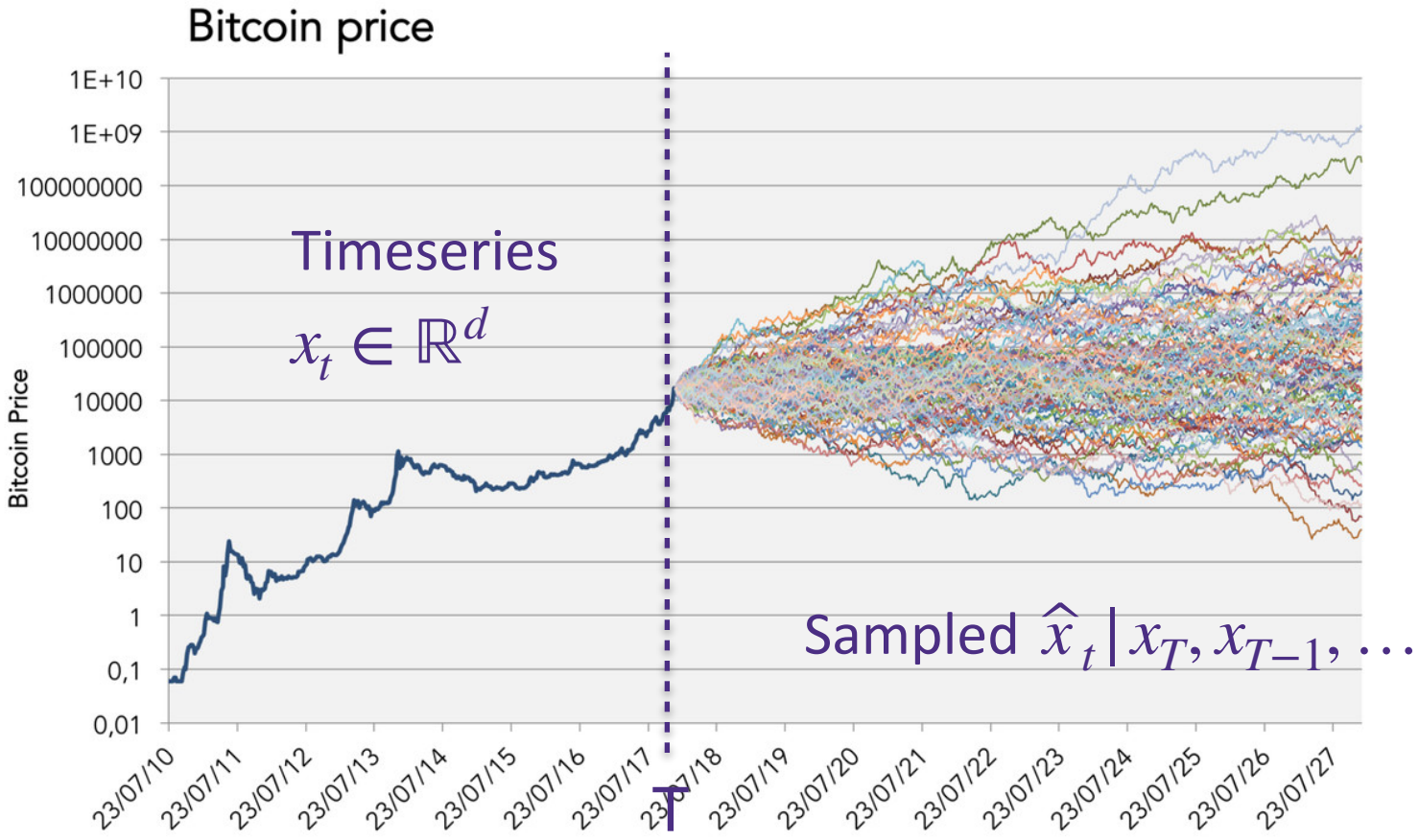


To predict x_t for $t > T$ we can learn a model $p_\theta(x_{t+1} \mid x_t, x_{t-1}, \ldots, x_1)$.

We can then sample from the model:

$$\hat{x}_{t+1} \sim p_\theta(\cdot \mid x_t, x_{t-1}, \ldots)$$
$$\hat{x}_{t+2} \sim p_\theta(\cdot \mid \hat{x}_{t+1}, x_t, x_{t-1}, \ldots)$$
$$\hat{x}_{t+3} \sim p_\theta(\cdot \mid \hat{x}_{t+2}, \hat{x}_{t+1}, x_t, x_{t-1}, \ldots)$$

Time-dependent data



To predict x_t for $t > T$ we can learn a model $p_\theta(x_{t+1} \mid x_t, x_{t-1}, \ldots, x_1)$.

We can then sample from the model:

$$\hat{x}_{t+1} \sim p_\theta(\cdot \mid x_t, x_{t-1}, \ldots)$$
$$\hat{x}_{t+2} \sim p_\theta(\cdot \mid \hat{x}_{t+1}, x_t, x_{t-1}, \ldots)$$
$$\hat{x}_{t+3} \sim p_\theta(\cdot \mid \hat{x}_{t+2}, \hat{x}_{t+1}, x_t, x_{t-1}, \ldots)$$

Time-dependent data

How do we choose a model $p_\theta(x_{t+1} \mid x_t, x_{t-1}, \ldots, x_1)$?

One choice is a Hidden Markov Model:

$$h_{t+1} = f_\theta(x_t, h_t)$$ Neural network

$$x_{t+1} \sim g_\theta(h_{t+1})$$ Probability distribution over $\mathbb{R}^d$

Time-dependent data

How do we choose a model $p_\theta(x_{t+1} \mid x_t, x_{t-1}, \ldots, x_1)$?

One choice is a Hidden Markov Model:

$$h_{t+1} = f_\theta(x_t, h_t)$$ Neural network

$$x_{t+1} \sim g_\theta(h_{t+1})$$ Probability distribution over $\mathbb{R}^d$

Example:

$$h_{t+1} = \Theta_1 \mathrm{ReLu}(\Theta_2 x_t + \Theta_3 h_t)$$
$$x_{t+1} = \Theta_4 h_t + \eta_t \qquad \eta_t \sim \mathcal{N}(0,1)$$

Note that h_t could be much higher dimensional than x_t

Time-dependent data

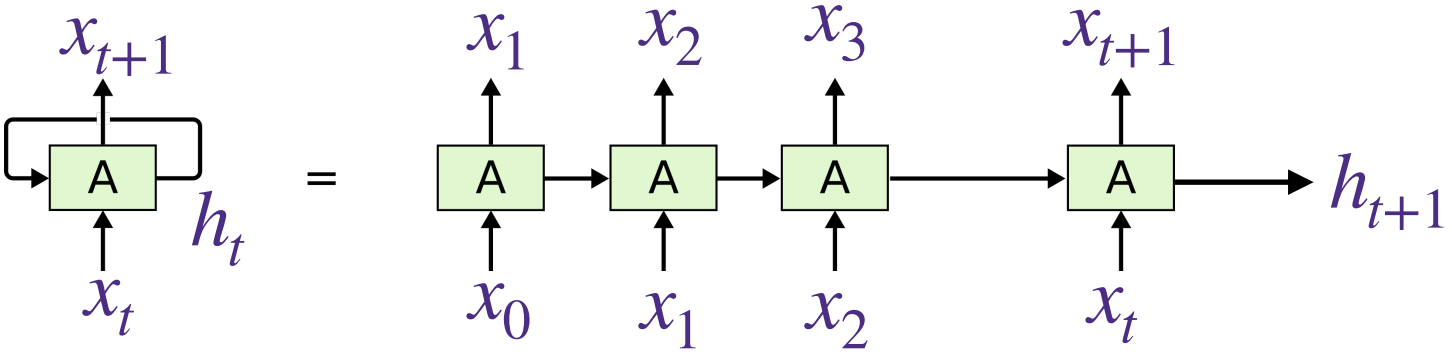
How do we choose a model $p_\theta(x_{t+1} \mid x_t, x_{t-1}, \ldots, x_1)$?

One choice is a Hidden Markov Model:

$$h_{t+1} = f_\theta(x_t, h_t)$$ Neural network

$$x_{t+1} \sim g_\theta(h_{t+1})$$ Probability distribution over $\mathbb{R}^d$

Recurrent Neural Network



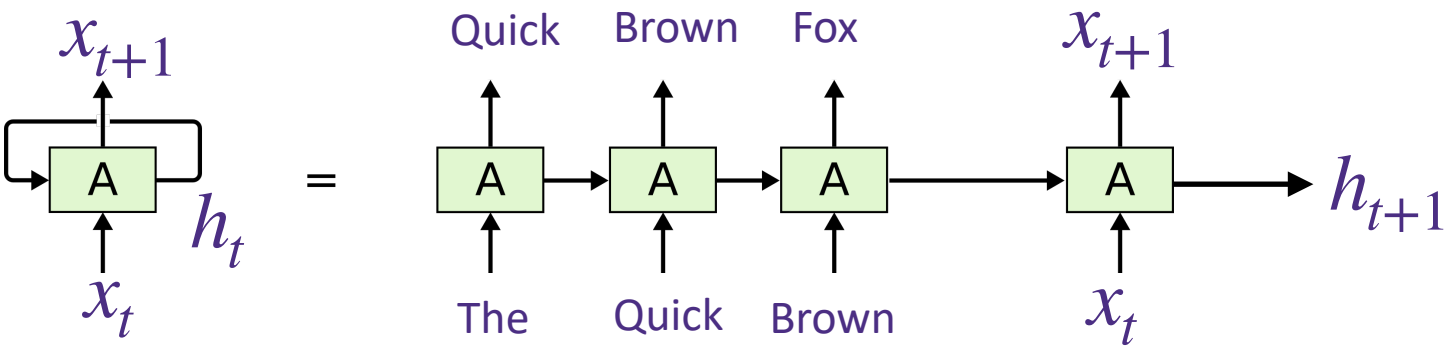
Slide: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Works with text too

One-hot encode each word so that $x_t \in \{0,1\}^D$ where *D* is number of words in dictionary.

$$h_{t+1} = f_\theta(x_t, h_t)$$ Neural network

$$x_{t+1} \sim g_\theta(h_{t+1})$$ Probability distribution over $\mathbb{R}^d$

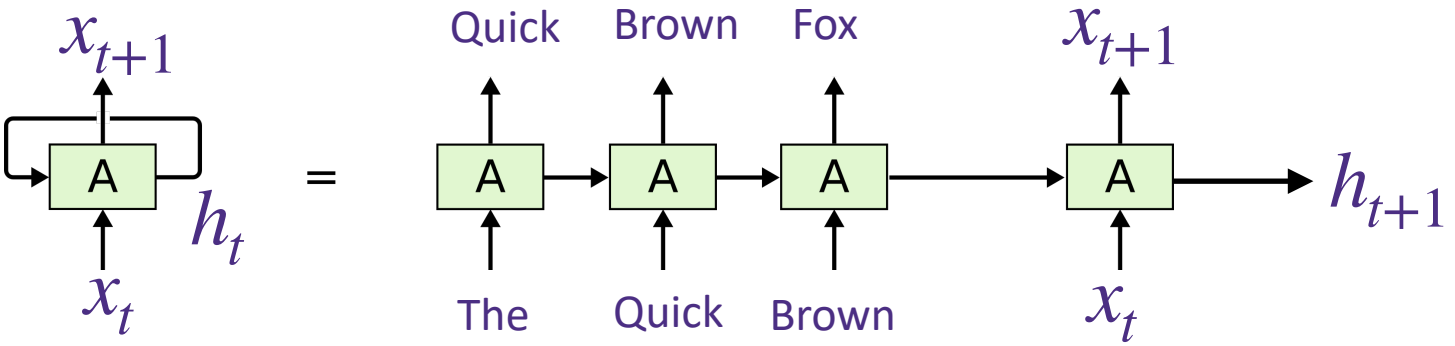


Works with text too

One-hot encode each word so that $x_t \in \{0,1\}^D$ where *D* is number of words in dictionary.

$$h_{t+1} = \Theta_1 \text{ReLu}(\Theta_2 x_t + \Theta_3 h_t)$$

$$x_{t+1} \sim \text{softmax}(h_{t+1})$$

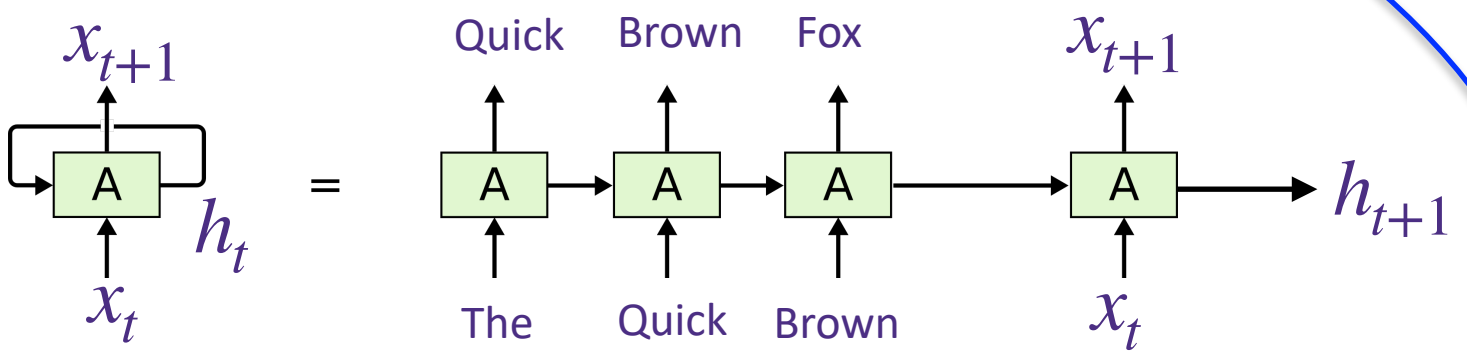


Works with text too

One-hot encode each word so that $x_t \in \{0,1\}^D$ where *D* is number of words in dictionary.

$$h_{t+1} = \Theta_1 \text{ReLu}(\Theta_2 W x_t + \Theta_3 h_t)$$

$$x_{t+1} \sim \text{softmax}(W^\top h_{t+1})$$

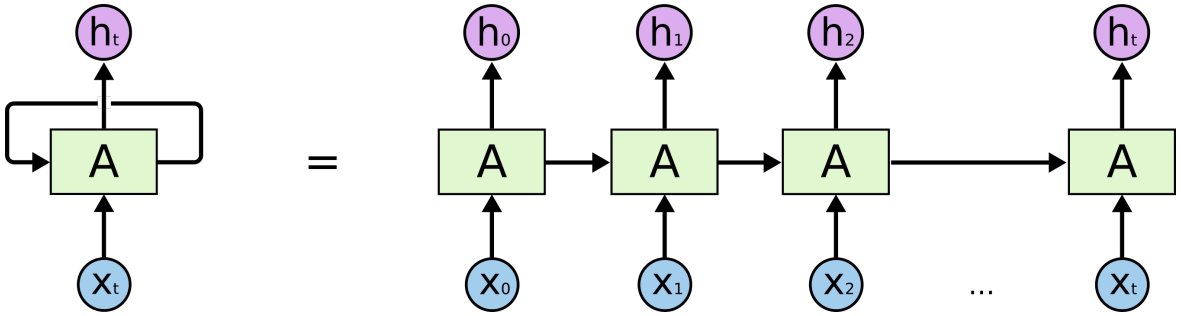


But since h_t and x_t are same dimension, that's a huge hidden state dimension!

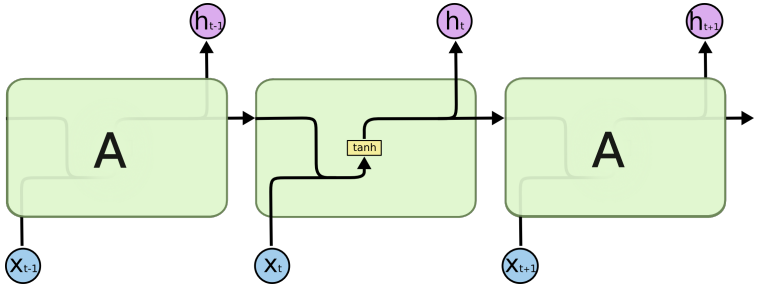
Let $W \in \mathbb{R}^{d \times D}$ be a word embedding (e.g., word2vec) with $d \ll D$ then

Variable length sequences

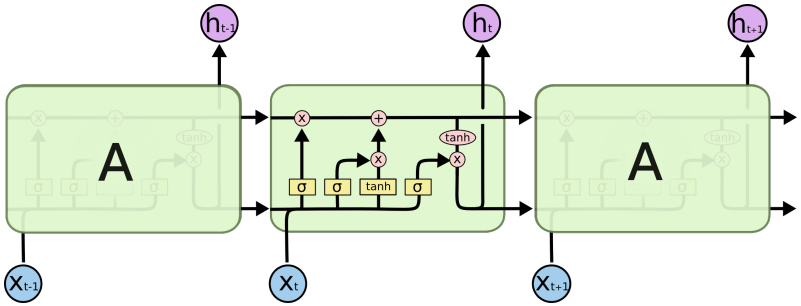
Recurrent Neural Network

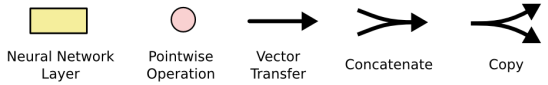


Standard RNN



Gated RNN
LSTM





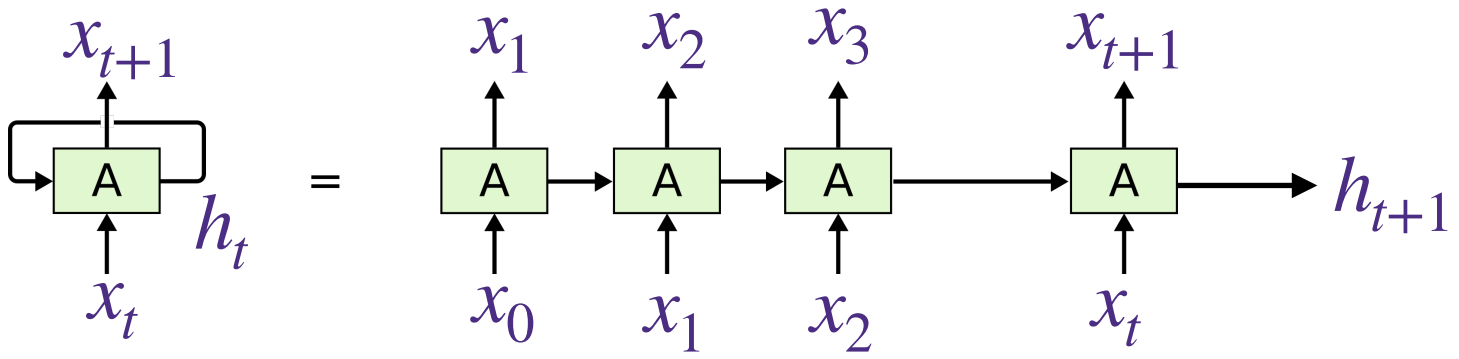
But world is quickly turning towards transformers…

Slide: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Supervised training

$$h_{t+1} = f_\theta(x_t, h_t)$$ Neural network

$$x_{t+1} \sim g_\theta(h_{t+1})$$ Probability distribution over $\mathbb{R}^d$

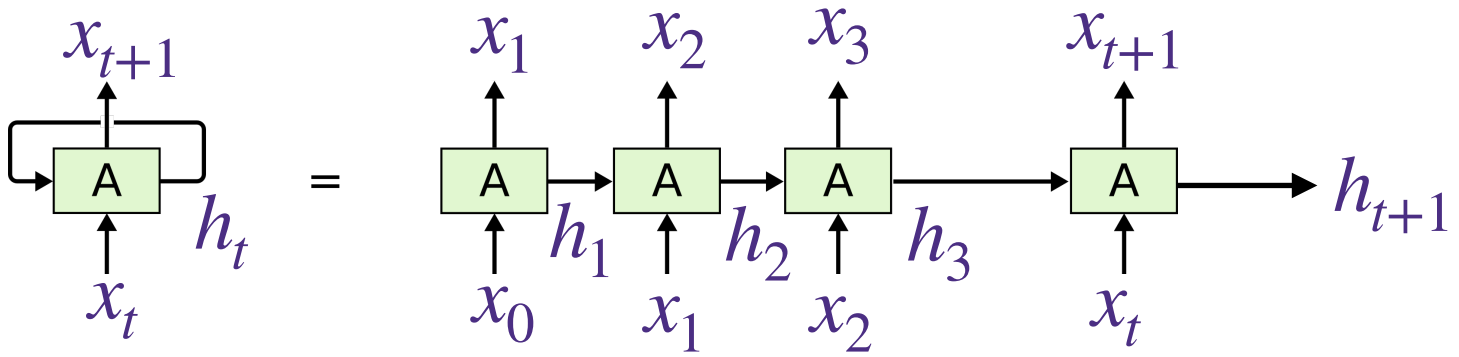


The hidden state encodes everything about the sequence. So you can also train on examples $\left\{ \left((x_t^{(i)}, x_{t-1}^{(i)}, \ldots, x_1^{(i)}), y^{(i)} \right) \right\}_i$ where $x^{(i)}$ is a review and $y^{(i)}$ is sentiment

Attention

$$h_{t+1} = f_\theta(x_t, h_t)$$ Neural network

$$x_{t+1} \sim g_\theta(h_{t+1})$$ Probability distribution over $\mathbb{R}^d$



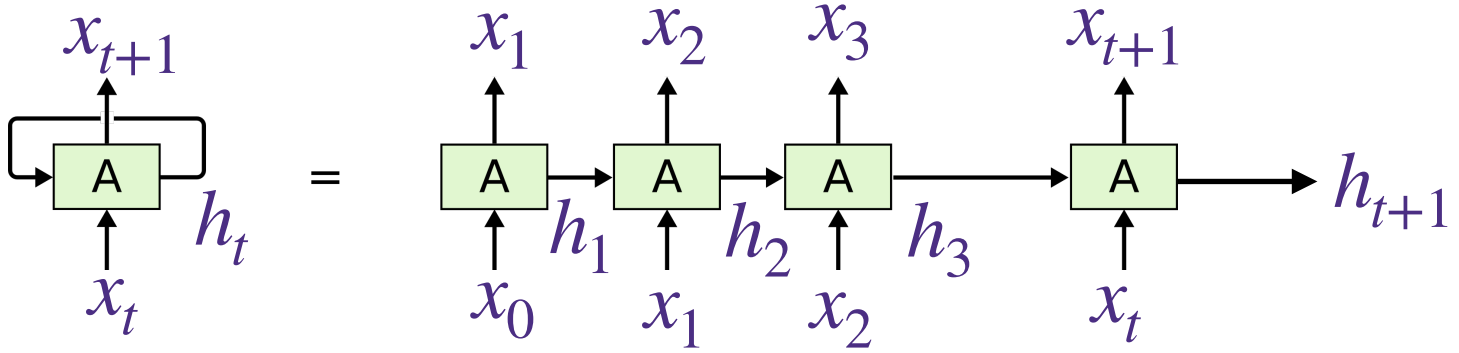
In this model h_{t+1} encodes <u>everything</u> about $\{x_s\}_{s \le t}$ necessary to generate future samples. In practice, the hidden state h_t "**forgets**" the past.

Example: $$h_{t+1} = \Theta_1 \text{ReLu}(\Theta_2 x_t + \Theta_3 h_t)$$

Attention

$$h_{t+1} = f_\theta(x_t, h_t)$$ Neural network

$$x_{t+1} \sim g_\theta(h_{t+1})$$ Probability distribution over $\mathbb{R}^d$



In this model h_{t+1} encodes <u>everything</u> about $\{x_s\}_{s \le t}$ necessary to generate future samples. In practice, the hidden state h_t "**forgets**" the past.

Example: $$h_{t+1} = \Theta_1 \text{ReLu}(\Theta_2 x_t + \Theta_3 \sum_{k=1}^{t} \text{softmax}(h_t; \{h_\ell\}_{\ell=1}^t)_k \, h_k)$$

Attention fixes this by combining past hidden states, weighted by current state

Transformers (Decoder only)

Transformers discard the RNN architecture altogether and just use **attention on the input**

$$h_{t+1} = \Theta_1 \text{ReLU}\Big(\Theta_2 \sum_{k=1}^{t} \text{softmax}(x_t; \{x_\ell\}_{\ell=1}^{t})_k \, x_k\Big)$$

$$x_{t+1} \sim \text{softmax}(h_{t+1})$$

Vaswani et al, “Attention is all you need” 2017

Transformers (Decoder only)

Transformers discard the RNN architecture altogether and just use **attention on the input**

$$h_t^{(0)} = x_t$$

Stack L layers!

$$\ell = 1, \ldots, L$$

$$h_{t+1}^{(\ell+1)} = \Theta_1 \text{ReLU}\Big(\Theta_2 \sum_{k=1}^{t} \text{softmax}(h_t^{(\ell)}; \{h_s^{(\ell)}\}_{s=1}^{t})_k \, h_k^{(\ell)}\Big)$$

$$x_{t+1} \sim \text{softmax}(h_{t+1}^{(L)})$$

Vaswani et al, “Attention is all you need” 2017

Transformers (Decoder only)

Transformers discard the RNN architecture altogether and just use **attention on the input**

$$h_t^{(0)} = Wx_t$$

Stack L layers!

$$\ell = 1, \ldots, L$$

$$h_{t+1}^{(\ell+1)} = \Theta_1 \text{ReLU}\Big(\Theta_2 \sum_{k=1}^{t} \text{softmax}(h_t^{(\ell)}; \{h_s^{(\ell)}\}_{s=1}^{t})_k \, h_k^{(\ell)}\Big)$$

$$x_{t+1} \sim \text{softmax}(W^\top h_{t+1}^{(L)})$$

For text, one hot encode each word as $x_t \in \{0,1\}^d$.

Let $W \in \mathbb{R}^{d \times D}$ be a word embedding (e.g., word2vec) with $d \ll D$, then

Vaswani et al, "Attention is all you need" 2017

The basis for ChatGPT

Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I., 2018. *Improving language understanding by generative pre-training.*

3.1 Unsupervised pre-training

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$, we use a standard language modeling objective to maximize the following likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \quad (1)$$

where k is the size of the context window, and the conditional probability P is modeled using a neural network with parameters Θ. These parameters are trained using stochastic gradient descent [51].

In our experiments, we use a multi-layer *Transformer decoder* [34] for the language model, which is a variant of the transformer [62]. This model applies a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned} \quad (2)$$

where $U = (u_{-k}, \ldots, u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, and W_p is the position embedding matrix.

Full, clean example code: https://github.com/karpathy/nanoGPT

Questions?