Deep Probabilistic Models

Wilker Aziz
ILLC @ UvA

Outline

Overview

In DL4NLP you learn about

- a representative sample of NLP problems
- various NN architectures and how they power selected NLP applications
- tricks of the trade necessary to get DL4NLP off the ground
- and also how to go beyond the designs and applications we cover in the course

To help with the last point, in this part of the course we will revisit the way we see, specify, and talk about *DL models* to put them under the lens of the probabilistic modelling framework.

We will use a bit of the language developed to talk about probabilistic graphical models. If you would like to learn *all* about PGMs, check the excellent book by Koller and Friedman (2009). I'd recommend Part I (on representation of distributions) to *anyone* working with machine learning. You can safely skip all proofs in the book, if you are not interested in that kind of thing.

Probabilistic models

A probabilistic model *predicts* possible outcomes of a random experiment.

Most modern ML models, including DL models, are probabilistic.

DL4NLP is no exception, but DL literature often emphasises implementation and algorithmic aspects over statistical considerations about the data and the model.

This part of the course is meant to bring statistics back to the fore as doing so allows us to

- talk about models in greater generality
- extend existing models in interesting ways
- go far beyond the examples we will cover in the course

DL, as any field, has developed a highly specialised language to talk about its important concepts and ideas. Though it's only expected that a community optimises its means of communicating ideas, a warning is due. More often than not this language conflates ideas pertinent to different aspects of a complete ML solution such as statistical dependence, parameterisation, statistical inference, parameter estimation, decision making, as well as algorithmic tricks and popular design choices. It also often makes DL-specific use of terms that have been previously established in other contexts (such as ML more generally or statistics), which can be confusing depending on where you approach DL from.

On DL language

In DL we typically talk about models in terms of their implementation. This causes us to conflate the statistical model, its parameterisation, statistical inference, parameter estimation, decision making, and even tricks of the trade (‘empirical common wisdom’).

Example (NMT): we use a BiLSTM encoder, and an attention-based LSTM decoder with input feeding, at each time step the model is trained to predict the next target word with teacher forcing, the model is trained to minimize the categorical cross entropy loss.

We will learn to disentangle the different ingredients mentioned in the example. In times, this will require distancing ourselves from DL design and practice.

MT data consist of pairs of sentences that convey the same ideas but are expressed in different languages. The *source* language is the language we want to translate from, the *target* language is the language we want to translate into.

A **BiLSTM encoder** is a combination of two LSTMs they process the same source sentence, but in opposite order. An **attention-based decoder** is an LSTM that processes the target sentence one token at a time while conditioning on the outputs of the encoder. As the encoder exposes a variable number of hidden states, the decoder uses a weighted average of encoder states with weights specific to each time step. The coefficients of this average are predicted by the so called attention mechanism. **Input feeding** is a very specific way to “wire” the encoder, decoder, and attention mechanism named as such by Luong et al. (2015). **Teacher forcing** is when NNs condition on inputs sampled from the data (gold-standard) rather than sampled from the model itself.

This description also suggests that NNs learn by predicting data, this is seldom the case. More often than not, NNs learn by assigning probability to observed data. Such NNs predict probability values (not data).

On DL design and practice

Implementation aspects are important for obvious reasons: ultimately one has to implement a model that works! This requires being concrete about

- the design of NN architecture blocks (e.g., encoder, decoder)
- how to wire blocks together (e.g., residual connections)
- how to train the model (e.g., objective, curriculum, optimiser)
- and how to make predictions (e.g., beam search, sampling).

Example I: LSTMs are superior to vanilla RNNs for they address numerical problems with repeated application of the chain rule of derivatives. That said, from a statistical standpoint, LSTMs and vanilla RNNs play interchangeable roles (i.e., allow to condition on a sequence of arbitrary length).

Example II: Dropout addresses overfitting in FFNNs, but leads to catastrophic forgetting in recurrent models. From a probabilistic perspective, we can explain why dropout works and how to 'fix it' to support the recurrent case.

For certain (statistical) arguments it is irrelevant whether we employ RNNs, LSTMs, GRUs, RCNNs, or certain instances of the Transformer.

Though note the point is not a dismissive one and when it comes to implementation of a certain model decisions of this kind certainly matter.

For example, these architectures are all quite different in their design, some are smaller, some are rather big, some are more parallelisable, some less, etc. Training them effectively is usually non-trivial, often requiring awareness of a number of empirically-validated recipes.

The point is one of zooming in and zooming out depending on our goals.

Dropout (Srivastava et al., 2014) is a technique where we randomly set some inputs of a layer to 0. A special type of approximations to a Bayesian model corresponds very closely to dropout (Gal and Ghahramani, 2016b) and leads to extensions to more complex architecture blocks (Gal and Ghahramani, 2016a,c).

Beyond DL design and practice

Many architectural differences have little statistical substance and abstracting away from them helps us concentrate on issues that are statistical in nature.

Statistical considerations concern the specification of joint distributions

- Are there conditional independences that should or should not hold?
- Are the distributions (and their marginals) complex enough?
  - Can they capture enough modes?
  - Do they underestimate variance of data by design?

These considerations become more and more prominent where we lack supervision for some aspects of our problem, as we shall see.

Example (graph networks):

In GNs, a node encoding is updated by composing a transformation of messages from its neighbouring nodes in a graph. A message is just a vector of scalars, and composition is usually done via weighted average (attention) with learned/predicted coefficients. The coefficients of this average may be independently normalised to 1 (via sigmoid) or jointly normalised to 1 (via softmax). Note that, with K neighbours, sigmoid attention coefficients add to at most K while softmax attention coefficients always add to 1.

Which activation will work best in an application *is not* a matter of statistics, valid arguments may concern numerical stability, the nature of the problem and inductive biases inherent to the activation (e.g., does node connectivity vary widely through the graph?).

Beyond DL design and practice - Example

Have you ever wondered why we have some pre-determined pairings of output activation and loss function that just go well together?

Understanding the statistical model we are specifying (i.e., being conscious about the choices we make) will make these pairings rather obvious to us.

We will also learn to move beyond the list of pairs you've come across, we will be able to create our own (even fairly non-trivial) pairs.

We will see that different losses are different implementations of the same basic principles but under different modelling choices.

In the frequentist case, this is the likelihood principle, which will motivate several algorithms for parameter estimation, the most popular algorithms you know are likely in this class.

In the Bayesian case, which we discuss later in the course, this basic principle is nothing but probabilistic inference which allows us to update our beliefs about hypotheses as more information becomes available.

Outline

Modelling random experiments

We are now primarily interested in data (as opposed to tasks).

- Think of a task as a potential application of a probabilistic model.
- Think of the data as the main subject of statistical interest.
- Think of the model as an attempt to shed light onto properties of data (or our own understanding of such properties).

Data are the outcomes of experiments involving random variables.

We are no longer modelling tasks. We are modelling random experiments. We will de-emphasise the intended use of the model (the downstream task) and emphasise the job of the designer (the statistical task).

Modelling observed random variables

Our goal is to learn a distribution over a set of **observed** random variables.

Observed random variables are the result of random experiments that have already happened: e.g., sentences in a collection of news articles, number of stars in a product review.

Think about this for a second. There's at least one aspect of a model that *cannot* possibly be observed. Thus saying that a lot of ML is about joint distributions over observed random variables only means we are adopting a particular view of statistics. What might that view be? And what would the alternative be?

Is there any case where *predicting the future* is not the goal in ML? How about in NLP specifically?

Modelling observed random variables

Our goal is to learn a distribution over a set of **observed** random variables.

Observed random variables are the result of random experiments that have already happened: e.g., sentences in a collection of news articles, number of stars in a product review.

Typical use in ML: *conditional models.*
▷ *We are given some variables, call them inputs (for now), and we are interested in making predictions about other variables, outputs (for now)*

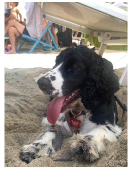
- such inputs are also called *predictors* (or *covariates*)
- with some probability, *predicted by the model*, an output takes on a certain *outcome* in a sample space

More often than not our goal is to predict output values for future inputs.

Think about this for a second. There's at least one aspect of a model that *cannot* possibly be observed. Thus saying that a lot of ML is about joint distributions over observed random variables only means we are adopting a particular view of statistics. What might that view be? And what would the alternative be?

Is there any case where *predicting the future* is not the goal in ML? How about in NLP specifically?

Modelling conditionally - Examples

Predictor	Outcome	Sample space
Why did they bother recording this???	$\star$	$\{\star, \star\star, \star\star\star, \star\star\star\star, \star\star\star\star\star\}$
Source: geen standaard MT: no standard	compare('no step')=0.5	$[0, 1]$
he proposed a famous solution to an inverse probability problem in the 18th century	https://en.wikipedia.org/wiki/Thomas_Bayes	$\mathcal{W}_{\text{en}}$
	*Pepper loves the beach!*	Σ^*_{en}
That's not possible!	*Dat is niet mogelijk!*	Σ^*_{nl}

Here we have some observations, inputs variables and output variables. A probabilistic model of the output variables conditioned on the input variables is a procedure that specifies a probability distribution over the sample space of the observed output variable.

The first example shows an instance of text classification. The model uses a piece of text to predict a discrete distribution over 5 outcomes. As a task, the goal might be to decide how many stars a new record deserves.

The second example is an instance of MT quality estimation. The model uses a source sentence and its machine translation to predict a distribution over a continuous interval of quality scores. As a task, the goal might be to decide whether the translation is good enough for publication or requires post-editing (this might be based on a threshold or some other rule).

The third example is an instance of question answering. The model uses a passage of text to predict a distribution over Wikipedia pages. As a task, the goal might be to spot the entity that the passage most likely refers to.

The fourth example is an instance of image captioning. The model uses an image to predict a distribution over English sentences. As a task, we might want to describe images to the visually impaired.

The last example is an instance of translation. The model uses an English sentence to predict a distribution over Dutch sentences. As a task, we might be interested in giving users on the web quick access to translated search results. In the last two examples, the sample space is discrete and *unbounded*.

To model or not to model?

Oftentimes, a model sees some observations as *deterministic* predictors. These are never *modelled*, they are only *conditioned on*:

- if a variable is not modelled, our statistical model cannot assign a probability to any observed value of that variable nor generate random draws for that variable
- the variable can, however, be used in some calculation

Example: a review in a sentiment classifier

- some NN "reads it" to compute a probability distribution over sentiment levels (e.g., negative, neutral, positive)
- the model has no clue how reviews come about, it is only concerned with reading them, not modelling their generative process

Deterministic predictors are usually the kinds of input we talk about when we mean 'inputs at test time' (e.g., the source sentence in MT).

An example of stochastic predictor is the prefix of already generated words in a partial translation. The model can assign probability to those (for example, in training), and it might even have generated those (for example, in test). Once those outcomes are already in place, they act as predictors so we can continue translating.

What if we model all observations?

That is, including the predictors. Then we get *joint* or *generative models*.

Think of these models as models that can generate their own predictors with some probability.

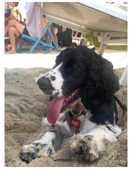
All previous examples can be modelled generatively.

In some contexts, especially classification and regression, conditional models are called discriminative models.

Generative models are sometimes also called joint models.

Don't get too caught up with the generative vs discriminative debate. Again, different points of view will lead to different uses of these labels. Some people will read 'discriminative' as something about the training algorithm, others as something about the nature of the distribution, others just appeal to analogies or traditions, there are also cases where things get tricky because of other disciplines (e.g., generative syntax in linguistics has nothing to do with statistics, though generative syntactic formalisms can be given statistical treatment, including via discriminative models).

Joint modelling - Examples

Joint outcome		Sample space
Why did they bother recording this???	$\star$	$\Sigma_{\text{en}}^* \times \{\star, \star\star, \star\star\star, \star\star\star\star, \star\star\star\star\star\}$
Source: geen standaard MT: no standard	compare('no step')=0.5	$\Sigma_{\text{nl}}^* \times \Sigma_{\text{en}}^* \times [0, 1]$
he proposed a famous solution to an inverse probability problem in the 18th century	https://en.wikipedia.org/wiki/Thomas_Bayes	$\Sigma_{\text{en}}^* \times \mathcal{W}_{\text{en}}$
	*Pepper loves the beach!*	$[256, 256, 256]^{h \times w} \times \Sigma_{\text{en}}^*$
That's not possible!	*Dat is niet mogelijk!*	$\Sigma_{\text{en}}^* \times \Sigma_{\text{nl}}^*$

The statistical model predicts a distribution over the sample space. Sample spaces can grow rather large, especially so when former deterministic predictors are to be treated as random variables (that is, they are now part of the random experiment we aim to predict).

Modelling unobserved random variables

Unobserved random variables are variables that are

- observable in principle, but for some reason are not available for observation (e.g., the topic of a piece of text, a binary bracketing tree)
- unobservable (e.g., a 100-dimensional continuous representation of a sentence, the collection of continuous parameters of a FFNN)

they help us prescribe and even estimate our models.

Our goal is to learn a distribution over *observed and unobserved rvs*

- make explicit assumptions about statistical dependence
- discover hidden structure (i.e., statistical dependence)
- mimic intuitions or knowledge about the data generating process
- deal with missing data
- estimate uncertainty about predictions

Deterministic predictors may also be available.

Unobserved random variables are also called latent variables.

For those interested in Bayesian statistics, note that the presence of unobserved random variables does not imply Bayesian modelling. Bayesian principles are a collection of ideas organised in what is called the Bayesian Theory (or Bayesian Decision Theory) for rational decision making under uncertainty (Bernardo and Smith, 2009). These ideas may cross paths with many aspects of our ML solutions.

Having latent variables is nothing but a design choice. Indeed embracing Bayesian principles calls for recognising at least some inherently unobservable variables (such as parameters in a parametric model, functions in non-parametric models, and sometimes even the model structure itself) and making inferences by application of probability calculus.

We will discuss models with latent variables and approach them both via Frequentist estimation and via approximations to Bayesian inference.

What are the benefits of probabilistic models?

Probabilistic models allows to incorporate assumptions through

- the choice of distribution
- dependencies among random variables
- the way that distributions uses side information
- stipulate unobserved data and their properties

They return a distribution over outcomes which can be used to

- generate data
- account for unobserved data
- provide explanation and suggest improvements
- inform decision makers

Enough to motivate combining them with NNs!

Any warnings?

In the presence of latent variables, crucial **computations become challenging** and often require approximations.

Discrete latent variables, particularly interesting in NLP, pose further challenges.

Stochastic parameters, typical of Bayesian modelling, too call for special treatment since NNs typically contain **a lot of parameters**.

The field that studies principled ways to approximate computations in probabilistic ML is known as *approximate inference*.

In most scenarios, you can think of the word inference as synonym to computation.

In DL the word inference is typically associated with a search procedure employed after training to make predictions.

In probabilistic ML, inference typically refers to computations that depend on solving a marginalisation or an expectation. Inference in that sense is a computation that aggregates information from various viewpoints. You can think of inference as standing in contrast to optimisation, which is not about aggregating, but about singling out.

Many linguistic generalisations/representations are discrete, but discreteness can lead to combinatorial growth and discontinuous cumulative distribution functions which have an impact on inference and optimisation.

What are you getting out of this?

As we progress we will

- develop a shared vocabulary to talk about probabilistic models powered by NNs
- derive crucial results step by step
- connect concepts and implementation

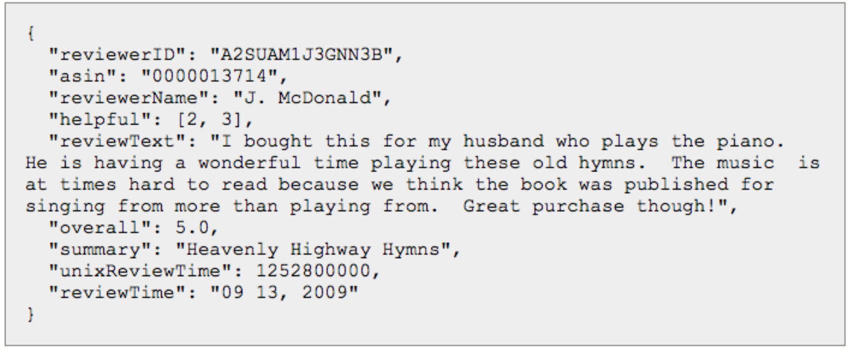
Goal

- you should be able to navigate through fresh literature
- and start combining probabilistic models and NNs

Outline

Example - Music reviews

I downloaded some reviews from Amazon, this is what a datum looks like

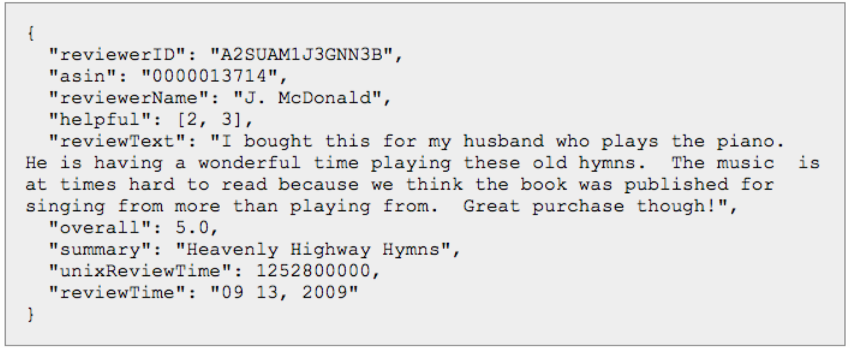


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.
He is having a wonderful time playing these old hymns.  The music  is
at times hard to read because we think the book was published for
singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```

We can observe the outcomes of many random variables here.
But do we care about all of them?

Example - Music reviews

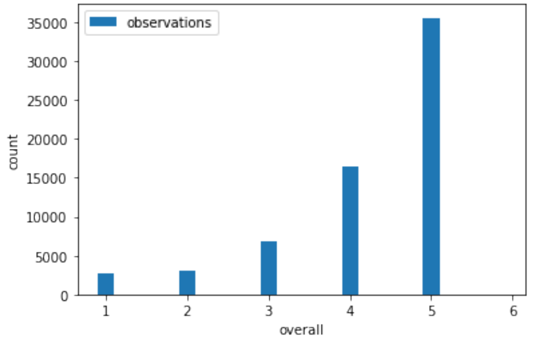
I downloaded some reviews from Amazon, this is what a datum looks like



```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.
He is having a wonderful time playing these old hymns.  The music  is
at times hard to read because we think the book was published for
singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```

Let's say we care about outcomes of *overall* score and let's visualise the observations available.

Example - Visualise data



Let's say we take the overall score assigned to any one review as a random variable (and ignore everything else in the dataset).

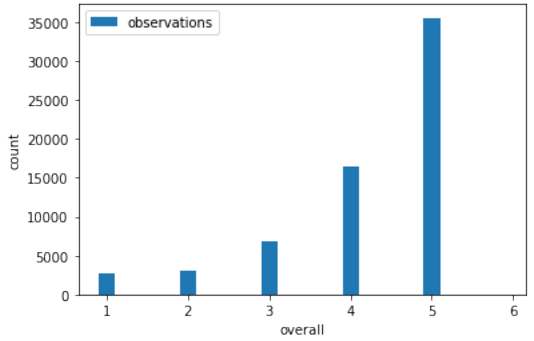
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



Can we capture the general pattern by prescribing some known probability distribution?

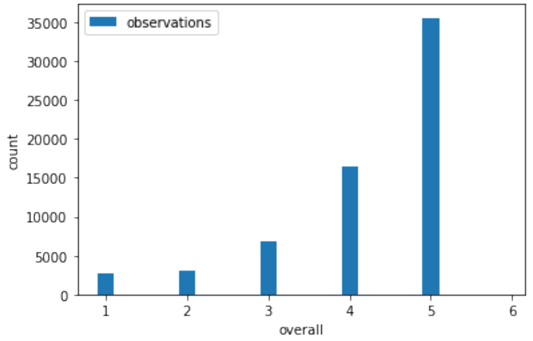
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



Let's say that overall scores are drawn from some Categorical distribution. That's fine, but which one?

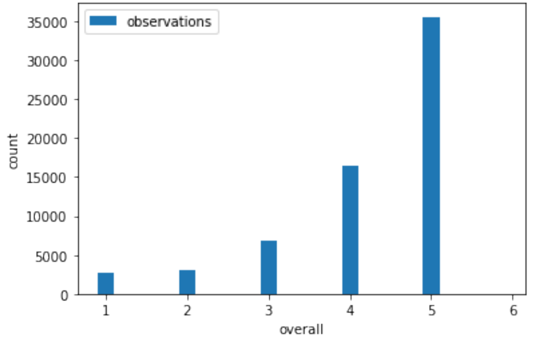
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



A member of the Categorical family of distributions is specified by a parameter: a vector of dense probability values whose elements sum to 1.

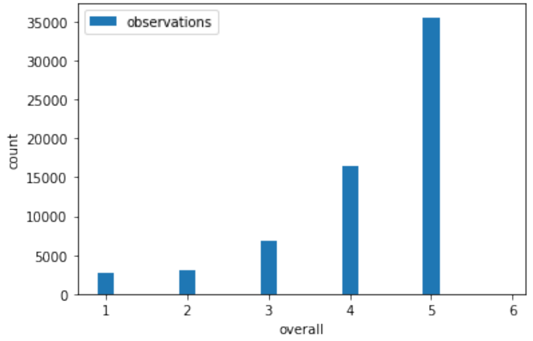
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



Let's pick the one that assigns maximum *likelihood* to our observations. And let's assume that our observations were obtained independently.

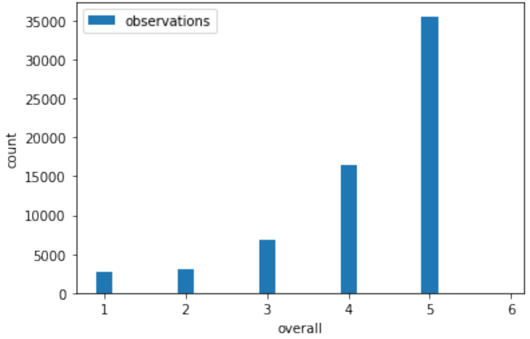
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



Let's see how we get to a *likelihood* value: we decided that $Y^{(1)} \sim Y^{(2)} \sim \cdots \sim Y^{(N)} \sim \text{Cat}(\phi_1, \ldots, \phi_5)$.

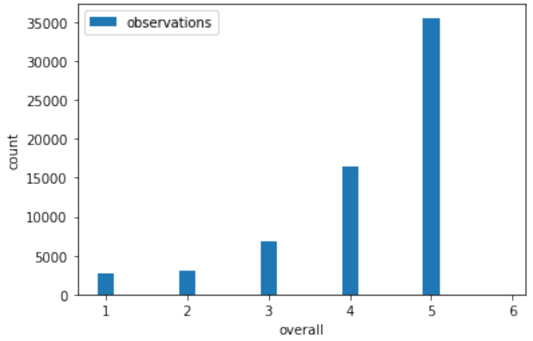
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



Thus $p(y^{(1)}, \ldots, y^{(N)}|\phi) = \prod_{i=1}^{N} \text{Cat}(y^{(i)}|\phi) = \prod_{i=1}^{N} \phi_{y^{(i)}}$ is the likelihood of observing the data by drawing N times from $\text{Cat}(\phi)$.

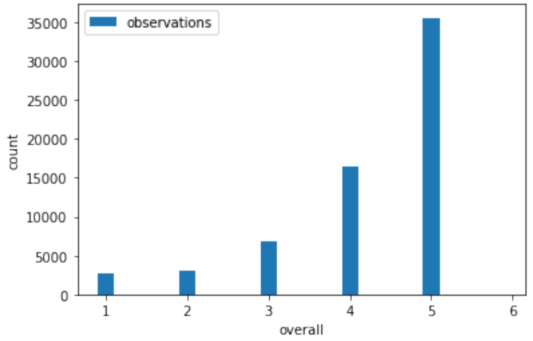
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



Let's think of this as a function $L(\phi)$ of the parameter ϕ, and solve $\arg\max_{\phi} L(\phi)$.

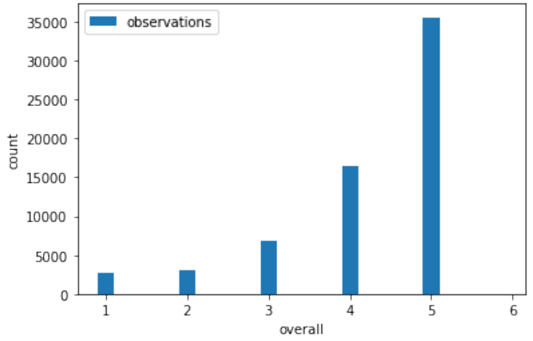
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



We call $L(\phi)$ the *likelihood function*, named this way because it is what we get when we interpret the *likelihood* of the data as function of ϕ.

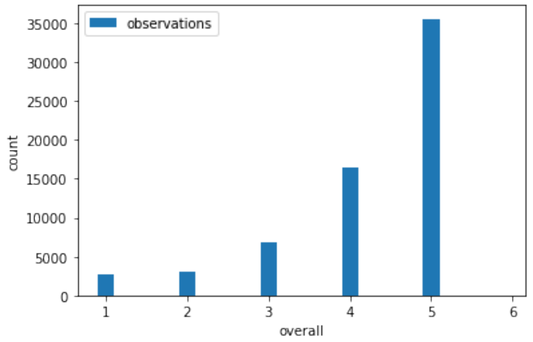
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



We can equivalently search for ϕ under the log-likelihood function $\mathcal{L}(\phi) = \log L(\phi)$ and solve $\arg\max_{\phi} \mathcal{L}(\phi)$.

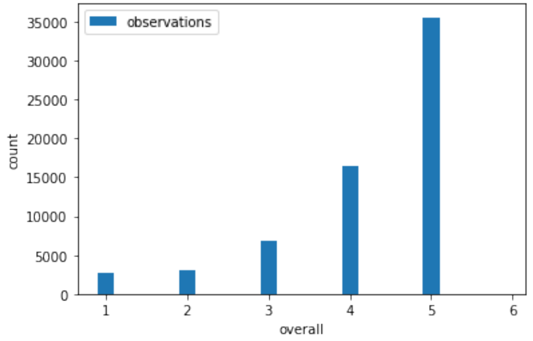
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Visualise data



If you solve this, you will see that the *maximum likelihood estimate* of the Categorical distribution is given by dividing the bars in the plot by N.

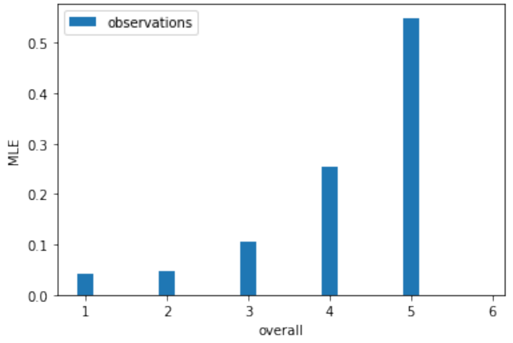
The first thing to note is that we can observe many outcomes.

The second thing to note is that we can capture the general pattern using a statistical distribution.

Let's, for example, assume that overall scores are drawn from some Categorical distribution. Which one? Well, one option is to pick the one that makes these observations as likely as possible.

To prove MLE for a Categorical likelihood you will need some proficiency with partial derivatives and Lagrangian multipliers. It's okay to also just find the MLE on Wikipedia or in a textbook.

Example - Fit a Categorical likelihood model by MLE



This clearly captures the exact pattern we saw before. Does this level of analysis meet the expectation of our target audience or do we need a more fine grained picture?

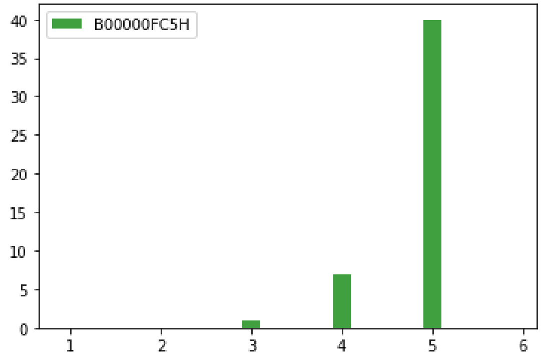
Whether we 'meet the expectation of the audience' is a matter of application. From the point of view of the statistical model, the job is done.

If the user is interested in learning more about products, buyers, and their relationship then this is probably insufficient.

For example, can we get a more fine grained picture if we condition on some predictors?
Recall, the review record contained a lot more information.

Example - Conditioning on predictors

Let's see how overall scores distribute for some **products**



There is a certain amount of variance that is characteristic of how people feel about the product. Modelling the data means modelling this variance, not ignoring it!

Can we say the first product is highly appreciated? Can we say the opposite about the second one?

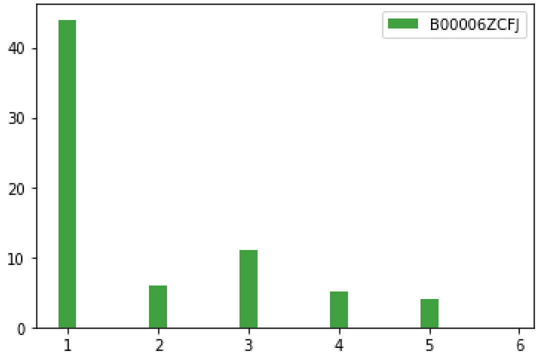
What can be said about the 3rd and the 4th?

Generally, what can be said is that committing to any overall score hides variance present in the data.

Whereas for some purposes we may have to choose a single score for a product, from the point of view of the statistical model that is not at all the goal. The goal is to model the data well, that is, closely reproducing statistical properties of the observed data (mean, variance, skew, ...).

Example - Conditioning on predictors

Let's see how overall scores distribute for some **products**



There is a certain amount of variance that is characteristic of how people feel about the product. Modelling the data means modelling this variance, not ignoring it!

Can we say the first product is highly appreciated? Can we say the opposite about the second one?

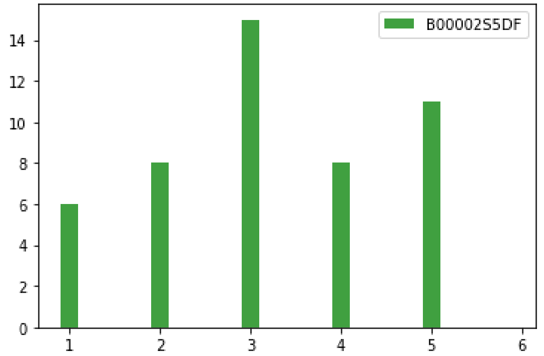
What can be said about the 3rd and the 4th?

Generally, what can be said is that committing to any overall score hides variance present in the data.

Whereas for some purposes we may have to choose a single score for a product, from the point of view of the statistical model that is not at all the goal. The goal is to model the data well, that is, closely reproducing statistical properties of the observed data (mean, variance, skew, ...).

Example - Conditioning on predictors

Let's see how overall scores distribute for some **products**



There is a certain amount of variance that is characteristic of how people feel about the product. Modelling the data means modelling this variance, not ignoring it!

Can we say the first product is highly appreciated? Can we say the opposite about the second one?

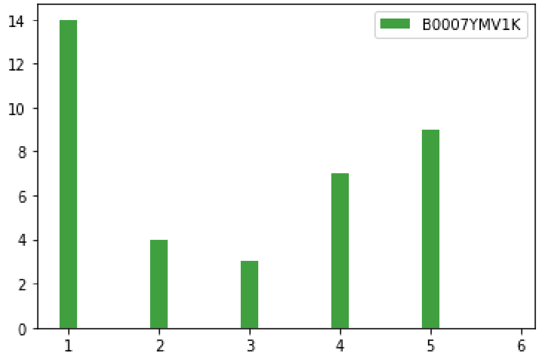
What can be said about the 3rd and the 4th?

Generally, what can be said is that committing to any overall score hides variance present in the data.

Whereas for some purposes we may have to choose a single score for a product, from the point of view of the statistical model that is not at all the goal. The goal is to model the data well, that is, closely reproducing statistical properties of the observed data (mean, variance, skew, ...).

Example - Conditioning on predictors

Let's see how overall scores distribute for some **products**



There is a certain amount of variance that is characteristic of how people feel about the product. Modelling the data means modelling this variance, not ignoring it!

Can we say the first product is highly appreciated? Can we say the opposite about the second one?

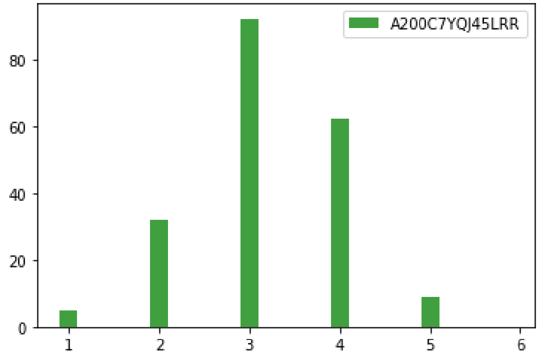
What can be said about the 3rd and the 4th?

Generally, what can be said is that committing to any overall score hides variance present in the data.

Whereas for some purposes we may have to choose a single score for a product, from the point of view of the statistical model that is not at all the goal. The goal is to model the data well, that is, closely reproducing statistical properties of the observed data (mean, variance, skew, ...).

Example - Conditioning on predictors

Let's see how overall scores distribute for some **reviewers**



There is a certain amount of variance that is typical of a reviewer (e.g., some people might not bother leaving a review unless they have a strong opinion).

Note that no matter how we view the data, we still find variance.

Explanation 1: *the data is the data* is the data.

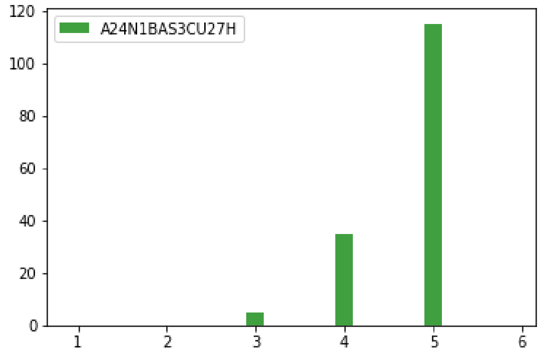
Explanation 2: noisy data.

We need to agree that *noisy data* is *the data*. Everything you observe gets to be called *data*. There is no such a thing as noisy-free data and no such a thing as *outliers*. If you get a dataset and *remove outliers* you essentially made different dataset that is artificially easier to model.

You will often see authors blaming the data for their oversight. That's either sloppy use of language or an incoherent view of statistics.

Example - Conditioning on predictors

Let's see how overall scores distribute for some **reviewers**



There is a certain amount of variance that is typical of a reviewer (e.g., some people might not bother leaving a review unless they have a strong opinion).

Note that no matter how we view the data, we still find variance.

Explanation 1: *the data is the data* is the data.

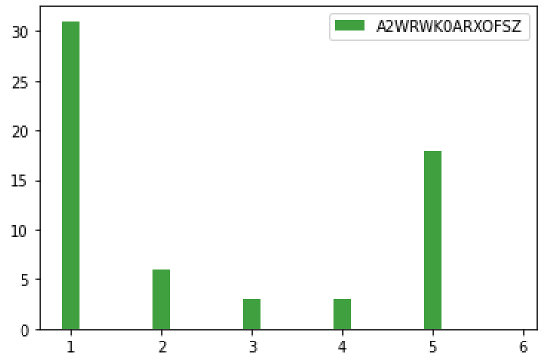
Explanation 2: noisy data.

We need to agree that *noisy data* is *the data*. Everything you observe gets to be called *data*. There is no such a thing as noisy-free data and no such a thing as *outliers*. If you get a dataset and *remove outliers* you essentially made different dataset that is artificially easier to model.

You will often see authors blaming the data for their oversight. That's either sloppy use of language or an incoherent view of statistics.

Example - Conditioning on predictors

Let's see how overall scores distribute for some **reviewers**



There is a certain amount of variance that is typical of a reviewer (e.g., some people might not bother leaving a review unless they have a strong opinion).

Note that no matter how we view the data, we still find variance.

Explanation 1: *the data is the data* is the data.

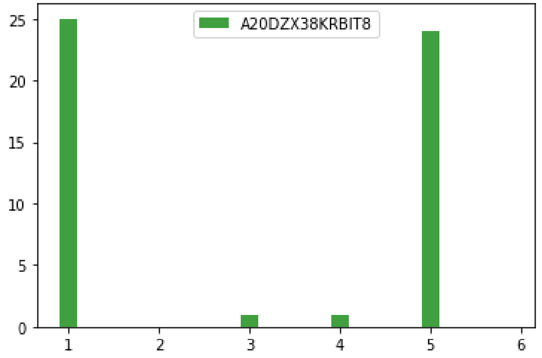
Explanation 2: noisy data.

We need to agree that *noisy data* is *the data*. Everything you observe gets to be called *data*. There is no such a thing as noisy-free data and no such a thing as *outliers*. If you get a dataset and *remove outliers* you essentially made different dataset that is artificially easier to model.

You will often see authors blaming the data for their oversight. That's either sloppy use of language or an incoherent view of statistics.

Example - Conditioning on predictors

Let's see how overall scores distribute for some **reviewers**



There is a certain amount of variance that is typical of a reviewer (e.g., some people might not bother leaving a review unless they have a strong opinion).

Note that no matter how we view the data, we still find variance.

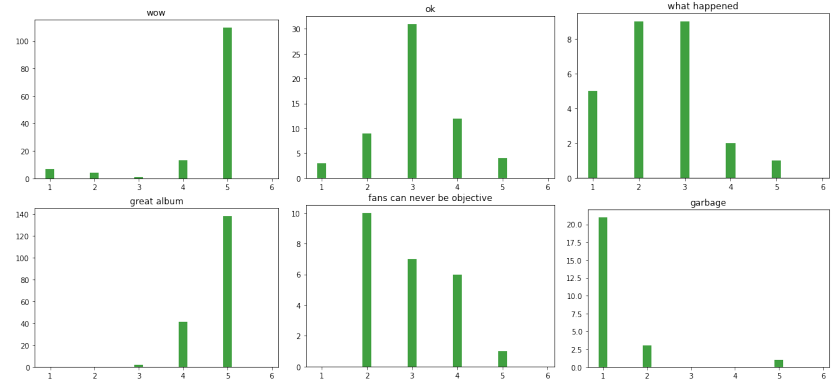
Explanation 1: *the data is the data* is the data.

Explanation 2: noisy data.

We need to agree that *noisy data* is *the data*. Everything you observe gets to be called *data*. There is no such a thing as noisy-free data and no such a thing as *outliers*. If you get a dataset and *remove outliers* you essentially made different dataset that is artificially easier to model.

You will often see authors blaming the data for their oversight. That's either sloppy use of language or an incoherent view of statistics.

Example - Conditioning on rich predictors (text)



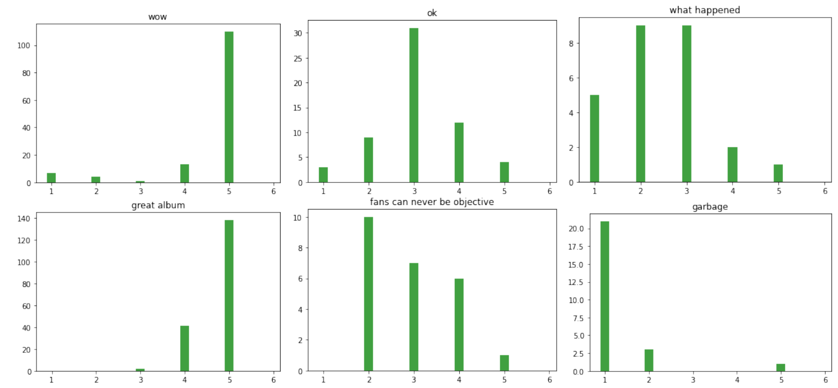
Reviewers contribute long reviews, but also short summaries. These are short enough that we can gather more than 20 different reviews per summary.

Top-left: wow
Top-middle: ok
Top-right: what happened
Bottom-left: great album
Bottom-middle: fans can never be objective
Bottom-right: garbage

Some remarks:

- note the trends are quite different from the general trend
- note that we cannot always be very confident about the overall score

Example - Conditioning on rich predictors (text)



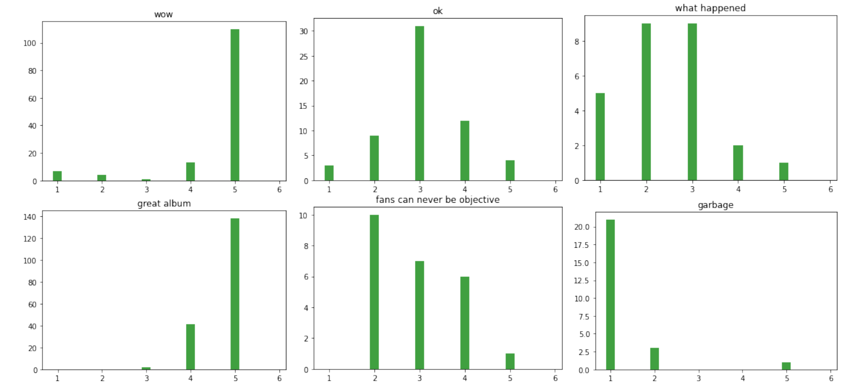
We could model the overall score $Y^{(i)}$ given a summary $x^{(i)} = s$ as a draw from the summary-specific Categorical distribution $\mathrm{Cat}(\phi^{(s)})$.

Top-left: wow
Top-middle: ok
Top-right: what happened
Bottom-left: great album
Bottom-middle: fans can never be objective
Bottom-right: garbage

Some remarks:

- note the trends are quite different from the general trend
- note that we cannot always be very confident about the overall score

Example - Conditioning on rich predictors (text)



But then we would have to estimate as many Categorical distributions as there are unique summaries. Clearly we will struggle in the future, when a novel summary pops up.

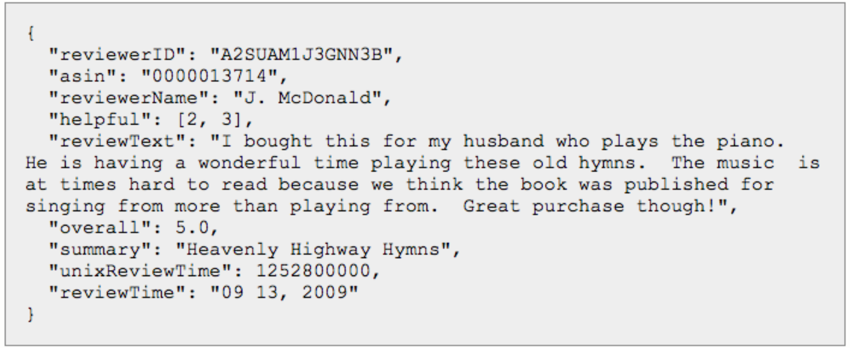
Top-left: wow
Top-middle: ok
Top-right: what happened
Bottom-left: great album
Bottom-middle: fans can never be objective
Bottom-right: garbage

Some remarks:

- note the trends are quite different from the general trend
- note that we cannot always be very confident about the overall score

Example - Very rich predictors

Consider a predictor like `reviewText`



```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.
He is having a wonderful time playing these old hymns.  The music  is
at times hard to read because we think the book was published for
singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```

We could not use it in the same way we used the summary. Given a certain reviewText, we typically only get 1 data point making the problem look deterministic. This is a fallacy though, it only *looks* deterministic because of a modelling choice.

If you are not convinced that the determinism is artificial, consider the following thought experiment. Ask everyone in the classroom to assign stars to a product based on a given review, what might you observe?

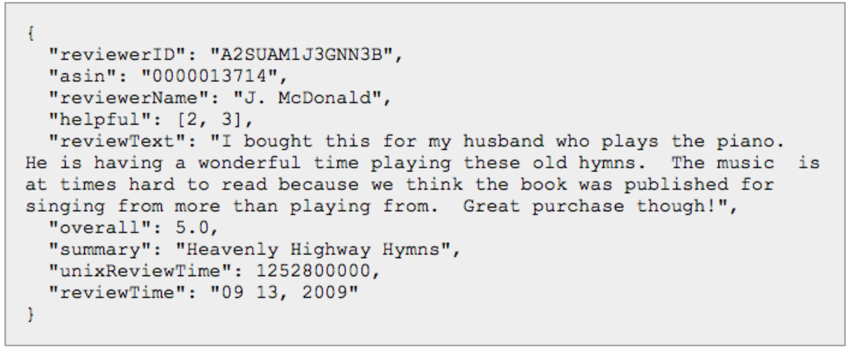
We cannot expect determinism, neither we should really look for it. Variance does not mean we are doing something wrong, the data are like that indeed: we lack knowledge about all factors involved in the data generating process.

Consider a person performing some annotation: variability can be result of lapse in attention, long working hours, or simply inherent to the task.

Whereas determining logical entailment seems rather trivial (well, it still depends on the excerpts of text we are given), translating is a much more creative process.

Machine learning and pattern recognition

Here is where things get interesting, even if a bit less well defined



```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.
He is having a wonderful time playing these old hymns.  The music  is
at times hard to read because we think the book was published for
singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```

We want to condition on rich predictors but we want to learn to identify and exploit patterns that *generalise* a certain hidden aspect of the data (e.g., how people relate products, their views, and a certain score).

Why do I say this is a bit less well defined?

We are embracing the idea of learning patterns that are specific enough to explain away most variance in the data, yet general enough to be reusable in the future.

We are alluding to some notion of *generalisation* without a clear idea of what it is or how to operationalise it.

Example - Learning to use very rich predictors

Pre 2010 NLP dealt with this mostly by identifying informative features $\varphi(x^{(i)})$ of the available predictor $x^{(i)}$. We would then map these features to the parameter of a Categorical distribution (e.g., via a log-linear model) on demand: $Y^{(i)}|w, x^{(i)} \sim \text{Cat}(\text{softmax}(W\varphi(x^{(i)}) + b))$.

In $Y^{(i)}|x^{(i)} \sim \text{Cat}(f(x^{(i)}; \theta))$, $f(\cdot; \theta)$ is a NN architecture with parameters θ, it maps any covariate $x^{(i)}$, say a long review in English, to the parameters of the distribution that governs the ith data point, say a probability vector over 5 classes.

Example - Learning to use very rich predictors

Pre 2010 NLP dealt with this mostly by identifying informative features $\varphi(x^{(i)})$ of the available predictor $x^{(i)}$. We would then map these features to the parameter of a Categorical distribution (e.g., via a log-linear model) on demand: $Y^{(i)}|w, x^{(i)} \sim \text{Cat}(\text{softmax}(W\varphi(x^{(i)}) + b))$.

In DL4NLP we condition on **everything available to us** by learning how to map from **arbitrarily complex** data to the parameters of our distributions. We do so with NNs: $Y^{(i)}|\theta, x^{(i)} \sim \text{Cat}(f(x^{(i)}; \theta))$.

There's a lot of research on how to design $f(\cdot; \theta)$ and estimate θ effectively. This course covers those aspects too!

In $Y^{(i)}|x^{(i)} \sim \text{Cat}(f(x^{(i)}; \theta))$, $f(\cdot; \theta)$ is a NN architecture with parameters θ, it maps any covariate $x^{(i)}$, say a long review in English, to the parameters of the distribution that governs the ith data point, say a probability vector over 5 classes.

Shallow statistical models

We have data $y^{(1)}, \dots, y^{(N)}$ e.g. sentences, images
generated by some **unknown** procedure which we assume can be captured by a probabilistic model

- with **known** probability (mass/density) function e.g.

$$Y \sim \text{Cat}(\phi_1, \dots, \phi_K) \qquad \text{or} \qquad Y \sim \mathcal{N}(\mu, \sigma^2)$$

Shallow statistical models

We have data $y^{(1)}, \ldots, y^{(N)}$ e.g. sentences, images
generated by some **unknown** procedure which we assume can be captured by a probabilistic model

- with **known** probability (mass/density) function e.g.

$$Y \sim \mathrm{Cat}(\phi_1, \ldots, \phi_K) \qquad \text{or} \qquad Y \sim \mathcal{N}(\mu, \sigma^2)$$

and estimate parameters that assign maximum likelihood to observations

Parameterisation by NNs

Let x be all side information available
e.g. *inputs/features/predictors/covariates*

Have neural networks predict parameters of our probabilistic model

$$Y|x \sim \mathrm{Cat}(f(x;\theta)) \qquad \text{or} \qquad Y|x \sim \mathcal{N}(\mu(x;\theta), \sigma(x;\theta)^2)$$

and proceed to estimate parameters θ of the NNs

NNs compute the parameters of the statistical model. We estimate NN parameters.

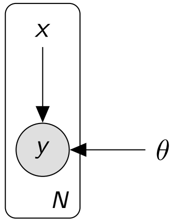
Graphical model

Random variables

- observed data
  $y^{(1)}, \ldots, y^{(N)}$

Deterministic variables

- predictors $x^{(1)}, \ldots, x^{(N)}$
  non-random observed variable
- model parameters θ
  non-random and unobservable variable



For now we are taking the Frequentist point of view where parameters are assumed known. Clearly we do not just happen to *know* the parameters of a statistical model, though we may be able to make a somewhat informed choice based of the available (training) data.

In Frequentism parameters are *determined* via optimisation of a likelihood-based criterion. This is known as parameter estimation.

When we discuss Bayesian principles we will see that alternatively we may acknowledge that parameters are random variables and that we don't know much about them (besides what can be coded in a choice of governing distribution known as prior). Then we dispense with parameter estimation altogether, rather using probabilistic inference to reason about quantities of interest such as probability queries about unobserved random variables given observed data. This is called posterior inference.

Task-driven feature extraction

Often our side information is itself some high dimensional object

- x is a sentence and y a tree
- x is the source sentence and y is the target
- x is an image and y is a caption

and part of the job of the NNs that parametrise our models is to also deterministically encode that input in a low-dimensional space

In representation learning, these encodings are the subject of interest, much more than the model itself.

Example: word embedding models learn to predict probability distributions over neighbouring words, but ultimately one only cares about the representations of those words, which is internal to the parameterisation of the distribution.

In fact, in representation learning we find many instances of deep learning models that are not probabilistic.

NN as efficient parametrisation

From a statistical point of view, NNs do not generate data

- they parametrise distributions that *by assumption* generate data
- compact and efficient way to map from complex side information to parameter space

From a statistical point of view, it is inconsequential whether you can do anything useful with intermediate representations inside of an NN.

Example: word embeddings by word2vec are *accidental* from a statistical point of view. The embeddings are not a unobserved random variable we modelled, they are part of the specification of some other distribution. I use the word 'accidental' just to get your attention. Clearly, the inventor engineered a specific task, and a specific classifier to get word2vec embeddings to be useful in the way they are. The inventor exploited inductive biases aimed at making embeddings function as if they captured lexical semantics.

What I hope is that you will see that while that there multiple, complementary, ways to code inductive biases. Manipulating statistical properties of the model is another one.

NN as efficient parametrisation

From a statistical point of view, NNs do not generate data

- they parametrise distributions that *by assumption* generate data
- compact and efficient way to map from complex side information to parameter space

Prediction is done by a decision rule outside the statistical model

- e.g. argmax, beam search

From a statistical point of view, it is inconsequential whether you can do anything useful with intermediate representations inside of an NN.

Example: word embeddings by word2vec are *accidental* from a statistical point of view. The embeddings are not a unobserved random variable we modelled, they are part of the specification of some other distribution. I use the word ‘accidental’ just to get your attention. Clearly, the inventor engineered a specific task, and a specific classifier to get word2vec embeddings to be useful in the way they are. The inventor exploited inductive biases aimed at making embeddings function as if they captured lexical semantics.

What I hope is that you will see that while that there multiple, complementary, ways to code inductive biases. Manipulating statistical properties of the model is another one.

Outline

Maximum likelihood estimation

We have a probability model of a random variable Y, and this model may condition on available covariates X. This model has parameters θ and assigns probability $p(y|x, \theta)$ to an observation.

Maximum likelihood estimation

We have a probability model of a random variable Y, and this model may condition on available covariates X. This model has parameters θ and assigns probability $p(y|x, \theta)$ to an observation.

Given a dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ of i.i.d. observations,

Maximum likelihood estimation

We have a probability model of a random variable Y, and this model may condition on available covariates X. This model has parameters θ and assigns probability $p(y|x, \theta)$ to an observation.

Given a dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ of i.i.d. observations, the log-likelihood function gives us a criterion for parameter estimation

$$\mathcal{L}_{\mathcal{D}}(\theta) =$$

Maximum likelihood estimation

We have a probability model of a random variable Y, and this model may condition on available covariates X. This model has parameters θ and assigns probability $p(y|x, \theta)$ to an observation.

Given a dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ of i.i.d. observations, the log-likelihood function gives us a criterion for parameter estimation

$$\mathcal{L}_{\mathcal{D}}(\theta) = \log \prod_{s=1}^{N} p(y^{(s)}|x^{(s)}, \theta) =$$

Maximum likelihood estimation

We have a probability model of a random variable Y, and this model may condition on available covariates X. This model has parameters θ and assigns probability $p(y|x,\theta)$ to an observation.

Given a dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ of i.i.d. observations, the log-likelihood function gives us a criterion for parameter estimation

$$\mathcal{L}_{\mathcal{D}}(\theta) = \log \prod_{s=1}^{N} p(y^{(s)}|x^{(s)}, \theta) = \sum_{s=1}^{N} \log p(y^{(s)}|x^{(s)}, \theta)$$

MLE via gradient-based optimisation

If the log-likelihood is **differentiable** and **tractable** then backpropagation gives us the gradient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) =$$

Differentiable

Consider the example of a Categorical likelihood:

- for a data point (x, y) the log-likelihood is $\log \mathrm{Cat}(y|f(x;\theta)) = \log f_y(x;\theta)$
  This shows that the Categorical likelihood $\mathrm{Cat}(y|f(x;\theta))$ is differentiable with respect to its parameter $f_y(x;\theta)$.
- To satisfy differentiability with respect to θ for any (x, y), we need $f(\cdot;\theta)$, to be differentiable with respect to θ in its domain (the space $\mathcal{X}$ of all covariates).

Tractable The evaluation of $f(x;\theta)$ is tractable for any $x \in \mathcal{X}$.

Beyond Think about other likelihoods (e.g., Bernoulli, Binomial, Multinomial, Poisson, Geometric, Gaussian, Exponential, Gamma), can you imagine differentiable and tractable parameterisations of the model?

MLE via gradient-based optimisation

If the log-likelihood is **differentiable** and **tractable** then backpropagation gives us the gradient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \nabla_\theta \sum_{s=1}^{N} \log p(y^{(s)}|x^{(s)}, \theta) \quad =$$

Differentiable

Consider the example of a Categorical likelihood:

- for a data point (x, y) the log-likelihood is $\log \mathrm{Cat}(y|f(x;\theta)) = \log f_y(x;\theta)$
  This shows that the Categorical likelihood $\mathrm{Cat}(y|f(x;\theta))$ is differentiable with respect to its parameter $f_y(x;\theta)$.
- To satisfy differentiability with respect to θ for any (x, y), we need $f(\cdot;\theta)$, to be differentiable with respect to θ in its domain (the space $\mathcal{X}$ of all covariates).

Tractable The evaluation of $f(x;\theta)$ is tractable for any $x \in \mathcal{X}$.

Beyond Think about other likelihoods (e.g., Bernoulli, Binomial, Multinomial, Poisson, Geometric, Gaussian, Exponential, Gamma), can you imagine differentiable and tractable parameterisations of the model?

MLE via gradient-based optimisation

If the log-likelihood is **differentiable** and **tractable** then backpropagation gives us the gradient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \nabla_\theta \sum_{s=1}^{N} \log p(y^{(s)}|x^{(s)}, \theta) \quad = \sum_{s=1}^{N} \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)$$

Differentiable

Consider the example of a Categorical likelihood:

- for a data point (x, y) the log-likelihood is $\log \mathrm{Cat}(y|f(x;\theta)) = \log f_y(x;\theta)$
  This shows that the Categorical likelihood $\mathrm{Cat}(y|f(x;\theta))$ is differentiable with respect to its parameter $f_y(x;\theta)$.
- To satisfy differentiability with respect to θ for any (x, y), we need $f(\cdot;\theta)$, to be differentiable with respect to θ in its domain (the space $\mathcal{X}$ of all covariates).

Tractable The evaluation of $f(x;\theta)$ is tractable for any $x \in \mathcal{X}$.

Beyond Think about other likelihoods (e.g., Bernoulli, Binomial, Multinomial, Poisson, Geometric, Gaussian, Exponential, Gamma), can you imagine differentiable and tractable parameterisations of the model?

MLE via gradient-based optimisation

If the log-likelihood is **differentiable** and **tractable** then backpropagation gives us the gradient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \nabla_\theta \sum_{s=1}^{N} \log p(y^{(s)}|x^{(s)}, \theta) \quad = \sum_{s=1}^{N} \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)$$

and we can update θ in the direction

$$\gamma \nabla_\theta \mathcal{L}_\mathcal{D}(\theta)$$

to attain a local maximum of the likelihood function

Differentiable

Consider the example of a Categorical likelihood:

- for a data point (x, y) the log-likelihood is $\log \text{Cat}(y|f(x;\theta)) = \log f_y(x;\theta)$
  This shows that the Categorical likelihood $\text{Cat}(y|f(x;\theta))$ is differentiable with respect to its parameter $f_y(x;\theta)$.
- To satisfy differentiability with respect to θ for any (x, y), we need $f(\cdot;\theta)$, to be differentiable with respect to θ in its domain (the space $\mathcal{X}$ of all covariates).

Tractable The evaluation of $f(x;\theta)$ is tractable for any $x \in \mathcal{X}$.

Beyond Think about other likelihoods (e.g., Bernoulli, Binomial, Multinomial, Poisson, Geometric, Gaussian, Exponential, Gamma), can you imagine differentiable and tractable parameterisations of the model?

Big Data

For large N, computing the gradient is inconvenient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\sum_{s=1}^{N} \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)}_{\text{too many terms}}$$

We are looking for a principled way to approximate the exact gradient. Being principled here means enjoying some guarantees (this usually requires satisfying certain properties, as we shall see).

Note that we introduced the notion of a *stochastic gradient*, a random variable whose sample space is the space of gradient vectors of our model's log-likelihood function.

We have expressed the exact gradient as the expected value of that random variable. Can you see how we are going to estimate it with a computation that does not depend on N?

Big Data

For large N, computing the gradient is inconvenient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\sum_{s=1}^{N} \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)}_{\text{too many terms}}$$

$$= \sum_{s=1}^{N} \frac{1}{N} N \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)$$

We are looking for a principled way to approximate the exact gradient. Being principled here means enjoying some guarantees (this usually requires satisfying certain properties, as we shall see).

Note that we introduced the notion of a *stochastic gradient*, a random variable whose sample space is the space of gradient vectors of our model's log-likelihood function.

We have expressed the exact gradient as the expected value of that random variable. Can you see how we are going to estimate it with a computation that does not depend on N?

Big Data

For large N, computing the gradient is inconvenient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\sum_{s=1}^{N} \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)}_{\text{too many terms}}$$

$$= \sum_{s=1}^{N} \frac{1}{N} N \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)$$

$$= \sum_{s=1}^{N} \mathcal{U}(s|1/N) N \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)$$

We are looking for a principled way to approximate the exact gradient. Being principled here means enjoying some guarantees (this usually requires satisfying certain properties, as we shall see).

Note that we introduced the notion of a *stochastic gradient*, a random variable whose sample space is the space of gradient vectors of our model's log-likelihood function.

We have expressed the exact gradient as the expected value of that random variable. Can you see how we are going to estimate it with a computation that does not depend on N?

Big Data

For large N, computing the gradient is inconvenient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\sum_{s=1}^{N} \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)}_{\text{too many terms}}$$

$$= \sum_{s=1}^{N} \frac{1}{N} N \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)$$

$$= \sum_{s=1}^{N} \mathcal{U}(s|1/N) N \nabla_\theta \log p(y^{(s)}|x^{(s)}, \theta)$$

$$= \mathbb{E}_{S\sim\mathcal{U}(1/N)}\left[N \nabla_\theta \log p(y^{(S)}|x^{(S)}, \theta)\right]$$

S selects data points uniformly at random

We are looking for a principled way to approximate the exact gradient. Being principled here means enjoying some guarantees (this usually requires satisfying certain properties, as we shall see).

Note that we introduced the notion of a *stochastic gradient*, a random variable whose sample space is the space of gradient vectors of our model's log-likelihood function.

We have expressed the exact gradient as the expected value of that random variable. Can you see how we are going to estimate it with a computation that does not depend on N?

Stochastic optimisation

For large N, we can use a gradient estimate

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\mathbb{E}_{S\sim\mathcal{U}(1/N)}\left[N\nabla_\theta \log p(y^{(S)}|x^{(S)},\theta)\right]}_{\text{expected gradient :)}}$$

The theory of stochastic optimisation (**?**) tells us that we will converge to a local optimum of the objective as long as we take steps that are correct *on average*. This means we can optimisation with stochastic gradient estimates, for as long as they are unbiased estimates of the exact gradient.

Do you see the guarantee and the condition?

There are more conditions, however. The learning rate must comply with some key properties. Luckily many learning rate schedules have been documented in the literature, and most our famous optimisers meet the Robbis and Monro conditions (though not all).

If you want to read more, but need something more accessible than the 1951 paper, check (**?**).

Stochastic optimisation

For large N, we can use a gradient estimate

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\mathbb{E}_{S\sim\mathcal{U}(1/N)}\left[N\nabla_\theta \log p(y^{(S)}|x^{(S)},\theta)\right]}_{\text{expected gradient :)}}$$

$$\overset{\text{MC}}{\approx} \frac{1}{M}\sum_{m=1}^{M} N\nabla_\theta \log p(y^{(s_m)}|x^{(s_m)},\theta)$$

$$S_m \sim \mathcal{U}(1/N)$$

The theory of stochastic optimisation (**?**) tells us that we will converge to a local optimum of the objective as long as we take steps that are correct *on average*. This means we can optimisation with stochastic gradient estimates, for as long as they are unbiased estimates of the exact gradient.

Do you see the guarantee and the condition?

There are more conditions, however. The learning rate must comply with some key properties. Luckily many learning rate schedules have been documented in the literature, and most our famous optimisers meet the Robbis and Monro conditions (though not all).

If you want to read more, but need something more accessible than the 1951 paper, check (**?**).

Stochastic optimisation

For large N, we can use a gradient estimate

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\mathbb{E}_{S\sim\mathcal{U}(1/N)}\left[N\nabla_\theta \log p(y^{(S)}|x^{(S)},\theta)\right]}_{\text{expected gradient :)}}$$

$$\overset{\text{MC}}{\approx} \frac{1}{M}\sum_{m=1}^{M} N\nabla_\theta \log p(y^{(s_m)}|x^{(s_m)},\theta)$$

$$S_m \sim \mathcal{U}(1/N)$$

and take a step in the direction

$$\gamma\frac{N}{M}\underbrace{\nabla_\theta\mathcal{L}_\mathcal{B}(\theta)}_{\text{stochastic gradient}}$$

where $\mathcal{B} = \{(x^{(s_1)}, y^{(s_1)}), \ldots, (x^{(s_M)}, y^{(s_M)})\}$ is a random mini-batch

The theory of stochastic optimisation (**?**) tells us that we will converge to a local optimum of the objective as long as we take steps that are correct *on average*. This means we can optimisation with stochastic gradient estimates, for as long as they are unbiased estimates of the exact gradient.

Do you see the guarantee and the condition?

There are more conditions, however. The learning rate must comply with some key properties. Luckily many learning rate schedules have been documented in the literature, and most our famous optimisers meet the Robbis and Monro conditions (though not all).

If you want to read more, but need something more accessible than the 1951 paper, check (**?**).

DL in NLP recipe

Maximum likelihood estimation

- tells you which loss to optimise
  (i.e. negative log-likelihood)

Automatic differentiation (*backprop*)

- "give me a tractable forward pass and I will give you gradients"

Stochastic optimisation powered by backprop

- general purpose gradient-based optimisers

How about binary cross entropy? Or Categorical cross-entropy? Or MSE?

Those are all (very) closely-related to the negative log-likelihood of a probability model under a certain choice of output distribution. There is no need to memorise any such cross entropy, you can derive the correct expression from basic principles. It's also easier to talk about and think of it in terms of log-likelihood, as a 'cross-entropy' requires a non-trivial understanding of the data in terms of *observed distributions* (rather than observed outcomes).

Constraints

Differentiability

- intermediate representations must be continuous
- activations must be differentiable

Tractability

- the likelihood function must be evaluated exactly, thus it's required to be tractable

Next class

Deep latent variable models with discrete latent variables

- Exact inference
- Approximate inference

Outline

Probability versus simulation: what came first?

Sometimes a model has a mechanism to generate random draws and this mechanism could be used–in principle–to compute probability values, but the computation is intractable. These are called **implicit** models or *simulators*.

The alternative to an implicit model is a **prescribed** model. In this case the model has an explicit mechanism to assess the probability value of a given outcome. This means that–in principle–the model can also be used to generate random draws, but sometimes this requires intractable computations.

An example of the former is a GAN, an example of the latter is a Boltzmann machine.

Inputs and outputs

Watch out! What is to be considered an input or an output depends on who is doing the processing.

Point of view	Input	Output
Statistician	domain knowledge data	model specification
Parameter estimation	model specification observed data	parameters
Statistical model	data	probability distribution
Decision maker	model specification parameters decision rule novel data	decision

The statistician designs a model

Data can be used to estimate free parameters

The model predicts a distribution

Decisions are made based on the output of the model (a distribution).

Besides, NNs have inputs, they are encodings of variables that undergo transformation (sometimes these inputs were not available from some dataset of observations, they are outputs from the model via some decision rule).

NNs have outputs, they can be thought of as alternative views of data (encodings) or statistical parameters of distributions. But they are not model outputs, nor outputs from the point of view of a specific application (end user).

Latent or Hidden?

For us, and for enough of the community out there, a **latent** variable is an **unobserved random variable**. The two terms are equivalent.

The word **hidden** is more overloaded and we will simply avoid it (except perhaps when talking about NN architecture design)

- The *hidden state* of the classic Hidden Markov Model is a latent variable
- A *hidden unit* in an NN is seldom a latent variable. In fact most hidden units are not at all unknown: it's just that it takes a forward pass through the network for one to be able to 'see' (or get to know) them.

In doubt, when talking about a latent variable you can simply emphasise stochasticity by saying 'unobserved random variable' or 'latent random variable' (to some the latter will sound repetitive, but not enough to sound strange).

Supervised, unsupervised, semi-, self-, . . .

Oh this is a difficult one! Just too many views, many reasonably logical.

I like to think of it as learning in the presence or absence of latent variables. If we have unobserved random variables that stay unobserved throughout, that's unsupervised learning. If a subset of my unobserved variables become available as observations, that's semi-supervise. Otherwise it's supervised.

But remember word2vec? Its inventor wanted to learn word embeddings (sounds unsupervised, right?), but these are parameters of a binary classifier involving only observed random variables. How about 'self-supervised'?

I could confuse you for the rest of your lives if I were to get into all proposals.

My advice: stay away from the debate, just be clear about what you do. And if you can, stay coherent without being too confrontational.

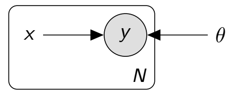
If I were to justify my point of view I would say it is model-centred (as opposed to application-centred): that means I do not take the intended use of the model into account in order to assign one such label. I only consider the presence or absence of unobserved random variables.

In my view then something like word2vec, or anything by today's standard 'self-supervised', is supervised learning. That is, learning in the complete absence of latent variables.

You can think of these terms from the point of view of the nature of the random variables (observed, unobserved), or from the point of view of the purpose of the model, or from the point of view of the type of distribution, each view leads to a different way to assign the labels. And there are more, some will say that transfer learning techniques lead to semi-supervised models.

On self-supervised learning: note that it sounds like the learner (say the model) found its own supervision, but really we (modellers) were the ones to find a task for which a cheap-to-obtain observation leads to some representation of the data that's useful in other situations.

Multiple problems, same language



(Conditional) Density estimation

	Predictor (x)	Outcome (y)
Parsing	a sentence	its syntactic/semantic parse tree/graph
Translation	a sentence	its translation
Captioning	an image	caption in English
Entailment	a text and hypothesis	entailment relation

References I

José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with Bernoulli approximate variational inference. In *ICLR - workshop track*, 2016a.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, June 2016b. PMLR. URL http://proceedings.mlr.press/v48/gal16.html.

References II

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in neural information processing systems 29*, pages 1019–1027. Curran Associates, Inc., 2016c. URL http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-rec pdf.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

References III

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://www.aclweb.org/anthology/D15-1166.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15 (56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.