

# Applied Causal Inference Powered by ML and AI

Victor Chernozhukov\*

Christian Hansen<sup>†</sup>

Nathan Kallus<sup>‡</sup>

Martin Spindler<sup>§</sup>

Vasilis Syrgkanis<sup>¶</sup>

February 28, 2024

Publisher: Online

Version 0.1.1

\* MIT

<sup>†</sup> Chicago Booth

<sup>‡</sup> Cornell University

<sup>§</sup> Hamburg University

<sup>¶</sup> Stanford University

# Feature Engineering for Causal and Predictive Inference

# 11

"It's all about paying attention. [...] Attention is vitality. It connects you with others."

– Susan Sontag [1].

11.1 Introduction . . . . .	288
11.2 From Principal Components to Autoencoders . . . . .	289
11.3 From Auto-Encoders to Gen- eral Embeddings . . . . .	294
11.4 Text Embeddings . . . . .	295
Revisiting the Price Elastic- ity for Toy Cars . . . . .	305
11.5 Image Embeddings . . . . .	306
Application: Hedonic Prices . . . . .	307

Here we discuss feature engineering as an approach to transform complex objects such as text and images into a collection of relatively low-dimensional numerical features (embeddings) that can be used for standard predictive or causal applications, for example as regressors in a prediction problem. We consider principal components, variational autoencoders and neural networks as general approaches to generate embeddings. We then consider text embeddings in detail, introducing two popular neural network-based Natural Language Processing (NLP) algorithms: ELMo and BERT. We finally consider image embeddings, applying a hedonic price model to apparel data using a neural network algorithm (ResNet50) to generate embeddings.

## 11.1 Introduction

Thus far, we have imposed a significant restriction on the kinds of data on which we can perform inference. While empiricists often consider simple datasets that include variables that have a numeric representation (binary, factor and continuous variables), researchers are increasingly confronted with complex forms of data, such as images and text, that encode a vast amount of information. In this section, we generalize our approach to allow using these types of data.

As a motivating example, we consider the problem of predicting prices of products using the types of characteristics that one might find on a webpage, namely the text in the product description and the product's image. The resulting predicted prices are called hedonic prices, and predictive modeling of this form is motivated by the hedonic price models of economics.

In order to predict prices, we have to convert text and images into relatively low-dimensional numerical features, called "embeddings." The minimal requirement on embeddings is that similar products should have similar embeddings. This requirement guarantees that price predictions for similar products are also similar. The maximal requirement on embeddings is that they should parsimoniously approximate maximal information from text and images that is relevant for price predictions.

The main methods for generating successful embeddings include the following, in order of increasing generality:

- ▶ classical principal component analysis,
- ▶ auto-encoders, and
- ▶ neural networks solving auxiliary prediction tasks.

The auxiliary tasks in the final method may include solving image processing problems, such as object classification and image compression, or natural language processing problems, such as summarization and machine translation.

These auxiliary tasks are not the same as the "main" task. In our price prediction example, the main task is predicting product prices. Before turning to the primary price prediction task, we consider ResNet-50, which is a Residual Network of depth 50, which is designed to perform well on various tasks of object type classification. Consequently, application of ResNet-50 produces embeddings that are useful inputs for solving this auxiliary object classification task. However, because product type is an important determinant of price, the embeddings produced by

ResNet-50 that help classify products can also serve as useful inputs to the main task – price prediction.

Analogously, a neural network such as BERT is trained on auxiliary tasks aimed to make it learn word similarity and contextual meaning of words. Consequently, BERT can produce embeddings that provide a useful numerical summary of a product's text description. Because the product description is an important determinant of the price, these embeddings can also serve as useful inputs to the price prediction task.

Embeddings are useful in a variety of predictive and causal inference problems. For example, we can imagine using

- ▶ embeddings of product images and descriptions for modeling variety and demand for products;
- ▶ embeddings of text resumes for studying the wage offer structure;
- ▶ embeddings of countries' characteristics for studying the effect of institutions;
- ▶ and please list many of your own here (homework).

There is an emerging literature on the use of embeddings for causal inference; see this [repository of papers about using text data in causal inference](#). See also [2] for a recent review article on the importance and subtleties of using text as data in the social sciences.

## 11.2 From Principal Components to Autoencoders

Principal components are probably the earliest classical example of embeddings. One way to frame principal components is that principal components find unit length orthogonal linear combinations, directions, of a collection of variables that are "best" at reproducing the underlying data. The idea is then that a small number of principal components should capture most of the variability in the original variables and thus may provide a useful low-dimensional summary of the original data.

Specifically, let  $(W_1, \dots, W_n)$  be a sample of  $n$  observations of a high-dimensional centered random vector  $W_i$  in  $\mathbb{R}^d$ ,<sup>1</sup> and let  $\Sigma_n = \mathbb{E}_n[WW'] \in \mathbb{R}^{d \times d}$  denote the empirical covariance matrix. In order to reduce the dimension of  $W_i$ , suppose we wish to find  $K \ll d$  mutually orthogonal rotations

$$X_{ki} := c'_k W_i, \quad k = 1, \dots, K,$$

1: Thus,  $\mathbb{E}_n[W_j] = 0$  for  $j = 1, \dots, d$ .

of the original  $W_i$ 's where

$$c'_\ell c_k = 0 \text{ for } \ell \neq k \text{ and } c'_k c_k = 1 \text{ for each } k$$

such that linear combinations of these variables approximate the original data. These rotations are called principal components of  $W_i$ . In applications,  $W_i$  represent high-dimensional raw features (images, for example), and the principal components

$$X_i^K = (X_{i1}, \dots, X_{iK})'$$

represent a lower-dimensional encoding or embedding of  $W_i$ .

More formally, we wish to solve

$$\min_{\{a_j\}_{j=1}^d, \{c_k\}_{k=1}^K} \sum_{j=1}^d \sum_{i=1}^n (W_{ji} - \hat{W}_{ji})^2$$

subject to

$$\hat{W}_{ji} := a'_j X_i^K \text{ for } X_i^K = (X_{i1}, \dots, X_{iK})' \text{ } j = 1, \dots, d \text{ and } i = 1, \dots, n,$$

$$X_{ki} = c'_k W_i \text{ for } i = 1, \dots, n \text{ and } k = 1, \dots, K,$$

$$c'_k c_k = 1 \text{ for } k = 1, \dots, K$$

$$c'_k c_\ell = 0 \text{ for } \ell \neq k \leq K.$$

The constructed variables resulting from solving this problem,

$$X_i^K = (X_{i1}, \dots, X_{iK})'$$

are the first  $K$  principal components.

**Remark 11.2.1** The analytical solution to the principal components problem is as follows: The optimal  $C_K = [c_1, \dots, c_K]$  are the eigenvectors of  $\Sigma_n = \mathbb{E}_n[WW']$  corresponding to the  $K$  largest eigenvalues  $\lambda_1, \dots, \lambda_K$  of  $\Sigma_n$ . That is,  $\Sigma_n c_k = \lambda_k c_k$  for each  $k$ . Furthermore, the optimal  $a_j$  is the  $j$ -th column of  $C'_K$ .

Another interesting feature of principal components is that they satisfy

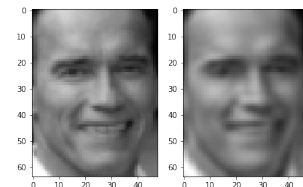
$$\mathbb{E}_n[X_k^2] = \lambda_k$$

for  $k = 1, \dots, K$  and

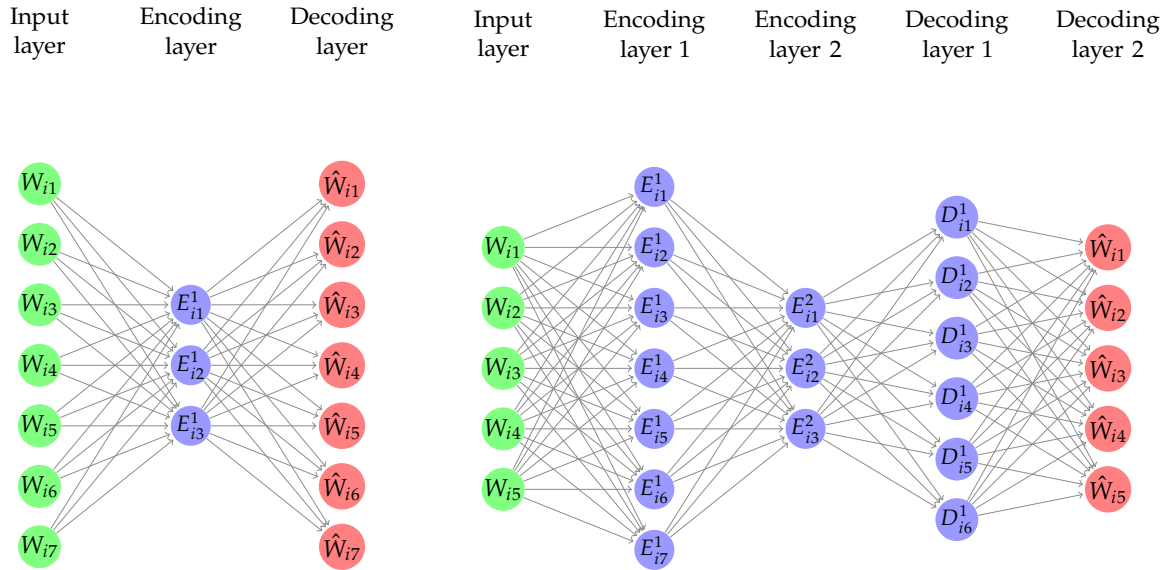
$$\mathbb{E}_n[X_k X_\ell] = 0$$

for  $\ell \neq k$ . These properties result from the fact that the  $c_k$  are eigenvectors of  $\Sigma_n$ .

Finding principal components offers one way to produce encod-



**Figure 11.1:** Featurizing a talented man: The original 3072-dimensional image  $W$  and image  $\hat{W}$  produced from a 256-dimensional principal component embedding. As a by-product, we've just made an important causal discovery that, surprisingly, doing embedding causes one to be younger ;).



**Figure 11.2:** The left panel shows a linear single layer autoencoder, such as linear principal components. The right panel shows a three layer nonlinear autoencoder; the middle layers can be used as embeddings.

ings/embeddings of raw inputs. Once we have encodings/embeddings from any method, we can look at how similar the raw inputs  $W_k$  and  $W_l$  are via the cosine similarity of the embeddings:

$$\text{sim}(W_k, W_l) = X'_k X_l / (\|X_k\| \|X_l\|).$$

In the context of product embeddings, this approach can be used, for example, to find products that are similar to a given product.

This predictive exercise underlying principal components can be seen as a linear neural network:

$$W_i \xrightarrow{d \times 1} C'_K W_i =: E \xrightarrow{k \times 1} A'E =: \hat{W}_i, \quad d \times 1$$

for  $A = [a_1, \dots, a_d]$ . The first step is said to be "encoding" the information in the input, and the second step is said to be "decoding" in the sense of returning the encoded information to the original space. Therefore, principal components are embeddings generated by a linear "encoder-decoder" network (an *autoencoder*, for short). For principal components, the relationship between the encoder and decoder happens to be rather simple, in that  $A = C'_K$  (see Remark 11.2.1).

This framing suggests that we can immediately generalize this approach to nonlinearly generated encoders and decoders that

have multiple layers:

$$W_i \xrightarrow{g_1} E_i^1 \dots \xrightarrow{g_k} E_i^k \xrightarrow{g_{k+1}} D_i^{k+1} \dots \xrightarrow{g_m} D_i^m =: \hat{W}_i,$$

where maps  $g_\ell$ 's are neuron-generating maps. The middle layer or layers of low dimension, represented by the  $E_i^k$ , are taken to be encoders. The layers of neurons are mnemonically labelled as either "E" or "D," depending on whether they are doing "encoding" or "decoding," though note that there is no strict formal distinction between these types of layers.

Autoencoders are a way of discovering latent, low-dimensional structures in a dataset. In particular, a random data vector  $W \in \mathbb{R}^d$  can be said to have low-dimensional structure if we can find some "well-behaved" functions  $e : \mathbb{R}^d \rightarrow \mathbb{R}^k$  and  $d : \mathbb{R}^k \rightarrow \mathbb{R}^d$ , with  $k \ll d$ , such that

$$(d(e(W))) \approx W.$$

In other words,  $X = e(W)$  is a parsimonious,  $k$ -dimensional representation of  $W$  that contains all of the information necessary to approximately reconstruct the full vector  $W$ . Traditionally, the map  $e(\cdot)$  is called an encoder, and the map  $d(\cdot)$  is called a decoder function. Given this, a general formulation of autoencoders is to minimize the average reconstruction loss,

$$\mathbb{E}_n[\text{loss}(W, d(e(W))),$$

over "well-behaved" functions  $d \in D$  and  $e \in E$ . These classes are often linear, as in principal components, or generated via neural networks.

The qualification of "well-behaved" is important since it is always possible to write down some (completely wild) one-to-one function  $e : \mathbb{R}^d \rightarrow \mathbb{R}^1$  such that  $e^{-1}e(W) = W$ .<sup>2</sup>

2: Google "Borel Isomorphism."

**Remark 11.2.2** (Independent Component Analysis) Principal component analysis defines "well-behaved" functions as linear functions whose output  $(X_1, \dots, X_k) = e(W)$  has uncorrelated entries, i.e.  $E[X_i X_j] = 0$ . In other words, PCA tries to find latent embedding vectors  $(X_1, \dots, X_k)$  that have the ability to reconstruct the original covariate vectors and are uncorrelated with each other. One could take a step further and require that these latent embeddings are independent of

each other, not just uncorrelated, i.e.  $X_i \perp\!\!\!\perp X_j$ . This leads to the method called linear Independent Component Analysis (ICA) [3]. The intuition of ICA when taken to neural network representations has led to the notion of disentangled representations, i.e. embeddings that encode independent latent dimensions of variation in the data. However, beyond the linear ICA setting, non-linear neural network based versions of ICA have more brittle theoretical foundations in the absence of auxiliary task-related information and task-related outcome variables [4].

**Remark 11.2.3** (Variational Auto-Encoders [5]) The encoding and decoding functions so far in our discussion have been restricted to be deterministic. Implicitly, this assumes that given the observed high-dimensional variables  $W$ , we can uniquely identify the low-dimensional variables that contain all the information in  $W$ . Such unique mapping from observed factors to latent factors is not always possible. An important extension of the auto encoder framework is allowing for these mappings to be stochastic. This extension is inspired by Bayesian probabilistic modelling that views the embeddings as latent factors and imagines that the data are drawn by first drawing the latent factors and then drawing the data samples from some distribution that is dependent on the latent factors. The Variational Auto-Encoder is attempting to reverse engineer and learn the latent factor space and the posterior distribution of the latent factors conditional on the observed data using computationally tractable approximate versions of the maximum likelihood method. In the end the method looks very similar to auto-encoders (though stemming from a different reasoning), albeit introducing randomness in the encoding phase. Roughly speaking, we optimize a loss of the form:

$$\mathbb{E}_n[\text{loss}(W, d(e(W, Z)))] + \text{penalty}(\mathbf{e}),$$

where  $Z$  is an exogenous jointly independent Gaussian random noise vector and the penalty term forces the encoding function to be non-deterministic and stems from derivations related to the objective of learning the posterior distribution of latent factors. Conditional on an observed  $W$ , the random variable  $e(W, Z) \mid W$  can be interpreted as a random sample from the posterior distribution of the latent factors that could have generated the observed sample  $W$ . Moreover, the function  $e(W, Z)$ , is typically of the form  $e(W, Z) = \mu(W) + \Sigma(W)Z$ , where the deterministic functions  $\mu(W)$  and  $\Sigma(W)$  encode the mean and the covariance of the posterior distribution of



the latent factors. These deterministic functions  $\mu(W), \Sigma(W)$  can be viewed as deterministic embeddings of  $W$  and can be used as engineered features in downstream tasks (see e.g. [6]). Alternatively, one can use only  $\mu(W)$ , which is approximating the mode of the posterior. For a more in-depth introduction to variational auto-encoders see [7].

### 11.3 From Auto-Encoders to General Embeddings

The notion of loss can also be generalized to other loss functions, where the target outcome  $A$  may be not  $W$ , but something else. We can search for embeddings that minimize average prediction loss,

$$\mathbb{E}_n[\text{loss}(A, f(e(W)))],$$

where the role of  $f(\cdot)$  is no longer just to decode but to predict  $A$  (rather than  $W$ ).

For example, in feature engineering from images,  $A$  could be a product type or subtype, and  $W$  could be the image. In feature engineering from text,  $A$  could be a masked word in a sentence and  $W$  the sentence containing this word. These alternative approaches could be more useful in relation to the final learning task. For example, to build good hedonic price models, we may be more interested in image or text embeddings that best help to accurately describe the type or subtype of a product (rather than reconstruct the image or text itself).

This approach is generally implemented via neural networks as follows

$$W_i \xrightarrow{g_1} E_i^1 \dots \xrightarrow{g_k} E_i^k \xrightarrow{g_{k+1}} F_i^{k+1} \dots \xrightarrow{g_m} F_i^m =: \hat{A}_i,$$

where  $g_\ell$ 's are neuron-generating maps. The middle layers  $E_i^k$  are taken to be embedding layers, and  $F_i^{k'}$ 's are predictive layers, aimed to create good predictions of auxiliary targets.

## 11.4 Text Embeddings

### First generation: Word2Vec Embeddings

We first review some basic ideas underlying the Word2Vec algorithm (Mikolov et al., 2013). One way we could encode words that appear in a corpus of documents (e.g. product descriptions) into a vector is to consider a very high-dimensional vector of dimension  $d$ , where  $d$  is the total number of words in the corpus. Then the  $j$ -th word in the corpus (e.g. in alphabetical order) can be represented as:

$$e_j = (0, \dots, 0, 1, 0, \dots, 0)',$$

with 1 in the  $j$ -th position. This encoding has a very high dimension limiting its usefulness. Furthermore, this representation does not capture word similarity – e.g., cosine similarity between two different words  $j$  and  $k$  is always zero since  $e_j' e_k = 0$ .

Instead we aim to represent words by vectors of much lower dimension,  $r$ , that are able to capture word similarity. We denote the representation of the  $j$ -th word by  $u_j$ , so the dictionary is an  $r \times d$  matrix

$$\omega = \{u_1, \dots, u_d\},$$

where  $r$  is the reduced dimensionality of the dictionary. This dictionary is a linear rotation of the original dictionary  $E = \{e_1, \dots, e_d\}$ , where

$$\omega = \omega E.$$

Therefore, the problem of finding the rotation  $\omega$  is analogous to the problem of finding principal components, except that our goal is now to find representations  $\omega$  that are able to capture word similarity. Once we are done, each word  $t_j$  in a human-readable dictionary can be represented by a new "word"  $u_j$ . The goal of Word2Vec is to find an effective representation with the dimension  $r$  of the embedding being much smaller than the total number of words in the corpus,  $d$ . We achieve this goal by treating  $\omega$  as parameters and estimating them so that the model performs well in some basic natural language processing tasks. These tasks are typically not related to downstream tasks, such as predicting hedonic prices or performing causal inference using text as control features, but are related to language prediction tasks.

Figure 11.3 shows components of embeddings for several words produced by a trained Word2Vec map. The numbers presented in the table are not particularly interpretable in isolation. Each

### Example of Wor2Vec features

womens	0.387542	0.03051	-0.19703	0.179724	-0.222901	-0.606905	0.306091	-0.597467
mens	0.758868	0.372418	0.370116	0.706623	-0.124954	0.5088	0.106177	0.208935
clothing	0.149283	0.5161	-0.027684	0.218484	-0.851416	-0.409885	0.386088	0.170605
shoes	1.323812	-0.358704	-0.007683	-0.552144	0.011261	0.365239	0.228273	-0.565655
women	0.601477	-0.045845	-0.099481	0.010576	-0.096852	-0.605281	0.25606	-0.550759
girls	0.417473	-0.005265	-0.40939	-0.531189	-1.31938	-0.034746	-0.940507	-0.361215
men	0.778298	0.406613	0.426292	0.534272	-0.056103	0.51756	0.107846	0.245275
boys	0.896637	-0.016821	-0.001602	-0.181901	-1.313441	0.449006	-0.828408	0.52121
accessories	0.86825	-0.378385	-1.247708	1.541265	0.323952	0.282909	-0.491176	0.081314
socks	0.27636	0.354296	0.185734	0.301311	-0.643142	-0.021945	0.320751	0.240676
luggage	0.796763	1.749548	-2.30671	-0.559585	0.03054	0.921458	0.417333	0.313436
dress	0.282053	0.233192	0.043318	0.174759	-0.50114	-0.381047	0.297995	-0.026033
baby	0.346065	-0.550016	-1.136202	-0.043899	-2.004979	0.689747	-1.091575	0.009901
jewelry	-0.315784	0.347808	-0.308736	0.878713	-0.766016	1.124318	-0.079883	-2.039485
black	0.427496	0.030204	-0.019082	0.224096	-0.162242	-0.325359	0.170407	-0.172714
boots	1.009074	-0.30359	0.03197	-0.334004	-0.095679	0.111328	0.11769	-0.51878
shirts	0.444152	0.452918	0.393656	0.517929	-0.531462	0.099621	0.146202	0.204338
shirt	0.328998	0.421561	0.226565	0.455649	-0.700352	0.067224	0.106364	0.233862
underwear	0.230821	0.490978	0.226338	0.202376	-0.774363	0.004693	0.228712	0.310215

**Figure 11.3:** Examples of words converted to numerical features via Word2Vec. Compare embeddings for words "shirt" and "shirts" and for "luggage" and "dress".

column represents a "trait" and the cell entry represents the loading of the word in the row in that trait. The numbers are more useful in comparison with each other across different rows which allows us to understand word similarity. For example, we can see that the very similar words "shirt" and "shirts" have very similar embeddings while the embeddings for the seemingly relatively different words "luggage" and "dress" are quite dissimilar.

In our context, we can think of each word appearing in a datum (e.g. a product description) as a random variable  $T$  and denote its corresponding embedding representation by  $U$ .

One of the ways to train the word embeddings is to predict the middle word from the words that surround it in word sentences.

Given a subsentence  $s$  of  $K + 1$  words, we have a central word  $T_{c,s}$  whose identity we would like to predict. As predictors, we have the context words  $\{T_{o,s}\}$  that surround the central word  $T_{c,s}$ . One approach for forming the prediction starts by collapsing the embeddings for context words by a sum,<sup>3</sup>

$$\bar{U}_o = \frac{1}{K} \sum_o U_{o,s},$$

where  $U_{o,s}$  is the element of  $\omega$  corresponding to the word  $T_{o,s}$ . This step imposes a drastically simplifying assumption that the context words are exchangeable – i.e. the position of each word is not important.

The probability of the middle word  $T_{c,s}$  being equal to  $t$  is

3: Why not? We can try it and see if it works.

modeled via the multinomial logit function:

$$p_s(t; \pi, \omega) := P(T_{c,s} = t \mid \{T_{o,s}\}; \omega) = \frac{\exp(\pi'_t \bar{U}_s(\omega))}{\sum_{\bar{t}} \exp(\pi'_{\bar{t}} \bar{U}_s(\omega))},$$

where  $\pi = (\pi_1, \dots, \pi_d)$  is an  $m \times d$  matrix of parameter vectors defining the choice probabilities. The model constrains the choice probabilities  $\pi$  to be  $\omega$ , and estimates  $\omega$  using the maximum quasi-likelihood method:

$$\max_{\omega=\pi} \sum_{s \in \mathcal{S}} \log p_s(T_{i,s}; \pi, \omega),$$

where we sum the log-probabilities over many examples  $\mathcal{S}$  of subsentences  $s$ . Once we are done training, we can generate the embedding for the title or description of product  $i$ , containing the embedded words  $\{U_{j,i}\}_{j=1}^J$  by simply averaging them:

$$W_i = \frac{1}{J} \sum_{j=1}^J U_{j,i}. \quad (11.4.1)$$

**Remark 11.4.1** In summary, the Word2Vec algorithm transforms text into a vector of numbers that can be used to compactly represent words. The algorithm trains a neural network in a supervised manner such that contextual information is used to predict another part of the text.

For example, let's say that the title description of the item is: "Hiigoo Fashion Women's Multi-pocket Cotton Canvas Handbags Shoulder Bags Totes Purses." The model will be trained using many  $n$ -word subsentence examples, such that the center word is predicted from the rest. If we just use  $n = 3$  subsentence examples, then we train the model using the following examples: (Hiigoo, Women's)  $\rightarrow$  Fashion, (Fashion, Multi-pocket)  $\rightarrow$  Women's, (Women's, Cotton)  $\rightarrow$  Multi-pocket, and so on.

How do we judge whether the text embedding is successful or not? In the hedonic price context, we can check whether Word2Vec features improve the quality of prediction of the price by the hedonic model. We can also check if similar words  $T_k$  and  $T_l$  have similar embeddings. We can measure the similarity through cosine similarity:

$$\text{sim}(T_k, T_l) = U'_k U_l / (\|U_k\| \|U_l\|) \in [-1, 1].$$

The more similar the words are, according to our human notion of similarity, the higher the value our formal measure of similarity should take. For example, the following are the two words that are most similar to "tie" under the similarity measure: "necktie" and "bowtie." The dense embedding also induces an interesting vector space on the set of words, which seems to encode analogues well. For example, the word "briefcase" is very cosine-similar to the artificial latent word<sup>4</sup>

$$\begin{aligned} \text{Artificial word} &= \text{Word2Vec}(\text{men's}) \\ &+ \text{Word2Vec}(\text{handbag}) - \text{Word2Vec}(\text{women's}). \end{aligned}$$

This similarity between a real word and our constructed latent word gives some justification for the "averaging" of embeddings to summarize whole sentences or descriptions.

Word2vec embeddings were among the first generation of early successful embedding algorithms. These algorithms have been improved by the next generation of NLP algorithms, such as ELMo and BERT, which are discussed next.

## Second Generation: Sequence Models

A major advance in language modeling has been to represent text as a sequence using recurrent (autoregressive) models. Among the various benefits, this in particular allows for better capturing the context around words. Of note is the Embeddings from Language Models (ELMo) algorithm [10], which uses the idea of the Shannon game where we aim to guess a word in a sentence,  $m$ , consisting of  $n$  total words. Specifically, we consider the problem of predicting word  $k + 1$  using the preceding  $k$  words via

$$p_{k,m}^f(t) = P[T_{k+1,m} = t \mid T_{1,m}, \dots, T_{k,m}; \theta]$$

and similarly consider the reverse prediction via

$$p_{k,m}^b(t) = P[T_{k-1,m} = t \mid T_{k,m}, \dots, T_{n,m}; \theta],$$

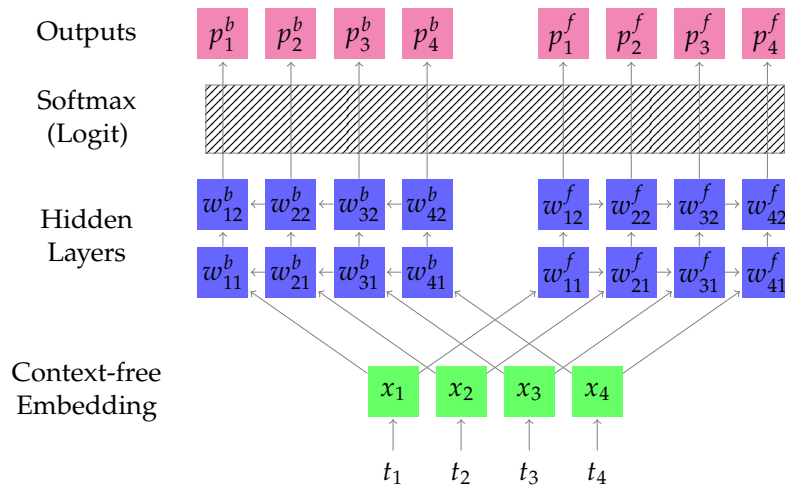
where  $\theta$  is a parameter vector. ELMo then uses recurrent neural networks (RNNs) to model these probabilities. RNN is a particular architecture for sequence input and output where we use neurons from the previous prediction to make the current prediction.<sup>5</sup> Parameters are estimated by maximizing parameterized approximate versions of the log-likelihoods of the observed data (aka quasi-likelihoods), typically referred to

4: This example also shows us how word embeddings very easily encode and propagate biases that exist in document corpora that are typically used in machine learning; a realization that has been highlighted by several recent works [8]. One should always be cognizant of such inherent biases in trained embeddings. Recent works in machine learning (e.g. [8]) provide automated approaches that partially correct for these biases, though not completely removing the problem [9].

5: RNNs are essentially the neural network versions of linear autoregressive models, such as ARIMA models, which go back to the early work of statisticians George Box and Gwilym Jenkins [11, 12] and have also been used in economics to model volatility of financial assets in the GARCH model of economist Tim Bollerslev [13].

In its simplest form a RNN parses inputs in a serial manner  $T_1, \dots, T_t, \dots, T_k$  and at each step  $t$  produces a state vector  $S_t = \sigma(AT_t + BS_{t-1} + c)$  that is a non-linear function (a set of neurons) of the current input and the previous state vector. That is,  $\sigma$  is an activation function as presented in Section 9.3 applied elementwise to each coordinate. Moreover, a RNN produces an output prediction vector  $y_t = \sigma(DS_t + e)$  that is a non-linear function (a set of neurons) of the current state. The parameters  $A, B, c, D, e$  of all these neurons are the same (shared) across steps.

ELMo uses a particular form of recursive neural network called Long Short-Term Memory (LSTM) network. LSTMs improve upon the numerical stability of RNNs by allowing for the "state" to pass through the current step as-is, without any non-linearity applied. Allowing the state to pass through steps without alteration helps in propagating information across distant steps and thus better accommodates long-term memory.



**Figure 11.4:** ELMo Architecture. ELMo network for a string of 4 words, with  $L = 2$  hidden layers. Here, the softmax layer (multinomial logit) is a single function mapping each input in  $\mathbb{R}^d$  to a probability distribution over the dictionary  $\Sigma$ .

as quasi-maximum log-likelihood methods, where the forward and backward log quasi-likelihoods are added together.

To give a simple example, suppose we wanted to grasp the positional context better in the previous example. Rather than start by collapsing the embeddings for context words surrounding a target central word via a sum, we could instead keep track of word order and assign individual parameters to each context. For example, we could model the forward predicted probability of word  $k$  in sentence  $m$  as

$$P(T_{k,m} = t \mid \{T_{j,m}\}_{j=1}^{k-1}) = \frac{e^{\sum_{j=1}^{k-1} \pi'_{t,j} U_{j,m}(\omega)}}{\sum_{\bar{t}} e^{\sum_{j=1}^{k-1} \pi'_{\bar{t},j} U_{j,m}(\omega)'}}$$

and similarly model the reverse prediction problem where we recall that  $U_{j,m}(\omega)$  is the embedding corresponding to word  $j$  in sentence  $m$ . ELMo uses a more sophisticated (and more parsimonious) recursive nonlinear regression (specifically a recurrent neural network) model to build these probabilities. We illustrate a simple ELMo structure in Figure 11.4.

#### The basic structure of ELMo.

Given a sentence  $m$  of  $n$  words,

1. Words are mapped to context-free embeddings in  $\mathbb{R}^d$
2. A network is trained to predict each word  $T_{k,m}$  of a string given (a) words  $(T_{1,m}, \dots, T_{k-1,m})$  (forward prediction) or (b) words  $(T_{k+1,m}, \dots, T_{n,m})$  (backward prediction). The objective is to maximize the average over the sum of the log-likelihoods of the  $2n - 2$  words being predicted, where the average is taken

over all sentences.

3. The embedding of word  $T_{k,m}$  is given by a weighted average of outputs of certain hidden neurons corresponding to this word's entire context. Importantly, a subset of the parameters is coupled across the forward and backward prediction problems (2a) and (2b). In particular, the first layer that goes out of the context-free embedding and the final ("softmax") layer that produces the probabilistic predictions is the same for the two prediction objectives (2a) and (2b). Thus the inputs to this layer, which represent the forward and backward context, are constrained to lie in "the same space."

A softmax layer assigns probabilities to each class in a multi-class problem. It is a multi-class generalization of logistic regression that assumes mutually exclusive classes.

## Training

In Figure 11.4, the output probability distribution  $p_k^f$  is taken as a prediction of  $T_{k+1,m}$  using words  $(T_{1,m}, \dots, T_{k,m})$ . Similarly,  $p_k^b$  is taken as a prediction of  $T_{k-1,m}$  using words  $(T_{k,m}, \dots, T_{n,m})$ . The parameters of the network,  $\theta$ , are obtained by maximizing the quasi-log-likelihood:

$$\max_{\theta} \sum_{m \in \mathcal{M}} \left( \sum_{k=1}^{n-1} \log p_{k,m}^f(T_{k+1,m}; \theta) + \sum_{k=2}^n \log p_{k,m}^b(T_{k-1,m}; \theta) \right),$$

where  $\mathcal{M}$  is a collection of sentences. In our example,  $\mathcal{M}$  is the collection of titles and product descriptions taken from product web pages.

## Producing embeddings

To produce embeddings from the trained network, each word  $t_k$  in a sentence  $m = (t_1, \dots, t_n)$  is mapped to a weighted average of the outputs of the hidden neurons indexed by  $k$ :

$$t_k \mapsto w_k := \sum_{i=1}^L (\gamma_i w_{ki}^f + \bar{\gamma}_i w_{ki}^b).$$

The embedding for the sentence (or an entire product description in our example) is produced by summing the embeddings for each individual word. The weights  $\gamma$  and  $\bar{\gamma}$  can be tuned by the neural network performing the final task. In principle, the whole network could be plugged in to the network performing

the final task and allowed to update. However, the ELMo architecture and methodology is more inline with being used as a feature extractor, with only the final linear layer being trained towards the target task (in sharp contrast to the BERT model that we outline next; see e.g. [14]).

### Third generation: Transformers

A subsequent major advance in language modeling has been the development and use of the transformer architecture. Going beyond backwards and forwards sequences, transformers use a mechanism termed "self-attention" [15] in order to model the importance of different parts of the text in understanding any one other part. Like RNNs, this allows for understanding context, but unlike RNNs, it allows the model to better focus on potentially far away parts of the text that may be relevant. For example, this allows to better understand that "it" in one sentence refers to a particular word from a previous sentence.

An early and prominent example of transformer-based language models is Bidirectional Encoder Representations from Transformers (BERT) [16]. Unlike the language model in ELMo which predicts the next word from previous words, the BERT model is trained on two self-supervised tasks simultaneously:

- ▶ Mask Language Model: Randomly mask a certain percentage of the words in a sentence and predict the masked words.
- ▶ Next Sentence Prediction: Given a pair of sentences, predict whether one sentence precedes another.

#### The basic structure of BERT.

1. Each word in the input sentence is broken into subwords (tokenized) and each piece is called a "token." Each token is encoded using a context-free embedding called WordPiece. A special token [cls] is added to the beginning of the sequence.  $x\%$  of the tokens representing individual words are replaced by [mask].
2. For each token, its input representation consists of i) its token embedding from (1), ii) its position embedding indicating the position of the token in the sentence, and iii) its segment embedding indicating



whether it belongs to sentence A or B.

3. The input representation of tokens in the sequence is fed into the main model architecture: L layers of Transformer-Encoder blocks. Each block consists of a "multi-head attention layer" (described below), followed by a feed forward layer.
4. The output representation of the mask token [mask] is used to predict the masked word via a softmax layer, and the output representation of the special [cls] token is used for Next Sentence Prediction. The loss function is a combination of the two losses.

We next focus in detail on the main structure step in (3), especially the "multi-head attention" layer.

### Computing the Attention

We begin with the context-free embeddings  $(x_1, x_2, \dots, x_n)$ , for  $n$  words, with each  $x_k \in \mathbb{R}^d$ . Let  $X$  denote the matrix whose  $k$ -th row is the embedding  $x_k$ . An attention module transforms this matrix of  $n$  embeddings  $X$  into another matrix of  $n$  embeddings, where now each row  $k$  contains an embedding of the "information" in a "neighborhood" around token  $k$ . The notion of "neighborhood" and the notion of "information" are all parameterized by neural network parameters of the attention module and learnable in a data-driven manner as we describe below.

The goal of an attention module is to create weighted neighborhoods (or attention regions) of seemingly distant tokens (in a data-driven manner) and then create embeddings that correspond to linear combinations of the embeddings of the tokens in these neighborhoods (or attention regions). One way to achieve this is to decouple the "neighborhood" representation of a token with the representation of each "meaning" or "value." Thus we will transform each token embedding  $x_k$  into a *key embedding*  $\kappa_k := x'_k \omega^K$ , where  $\omega^K \in \mathbb{R}^{d \times d_k}$  is a learnable matrix parameter, and a *value embedding*  $v_k := x'_k \omega^V$ , where  $\omega^V \in \mathbb{R}^{d \times d_v}$  is a learnable matrix parameter. Then a neighborhood can be encoded by a *query vector*  $q$  that lies in the same space as the space of keys and such that the weighted neighborhood is defined via a similarity metric between the vector  $q$  and the key vectors. Attention mechanisms used in Transformers use a scaled inner

product as the similarity, i.e.  $s_k := q' \kappa_k / \sqrt{d_k}$ . Then this similarity is passed through a soft-max function  $\sigma(\cdot)$  to map it to a selection probability in  $[0, 1]$ . Finally, as we alluded to in the beginning the *embedding of the neighborhood* that corresponds to this query  $q$  is simply the weighted average of the value embeddings of the tokens, i.e.  $a := \sum_{k=1}^n \sigma(s_k) v_k$ .

Suppose now that we had  $n$  neighborhood queries  $q_1, \dots, q_n$ , then we could create  $n$  such neighborhood embeddings  $a_1, \dots, a_n$ . Transformers consider "self-attention" queries, where each of the  $n$  queries  $q_k$  corresponds to a *query embedding* associated with a particular token and is yet another linear embedding of the form  $x'_k \omega^Q$ , where  $\omega^Q \in \mathbb{R}^{d \times d_k}$  is a learnable matrix parameter. Then for each such query we can calculate the corresponding *neighborhood embedding*  $a_k$ .

Overall this transformation takes as input a matrix  $X \in \mathbb{R}^{n \times d}$ , where each row corresponds to an original token embedding and transforms it into a matrix  $A$ , where each row  $k$  corresponds to the neighborhood embedding associated with query  $q_k$ , which in turn is associated with token  $x_k$ . We can write this calculation we just described in matrix form; let  $Q = X \omega^Q$  denote the matrix with rows corresponding to query embeddings, let  $K = X \omega^K$  denote the matrix with rows corresponding to key embeddings, and let  $V = X \omega^V$  denote the matrix with rows corresponding to value embeddings. Then the attention embeddings (or neighborhood embeddings) can be written in matrix form as

$$A = \text{Attention}(Q, K, V) := \sigma \left( QK^\top / \sqrt{d_k} \right) V.$$

A Multi-Head Attention mapping, which is the building block of the BERT model, builds many such attention transformations, for different matrix parameters  $\{\omega_i^Q, \omega_i^K, \omega_i^V\}_{i=1}^h$ , calculates the corresponding attention embedding matrices  $A_i \in \mathbb{R}^{n \times d_v}$ , then concatenates the results in a big embedding matrix  $A = \text{Concatenate}(A_1, \dots, A_h) \in \mathbb{R}^{n \times h \cdot d_v}$  and applies a linear projection transformation  $A \omega^O$ , where  $\omega^O \in \mathbb{R}^{h \cdot d_v \times d_o}$ , to produce the final output encoding. Thus we can define the basic Multi-Head Attention transformation:

$$X \mapsto \text{MultiHead}(X) := \text{Concatenate}(\text{Head}_1, \dots, \text{Head}_h) \omega^O,$$

$$\text{Head}_i = \text{Attention}(X \omega_i^Q, X \omega_i^K, X \omega_i^V),$$

Each Transformer building block in BERT consists of a series several repetitions of multi-head attention encodings, followed by a fully connected neural network (applied to each of the  $n$

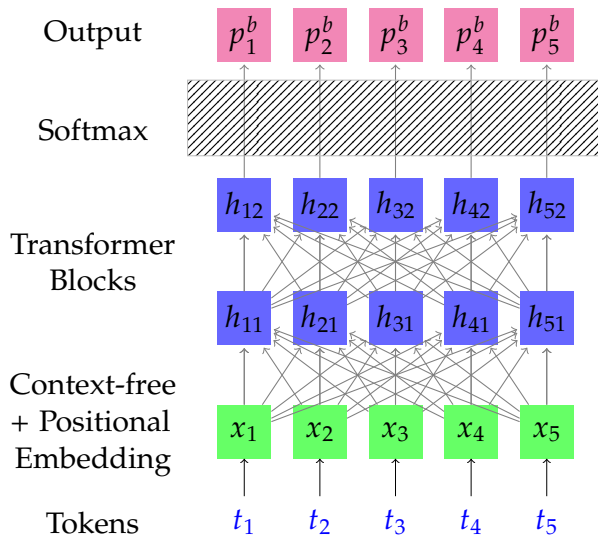


Figure 11.5: BERT Architecture

output encodings separately). The input  $n$  encodings of each repetition is the output of the previous repetition.

### Generating product embeddings

Depending on specific tasks and resources, Devlin et al. [16] suggested to construct BERT embeddings in various ways:<sup>6</sup>

- ▶ Use the last layer, second-to-last layer, or concatenate the last 4 layers of the encoder outputs from the pre-trained BERT model.
- ▶ Fine tune the whole BERT model using the downstream task.
- ▶ Train the BERT language model from scratch on new data.

In the hedonic price example below, the feature-based approach was chosen, where the second-to-last layer from a pre-trained BERT model was extracted as embeddings. Each product's text embedding is the average of the embeddings of each word/token from the input text field.

### Beyond ELMo and BERT

ELMo and BERT are both important breakthroughs in NLP. The former marked the first contextual word embedding trained from a deep language model, and the latter was the first contextual word embedding using Transformer architecture. The biggest difference lies in the choice of fundamental architectures: ELMo is based on a Recurrent Neural Network (RNN),

6: Tuning only a final linear layer on top of a pre-trained embedding network and freezing all other parameters of the embedding is referred to in the machine learning literature as "linear probing." If one allows for the parameters of the embedding itself to be updated when optimizing for a particular downstream prediction task, then this practice is referred to as "fine-tuning." See also the recent work of [17] for a better way of blending the two modes by first training the final linear layer and then unfreezing the remaining parameters of the embedding and continuing to train. This blending seems to produce substantial gains in generalization ability and accuracy of the resulting predictive model.

while BERT is based on the Transformer architecture. RNNs can struggle to capture long-term dependencies, whereas the Transformer architecture is more efficient at capturing long-range dependencies in the text. Furthermore, ELMo creates context by using the left-to-right and right-to-left language model representations, while BERT models the entire context simultaneously.

Large language models are continuously evolving and becoming ever more powerful and sophisticated in their understanding of language and meaning. The latest generation of large language models lie in the Generative Pre-trained Transformer (GPT) family [18–20]. While BERT can be understood to use the transformer architecture as an *encoder*, GPT models use the transformer architecture in a *decoder* for a generative model of the probabilities in an auto-regressive model reminiscent of the one used in ELMo. It combines these modeling ideas from such successful precursors together with pre-training on a large corpus of text. With the rapid development in the space of large language and multi-modal models, the latest and greatest models will certainly advance beyond the descriptions in this book, but the principles of using these models to understand complex data and use it to support robust causal inference may remain the same.

## Revisiting the Price Elasticity for Toy Cars

In Chapter 0, Chapter 4, and Chapter 10, we saw how using increasingly powerful learning methods (OLS, LASSO, nonlinear regression) to control for confounding in the price-sales data for toy cars lead to increasingly more negative estimates and confidence intervals for elasticity. However, these only use the categorical (brand, subcategory) and numeric (physical dimensions) features, while we actually observe much richer data: all the text on the product page, including the product description. Text embeddings are a great way to leverage these data and include them in a causal analysis of price elasticity. We can take BERT and plug it into neural networks, one to predict price and one for sales: we take the text as input, pass it through BERT initialized at the pre-trained model, add an additional dense layer, and train the whole network. Doing this over 5 folds, we obtain a cross-validated  $R^2$  of 0.55 for predicting  $Y$  and 0.027 for predicting  $D$ , improving upon the best nonlinear methods considered in the previous chapter. Applying DML with these new predictors leads to a point estimate for elasticity of -0.174 and 95% confidence interval of [-0.214, -0.135]. This suggests

we are able to better control for observed confounders, which generally push apparent elasticity up, leading to more negative estimates.

At the end of the chapter we provide a notebook wherein we repeat the exercise of constructing neural nets using BERT for predicting  $Y$  and  $D$  and plug them into DML. The results are different as we use a publicly available dataset (which does not have the same range of numeric features and therefore could not be used for comparison to high-dimensional and nonlinear predictive methods in the absence of feature engineering).

## 11.5 Image Embeddings

One of the most successful deep learning models for image classification was the ResNet50 model developed by He et al. [21]. At the time of the release, the paper achieved the best results in image classification, in particular for the ImageNet and COCO datasets.

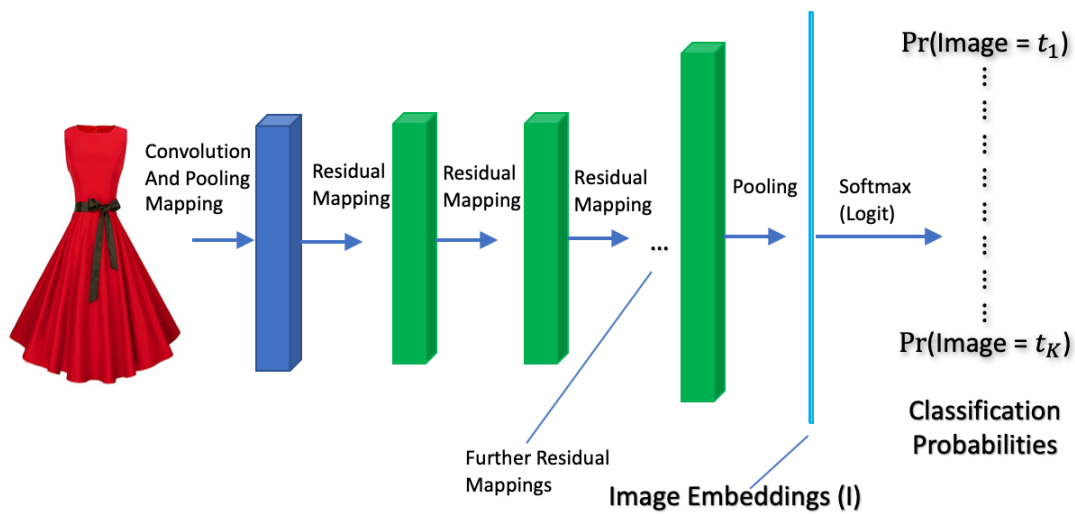
The central idea of the paper is to exploit "partial linearity": traditional nonlinearly-generated neurons are combined (or added together) with the previous layer of neurons. More specifically, ResNet50 takes a standard feed-forward convolutional neural network and adds skip connections that bypass two (or one or several) convolutional layers at a time. Each skipping step generates a residual block in which the convolution layers predict a residual.

Formally, each  $k$ -th residual block is a neural network mapping

$$\begin{aligned} v &\mapsto (v, \sigma_k^0(\omega_k^0 v)) \mapsto (v, \sigma_k^1 \circ \omega_k^1 \sigma_k^0(\omega_k^0 v)) \\ &\mapsto v + \sigma_k^1 \circ \omega_k^1 \sigma_k^0(\omega_k^0 v), \end{aligned}$$

where  $\omega$ 's are matrix-valued parameters or "weights." This structure can be seen as a special case of general neural network architecture, designed so that it is easy to learn the identity sub-maps (entering the composition of the entire network). Putting together many blocks like these sequentially results in the overall architecture depicted in Figure 11.6.

The deep feed-forward convolutional networks developed in prior work suffered from major optimization problems – once the depth was sufficiently high, additional layers often resulted in much higher validation and training error. It was argued that this phenomenon was a result of "vanishing gradients," where



**Figure 11.6:** The ResNet50 operates on numerical 3-dimensional arrays representing images. It first does some pre-processing by applying convolutional and pooling filters, then it applies many L-residual block mappings, producing the arrays shown in green. The penultimate layer produces a high-dimensional vector  $I$ , the image embedding, which is then used to predict the image type.

in a network of  $n$  layers, computation by backpropagation using the chain rule involves multiplying  $n$  small numbers (if using traditional activation functions, recent popular activation functions such as RELU do not induce such a small derivative), causing the gradient to "vanish" for early layers and posing a computational challenge. The residual network architecture addresses this by using the residual block architecture: including the residual directly via skip connections reduces the minimizing impact of the activation function. The creation of this architecture has allowed for high quality training even for very deep networks.

Just like with text embeddings, we are not interested in the final predictions of these networks but rather in the last hidden layer, which is taken to be the image embedding. In the next example, one can rely on a publicly trained ResNet50 model to generate the image embeddings.

### Application: Hedonic Prices

Here we apply our new knowledge of embeddings to review an empirical application considered in Bajari et al. [22]. The application is a prediction problem which deals with hedonic price models. An empirical hedonic model is a predictive model for price given a traded object's characteristics.<sup>7</sup> Here, the goal is

7: It can be given structural or causal interpretation using the so-called hedonic price models from economics.

to predict the price of apparel bought and sold on Amazon.com using the product's image and description:

$$P_{it} = H_{it} + \epsilon_{it} = h_t(X_{it}) + \epsilon_{it}, \quad E[\epsilon_{it} | X_{it}] = 0, \quad (11.5.1)$$

where  $P_{it}$  is the price of product  $i$  at time  $t$  (in months),  $X_{it}$  are the product features, and the price function  $x \mapsto h_t(x)$  can change from period to period, reflecting the fact that product attributes/features may be valued differently in different periods. [22] use the data from time period  $t$  to estimate the function  $h_t$  using modern nonlinear regression methods, such as deep neural network methods. The results are contrasted with classical linear regression methods as well as other modern regression methods, such as the random forest.

One of the main uses of hedonic prices is construction of cost of living indices. The use of hedonic prices allows us to "price" the product attributes as well as entire "baskets of attributes" that consumers buy. Then, given a reference "basket of attributes," one can look at the hedonic cost of a basket today compared to its cost in an earlier reference period to determine whether the cost increased or decreased. These types of calculations underlie the construction of commonly used consumer price indices (measuring inflation rates), at least for categories such as apparel products.

A key component of the approach taken in [22] is the use of product features  $X_{it}$  generated as neural network embeddings of text and image information about the product. Specifically,  $X_{it}$  consists of text embedding features  $W_{it}$ , constructed by converting the title and product description available on a product's web page into numeric vectors, and image embedding features  $I_{it}$  constructed by converting the product image into numeric vectors:

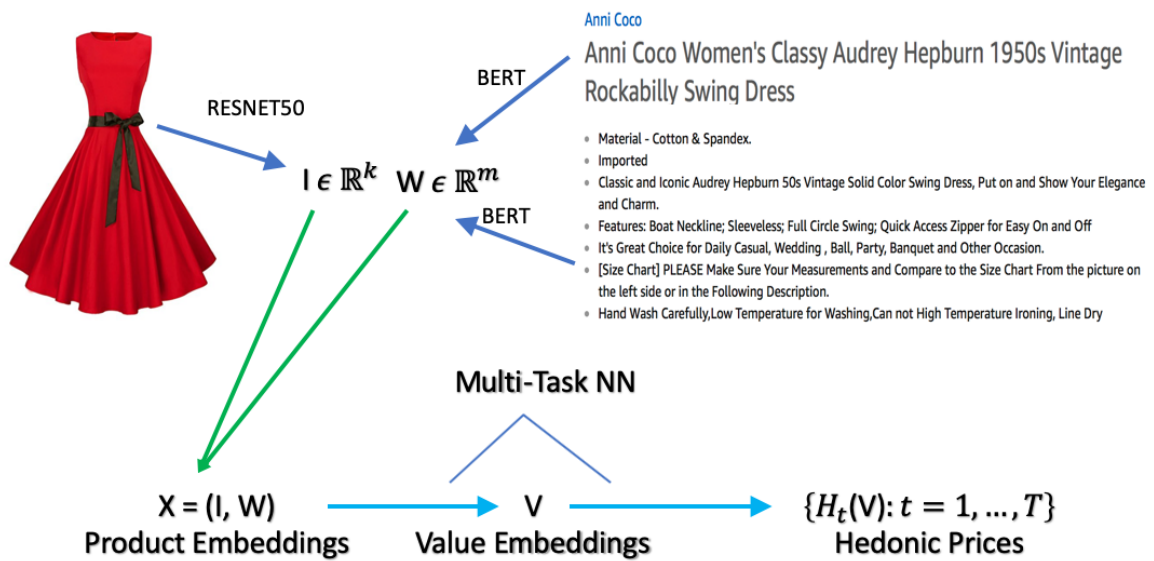
$$X_{it} = (W'_{it}, I'_{it})'. \quad (11.5.2)$$

These embedding features are generated respectively by applying the BERT and ResNet50 mappings.

The model takes high-dimensional text and image features as inputs, converts them into a lower dimensional vector of value embeddings using deep learning methods, and outputs simultaneous predictions of price in all time periods.

The general structure of the model takes the form

$$Z_{it} = \begin{bmatrix} \text{Text}_{it} \\ \text{Image}_{it} \end{bmatrix} \xrightarrow{e} X_{it} \\ \xrightarrow{g^1} E_{it}^{(1)} \dots \xrightarrow{g^m} E_{it}^{(m)} =: V_{it} \xrightarrow{\theta'} \{H_{it}\}_{t=1}^T := \{\beta'_t V_{it}\}_{t=1}^T.$$



**Figure 11.7:** The structure of the predictive model in Bajari et al. [22]. The input consists of images and unstructured text data. The first step of the process creates numerical embeddings  $I$  and  $W$  for images and text data via state of the art deep learning methods, such as ResNet50 and BERT. The second step of the process takes as its input  $X = (I, W)$  and creates predictions for hedonic prices  $H_t(X)$  using deep learning methods with a multi-task structure. The models of the first step are trained on tasks unrelated to predicting prices (e.g., image classification or word prediction), where embeddings are extracted as hidden layers of the neural networks. The models of the second step are trained by price prediction tasks. The multitask price prediction network creates an intermediate lower dimensional embedding  $V = V(X)$ , called a value embedding, and then predicts the final prices in all time periods  $\{H_t(V), t = 1, \dots, T\}$ . Some variations of the method include fine-tuning the embeddings produced by the first step to perform well for price prediction tasks (i.e. optimizing the embedding parameters so as to minimize price prediction loss).

Here  $Z_{it}$ ,<sup>8</sup> the original input which lies in a very high-dimensional space, is nonlinearly mapped into an embedding vector  $X_{it}$  which is of moderately high dimension (up to 5120 dimensions in this example).  $X_{it}$  is then further nonlinearly mapped into a lower dimension vector  $E_{it}^{(1)}$ . This process is repeated to produce the final hidden layer,  $V_{it} = E_{it}^{(m)}$ , which is then linearly mapped to the final output that consists of hedonic price  $H_{it}$  for product  $i$  in all time periods  $t = 1, \dots, T$ .

The last hidden layer  $V = E^{(m)}$  is called the *value embedding* in this context – the value embedding represents latent attributes to which dollar values are attached. The embeddings produced in this example are moderately high-dimensional (up to 512 dimensions) summaries of the product, derived from the most common attributes that directly determine the price of the predicted hedonic price of the product. Note that the embeddings  $V$  in this example do not depend on time and so may be thought of as representing intrinsic, potentially valuable attributes of the product. However, the predicted price does depend on

8: As a practical matter, most of the product attributes in [22] are time-invariant - that is,  $Z_{it} = Z_i$  has no time variation. We state the model in more generality here.



time  $t$  via the coefficient  $\beta_t$ , reflecting the fact that the different intrinsic attributes are valued differently across time.

The network mapping above comprises a deep neural network with neurons  $E_{k,\ell}$  of the form

$$g_\ell : v \mapsto \{E_{k,\ell}(v)\}_{k=1}^{K_\ell} := \{\sigma_{k,\ell}(v' \alpha_{k,\ell})\}_{k=1}^{K_\ell}. \quad (11.5.3)$$

Here  $\sigma_{k,\ell}$  is the activation function that can vary with the layer  $\ell$  and can vary with  $k$ , from one neuron to another.

The model is trained by minimizing the loss function

$$\min_{\eta \in \mathcal{N}, \{\beta_t\}_{t=1}^T} \sum_t \sum_i (P_{it}^c - \beta_t' V_{it}(\eta))^2 Q_{it}, \quad (11.5.4)$$

where  $\eta$  denotes all of the parameters of the mapping

$$X_{it} \mapsto V_{it}(\eta)$$

and  $\mathcal{N}$  represents the parameter space. Here, we are using a weighted loss where we weight by the quantity of product  $i$  sold at time  $t$ ,  $Q_{it}$ .

Next we review how the initial embedding is generated. A multilingual BERT model is used to convert text information and the ResNet50 model is used to convert images into a subvector of  $E_{it}^{(1)}$ . These models are trained on auxiliary prediction tasks with auxiliary outputs  $A_{T_{it}}$  for text and  $A_{I_{it}}$  for images. Introducing these auxiliary tasks can be illustrated diagrammatically as

$$X_{it} = \begin{bmatrix} \text{Text}_{it} \\ \text{Image}_{it} \end{bmatrix} \xrightarrow{e} \begin{array}{c} A_{T_{it}} \\ \uparrow \\ W_i \\ I_i \\ \downarrow \\ A_{I_{it}} \end{array} =: E_{it}^{(1)} \dots \mapsto E_{it}^{(m)} := V_{it} \mapsto \{\hat{P}_{it}^*\}_{t=1}^T, \quad (11.5.5)$$

The embeddings  $W_{it}$  and  $X_{it}$  forming  $E_{it}^{(1)}$  are obtained by mapping them into auxiliary outputs  $A_{T_j}$  and  $A_{I_j}$  that are scored on natural language processing tasks and image classification tasks respectively. This step uses data that are not related to prices, as described in detail in the previous sections. The parameters of the mapping generating  $E_{it}^{(1)}$  are considered as fixed in our analysis.

The price prediction network we employ in this example contains three hidden layers, with the last hidden layer containing 400 neurons. The network is trained on a large data set with

more than 10 million observations. A large enough data set is crucial for training successful neural networks.

The accuracy of prediction as measured by the  $R^2$  on the test sample is about

90%.

In contrast,

- ▶ random forests using embeddings deliver  $R^2$  in the ballpark of 80%;
- ▶ the linear model using least squares applied to embeddings delivers  $R^2$  in the ballpark of 70%;
- ▶ the linear model, using simple catalogue features (without embeddings), delivers an  $R^2$  lower than 40%.

Thus, embeddings offer a means of making use of complex data for predictions and, at least for large data sets, neural nets can offer predictive improvements relative to competing machine learning approaches.

## Notes

[23] develop "DoubleMLDeep" which allows to use additionally multimodal data, in particular text and images, as confounding variables in the DML framework.

## Notebooks

- ▶ [Python Auto-Encoders Notebook](#) provides an introduction to auto-encoders, starting from classical principal components.
- ▶ [Python Toys and Prices Notebook](#) provides an introduction to text embeddings via BERT and provides an application to predicting demand for toys.

## Study Problems

1. Work through the Auto-Encoders notebook. Try to improve the performance of the auto-encoders. Report your findings (even if you don't manage to improve them! :-)).

2. Work through the BERT notebook. Try to experiment with the structure of the neural nets and demand estimation procedure. Report your findings.

# Bibliography

- [1] Susan Sontag. 'Turning Points'. In: *Irish Pages* 2.1 (2003), pp. 186–191 (cited on page 287).
- [2] Matthew Gentzkow, Bryan Kelly, and Matt Taddy. 'Text as Data'. In: *Journal of Economic Literature* 57.3 (2019), pp. 535–74. DOI: [10.1257/jel.20181020](https://doi.org/10.1257/jel.20181020) (cited on page 289).
- [3] Pierre Comon. 'Independent component analysis, A new concept?' In: *Signal Processing* 36.3 (1994). Higher Order Statistics, pp. 287–314. DOI: [https://doi.org/10.1016/0165-1684\(94\)90029-9](https://doi.org/10.1016/0165-1684(94)90029-9) (cited on page 293).
- [4] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. 'Challenging Common Assumptions in the Un-supervised Learning of Disentangled Representations'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 4114–4124 (cited on page 293).
- [5] Diederik P. Kingma and Max Welling. 'Auto-Encoding Variational Bayes'. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014 (cited on page 293).
- [6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 'Representation learning: A review and new perspectives'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828 (cited on page 294).
- [7] Diederik P Kingma, Max Welling, et al. 'An introduction to variational autoencoders'. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392 (cited on page 294).
- [8] Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. 'Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems. NIPS'16*. Barcelona, Spain: Curran Associates Inc., 2016, 4356–4364 (cited on page 298).

- [9] Hila Gonen and Yoav Goldberg. ‘Lipstick on a pig: De-biasing methods cover up systematic gender biases in word embeddings but do not remove them’. In: *arXiv preprint arXiv:1903.03862* (2019) (cited on page 298).
- [10] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. ‘Deep contextualized word representations’. In: *CoRR abs/1802.05365* (2018) (cited on page 298).
- [11] George EP Box and Gwilym M Jenkins. *Time Series Analysis Forecasting and Control*. Tech. rep. WISCONSIN UNIV MADISON DEPT OF STATISTICS, 1970 (cited on page 298).
- [12] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015 (cited on page 298).
- [13] Tim Bollerslev. ‘A conditionally heteroskedastic time series model for speculative prices and rates of return’. In: *Review of Economics and Statistics* (1987), pp. 542–547 (cited on page 298).
- [14] Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. ‘To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks’. In: *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 7–14. doi: [10.18653/v1/W19-4302](https://doi.org/10.18653/v1/W19-4302) (cited on page 301).
- [15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. ‘Attention is All you Need’. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017 (cited on page 301).
- [16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. ‘BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding’. In: *CoRR abs/1810.04805* (2018) (cited on pages 301, 304).
- [17] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. ‘Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution’. In: *International Conference on Learning Representations*. 2022 (cited on page 304).

- [18] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 'Improving language understanding by generative pre-training'. In: (2018) (cited on page 305).
- [19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 'Language models are unsupervised multitask learners'. In: *OpenAI blog* 1.8 (2019), p. 9 (cited on page 305).
- [20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 'Language models are few-shot learners'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1877–1901 (cited on page 305).
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cited on page 306).
- [22] Patrick L. Bajari, Zhihao Cen, Victor Chernozhukov, Manoj Manukonda, Jin Wang, Ramon Huerta, Junbo Li, Ling Leng, George Monokroussos, Suhas Vijaykumar, et al. *Hedonic prices and quality adjusted price indices powered by AI*. Tech. rep. cemmap working paper, 2021 (cited on pages 307–309).
- [23] Sven Klaassen, Jan Teichert-Kluge, Philipp Bach, Victor Chernozhukov, Martin Spindler, and Suhas Vijaykumar. *DoubleMLDeep: Estimation of Causal Effects with Multimodal Data*. 2024 (cited on page 311).

# **ADVANCED TOPICS**