Part V

Advanced Topics

16
Machine Learning and Ethics

Throughout this course, we have discussed the technical aspects of model design, training, and testing in depth. However, we have not yet discussed some of the social implications of this technology. What are some ethical and legal issues in deployment of ML techniques in society? What are the caveats and limitations to temper our exuberance about the possibilities of ML? This brief chapter addresses these issues, and we hope as technologists you will continue to investigate and consider such issues throughout your career.

16.1 *Facebook's Suicide Prevention*

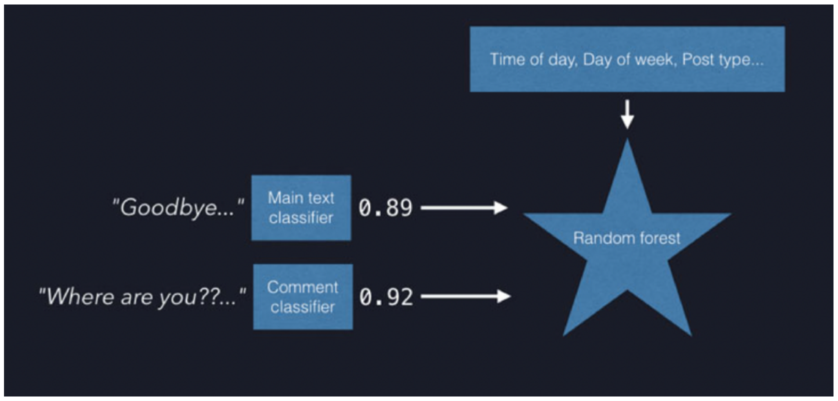


Figure 16.1: A visualization of the Facebook model to predict suicides.

In 2017, Facebook launched a program to use a machine learning algorithm to predict suicide risk amongst its user population. It has continued with various iterations over the years. Figure 16.1 gives a visualization of the four-step process:

1. ML algorithm automatically analyzes a post by processing its text content and comments

2. Algorithm additionally uses spatial-temporal context of the post to perform a risk prediction

3. A human behind the algorithm performs a personal review to finally verify if a threshold is reached

4. If the post is poses a serious risk, Facebook performs a wellness check through the person's contacts, community organizations, etc.

At first sight, this may appear to be very good idea: even if it saves just one life, surely the project is worth it? But the announcement of the project cause a lot of controversy among people. The following are some of the potential problems that people identified:

1. False positives may result in stigmatization.

2. Many people who contemplate suicide do not end up going through with it. Facebook's reporting could lead to criminal penalties (in regions where suicide is a crime), involuntary hospitalization, stigmatization etc.

3. Involvement of authorities (*e.g.*, law enforcement) raises risk of illegal seizures

4. Should Facebook be liable for any problem caused by misdetection?

Beyond these points, there are deep philosophical questions associated with the concept of suicide as well. For instance, is suicide actually immoral? Even if it is immoral, is it the responsibility of Facebook to get involved? Is it moral for Facebook to use personal information to assess suicide risk? Opinions differ.

16.2 *Racial Bias in Machine Learning*

Suppose we are designing a machine learning approach for loan approval. The general approach will be to take a dataset of $(\vec{\mathbf{x}}, y)$, where $\vec{\mathbf{x}}$ is a vector of the individual's attributes (*e.g.*, age, education, alma mater, address, etc.) who got a loan and $y \in \{-1, 1\}$ indicates whether they actually paid off the loan or not. Using the approaches we learned in Section 4.2, we could train a binary classifier through logistic regression. Civil rights legislation forbids using the individual's race in many of these decisions, so while training we could simply mask out any coordinates which identify race. However, this does not guarantee that the classifier will be entirely "race-neutral." [1]

[1] The reason is that race happens to be correlated with many other attributes. Thus if a classifier uses any of the correlated attributes, it may be implicitly using racial information in the decision making process.

In 2016, a study [2] found that COMPAS, a leading software for assessing the probability that a prison inmate would commit another

[2] *Machine Bias*, by Anwin et al., in *Pro Publica* 2016.

serious crime, disproportionately tags African-American as being likely to commit crimes — in the sense that African-Americans who were tagged as likely to commit another crime were only half as likely to actually commit a crime than a similarly-tagged person of another race.

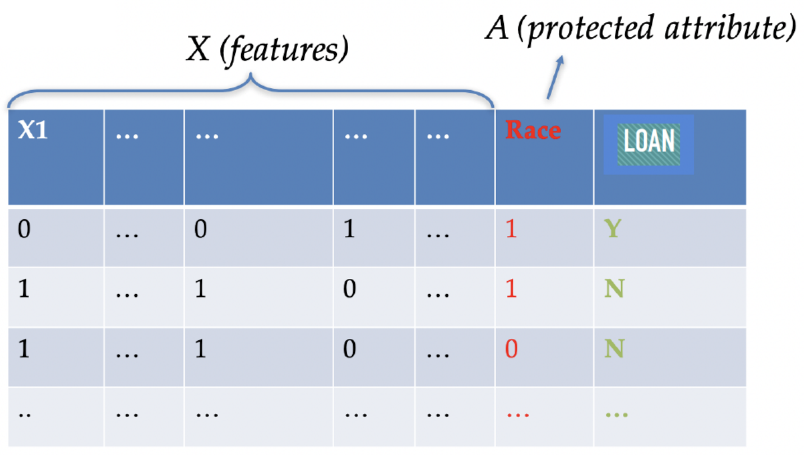
	White	**African-American**
Labeled Higher Risk & Did *Not* Re-offend	23.5%	44.9%
Labeled Lower Risk & *Did* Re-offend	47.7%	28.0%

Table 16.1: COMPAS correctly predicts recidivism 61 percent on average. But African-Americans are almost twice as likely as whites to be labeled a higher risk but not actually re-offend. Conversely, whites are twice as likely as African-Americans to be labeled lower risk but go on to commit other crimes.

16.3 *Conceptions of Fairness in Machine Learning*

We will briefly consider possible ways to formulate fairness in machine learning. Keep in mind that this task is intrinsically difficult, as we are attempting to assign a technological perspective to a fundamentally normative problem. The first property we might want a ML classifier to have is called *demographic parity*, which effectively enforces that the output of classifier does not depend on a protected attribute (*e.g.*, race, ethnicity, gender).

Definition 16.3.1 (Demographic Parity). *We say that a binary classifier that outputs $y \in \{-1, 1\}$ satisfies **demographic parity** if* $\Pr[y \mid x_i = a] = \Pr[y \mid x_i = b]$ *where a, b are any two values that a protected attribute x_i can take.*


X (features)
A (protected attribute)

X1	…	…	…	…	Race	LOAN
0	…	0	1	…	1	Y
1	…	1	0	…	1	N
1	…	1	0	…	0	N
..	…	…	…	…	…	…

Figure 16.2: A hypothetical application of ML to a loan approval application. *Race* has been made a protected attribute in an attempt to prevent bias during training.

A visualization of how a protected attribute could be specified in a dataset is shown in Figure 16.2. Consider the loan approval example from the previous section. If the binary classification model for the loan approval satisfies the demographic parity property, then the model approve loans for different races at similar rates. One way to achieve this condition is to use a regularizer term $\lambda(\Pr[y \mid x_i = a] - \Pr[y \mid x_i = b])^2)$ during training. [3]

[3] Does this seem like a good formulation of fairness?

Another property we want a "fair" model to satisfy is called the *predictive parity*. This is the property that the model in Table 16.1 failed to satisfy.

Definition 16.3.2 (Predictive Parity)**.** *We say that a binary classifier that outputs $y \in \{-1, 1\}$ satisfies **predictive parity** if the true negative/false negative/false positive/true positive rates are the same for any values of a sensitive attribute.*

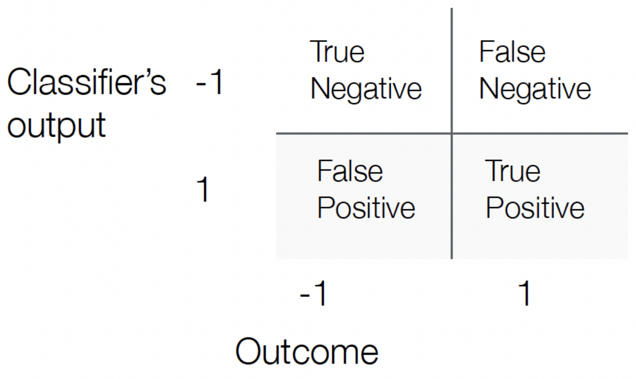


Figure 16.3: A table of all possible outcomes based on the model output and the ground truth outcome. This is also known as a *confusion matrix*.

Ideally, we want a ML model to satisfy both the demographic parity and predictive parity. However, it turns out that these two notions are incompatible!

Theorem 16.3.3 (Fairness Impossibility Theorem)**.** [4] *Under fairly general conditions,* demographic parity *and* predictive parity *are incompatible.*

[4] See *Inherent Trade-Offs in the Fair Determination of Risk Scores*, Kleinberg, Mullainathan, and Raghavan, *ITCS* 2017. The paper actually considered three possible definitions of "fairness" and showed every pair of them are mutually incompatible.

There are other formulations of fairness, but it is difficult to find a combination of these notions that are compatible with each other. So one way or another, we need to sacrifice some notions of "fairness."

16.4 *Limitations of the ML Paradigm*

The predictive power of ML seems immense, but is it true that if we have enough data and the right algorithm, then everything become predictable? If yes, then one could imagine societal programs leveraging this to precisely target help to where it would be more effective. We first consider a famous — and somewhat amusing — example of a study [5] that turned out to be false.

[5] *Extraneous factors in judicial decisions*, Danziger et al., *PNAS* 2011.

16.4.1 *Hungry Judge Effect*

The study analyzed the parole decisions made by 8 Israeli judges in over 1,100 cases. The data in Figure 16.4 shows that prisoners were much more likely to be granted parole after the judge took a lunch break or a coffee break. The study therefore suggested that judges tend to be stricter before a break (maybe because they are "hangry") but more lenient when they return from the break.

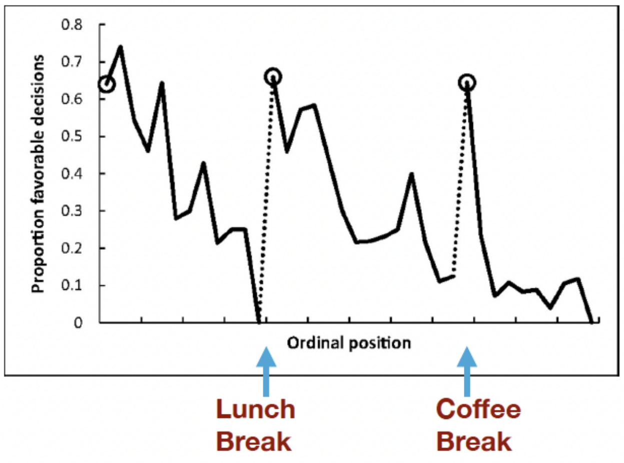


Figure 16.4: Data from the study shows an uptick in favorable decisions following a lunch break or a coffee break.

Nevertheless, it turns out that this "hungry judge effect" can be explained by a completely different reason. A followup study [6] found that the ordering of cases presented to the judge was not random: prisoners with attorneys were scheduled at the beginning of each session, while prisoners without an attorney were scheduled at the end of a session. The former group were let on parole with a rate of 67%, while the rate was just 39% for those without attorneys. Another important observation was that attorneys tended to present their cases in decreasing order of strength of case, with the average attorney having 4.1 clients. Computer simulations of hunger-immune judges faced with cases presented according to these percentages showed the same see-saw effect of Figure 16.4.

[6] *Overlooked factors in the analysis of parole decisions*, Weinshall-Margel and Shepard, *PNAS*, 2012.

16.4.2 *Fragile Families Challenge*

The Fragile Families Challenge is a collaborative project initiated by the Center for Research on Child Wellbeing at Princeton University. A brief description of the initiative's motivation is provided on the website: [7]

The Fragile Families Challenge is a mass collaboration that combines predictive modeling, causal inference, and in-depth interviews to yield insights that can improve the lives of disadvantaged children in the United States.

[7] Source: `https://www.fragilefamilieschallenge.org`.

By working together, we can discover things that none of us can discover individually.

The Fragile Families Challenge is based on the Fragile Families and Child Wellbeing Study, which has followed thousands of American families for more than 15 years. During this time, the Fragile Families study collected information about the children, their parents, their schools, and their larger environments.

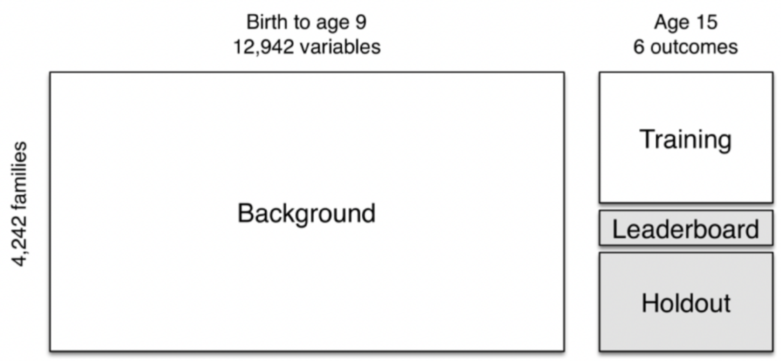


Figure 16.5: Diagram illustrating the dataset of the Fragile Families Challenge.

The initiative has collected immense data on multiple families, including interviews with mothers, fathers, and/or primary caregivers at several ages. Interviewees were inquired as to attitudes, relationships, parenting behavior, economic and employment status, etc. Additionally, in-home assessments of children and their home environments were performed to assess cognitive and emotional development, health, and home environment. The goal was to predict six key outcomes at age 15 (*e.g.*, whether or not the child is attending school) given background data from birth to age 9 as shown in 16.5. However, up to this point no method has done better than random guessing.

This is food for thought: what is going on?

16.4.3 *General Limits to Prediction*

Matt Salganick and Arvind Narayanan, professors at Princeton University, recently started a course [8] which aims to explore the extent to which interdisciplinary problems in social science and computer science can be predictable. In general, following are some major themes that can make prediction difficult:

[8] The course, COS 597E/SOC 555 is a seminar first offered in Fall 2020.

1. The distribution associated with data can shift over time
2. The relationship between input data and desired outputs can change over time

3. There is a possibility for unknown coordinates to be unintentionally ignored (*i.e.*, as in the hungry judge effect)

4. The "8 billion problem," which outlines how data available in the real world is fundamentally finite and limited

16.5 Final Thoughts

As described in the preceding sections, users and designers of machine learning will often face ethical dilemmas. Designers may have to operate without moral clarity or easy technical fixes. In fact, technical solutions may even be impossible. To appropriately acknowledge these limitations, it is important to embrace a culture of measuring and openly discussing the impact of the system being built. Indeed, a general principle to follow is to avoid harm when trying to do good.

17 Deep Learning for Natural Language Processing

17.1 Word Embeddings

In traditional NLP, each word is regarded as discrete symbols each with a single value of weight. For example, in Chapter 1, we learned how to use linear regression on sentiment prediction. But with this approach, it is hard for the computer to learn the *meaning* of the word; instead, each of the words remain as some abstract symbols with numeric weights.

But how do computers know the meaning of words? We can easily think of one solution: we can look up words in a dictionary. For example, WordNet is a project that codes the meaning of the words and the relationship between the words, so that the data can be used for computers to parse. [1] But resources like WordNet require human labor to create and adapt, and it is impossible to keep up-to-date (because new words are coined up and new meanings appear out of existing words).

[1] For more information, check http://wordnetweb.princeton.edu.

An alternative approach is to represent words as short (50 - 300 dimensions [2]), real-valued vectors. These vectors encode the meaning and other properties of words. In this representation, the distance between vectors represent the *similarity* between words. This vectorized form of the words are much easier to be used as inputs in modern ML systems (especially neural networks). This vector form of words is known as *word embedding*. In this section, we explore the process of how to learn a good word embedding.

[2] The dimension of word vectors is a hyperparameter that needs to be decided first.

17.1.1 Distributional Hypothesis

Word embedding is based on a concept called the *distributional hypothesis*, a theory developed by John Rupert Firth. The hypothesis, one of the most successful ideas of modern statistical NLP, says that words that occur in similar contexts tend to have similar meaning.

Definition 17.1.1 (Context). *When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).*

Example 17.1.2. *Assume that you first heard the word **tejuino** and have no idea what the word means. But you learn that the word may appear in the following four contexts.*

- *C1: A bottle of ____ is on the table.*
- *C2: Everybody likes ____.*
- *C3: Don't have ____ before you drive.*
- *C4: We make ____ out of corn.*

Based on these contexts, it is reasonable to conclude that the word "tejuino" refers to some form of alcoholic drink made from corn.

Problem 17.1.3. *To find words with similar meanings as "tejuino," we tried filling out the contexts from Example 17.1.2 with 5 other words. The results are given in Table 17.1, where 1 means that the word was appropriate to be used in that context, and 0 means that it was inappropriate.*

	C1	C2	C3	C4
tejuino	1	1	1	1
loud	0	0	0	0
motor-oil	1	0	0	0
tortillas	0	1	0	1
choices	0	1	0	0
wine	1	1	1	0

Table 17.1: Data showing if 6 words are appropriate for the four contexts in Example 17.1.2.

Which word is closest to "tejuino"?

17.1.2 Word-word Co-occurrence Matrix

Given a very large collection of documents with words from a dictionary V, we construct a $|V| \times |V|$ matrix X, where the entry at the i-th row, j-th column denotes the number of times (*i.e.*, frequency) that w_j appears in the context window of w_i. This matrix is called the *word-word co-occurrence matrix*.

Example 17.1.4. *Table 17.2 shows a portion of a word-word co-occurrence matrix. Each row corresponds to the* center word w_i, *and each column corresponds to the* context word w_j. *The value X_{ij} at the (i, j) entry means that the context word w_j appeared X_{ij} times in the context (of length 4) of w_i in total.*

Although the portion shown in Table 17.2 mostly has non-zero entries, in general, the entries of the matrix are mostly zero.

	···	computer	data	result	pie	sugar	···
cherry	···	2	8	9	442	25	···
strawberry	···	0	0	1	60	19	···
digital	···	1670	1683	85	5	4	···
information	···	3325	3982	378	5	13	···

Table 17.2: A portion of a word-word co-occurrence matrix for a corpus of Wikipedia articles. Source: `https://www.english-corpora.org/wiki/`.

17.1.3 *Factorization of Word-word Co-occurrence Matrix*

Recall the example of movie recommendation through matrix factorization in Chapter 9. In that example $m \times n$ matrix M was factorized into $M \approx AB$ where the i-th row of A was a d-dimensional vector that represented user i and the j-th column of B was a d-dimensional vector that represented movie j.

We can imagine a similar factorization on the word-word co-occurrence matrix. That is, we can represent each *center word* and each *context word* as a d-dimensional vector such that $X_{ij} \approx A_{i*} \cdot B_{*j}$. But this particular idea does not work on the word-word co-occurrence matrix. The key difference is that X is a complete matrix with no missing entries (although most entries are zero). Therefore we instead use other standard matrix factorization techniques (*e.g.*, Singular Value Decomposition).

One popular choice of factorization is running the Singular Value Decomposition (SVD) on a weighted co-occurrence matrix. [3] This idea originates from a concept called *Latent Semantic Anlysis*. [4] If the SVD returns the following decomposition,

[3] The particular weighting scheme is called *PPMI*. We will not get into the detail here.
[4] From *Indexing by Latent Semantic Analysis* by Deerwester et al., 1990.

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix} = \begin{bmatrix} & & \\ & W & \\ & & \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_d \end{bmatrix} \begin{bmatrix} & & \\ & W^\intercal & \\ & & \end{bmatrix}$$

where X is a $|V| \times |V|$ matrix and W is a $|V| \times d$ matrix, then the i-th row of matrix W can be regarded as the embedding for word w_i.

Other modern approaches tend to treat word vectors as parameters to be optimized for some objective function and apply the gradient descent algorithm. But the principle is the same: "words that occur in similar contexts tend to have similar meanings." Some of the popular algorithms with this approach include: *word2vec* (Mikolov et al., 2013), *GloVe* (Pennington et al., 2014), and *fastText* (Bojanowski et al., 2017).

Here we briefly explain the GloVe algorithm. Given the co-occurrence table X, we will construct a *center word vector* $\vec{\mathbf{u}}_i \in \mathbb{R}^d$ and a *context word vector* $\vec{\mathbf{v}}_j \in \mathbb{R}^d$ such that they optimize the following objective:

$$J(\theta) = \sum_{i,j \in V} f(X_{ij}) \left(\mathbf{u}_i \cdot \mathbf{v}_j + b_i + \widetilde{b_j} - \log X_{ij} \right)^2 \tag{17.1}$$

where f is some non-linear function and $b_i, \widetilde{b_j}$ are bias terms. This is within the same line of logic as optimizing

$$L(A, B) = \frac{1}{|\Omega|} \sum_{i,j \in \Omega} (M_{ij} - (AB)_{ij})^2 \qquad \text{((9.5) revisited)}$$

17.1.4 Properties of Word Embeddings

A good word embedding should represent the meaning of the words and their relationship with other words as accurately as possible. Therefore there are some properties that we would like a word embedding to preserve. We will discuss three such properties and see how the current algorithms for word embedding perform on preserving those properties.

1. Similar words should have similar word vectors: This is the most important property we can think of.

Example 17.1.5. *In a certain word embedding, the following is the list of* 9 *most nearest words to the word "sweden."*

Word	*Cosine distance*
norway	*0.760124*
denmark	*0.715460*
finland	*0.620022*
switzerland	*0.588132*
belgium	*0.585835*
netherlands	*0.574631*
iceland	*0.562368*
estonia	*0.547621*
slovenia	*0.531408*

Notice Scandanavian countries are the top 3 *entries on the list, and the rest are also European country names.*

2. Vector difference should encode the relationship between words: If there are two or more pairs of words where each pair of words are distinguishable by the same attribute, you can imagine that the vector difference within each pair is nearly the same.

Example 17.1.6. *In Figure 17.1, notice that $v_{man} - v_{woman} \approx v_{king} - v_{queen}$. The vector difference in common can be understood as representing the male-female relationship. Similarly, there seems to be a common vector difference for representing the difference in verb tense.*

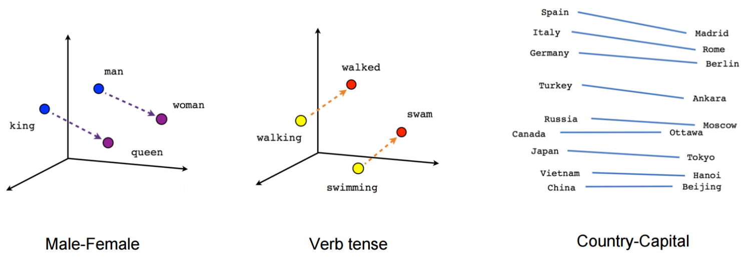


Figure 17.1: Two pairs of words that differ in the same attribute show a similar difference in their word embeddings.

3. *The embeddings should be translated between different languages:* When we independently find the word embedding in different languages, we can expect to have a bijective mapping that preserves the structure of the words in each language. [5]

[5] From *Exploiting Similarities among Languages for Machine Translation* by Mikolov et at., 2013.

Example 17.1.7. *In Figure 17.2, notice that if we let W to be the mapping from English to Spanish word embeddings,* $v_{cuatro} \approx W \circ v_{four}$

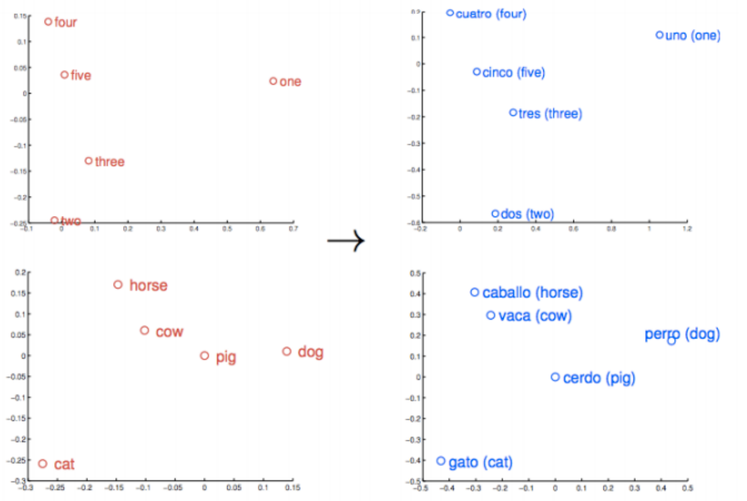


Figure 17.2: Word embeddings are translated into the embeddings of other languages.

17.2 N-gram Model Revisited

Recall the n-gram model from Chapter 8. It assigned a probability $\Pr[w_1 w_2 \ldots w_n]$ to every word sequence $w_1 w_2 \ldots w_n$. We discussed the concept of perplexity of the model to compare the performance of unigram, bigram, and trigram models. While the n-gram model is impressive, it has obvious limitations.

Problem 17.2.1. *"The students opened their ______." Can you guess the next word?*

Problem 17.2.2. *"As the proctor started the clock, the students opened their ______." Can you guess the next word?*

In a lot of cases, words in a sentence are closely related to other words and phrases that are far away. But the n-gram model cannot look beyond the specified frame.

Example 17.2.3. *The following is a text generated by a 4-gram model*

> *Today the price of gold per tan, while production of shoe lasts and shoe industry, the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks, sept 30 and primary 76 cts a share.*

The generated text is surprisingly grammatical, but incoherent.

Example 17.2.3 shows that we need to consider more than three words at a time if we want to model language well. But if we use a larger value of n for the n-gram model, the data will become too sparse to estimate the probabilities. But even when we restrict ourselves to words that appear in the dictionary, there are 10^{21} distinct sequences of 4 words.

17.2.1 Feedforward Neural Language Model

The idea of *feedforward neural language model* was proposed by Bengio et al. in 2003 in a paper called *A Neural Probabilistic Language Model*. The intuition is to use a neural network to learn the probabilistic distribution of language, instead of estimating raw probabilities. The key ingredient in this model is the word embeddings we discussed earlier.

Example 17.2.4. *Assume we are given two contexts "You like green ______" and "You like yellow ______" to fill the blanks in. A n-gram model will try to calculate the raw probabilities* $\Pr[w \mid \textit{You like green}]$ *and* $\Pr[w \mid \textit{You like yellow}]$*. However, if the word embeddings showed that* $v_{green} \approx v_{yellow}$*, then we can imagine that the two contexts are similar enough. Then we may be able to estimate the probabilities better.*

Now we show how to use feedforward neural language model on a n-gram model. Assume we want to estimate the probability $\Pr[w_{n+1} \mid w_1 \dots w_n]$. Then the first step is to find a find a word embedding

$$v_1, v_2, \dots, v_n \in \mathbb{R}^d$$

of each word $w_1, w_2, \dots, w_n$. Then we concatenate the word embeddings into [6]

$$\vec{\mathbf{x}} = (v_1, \dots, v_n) \in \mathbb{R}^{nd}$$

[6] the order of the input vectors cannot change

This will be the input layer. Then we define the fully connected hidden layer as

$$\vec{\mathbf{h}} = \tanh(\mathbf{W}\vec{\mathbf{x}} + \vec{\mathbf{b}}) \in \mathbb{R}^h$$

where $\mathbf{W} \in \mathbb{R}^{h \times nd}$ and $\vec{\mathbf{b}} \in \mathbb{R}^h$. Then we define the output layer as

$$\vec{\mathbf{z}} = \mathbf{U}\vec{\mathbf{h}} \in \mathbb{R}^{|V|}$$

where $\mathbf{U} \in \mathbb{R}^{|V| \times h}$. Then finally, the probability will be calculated with the softmax function:

$$\Pr[w = i \mid w_1 \ldots w_n] = \text{softmax}_i(\vec{\mathbf{z}}) = \frac{e^{z_i}}{\sum\limits_{k \in V} e^{z_k}}$$

So the total number of parameters to train in this network is

$$d\,|V| + ndh + h + h\,|V|$$

where the terms are respectively for the input embeddings, $\mathbf{W}, \vec{\mathbf{h}}, \mathbf{U}$. When $d = h$, sometimes we tie the input and output embeddings. That is, we can consider $\mathbf{U}$ to be the parameters required for the output embeddings. At this point, the language model reduces to a $|V|$-way classification, and we can create lots of training example by sliding the input-output indices. That is, when given a huge text, we can create lots of input-output tuple as follows: $((w_1, \ldots, w_n), w_{n+1}), ((w_2, \ldots, w_{n+1}), w_{n+2}), \ldots$.

17.2.2 *Beyond Feedforward Neural Language Model*

But the feedforward language model still has its limitations. The main reason is that $\mathbf{W} \in \mathbb{R}^{h \times nd}$ scales linearly with the window size. Of course, this is better than the traditional n-gram model which scales exponentially with n. Another limitation of the neural LM is that the model learns separate patterns for the same item. That is, a substring $w_k w_{k+1}$, for example, will correspond to different parameters in $\mathbf{W}$ when trained on $(w_k w_{k+1} \ldots w_{k+n-1})$ or on $(w_{k-1} w_k \ldots w_{k+n-2})$.

To mitigate these limitations, we can choose to use similar modeling ideas but use better and bigger neural network architectures like *recurrent neural networks (RNN)* or *transformers*.

Here we briefly explain the core ideas of a RNN. RNNs are a family of neural networks that handle variable length inputs. Whereas feedforward NNs map a fixed-length input to a fixed-length output, recurrent NNs map *a sequence* of inputs to *a sequence* of outputs. The sequence length can vary and the key is to *reuse the weight matrices* at different time steps. When the inputs are given as $\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots \vec{\mathbf{x}}_T \in \mathbb{R}^d$ and we want to find outputs $\vec{\mathbf{h}}_1, \vec{\mathbf{h}}_2, \ldots \vec{\mathbf{h}}_T \in \mathbb{R}^h$, we train the parameters

$$\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}, \vec{\mathbf{b}} \in \mathbb{R}^h$$

such that

$$\vec{\mathbf{h}}_t = g(\mathbf{W}\vec{\mathbf{h}}_{t-1} + \mathbf{U}\vec{\mathbf{x}}_t + \vec{\mathbf{b}}) \in \mathbb{R}$$

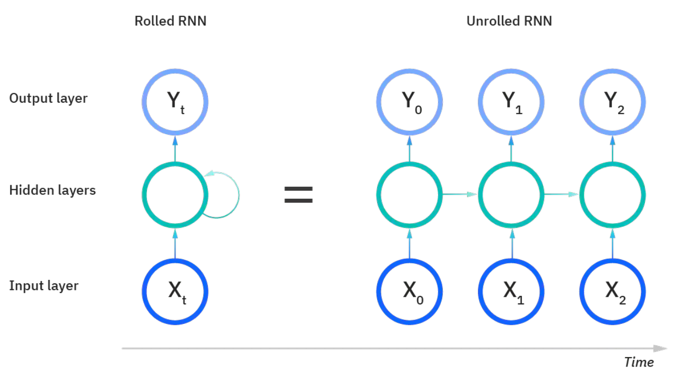


Figure 17.3: A visual representation of an RNN architecture.

where g is some non-linear function (*e.g.*, ReLU, tanh, sigmoid). We can also set $\vec{\mathbf{h}}_0 = \vec{\mathbf{0}}$ for simplicity.