Sequencing Legal DNA

NLP for Law and Political Economy

5. Deep Learning for NLP

Weekly Q&A Page

bit.ly/NLP-QA05

Objectives of a (Deep) Machine Learning Project

1. **What is the question or problem?**

Objectives of a (Deep) Machine Learning Project

1. **What is the question or problem?**
2. Corpus and Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata

Objectives of a (Deep) Machine Learning Project

1. **What is the question or problem?**
2. Corpus and Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata
3. Machine learning:
   - Select a model and train it.
   - Fine-tune hyperparameters for out-of-sample fit.
   - Interpret predictions using model explanation methods.

Objectives of a (Deep) Machine Learning Project

1. **What is the question or problem?**
2. Corpus and Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata
3. Machine learning:
   - Select a model and train it.
   - Fine-tune hyperparameters for out-of-sample fit.
   - Interpret predictions using model explanation methods.
4. Empirical analysis
   - Produce statistics or predictions with the trained model.
   - **Answer the question / solve the problem.**

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.
  - This vector of class probabilities is a **compressed representation** of the outcome-predictive text features $\boldsymbol{x}_i$

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.
  - This vector of class probabilities is a **compressed representation** of the outcome-predictive text features $\boldsymbol{x}_i$
  - the vector of features, $\boldsymbol{x}_i$, is itself a compressed representation of the unprocessed document $\mathcal{D}_i$.

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.
  - This vector of class probabilities is a **compressed representation** of the outcome-predictive text features $\boldsymbol{x}_i$
  - the vector of features, $\boldsymbol{x}_i$, is itself a compressed representation of the unprocessed document $\mathcal{D}_i$.
- Correspondingly: the learned parameters $\hat{\theta}$ can also be understood as a **learned compressed representation of the whole dataset**:
  - it contains information about the training corpus, the text features, and the outcomes.

Information in $\hat{\theta}$

Say we train a multinomial logistic regression on a bag-of-words representation of the documents:

- Let θ be the learned matrix of parameters relating words to outcomes:
  - It contains n_y columns, which are n_x-vectors representing the outcome classes.

Information in $\hat{\theta}$

Say we train a multinomial logistic regression on a bag-of-words representation of the documents:

- Let θ be the learned matrix of parameters relating words to outcomes:
  - It contains n_y columns, which are n_x-vectors representing the outcome classes.
  - It contains n_x rows, which are n_y-vectors representing each word in the vocabulary.

Information in $\hat{\theta}$

Say we train a multinomial logistic regression on a bag-of-words representation of the documents:

- Let θ be the learned matrix of parameters relating words to outcomes:
  - It contains n_y columns, which are n_x-vectors representing the outcome classes.
  - It contains n_x rows, which are n_y-vectors representing each word in the vocabulary.
- We could already use θ! e.g.:
  - cluster the column vectors $\rightarrow$ which outcomes are similar/related.
  - cluster the row vectors $\rightarrow$ which features are similar/related.

Information in $\hat{\theta}$: Preview of Word Embeddings

θ = matrix of parameters learned from logit, relating words to outcomes.

- If $\boldsymbol{x}$ is a bag-of-words representation for a document consisting of a list of tokens $\{w_1, ..., w_t, ..., w_n\}$, we can write

$$\boldsymbol{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$$

  - where x_t is an n_x-dimensional one-hot vector – all entries are zero except equals one for the word at t.

Information in $\hat{\theta}$: Preview of Word Embeddings

θ = matrix of parameters learned from logit, relating words to outcomes.

- If $\boldsymbol{x}$ is a bag-of-words representation for a document consisting of a list of tokens $\{w_1, ..., w_t, ..., w_n\}$, we can write

$$\boldsymbol{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$$

  - where x_t is an n_x-dimensional one-hot vector – all entries are zero except equals one for the word at t.
- Let θ_t be the row of θ corresponding to the word w_t: a **word embedding** for w_t containing the outcome-relevant information for that word.

Information in $\hat{\theta}$: Preview of Word Embeddings

$\theta =$ matrix of parameters learned from logit, relating words to outcomes.

- If $\boldsymbol{x}$ is a bag-of-words representation for a document consisting of a list of tokens $\{w_1, ..., w_t, ..., w_n\}$, we can write

$$\boldsymbol{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$$

  - where x_t is an n_x-dimensional one-hot vector – all entries are zero except equals one for the word at t.
- Let θ_t be the row of θ corresponding to the word w_t: a **word embedding** for w_t containing the outcome-relevant information for that word.
- We can construct a **document vector**

$$\vec{\boldsymbol{d}} = \frac{1}{n}\sum_{t=1}^{n_i} \theta_t$$

  the sum of the n_y-dimensional word representations (the row vectors from above).

  - this is called the "continuous bag of words (CBOW)" representation (Goldberg 2017).
  - Note that $\vec{\boldsymbol{d}} = \theta \cdot \boldsymbol{x}$, and thus θ is a **word embedding matrix**.

Recap: Cross-Domain Learning

see link in zoom chat

Outline

Outline

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use tensorflow/keras or torch, rather than sklearn or xgboost.

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use tensorflow/keras or torch, rather than sklearn or xgboost.
- **why use neural nets?**
  - sometimes outperform standard ML techniques on standard problems
  - greatly outperform standard ML techniques on some problems, for example language modeling

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use tensorflow/keras or torch, rather than sklearn or xgboost.
- **why use neural nets?**
  - sometimes outperform standard ML techniques on standard problems
  - greatly outperform standard ML techniques on some problems, for example language modeling
- **why not use neural nets?**
  - usually worse than standard ML on standard problems, and harder to implement.
  - Computational constraints: Recent models like OpenAI's GPT-3 would take ETH Deep Learning Cluster 18 months to train.

“Neural Networks” / “Deep Learning”

- “**Neural**”:
  - NN’s do not work like the brain – such metaphors are misleading.

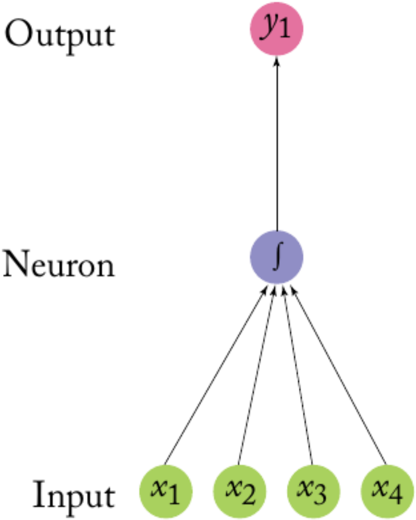
“Neural Networks” / “Deep Learning”

- “**Neural**”:
  - NN’s do not work like the brain – such metaphors are misleading.
- “**Networks**”:
  - NNs are not “networks” as that is understood in mathematical network theory or social science.

"Neural Networks" / "Deep Learning"

- "**Neural**":
  - NN's do not work like the brain – such metaphors are misleading.
- "**Networks**":
  - NNs are not "networks" as that is understood in mathematical network theory or social science.
- "**Deep**" Learning:
  - does not speak to profundity or effectiveness.
  - an unfortunate source of hype.

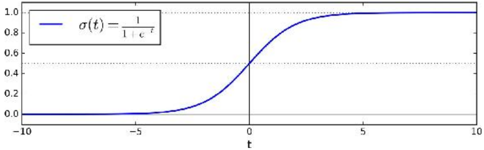
A "Neuron"



- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear "activation function" to the sum
  - (e.g., the $\int$ shape indicates a sigmoid transformation)
- passes the output.

“Neuron” = Logistic Regression

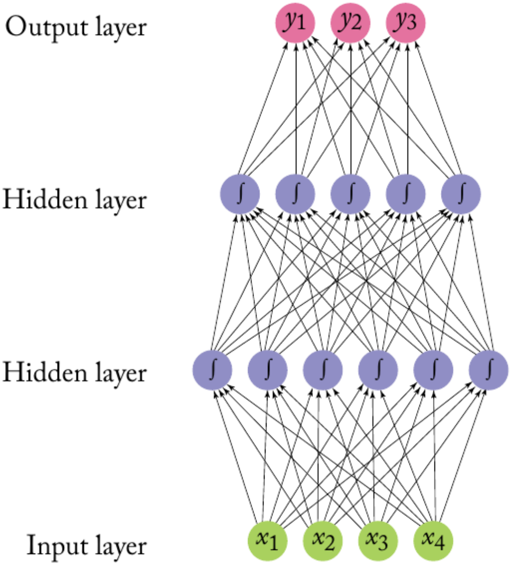
$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear “activation function” (sigmoid) to the sum
- passes the output.

Outline

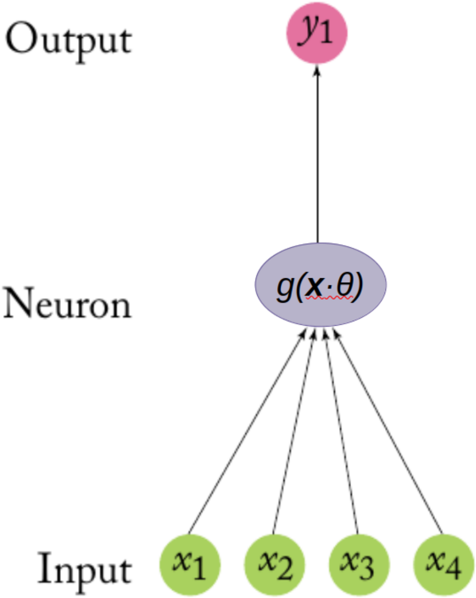
Multi-Layer Perceptron (MLP)



- A multilayer perceptron (also called a feed-forward network or sequential model) stacks neurons horizontally and vertically.
- alternatively, think of it as a stacked ensemble of logistic regression models.
- this vertical stacking is the "deep" in "deep learning"!

- MLP's are composed of "**Dense**" layers, meaning all neurons are connected.
- The tragic result in mathematics of neural nets (Hornik et al 1989, Cybenko 1989):
  - MLP with a single hidden layer, with sigmoid activation, can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$, and any mapping from one finite discrete space to another finite discrete space .
- Telgarsky (2016): NN would have to be exponentially large in many cases.

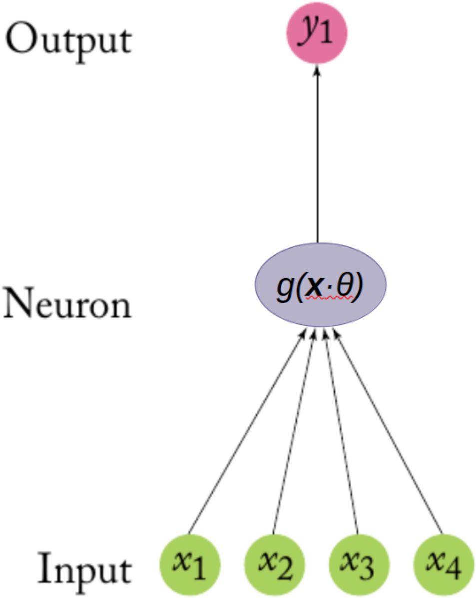
Activation functions $g(\boldsymbol{x} \cdot \theta)$



Previously we had

$$g(\boldsymbol{x} \cdot \theta) = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

Activation functions $g(\boldsymbol{x} \cdot \theta)$

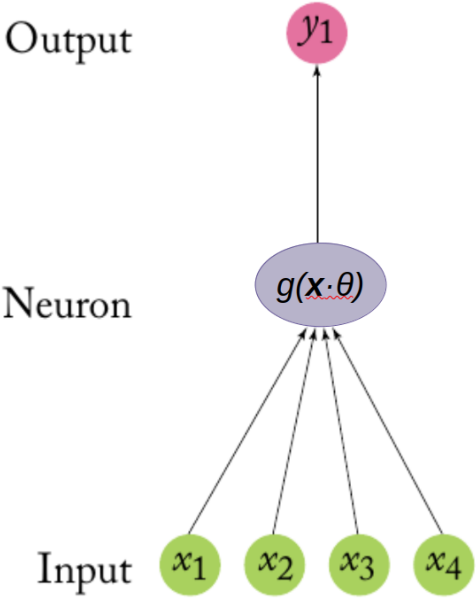


Previously we had

$$g(\boldsymbol{x} \cdot \theta) = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

Activation functions $g(\boldsymbol{x} \cdot \theta)$



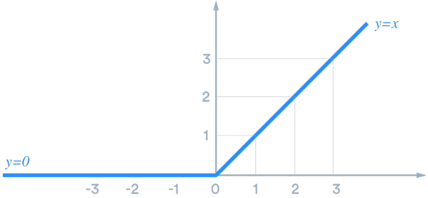
Previously we had

$$g(\boldsymbol{x} \cdot \theta) = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

ReLU (rectified linear unit) function:

$$g(\boldsymbol{x} \cdot \theta) = \text{ReLU}(\boldsymbol{x} \cdot \theta) = \max\{0, \boldsymbol{x} \cdot \theta\}$$



Equation Notation: Multi-Layer Perceptron

- An multi-layer perceptron (MLP) with two hidden layers is

$$\boldsymbol{y} = \boldsymbol{g}_2(\boldsymbol{g}_1(\boldsymbol{x} \cdot \omega_1) \cdot \omega_2) \cdot \boldsymbol{\omega}_y$$

$$\boldsymbol{y} \in \{0,1\}^{n_y}, \boldsymbol{x} \in \mathbb{R}^{n_x}, \boldsymbol{\omega}_1 \in \mathbb{R}^{n_x \times n_1}, \boldsymbol{\omega}_2 \in \mathbb{R}^{n_1 \times n_2}, \boldsymbol{\omega}_y \in \mathbb{R}^{n_2 \times n_y}$$

  - n_1, n_2 = dimensionality in first and second hidden layer.
  - $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_y$ = set of learnable weights for the first hidden, second hidden, and output layer
  - $\boldsymbol{g}_1(\cdot), \boldsymbol{g}_2(\cdot)$ = element-wise non-linear functions for first and seond layer.

Equation Notation: Multi-Layer Perceptron

- An multi-layer perceptron (MLP) with two hidden layers is

$$\boldsymbol{y} = \boldsymbol{g}_2(\boldsymbol{g}_1(\boldsymbol{x} \cdot \boldsymbol{\omega}_1) \cdot \boldsymbol{\omega}_2) \cdot \boldsymbol{\omega}_y$$

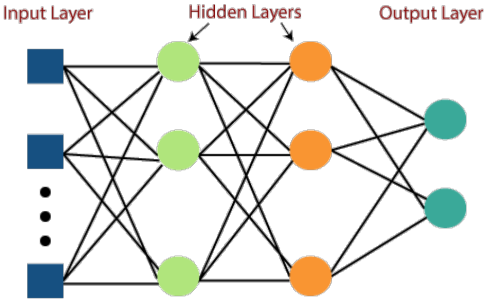
$$\boldsymbol{y} \in \{0,1\}^{n_y}, \boldsymbol{x} \in \mathbb{R}^{n_x}, \boldsymbol{\omega}_1 \in \mathbb{R}^{n_x \times n_1}, \boldsymbol{\omega}_2 \in \mathbb{R}^{n_1 \times n_2}, \boldsymbol{\omega}_y \in \mathbb{R}^{n_2 \times n_y}$$

  - $n_1, n_2 =$ dimensionality in first and second hidden layer.
  - $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_y =$ set of learnable weights for the first hidden, second hidden, and output layer
  - $\boldsymbol{g}_1(\cdot), \boldsymbol{g}_2(\cdot) =$ element-wise non-linear functions for first and seond layer.

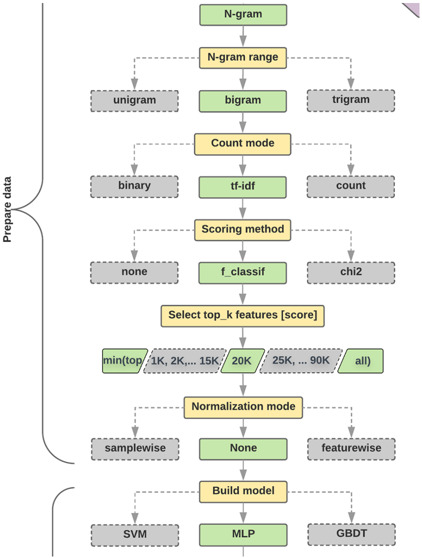
- Can also be written in decomposed notation:

$$\boldsymbol{h}_1 = \boldsymbol{g}_1(\boldsymbol{x} \cdot \boldsymbol{\omega}_1)$$
$$\boldsymbol{h}_2 = \boldsymbol{g}_2(\boldsymbol{h}_1 \cdot \boldsymbol{\omega}_2)$$
$$\boldsymbol{y} = \boldsymbol{h}_2 \cdot \boldsymbol{\omega}_y$$

where $\boldsymbol{h}_l$ indicate hidden layers.



- The Google Developers Guide recommends an MLP for text classification with relatively few but longer documents.:
  - $x =$ **tf-idf-weighted bigrams** as a baseline specification for text classification tasks.
    - select 20,000 features using supervised feature selection in training set.
  - $f(\cdot) =$ MLP with two-dimensional hidden layers.
- A simple MLP is one of the models tried by the U.K. parliament paper (Peterson and Spirling 2018).



Outline

- See the Geron book and sample notebooks for Keras examples.
  - "Dense" layer is the DNN baseline – means that all neurons are connected.
  - performance improvements of making the model deeper than 2 layers are minimal (Reimers & Gurevych, 2017)

- See the Geron book and sample notebooks for Keras examples.
  - "Dense" layer is the DNN baseline – means that all neurons are connected.
  - performance improvements of making the model deeper than 2 layers are minimal (Reimers & Gurevych, 2017)
- Neural nets have *many* dimensions for tuning.
  - the # of layers, # of neurons, activation functions, regularization, optimizer, learning rate, normalization, etc are all tunable hyperparameters.
  - this is a major practical downside of using neural nets rather than sklearn or xgboost for most tasks.

- See the Geron book and sample notebooks for Keras examples.
  - "Dense" layer is the DNN baseline – means that all neurons are connected.
  - performance improvements of making the model deeper than 2 layers are minimal (Reimers & Gurevych, 2017)
- Neural nets have *many* dimensions for tuning.
  - the # of layers, # of neurons, activation functions, regularization, optimizer, learning rate, normalization, etc are all tunable hyperparameters.
  - this is a major practical downside of using neural nets rather than sklearn or xgboost for most tasks.
  - cross-validating these architectural chooices is usually too computationally expensive.
  - instead, make a big model (too many layers, too many neurons) and regularize with dropout and early stopping.

Dropout

An elegant regularization technique:

- at every training step, every neuron has some probability (typically $p = 0.5$) of being temporarily dropped out, so that it will be ignored at this step.
- at test time, neurons dont get dropped anymore but coefficients are down-weighted by p.

Dropout

An elegant regularization technique:

- at every training step, every neuron has some probability (typically $p = 0.5$) of being temporarily dropped out, so that it will be ignored at this step.
- at test time, neurons dont get dropped anymore but coefficients are down-weighted by p.

Why it works:

- Approximates an ensemble of N models (where N is the number of neurons).
- Neurons cannot co-adapt with neighbors; they must be independently useful.
- Layers cannot rely excessively on just a few inputs.

Early Stopping

- A second elegant regularization technique, used in both xgboost and in neural nets:
  - gradually train the model gradients while checking fit in a held-out validation set.
  - when model starts overfitting, stop training.

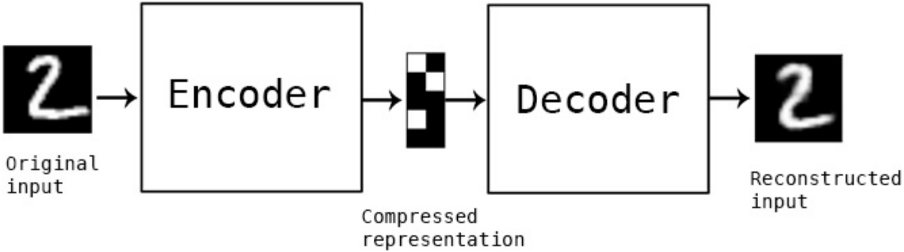
Early Stopping

- A second elegant regularization technique, used in both xgboost and in neural nets:
  - gradually train the model gradients while checking fit in a held-out validation set.
  - when model starts overfitting, stop training.
- Requires user to specify two additional parameters:
  - validation set: a third sample of the data, separate from training set and test set
  - early stopping rounds: stop training if we have done this many epochs without improving validation set performance.

Outline

Autoencoders: Optimal Compression Algorithms

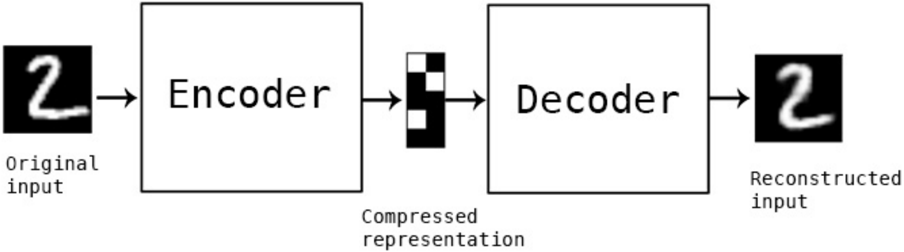
- Autoencoders = neural nets that perform domain-specific lossy compression:



- Learned encodings can be decoded back to a *reconstruction* – a (minimally) lossy representation of the original data.

Autoencoders: Optimal Compression Algorithms

- Autoencoders = neural nets that perform domain-specific lossy compression:



- Learned encodings can be decoded back to a *reconstruction* – a (minimally) lossy representation of the original data.
- AE's can memorize complex, unstructured data.

Autoencoder Architecture

- Stacked layers gradually decrease in dimensionality to create the compressed representation
- then gradually increase in dimensionality to try to reconstruct the input.

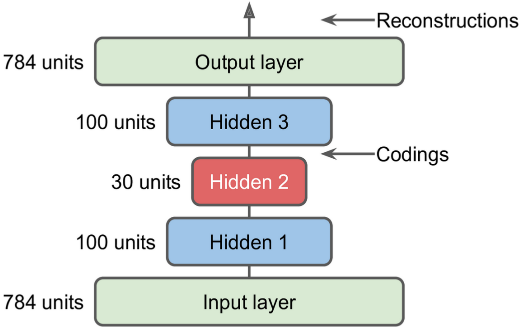


Figure 17-3. Stacked autoencoder

Reconstruction from encoded vector

Figure 17-4. Original images (top) and their reconstructions (bottom)

Autoencoders for Data Visualization

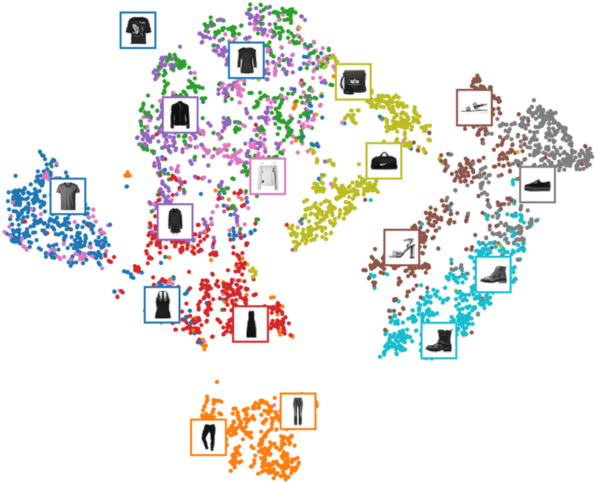
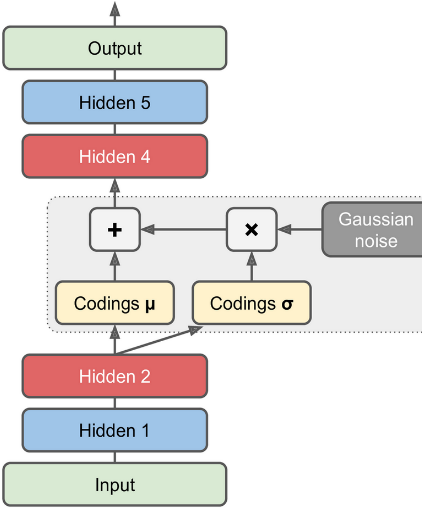


Figure 17-5. Fashion MNIST visualization using an autoencoder followed by t-SNE

- Decent baseline for visualizing the encodings:
  - use an autoencoder to compress your data to relatively low dimension (e.g. 32 dimensions)
  - then use t-SNE for mapping the compressed data to a 2D plane.

Variational Autoencoders

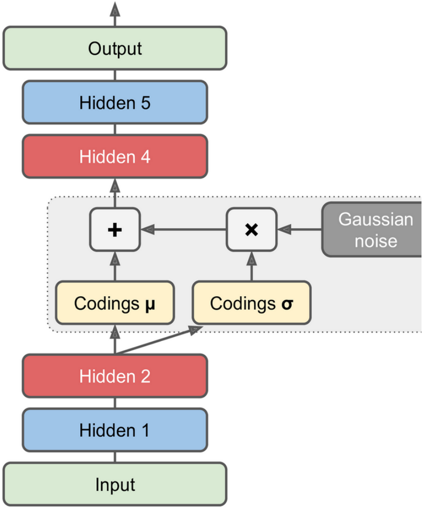
Encodings taken as parameters of a gaussian (means μ and variances σ^2)



Decoder draws from the distribution to produce first layer.

Variational Autoencoders

Encodings taken as parameters of a gaussian (means μ and variances σ^2)

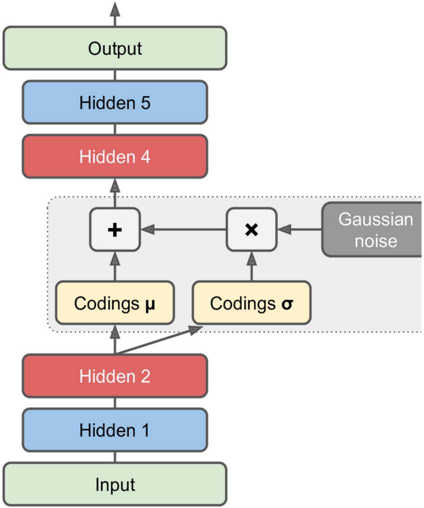


Decoder draws from the distribution to produce first layer.

- Can then sample from the normal distribution (or just choose numbers) and generate reconstructions.

Variational Autoencoders

Encodings taken as parameters of a gaussian (means μ and variances σ^2)



Decoder draws from the distribution to produce first layer.

- Can then sample from the normal distribution (or just choose numbers) and generate reconstructions.
- VAE's do *semantic interpolation*: picking an encoding vector between two encodings will produce a reconstruction that is "between" the associated images

Activity: Zoom Poll 5.1

- Check each true item.

Outline

Sequence Data

- The real break-through from deep learning for NLP:
  - moving from bag-of-X representations to sequence representations.

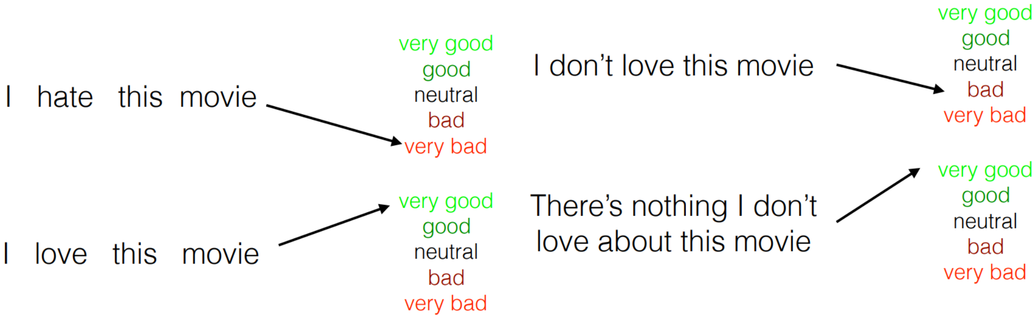
Sequence Data

- The real break-through from deep learning for NLP:
  - moving from bag-of-X representations to sequence representations.
  - Rather than inputting **counts over words** $\boldsymbol{x}$, take as input a **sequence of tokens** $\{w_1, ..., w_t, ...w_n\}$.

Sequence Data

- The real break-through from deep learning for NLP:
  - moving from bag-of-X representations to sequence representations.
  - Rather than inputting **counts over words $\boldsymbol{x}$**, take as input a **sequence of tokens** $\{w_1, ..., w_t, ... w_n\}$.
- "Traditional" architectures:
  - Convolutional neural nets (CNNs)
  - Recurrent Neural Nets (RNNs)
- Since 2018, CNNs and RNNs (as currently implemented) usually get worse performance than attentional neural nets (transformers).

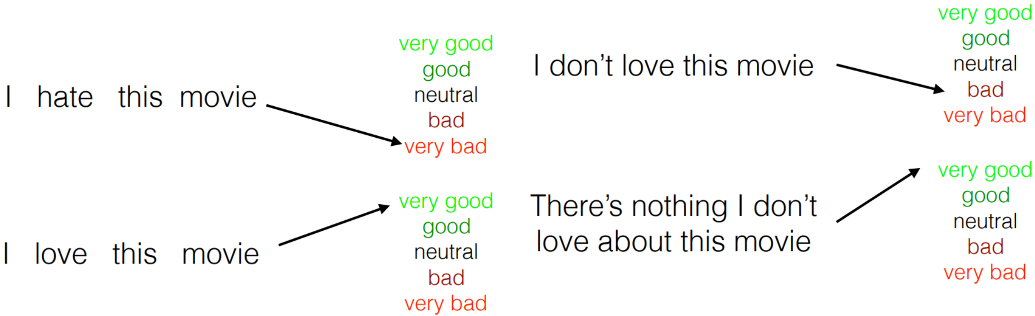
The Classic Sentence Classification Problem



Source: Graham Neubig slides.

- (continuous) bag of words models (even with hidden layers) won't capture the importance of "don't love" or "nothing I don't love".

The Classic Sentence Classification Problem



Source: Graham Neubig slides.

- (continuous) bag of words models (even with hidden layers) won't capture the importance of "don't love" or "nothing I don't love".
- Our current solution, N-grams, has the downsides of a large feature space and no sharing of information/weights across similar words/n-grams.

Convolutional Neural Nets $\leftrightarrow$ N-gram Detectors

- A neural net architecture that constructs **filters** that slide across input sequences and extract **local predictive structure**.

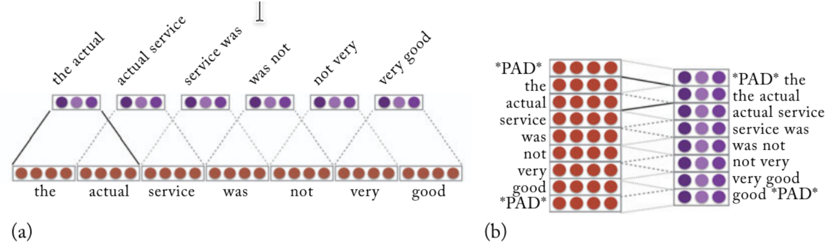


Figure 13.1: The inputs and outputs of a narrow and a wide convolution in the vector-concatenation and the vector-stacking notations. (a) A *narrow* convolution with a window of size $k = 2$ and 3-dimensional output ($\ell = 3$), in the vector-concatenation notation. (b) A *wide* convolution with a window of size $k = 2$, a 3-dimensional output ($\ell = 3$), in the vector-stacking notation.

Johnson and Zhang 2015

- Use CNN for classifying documents by topic and sentiment.
- Interesting part is showing what CNN used for the prediction.

Johnson and Zhang 2015

- Use CNN for classifying documents by topic and sentiment.
- Interesting part is showing what CNN used for the prediction.
- Linear model (SVM) relied on single words and just a few n-grams:
  - poor, useless, returned, not worth, return, worse, disappointed, terrible, worst, horrible
  - great, excellent, perfect, love, easy, amazing, awesome, no problems, perfectly, beat

Johnson and Zhang 2015

- Use CNN for classifying documents by topic and sentiment.
- Interesting part is showing what CNN used for the prediction.
- Linear model (SVM) relied on single words and just a few n-grams:
  - poor, useless, returned, not worth, return, worse, disappointed, terrible, worst, horrible
  - great, excellent, perfect, love, easy, amazing, awesome, no problems, perfectly, beat
- CNN recovers longer, more interesting phrases:

N1	completely useless ., return policy .
N2	it won't even, but doesn't work
N3	product is defective, very disappointing !
N4	is totally unacceptable, is so bad
N5	was very poor, it has failed
P1	works perfectly !, love this product
P2	very pleased !, super easy to, i am pleased
P3	'm so happy, it works perfect, is awesome !
P4	highly recommend it, highly recommended !
P5	am extremely satisfied, is super fast

Table 5: Examples of predictive text regions in the training set.

were unacceptably bad, is abysmally bad, were universally poor, was hugely disappointed, was enormously disappointed, is monumentally frustrating, are endlessly frustrating
best concept ever, best ideas ever, best hub ever, am wholly satisfied, am entirely satisfied, am incredicbly satisfied, 'm overall impressed, am awfully pleased, am exceptionally pleased, 'm entirely happy, are acoustically good, is blindingly fast,

Table 6: Examples of text regions that contribute to prediction. They are from the *test set*, and they did *not* appear in the training set, either entirely or partially as bi-grams.

- but overall, CNNs do not work well in NLP.

Outline

What is an Embedding?

- In a broad sense, **“embedding”** refers to a lower-dimensional dense vector representation of a higher-dimensional object.
  - in NLP, this higher-dimensional object will be a document.

What is an Embedding?

- In a broad sense, **“embedding”** refers to a lower-dimensional dense vector representation of a higher-dimensional object.
  - in NLP, this higher-dimensional object will be a document.
- Not embeddings:
  - counts over LIWC dictionary categories.
  - sklearn CountVectorizer count vectors

What is an Embedding?

- In a broad sense, **"embedding"** refers to a lower-dimensional dense vector representation of a higher-dimensional object.
  - in NLP, this higher-dimensional object will be a document.
- Not embeddings:
  - counts over LIWC dictionary categories.
  - sklearn CountVectorizer count vectors
- Embeddings:
  - PCA reductions of the word count vectors
  - LDA topic shares
  - compressed encodings from an autoencoder

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensonal categorical variable (e.g. area of law with 1000 categories)
- including dummy variables A for each category in your ML model is computationally expensive.

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensonal categorical variable (e.g. area of law with 1000 categories)
- including dummy variables A for each category in your ML model is computationally expensive.

Embedding approaches:

1. PCA applied to the dummy variables A to get lower-dimensional $\tilde{A}$.

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensonal categorical variable (e.g. area of law with 1000 categories)
- including dummy variables A for each category in your ML model is computationally expensive.

Embedding approaches:

1. PCA applied to the dummy variables A to get lower-dimensional $\tilde{A}$.
2. Regress Y on A, predict $\hat{Y}(A_i)$, add that as a predictor in your model instead.

Categorical Embeddings = dense representations of categorical variables

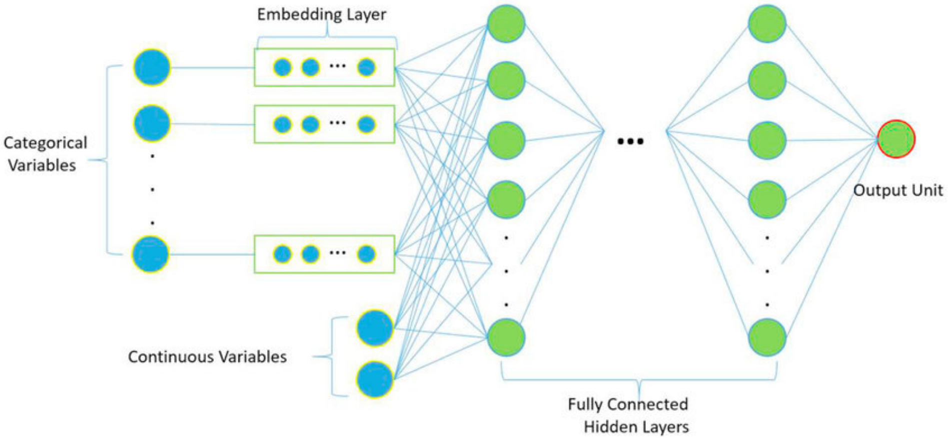
Say we have a binary classification problem with outcome Y:

- we have a high-dimensonal categorical variable (e.g. area of law with 1000 categories)
- including dummy variables A for each category in your ML model is computationally expensive.

Embedding approaches:

1. PCA applied to the dummy variables A to get lower-dimensional $\tilde{A}$.
2. Regress Y on A, predict $\hat{Y}(A_i)$, add that as a predictor in your model instead.

(2) is quite close to what embedding layers do in neural nets.



An embedding layer is matrix multiplication:

$$\underbrace{h_1}_{n_E \times 1} = \underbrace{\omega_E}_{n_E \times n_w} \cdot \underbrace{x}_{n_x \times 1}$$

- x = a categorical variable (e.g., representing a word)
    - one-hot vector with a single item equaling one. Input to the embedding layer.
- h_1 = the first hidden layer of the neural net
    - The output of the embedding layer.

The embedding matrix ω_E encodes predictive information about the categories, has a spatial interpretation when projected to two dimensions.

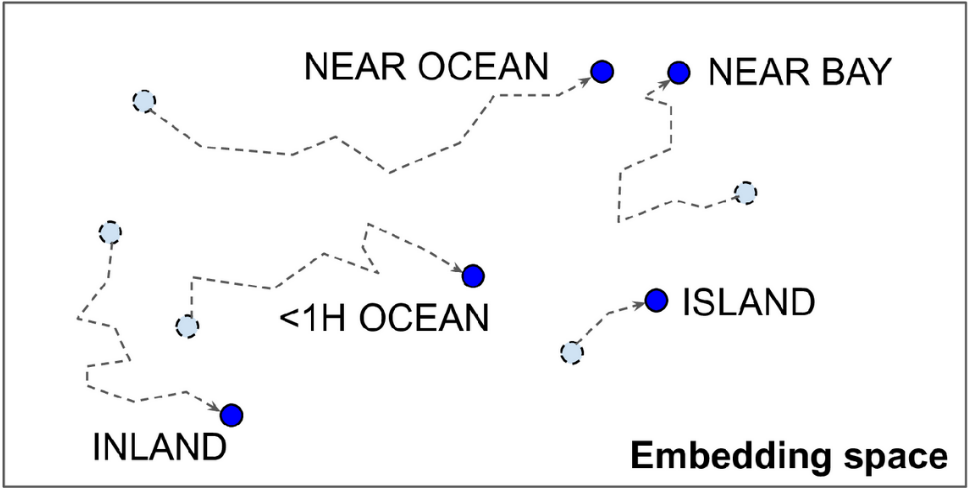


Figure 13-4. Embeddings will gradually improve during training

Embedding Layers versus Dense Layers

- An embedding layer is statistically equivalent to a fully-connected dense layer with one-hot vectors as input and linear activation.
  - embedding layers are much faster for many categories ($>\sim$50)

Outline

RNNs can input and output arbitrary-length sequences

- The models we have looked at so far (including CNNs) took inputs of fixed dimensions and output scalars or class probabilities.

RNNs can input and output arbitrary-length sequences

- The models we have looked at so far (including CNNs) took inputs of fixed dimensions and output scalars or class probabilities.
- Recurrent Neural Nets (RNNs) work with **sequences of arbitrary length**, both as **inputs** and **outputs**:
  - can *encode* sequences into vectors.
  - can *decode* vectors into sequences.
- therefore especially useful for language tasks such as translation.

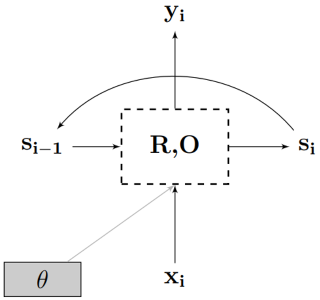
RNN Architecture

- At each step t:
  - a recursion function $R(s_{t-1}, x_t)$ computes the state vector s_t given current word x_t and previous state s_{t-1}. s_0 initialized to zeros or randomly.

RNN Architecture

- At each step t:
  - a recursion function $R(s_{t-1}, x_t)$ computes the state vector s_t given current word x_t and previous state s_{t-1}. s_0 initialized to zeros or randomly.
  - An output function $O(s_t)$ computes an output vector y_t (to be compared to the output data).

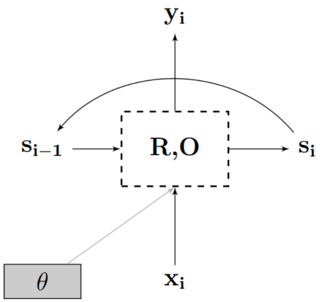
$$\hat{\boldsymbol{y}}_t = O(\boldsymbol{s}_t)$$
$$\boldsymbol{s}_t = R(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t)$$



RNN Architecture

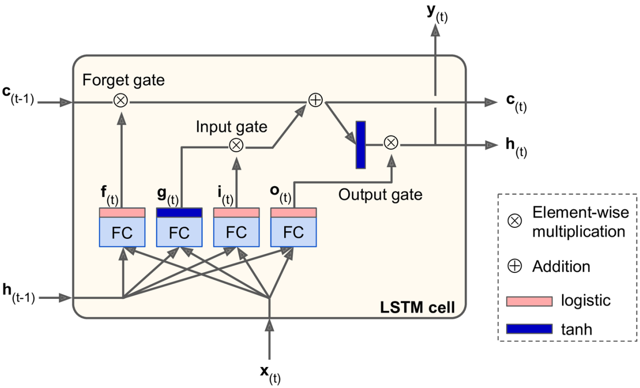
- At each step t:
  - a recursion function $R(s_{t-1}, x_t)$ computes the state vector s_t given current word x_t and previous state s_{t-1}. s_0 initialized to zeros or randomly.
  - An output function $O(s_t)$ computes an output vector y_t (to be compared to the output data).

$$\hat{\boldsymbol{y}}_t = O(\boldsymbol{s}_t)$$
$$\boldsymbol{s}_t = R(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t)$$



- in a "Simple RNN", $R(\cdot)$ is just a dense layer with ReLU activation, and $O(s_t) = s_t$.

Gated Architectures – LSTM (Long Short-Term Memory)



- gating mechanisms prevent vanishing/exploding gradients (see also GRU).

RNNs: Practical Use for Sequence-to-Vector Task

For example, sentiment analysis (using keras functional API):

- tokenize the documents into $\boldsymbol{w}_{1:n}$
- embedding $\boldsymbol{x}_t = \boldsymbol{w}_t \cdot \boldsymbol{\omega}_E$
  - embedding matrix $\boldsymbol{\omega}_E$ can be initialized with pre-trained GloVe embeddings.
- bidirectional LSTM on $\boldsymbol{x_{1:n}}$ (and $\boldsymbol{x_{n:1}}$) to generate document vector $\boldsymbol{s}$
  - that is, document is fed in backwards and forwards to two parallel LSTMs,
  - Use layer normalization (normalizes each instance across feature dimensions) rather than batch normalization (normalizes each feature across a sample of instances).
- $\boldsymbol{s}$ input to MLP to predict $\boldsymbol{y}$

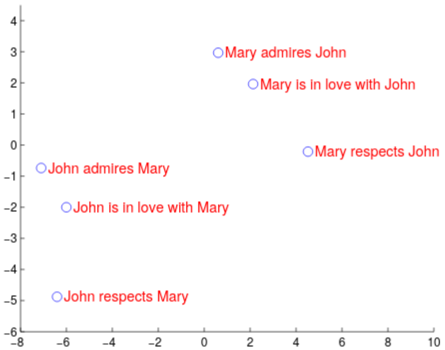
see Richard Socher's lecture notes for some additional tips re initialization, learning rate, and gradient clipping.

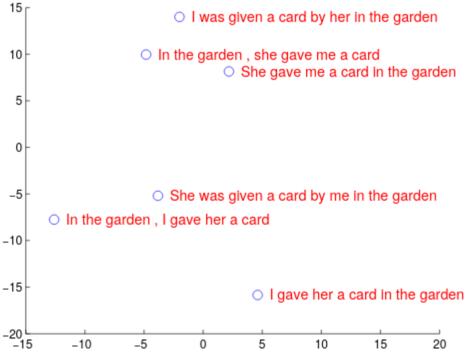
Geometry of Embedded Sequences

- The RNN's document vector for a sequence, $\boldsymbol{s}$, is an embedding with interpretable geometry.

Geometry of Embedded Sequences

- The RNN's document vector for a sequence, $\boldsymbol{s}$, is an embedding with interpretable geometry.





Sutskever, Vinyals, and Le, "Sequence to sequence learning with neural networks."

Types of RNNs: Encoding and Decoding

top left: sequence to sequence; top right: sequence to vector

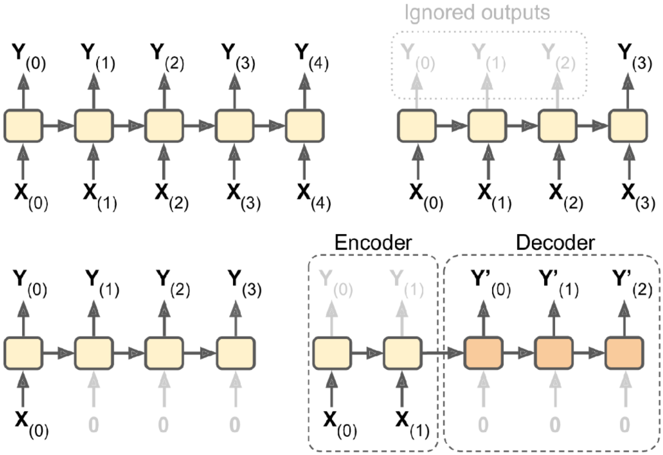


Figure 15-4. Seq-to-seq (top left), seq-to-vector (top right), vector-to-seq (bottom left), and Encoder–Decoder (bottom right) networks

bottom left: vector to sequence; bottom right: encoder-decoder.

Application: RNN's for predicting partisanship

Iyyer, Enns, Boyd-Graber, and Resnik (2014)

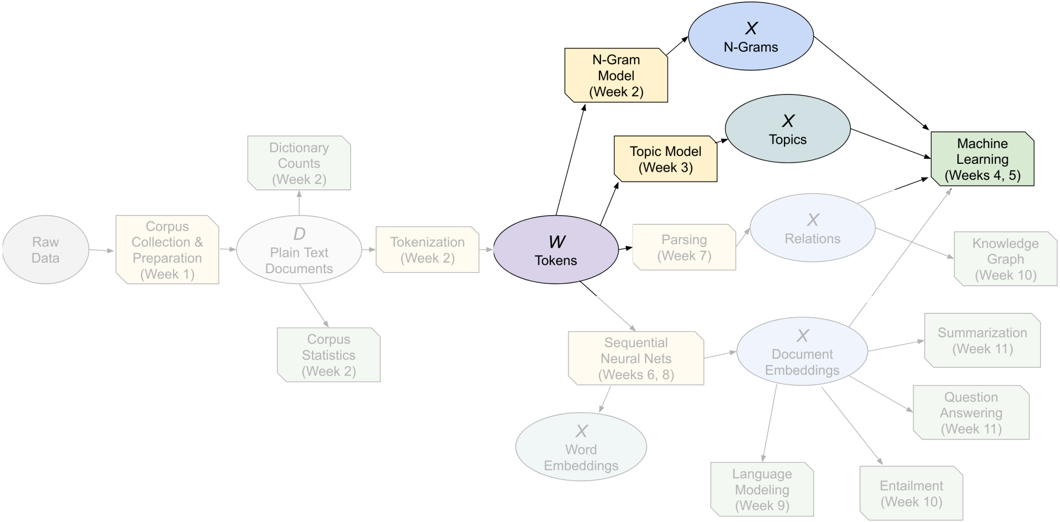
Application: RNN's for predicting partisanship

Iyyer, Enns, Boyd-Graber, and Resnik (2014)

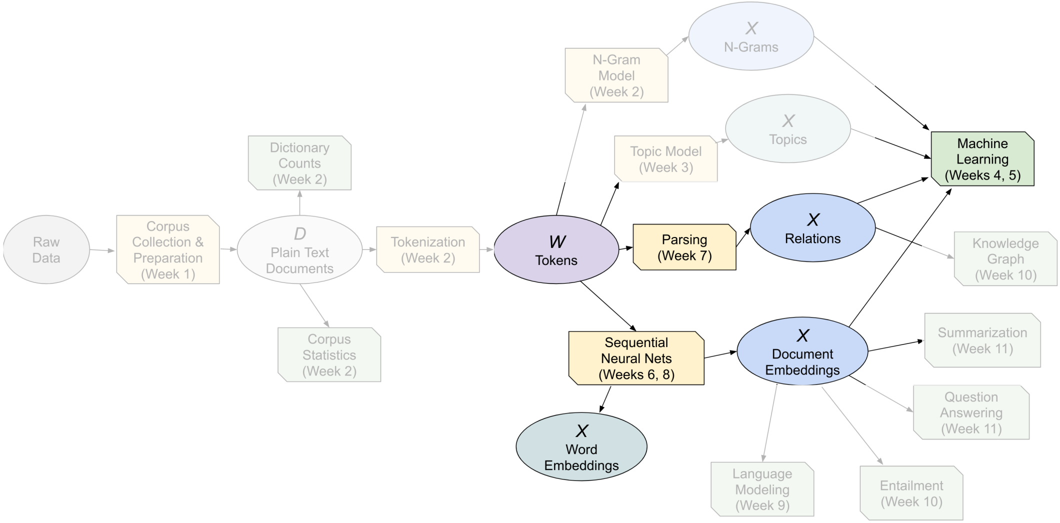
n	Most conservative n-grams	Most liberal n-grams
1	Salt, Mexico, housework, speculated, consensus, lawyer, pharmaceuticals, ruthless, deadly, Clinton, redistribution	rich, antipsychotic, malaria, biodiversity, richest, gene, pesticides, desertification, Net, wealthiest, labor, fertilizer, nuclear, HIV
3	prize individual liberty, original liberal idiots, stock market crash, God gives freedom, federal government interference, federal oppression nullification, respect individual liberty, Tea Party patriots, radical Sunni Islamists, Obama stimulus programs	rich and poor,"corporate greed", super rich pay, carrying the rich, corporate interest groups, young women workers, the very rich, for the rich, by the rich, soaking the rich, getting rich often, great and rich, the working poor, corporate income tax, the poor migrants
5	spending on popular government programs, bailouts and unfunded government promises, North America from external threats, government regulations place on businesses, strong Church of Christ convictions, radical Islamism and other threats	the rich are really rich, effective forms of worker participation, the pensions of the poor, tax cuts for the rich, the ecological services of biodiversity, poor children and pregnant women, vacation time for overtime pay
7	government intervention helped make the Depression Great, by God in His image and likeness, producing wealth instead of stunting capital creation, the traditional American values of limited government, trillions of dollars to overseas oil producers, its troubled assets to federal sugar daddies, Obama and his party as racialist fanatics	African Americans and other disproportionately poor groups; the growing gap between rich and poor; the Bush tax cuts for the rich; public outrage at corporate and societal greed; sexually transmitted diseases , most notably AIDS; organize unions or fight for better conditions, the biggest hope for health care reform

Table 2: Highest probability n-grams for conservative and liberal ideologies, as predicted by the **RNN2-(W2V)** model.

Course Progress (Weeks 2-5)



Course Progress (Weeks 5-8)



Activity: Zoom Poll 5.2