Lexical alignment: feature-rich models

Miguel Rios

April 14, 2019

Content

Alignment distribution

Position parameterisation $L^2 \times M^2$ Jump distribution [Vogel et al., 1996]

- define a jump function $\delta(a_j, j, l, m) = a_j - \left\lfloor j \frac{l}{m} \right\rfloor$

Alignment distribution

Position parameterisation $L^2 \times M^2$ Jump distribution [Vogel et al., 1996]

- define a jump function $\delta(a_j, j, l, m) = a_j - \left\lfloor j \frac{l}{m} \right\rfloor$
- $p(a_j | l, m) = \mathrm{Cat}(\Delta | \delta)$

Alignment distribution

Position parameterisation $L^2 \times M^2$ Jump distribution [Vogel et al., 1996]

- define a jump function $\delta(a_j, j, l, m) = a_j - \lfloor j\frac{l}{m} \rfloor$
- $p(a_j|l,m) = \mathrm{Cat}(\Delta|\delta)$
- Δ takes values from $-$longest to $+$longest
  where $\Delta = \langle \delta_{-L}, ..., \delta_L \rangle$ is a vector of parameters called jump probabilities

Alignment distribution

Position parameterisation $L^2 \times M^2$ Jump distribution [Vogel et al., 1996]

- define a jump function $\delta(a_j, j, l, m) = a_j - \lfloor j \frac{l}{m} \rfloor$
- $p(a_j|l, m) = \mathrm{Cat}(\Delta|\delta)$
- Δ takes values from $-$longest to $+$longest
  where $\Delta = \langle \delta_{-L}, ..., \delta_L \rangle$ is a vector of parameters called jump probabilities
- The categorical distribution is defined for jumps ranging from $-L$ to L
  The jump function defines the support of the alignment distribution

Alignment distribution

Position parameterisation $L^2 \times M^2$ Jump distribution [Vogel et al., 1996]

- define a jump function $\delta(a_j, j, l, m) = a_j - \lfloor j \frac{l}{m} \rfloor$
- $p(a_j|l, m) = \mathrm{Cat}(\Delta|\delta)$
- Δ takes values from $-$longest to $+$longest
  where $\Delta = \langle \delta_{-L}, ..., \delta_L \rangle$ is a vector of parameters called jump probabilities
- The categorical distribution is defined for jumps ranging from $-L$ to L
  The jump function defines the support of the alignment distribution
- A jump quantifies a notion of mismatch in linear order between French and English
  Leads to a very small number of parameters, $2 \times L$

IBM 2 EM

$N \leftarrow$ number of sentence pairs
$I \leftarrow$ number of iterations
$\boldsymbol{\lambda} \leftarrow$ lexical parameters
$\boldsymbol{\gamma} \leftarrow$ alignment parameters

for $i \in [1, \ldots, I]$ **do**
  E step:
  $n(\lambda_{\mathsf{e},\mathsf{f}}) \leftarrow 0 \qquad \forall (\mathsf{e}, \mathsf{f}) \in V_E \times V_F$
  $n(\gamma_x) \leftarrow 0 \qquad \forall x \in [-L, L]$
  for $s \in [1, \ldots, N]$ **do**
    for $j \in [1, \ldots, m^{(s)}]$ **do**
      for $i \in [0, \ldots, l^{(s)}]$ **do**
        $x \leftarrow \text{jump}(i, j, l^{(s)}, m^{(s)})$
        $n(\lambda_{e_i,f_j}) \leftarrow n(\lambda_{e_i,f_j}) + \frac{\lambda_{e_i,f_j} \times \gamma_x}{\sum_{k=0}^{l} \lambda_{e_k,f_j} \times \gamma_{\text{jump}(k,j,l^{(s)},m^{(s)})}}$
        $n(\gamma_x) \leftarrow n(\gamma_x) + \frac{\lambda_{e_i,f_j} \times \gamma_{\text{jump}(i,j,l,m)}}{\sum_{k=1}^{l} \lambda_{e_k,f_j} \times \gamma_{\text{jump}(k,j,l^{(s)},m^{(s)})}}$
      end for
    end for
  end for

  M step:
  $\lambda_{\mathsf{e},\mathsf{f}} \leftarrow \frac{n(\lambda_{\mathsf{e},\mathsf{f}})}{\sum_{\mathsf{f}' \in V_F} n(\lambda_{\mathsf{e},\mathsf{f}'})} \qquad \forall (\mathsf{e}, \mathsf{f}) \in V_E \times V_F$
  $\gamma_x \leftarrow \frac{n(\gamma_x)}{\sum_{x' \in [-L,L]} n(\gamma_{x'})} \qquad \forall x \in [-L, L]$
end for

EM non identifiability

IBM 1

- The mixture weights are fixed and uniform,
  EM is guaranteed to arrive at a global maximum.

EM non identifiability

IBM 1

- The mixture weights are fixed and uniform,
  EM is guaranteed to arrive at a global maximum.
- But there may be local maxima

EM non identifiability

IBM 1

- The mixture weights are fixed and uniform,
  EM is guaranteed to arrive at a global maximum.
- But there may be local maxima
- It is not strictly convex, where multiple parameter settings that achieve the same global optima

EM non identifiability

IBM 1

- The mixture weights are fixed and uniform,
  EM is guaranteed to arrive at a global maximum.
- But there may be local maxima
- It is not strictly convex, where multiple parameter settings that achieve the same global optima
- The possible MLEs the EM algorithm finds depends on the starting parameters

EM non identifiability

IBM 1

- The mixture weights are fixed and uniform,
  EM is guaranteed to arrive at a global maximum.
- But there may be local maxima
- It is not strictly convex, where multiple parameter settings that achieve the same global optima
- The possible MLEs the EM algorithm finds depends on the starting parameters
- In practice, one usually starts from uniform parameters.
  [Toutanova and Galley, 2011] show better initialisations

EM non identifiability

IBM 2

- Mixture weights are not fixed and we add several new parameters
  Given asymmetric mixture weights, most maxima are now local.

EM non identifiability

IBM 2

- Mixture weights are not fixed and we add several new parameters
  Given asymmetric mixture weights, most maxima are now local.
- Mixture weights are not uniform
  No guaranteed to be a global maximum.

EM non identifiability

IBM 2

- Mixture weights are not fixed and we add several new parameters
  Given asymmetric mixture weights, most maxima are now local.
- Mixture weights are not uniform
  No guaranteed to be a global maximum.
- Changing weights may change in the component distributions and the other way around.

EM non identifiability

IBM 2

- Mixture weights are not fixed and we add several new parameters
  Given asymmetric mixture weights, most maxima are now local.
- Mixture weights are not uniform
  No guaranteed to be a global maximum.
- Changing weights may change in the component distributions and the other way around.
- In practice, one initialises the component distributions of IBM2 (i.e. its translation parameters) with IBM1 estimates.

EM non identifiability

IBM 2

- Mixture weights are not fixed and we add several new parameters
  Given asymmetric mixture weights, most maxima are now local.
- Mixture weights are not uniform
  No guaranteed to be a global maximum.
- Changing weights may change in the component distributions and the other way around.
- In practice, one initialises the component distributions of IBM2 (i.e. its translation parameters) with IBM1 estimates.
- The alignment distributions are initialised uniformly. Notice we first have to train IBM1 before proceeding to IBM2

Content

IBM 1-2: strong assumptions

Independence assumptions

- $p(a|m,n)$ does not depend on lexical choices
  a$_1$ cute$_2$ house$_3$ $\leftrightarrow$ una$_1$ casa$_3$ bella$_2$

IBM 1-2: strong assumptions

Independence assumptions

- $p(a|m,n)$ does not depend on lexical choices
  a_1 cute_2 $\text{house}_3 \leftrightarrow \text{una}_1$ casa_3 bella_2
  a_1 cosy_2 $\text{house}_3 \leftrightarrow \text{una}_1$ casa_3 confortable_2

IBM 1-2: strong assumptions

Independence assumptions

- $p(a|m, n)$ does not depend on lexical choices
  a$_1$ cute$_2$ house$_3$ $\leftrightarrow$ una$_1$ casa$_3$ bella$_2$
  a$_1$ cosy$_2$ house$_3$ $\leftrightarrow$ una$_1$ casa$_3$ confortable$_2$
- $p(f|e)$ can only reasonably explain one-to-one alignments
  I will be leaving soon $\leftrightarrow$ voy a salir pronto

IBM 1-2: strong assumptions

Independence assumptions

- $p(a|m,n)$ does not depend on lexical choices
  a$_1$ cute$_2$ house$_3$ $\leftrightarrow$ una$_1$ casa$_3$ bella$_2$
  a$_1$ cosy$_2$ house$_3$ $\leftrightarrow$ una$_1$ casa$_3$ confortable$_2$
- $p(f|e)$ can only reasonably explain one-to-one alignments
  I will be leaving soon $\leftrightarrow$ voy a salir pronto

Parameterisation

- categorical events are unrelated
  prefixes/suffixes: normal, normally, abnormally, ...
  verb inflections: comer, comi, comia, comio, ...
  gender/number: gato, gatos, gata, gatas, ...

Conditional probability distributions

CPD: condition $c \in \mathcal{C}$, outcome $o \in \mathcal{O}$, and $\theta_c \in \mathbb{R}^{|\mathcal{O}|}$

$$p(o|c) = \mathrm{Cat}(\theta_c) \tag{1}$$

- $p(o|c) = \theta_{c,o}$

How bad is it for IBM model 1?

Conditional probability distributions

CPD: condition $c \in \mathcal{C}$, outcome $o \in \mathcal{O}$, and $\theta_c \in \mathbb{R}^{|\mathcal{O}|}$

$$p(o|c) = \text{Cat}(\theta_c) \tag{1}$$

- $p(o|c) = \theta_{c,o}$
- $0 \leq \theta_{c,o} \leq 1$

How bad is it for IBM model 1?

Conditional probability distributions

CPD: condition $c \in \mathcal{C}$, outcome $o \in \mathcal{O}$, and $\theta_c \in \mathbb{R}^{|\mathcal{O}|}$

$$p(o|c) = \text{Cat}(\theta_c) \quad (1)$$

- $p(o|c) = \theta_{c,o}$
- $0 \leq \theta_{c,o} \leq 1$
- $\sum_o \theta_{c,o} = 1$

How bad is it for IBM model 1?

Conditional probability distributions

CPD: condition $c \in \mathcal{C}$, outcome $o \in \mathcal{O}$, and $\theta_c \in \mathbb{R}^{|\mathcal{O}|}$

$$p(o|c) = \mathrm{Cat}(\theta_c) \tag{1}$$

- $p(o|c) = \theta_{c,o}$
- $0 \leq \theta_{c,o} \leq 1$
- $\sum_o \theta_{c,o} = 1$
- $O(|\rfloor| \times |\wr|)$ parameters

How bad is it for IBM model 1?

Probability tables

$$p(f|e)$$

ENGLISH ↓	FRENCH →			
	anormal	normal	normalmente	...
abnormal	0.7	0.1	0.01	...
normal	0.01	0.6	0.2	...
normally	0.001	0.25	0.65	...

- grows with size of vocabularies
- no parameter sharing

Logistic CPDs

CPD: condition $c \in \mathcal{C}$ and outcome $o \in \mathcal{O}$

$$p(o|c) = \frac{\exp(w^\top h(c, o))}{\sum_{o'} \exp(w^\top h(c, o'))} \tag{2}$$

- $w \in \mathbb{R}^d$ is a weight vector
- $h : \mathcal{C} \times \mathcal{O} \to R^d$ is a feature function
- d parameters
- computing CPD requires $O(|\rfloor| \times |\wr| \times d)$ operations

How bad is it for IBM model 1?

CPDs as functions

$$h : \mathcal{E} \times \mathcal{F} \rightarrow R^d$$

EVENTS ↓		FEATURES →				
ENGLISH	FRENCH	**normal** **normal**	*normal-* *normal-*	-normal -normal	ab- a-	-ly -mente
abnormal	anormal	0	0	1	1	0
	normal	0	0	1	0	0
	*normal*mente	0	1	0	0	0
normal	anormal	0	0	1	0	0
	normal	1	0	0	0	0
	*normal*mente	0	1	0	0	0
normally	anormal	0	0	1	0	0
	normal	0	1	0	0	0
	*normal*mente	0	1	0	0	1
WEIGHTS →		1.5	0.3	0.3	0.8	1.1

- computation still grows with size of vocabularies
- but far less parameters to estimate

Content

Log-linear models

- Log-linear models revolve around the concept of features. In short, features are basically,
  Something about the context that will be useful in predicting

Log-linear models

- Log-linear models revolve around the concept of features. In short, features are basically,
  Something about the context that will be useful in predicting
- Enhancing models with features that capture the dependencies between different morphologically inflected word forms. The standard parameterisation using categorical distributions is limited with respect to the features it can capture

Content

Berg-Kirkpatrick et al. [2010]

Lexical distribution in IBM model 1

$$p(f|e) = \frac{\exp(w_{\mathsf{lex}}^{\top} h_{\mathsf{lex}}(e, f))}{\sum_{f'} \exp(w_{\mathsf{lex}}^{\top} h_{\mathsf{lex}}(e, f'))} \tag{3}$$

Features

- $f \in V_F$ is a French word (decision), $e \in V_E$ is an English word (conditioning context), $w \in R^d$ is the parameter vector, and $h : V_F V_E \to R^d$ is a feature vector function.
- prefixes/suffixes
- character n-grams
- POS tags

Extension: lexicalised jump distribution

$$p(\delta|e) = \frac{\exp(w_{\text{dist}}^{\top} h_{\text{dist}}(e, \delta))}{\sum_{\delta'} \exp(w_{\text{dist}}^{\top} h_{\text{dist}}(e, \delta'))} \tag{4}$$

Features

- POS tags
- suffixes/prefixes
- lemma
- jump values
- m, n, j, i (values used to compute jump)

Feature name	Description
word	Whole lexical entry
prefix	Prefix of specified length
suffix	Suffix of specified length
category	Boolean: checks if lexical entry contains digit(s)

Problems with features

- We can see $e_{t-2} =$ farmers is compatible with $e_t =$ hay (in the context farmers grow hay)

Problems with features

- We can see e_{t-2} = farmers is compatible with e_t = hay (in the context farmers grow hay)
- and e_{t-1} = eat is also compatible (in the context cows eat hay).

farmers eat	steak →	**high**	cows eat	steak →	**low**
	hay →	**low**		hay →	**high**
farmers grow	steak →	**low**	cows grow	steak →	**low**
	hay →	**high**		hay →	**low**

Problems with features

- Features depend on e_{t-1}, and another set of features dependent on e_{t-2}, neither set of features can rule out the unnatural phrase farmers eat hay

Problems with features

- Features depend on e_{t-1}, and another set of features dependent on e_{t-2}, neither set of features can rule out the unnatural phrase farmers eat hay
- Combination of features greatly expands the parameters: instead of $O(|V|^2)$ parameters for each pair e_{i-1}, e_i,
  We need $O(|V|^3)$ parameters for each triplet e_{i-2}, e_{i-1}, e_i

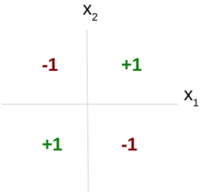
Problems with features

- Features depend on e_{t-1}, and another set of features dependent on e_{t-2}, neither set of features can rule out the unnatural phrase farmers eat hay
- Combination of features greatly expands the parameters: instead of $O(|V|^2)$ parameters for each pair e_{i-1}, e_i,
  We need $O(|V|^3)$ parameters for each triplet e_{i-2}, e_{i-1}, e_i
- Learning using these combination features, e.g. neural networks

Content

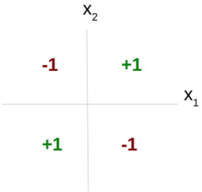
Function that cannot be solved by a linear transformation

- For example the function $x \in -1, 1$ and outputs $y = 1$ if both x_1 and x_2 are equal and $y = -1$ otherwise.



Function that cannot be solved by a linear transformation

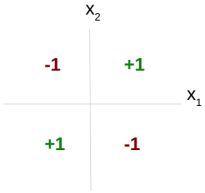
- For example the function $x \in -1, 1$ and outputs $y = 1$ if both x_1 and x_2 are equal and $y = -1$ otherwise.



- We can use a linear combination $y = Wx + b$

Function that cannot be solved by a linear transformation

- For example the function $x \in -1, 1$ and outputs $y = 1$ if both x_1 and x_2 are equal and $y = -1$ otherwise.



- We can use a linear combination $y = Wx + b$
- Or a multi-layer perceptron:

$$\begin{aligned} h &= step(W_x h_x + b_h) \\ y &= w_{hy} h + b_y. \end{aligned} \tag{5}$$

Function that cannot be solved by a linear transformation

- Computation is split into two stages:

Function that cannot be solved by a linear transformation

- Computation is split into two stages:
- Calculation of the hidden layer , which takes in input x and outputs a vector of hidden variables h

Function that cannot be solved by a linear transformation

- Computation is split into two stages:
- Calculation of the hidden layer , which takes in input x and outputs a vector of hidden variables h
- and calculation of the output layer, which takes in h and calculates the final result y.

Function that cannot be solved by a linear transformation

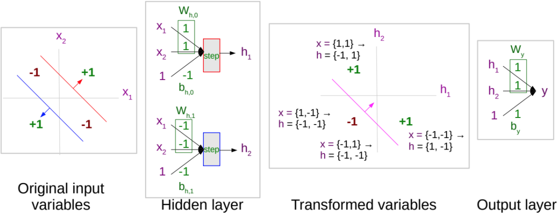
- Computation is split into two stages:
- Calculation of the hidden layer , which takes in input x and outputs a vector of hidden variables h
- and calculation of the output layer, which takes in h and calculates the final result y.
- Both layers consist of an affine transform using weights W and biases b, followed by a $step()$ function, which calculates the following:

$$step(x) = \begin{cases} 1, & \text{if } x > 0. \\ -1, & \text{otherwise.} \end{cases} \tag{6}$$

Function that cannot be solved by a linear transformation

- Computation is split into two stages:
- Calculation of the hidden layer , which takes in input x and outputs a vector of hidden variables h
- and calculation of the output layer, which takes in h and calculates the final result y.
- Both layers consist of an affine transform using weights W and biases b, followed by a $step()$ function, which calculates the following:

$$step(x) = \begin{cases} 1, & \text{if } x > 0. \\ -1, & \text{otherwise.} \end{cases} \tag{6}$$



Training Neural Networks

- We would like to train the parameters of the MLP

Training Neural Networks

- We would like to train the parameters of the MLP
- We need to define the loss function $l()$, calculate the derivative of the loss with respect to the parameters, then take a step in the direction that will reduce the loss.

Training Neural Networks

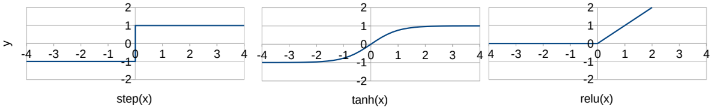
- We would like to train the parameters of the MLP
- We need to define the loss function $l()$, calculate the derivative of the loss with respect to the parameters, then take a step in the direction that will reduce the loss.
- e.g. squared-error loss, common in regression problems which measures the difference between the calculated value y and correct value y^*:
  $l(y^*, y) = (y^* - y)^2$

Training Neural Networks

- We would like to train the parameters of the MLP
- We need to define the loss function $l()$, calculate the derivative of the loss with respect to the parameters, then take a step in the direction that will reduce the loss.
- e.g. squared-error loss, common in regression problems which measures the difference between the calculated value y and correct value y^*:
  $l(y^*, y) = (y^* - y)^2$
- however the $step()$ function is not very derivative friendly

Training Neural Networks

- We would like to train the parameters of the MLP
- We need to define the loss function $l()$, calculate the derivative of the loss with respect to the parameters, then take a step in the direction that will reduce the loss.
- e.g. squared-error loss, common in regression problems which measures the difference between the calculated value y and correct value y^*:
  $l(y^*, y) = (y^* - y)^2$
- however the $step()$ function is not very derivative friendly
- We can use non-linear functions, hyperbolic tangent (tanh) function



Training Neural Networks

- We perform the full calculation of the loss function:

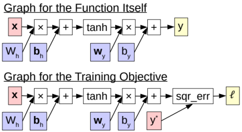
$$
\begin{aligned}
\boldsymbol{h}' &= W_{xh}\boldsymbol{x} + \boldsymbol{b}_h \\
\boldsymbol{h} &= \tanh(\boldsymbol{h}') \\
y &= \boldsymbol{w}_{hy}\boldsymbol{h} + b_y \\
\ell &= (y^* - y)^2.
\end{aligned}
$$

Training Neural Networks

- We perform the full calculation of the loss function:

$$\begin{aligned} \boldsymbol{h}' &= W_{xh}\boldsymbol{x} + \boldsymbol{b}_h \\ \boldsymbol{h} &= \tanh(\boldsymbol{h}') \\ y &= \boldsymbol{w}_{hy}\boldsymbol{h} + b_y \\ \ell &= (y^* - y)^2. \end{aligned}$$

- Computation graph:

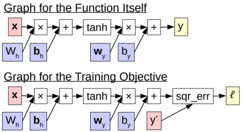


Training Neural Networks

- We perform the full calculation of the loss function:

$$\boldsymbol{h}' = W_{xh}\boldsymbol{x} + \boldsymbol{b}_h$$
$$\boldsymbol{h} = \tanh(\boldsymbol{h}')$$
$$y = \boldsymbol{w}_{hy}\boldsymbol{h} + b_y$$
$$\ell = (y^* - y)^2.$$

- Computation graph:



- We use chain rule of derivatives for each set of parameters:

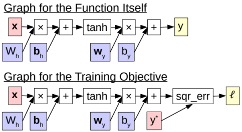
$$\frac{d\ell}{db_y} = \frac{d\ell}{dy}\frac{dy}{db_y}$$
$$\frac{d\ell}{d\boldsymbol{w}_{hy}} = \frac{d\ell}{dy}\frac{dy}{d\boldsymbol{w}_{hy}}$$
$$\frac{d\ell}{d\boldsymbol{b}_h} = \frac{d\ell}{dy}\frac{dy}{d\boldsymbol{h}}\frac{d\boldsymbol{h}}{d\boldsymbol{h}'}\frac{d\boldsymbol{h}'}{d\boldsymbol{b}_h}$$
$$\frac{d\ell}{dW_{xh}} = \frac{d\ell}{dy}\frac{dy}{d\boldsymbol{h}}\frac{d\boldsymbol{h}}{d\boldsymbol{h}'}\frac{d\boldsymbol{h}'}{dW_{xh}}.$$

Training Neural Networks

- We perform the full calculation of the loss function:

$$\begin{aligned} \boldsymbol{h}' &= W_{xh}\boldsymbol{x} + \boldsymbol{b}_h \\ \boldsymbol{h} &= \tanh(\boldsymbol{h}') \\ y &= \boldsymbol{w}_{hy}\boldsymbol{h} + b_y \\ \ell &= (y^* - y)^2. \end{aligned}$$

- Computation graph:



- We use chain rule of derivatives for each set of parameters:

$$\begin{aligned} \frac{d\ell}{db_y} &= \frac{d\ell}{dy}\frac{dy}{db_y} \\ \frac{d\ell}{d\boldsymbol{w}_{hy}} &= \frac{d\ell}{dy}\frac{dy}{d\boldsymbol{w}_{hy}} \\ \frac{d\ell}{d\boldsymbol{b}_h} &= \frac{d\ell}{dy}\frac{dy}{d\boldsymbol{h}}\frac{d\boldsymbol{h}}{d\boldsymbol{h}'}\frac{d\boldsymbol{h}'}{d\boldsymbol{b}_h} \\ \frac{d\ell}{dW_{xh}} &= \frac{d\ell}{dy}\frac{dy}{d\boldsymbol{h}}\frac{d\boldsymbol{h}}{d\boldsymbol{h}'}\frac{d\boldsymbol{h}'}{dW_{xh}}. \end{aligned}$$

Content

IBM: non-linear models

Nothing prevents us from using more expressive functions

[Kočiský et al., 2014]

- $p(o|c) = \mathrm{softmax}(f_\theta(c))$

where $f_\theta(\cdot)$ is a neural network with parameters θ

Features

IBM: non-linear models

Nothing prevents us from using more expressive functions

[Kočiský et al., 2014]

- $p(o|c) = \mathrm{softmax}(f_\theta(c))$
- $p(o|c) = \frac{\exp(f_\theta(c,o)))}{\sum_{o'} \exp(f_\theta(c,o')))}$

where $f_\theta(\cdot)$ is a neural network with parameters θ

Features

IBM: non-linear models

Nothing prevents us from using more expressive functions
[Kočiský et al., 2014]

- $p(o|c) = \text{softmax}(f_\theta(c))$
- $p(o|c) = \frac{\exp(f_\theta(c,o)))}{\sum_{o'} \exp(f_\theta(c,o')))}$

where $f_\theta(\cdot)$ is a neural network with parameters θ

Features

- induce features (word-level, char-level, n-gram level)

IBM: non-linear models

Nothing prevents us from using more expressive functions

[Kočiský et al., 2014]

- $p(o|c) = \mathrm{softmax}(f_\theta(c))$
- $p(o|c) = \frac{\exp(f_\theta(c,o)))}{\sum_{o'} \exp(f_\theta(c,o')))}$

where $f_\theta(\cdot)$ is a neural network with parameters θ

Features

- induce features (word-level, char-level, n-gram level)
- pre-trained embeddings

Neural IBM

- $f_\theta(e) = \mathrm{softmax}(W_t H_E(e) + b_t)$ note that the softmax is necessary to make t_θ produce valid parameters for the categorical distribution
  $W_t \in \mathbb{R}^{|V_F| \times d_h}$ and $b_t \in \mathbb{R}^{|V_F|}$

Neural IBM

- $h_E(e)$ is defined below with $W_{h_E} \in \mathbb{R}^{d_h \times d_e}$ and $b_{h_E} \in \mathbb{R}^{d_h}$
  $$h_E(e) = \underbrace{\tanh(\underbrace{W_{h_E} r_E(e) + b_{h_E}}_{\text{affine}})}_{\text{elementwise nonlinearity}}$$

where

Neural IBM

- $h_E(e)$ is defined below with $W_{h_E} \in \mathbb{R}^{d_h \times d_e}$ and $b_{h_E} \in \mathbb{R}^{d_h}$
  $$h_E(e) = \underbrace{\tanh(\underbrace{W_{h_E} r_E(e) + b_{h_E}}_{\text{affine}})}_{\text{elementwise nonlinearity}}$$
- $r_E(e) = W_{r_E} v_E(e)$ is a word embedding of e with $W_{r_E} \in \mathbb{R}^{d_e \times |V_E|}$

where

Neural IBM

- $h_E(e)$ is defined below with $W_{h_E} \in \mathbb{R}^{d_h \times d_e}$ and $b_{h_E} \in \mathbb{R}^{d_h}$
$$h_E(e) = \underbrace{\tanh(\underbrace{W_{h_E} r_E(e) + b_{h_E}}_{\text{affine}})}_{\text{elementwise nonlinearity}}$$
- $r_E(e) = W_{r_E} v_E(e)$ is a word embedding of e with $W_{r_E} \in \mathbb{R}^{d_e \times |V_E|}$
- $v_E(e) \in \{0, 1\}^{v_E}$ is a one-hot encoding of e, thus $\sum_i v_E(e)_i = 1$

where

Neural IBM

- $h_E(e)$ is defined below with $W_{h_E} \in \mathbb{R}^{d_h \times d_e}$ and $b_{h_E} \in \mathbb{R}^{d_h}$
  $$h_E(e) = \underbrace{\tanh(\underbrace{W_{h_E} r_E(e) + b_{h_E}}_{\text{affine}})}_{\text{elementwise nonlinearity}}$$
- $r_E(e) = W_{r_E} v_E(e)$ is a word embedding of e with $W_{r_E} \in \mathbb{R}^{d_e \times |V_E|}$
- $v_E(e) \in \{0, 1\}^{v_E}$ is a one-hot encoding of e, thus $\sum_i v_E(e)_i = 1$
- $\theta = \{W_t, b_t, W_{h_E}, b_{h_E}, W_{r_E}\}$

where

Neural IBM

- $h_E(e)$ is defined below with $W_{h_E} \in \mathbb{R}^{d_h \times d_e}$ and $b_{h_E} \in \mathbb{R}^{d_h}$
$$h_E(e) = \underbrace{\tanh(\underbrace{W_{h_E} r_E(e) + b_{h_E}}_{\text{affine}})}_{\text{elementwise nonlinearity}}$$
- $r_E(e) = W_{r_E} v_E(e)$ is a word embedding of e with $W_{r_E} \in \mathbb{R}^{d_e \times |V_E|}$
- $v_E(e) \in \{0, 1\}^{v_E}$ is a one-hot encoding of e, thus $\sum_i v_E(e)_i = 1$
- $\theta = \{W_t, b_t, W_{h_E}, b_{h_E}, W_{r_E}\}$
- Other architectures are also possible, one can use different parameterisations that may use more or less parameters. For example, with a CNN one could make this function sensitive to characters in the words, something along these lines could also be done with RNNs.

where

MLE

- We can use maximum likelihood estimation (MLE) to choose the parameters of our deterministic function f_θ.

MLE

- We can use maximum likelihood estimation (MLE) to choose the parameters of our deterministic function f_θ.
- We know at least one general (convex) optimisation algorithm, i.e. gradient ascent.
  This is a gradient-based procedure which chooses θ so that the gradient of our objective with respect to θ is zero.

MLE

- We can use maximum likelihood estimation (MLE) to choose the parameters of our deterministic function f_θ.
- We know at least one general (convex) optimisation algorithm, i.e. gradient ascent.
  This is a gradient-based procedure which chooses θ so that the gradient of our objective with respect to θ is zero.
- IBM1 would be convex with standard tabular CPDs, but FFNNs with 1 non-linear hidden layer (or more) make it non-convex.

MLE

- We can use maximum likelihood estimation (MLE) to choose the parameters of our deterministic function f_θ.
- We know at least one general (convex) optimisation algorithm, i.e. gradient ascent.
  This is a gradient-based procedure which chooses θ so that the gradient of our objective with respect to θ is zero.
- IBM1 would be convex with standard tabular CPDs, but FFNNs with 1 non-linear hidden layer (or more) make it non-convex.
- Nowadays, we have tools that can perform automatic differentiation for us.
  If our functions are differentiable, we can get gradients for them.

MLE

- We still need to be able to express the functional form of the likelihood.

MLE

- We still need to be able to express the functional form of the likelihood.
- Let us then express the log-likelihood (which is the objective we maximise in MLE) of a single sentence pair as a function of our free parameters:

$$\mathcal{L}(\theta|e_0^m, f_1^n) = \log p_\theta(f_1^m|e_0^l) \tag{7}$$

MLE

- We still need to be able to express the functional form of the likelihood.
- Let us then express the log-likelihood (which is the objective we maximise in MLE) of a single sentence pair as a function of our free parameters:

$$\mathcal{L}(\theta|e_0^m, f_1^n) = \log p_\theta(f_1^m|e_0^l) \quad (7)$$

- Note that in fact our log-likelihood is a sum of independent terms $\mathcal{L}_j(\theta|e_0^m, f_j)$, thus we can characterise the contribution of each French word in each sentence pair as

MLE

- NN toolkits implement gradient-based optimisation for us.

MLE

- NN toolkits implement gradient-based optimisation for us.
- To get a loss, we simply negate our objective.
  You will find a lot of material that mentions some categorical cross-entropy loss.

$$
\begin{aligned}
l(\theta) &= - \sum_{(e_0^m, f_1^l)} p_\star(f_1^l | e_0^m) \log p_\theta(f_1^m | e_0^l) \\
&\approx -\frac{1}{S} \log p_\theta(f_1^l | e_0^m)
\end{aligned} \tag{8}
$$

MLE

- With SGD we sample a subset $\mathcal{S}$ of the training data and compute a loss for that sample.

MLE

- With SGD we sample a subset $\mathcal{S}$ of the training data and compute a loss for that sample.
- We then use automatic differentiation to obtain a gradient $\nabla_\theta \updownarrow(\theta|\mathcal{S})$. This gradient is used to update our deterministic parameters θ.

$$\theta^{(t+1)} = \theta^{(t)} - \delta_t \nabla_{\theta^{(t)}} l(\theta^{(t)}|\mathcal{S}) \tag{9}$$

References I

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 582–590, Los Angeles, California, June 2010. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/N10-1083`.

Tomáš Kočiský, Karl Moritz Hermann, and Phil Blunsom. Learning bilingual word representations by marginalizing alignments. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 224–229, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P14-2037`.

References II

Kristina Toutanova and Michel Galley. Why initialization matters for ibm model 1: Multiple optima and non-strict convexity. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 461–466, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2*, COLING '96, pages 836–841, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. doi: 10.3115/993268.993313. URL http://dx.doi.org/10.3115/993268.993313.