Applications of Language models in Educational Assessment

Christopher Ormerod, Amir Jafari, and Susan Lottridge

NCME 2022

Cambium Assessment

What we seek to achieve

What

Our goal is to introduce the following:

- Machine Learning and NLP.
- How to use machine learning in Automated Scoring.
- Demonstrate the abilities of Language models.
- Apply pretrained language models to Automated Scoring.
- Give some consideration to some of the problems and difficulties in each of these areas.

Cambium Assessment

Why we are doing this?

Why

There are several advantages to using pretrained Language models in scoring texts that are generally true:

- High agreement between the engine and resolved scores relative to other neural-network based models.
- They require less data to train than other neural-network based models.
- Models that are more robust to different types of input and misspellings.
- They are easier to train than some other neural network-based approaches.

Cambium Assessment

Schedule

8:00 - 9:00 - Machine Learning

9:15 - 10:15 - Pandas, sci-kit-learn, and BoW models and Metrics

10:30 - 11:30 - Language Models

1:00 - 2:00 - Pytorch and Neural Networks

2:15 - 3:15 - Recurrent Neural Network

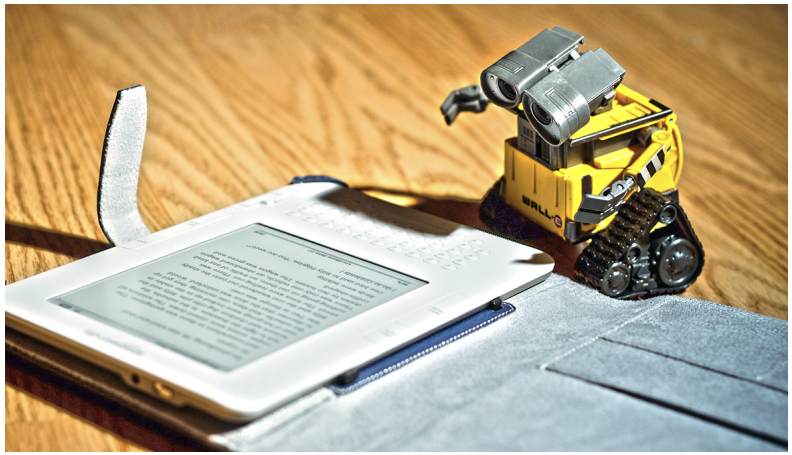
3:30 - 4:30 - Transformers, Language models and Fine tuning

Cambium Assessment

Colab

https://github.com/5hogun-Ormerod/Machine-learning-Python/blob/master/NCME.ipynb

Machine Learning



Aims

- To understand what machine learning is and what is NLP, and what problems it seek to solve.
- Establish an understanding of a few key methods; Logistic and Linear regression, random forests, and k-means clustering.
- Establish an understanding of some of the problems we face in deploying machine learning.

Cambium Assessment

Machine Learning

Machine Learning

Machine Learning describes a class of algorithms and models whose performance improves by using data.

Within machine learning, we have three broad categories:

- **Supervised Learning**: Where a set of data is provided with a label and the algorithms attempt to match the data with the labels. E.g., Classification tasks.
- **Unsupervised Learning**: Where a set of data is provided without labels and the algorithms attempt to characterize the data. E.g., Clustering.
- **Reinforcement Learning**: An algorithm whose output interacts with a dynamic environment to maximize a goal function. E.g., AlphaZero.

Cambium Assessment

Loss Functions

Within each task, we can associate a loss function that can be either maximized or minimized:

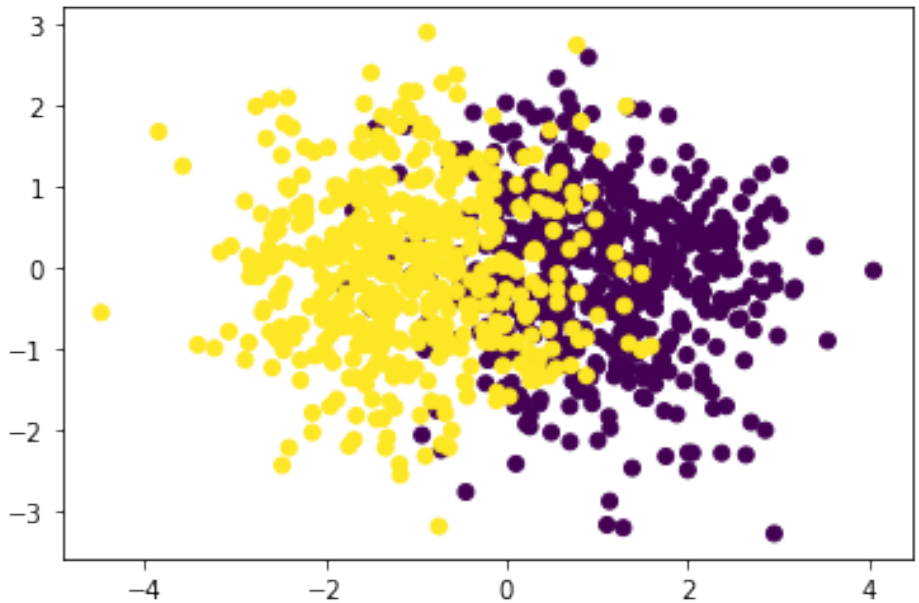
- **Classification**: We can minimize the difference between the predicted labels (or their probabilities) and the true labels.
- **Regression**: We can minimize the difference between points and a parameterized curve.
- **Clustering**: We can minimize the distances between cluster centers and points in each cluster.
- **Reinforcement learning**: We want to maximize a value function like a win-rate or score.

For each algorithm, we have a set of parameters, A, and set of data, X. The loss function should reflect the task.

Cambium Assessment

Logistic Regression

Suppose we had a collection of two-dimensional points, each of which was one of two classes; 0 or 1.



We can assume that the 0's and 1's come from two separate distributions.

Cambium Assessment

Logistic Regression

Each point is given by a row (x_0, x_1) in a matrix X and their corresponding labels, 0 or 1, are stored in a vector, y.

The i-th row of X labelled by the i-th element of y.

Let $a = [a_0, a_1, a_2]$ be a vector, and

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

be the sigmoid function. Let

$$f(x) = \sigma(a_0 x_0 + a_1 x_1 + a_2)$$

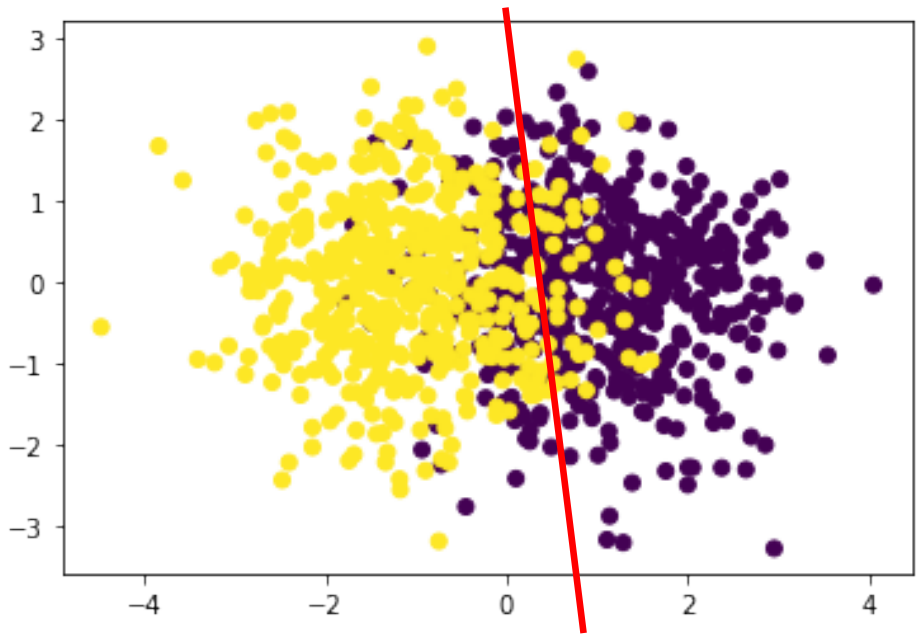
Cambium Assessment

Logistic Regression

For each a we can sum the square of the distance between the sigmoid function and the label. We call this a Loss function:

$$L(a) = \sum_{p \in X} |f(p_i) - y_i|^2.$$

Logistic regression is finding the $a \in \mathbb{R}^3$ so that this function is minimized. In visual terms, it is about separating these points by a line ($a_0 x_0 + a_1 x_1 + a_2 = 0$).

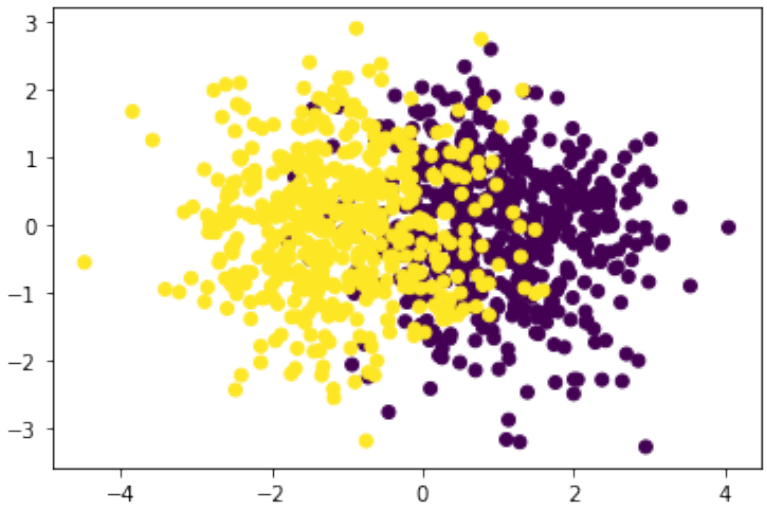


Cambium Assessment

Creating the data

We start by creating some data; 500 random points centered at $(1, 0)$ and 500 at $(-1, 0)$, each labelled 0 or 1:

```
X1 = np.random.multivariate_normal([1,0],
[[1,0],[0,1]], 500)
X2 = np.random.multivariate_normal([-1,0],
[[1,0],[0,1]], 500)
X = np.concatenate([X1,X2],0)
y = np.concatenate([np.zeros(500),np.ones(500)])
```



In code

Our loss function is

$$L(a) = \sum_{p \in X} |f(p_i) - y_i|^2.$$

which we assign the function $loss(a)$ as follows:

```
sigmoid = lambda x: 1/(1-np.exp(-x))
f = lambda a,x:sigmoid(a[0]*x[0]+a[1]*x[1] + a[2])
dist = lambda a,x,y: (f(a,x) - y)**2
loss = lambda a:sum([dist(a,x,y) for x,y in zip(X,y)])

a0 = [-10,0,1]
res = scipy.optimize.minimize(loss, a0)
a = res['x']
```

This code finds an appropriate a from an initial guess of $a_0 = (-10, 0, 1)$.

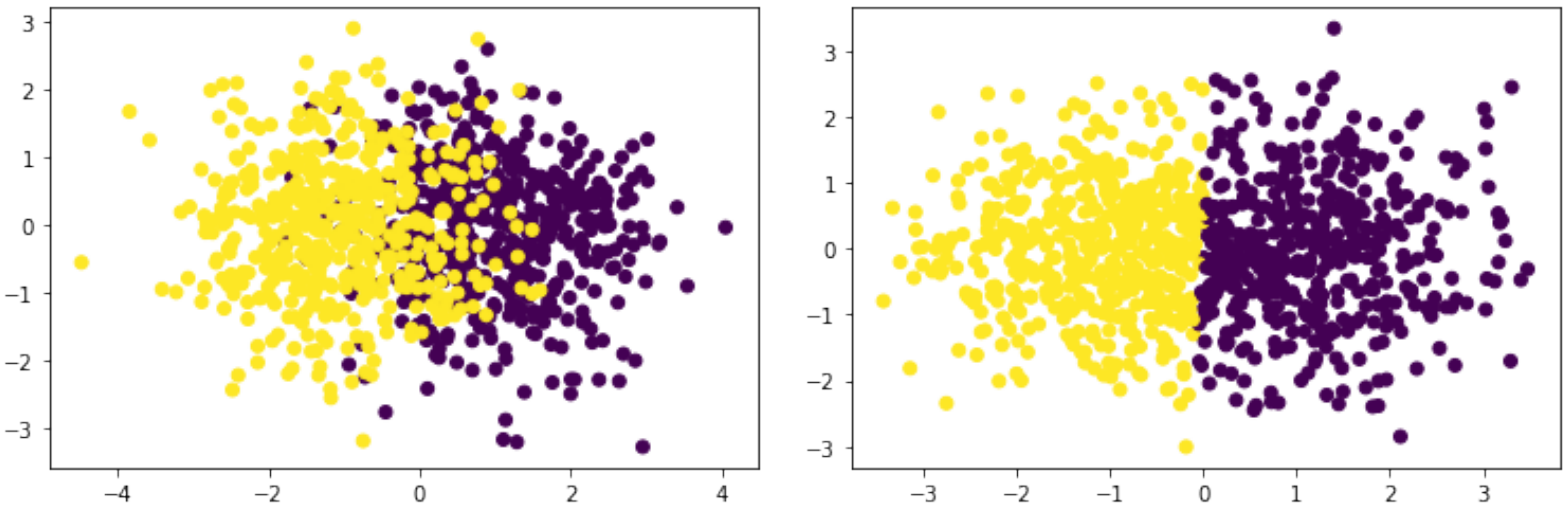
Cambium Assessment

Predictions

Once we know $a = (a_0, a_1, a_2)$, we can determine whether a point is 0 or 1 using

$$p(x) = \begin{cases} 0 & \text{If } \sigma(a_0 x_0 + a_1 x_1 + a_2) < 1/2 \\ 1 & \text{Otherwise} \end{cases}$$

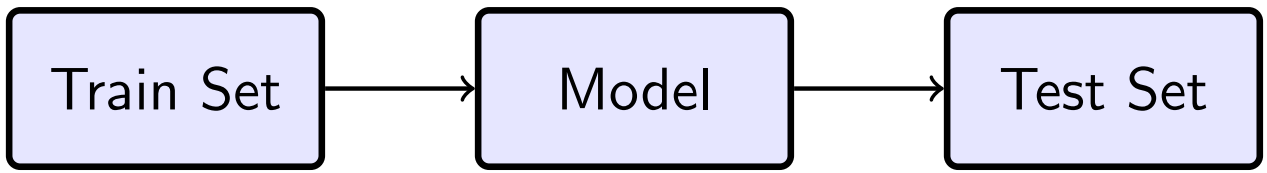
giving us a good picture as to how this method works.



On the left we have the original, on the right we have predictions.

Model Selection

Generally, we want to know how this model works on unseen data. For models with very high numbers of parameters (neural networks), there can be a large difference between performance.

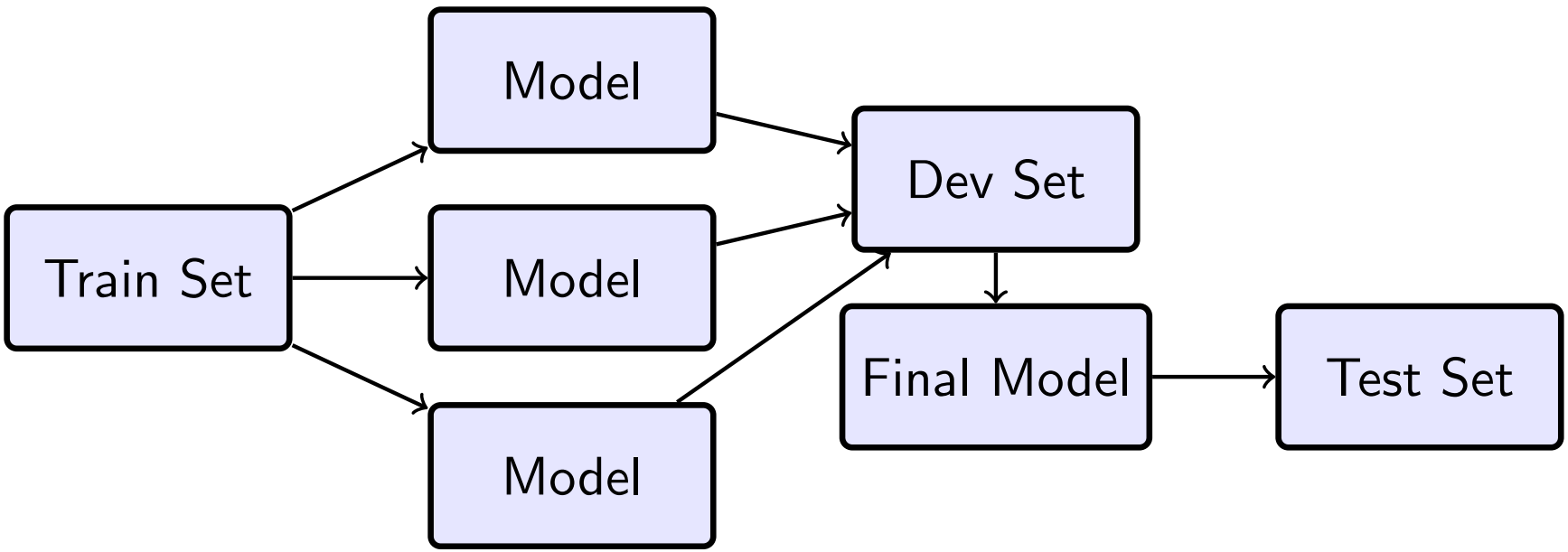


To get an accurate picture of how a model performs of unseen data, we need to perform a train-test split to report how the model performs on unseen data.

Cambium Assessment

Model Selection

Generally, we want to know how this model works on unseen data. For models with very high numbers of parameters (neural networks), there can be a large difference between performance.



In the case of choosing one model from multiple models, we also need a development set. Choosing a model from multiple models can also inaccurately reflect unseen data.

Cambium Assessment

Multi-label classification

Given a set of labels $\{1, \ldots, n\}$, our job is to distinguish between each class,

$$C_i = \{\text{set of points with label } \ i\}.$$

Two ways of doing this are as follows:

- **One-vs-Rest**: We build n classifiers, each is a binary classifier

$$f_i(x) = \begin{cases} 0 & \text{if } x \notin C_i \\ 1 & \text{if } x \in C_i \end{cases}$$

- **One-vs-One**: We build $\binom{n}{2}$ classifiers, for each i, j,

$$f_{i,j}(x) = \begin{cases} 0 & \text{if } x \in C_i \\ 1 & \text{if } x \in C_j \end{cases}$$

Cambium Assessment

One-vs-Rest

A one-vs-rest classifier we have the following data

- X is the $N \times d$-dimensional matrix with N points in $\mathbb{R}^d$.
- $\tilde{X} = (X|1)$ is a $N \times (d+1)$ with a final column containing 1s.
- A is a $(d+1) \times n$ dimensional matrix of parameters.
- Y is an $N \times n$ dimensional matrix that is 0 in entry i, j if point i is in C_j.

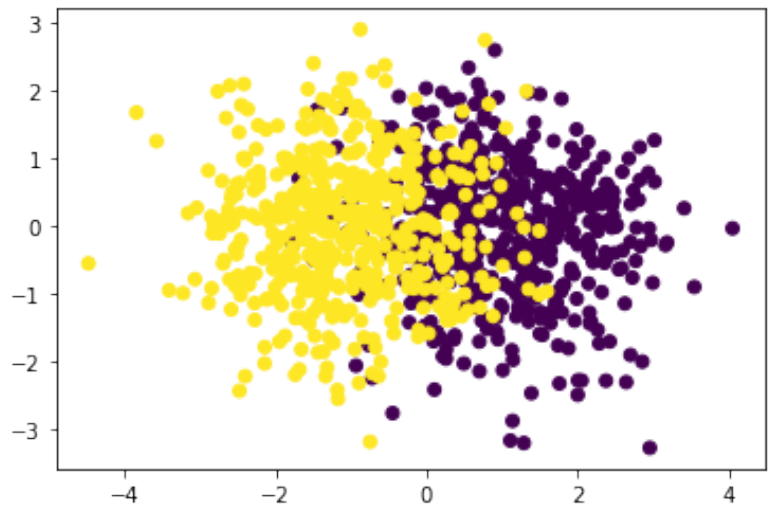
We can formulate this problem as a minimization of

$$F(A) = \sum_{i,j} |\sigma(\tilde{X}A) - Y|^2_{i,j}$$

which is a $n(d+1)$-dimensional minimization.

Decision Trees

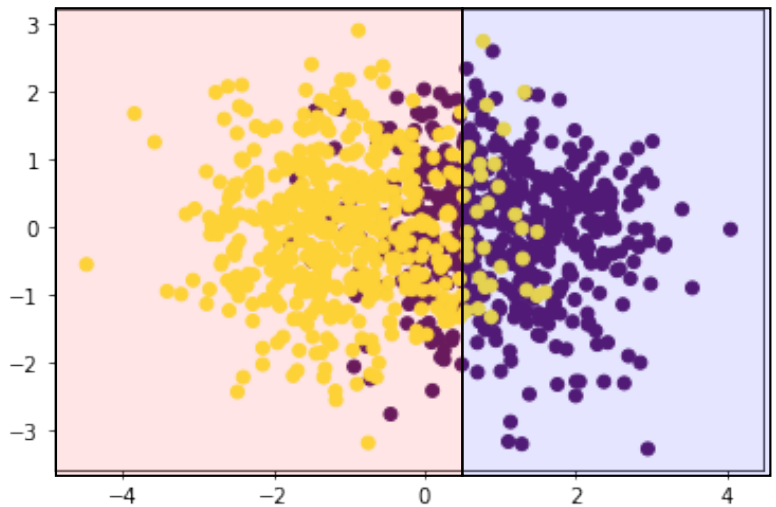
A decision tree is a different structure that looks to optimize information gain between the parent and children.



Decision Trees

A decision tree is a different structure that looks to optimize information gain between the parent and children.

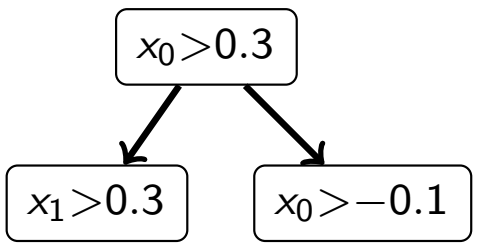
$x_0 > 0.3$

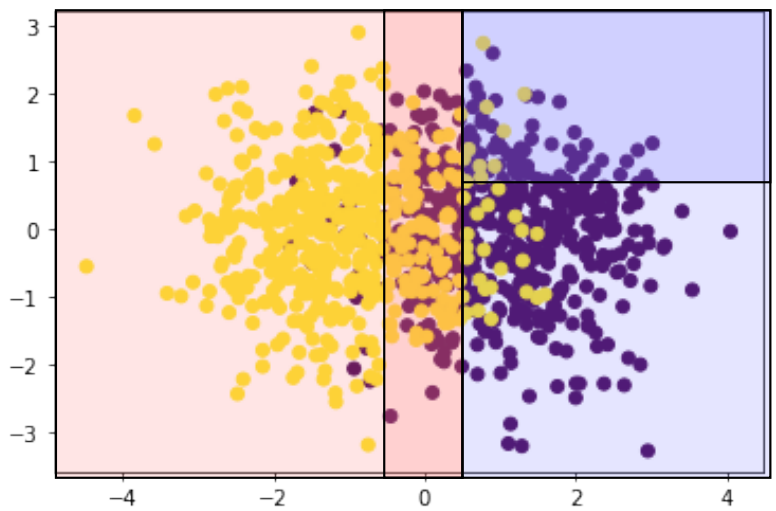


Cambium Assessment

Decision Trees

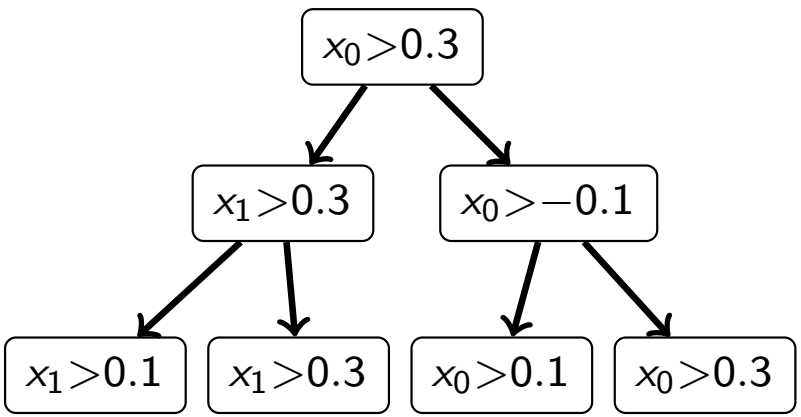
A decision tree is a different structure that looks to optimize information gain between the parent and children.

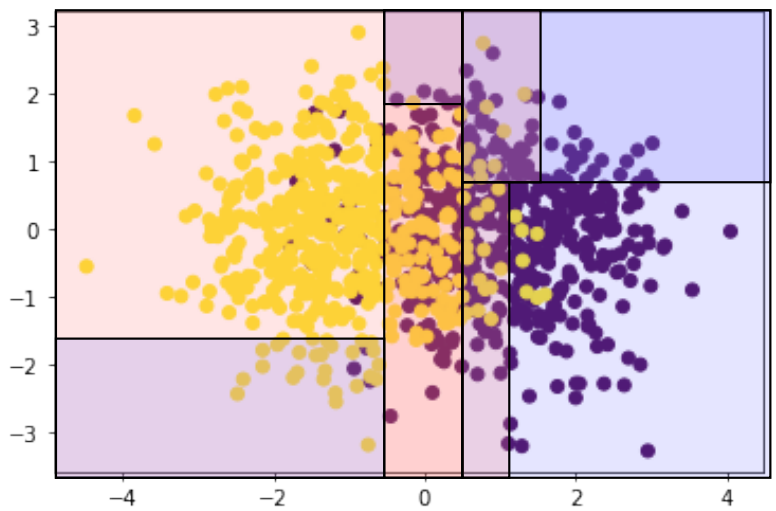




Decision Trees

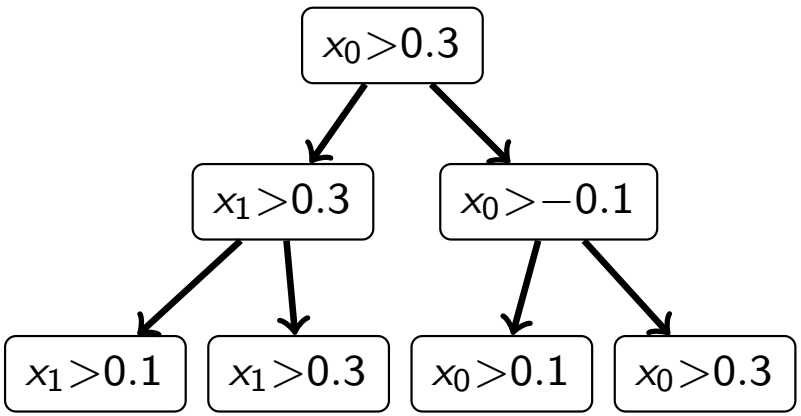
A decision tree is a different structure that looks to optimize information gain between the parent and children.

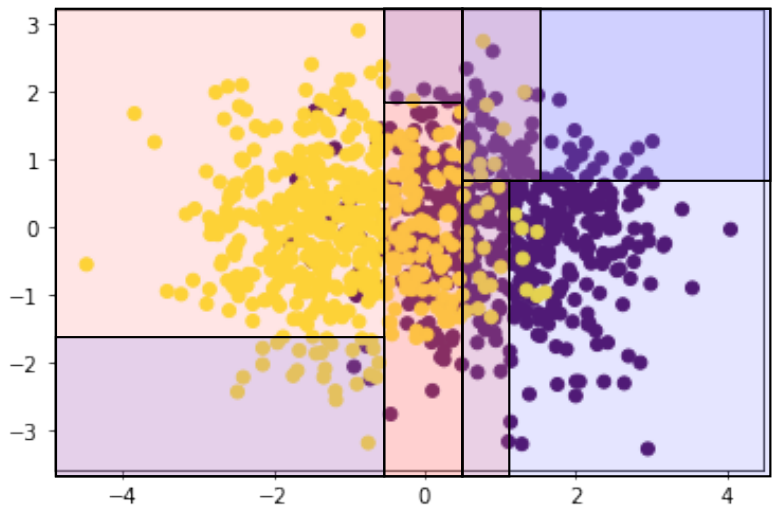




Decision Trees

A decision tree is a different structure that looks to optimize information gain between the parent and children.





We could continue to divide up the plane until every single region has only one class.

Overfitting

The main problem with decision trees is that while it is accurate on the training set, it often does not generalize to unseen data well.

Figure: The best way to explain overfitting.

Regularization in machine learning

Regularization

Regularization in machine learning is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

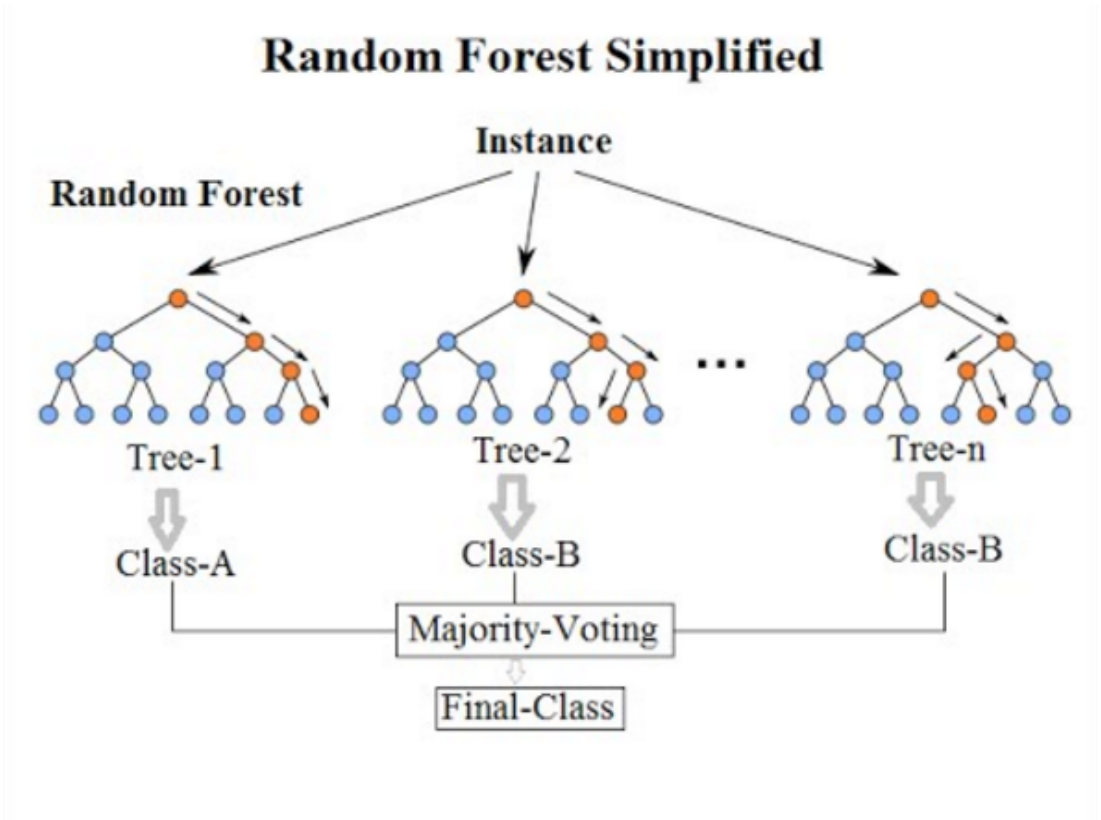
Some ways to modify a machine learning algorithm are as follows:

- Use methods with very very few parameters.
- Modify the loss function to constrain parameters.
- Constrain the size of allowable steps in iterative methods.
- Randomly hide data in training iterations.

Cambium Assessment

Random Forest

A modification of the decision tree is the random forest: we break the data up and ensemble smaller decision trees:



Generally, the training accuracy decreases and the validation accuracy increases.

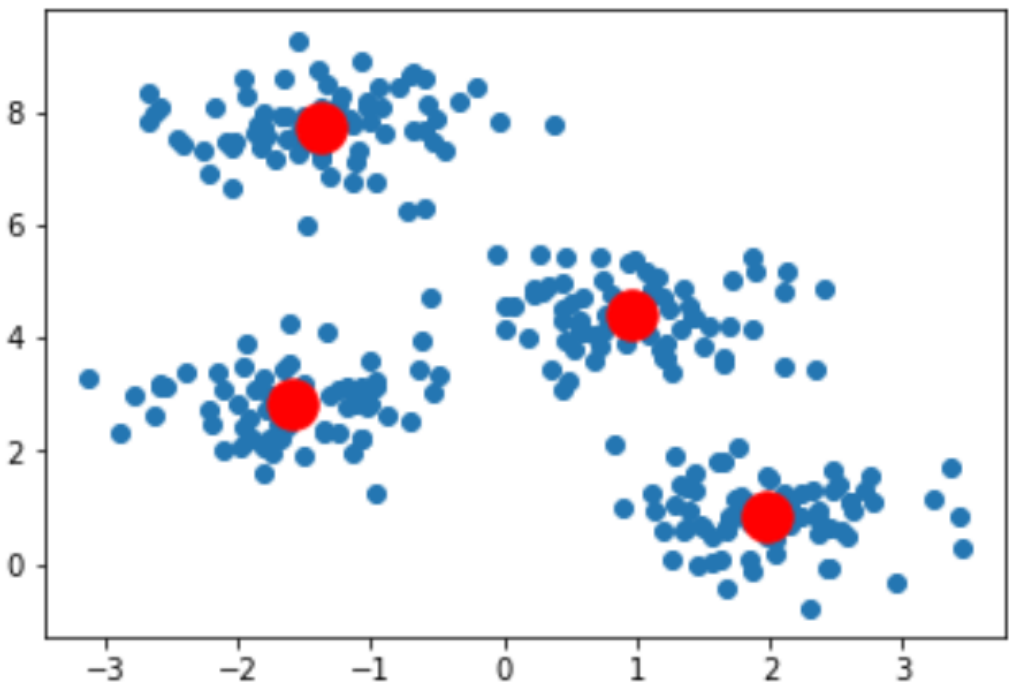
Cambium Assessment

K-means clustering

Given a set of unlabeled points, x_i, we initialize K centroids, $\{\mu_1, \ldots, \mu_k\}$. We seek to minimize

$$L = \sum_{i=1}^{N} \sum_{k=1}^{K} r_{n,k} ||x_i - \mu_k||^2$$

where $r_{i,k}$ is the indicator that x_i belongs to cluster k.



Assessment

In assessment, we have two major areas of application:

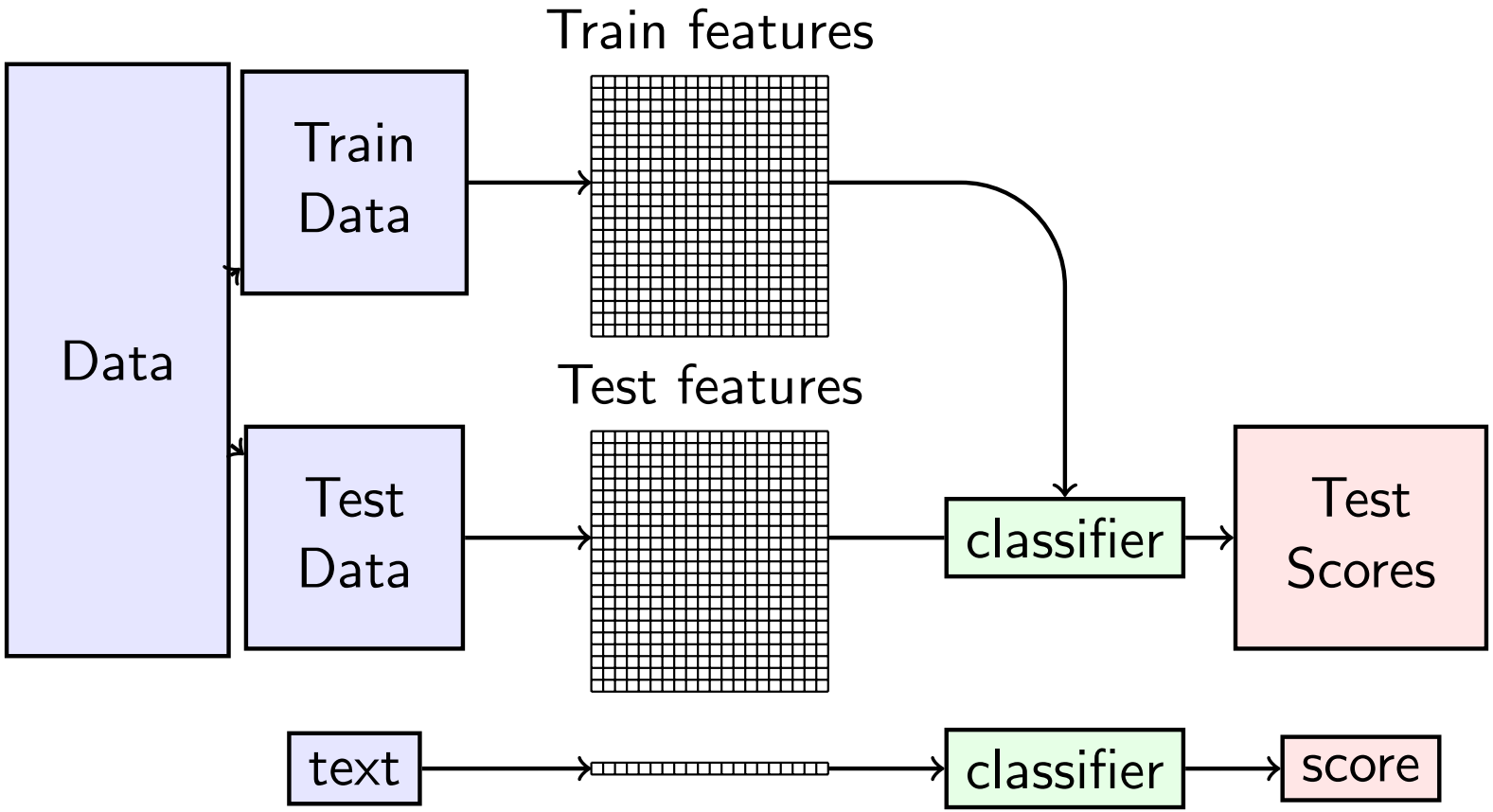
- **Automated Essay Scoring**: This is a supervised classification task in which a text is scored in accordance with a holistic rubric in a variety of traits. E.g., Conventions, Organization/Purpose, Elaboration.
- **Automated Short Answer Scoring**: Also a supervised classification task where a short text is scored with a rubric is around specific comprehension or analytic skills.

Auxiliary and more applications in the research realm include text normalization (spelling, grammar), feedback generation, response time prediction for items, item difficulty prediction, crisis paper alerting, and item enemy detection

Cambium Assessment

Using Machine Learning

Most text classification pipelines operates on a core principle; we extract a core collection of features and apply a machine learning classifier to those features:



What is natural language processing?

Text classification fits into the broader category of application of Natural Language Processing (NLP), which is possesses two main sub-categories:

1- Natural Language Understanding (NLU): text classification, summarization, question answering.

2- Natural Language Generation (NLG): Translation, word prediction, speech recognition.

Applications include

Search	Web Search	Documents Search	Autocomplete Words
Editing	Spelling Error	Grammar Correction	Style of Writing
Dialog	Chatbot	Speech Assistant	Question answering
Email	Spam	Classification	Ordering
Text mining	Summarization	Knowledge extraction	Topic modeling
News	Find Events	Fact checking	Headline detection
Sentiment analysis	Reviews	Product review	Customer care
Behavior prediction	Medical	Election forecasting	Business
Creative writing	Movie scripts	Transcriptions	Song lyrics

Bias

Let us give an example of bias in algorithms. We can use BERT as a word predictor:

fill in the blank

This man works as a _____?.

1. carpenter
2. lawyer
3. businessman

fill in the blank

This woman works as a _____?

1. nurse
2. maid
3. teacher

Cambium Assessment

Bias in assessment

Whenever we deploy a machine learning solution, we have to be aware of potential bias. There are a few ways to take this into account:

- Monitor any bias inherent in the data. E.g. language models taken from Wikipedia and bookscorpus. Monitor representation across subgroups.
- Monitor any changes in the performance for particular subgroups (using matched sampling).
- Be transparent about the methods used.
- Be aware the way demographics may change the nature of the data.
- Seek methods that mitigate any bias.

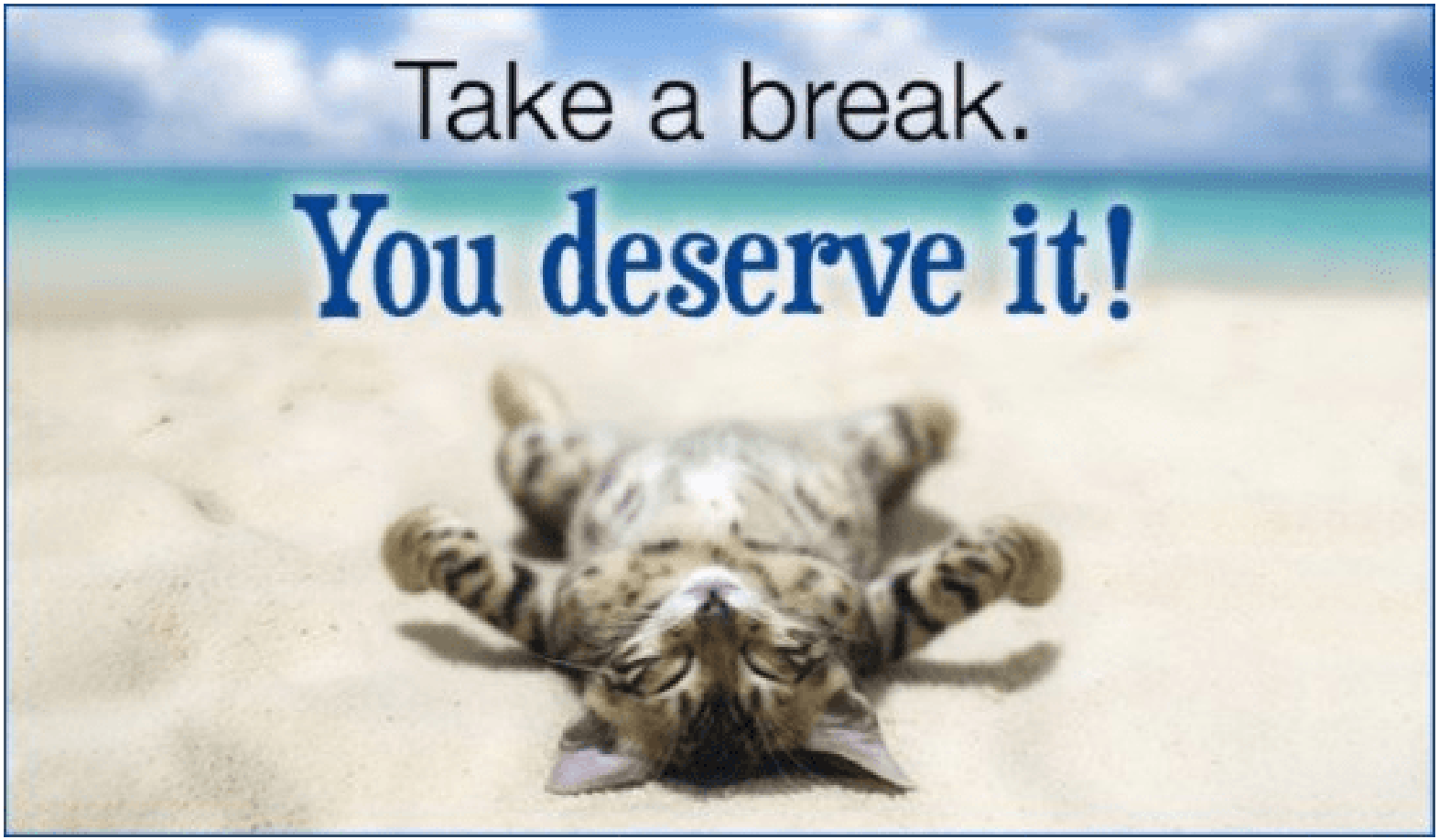
Cambium Assessment

Review Session 1

- We have introduced examples of the type of problems we seek to solve:
  - Classification
  - Regression
  - Clustering
- We have introduced some basic machine learning techniques:
  - Logistic Regression
  - Random Forest
  - K-means
- We have introduced some key problems in deploying machine learning:
  - Overfitting
  - Model Selection
  - Model Bias

Cambium Assessment

Take a break



Pandas, sci-kit-learn, BoW models, and Metrics

Aims

- To introduce the Panel-Data (panda) library as a way to load, save, and manipulate data.
- To present the scikit-learn library to implement a range of machine learning algorithms.
- To combine the above to implement a simple machine-learning automated essay scorer.
- To discuss the sorts of metrics we apply to automated text scoring systems.

Cambium Assessment

Experiementing with dataframes

We can create, save and load data to and from excel, csv, latex, SQL, html, pickle, json, ...

```
>> import pandas as pd
>> import numpy as np

>> df = pd.DataFrame(np.random.randn(6, 4),
      columns=['A','B','C','D'])
          A         B         C         D
0 -0.267845  0.233413 -0.913104 -0.285044
1  3.156970  0.588336  0.662703  0.508503
2 -0.532408 -0.711655 -1.147137  0.251682
3  0.964220 -1.737531 -2.745309 -0.494637
4  0.256017 -0.030337 -0.511307 -1.362467
5  0.707023 -0.127959 -0.040311  0.388600

>> df.save_excel("my_excel.xlsx")
>> df = pd.read_excel("my_excel.xlsx")
```

Subframes

We access elements using loc (index) or iloc (relative position) and drop elements with respect to index:

```
>> df1 = df.sample(2)
          A         B         C         D
5  0.707023 -0.127959 -0.040311  0.388600
3  0.964220 -1.737531 -2.745309 -0.494637
>> df2 = df[df['A'] < 0]
          A         B         C         D
0 -0.267845  0.233413 -0.913104 -0.285044
2 -0.532408 -0.711655 -1.147137  0.251682
>> df1.iloc[0,'A']
0.707023
>> df2.loc[2,'A']
-0.532408
>> df = df.drop(5)
```

Some import operations

We can extract and manipulate data using column selection and map/apply. Map applies a function to a column:

```
>> df['E'] = df['A'].map(lambda x:x**2)
```

Apply applies a function to a frame by rows or columns:

```
>> df['E'] = df.apply(lambda x:x['A']**2,1)
```

The result is equivalent.

Sci-kit-learn

We had three major areas of traditional machine learning tasks and a collection of algorithms that perform the task:

- **Classification** : Support vector machines, Random Forest, Naive Bayes, Logistic Regression, ...
- **Regression**: Linear Regression, Generalized Linear Regression, Voting Regressor, ...
- **Clustering**: K-means, mean-shift, spectral clustering,...

Sci-kit-learn simplifies the training and prediction functions for all these methods.

Cambium Assessment

A sample dataset

We use an sklearn dataset to demonstrate how to use sklearn:

```
>> from sklearn.datasets import load_diabetes
>> from sklearn.model_selection import train_test_split
>> data = load_diabetes(as_frame=True)
>> columns = data.feature_names
>> df = data['frame']
>> X = df[data.feature_names]
>> y = df['target']
```

We derive two problems; targets y are between 0 and 400 and $z \approx y/100$. We fit to y using regression and

```
>> X_train, X_test, y_train, y_test= train_test_split(X,y)
>> z_train = y_train.map(lambda x:int(x/100))
>> z_test = y_test.map(lambda x:int(x/100))
```

We can use y for regression and z for classification (4 classes). Options are available to stratify.

Cambium Assessment

Classifiers

Sci-kit-learn has implementations of a range of classifiers.

```
>> from sklearn.linear_model import LogisticRegression
>> from sklearn.metrics import accuracy_score

>> clf = LogisticRegression()
>> clf.fit(X_train, z_train)
>> preds = clf.predict(X_test)

>> print(accuracy_score(preds,z_test))

0.495495495495495
```

We used accuracy to measure agreement.

Cambium Assessment

Clustering

Sci-kit-learn has implementations of a range of clustering methods.

```
>> from sklearn.cluster import KMeans
>> from sklearn.metrics import accuracy_score

>> clf = KMeans()
>> clf.fit(X_train, z_train)
>> preds = clf.predict(X_test)

>> print(cohen_kappa_score(preds,z_test, weights="quadratic"))
>> print(accuracy_score(preds,z_test))

0.17117117117117117
```

We used accuracy to measure agreement.

Cambium Assessment

Regression

Sci-kit-learn has implementations of a range of Regressors:

```
>> from sklearn.linear_model import LinearRegression
>> from scipy.stats import spearmanr

>> clf = LinearRegression()
>> clf.fit(X_train,y_train)
>> preds = clf.predict(X_test)

>> print(spearmanr(preds, y_test))

SpearmanrResult(correlation=0.6065146873569046,
pvalue=1.7267304600102265e-12)
```

We used Spearmans r correlation to measure agreement.

Cambium Assessment

Automated Scoring

We are now in a position to consider an Automated scoring task. It could be either

- Automated Essay Scoring
- Automated Short answer Scoring.

In either case, we have the following continuous or categorical data.

	Id	text	label
0	7265	text	3
1	1974	some text	0
2	8740	some more text	4
3	2892	extra text	1
4	6429	additional text	3
5	4526	even more text	1
6	7273	even more additional text	3
7	3350	superfluous text	1

The label could be a trait or an overall score.

Setting up a dataset

Let us take a well known example, movie reviews, as an example of the task. For each text, the label is 0 (negative) or 1 (positive):

```
>> import pandas as pd
>> from datasets import load_dataset
>> train_data = load_dataset("imdb", split="train")
>> test_data = load_dataset("imdb", split="test")
```

We can directly convert these to dataframes

```
>> train_df = pd.DataFrame(train_data)
>> test_df = pd.DataFrame(test_data)
```

We can check the distribution of scores using a Counter:

```
>> from collections import Counter
>> Counter(train_df['label'])
>> Counter(train_df['label'])
Counter({0: 12500, 1: 12500})
Counter({0: 12500, 1: 12500})
```

Bag of Words

We have the following data:

- Let $D = D_{train} = \{d_1, \ldots, d_n\}$ be all the documents in training.
- Let $V = \{v_1, \ldots, v_M\}$ be all the words that appear in the training.
- Let $F = (f_{i,j})$ be the frequency matrix where $f_{i,j}$ is the frequency of v_j in d_i.

The term frequency is value of

$$\mathrm{tf}(v_j, d_i) = \frac{f_{i,j}}{\sum_k f_{k,d}}$$

while the inverse-document-frequency is

$$\mathrm{idf}(v_j) = \log \frac{N}{|\{d \in D : v_j \in D\}|}$$

Cambium Assessment

TF-IDF

Using these two components, we get a TF-IDF transformation

$$\text{tf-idf}(v_j, d_i) = \text{tf}(v_j, d_i) \cdot \text{idf}(v_j)$$

This induces a transformation

$$F : \text{text} \to \mathbb{R}^M$$

where $M = |V|$. This gives us matrices for training and testing

$$X_{train} = F(D_{train}), \tag{1}$$
$$X_{test} = F(D_{test}). \tag{2}$$

The number of dimensions here are typically in the thousands.

Dimension reduction use LSA

Latent semantic analysis (LSA) is a dimensional reduction of the tf-idf matrix using singular value decomposition. So

$$X_{train} = U\Sigma V^*$$

where Σ possesses all the eigenvalues in decreasing order. Choosing the largest d eigenvalues with associated columns of U and V gives

$$G : X \to X_d, \qquad X_d = U_d \Sigma_d V_d^*$$

which is a general transformation

$$G : \mathbb{R}^M \to \mathbb{R}^d$$

where $d << M$.

Cambium Assessment

Implementation

Scikit-learn has in-built functions for tf-idf and singular value decomposition:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

vect = TfidfVectorizer()
Z_train = vect.fit_transform(train_df['text'])
Z_test = vect.transform(test_df['text'])
y_train = train_df['label']
y_test = test_df['label']

LSA = TruncatedSVD(n_components=300)
X_train = LSA.fit_transform(Z_train)
X_test = LSA.transform(Z_test)
```

This gives dimension reduced X and y values.

Cambium Assessment

Main Metrics

We typically report on three main metrics in the following order of importance:

- **QWK**: Quadratic Weighted Kappa measures the agreement above and beyond pure chance.
- **SMD**: Standardized Mean Difference is the difference between the mean scores divided by the standard deviation.
- **Accuracy**: The exact agreement between scores.

These are outlined in a standard paper on automated scoring metrics:

- Williamson, David M., Xiaoming Xi, and F. Jay Breyer. *A framework for evaluation and use of automated scoring.* Educational measurement: issues and practice **31**, no. 1 (2012), 2–13.

Cambium Assessment

Evaluating scoring systems

The first step is to collect the data. We want to ensure that the data is of a high quality, so there are various steps we can take:

- We first curate a sample of responses to a particular response.
- Discard any junk responses that might adversely affect training.
- We have each response scored by two independent raters.
- If our two raters agree, we take that score. If they disagree, we have the score adjudicated by a third rater.
- Ensure that two raters have an acceptable agreement.

This becomes our dataset for training any engine.

Cambium Assessment

Evaluating our scoring system

The scikit-learn framework has these metrics

```
from sklearn.metrics import (cohen_kappa_score,
                             accuracy_score)
smd = lambda x,y: np.abs(np.mean(x) - np.mean(y))/
np.sqrt((np.std(x)**2+np.std(y)**2)/2)
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print(cohen_kappa_score(y_pred,y_test,
weights="quadratic"))
print(accuracy_score(y_pred,y_test))
print(smd(y_pred, y_test))
```

This gives us Accuracy of 86.76%, an SMD of 0.015, and a QWK of 0.735

Cambium Assessment

Adding Features

The LSA features tend to align words with high semantic similarity. E.g., smart and intelligent, business and workplace, money and currency. We can endow the LSA vectors with some meaning.

We could improve these results by adding features to the LSA vectors:

1. Prompt specific features such as prompt overlap and keywords.
2. Syntactic features like the number of nouns, verbs, adjectives.
3. Errors such as grammatical mistakes and misspellings.
4. Length-based features like the number of sentences, words, average sentence length.
5. Readability features like word difficulty, automated readability indices, syllable counts.

We need to ensure that these features are well aligned with the rubric.

Cambium Assessment

Typical Criteria

Typically, the automated scoring system is based on double-scored data (initial and reliability reads) with a possible resolved score. One would report:

- **IRR statistics**: QWK, SMD, and Accuracy between the two raters used to obtain the training data.
- **AS statistics**: QWK, SMD, and Accuracy between the automated scoring engine and a resolved score.
- **Subgroup SMDs**: The SMD for relevant subgroups between the automated scoring engine and a resolved score.

What we look for in our automated scoring engines are the follows:

- Differences in QWK of less than 0.1 and SMD less than 0.15.
- SMD for relevant subgroups less than 0.15 to indicate minimal bias.

Cambium Assessment

Is the model adding bias

SMD does not take into account the differences between subgroup sample abilities:

- Let S be a subgroup.
- Repeatedly subsample from the entire population with the same resolved score distribution. This allows us to model the distribution of automated score while holding the measured student ability relatively constant.
- Examine the subgroup performance with respect to this distribution of automated scores.
- Take into account additional covariates like Title I status. E.g., using additional stratification in the subsampling regime.
- Once bias is detected, it is important to know whether the differences in responses might be that is causing bias.

Better standards of evaluating bias is still an active research area.

Cambium Assessment

Other operational concerns

Given an automated scoring engine, there are other concerns worth noting:

- **Hotline**: How does an engine respond to responses in which the student indicates they are a danger to themselves or others?
- **Drift**: Do student responses change over time? Does an appropriate human interpretation of the rubric change over time?
- **Human-machine scoring**: Do you want full automated scoring or have humans involved in the process (changing scores, backreads etc.)?
- **Confidence**: Is there a way to flag responses that the automated engine is uncertain of?

Cambium Assessment

Review Session 2

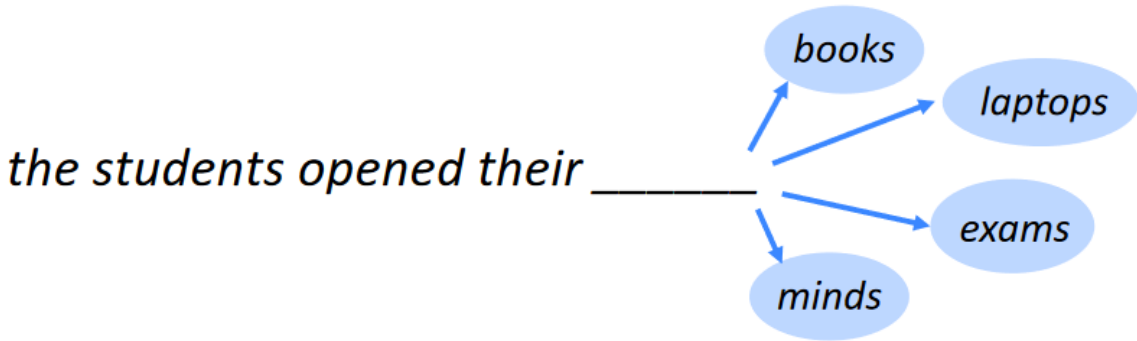
- We introduced pandas as a means to manage data.
- We presented scikit-learn as a way to implement standard algorithms.
- We implemented a Bag-of-Words method:
  - Used and defined the TD-IDF matrix.
  - Performed the Latent Semantic Analysis
  - Applied a classifier to this data.
- Explained the metrics and practices used to validate and monitor scoring systems:
  - Introduced the main metrics used in monitoring and reporting.
  - Considered some operational concerns.

Cambium Assessment

Take a break

Cambium Assessment

Classical Language Models



Aims

- To understand the what we mean by a language model.
- Explore how we can use word embeddings to extend language models.
- Understand the use of subwords to address the issues around traditional language modes and word-embeddings.

Cambium Assessment

Language Models

Language Model

A language model is a statistical model designed to predict the probability of a sequence of words occurring in a text.

Givn a vocabulary, V, and a sequence of words, $text = w_0, \ldots, w_n$ we want to estimate

$$P(text) = P(w_0, \ldots w_n)$$

in some distribution of texts.

Cambium Assessment

Uses for language models

We can use such a probability function to find:

- **Aberrant Responses**: Is a sequence of words very different from what is expected

$$P(text) < \epsilon$$

- **Next words**: Given a sequence of words, $w_0, \ldots, w_n$, predict w_{n+1}. I.e., maximize the function

$$f(v) = P(w_0, \ldots, w_n v)$$

for $v \in V$.

- **Masked words**: Given a sequence $w_0 \ldots w_n$, where some words are masked (not known), predict the masked words. I.e., maximise the function

$$f(v) = P(w_0, ..., v, \ldots w_n)$$

Cambium Assessment

Student data

We start this segment by isolating a set of appropriate data:

```
from datasets import load_dataset

source = "NahedAbdelgaber/evaluating-student-writing"

tokenizer = RegexpTokenizer(pattern=r'\w+|\$[\d\.]+|\S+')
text_dataset = load_dataset(source,
split='train')

train, test = train_test_split(text_dataset['text'])
train_text = ' '.join(train).lower()
test_text = ' '.join(test).lower()
```

Cambium Assessment

n-grams

The simplest way to build a language model is simply store *n*-grams. Using nltk we store all 3 grams.

```
all_text = ' '.join(text_dataset['text']).lower()
words = tokenizer.tokenize(all_text)
data = ngrams(words,n=3)
counts = Counter(data)
```

We can use these counts to predict the next word by randomizing but weighting those random choices by frequency:

```
def random_next(word1, word2):
  my_counts = {x:v for x,v in counts.items()
               if word1 == x[0] and word2 == x[1]}
  return np.random.choice([x[2] for x in my_counts],
                  p=[x[1]/sum(my_counts.values())
                  for x in my_counts.items()])
```

Cambium Assessment

n-grams

The output of starting with the words "i am" is the following:

> *i am not trying to collect data from a teacher designs the projects , such as procrastination and laziness is not good and the trouble if good for them so it can help each other with new electors were kind of emotions shes having because she knows way more about its gonna be a simpler overview on the road .*

Markov-assumption

By the chain rule, we have

$$P(x_1, x_2, ..., x_N) = P(x_1)P(x_2|x_1)...P(x_N|x_1, ...x_{N-1})$$

Then by the Markov-assumption for trigrams states

$$P(x_N|x_1, ...x_{N-1}) \approx P(x_N|x_{N-1}x_{N-2})$$

in which case our probabilities are given by

$$P(x_1, x_2, ..., x_N) \approx P(x_1)P(x_1x_2)\prod_{k=3}^{N} P(x_k|x_{k-1}x_{k-2}),$$

allowing us to use our stored trigrams to calculate the probability of a given sentence.

Cambium Assessment

Sparsity

It is often easier to deal with log probabilities, in which case,

$$\log P(x_1, x_2, ..., x_N) \approx \log P(x_1) + \log P(x_1 x_2) + \sum_{k=3}^{N} \log P(x_k | x_{k-1} x_{k-2}),$$

however, if x_k is a word not in the vocabulary the

$$logP(x_1 \ldots, x_n) = -\infty$$

Even if x_k is rare, unless the corpus this was built on was huge, we would need to see every word in every viable context, which is almost impossible.

Cambium Assessment

Why discounted models

We can shift some of the probabilities of words we know to unknown words:

```
train, test = train_test_split(text_dataset['text'],
test_size=0.5)

train_words= tokenizer.tokenize(train_text)
test_words= tokenizer.tokenize(test_text)

T1 = Counter(train_words)
T2 = Counter(test_words)
```

How often do words with frequency k in train appear in test?

Cambium Assessment

Discounts

By looking at the average frequencies on test we find something interesting:

```
>> words[i] = lambda k:{w:k for w,k in T1.items() if k==i}
>> {i:np.mean([T2[w] for w in words[i]])
for i in range(1,8)}
{1: 0.4428625377643505,
 2: 1.4194687289845327,
 3: 2.367684946632315,
 4: 3.390062821245003,
 5: 4.270450751252087,
 6: 5.313860252004582,
 7: 6.42514970059802}
```

If they appear in train with frequency k, on average they appear with frequency $k - \delta$ for $\delta = 0.6$ on test.

Conversely, a good approximation for words appearing in test but NOT in test is about $0.46 \approx 0.44$.

Kneser-Ney

For bigrams we have that if $c(w_{i-1}, w_i)$ is the count of the bigram (w_{i-1}, w_i), then

$$P(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_c(w_i)$$

where

$$P_c(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}}$$

and

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w : c(w_{i-1}, w) > 0\}|$$

I.e., $P_c(w_i)$ is the normalized number of words that can go before w_i and $\lambda(w_{i-1})$ the number of words that can follow w_{i-1}. (Both functions of d and the raw counts).

Implementation

There are a number of implementations of the KneserNey model to import.

```
from knlm import KneserNey
mdl = KneserNey(3, 4)
for x in text_dataset['text']:
  mdl.train(tokenizer.tokenize(x.lower()))
mdl.optimize()
```

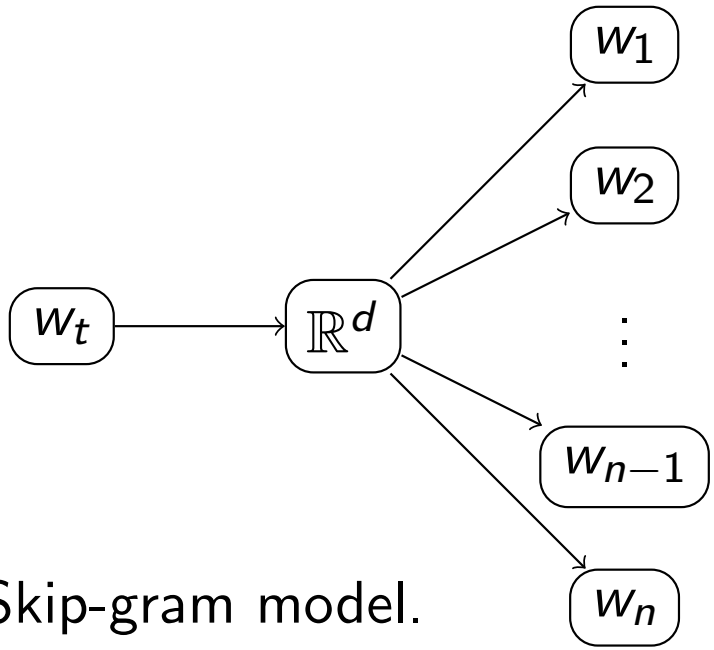
One the model has been compiled, we can use it to evaluation sentence probabilities

```
print(mdl.evaluateSent("hello , my name is alexander".split()))
print(mdl.evaluateSent("hello , my name is john".split()))
```

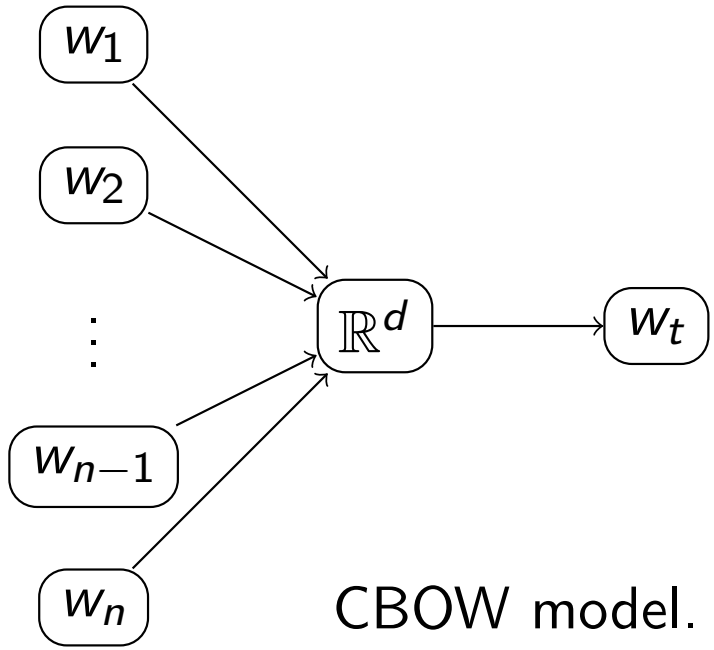
Naturally, changing the name less common names should have lower scores.

Types of Embedding

There are two types of Word embeddings: Skip-gram and Continuous Bag-of-Words (CBOW):



Skip-gram model.



CBOW model.

Skip gram predicts probabilities of context words based on a target, the CBOW predicts a target word based on context.

Probabilities from a skip-gram model

We can find the probability of a target word.

- If w_o is a target word with vector v_o.
- If w_i is a context word with vector v_i.

then the probability of w_o appearing in context w_i is

$$P(w_o|w_i) = \frac{\exp(v_o \cdot v_i)}{\sum_j \exp(v_i \cdot v_j)}$$

where v_j is the word for w_j in a vocabulary. Given a set of context vectors, we can maximize

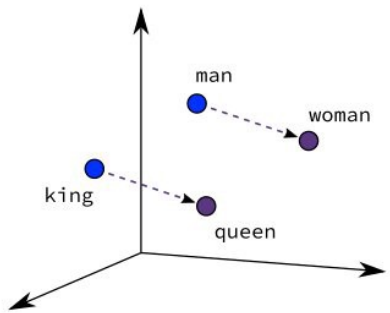
$$\log \prod_i P(w_o|w_i) = \sum_i \log P(w_o|w_i)$$

to find a missing word.

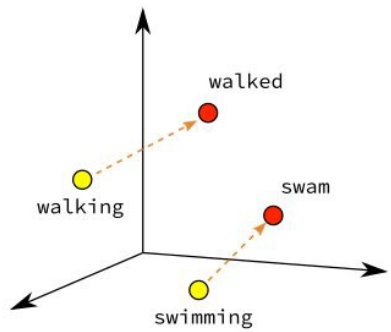
Cambium Assessment

Relations in embeddings

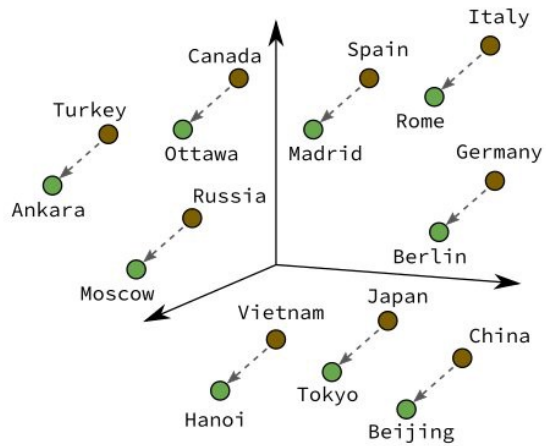
When we impose these conditions, we readily find that embeddings encode semantic relations:



Male-Female



Verb Tense



Country-Capital

The differences between the vectors for difference verb tenses, masculine and feminine, countries and their capitals, are very similar. Even at this level, embeddings can possess bias.

Making embeddings

The gensim library allows us to efficiently train embedding models:

```
import gensim

model = gensim.model.Word2Vec()
model.train(corpus)
```

Given a set of context words, we use the gensim api

```
model.most_similar("smart")

L = ["hotdog", "burger", "fries"]
model.predict_output_words(context_words_list=L,
topn=10)
```

Note: Given a small corpus, this list is not very accurate.

Cambium Assessment

Teh porblme wif woords

There are two main issues with words:

- **Sparsity:** Zipf's law states that the normalized frequency of the k-th most often appearing word is asymptotic to

$$f(k, s, n) \propto \frac{1}{k^s} \left(\sum_{n=1}^{N} \frac{1}{n^s} \right)^{-1}$$

  for some constant s. Representations are not going to be reliable:

```
>> model.most_similar('coffee')
[('shop', 0.8511), ('bag', 0.7919),
('mall', 0.7810), ('fire', 0.7808),
('pack', 0.7768), ('wife', 0.7729)]
```

- **Spelling:** Some rubrics are independent of spelling and grammar. (e.g., short answer)

Characters, Words, and everything in between

We can model language in a manner that a single token is a

1. **Character**: Here we do not suffer from sparsity, because there are a finite set of characters. These systems are also much more robust to mispelling. However, it is almost impossible to attribute meaning at a character level.

Characters, Words, and everything in between

We can model language in a manner that a single token is a

1. **Character**: Here we do not suffer from sparsity, because there are a finite set of characters. These systems are also much more robust to mispelling. However, it is almost impossible to attribute meaning at a character level.
2. **Word**: We do have the ability to attribute meaning, however, we suffer from both sparsity and issues with mispelling.

Cambium Assessment

Characters, Words, and everything in between

We can model language in a manner that a single token is a

1. **Character**: Here we do not suffer from sparsity, because there are a finite set of characters. These systems are also much more robust to mispelling. However, it is almost impossible to attribute meaning at a character level.
2. **Word**: We do have the ability to attribute meaning, however, we suffer from both sparsity and issues with mispelling.
3. **Subword**: We fix a size of vocabulary and try to cover a text with as few subwords as possible. This tokenization interpolates between words and characters.

Cambium Assessment

Tokens

For example, if we have the sentence

A red apple

We could tokenize in the following ways:

Tokens

For example, if we have the sentence

A red apple

We could tokenize in the following ways:

- Character-level models: When tokens are characters. [a, , r,e,d, ,a,p,p,l,e]

Cambium Assessment

Tokens

For example, if we have the sentence

A red apple

We could tokenize in the following ways:

- Character-level models: When tokens are characters.
  [a, , r,e,d, ,a,p,p,l,e]
- Word-level models: Based on standard tokenization into words.
  [a,red,apple]

Cambium Assessment

Tokens

For example, if we have the sentence

A red apple

We could tokenize in the following ways:

- Character-level models: When tokens are characters.
  [a, , r,e,d, ,a,p,p,l,e]
- Word-level models: Based on standard tokenization into words.
  [a,red,apple]
- Subword-level models: Breaking up into a finite number of pieces.
  [_a, _red, _app,le]

Tokenizers Library

Using the tokenizers library, it is easy and simple to define your own subword tokenizers:

```
from tokenizers import Tokenizer, models, normalizers, pre

tokenizer = Tokenizer(models.Unigram())
tokenizer.normalizer = normalizers.NFKC()
tokenizer.pre_tokenizer = pre_tokenizers.ByteLevel()
tokenizer.decoder = decoders.ByteLevel()

trainer = trainers.UnigramTrainer(
    vocab_size=20000,
    initial_alphabet=pre_tokenizers.ByteLevel.alphabet(),
    special_tokens=["<PAD>", "<BOS>", "<EOS>"],
)
```

Cambium Assessment

How many tokens

So how may tokens do we need:

- There are an estimated 175,000 English words currently in use.
- If we use too many, our embeddings are unreliable and subject to sparsity.
- If we use too few, we loose semantic information.
- The average English speaking adult knows about 20,000 to 42,000.
- Many words outside this set can be expressed in terms of a small number of characters.

Based on these assumptions, most language models using subwords have approximately 30,000 subword tokens.

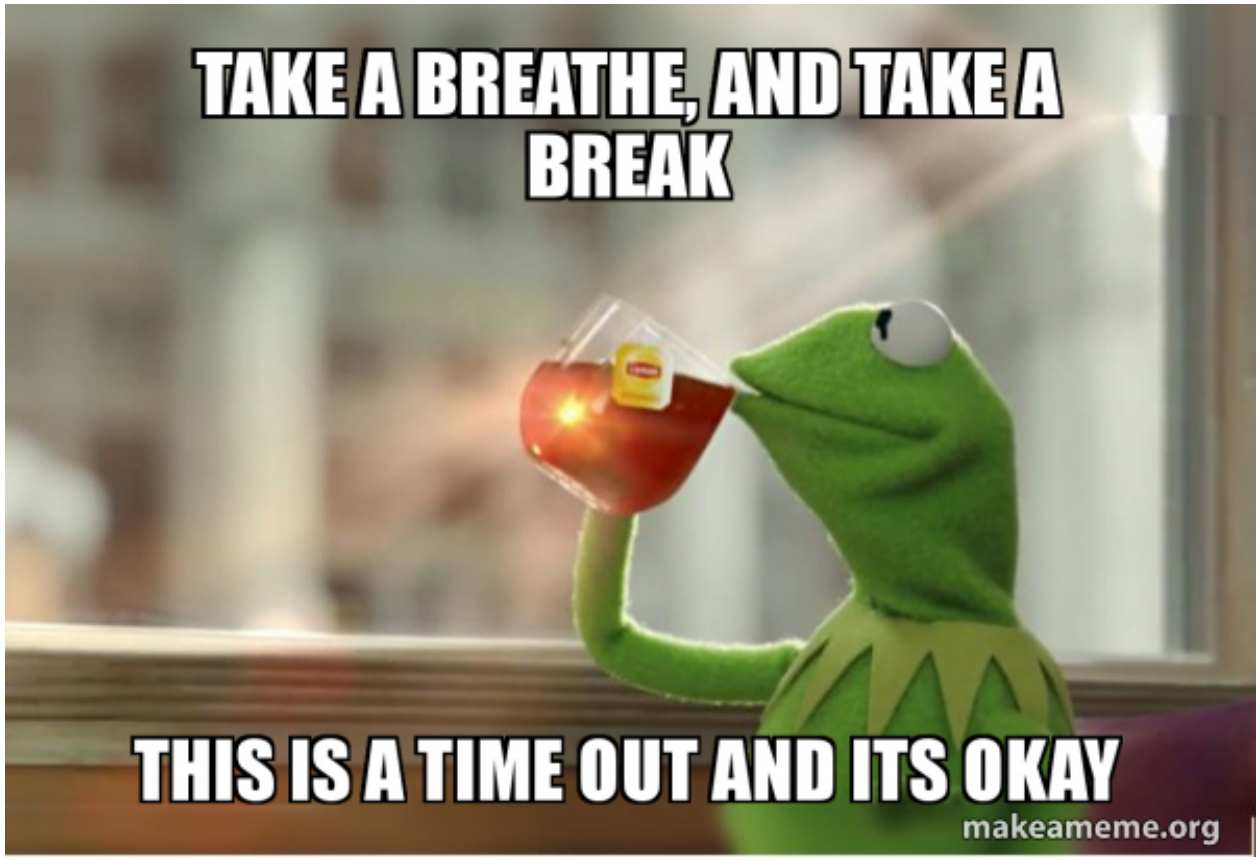
Cambium Assessment

Review Session 3

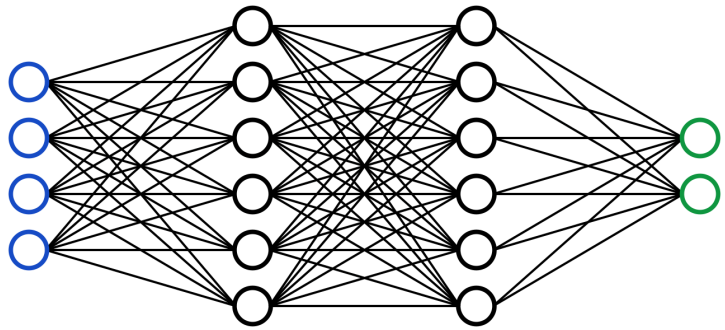
- We understood what we mean by a Language model
- We addressed the issue of sparsity in language modelling.
- We introduced the idea of word embeddings.
  - Continuous Bag-of-Words
  - Skipgram
  - Use the gensim library to define our own models or load pretrained ones.
- Considered subwords as an alternative to traditional characters and words as tokens.
- How subwords intepolate between characters and words.
- Use the tokenizers library to define our own tokenizers or load pretrained ones.

Cambium Assessment

Break


TAKE A BREATHE, AND TAKE A BREAK
THIS IS A TIME OUT AND ITS OKAY
makeameme.org

Pytorch and Neural Networks



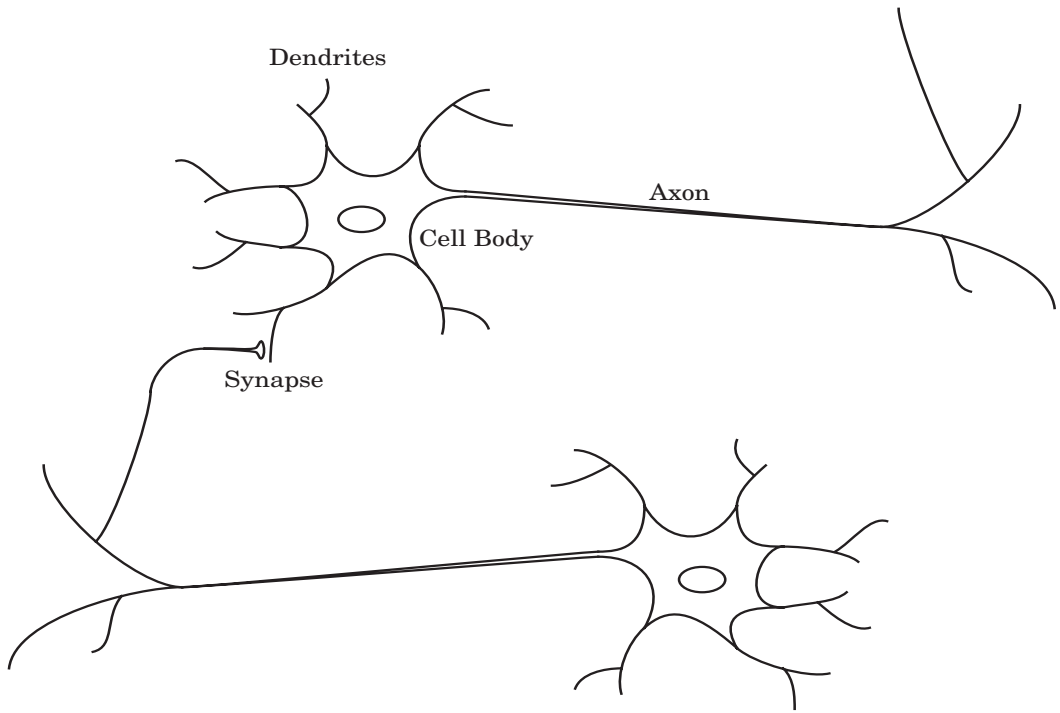
Aims

- Define a Neural Network.
- Introduce PyTorch as a tool to model with neural networks.
- Demonstrate the process of training language models with PyTorch.
- Discuss some of the potential problems in using neural networks.

Cambium Assessment

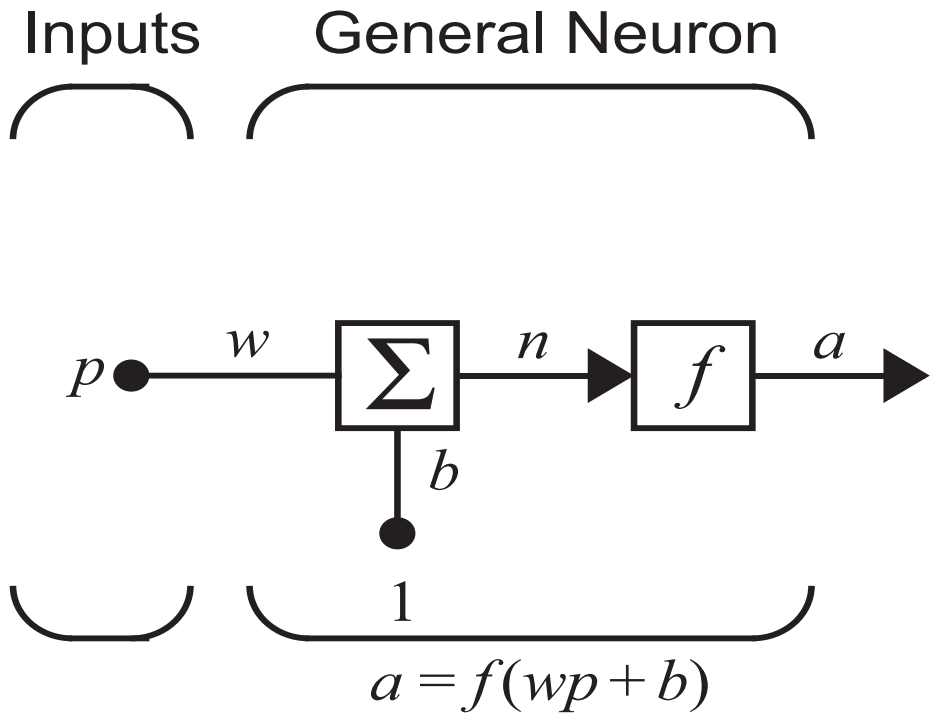
Brain Function

- Neurons Respond Slowly
  - 10^{-3} s compared to to 10^{-9} s for electrical circuits.
- The brain uses massively parallel comptation
  - $\approx 10^{11}$ neurons in the brain.
  - $\approx 10^{4}$ connections per neurons.



Single input model

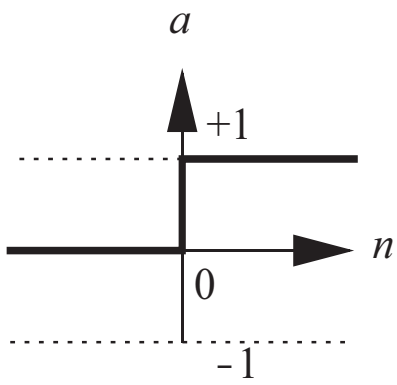
Given an input, we can apply a linear transformation, $y = wp + b$, and an activation function, $f(x)$:



This activation is either on or off. This is a computational model for a single neuron. The variables w and b are called a weight and bias.

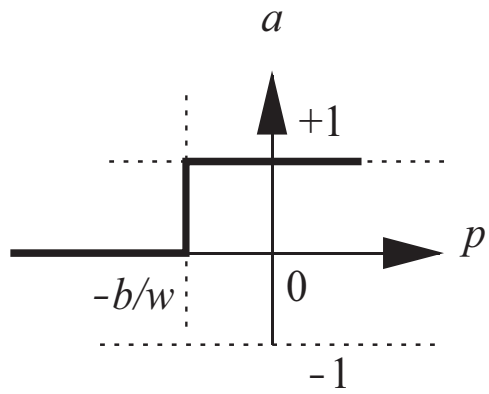
Transfer function

Different transfer functions give different behavior of the single neuron;



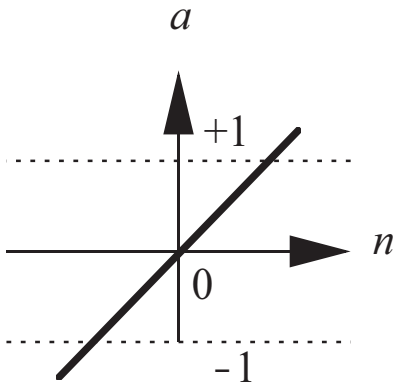
$a = hardlim(n)$

Hard Limit Transfer Function



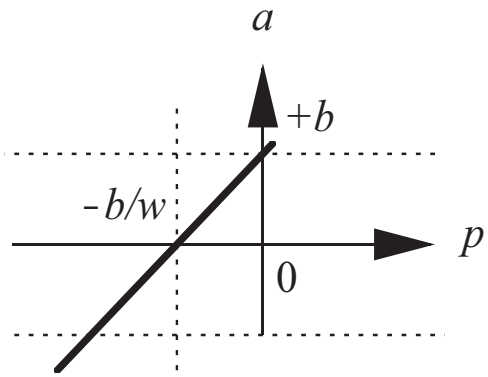
$a = hardlim(wp + b)$

Single-Input *hardlim* Neuron



$a = purelin(n)$

Linear Transfer Function

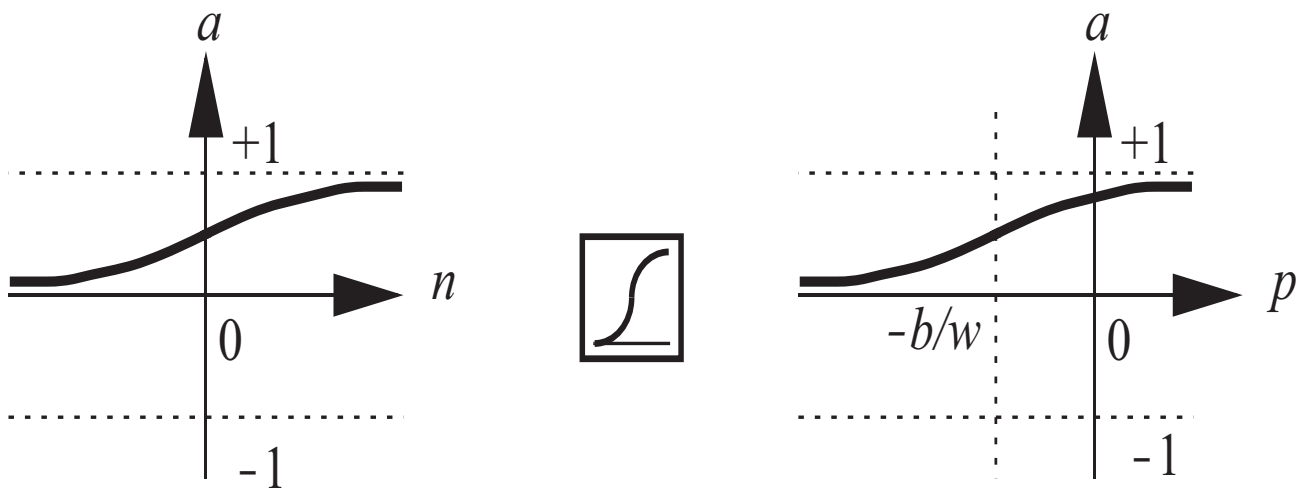


$a = purelin(wp + b)$

Single-Input *purelin* Neuron

Transfer function

Different transfer functions give different behavior of the single neuron;



$a = logsig(n)$

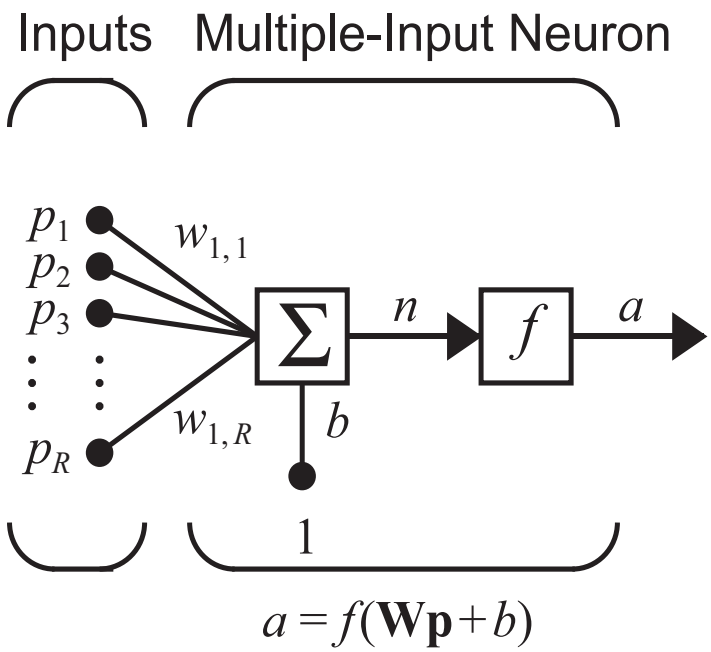
Log-Sigmoid Transfer Function

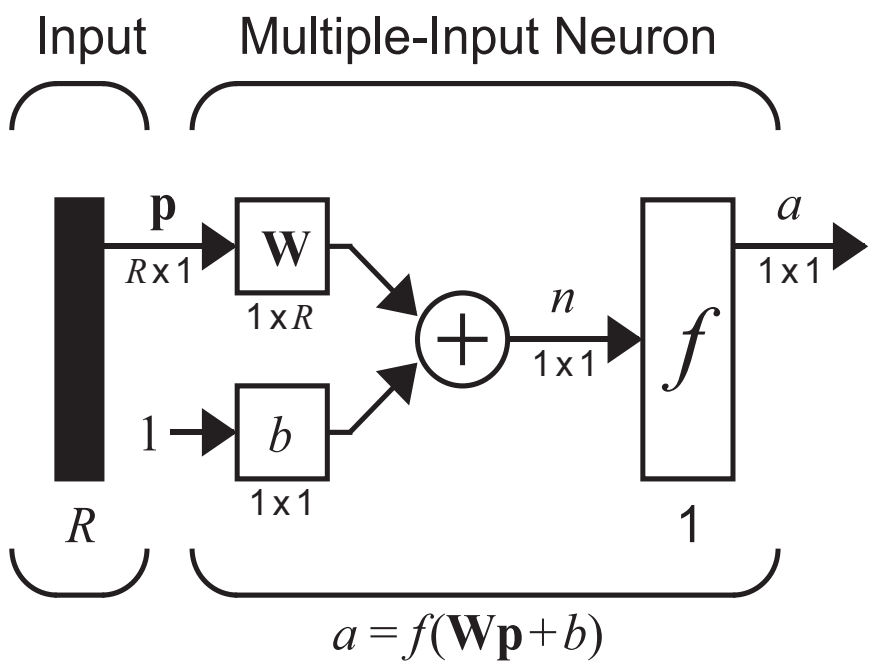
$a = logsig(wp + b)$

Single-Input *logsig* Neuron

Multiple input neuron

This generalizes to multiple inputs:

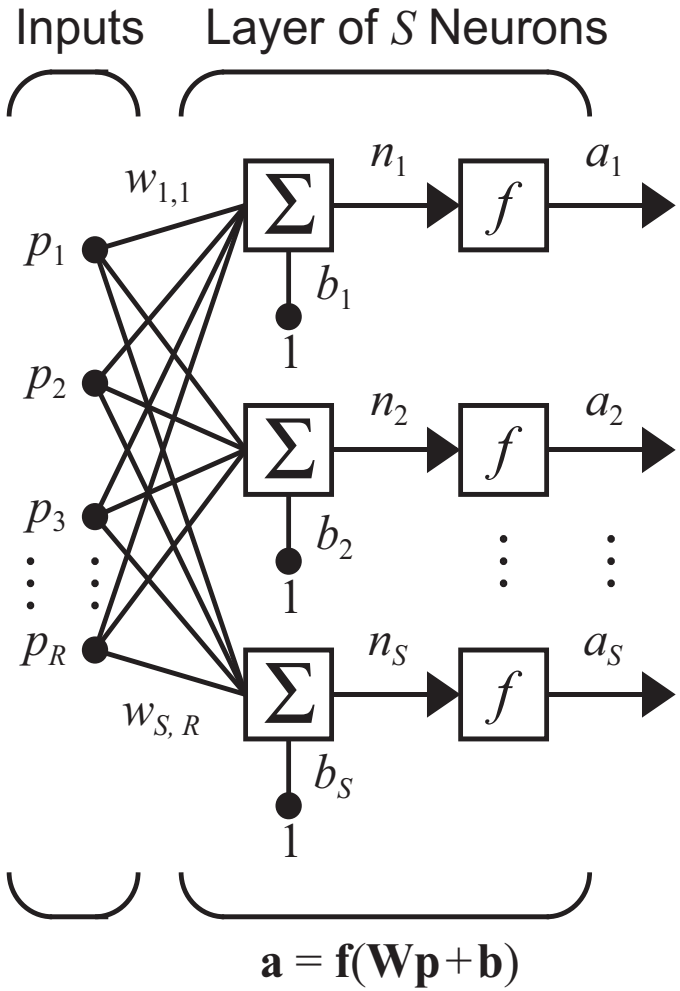




Abreviated Notation

Layer of neurons

This also generalizes to multiple outputs:

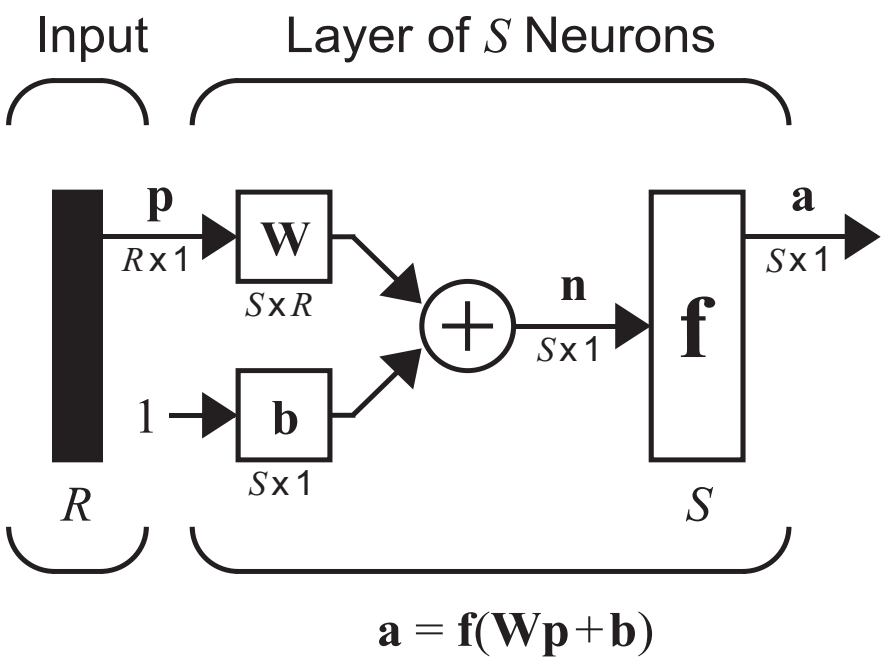


Abbreviated notation

When we abbreviate this notation, the fundamental relation is

$$a = f(Wp + b),$$

where $f(x)$ is an activation function,

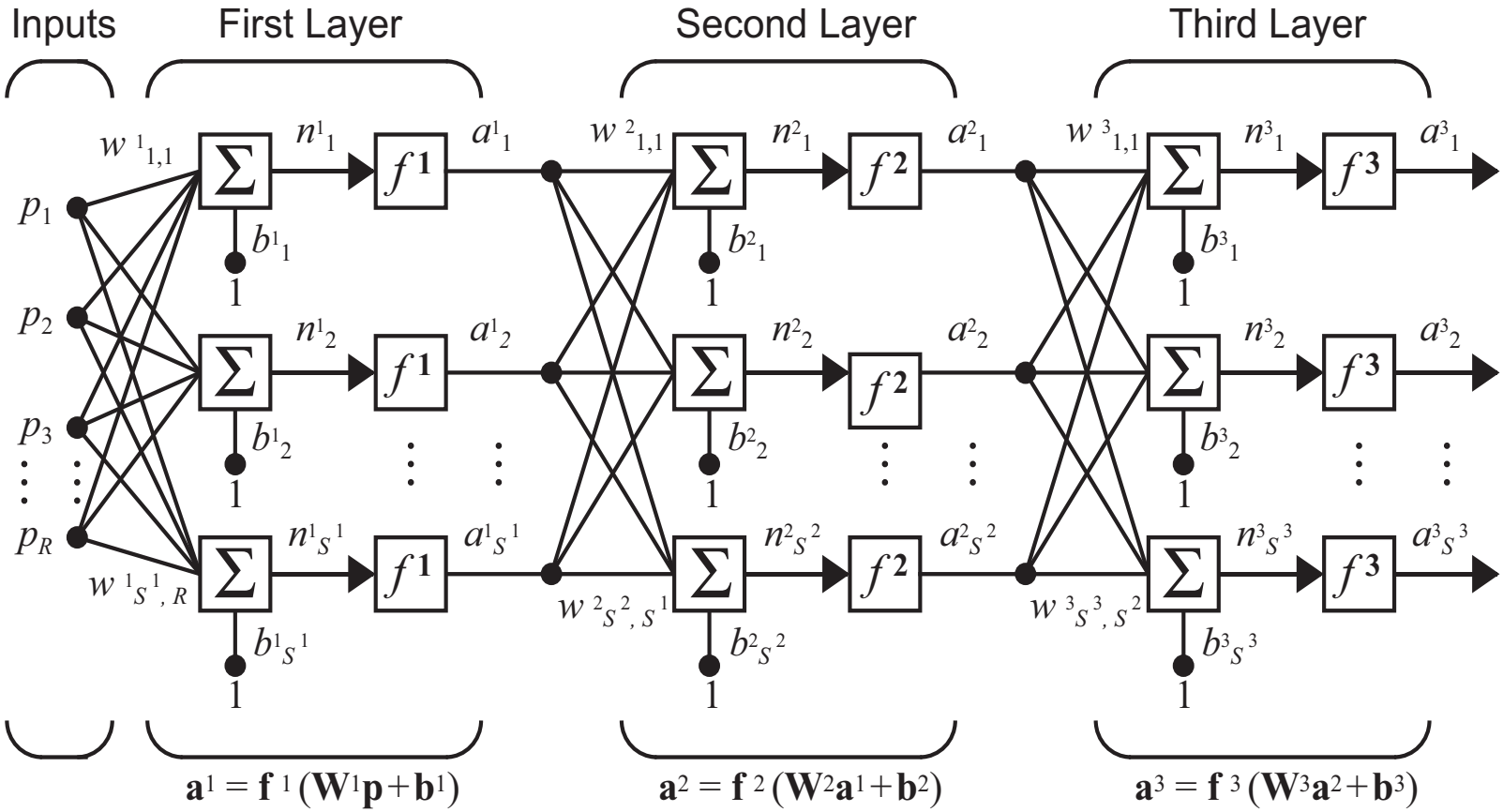


$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

Multilayer network

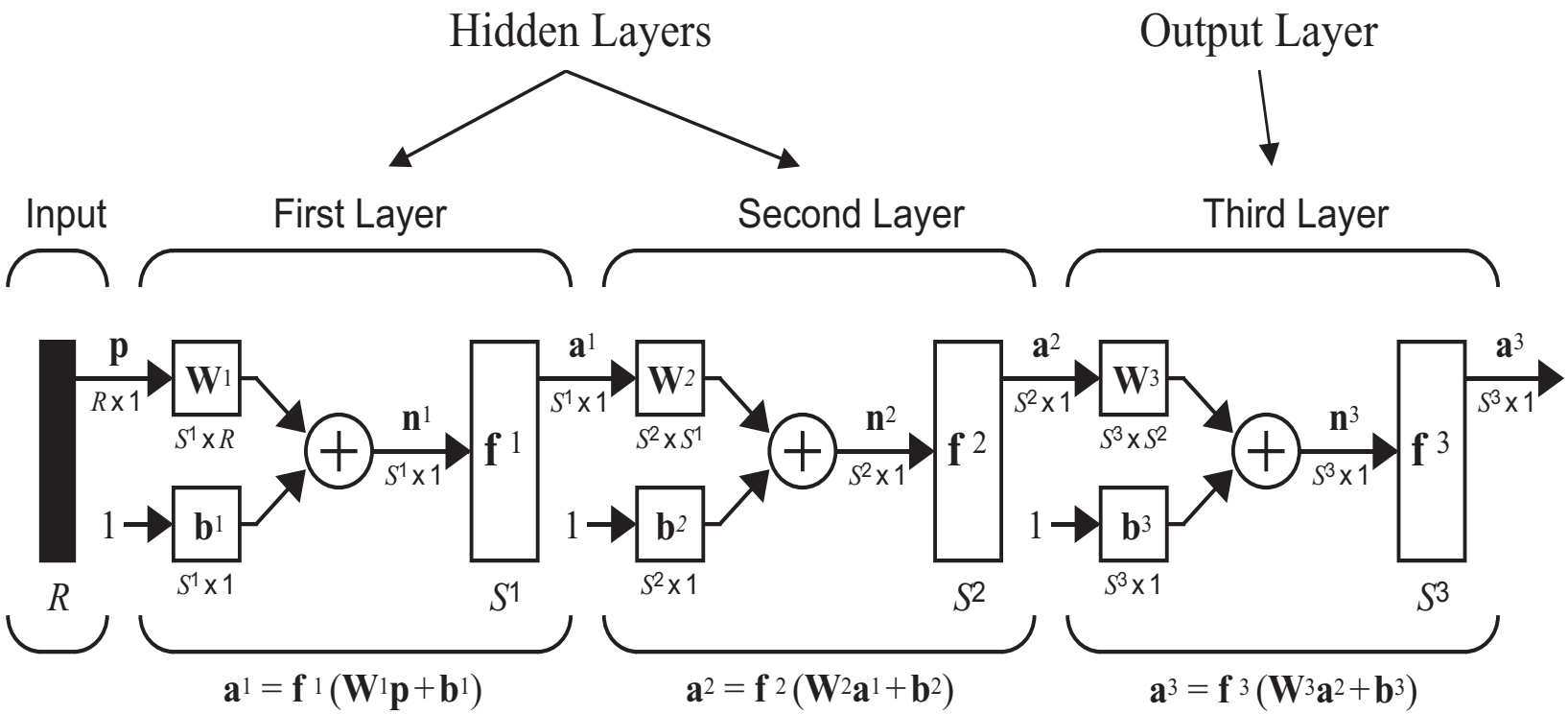
If we use the output of each layer as the input for another layer, we get a structure known as a Multilayer perceptron (MLP):



$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Abbreviated notation

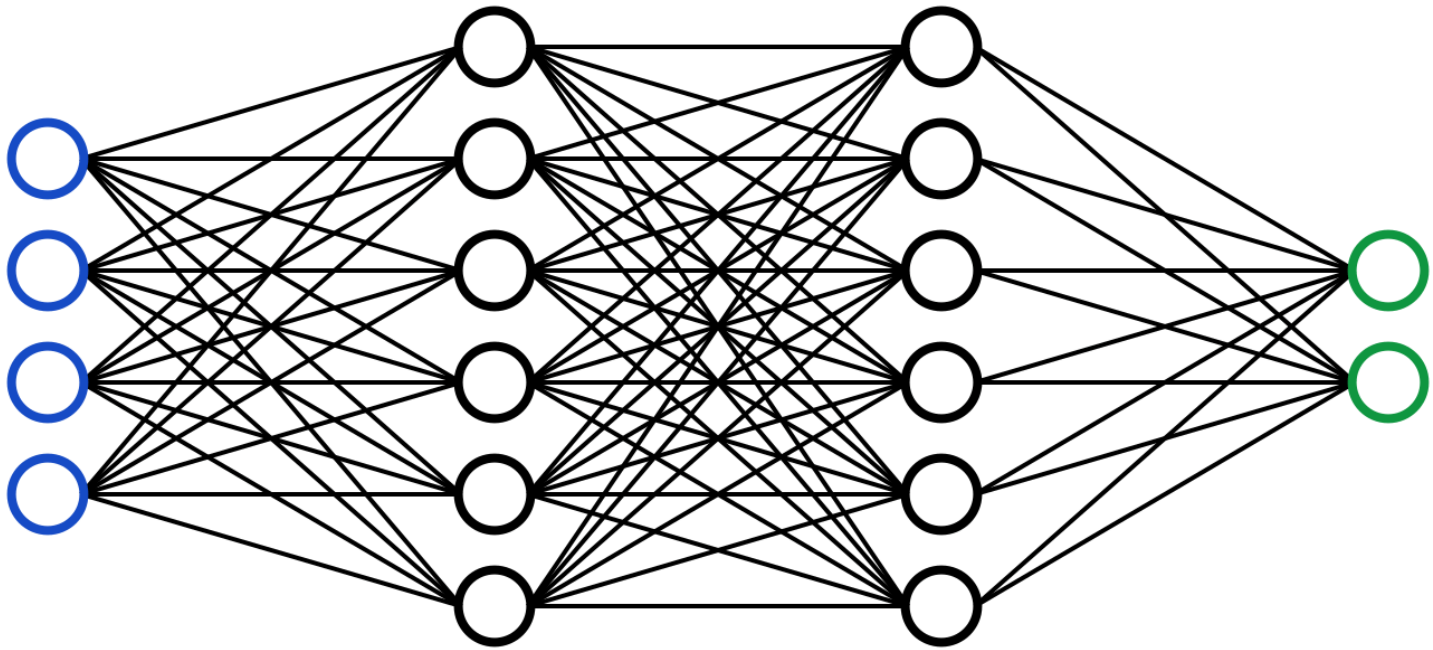
We call the layers between the input and last layer the "hidden layers":



Generally, each matrix element of each W and each b are called weight parameters. Deep neural networks can have millions of parameters.

Alternative representation

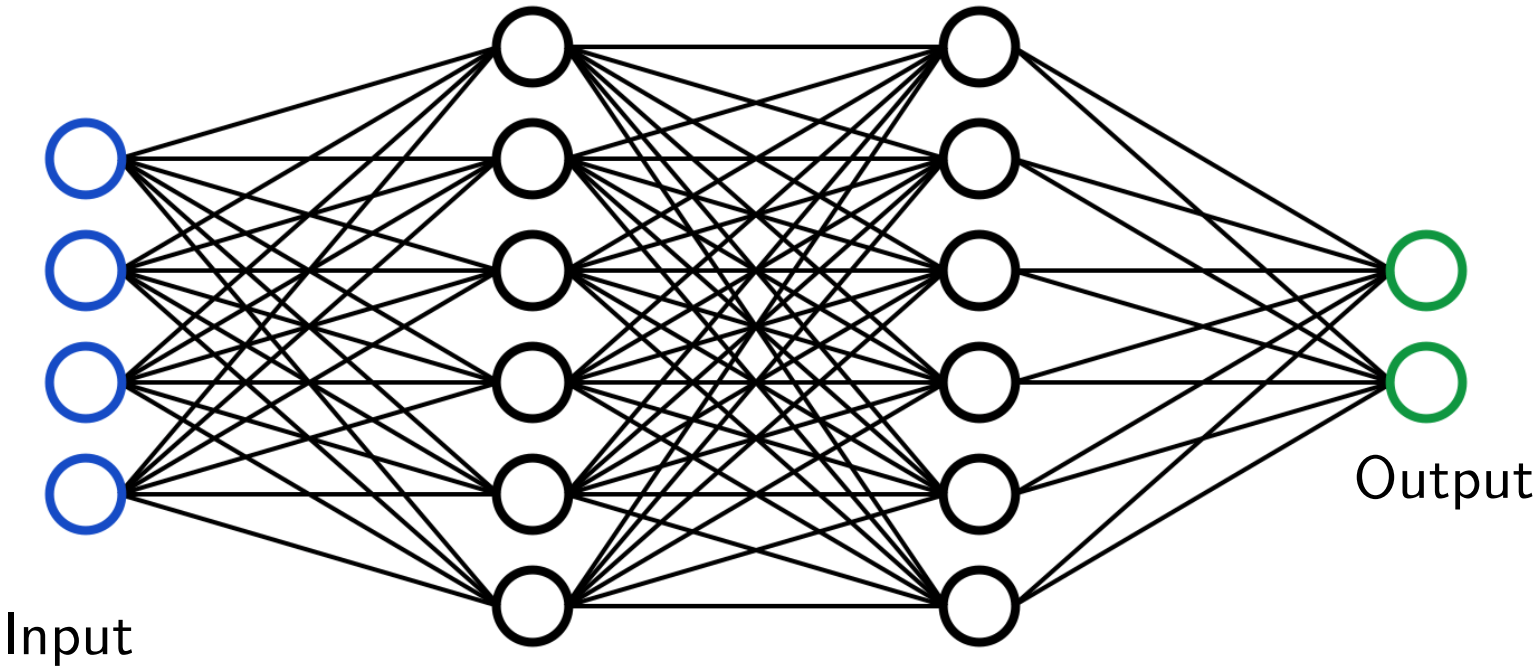
An alternative and common representation of this network is below.



Each line here represent a function of the form $f(wp + b)$.

Alternative representation

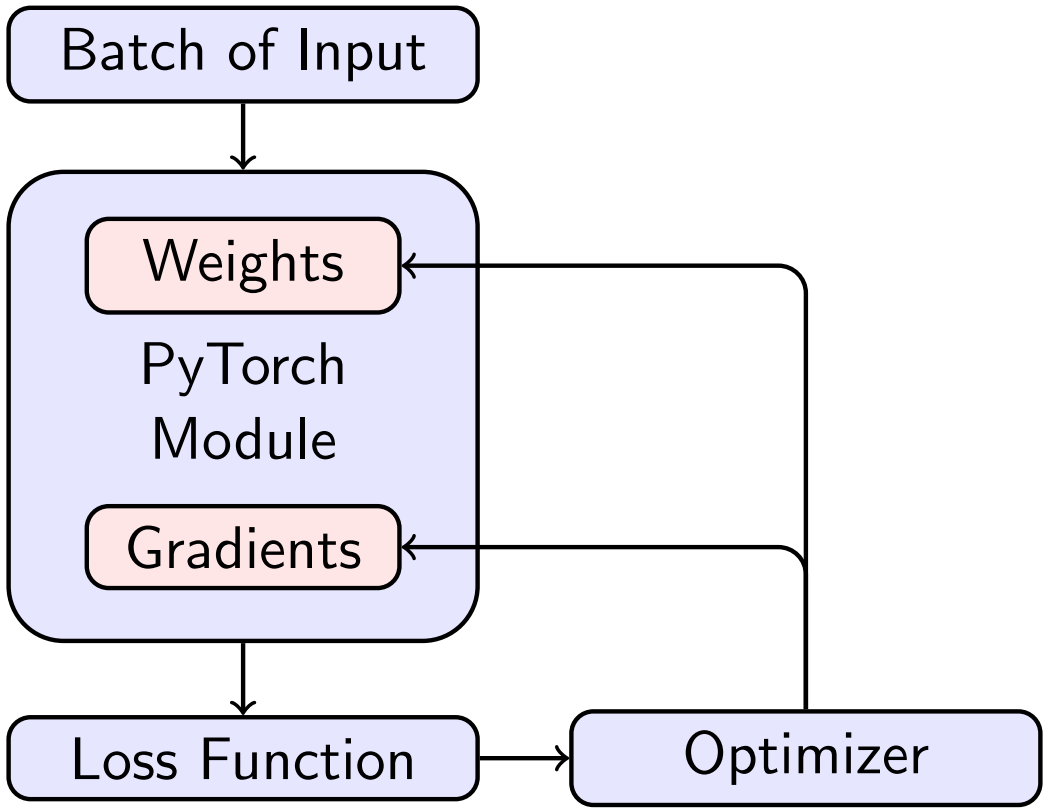
An alternative and common representation of this network is below.



Each line here represent a function of the form $f(wp + b)$.

Pytorch

Pytorch is a library for dealing with neural networks. Given a model, a single step requires input, a loss function, and an optimizer that updates the models weights:



Most optimizers use variants of the Stochastic Gradient Descent Algorithm. Cycling through training data once is called an epoch.

Pytorch

The operation $a = f(Wp + b)$ is given by the the nn.Linear object:

```
class MLP(nn.Module):
    def __init__(self, hidden_dim):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_size,
                                 hidden_dim)
        self.linear2 = nn.Linear(hidden_dim,
                                 target_size)
        self.act1 = torch.nn.Softmax(dim=1)
    def forward(self, x):
        out_em = self.linear1(x)
        output = self.linear2(out_em)
        output = self.act1(output)
        return output
```

The function "forward" determines the action of the network.

Training

In Pytorch, running a model consists of

- Initializing the model, the optimizer, and the loss function.

```
model = MLP(hidden)

# model.parameters() exposes the variables to
# the optimizer
optimizer = torch.optim.Adam(model.parameters(),
lr = 5e-5)

# We will be using the Mean Squared Error
# loss function
criterion = nn.MSELoss()
```

This sets up the necessary machinery to start training.

Cambium Assessment

One epoch

Once the model, optimizer, and loss function are set up, we can now cycle through the data. This involves:

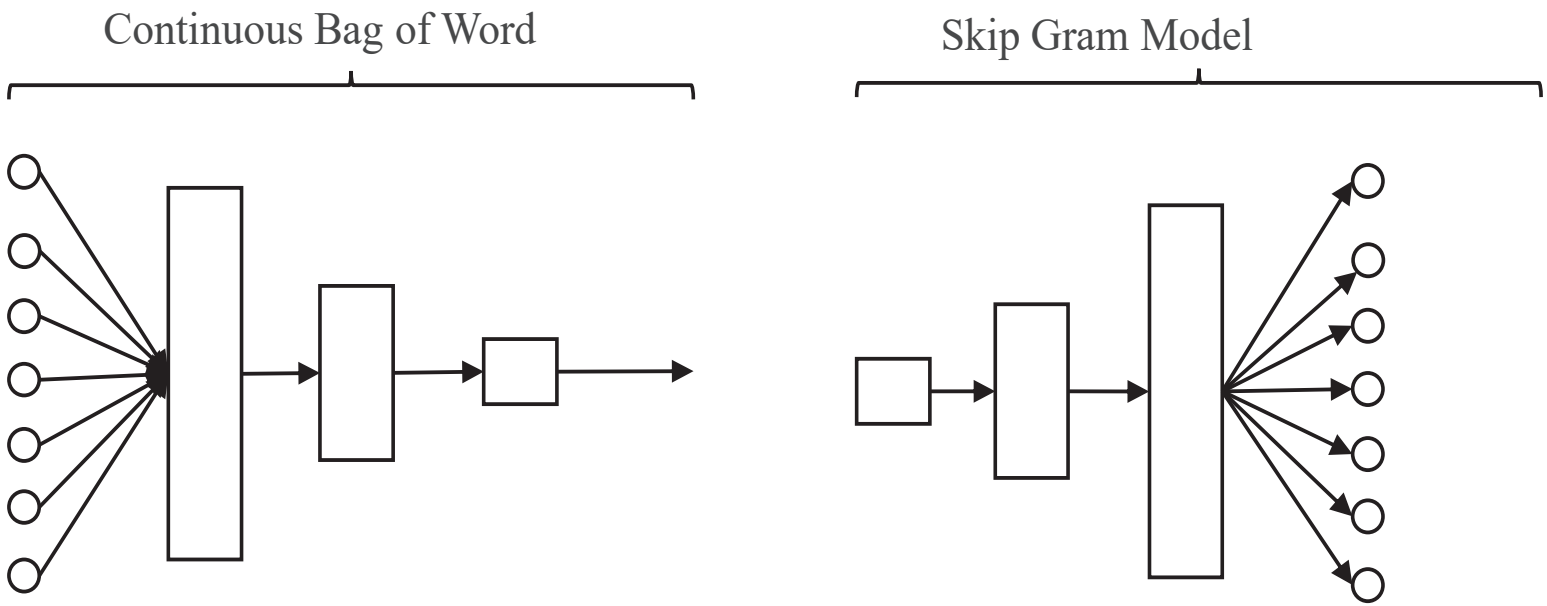
- Applying the model to the training data.
- Computing the loss function.
- Iterating the optimizer (which updates the weights).

```
optimizer.zero_grad()

# We assume the model input is in the p variable.
y_pred = model(p)

# We find the loss
loss = criterion(y, y_pred)

# We iterate the loss function
loss.backward()

# And update the weights using the optimizer.
optimizer.step()
```

Cambium Assessment

Example network

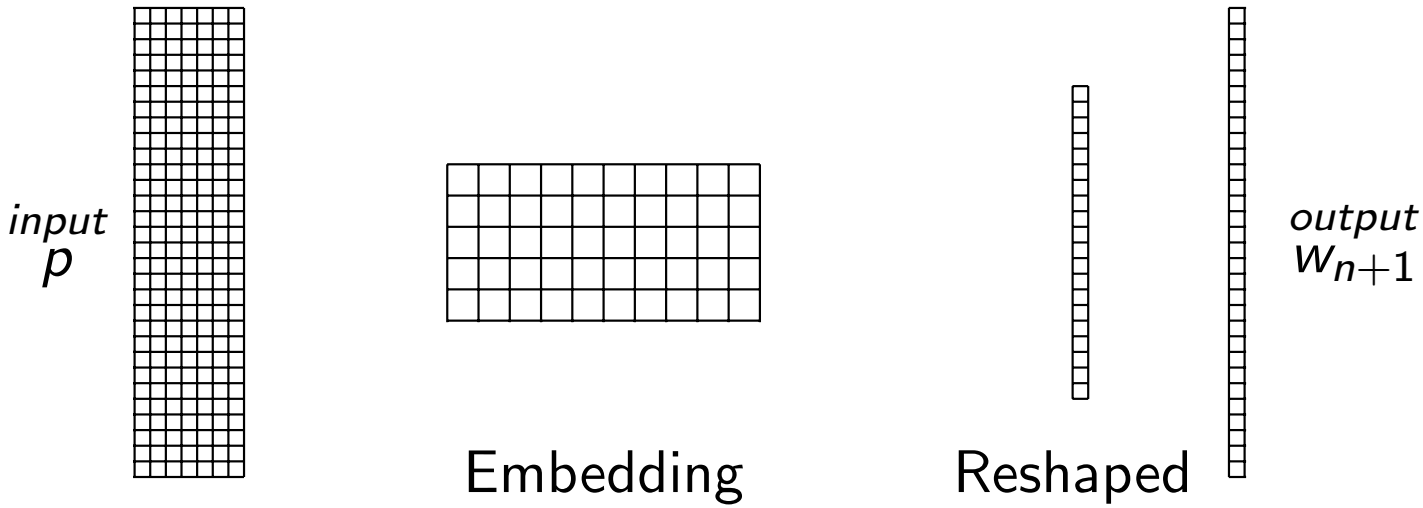
Let us review how this might work from a language modelling point of view: The word2vec architectures can be realized as neural networks.



We can use a MLP structure whose input and output is a one-hot-encoding of the vocabulary and hidden layer

Language model: Next word prediction.

Given a sequence of words, $p = [w_0 \dots w_n]$, we can train a neural network to predict w_{n+1}.

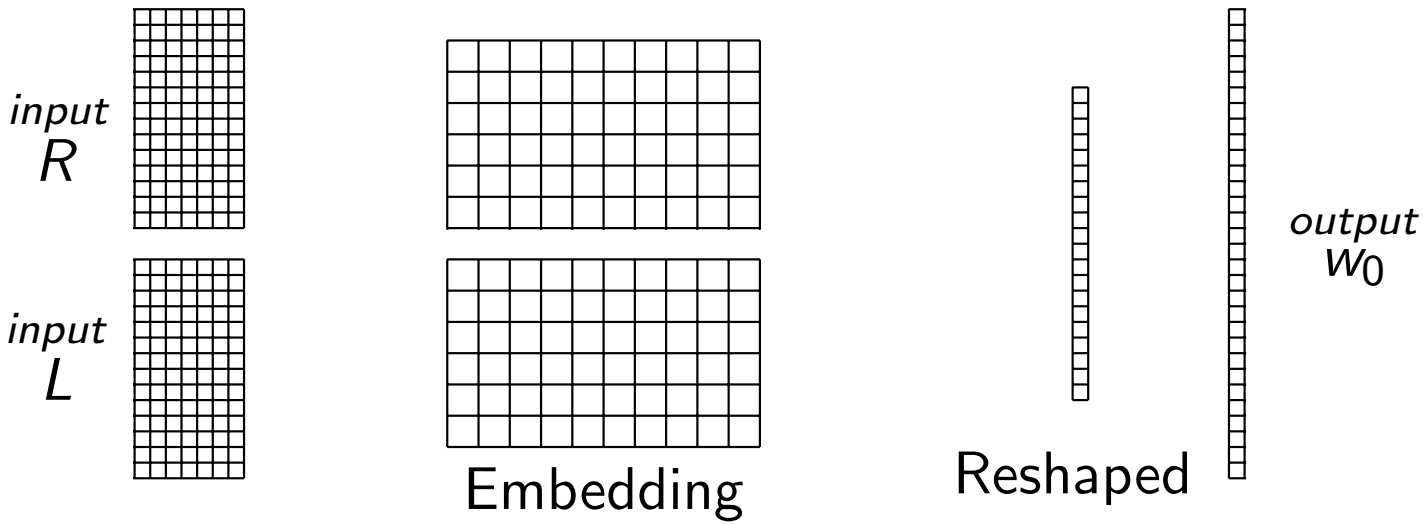


We can train this model to approximate the same information given by n-grams. e.g,

"This man works as a _____?"

Language model: Masked word prediction.

Given the words to the left and right, $L = [w_{-n} \dots w_{-1}]$ and $R = [w_1, \dots, w_n]$ we can train a neural network to predict w_0.



We can train this model to approximate the same information given by $(2n+1)$-grams. e.g,

"The man enjoyed a _____? of coffee."

Problems with these models

If we consider this a language model, we have the following problems:

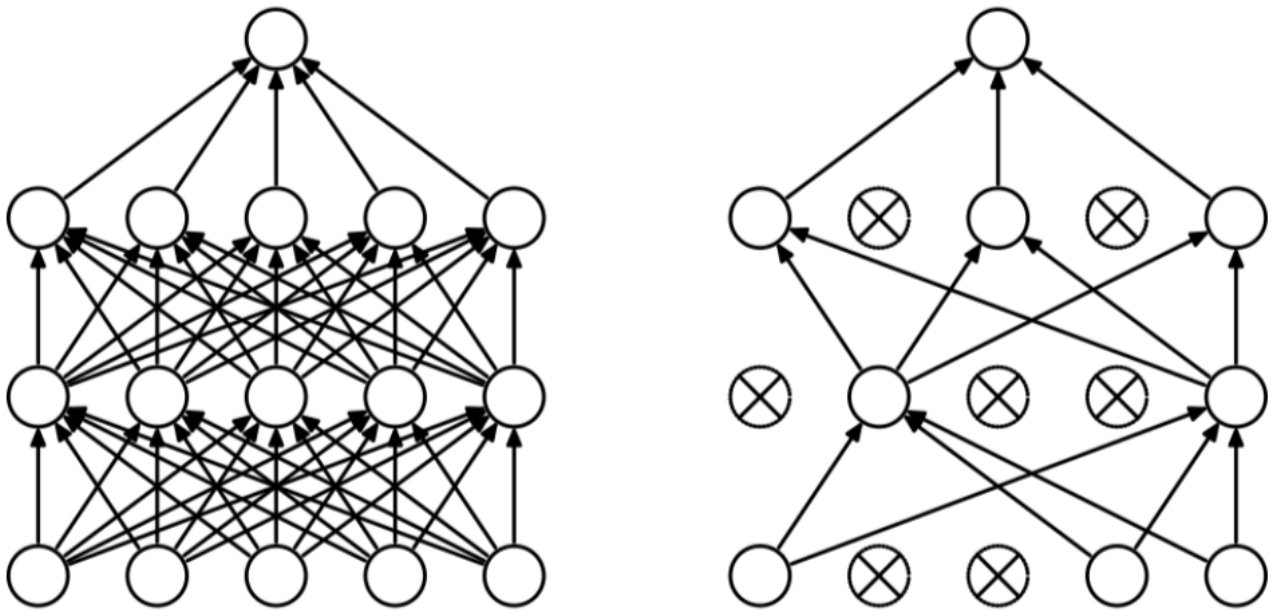
- Unlike n-grams, these models can be much larger and are subject to overfitting.
- The structures provided above only work on n-grams for a **fixed** n. I.e., these language models can't learn features beyond a fixed window.

These structures can be fast and applied to small devices like mobile phones, but they do not necessary give very high accuracy when storing n-grams is not feasible due to memory constraints.

Cambium Assessment

Dropout

Dropout is a mechanism used to regularize neural networks to prevent overfitting. The idea is that we generally neglect some inputs in the calculation:



(a) Standard Neural Net

(b) After applying dropout.

We can vary the dropout to get better results.

Difficulties

We have the following problems in moving to neural networks:

- The number of parameters required to define neural networks is much larger than other machine-learning methods. We can tend to overfit.
- The computational power required is a lot more than other methods. (GPU).
- The stochastic nature of the training makes it difficult to reproduce precisely.
- There are many more hyperparameters (learning rate, batch size, dropout) to define.
- Much more like a black box. Much less explainable.

Cambium Assessment

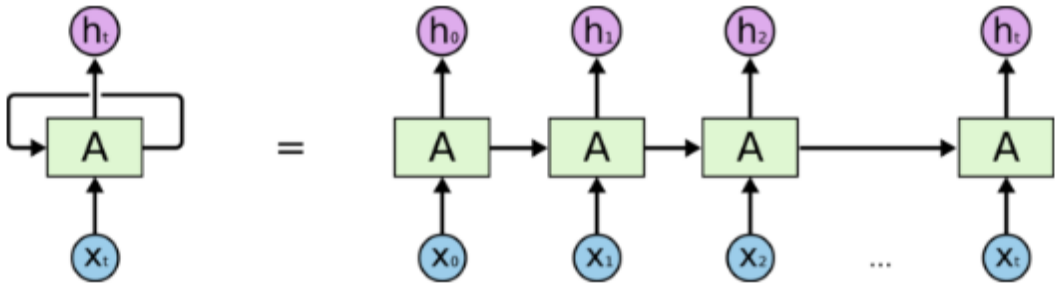
Review Session 4

1. We understood what we mean by a Neural Network:
   - We define in terms of collections of neurons.
   - Equivalent to linear transformations with nonlinear transfer/activation functions.
2. We used Pytorch to model a simple Multilayer perceptron:
   - These are given by multiple linear layers with activations.
3. We demonstrated the process of training a neural network to determine a word embedding.
   - Uses Stochastic Gradient Descent (or adaptive versions)
   - Iterates through the data in batches.
4. We discussed the pitfalls of neural networks
   - Many more parameters, much more computational power required to use, and subject to overfitting.
   - More difficult to train, often requiring expert knowledge.
   - Much more difficult to analyze.

Cambium Assessment

Break

Recurrent Neural Network



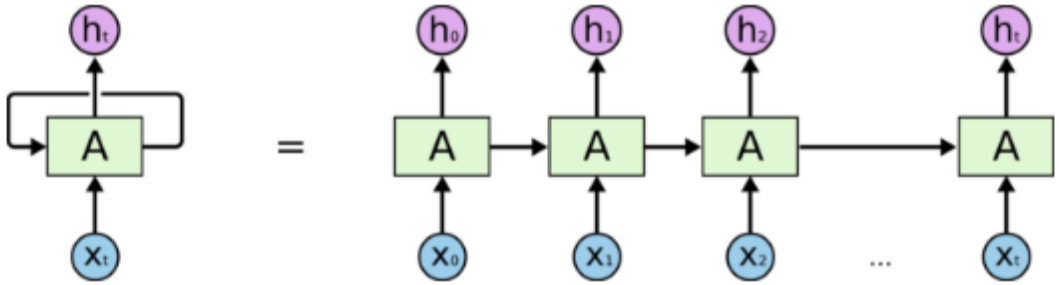
An unrolled recurrent neural network.

Aims

1. To understand feedback and recurrent nerual networks.
2. To discuss sequence-to-sequence models, encoder-decoder architectures.
3. Introduce attention, self attention and transformers.

Cambium Assessment

Recurrent Neural Networks



An unrolled recurrent neural network.

Consider the follow two tasks:

1. Predict the next word in a text.
2. Classify a text.

These problems have the following in common:

- Any accurate prediction should depend on each word.
- Each word is understood in terms of of previous words.

A recurrent neural network addresses this problem by allowing information to persist as it traverses a text.

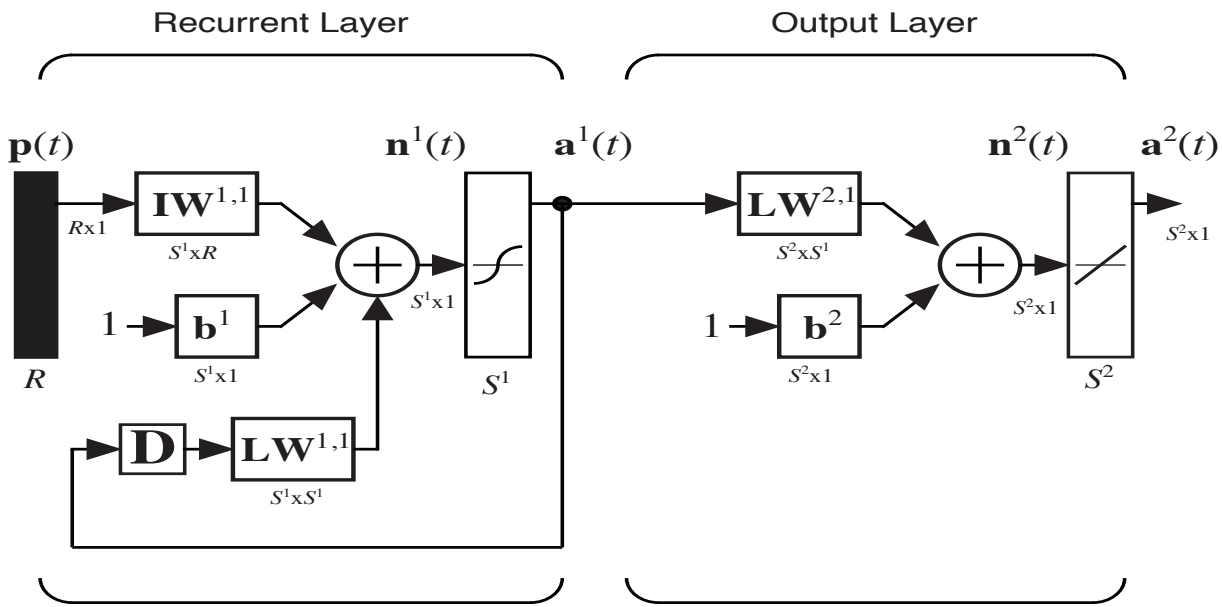
Cambium Assessment

Recurrent Networks

A recurrent unit operates on a sequence $p(t) = (p_0, \ldots, p_n)$. The input of any recurrent unit is

- An element of the sequence, $p(t)$.
- A function of the previous state/output.

This allows an RNN to have memory:

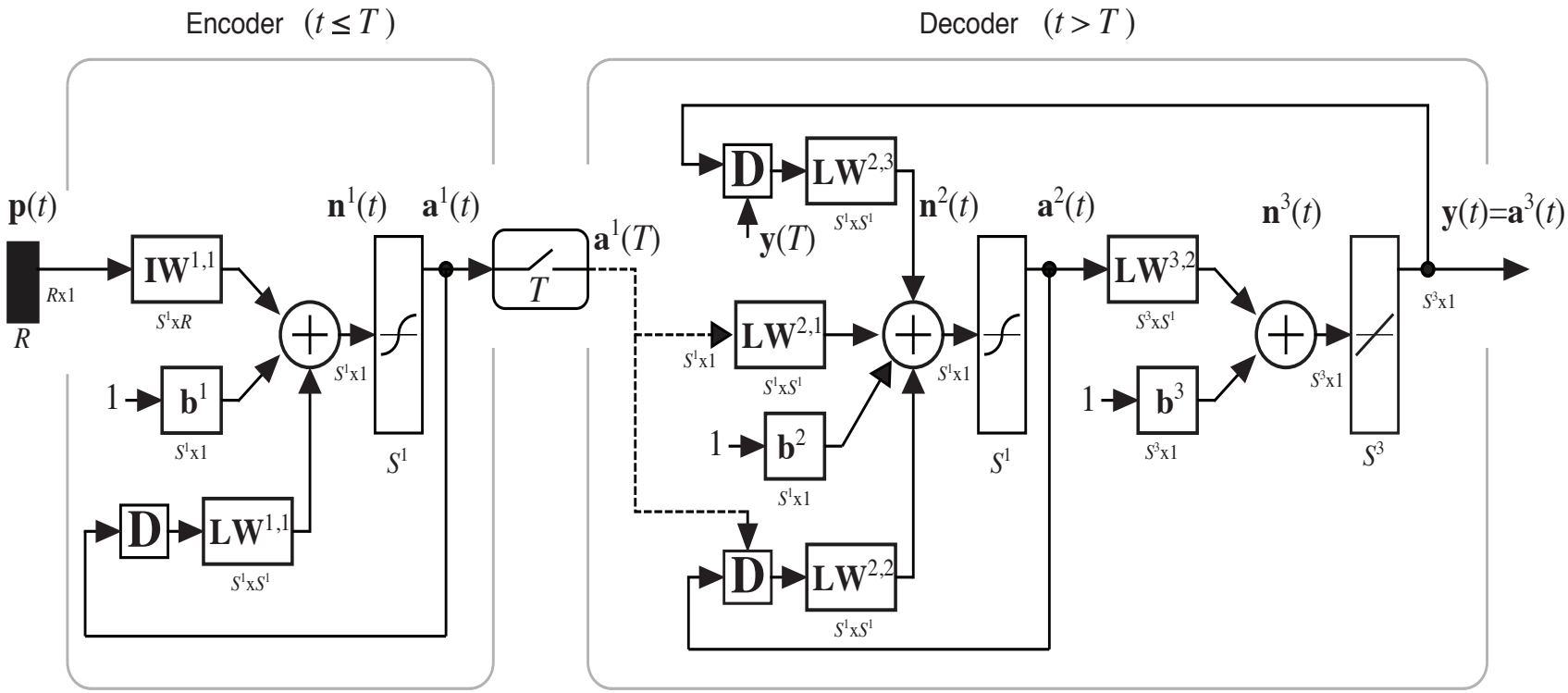


$$\mathbf{a}^1(t) = \mathbf{tansig}\left(\mathbf{IW}^{1,1}\mathbf{p}(t) + \mathbf{LW}^{1,1}\mathbf{a}^1(t-1) + \mathbf{b}^1\right)$$

$$\mathbf{a}^2(t) = \mathbf{LW}^{2,1}\mathbf{a}^1(t) + \mathbf{b}^2$$

Seq2seq Model

The input into the model is input for the encoder:



The output of the encoder is used as input for the decoder.

Encoder

- The encoder takes the input sequence, which has T time points (words, or word fragments), and produces a layer output $a^1(t)$.
- This output is sampled at the final time point $t = T$.
- The dotted line at the output of the sampler indicates that this value is fixed, and does not change with time.
- For example, the $u(t)$ input sequence might represent English words and the output sequence $y(t)$ might represent French words.

$$a^1(t) = tansig(IW^{1,1}p(t) + LW^{1,1}a^1(t-1) + b^1)$$

Cambium Assessment

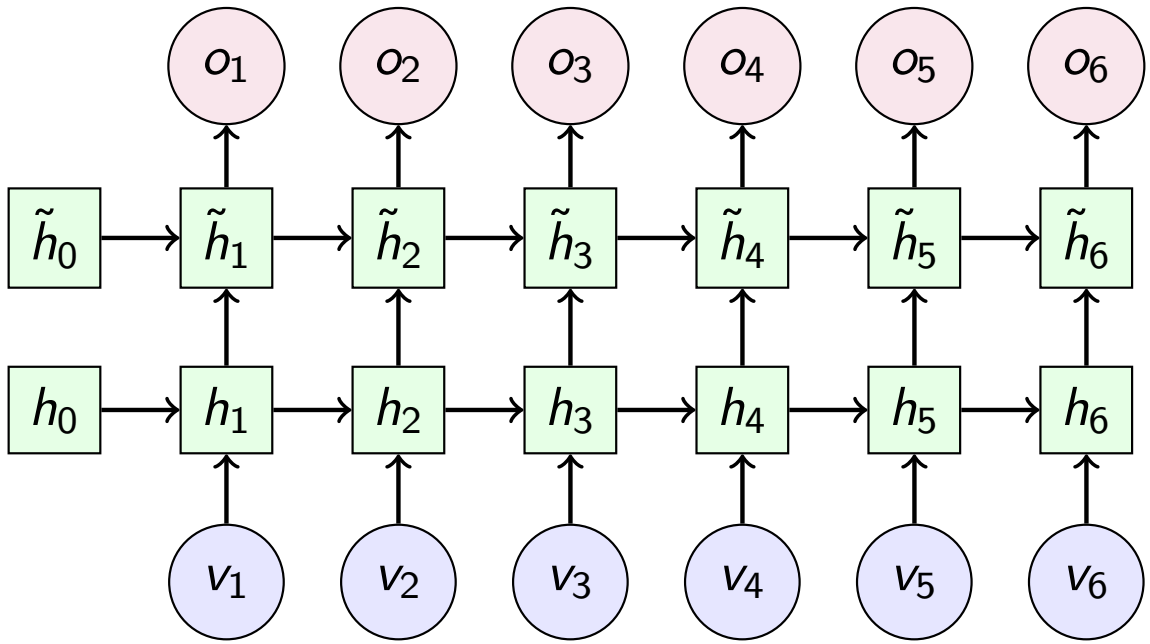
Decoder

- The decoder initializes its state with the final state of the encoder.
- The final state of the encoder also becomes a constant input to the first layer of the decoder.
- Hint: (There are many different formulations of the seq2seq model. In some cases the final encoder state is only used to initialize the state of the decoder.)
- This final encoder state is sometimes referred to as the **context**.

$$a^2(t) = tansig(LW^{2,3}a^3(t-1) + LW^{2,1}a^1(T) + LW^{2,2}a^2(t-1) + b^1)$$
$$a^3(t) = LW^{3,2}a^1 + b^3$$

Stacking

We can also use the output of a recurrent neural network as input into another recurrent neural network. This is called stacking:



The output is a sequence of vectors.

Two different applications

The output of an RNN can be used in two different ways: A single linear layer may be used to

- send the very last vector output of the RNN to a score.
- send each vector output of the RNN to the one-hot-encoding of the next token to form a Language model.

In the second application, the linear layer is playing the role of a decoder.

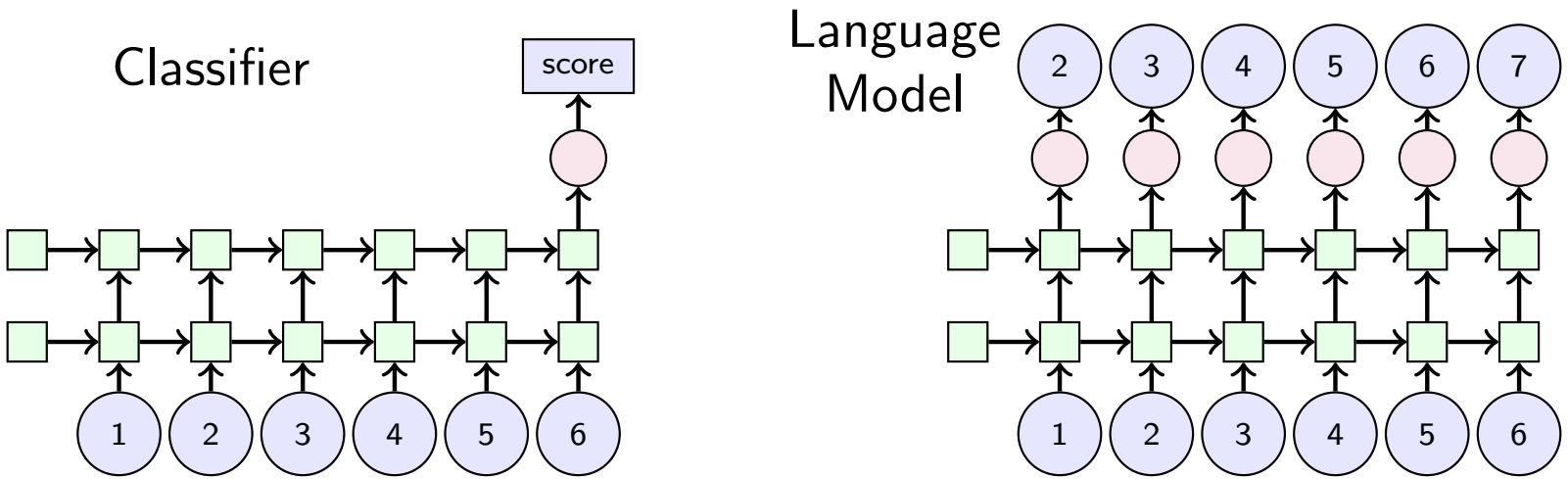


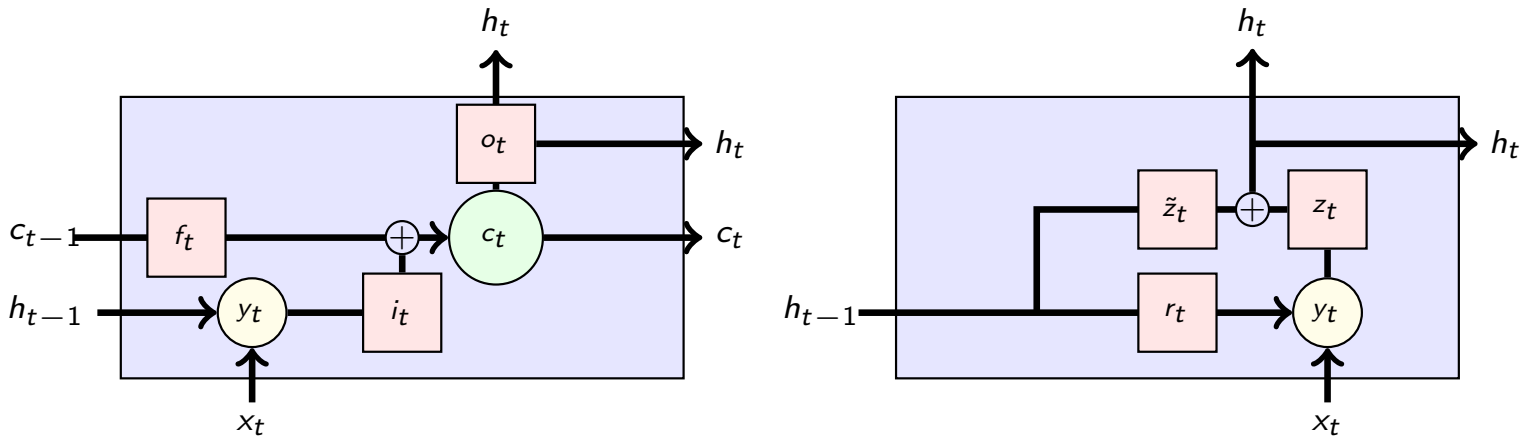
Figure: The structure of RNN-based classifiers and language models.

LSTM and GRU

In the literature, there are two dominant recurrent architectures:

- Long-short-term-memory (LSTM) unit networks
- Gated Recurrent Units (GRU) networks

The structure of the two units are as follows:



Given a sequence $x(t) = [x_0, \ldots, x_n]$, the output is a sequence $h(t) = [h_0, \ldots, h_n]$.

Colab

Pytorch allows us to model this effectively

```
class LSTMClassifier(nn.Module):
  def __init__(self):
    super(LSTMClassifier, self).__init__()
    self.embedding = nn.Embedding(num_embeddings=30000,
                                  embedding_dim=64)
    self.rnn = nn.LSTM(input_size = 64,
                       num_layers = 2,
                       hidden_size = 128,
                       batch_first = True)
    self.classifier = nn.Linear(in_features = 128,
                                out_features = 2)
    self.activation = nn.Softmax(1)
```

The components of the classifier are the embedding, recurrent layers, and a linear layer to perform the classification.

Forward function

The forward function first computes the embedding of the inputs, feeds those embeddings through the recurrent layer.

```
def forward(self, model_input):
  emb = self.embedding(model_input)
  lstm_out, _ = self.rnn(emb)
  features = lstm_out[:,-1,:]
  output = self.classifier(features)
  return self.activation(output)
```

The final hidden state is used to form the set of features used to form a classification.

Cambium Assessment

Testing the IMDB data

We simplify our workflow by using the datasets map library

```
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokenize = lambda x:tokenizer.encode_plus(x['text'],
return_tensors = "pt", padding='do_not_pad')

train_data = load_dataset("imdb", split="train")
test_data = load_dataset("imdb", split="test")
token_train_data = train_data.map(tokenize).shuffle()
token_test_data = test_data.map(tokenize).shuffl//e()
```

This gives us the input required for the LSTM.

Now we instantiate the model

```
model = LSTMClassifier()
optimizer = torch.optim.Adam(model.parameters())
critereon = nn.MSELoss()
```

Cambium Assessment

In code

Then we step through the training process:

```
targets = [torch.tensor([[1.0,0.0]]),
torch.tensor([[0.0,1.0]])]

for x in token_train_data:
  optimizer.zero_grad()
  y_pred = model(torch.tensor(x['input_ids']))
  y = targets[x['label']]
  loss = critereon(y_pred, y)
  loss.backward()
  optimizer.step()
```

The cycling through the data is a single epoch.

Cambium Assessment

Language model

The differences in code between the classifier and Language model are very minor. The definition of the module changes the linear layer to

```
self.decoder = nn.Linear(in_features = 128,
                         out_features = 30000)
```

And the forward function

```
lstm_out, _ = self.rnn(emb)
output = self.decoder(lstm_out)
```

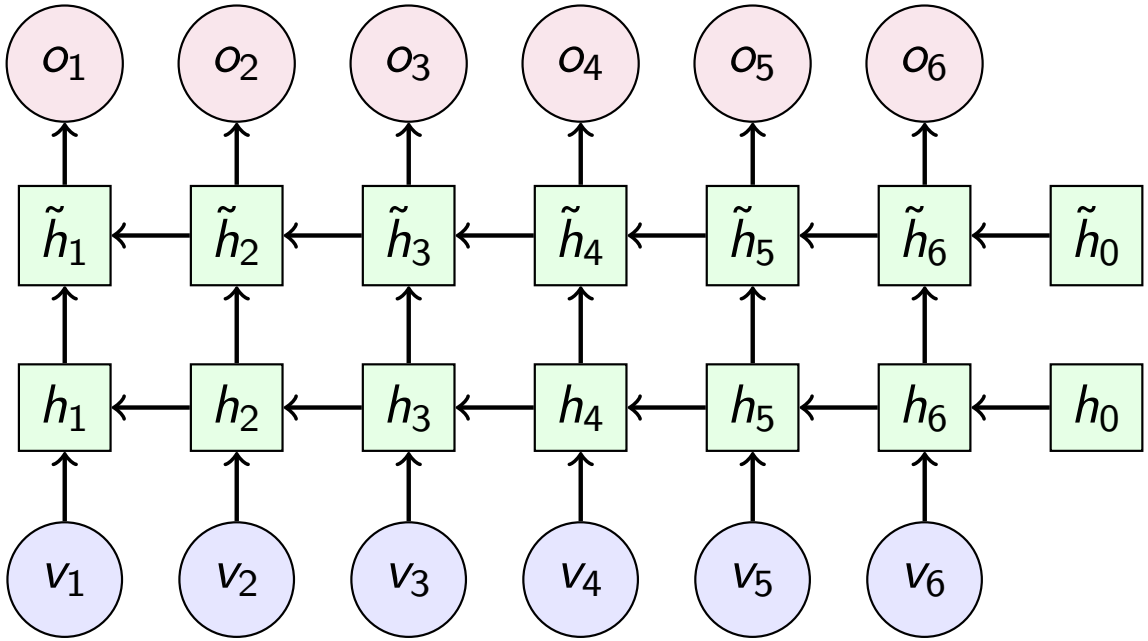
We just need to change the input to the text, and the output to the shifted text.

See the colab for more details.

Cambium Assessment

Bidirectional

We can also reverse the ordering



In this way o_1 depends on $v_1, \ldots, v_6$. This means $o_1 \in \mathbb{R}^d$ is a fixed dimensional vector.

A bidirectional recurrent network concatenates the outputs of recurrent units that go both ways. This is a more appropriate structure for translation tasks and masked language models.

Attention

- The recurrent layer of the encoder is able to store information from the beginning of the input sequence, but it may not be able to store it in the most efficient way.
- To provide more flexibility, we can add a tapped delay line of previous values of the encoder state, rather than using just the final value. This process is called attention.

Cambium Assessment

Attention

- The first step is to use a TDL_hstack to form a horizontal concatenation of previous encoder states, as in

$$A^1(t) = \begin{bmatrix} a11(t) & a^1(t-1) & ..... & a1(t-D+1) \end{bmatrix}$$

- We want to combine these vectors together in some way to form the context vector to send to the decoder (in place of just the final encoder state).
- The question is how much should each previous encoder state contribute to the context.
- This could be done by finding the inner product between the previous decoder state and all previous encoder states, as in the following equation

$$n^4(t) = \left[A^1(T)\right]^T a^2(t-1)$$

Cambium Assessment

Attention

- We should then normalize these correlations by using the softmax activation function, as in

$$a^4(t) = softmax(n^4(t))$$

- Finally, we combine the previous encoder state vectors according to the relative amount of correlation with the current decoder state.

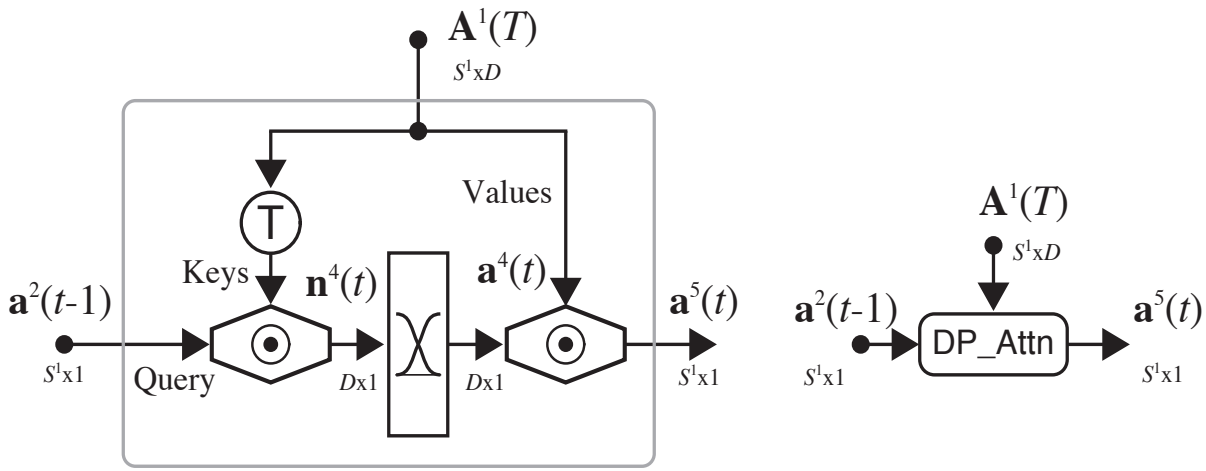
$$a^5(t) = A^1(T)a^4(t)$$

- The resulting $a^5(t)$ is the context vector that will be passed to the decoder.

Cambium Assessment

Attention Model

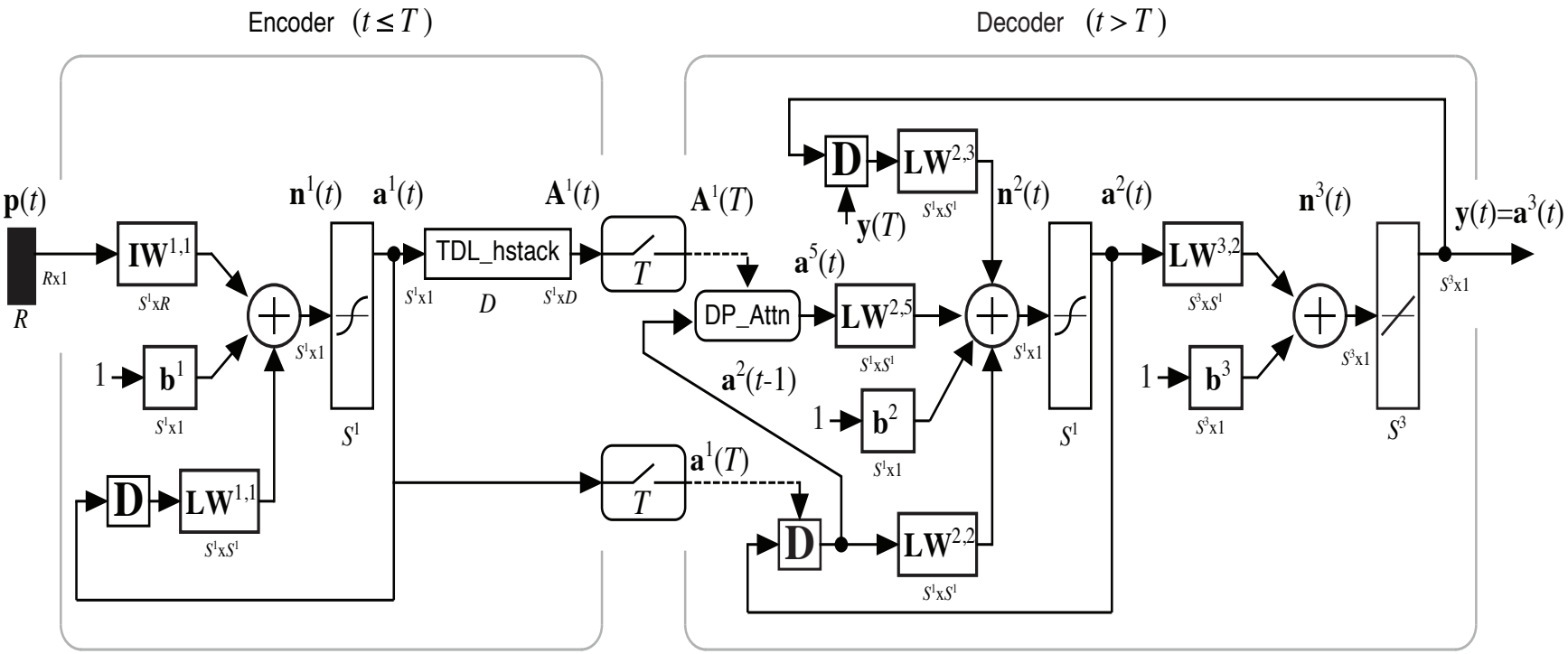
- The idea of attention is sometimes generalized as operations between a query vector and sets of key and value pairs.
- The query is compared to each of the keys to determine how much of each corresponding value is included in the result.



- In our case, the query is the previous decoder state $a^2(t-1)$, the keys are all previous encoder states, and the values are also the previous encoder states.

seq2seq with Attention Model

The attention model



Multi-headed attention

A multi-headed attention layer expresses the input space as the union of multiple smaller vector spaces and computes attention on each space.

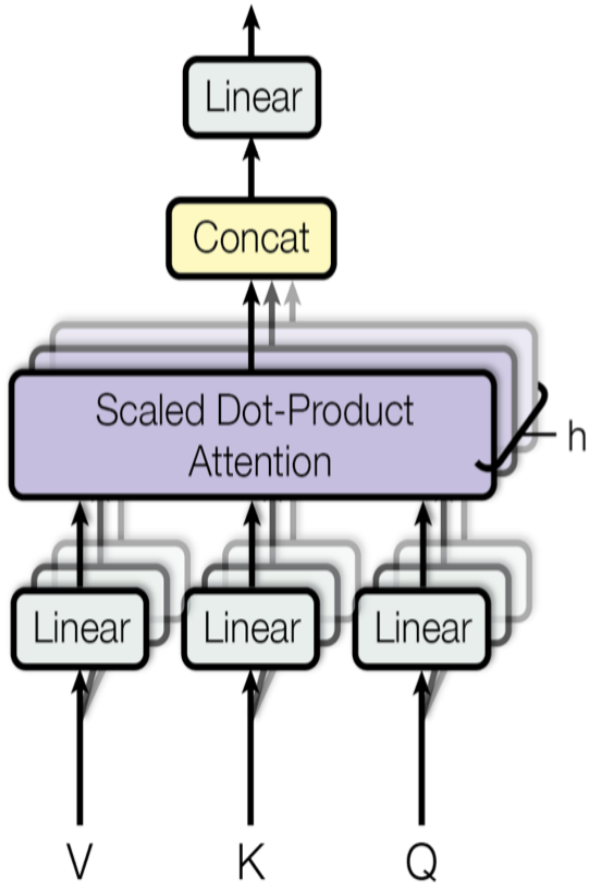
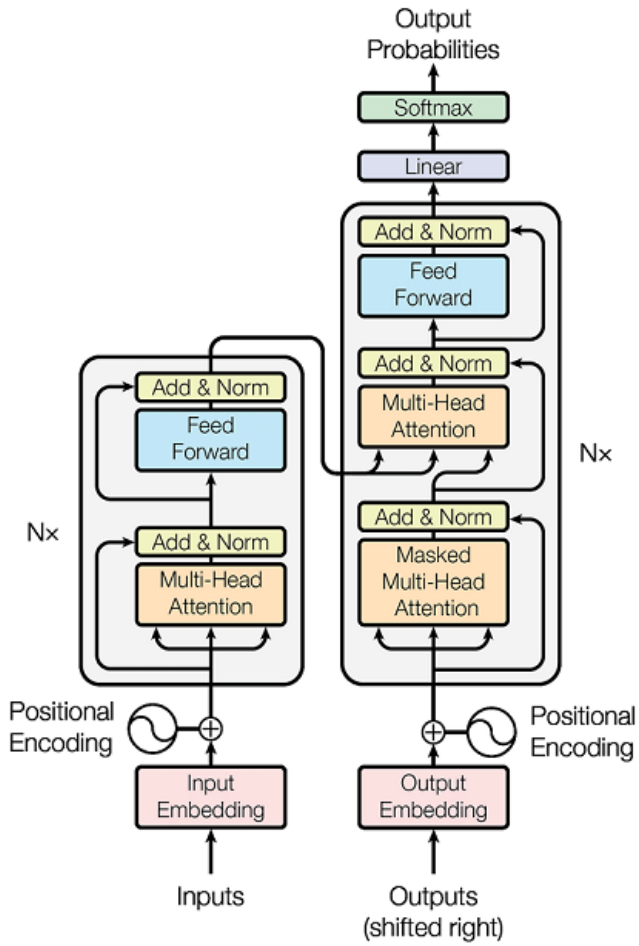


Figure: Caption

Transformers

Researchers eventually realized in 2017 that the encoder-decoder architectures based on self attention were more effective at storing information than RNN-based structures.



Review Session 5

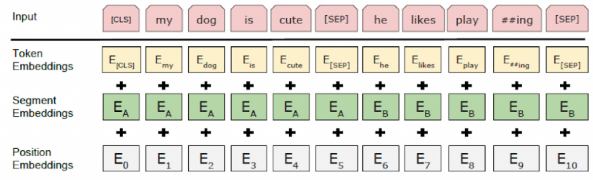
1. We considered recurrent neural networks
   - Simple RNN structures.
   - Long-Short-Term Memory (LSTM) networks.
   - Gated Recurrent Networks.
2. Considered Sequence to sequence models
   - These use Encoders and Decoders
   - Demonstrated the use of the sequence-to-sequence model as a language model.
3. Introduced attention as a way to more efficiently store importance information:
   - Modern forms of attention work on a key, query, and value principle.
   - Self attention has become the key to modern language model architectures.

Cambium Assessment

Break

Transformers and Language models

BERT (1969)



BERT (2019)

Aims

1. To introduce structure of the Bidirectional Encoding Representation by Transformers (BERT) model.
2. Understand how it can be fine-tuned to a variety of down-stream tasks in NLU and NLG.
3. Show how to fine-tune a model for a particular task using a Trainer.

Cambium Assessment

Context

Consider an embedding for the word “stick”.

The sentences “I poked it with a stick”, ”stick to the plan”, ”stick together”, and ”Wearing red will make me stick out” are all different ways to use the word ”stick”.

Cambium Assessment

Context

The aim of NLP tasks is **not only to understand single words** individually, but to be able to **understand the context of those words**

- **Classifying whole sentences**: Getting the sentiment of a review, detecting if an email is spam, determining if a sentence is grammatically correct or whether two sentences are logically related or not
- **Classifying each word in a sentence:** Identifying the grammatical components of a sentence (noun, verb, adjective), or the named entities (person, location, organization)

Cambium Assessment

Context

The aim of NLP tasks is **not only to understand single words** individually, but to be able to **understand the context of those words**

- **Generating text content**: Completing a prompt with auto-generated text, filling in the blanks in a text with masked words
- **Extracting an answer from a text**: Given a question and a context, extracting the answer to the question based on the information provided in the context
- **Generating a new sentence from an input text:** Translating a text into another language, summarizing a text

Cambium Assessment

Why is it Challenging?

- Computers don't process information in the same way as humans.
- For example, when we read the sentence "I am hungry," we can easily understand its meaning.
- Similarly, given two sentences such as "I am hungry" and "I am sad," we're able to easily determine how similar they are.
- For machine learning (ML) models, such tasks are more difficult.
- The text needs to be processed in a way that enables the model to learn from it.
- And because language is complex, we need to think carefully about how this processing must be done.

Cambium Assessment

Masked Language Model

BERT stands for Bidirectional Encoding Representation by Transformers was devised to meet these challenges:

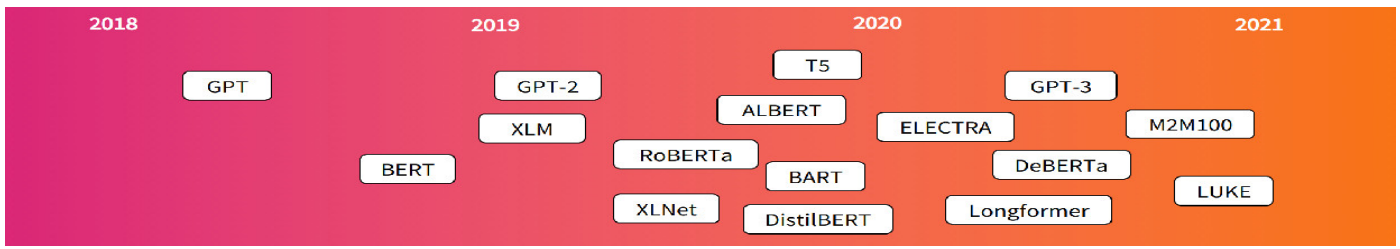
- The architecture of the BERT model uses layers of transformer units.
- The model is trained as a masked language model; 15% of the words in a large corpus of Wikipedia articles.
- The output of the model is a context aware representation of each word.
- BERT was one of the first (see GPT) such language models. Variations on the BERT architecture are now as numerous as BERT is large.

Cambium Assessment

A Bit of Transformer History

Here are some reference points in the (short) history of Transformer models:

- The Transformer architecture was introduced in June 2017.
- The focus of the original research was on translation tasks. This was followed by the introduction of several influential models, including:
  - GPT-like (also called auto-regressive Transformer models)
  - BERT-like (also called auto-encoding Transformer models)
  - BART/T5-like (also called sequence-to-sequence Transformer models)



Cambium Assessment

Transformers Are Language Models?

- All the Transformer models mentioned above (GPT, BERT, BART, T5, etc.) have been trained as language models.
- This means they have been trained on large amounts of raw text in a self-supervised fashion.
- Self-supervised learning is a type of training in which the objective is automatically computed from the inputs of the model.
- That means that humans are not needed to label the data,

Cambium Assessment

Transformers Are Big Models

- Apart from a few outliers (like DistilBERT), the general strategy to achieve better performance is by increasing the models' sizes as well as the amount of data they are pretrained on.
- Unfortunately, training a model, especially a large one, requires a large amount of data.
- Self-supervised learning is a type of training in which the objective is automatically computed from the inputs of the model.
- This becomes very costly in terms of time and compute resources.
- Imagine if each time a research team, a student organization, or a company wanted to train a model, it did so from scratch. This would lead to huge, unnecessary global costs!

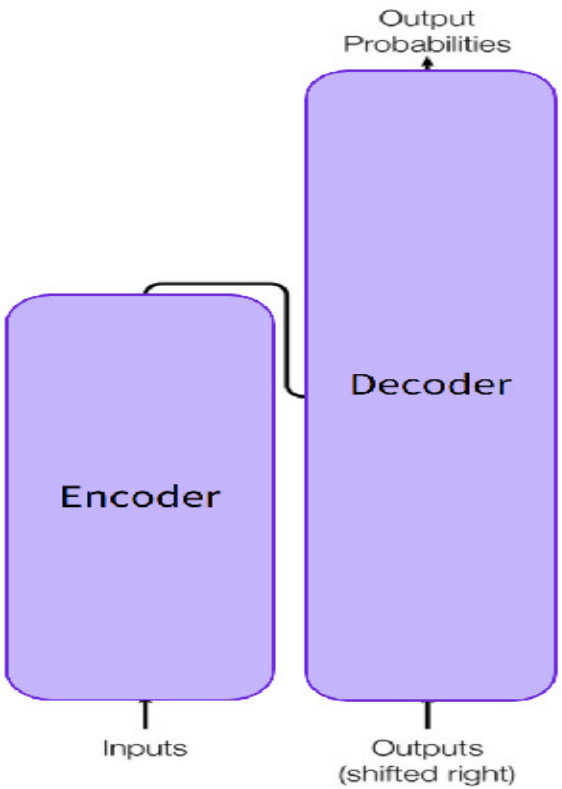
Cambium Assessment

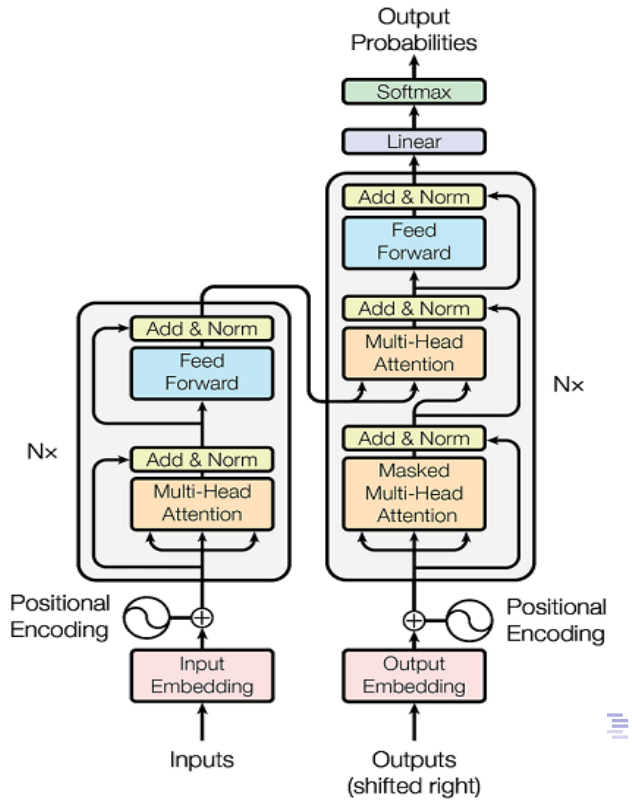
General Architecture

The model is primarily composed of two blocks:

Encoder: The encoder receives an input and builds a representation of it (its features). This means that the model is optimized to acquire understanding from the input.

Decoder: The decoder uses the encoder's representation (features) along with other inputs to generate a target sequence. This means that the model is optimized for generating outputs.





General Architecture

Each of these parts can be used independently, depending on the task:

- **Encoder-only models**: Good for tasks that require understanding of the input, such as sentence classification and named entity recognition.

- **Decoder-only models**: Good for generative tasks such as text generation.

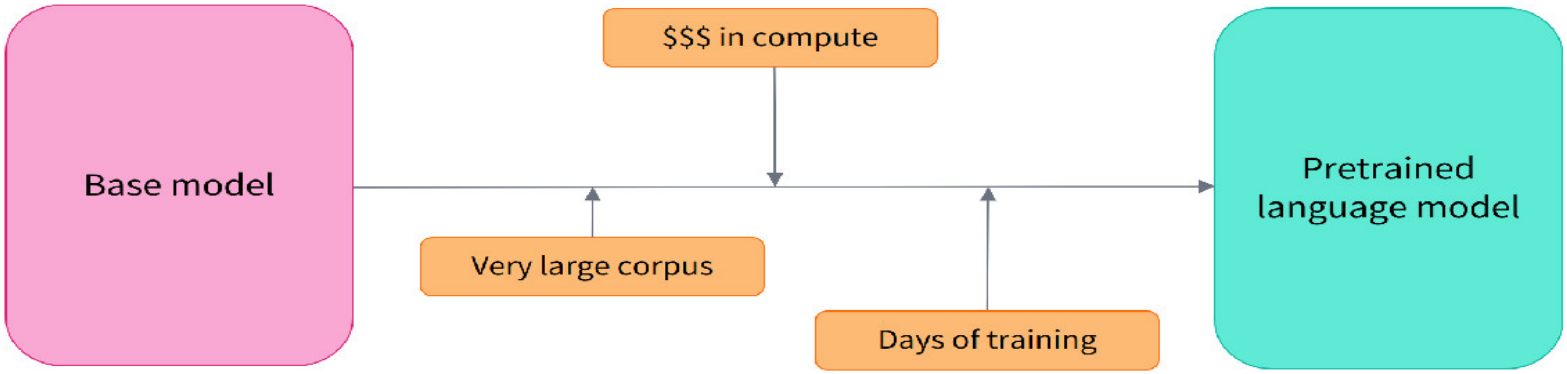
- **Encoder-decoder models or sequence-to-sequence models**: Good for generative tasks that require an input, such as translation or summarization.

Cambium Assessment

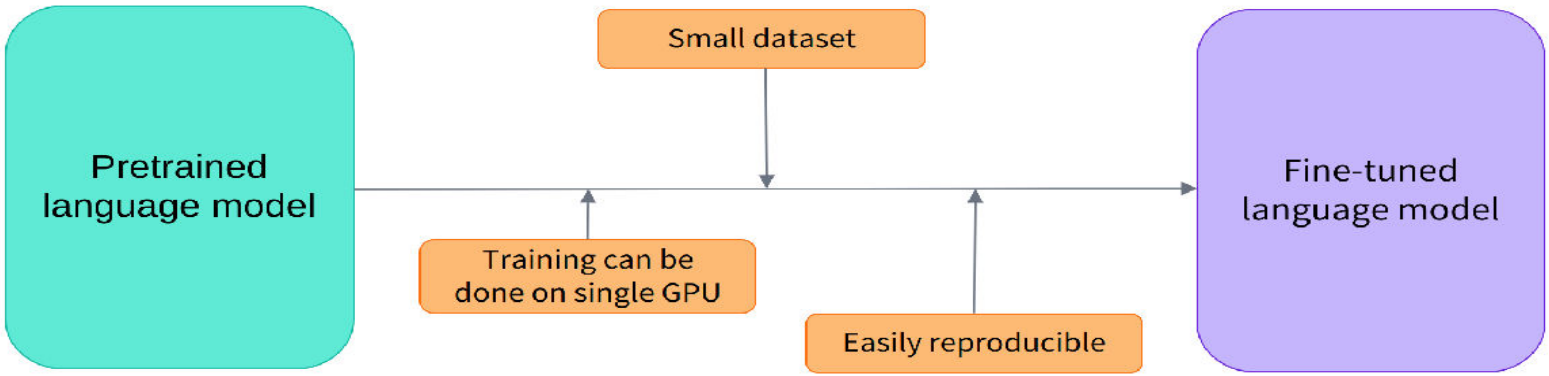
Transfer Learning

This process is called fine-tuning:

- Pretraining is the act of training a model from scratch.



- Fine-tuning, on the other hand, is the training done after a model has been pretrained.



Cambium Assessment

Attention Layers

Attention plays the key role in transformer-based inference:

- A key feature of Transformer models is that they are built with special layers called attention layers.
- All you need to know is that this layer will tell the model to pay specific attention to certain words in the sentence you passed it.
- The same concept applies to any task associated with natural language.
- A word by itself has a meaning, but that meaning is deeply affected by the context, which can be any other word (or words) before or after the word being studied.

Cambium Assessment

Accessing the models

We can access the pretrained model using the huggingface transformers library. We can simply import the pipeline, specify which pretrained model to use and call the pipeline:

```
from transformers import pipeline

text = "I am a [MASK] model"
model = "bert-base-uncased"
unmasker = pipeline(fill-mask, model=model)
unmasker(text)
```

This code exposes the base functionality of the BERT model. In particular this has the potential to expose bias in the model:

```
unmasker("This man works as a [MASK] .", top_k=3)
unmasker("This woman works as a [MASK] .", top_k=3)
```

Cambium Assessment

Transformer Heads

There are many different architectures available in Transformers, with each one designed around tackling a specific task.

Model (retrieve the hidden states)

ForCausalLM

ForMaskedLM

ForMultipleChoice

ForQuestionAnswering

ForSequenceClassification

ForTokenClassification

Cambium Assessment

Pipelines

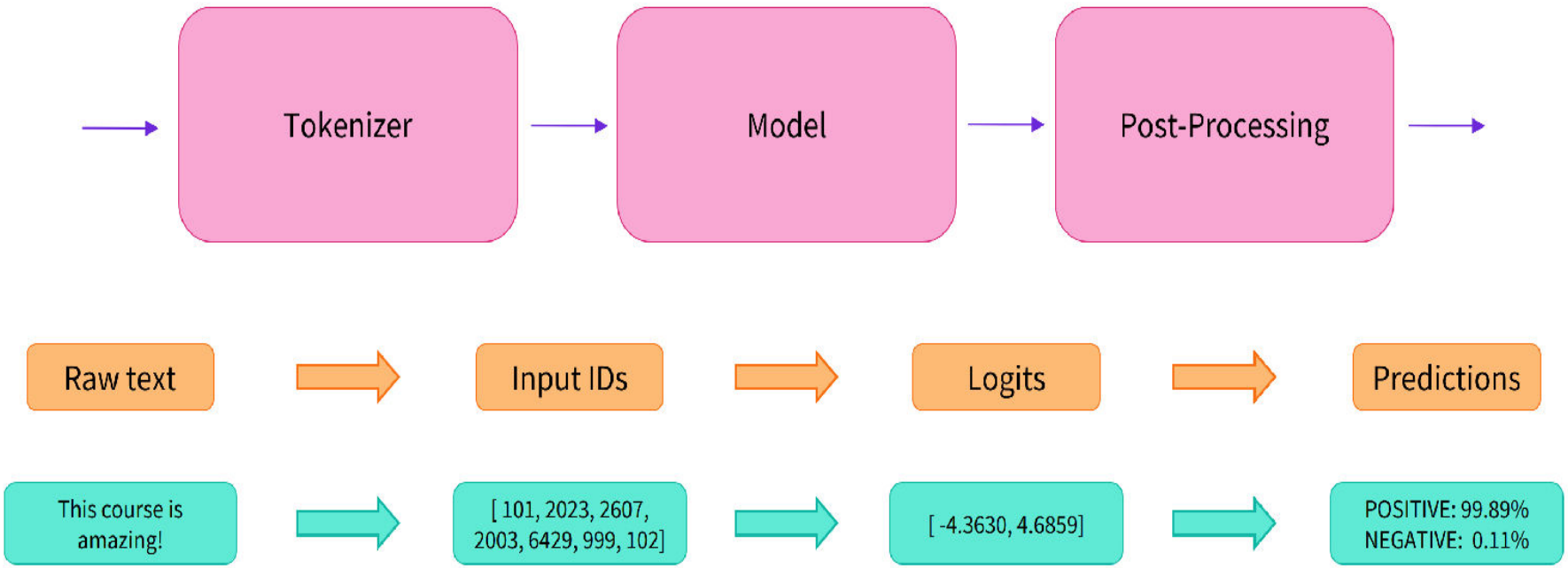
Pipelines can be used to simplify the use of a pretrained transformer model and a corresponding head.

Some examples of the pipelines can be used as follows:

```
from transformers
question_answerer = pipeline("question-answering")
question_answerer(
question="Where do I work?",
context="My name is Amir and I work at CL in in District of Columbia office.")

ner = pipeline("ner", grouped_entities=True)
ner("My name is Amir and I work at CL in District of Columbia office.")

generator = pipeline("text-generation", model="distilgpt2")
generator("In this session, we will teach you how to", max_length=30)

sentiment = pipeline("sentiment-analysis", truncation="only_first")
sentiment("Happy happy, joy joy.")

classifier = pipeline("zero-shot-classification")
classifier("This workshop session is about machine learning and artifical inteligence",
           candidate_labels=["education", "politics", "business", "data science"],)

summarizer = pipeline("summarization",max_length= 100)
summarizer(text)
```

Behind the Pipeline

- The pipeline groups together three steps: preprocessing, passing the inputs through the model, and postprocessing:



Applying a pipeline to IMDB

The IMDB dataset is well known, and to access a model trained on the data is a matter of accessing the pipeline. We use a distilled BERT model for speed:

```
>> sentiment = pipeline("sentiment-analysis",
truncation="only_first")
>> sentiment("It was the worst experience of my life")
[{'label': 'LABEL_0', 'score': 0.9974}]

>> preds = [sentiment(x['text']) for
x in tqdm(test_data)]
>> targets = [x['label'] for x in test_data]
```

The accuracy is 92.44%, the SMD is 0.031, and the QWK is 0.849.

Cambium Assessment

Results

Ultimately, Language models continue to give excellent results. If we compare the two other methods we gave, it is clear that the BERT model outperforms them easily:

Engine	QWK	SMD	Accuracy
BoW	0.737	0.015	86.8%
LSTM	0.781	0.046	89.0%
Distilled-BERT	0.849	0.022	92.2%

Table: The three primary statistics, QWK, Accuracy, and SMD, for the IMDB dataset.

Cambium Assessment

Trainer

Suppose we had our own dataset, we want to be able to fit our own data. The huggingface API gives us a range of tools to help with training. These tools have several advantages:

- Automates the training steps and epochs.
- Automatically handles different devices like GPUs.
- Integrates with a range of logging and diagnostic tools (weights and biases, tensorboard)
- Uses best practices for training by default (best optimizers, learning rate scheduler, gradient clipping).

There are also some key disadvantages:

- Steep learning curve due to a massive number of options.
- The trainer class is still in active development, not all changes are backwards compatible.
- The documentation to get specific functionality still needs work.
- Very difficult to debug when something goes wrong.

Cambium Assessment

Training a classifier

To start our training code, we first pick the model architecture and the data:

```
model = AutoModelForSequenceClassification.from_pretrained(
"bert-base-uncased", num_hidden_layers=4)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokenize = lambda x:tokenizer.encode_plus(x['text'],
truncation="only_first",  return_tensors = "pt",
padding="max_length", max_length=512)

train_data = load_dataset("imdb", split="train")
test_data = load_dataset("imdb", split="test")
token_train_data = train_data.map(tokenize).shuffle()
token_test_data = test_data.map(tokenize).shuffle()
```

This sets up the tokenization, model, and data.

Cambium Assessment

Collator

We just need to define a function that takes a list of our elements and formats it in the way the model expects:

```
def collator(batch):
  input_ids = torch.tensor([x['input_ids'][0]
  for x in batch])
  attention_mask = torch.tensor([x['attention_mask'][0]
  for x in batch])
  token_type_ids = torch.tensor([x['token_type_ids'][0]
  for x in batch])
  labels = torch.tensor([x['label'] for x in batch])
  return {"input_ids":input_ids,
          "attention_mask":attention_mask,
          "token_type_ids":token_type_ids,
          'labels':labels}
```

Cambium Assessment

Trainer

While the training arguments have an absolutely massive number of options, we present our simplest form of training as follows:

```
args = TrainingArguments(output_dir="tmp",
                         per_device_train_batch_size=2,
                         per_device_eval_batch_size=2)
trainer = Trainer(model,
                  args,
                  data_collator = collator,
                  train_dataset = token_train_data)
trainer.train()
```

This code will deploy on the GPU if the libraries are properly set up (not in Colab).

Cambium Assessment

Other considerations

There are several considerations when dealing with these models:

- They are very very large, and often pose considerable engineering difficulties in deployment.
- Their size makes them subject to potential over-fitting.
- There is a huge variety of smaller models (Tiny BERT, Electra, mobileBERT, ConvBERT) with much smaller footprints.
- We need to be aware that these models are trained on corpora that are of a vastly different nature to student text.
- The explainability of these models is still an open research area.
- These computations have only been made feasible by using a GPU instead of a CPU.

Cambium Assessment

Review for Session 6

- We introduced BERT:
  - The outputs of BERT are context dependent embeddings.
  - BERT is trained as a masked Language model over large corpora.
  - Uses an Encoder-decoder architecture consisting of attention-based layers known as transformers.
- A trained BERT model is an excellent starting point for a variety of tasks.
  - Sequence classification: Sentiment analysis, automated text scoring.
  - Token classification: POS tagging, Question answering.
  - Sequence-to-sequence tasks: Translation, summarization.
- Models are usually large:
  - Often provides state-of-the-art results across a range of tasks.
  - Explain-ability is a open area of research.
  - Difficult to deploy.
  - Subject to over-fitting.

Cambium Assessment