NLP1

Neural sequence modelling

Wilker Aziz
wilkeraziz.github.io
probabll.github.io

ILLC (L6.55)

w.aziz@uva.nl

Table of contents

Sequence modelling

Neural models of sequence prediction

Many NLP tasks involve conditioning on text and predicting sequences

- part-of-speech tagging [Ling et al., 2015]
- named-entity recognition [Lample et al., 2016]
- machine translation [Sutskever et al., 2014]
- text summarisation [Rush et al., 2015]
- entity retrieval [Cao et al., 2021]
- information extraction [Josifoski et al., 2022]

Neural models of sequence prediction

Many NLP tasks involve conditioning on text and predicting sequences

- part-of-speech tagging [Ling et al., 2015]
- named-entity recognition [Lample et al., 2016]
- machine translation [Sutskever et al., 2014]
- text summarisation [Rush et al., 2015]
- entity retrieval [Cao et al., 2021]
- information extraction [Josifoski et al., 2022]

Deploying a system for any of these tasks requires a lot of expert knowledge (about task, datasets, design decisions, etc.), but most solutions employ a similar backbone: a neural model of sequence prediction.

Sequence-to-sequence

We are interested in modelling a specific relationship between pairs of sequences:

- an input sequence x from an input space $\mathcal{X}$
- an output sequence y from an output space $\mathcal{Y}$

We will assume this relationship can be modelled directionally $(x \to y)$ in a non-deterministic way.[1]

[1] **Notation** capital letters for random variables (e.g., Y), lowercase letters for their assignments (e.g., y), calligraphic letters for sample spaces (e.g., $\mathcal{Y}$). We use Y_j to denote a step in a random sequence and $Y_{<j}$ to denote a prefix sequence (up until but not including the Y_j). P_Y is the distribution of Y, $P_{Y|X=x}$ is the distribution of Y given $X = x$. $P(Y = y|X = x)$ is the probability of observing $Y = y$ given $X = x$.

Probabilistic modelling

We will treat y as an observation for a random variable (rv) Y, which we draw conditionally given an observation x for the rv X.

The probability $P(Y = y|X = x)$ with which we observe $Y = y$ conditioned on $X = x$ is given by a parametric function with parameters θ:

$$P(Y = y|X = x) = f(x, y; \theta) \tag{1}$$

Our first job, as modellers, is to design this probability mass function (pmf). Once it is in place, we will discuss how to estimate parameters for it, and, finally, how to use it to make predictions.

Challenges

Designing a pmf involves

1. specifying the parametric family
2. picking a value for the parameter(s)

Let's concentrate on (1), assuming that we will be employing a form of gradient-based optimisation for (2).

Parameterisation

Conditional probability distributions (cpds) for structures

Given any $x \in X$, we want to be able to parameterise a distribution over outcomes of Y. There are 2 key challenges here:

- the input space $\mathcal{X}$ is very large (typically infinite)
- the output space $\mathcal{Y}$ is very large (either infinite or it grows combinatorially with the size of input x)

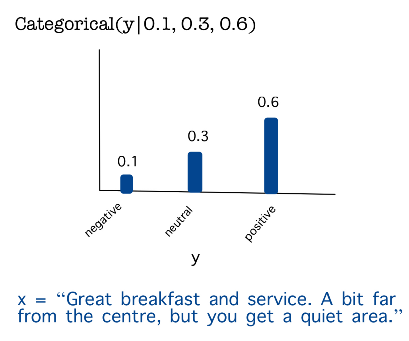
Structured $\mathcal{X}$

Pretend for a moment that $\mathcal{Y} = \{1, \ldots, C\}$. To prescribe a cpd for $Y|X = x$, we need C probabilities for any given $x \in \mathcal{X}$.

Structured $\mathcal{X}$

Pretend for a moment that $\mathcal{Y} = \{1, \ldots, C\}$. To prescribe a cpd for $Y|X = x$, we need C probabilities for any given $x \in \mathcal{X}$.

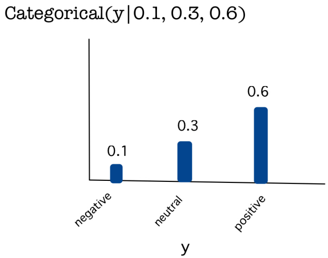
For a single $x \in \mathcal{X}$, this is not so difficult (we could store C probability values):



Structured $\mathcal{X}$

Pretend for a moment that $\mathcal{Y} = \{1, \ldots, C\}$. To prescribe a cpd for $Y|X = x$, we need C probabilities for any given $x \in \mathcal{X}$.

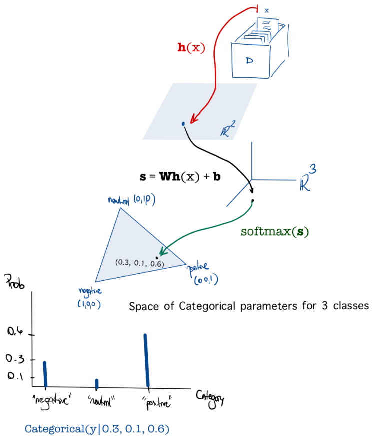
For a single $x \in \mathcal{X}$, this is not so difficult (we could store C probability values):



x = "Great breakfast and service. A bit far from the centre, but you get a quiet area."

But doing so for each and every possible $x \in \mathcal{X}$, including those we've never seen, requires a bit more ingenuity.

Log-linear cpds



- map x to a fixed number of features $\mathbf{h}(x) \in \mathbb{R}^D$
- map $\mathbf{h}(x)$ to C scores (a.k.a. logits), for example, linearly: $\mathbf{W}\mathbf{h}(x) + \mathbf{b}$
- constrain the outputs to the probability simplex

This will map *any* x that we can 'featurise' to a Categorical pmf.
Crucially, no matter how large $\mathcal{X}$ is, it only takes $D \times D + D$ parameters.

Encoding functions

The ability to 'encode' an arbitrary x into a D-dimensional space is essential for our parameterisation.

Pre-2010 these functions were handmade feature functions.

Nowadays they are part of the parameterisation. That is, we use NNs to represent the input and map it to output probability values.

A neural text classifier

Statistical model let the function $\mathbf{g}$ map from an input x to output distribution (a Categorical distribution over C classes):

$$Y|X = x \sim \mathrm{Cat}(\mathbf{g}(x; \theta)) \tag{2}$$

A neural text classifier

Statistical model let the function $\mathbf{g}$ map from an input x to output distribution (a Categorical distribution over C classes):

$$Y|X = x \sim \mathrm{Cat}(\mathbf{g}(x;\theta)) \tag{2}$$

Encoder-decoder suppose $I = |x|$

$$\begin{aligned}
\mathbf{e}_i &= \mathrm{embed}_D(x_i;\theta_{\text{inp}}) & i = 1,\ldots,I \\
\mathbf{h}_{1:I} &= \mathrm{LSTM}_H(\mathbf{e}_{1:I};\theta_{\text{enc}}) \\
\mathbf{s} &= \mathrm{linear}_C(\mathbf{h}_I;\theta_{\text{out}}) \\
\mathbf{g}(x;\theta) &= \mathrm{softmax}(\mathbf{s})
\end{aligned}$$

the parameters θ include the embedding matrix, the LSTM parameters, as well as the final linear transformation.

Conditional probability distributions (cpds) for structures

Given any $x \in X$, we want to be able to parameterise a distribution over outcomes of Y. There are 2 key challenges here:

- ~~the input space $\mathcal{X}$ is very large (typically infinite)~~
- the output space $\mathcal{Y}$ is very large (either infinite or it grows with size of input x)

Structured $\mathcal{Y}$

We exploit a decomposition into parts, where each part is drawn from a 'small' sample space.

Structured $\mathcal{Y}$

We exploit a decomposition into parts, where each part is drawn from a 'small' sample space.

For example, a POS tag sequence can be decomposed into a sequence of *word categories*:

$x = \langle \mathrm{I}, \mathrm{am}, \mathrm{going}, \mathrm{home} \rangle$
$y = \langle \mathrm{PRP}, \mathrm{VBP}, \mathrm{VBG}, \mathrm{NN} \rangle$

x

Structured $\mathcal{Y}$

We exploit a decomposition into parts, where each part is drawn from a 'small' sample space.

For example, a POS tag sequence can be decomposed into a sequence of *word categories*:

$x = \langle \text{I}, \text{am}, \text{going}, \text{home} \rangle$
$y = \langle \text{PRP}, \text{VBP}, \text{VBG}, \text{NN} \rangle$

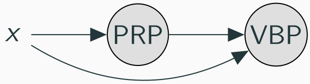
$$x \longrightarrow \boxed{\text{PRP}}$$

Structured $\mathcal{Y}$

We exploit a decomposition into parts, where each part is drawn from a 'small' sample space.

For example, a POS tag sequence can be decomposed into a sequence of *word categories*:

$$x = \langle \text{I}, \text{am}, \text{going}, \text{home} \rangle$$
$$y = \langle \text{PRP}, \text{VBP}, \text{VBG}, \text{NN} \rangle$$

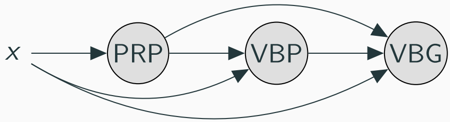


Structured $\mathcal{Y}$

We exploit a decomposition into parts, where each part is drawn from a 'small' sample space.

For example, a POS tag sequence can be decomposed into a sequence of *word categories*:

$x = \langle \text{I}, \text{am}, \text{going}, \text{home} \rangle$
$y = \langle \text{PRP}, \text{VBP}, \text{VBG}, \text{NN} \rangle$

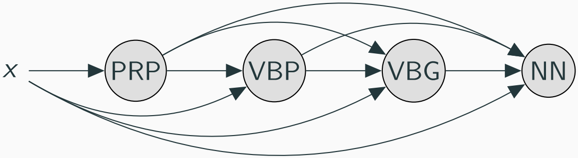


Structured $\mathcal{Y}$

We exploit a decomposition into parts, where each part is drawn from a 'small' sample space.

For example, a POS tag sequence can be decomposed into a sequence of *word categories*:

$$x = \langle \text{I}, \text{am}, \text{going}, \text{home} \rangle$$
$$y = \langle \text{PRP}, \text{VBP}, \text{VBG}, \text{NN} \rangle$$



Another example

A translation can be decomposed into a sequence of *target words*:

$x = \langle \text{How}, \text{are}, \text{you}, \text{doing}, ? \rangle$
$y = \langle \text{Hoe}, \text{gaat}, \text{het}, ? \rangle$

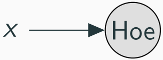
x

Another example

A translation can be decomposed into a sequence of *target words*:

$x = \langle \text{How}, \text{are}, \text{you}, \text{doing}, ? \rangle$

$y = \langle \text{Hoe}, \text{gaat}, \text{het}, ? \rangle$

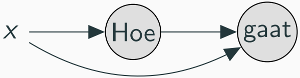


Another example

A translation can be decomposed into a sequence of *target words*:

$x = \langle \text{How}, \text{are}, \text{you}, \text{doing}, ? \rangle$

$y = \langle \text{Hoe}, \text{gaat}, \text{het}, ? \rangle$

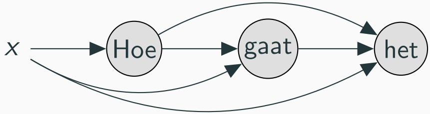


Another example

A translation can be decomposed into a sequence of *target words*:

$x = \langle \text{How}, \text{are}, \text{you}, \text{doing}, ? \rangle$

$y = \langle \text{Hoe}, \text{gaat}, \text{het}, ? \rangle$

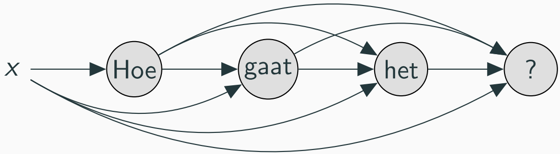


Another example

A translation can be decomposed into a sequence of *target words*:

$$x = \langle \text{How}, \text{are}, \text{you}, \text{doing}, ? \rangle$$

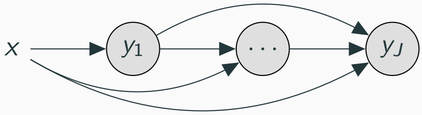
$$y = \langle \text{Hoe}, \text{gaat}, \text{het}, ? \rangle$$



In general

For an input-output pair:

$$x = \langle x_1, \ldots, x_I \rangle$$
$$y = \langle y_1, \ldots, y_J \rangle$$



$$P(Y = y|X = x) = \prod_{j=1}^{J} P(Y_j = y_j|\underbrace{X = x, Y_{<j} = y_{<j}}_{\text{parents of } j\text{th rv}}) \quad (3)$$

In the decomposition, conditioned on increasingly complex context, each part is drawn from a small sample space.

One C-way classifier, J steps

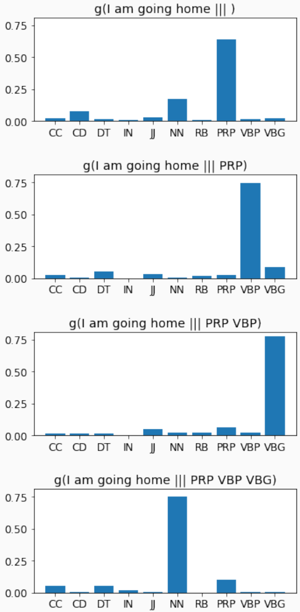
A general model of sequence prediction is in fact obtained by repeated application of a shared text classifier:

$$Y_j | X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{4}$$

Here, $\mathbf{g}$ maps from an input x and an partial output $y_{<j}$ to the probabilities of the possible outcomes for the jth step. Typically, the sample space across steps (e.g., all tags, all words).

POS tagging example

Same NN $\mathbf{g}(\cdot;\theta)$ reused over and over (as many times as there are steps in the input sequence), each time mapping from a growing context (x and $y_{<j}$) to a probability distribution over the same categorical space (i.e., space of tags).



A neural tagger

Statistical model let the function $\mathbf{g}$ map from an input x and prefix $y_{<j}$ to a distribution over C tags:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{5}$$

A neural tagger

Statistical model let the function $\mathbf{g}$ map from an input x and prefix $y_{<j}$ to a distribution over C tags:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{5}$$

Encoder-decoder $I = |x|$

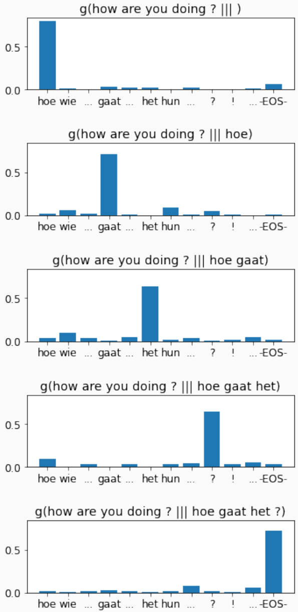
$$\begin{aligned}
\mathbf{e}_i &= \mathrm{embed}_D(x_i; \theta_{\text{words}}) && i = 1, \ldots, I \\
\mathbf{c}_{1:I} &= \mathrm{BiLSTM}_H(\mathbf{e}_{1:I}; \theta_{\text{enc}}) \\
\mathbf{t}_{j-1} &= \mathrm{embed}_D(y_{j-1}; \theta_{\text{tags}}) && j = 1, \ldots, I \\
\mathbf{h}_j &= \mathrm{rnnstep}_H(\mathbf{h}_{j-1}, [\mathbf{c}_j, \mathbf{t}_{j-1}]; \theta_{\text{dec}}) \\
\mathbf{s}_j &= \mathrm{linear}_C(\mathbf{h}_j; \theta_{\text{out}}) \\
\mathbf{g}(x, y_{<j}; \theta) &= \mathrm{softmax}(\mathbf{s}_j)
\end{aligned}$$

the parameters θ include the embedding matrices (words and tags), the parameters of the BiLSTM encoder and the LSTM decoder, as well as the final linear transformation.

Translation example

Same NN $\mathbf{g}(\cdot;\theta)$ reused over and over, each time mapping from a growing context (x and $y_{<j}$) to a probability distribution over the same categorical space (i.e., space of words).

In translation the output length is not determined by the length of the input, instead we repeat this process until a special terminating symbol is observed or generated.



A neural translation model

Statistical model let the function $\mathbf{g}$ map from an input x and prefix $y_{<j}$ to a distribution over V words:

$$Y_j | X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{6}$$

A neural translation model

Statistical model let the function $\mathbf{g}$ map from an input x and prefix $y_{<j}$ to a distribution over V words:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{6}$$

Encoder-decoder $I = |x|$ and $J = |x|$

$$\begin{aligned}
\mathbf{e}_i &= \mathrm{embed}_D(x_i; \theta_{\text{src}}) && i = 1, \ldots, I \\
\mathbf{h}_{1:I} &= \mathrm{BiLSTM}_H(\mathbf{e}_{1:I}; \theta_{\text{enc}}) \\
\mathbf{c}_j &= \mathrm{attention}(\mathbf{h}_{1:I}, \mathbf{t}_{j-1}; \theta_{\text{att}}) && j = 1, \ldots, J \\
\mathbf{w}_{j-1} &= \mathrm{embed}_D(y_{j-1}; \theta_{\text{tgt}}) \\
\mathbf{t}_j &= \mathrm{rnnstep}_H(\mathbf{t}_{j-1}, [\mathbf{c}_j, \mathbf{w}_{j-1}]; \theta_{\text{dec}}) \\
\mathbf{s}_j &= \mathrm{linear}_V(\mathbf{t}_j; \theta_{\text{out}}) \\
\mathbf{g}(x, y_{<j}; \theta) &= \mathrm{softmax}(\mathbf{s}_j)
\end{aligned}$$

the parameters θ include the embedding matrices, the parameters of the encoder-decoder with attention, as well as the final linear transformation.

Parameter estimation

Data and Task

Data a collection of pairs (x, y) where both x and y can be treated as a sequence of outcomes from small discrete sets.

Statistical task observe x and predict a conditional distribution over all possible sequences.

NLP task map a sequence x to a sequence y: for example via $\arg\max_{y \in \mathcal{Y}} \; P(Y = y | X = x)$.

Formalisation of statistical task

Statistical model let the function $\mathbf{g}$ map from x to a chain rule factorisation of the conditional distribution $Y|X = x$:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{7}$$

θ collectively refers to all trainable parameters in the model.

Formalisation of statistical task

Statistical model let the function $\mathbf{g}$ map from x to a chain rule factorisation of the conditional distribution $Y|X = x$:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{7}$$

θ collectively refers to all trainable parameters in the model.

Statistical objective maximum likelihood of model given a dataset of observations $\mathcal{D}$:

$$\mathcal{L}(\theta|\mathcal{D}) = \sum_{(x,y)\in\mathcal{D}} \log \underbrace{P(Y = y|X = x)}_{f(x,y;\theta)}$$

Formalisation of statistical task

Statistical model let the function $\mathbf{g}$ map from x to a chain rule factorisation of the conditional distribution $Y|X = x$:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{7}$$

θ collectively refers to all trainable parameters in the model.

Statistical objective maximum likelihood of model given a dataset of observations $\mathcal{D}$:

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{D}) &= \sum_{(x,y)\in\mathcal{D}} \log \underbrace{P(Y = y|X = x)}_{f(x,y;\theta)} \\ &= \sum_{(x,y)\in\mathcal{D}} \sum_{j=1}^{|y|} \log g_{y_j}(x, y_{<j}; \theta)\end{aligned} \tag{8}$$

Formalisation of statistical task

Statistical model let the function $\mathbf{g}$ map from x to a chain rule factorisation of the conditional distribution $Y|X = x$:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \mathrm{Cat}(\mathbf{g}(x, y_{<j}; \theta)) \tag{7}$$

θ collectively refers to all trainable parameters in the model.

Statistical objective maximum likelihood of model given a dataset of observations $\mathcal{D}$:

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{D}) &= \sum_{(x,y)\in\mathcal{D}} \log \underbrace{P(Y = y|X = x)}_{f(x,y;\theta)} \\ &= \sum_{(x,y)\in\mathcal{D}} \sum_{j=1}^{|y|} \log g_{y_j}(x, y_{<j}; \theta)\end{aligned} \tag{8}$$

Algorithm For concave $\mathcal{L}$ (or convex negative log-likelihood), find θ such that $\nabla_\theta \mathcal{L}(\theta|\mathcal{D}) = \mathbf{0}$.

Algorithm Solve the equation $\nabla_\theta \mathcal{L}(\theta|\mathcal{D}) = \mathbf{0}$ for θ. There is no closed form solution. But an optimum can be found via a fixed-point iteration: $\theta \leftarrow \theta + \gamma \nabla_\theta \mathcal{L}(\theta|\mathcal{D})$ for $\gamma > 0$.

Gradient-based optimisation

Let's unpack $\theta \leftarrow \theta + \gamma \nabla_\theta \mathcal{L}(\theta|\mathcal{D})$:

- $\mathbf{W} \leftarrow \mathbf{W} + \gamma \nabla_{\mathbf{W}} \mathcal{L}(\theta|\mathcal{D})$
- $\mathbf{b} \leftarrow \mathbf{b} + \gamma \nabla_{\mathbf{b}} \mathcal{L}(\theta|\mathcal{D})$
- and so on for every parameter in the model

Let's unpack it more

- $w_{k,d} \leftarrow w_{k,d} + \gamma \frac{\partial}{\partial w_{k,d}} \mathcal{L}(\theta|\mathcal{D})$
- $b_k \leftarrow b_k + \gamma \frac{\partial}{\partial b_k} \mathcal{L}(\theta|\mathcal{D})$

How do we obtain the partial derivatives (the coordinates of the gradient) we need?

Gradient-based optimisation

Let's unpack $\theta \leftarrow \theta + \gamma \nabla_\theta \mathcal{L}(\theta|\mathcal{D})$:

- $\mathbf{W} \leftarrow \mathbf{W} + \gamma \nabla_{\mathbf{W}} \mathcal{L}(\theta|\mathcal{D})$
- $\mathbf{b} \leftarrow \mathbf{b} + \gamma \nabla_{\mathbf{b}} \mathcal{L}(\theta|\mathcal{D})$
- and so on for every parameter in the model

Let's unpack it more

- $w_{k,d} \leftarrow w_{k,d} + \gamma \frac{\partial}{\partial w_{k,d}} \mathcal{L}(\theta|\mathcal{D})$
- $b_k \leftarrow b_k + \gamma \frac{\partial}{\partial b_k} \mathcal{L}(\theta|\mathcal{D})$

How do we obtain the partial derivatives (the coordinates of the gradient) we need? By differential calculus and with the help of high quality automatic differentiation software.

What if my dataset is massive?

The time and memory necessary to compute $\nabla_\theta \mathcal{L}(\theta|\mathcal{D})$ grows linearly with the size of the data.

What if my dataset is massive?

The time and memory necessary to compute $\boldsymbol{\nabla}_\theta \mathcal{L}(\theta|\mathcal{D})$ grows linearly with the size of the data.

Luckily, stochastic optimisation will converge in finite time even with gradient estimates, as long as they are unbiased, and as long as we use careful learning rate schedules [Robbins and Monro, 1951, Bottou and Cun, 2004].

What if my dataset is massive?

The time and memory necessary to compute $\nabla_\theta \mathcal{L}(\theta|\mathcal{D})$ grows linearly with the size of the data.

Luckily, stochastic optimisation will converge in finite time even with gradient estimates, as long as they are unbiased, and as long as we use careful learning rate schedules [Robbins and Monro, 1951, Bottou and Cun, 2004].

If $\mathcal{B}$ is a random subset ('mini batch') of $\mathcal{D}$, it holds that:

$$\nabla_\theta \mathcal{L}(\theta|\mathcal{D}) = \mathbb{E}_{\mathcal{B}\sim\mathcal{D}}[\nabla_\theta \mathcal{L}(\theta|\mathcal{B})] \tag{9}$$

Thus we can take iterative steps, each based on a small random subset of the data.

A soup of names

You probably heard of the *cross entropy loss*, which is identical to the *negative* of the quantity in the previous slide.

Some people will also call it the categorical cross entropy loss, or the softmax loss.

‘Softmax loss’ is a bit odd, softmax is a vector-valued function, it’s hard to imagine it as a loss.

Categorical cross entropy is clear, cross entropy can be clear enough in context.

Predictions

Making decisions

Our final job, as modellers, is to find a reasonable way to form predictions.

That is, given an input x, our model outputs a representation of an entire probability distribution $P_{Y|X=x}$ (i.e., over all of $\mathcal{Y}$).

We are now confronted with the task to map from $P_{Y|X=x}$ to a single output y. This is often formulated as a search, or discrete optimisation, problem.

Most probable output

A common algorithm for making decisions is to search for the candidate output c which is assigned highest probability:

$$y^\star = \arg\max_{c \in \mathcal{Y}} \; P(Y = c | X = x) \tag{10}$$

This can also be done in log space:
$\arg\max_{c \in \mathcal{Y}} \; \log P(Y = c | X = x)$.

Most probable output

A common algorithm for making decisions is to search for the candidate output c which is assigned highest probability:

$$y^\star = \arg\max_{c \in \mathcal{Y}} \; P(Y = c | X = x) \tag{10}$$

This can also be done in log space:
$\arg\max_{c \in \mathcal{Y}} \; \log P(Y = c | X = x)$.

This is intractable for most models! Common approximations are $\tilde{y}_j = \arg\max_{w \in [V]} \; P(Y_j = w | X = x, Y_{<j} = \tilde{y}_{<j})$ and beam search.

Most 'useful' in expectation

Let $u(y, c; x)$ quantify the utility of c when y is known to be a valid output for x.

In decision theory, a rational decision maker acts by maximising expected utility under the model:

$$y^\star = \arg\max_{c \in \mathcal{Y}} \ \mathbb{E}[u(Y, c; x)] \tag{11}$$

Expected utility can be approximated via Monte Carlo (MC):

$$\mathbb{E}[u(Y, c; x)] \overset{\text{MC}}{\approx} \frac{1}{K} \sum_{k=1}^{K} u(y^{(k)}, c; x) \tag{12}$$

with $y^{(k)} \sim P_{Y|X=x}$. See [Eikema and Aziz, 2020, 2022].

Enumerating candidates

- Greedy decoding
- Beam search
- Ancestral sampling
- Top-p and top-k sampling [Holtzman et al., 2019]
- Sampling without replacement [Kool et al., 2019]

Evaluation

Statistical: does the model fit the data well?

- perplexity
- statistics of model samples

Task-driven: does the model support good decisions (in a benchmark)?

- exact-match/precision/recall/F1 for short generations (QA, entity linking, information extraction)
- string similarity (e.g., BLEU, METEOR, BEER)
- semantic similarity (e.g., COMET, BLEURT)

What next

More architectures

- CNNs [Gehring et al., 2017]
- GCNs [Bastings et al., 2017]
- Transformers [Vaswani et al., 2017]

Alternative factorisations

- CRFs [Ma and Hovy, 2016]
- non-autoregressive models [Gu et al., 2018, Ghazvininejad et al., 2019]
- latent variable models [Zhang et al., 2016, Eikema and Aziz, 2019]

Links

Some background material

- Probabilistic graphical models [Koller and Friedman, 2009] (esp, part I on representation of probability distributions).
- Decision theory [Berger, 2013].
- On the origin of softmax: see Chapter 3 of Vlad Niculae's PhD thesis.[2]

Related courses

- DL4NLP (Christof Monz): state-of-the art architectures for most major sequence-to-sequence tasks.
- DL2 (Efstratios Gavves and Wilker Aziz): check it online.

[2] Learning Deep Models with Linguistically-Inspired Structure

Puzzle

Puzzle

Consider our typical sequence-to-sequence model, that is, a neural parameterisation of a chain rule factorisation of the joint distribution of our output random sequence Y given an outcome of an input random sequence $X = x$.

Suppose the output random sequence has length J. The probability of any outcome $\langle y_1, \ldots, y_J \rangle$ is given by

$$P(Y = \langle y_1, \ldots, y_J \rangle | X = x) = \prod_{j=1}^{J} P(Y_j = y_j | X = x, Y_{<j} = y_{<j}) \quad (13)$$

Suppose we obtain an output sequence but the kth step is missing. How can we generate outcomes for Y_k given the assignments of all other variables?

It is sufficient to solve this for an example: $X = x$, and $\langle Y_1 = \text{the}, Y_3 = \text{dog}, Y_4 = \text{EOS} \rangle$, with missing Y_2.

Puzzle - Solution

Let's denote by O the set of output random variables we observe (that is, $\{Y_1, Y_3, Y_4\}$ in the example) and o their observed values (that is, $\{\text{the}, \text{dog}, \text{EOS}\}$), and by U the set of variables that are unobserved (that is $\{Y_2\}$ in the example). We want to express the probability that U takes on some value u (e.g., $Y_2 = w$ for any word w in the vocabulary, in one case w might be the word 'cute' for example) given the assignments of the observed variables O and $X = x$.

We start by application of the definition of conditional probability:

$$P(U = u | O = o, X = x) = \frac{P(O = o, U = u | X = x)}{P(O = o | X = x)} \tag{14}$$

Puzzle - Solution

Note that the numerator is exactly the joint distribution we have access to. In our example: $P(O = o, U = u|X = x) =$
$P(\underbrace{Y_1 = \text{the}, Y_3 = \text{dog}, Y_4 = \text{EOS}}_{O=o}, \underbrace{Y_2 = w}_{U=u}|X = x)$.

Note the the denominator is the marginal of the numerator, where we marginalise out all possible assignments of U. In our example, we would marginalise out all possibilities for the second token. If $\mathcal{V}$ is the entire vocabulary, we would compute
$\sum_{t \in \mathcal{V}} P(Y_1 = \text{the}, Y_2 = t, Y_3 = \text{dog}, Y_4 = \text{EOS}|X = x)$.

Suppose in general an output sequence has length J and the vocabulary has size V. What's the computational complexity of evaluating the conditional probability of an assignment of some Y_k given everything else?

Puzzle - Solution

Probabilities that condition on past context are simple because those are directly predicted by our NN (provided we have access to observations for all variables in the past). Probabilities that condition on future are much more difficult because we need to assess the joint probability for every possible assignment of the unobserved variable (in the denominator of conditional probability).

Each joint probability takes J calls to our NN $\mathbf{g}(\cdot;\theta)$ (one per token in the sequence). We have to perform this computation V times, once per possible value of Y_k. So the total computation takes time proportional to $\mathcal{O}(JV)$.

References

Jasmijn Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1957–1967, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1209. URL https://aclanthology.org/D17-1209.

James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

Léon Bottou and Yann L. Cun. Large scale online learning. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 217–224. MIT Press, 2004.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=5k8F6UU39V`.

Bryan Eikema and Wilker Aziz. Auto-encoding variational neural machine translation. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 124–141, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4315. URL `https://aclanthology.org/W19-4315`.

Bryan Eikema and Wilker Aziz. Is MAP decoding all you need? the inadequacy of the mode in neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4506–4520, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.398. URL `https://aclanthology.org/2020.coling-main.398`.

Bryan Eikema and Wilker Aziz. Sampling-based minimum Bayes risk decoding for neural machine translation. In *EMNLP*, 2022.

Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin. A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 123–135, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1012. URL https://aclanthology.org/P17-1012.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China, November 2019. Association for

Computational Linguistics. doi: 10.18653/v1/D19-1633. URL https://aclanthology.org/D19-1633.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1l8BtlCb.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Martin Josifoski, Nicola De Cao, Maxime Peyrard, Fabio Petroni, and Robert West. GenIE: Generative information extraction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4626–4643, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.342. URL https://aclanthology.org/2022.naacl-main.342.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR, 2019.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1030. URL https://aclanthology.org/N16-1030.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi:

10.18653/v1/D15-1176. URL https://aclanthology.org/D15-1176.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1101. URL https://aclanthology.org/P16-1101.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 1951. doi: 10.1214/aoms/1177729586. URL http://dx.doi.org/10.1214/aoms/1177729586.

Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1044. URL `https://aclanthology.org/D15-1044`.

Ilya Sutskever, Oriol Vinyals, and Quoc V. V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *NIPS, 2014*, pages 3104–3112. Montreal, Canada, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 6000–6010, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. Variational neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 521–530, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1050. URL https://aclanthology.org/D16-1050.