Feature Learning

Feature Engineering

- ❑ Look at what most powerful feature engineering tool at our disposal
- ❑ Feature learning algorithms are able to take in cleaned data and create brand-new features by exploiting latent structures within data.
- ❑ This same as feature transformations
- ❑ The differences between these two families of algorithms are in the parametric assumptions that they make when attempting to create new features.

Feature Learning

We will be covering the following topics:

- ❑ Parametric assumptions of data
- ❑ Restricted Boltzmann Machines
- ❑ The BernoulliRBM
- ❑ Extracting RBM components from MNIST
- ❑ Using RBMs in a machine learning pipeline
- ❑ Learning text features—word vectorization

Parametric assumptions of data

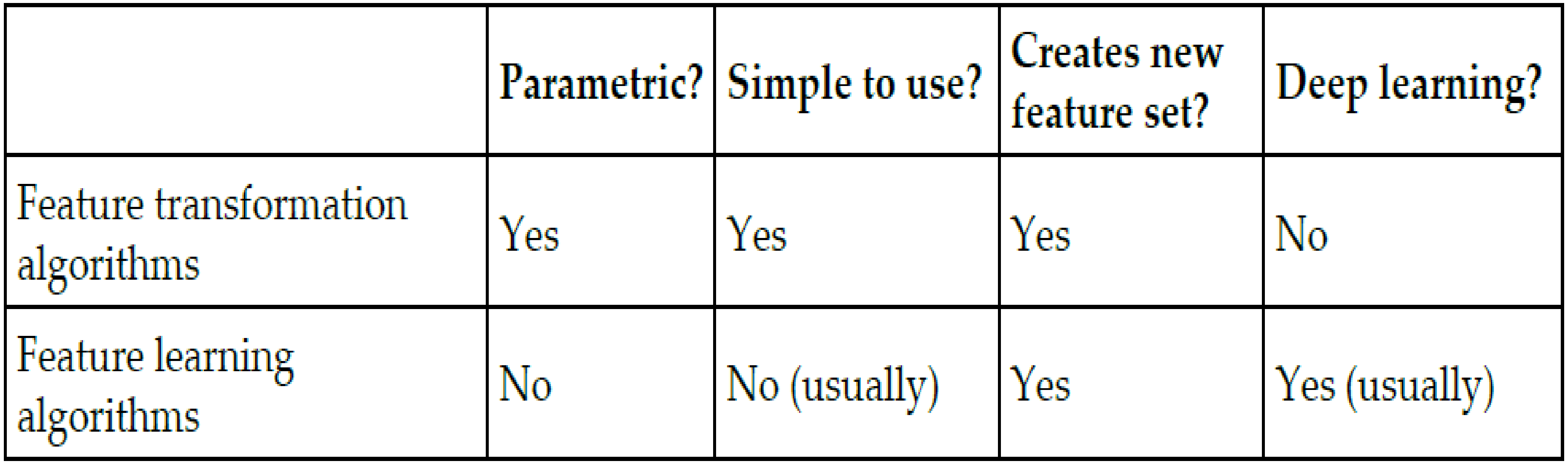
- **Parametric assumptions**, are the base assumptions that algorithms make about the *shape* of the data.
- **For PCA**, assumption that we were making was that the original data took on a shape that could be decomposed and represented by a single linear transformation (the matrix operation).
- But what if that is not true?
- What if PCA is unable to extract *useful* features from the original dataset?
- Algorithms such as PCA and **LDA** will always be able to find features, but they may not be useful at all.
- Moreover, these algorithms rely on a predetermined equation and will always output the same features each and every time they are run.
- This is why we consider both LDA and PCA as being *linear transformations.*

Parametric assumptions of data

- ❑ Feature learning algorithms attempt to solve this issue by removing that parametric assumption.
- ❑ They do not make any assumptions about the shape of the incoming data and rely on stochastic learning.
- ❑ Instead of throwing the same equation at the matrix of data every time, they will attempt to figure out the best features to extract by looking at the data points over and over again (in epochs) and converge onto a solution (potentially different ones at runtime).
- ❑ Bypassing parametric assumptions requires the use of complex algorithms.
- ❑ **Deep learning algorithms** are the choice of many data scientists and machine learning to learn new features from raw data.

Parametric assumptions of data

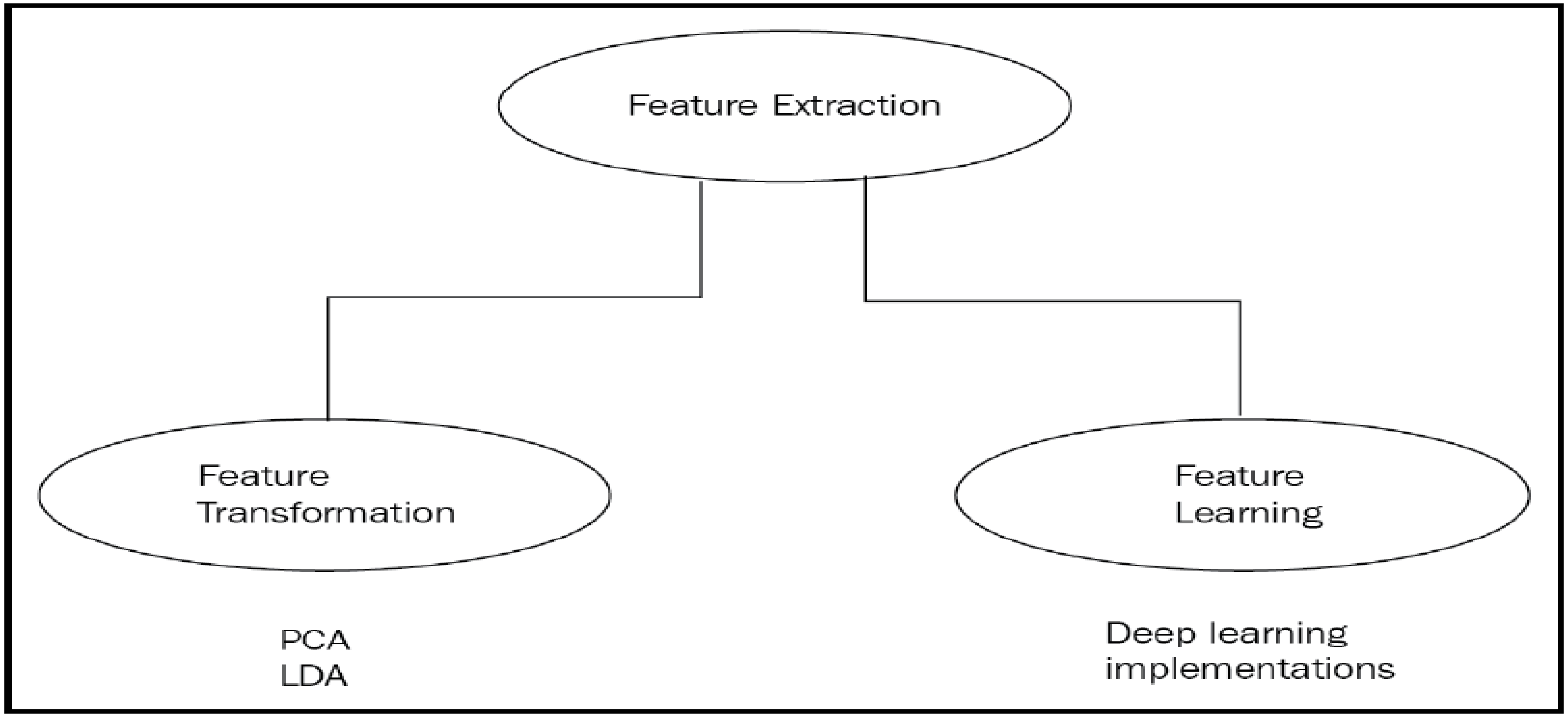
Differences between feature learning and transformation



	Parametric?	Simple to use?	Creates new feature set?	Deep learning?
Feature transformation algorithms	Yes	Yes	Yes	No
Feature learning algorithms	No	No (usually)	Yes	Yes (usually)

Parametric assumptions of data

- ❑ Both feature learning and feature transformation algorithms create new feature sets.
- ❑ Both of them as being under the umbrella of **feature extraction**.



Non-parametric fallacy

- ❑ Model being non-parametric doesn't mean that there are no assumptions at all made by the model during training.
- ❑ While the algorithms forgo the assumption on the shape of the data, they still may make assumptions on other aspects of data, for example, values of the cells.

The algorithms of this chapter

- ❑ **Restricted Boltzmann Machines (RBM**
- ❑ **Word embeddings**

The algorithms of this chapter

Restricted Boltzmann Machines (RBM)

- ❑ A simple deep learning architecture that is set up to learn a set number of new dimensions based on a probabilistic model that data follows.
- ❑ Implemented in scikit-learn (**BernoulliRBM**).
- ❑ The **BernoulliRBM** a non-parametric feature learner
- ❑ As name suggests, some expectations are set as to values of the cells of dataset.

The algorithms of this chapter

Word Embeddings

- Biggest contributors to the recent deep learning-fueled advancements of natural language processing/understanding/generation is the ability to project strings (words and phrases) into an n-dimensional feature set in order to grasp context and minute detail in wording.
- Word embeddings can be used to enhance the way we interact with text.

The algorithms of this chapter

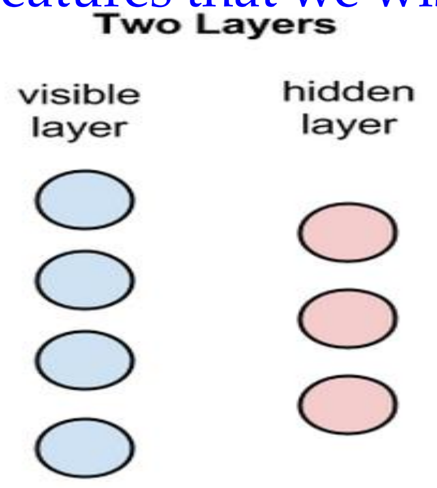
- ❑ All involve learning brand new features from raw data.
- ❑ Use these new features to enhance way that they interact with data.
- ❑ For all of these techniques, we will be focusing less on the very low-level inner workings of the models, and more on how they work to interpret data.
- ❑ Will start with the only algorithm that has a scikit-learn implementation, the Restricted Boltzmann Machine (RBM) family of algorithms.

Restricted Boltzmann Machines

- Family of unsupervised feature learning algorithms that use probabilistic models to learn new features.
- Like PCA and LDA, can use RBMs to extract a new feature set from raw data and use them to enhance ML pipelines.
- The features that are extracted by RBMs tend to work best when followed by linear models such as linear regression, logistic regression, perceptron's, and so on.
- The unsupervised nature of RBMs is important as they are more similar to PCA algorithms than they are to LDA.
- They do not require a ground-truth label for data points to extract new features.
- This makes them useful in a wider variety of machine learning problems.

Restricted Boltzmann Machines

- RBMs are shallow (two-layer) neural networks.
- Building blocks of a class of algorithms called **Deep Belief Networks (DBN).**
- There is a visible layer (the first layer), followed by a hidden layer (the second layer).
- First visible layer of network has as many layers as input feature dimension.
- Number of nodes in hidden layer is a human-chosen number and represents number of features that we wish to learn.



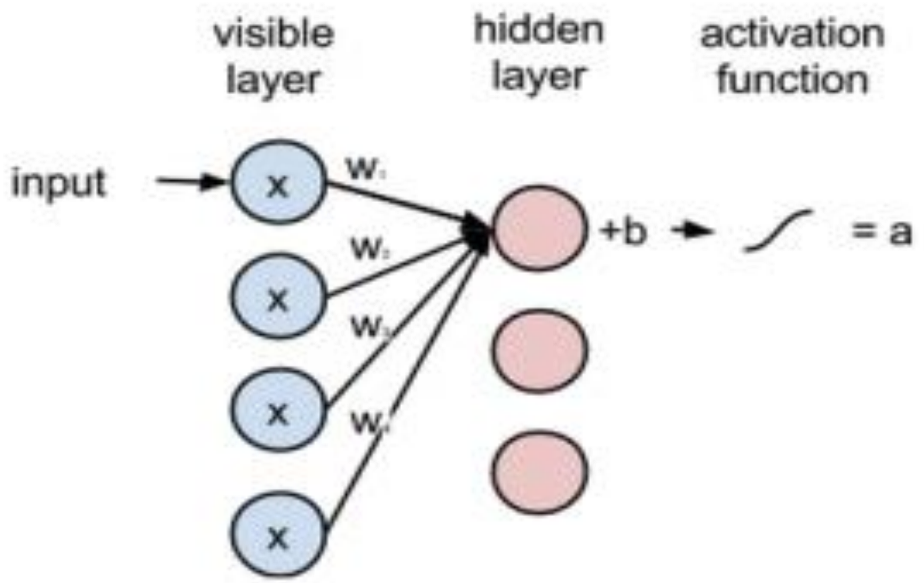
Restricted Boltzmann Machines

Not necessarily dimension reduction

- ❑ In PCA and LDA, we had severe limits to the number of components we were allowed to extract.
- ❑ For PCA, we were capped by the number of original features, while LDA enforced the much stricter imposition that caps the number of extracted features to the number of categories in the ground truth minus one.
- ❑ The only restriction on the number of features RBMs are allowed to learn is that they are limited by the computation power of the computer running the network and human interpretation.
- ❑ RBMs can learn fewer or *more* features than we originally began with.
- ❑ The exact number of features to learn is up to the problem and can be gridsearched.

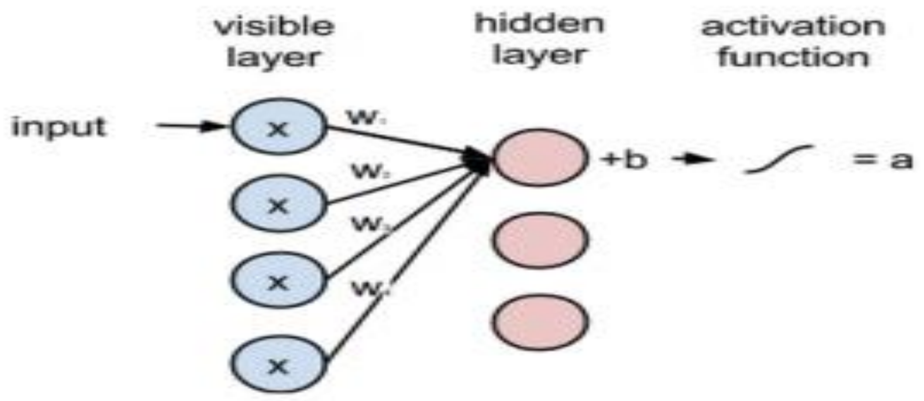
Restricted Boltzmann Machines

- ❑ Seen visible and hidden layers of RBMs, but not yet seen how they learn features.
- ❑ Each of the visible layer's nodes take in a single feature from the dataset to be learned from.
- ❑ This data is then passed from the visible layer to the hidden layer through weights and biases.



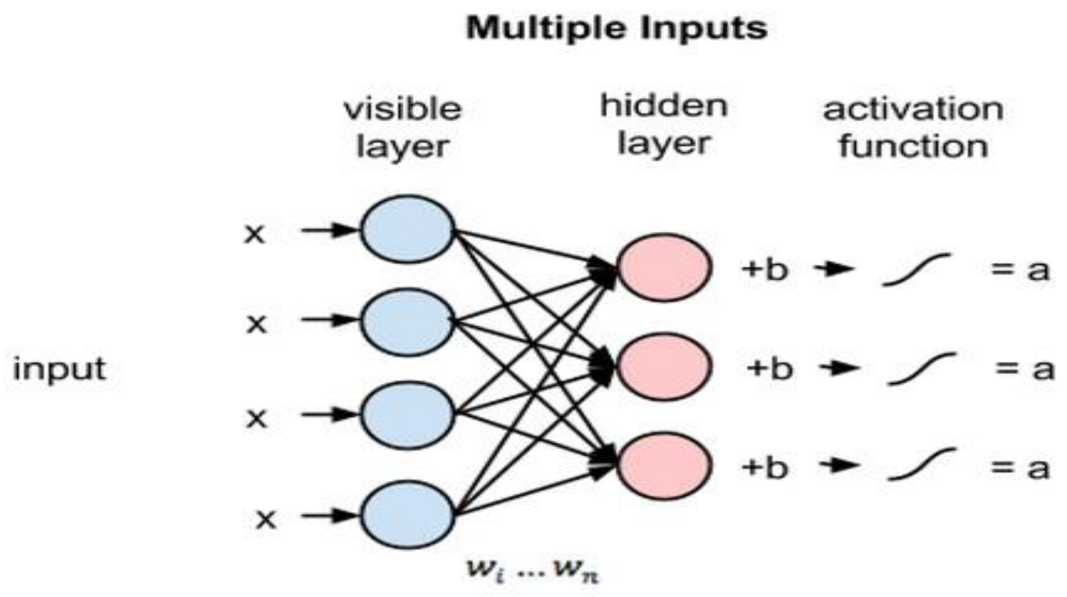
Restricted Boltzmann Machines

- RBM shows the movement of a single data point through the graph and through a single hidden node.
- The visible layer has four nodes, representing the four columns of the original data.
- Each arrow represents a single feature of the data point moving through the four visible nodes in the first layer of the RBM.
- Each of the feature values is multiplied by a weight associated to that feature and are added up together.
- This calculation can also be summed up by a dot product between an input vector of data and a weight vector.
- The resulting weighted sum of the data is added to a bias variable and sent through an activation function (sigmoidal is popular).
- The result is stored in a variable called a.



Restricted Boltzmann Machines

- In a real RBM, each of the visible nodes is connected to each of the hidden nodes.
- Because inputs from each visible node are passed to every single hidden node, an RBM can be defined as a **symmetrical bipartite graph**.
- The symmetrical part comes from the fact that the visible nodes are all connected with each hidden node.
- Bipartite means it has two parts (layers).



Restricted Boltzmann Machines

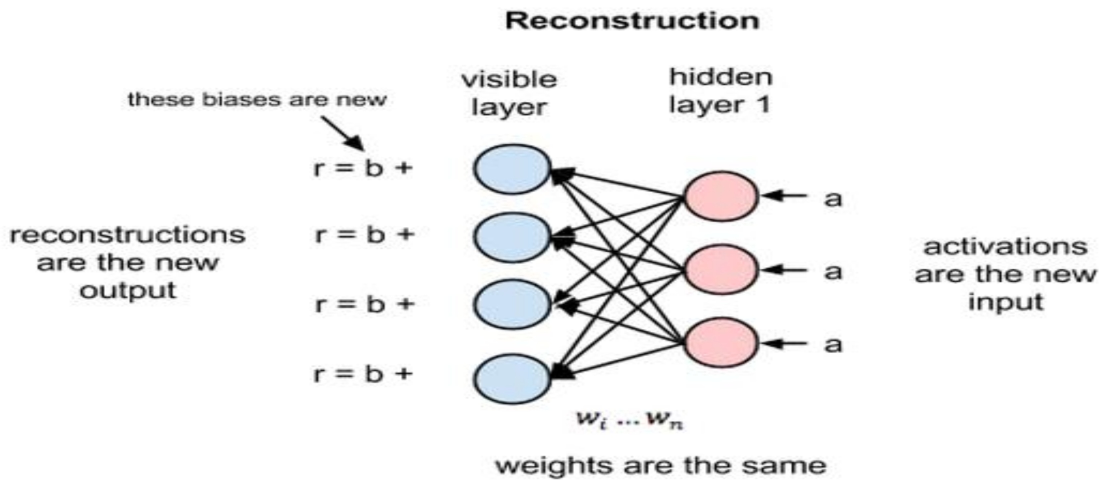
The restriction of a Boltzmann Machine

- ❑ With our two layers of visible and hidden nodes, we have seen the connection between the layers (inter-layer connections), but we haven't seen any connections between nodes in the same layer (intra-layer connections).
- ❑ That is because there aren't any.
- ❑ The restriction in the RBM is that we do not allow for any intra-layer communication.
- ❑ This lets nodes independently create weights and biases that end up being (hopefully) independent features for our data.

Restricted Boltzmann Machines

Reconstructing the data

- In this forward pass of the network, we can see how data goes forward through the network (from the visible layer to the hidden layer), but that doesn't explain how the RBM is able to learn new features from our data without ground truths.
- This is done through multiple forward and backward passes through the network between our visible and hidden layer.
- In the reconstruction phase, we switch the network around and let the hidden layer become the input layer and let it feed our activation variables (a) backwards into the visible layer using the same weights, but a new set of biases.
- The activated variables that are calculated during the forward pass are then used to reconstruct the original input vectors.



Restricted Boltzmann Machines

MNIST dataset

- ❑ MNIST dataset consists of 6,000 images of handwritten digits between zero and nine and a ground-truth label to learn from.
- ❑ It is not unlike most of the other datasets that we have been working with in that we are attempting to fit a machine learning model to classify a response variable given a set of data points.
- ❑ The main difference here is that we are working with very low-level features as opposed to more interpretable features.
- ❑ Each data point will consist of 784 features (pixel values in a grey-scale image).

Restricted Boltzmann Machines

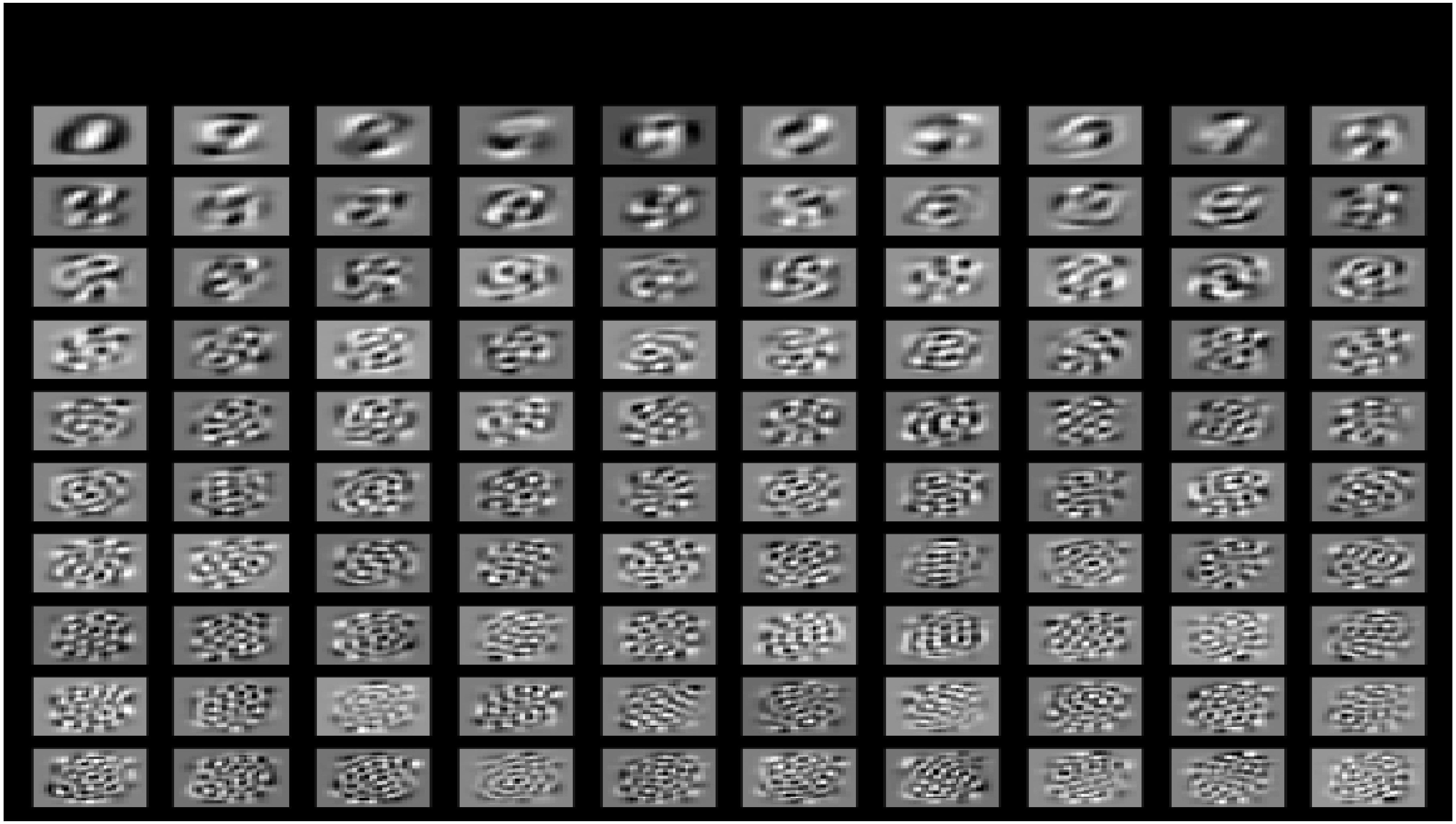
https://github.com/PacktPublishing/Feature-Engineering-Made-Easy/blob/master/Chapter07/Ch_7.ipynb

The BernoulliRBM

- ❑ scikit-learn implemented version of a RBM is called **BernoulliRBM**
- ❑ It imposes a constraint on the type of probability distribution it can learn.
- ❑ The Bernoulli distribution allows for data values to be between zero and one.
- ❑ The scikit-learn documentation states that the model *assumes the inputs are either binary values or values between zero and one*.
- ❑ To account for this, we will alter our dataset to account for only hardcoded white/black pixel intensities.
- ❑ By doing so, every cell value will either be zero or one (white or black) to make learning more robust.
- ❑ We will accomplish this in two steps:
  1. We will scale the values of the pixels to be between zero and one
  2. We will change the pixel values in place to be true if the value is over 0.5, and false otherwise

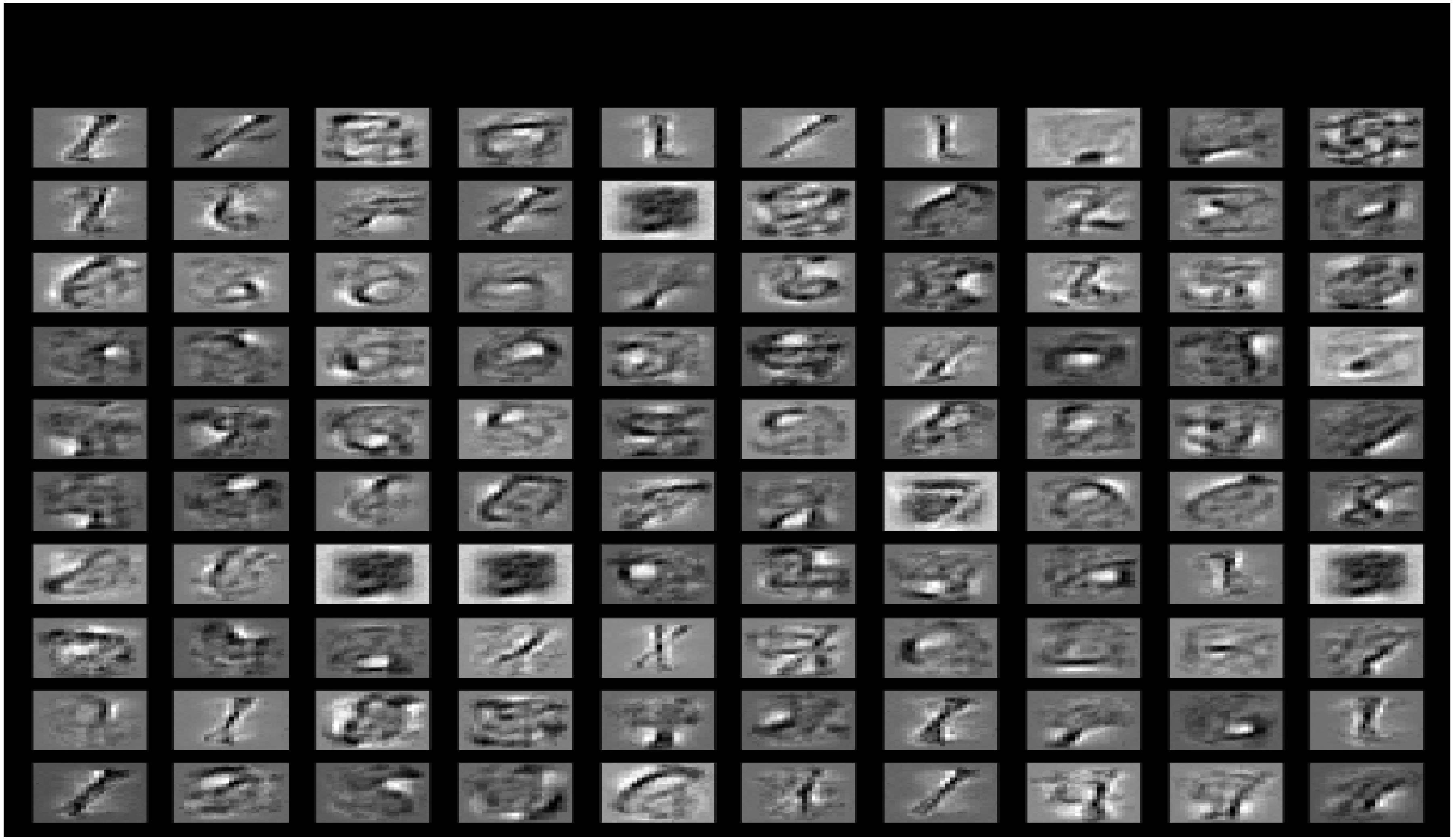
Extracting PCA components from MNIST

Let's take 100 components from the possible 784 and plot the components to see what the extracted features look like.



Extracting RBM components from MNIST

- ❑ Let's now create our first RBM in scikit-learn.
- ❑ Instantiate a module to extract 100 components from our MNIST dataset.



Learning text features – word vectorizations

- ❑ Able to vectorize documents by counting the number of words appeared in each document.
- ❑ By constructing new count-based features, we were able to use text in our supervised machine learning pipelines.
- ❑ This is very effective, up until a point.
- ❑ They were only a **Bag of Words** (**BOW**).
- ❑ This means that we regard documents as being nothing more than a collection of words out of order.
- ❑ What's more is that each word on its own has no meaning.
- ❑ It is for this reason that we will turn our attention away from scikit-learn and onto a module called gensim for computing word embeddings.

Word embedding

- Word embedding is any of a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers.
- Conceptually it involves a mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension.
- Methods to generate this mapping include
  - neural networks
  - dimensionality reduction on the word co-occurrence matrix,
  - probabilistic models,
  - explainable knowledge base method,
  - explicit representation in terms of the context in which words appear

Word embedding

- Word embedding is one of the most popular representation of document vocabulary.
- It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.
- What are word embeddings exactly?
  - Loosely speaking, they are vector representations of a particular word.
  - Having said this, what follows is how do we generate them?
  - More importantly, how do they capture the context?
- Word2Vec is one of the most popular technique to learn word embeddings using **shallow neural network**.
- It was developed by **Tomas Mikolov in 2013 at Google.**

Word embedding

For example, if we were to ask you the following questions, you may give the following answers:

Q: *What would you get if we took a king, removed the man aspect of it, and replaced it with a woman?*

A: A queen

Q: *London is to England as Paris is to ____.*

A: France

- ❑ *Human, find these questions simple, but how would a machine ?*
- ❑ *This is, in fact, one of the greatest challenges that we face in natural language processing (NLP) tasks.*

Word embedding

- Word embeddings are one approach to helping a machine understand context.
- A **word embedding** is a vectorization of a single word in a feature space of n dimensions, where *n* represents the number of latent characteristics that a word can have.
- This means that every word in our vocabulary is not longer, just a string, but a vector in and of itself.
- For example, if we extracted n=5 characteristics about each word, then each word in our vocabulary would correspond to a 1 x 5 vector.
- For example, we might have the following vectorizations:

Word embedding

- For example, we might have the following vectorizations:

```
# set some fake word embeddings
king = np.array([.2, -.5, .7, .2, -.9])
man = np.array([-.5, .2, -.2, .3, 0.])
woman = np.array([.7, -.3, .3, .6, .1])
queen = np.array([ 1.4, -1. , 1.2, 0.5, -0.8])
```

- We can tackle the question *What would you get if we took a king, removed the man aspect of it, and replaced it with a woman?* by performing

  king - man + woman

  In code, this would look like:

```
np.array_equal((king - man + woman), queen)
True
```

Word embedding

Few caveats:

- ❑ Context (in the form of word embeddings) changes from corpus to corpus as does word meanings.
- ❑ Static word embeddings by themselves are not always the useful
- ❑ Word embeddings are dependent on the corpus that they were learned from.
- ❑ Word embeddings allow us to perform very precise calculations on single words to achieve what we might consider context.

Word embedding

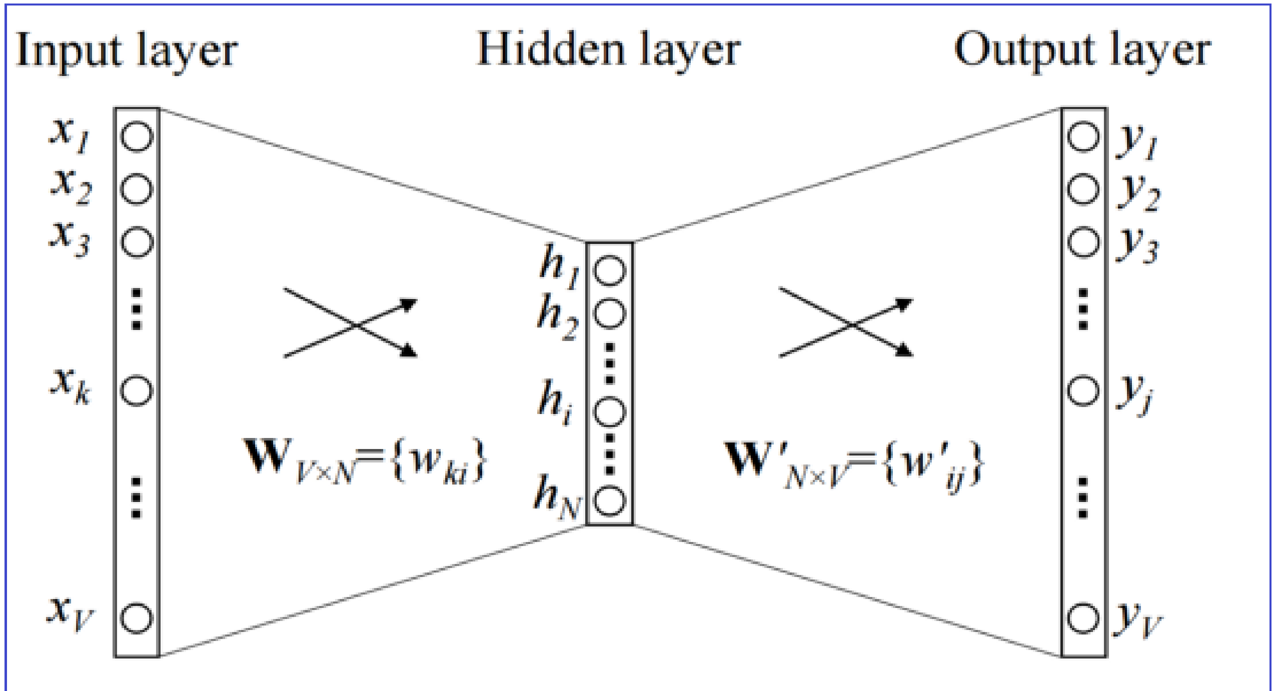
- **Two approaches to word embeddings**
  1. Word2vec and
  2. GloVe
- Both methods are responsible for producing word embeddings by learning from very large corpus.
- GloVe algorithm (Stanford) learns word embeddings through a series of matrix statistics
- Word2vec (Google), learns them through a deep learning approach.

How does Word2Vec work?

- Word2Vec is a method to construct such an embedding.
- It can be obtained using two methods (both involving Neural Networks):
  1. Skip Gram and
  2. Common Bag Of Words (CBOW)

How does Word2Vec work?

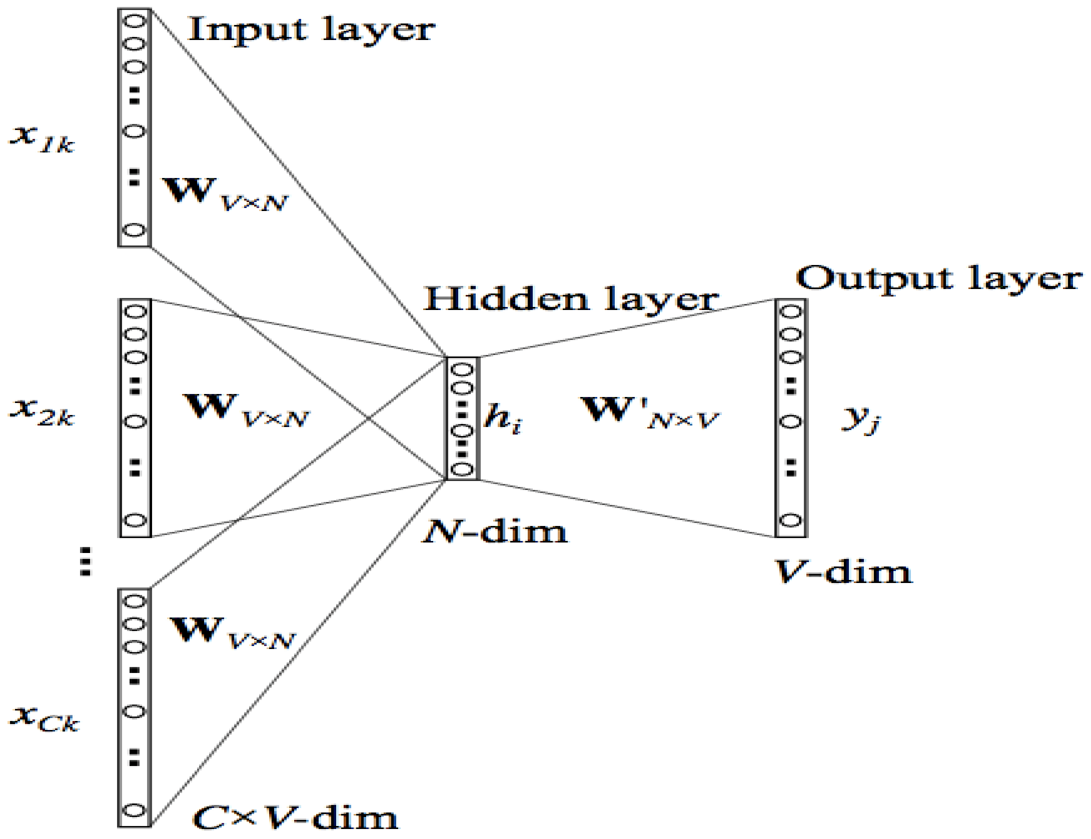
CBOW Model: This method takes the context of each word as the input and tries to predict the word corresponding to the context.



A simple CBOW model with only one word in the context

How does Word2Vec work?

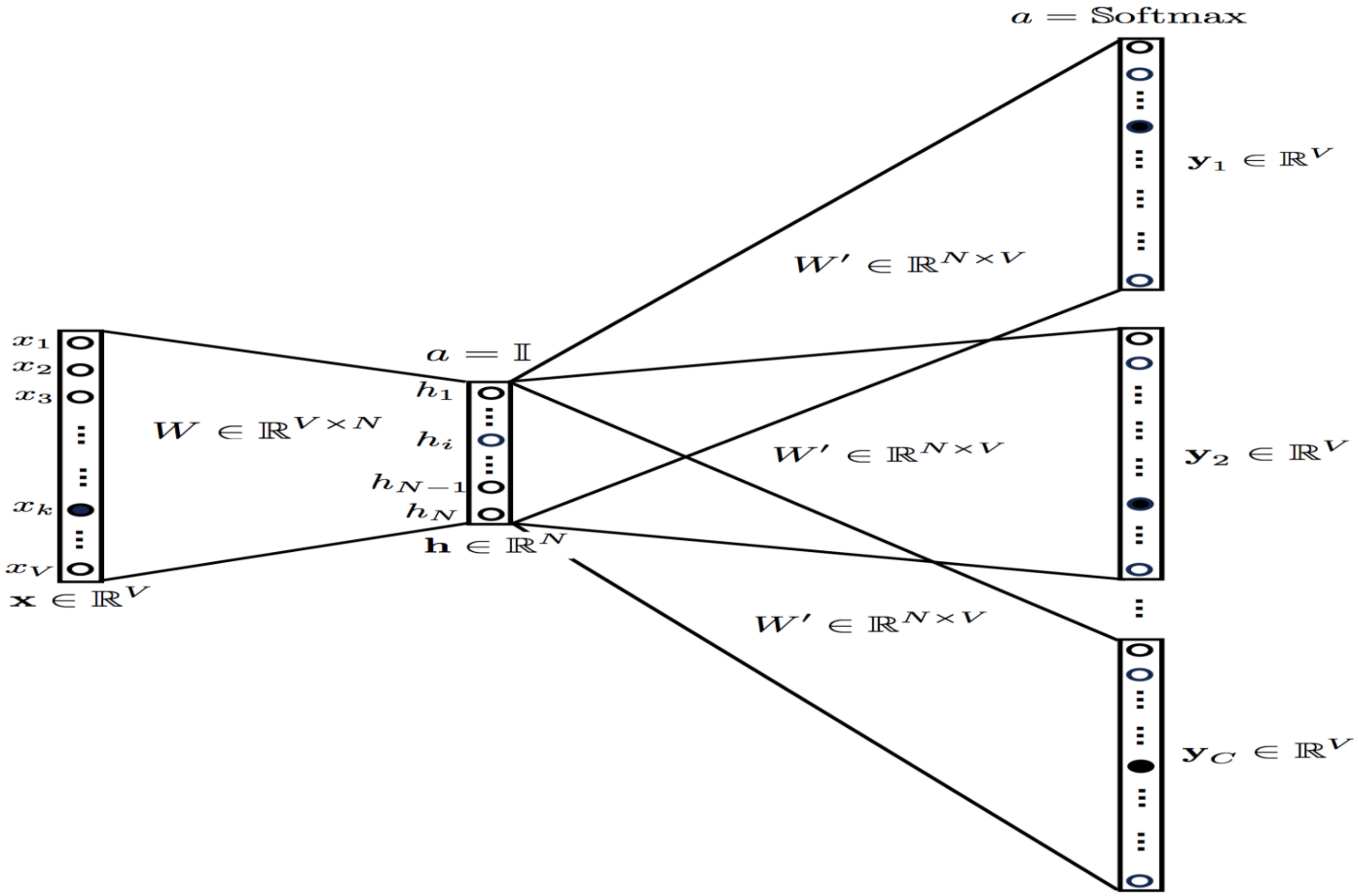
- ❑ Earlier model used a single context word to predict the target.
- ❑ Can use multiple context words to do the same.



How does Word2Vec work?

Skip-Gram model:

This looks like multiple-context CBOW model just got flipped.


$a = \text{Softmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
x_1
x_2
x_3
x_k
x_V
$\mathbf{x} \in \mathbb{R}^V$
$W \in \mathbb{R}^{V \times N}$
$a = \mathbb{I}$
h_1
h_i
h_{N-1}
h_N
$\mathbf{h} \in \mathbb{R}^N$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_C \in \mathbb{R}^V$

Word2Vec - gensim package

# import the gensim package

```
import genism
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s’,
level=logging.INFO)
```

Now, let's create our corpus:

```
from gensim.models import word2vec, Word2Vec
sentences = word2vec.Text8Corpus('../data/text8’)
# instantiate a gensim module on the sentences from above
# min_count allows us to ignore words that occur strictly less than this
value
# size is the dimension of words we wish to learn
model = gensim.models.Word2Vec(sentences, min_count=1, size=20)
```

Word2Vec - gensim package

```
# get the vectorization of a word

model.wv['king’]

array([-0.48768288, 0.66667134, 2.33743191, 2.71835423, 4.17330408,
2.30985498, 1.92848825, 1.43448424, 3.91518641, -0.01281452, 3.82612252,
0.60087812, 6.15167284, 4.70150518, -1.65476751, 4.85853577, 3.45778084,
5.02583361, -2.98040175, 2.37563372], dtype=float32)
```

Word2Vec - gensim package

The gensim has built-in methods to get the most out of our word embeddings.
For example, to answer the question about the king, we can use the the most_similar method:

```
# woman + king - man = queen
model.wv.most_similar(positive=['woman', 'king'], negative=['man'],
topn=10)
```

```
[(u'emperor', 0.8988120555877686), (u'prince', 0.87584388256073),
(u'consul', 0.8575721979141235), (u'tsar', 0.8558996319770813),
(u'constantine', 0.8515684604644775), (u'pope', 0.8496872782707214),
(u'throne', 0.8495982885360718), (u'elector', 0.8379884362220764),
(u'judah', 0.8376096487045288), (u'emperors', 0.8356839418411255)]
```

- **Unfortunately this isn't giving us the answer we'd expect: queen.**

Word2Vec - gensim package

Can use pre-trained embeddings.
We can do this by using built-in importer tools in gensim:

```
# use a pretrained vocabulary with 3,000,000 words
import gensim
model =
gensim.models.KeyedVectors.load_word2vec_format('../data/Google
Newsvectors- negative300.bin', binary=True)

# 3,000,000 words in our vocab
len(model.wv.vocab)
3000000
```

Word2Vec - gensim package

- Trained using vastly more powerful machines
- For a much longer period of time.
- Let's try our word problems out now:

```
# woman + king - man = queen
model.wv.most_similar(positive=['woman', 'king'], negative=['man’], topn=1)

        [(u'queen', 0.711819291114807l)]

# London is to England as Paris is to ____
model.wv.most_similar(positive=['Paris', 'England'], negative=['London’],
topn=1)

        [(u'France', 0.6676377654075623)]
```

Word embeddings

Application of word embeddings – information retrieval

- ❑Countless applications for word embeddings
- ❑One of these is in the field of information retrieval.
- ❑When humans input keywords and key phrases into search engines, search engines are able to recall and surface specific articles/stories that match those keywords exactly.
- ❑For example, if we search for articles about dogs, we will get articles that mention the word dog.
- ❑Can use word embeddings in search engines

Summary

- Focused on two feature learning tools: RBM and word embedding processes.
- Both of these processes utilized deep learning architectures in order to learn new sets of features based on raw data.
- Both techniques took advantage of shallow networks in order to optimize for training times and used the weights and biases learned during the fitting phase to extract the latent structure of the data.

Thank You !!!

https://www.youtube.com/watch?v=clXGCgR6Vng

https://towardsdatascience.com/restricted-boltzmann-machines-simplified-eab1e5878976