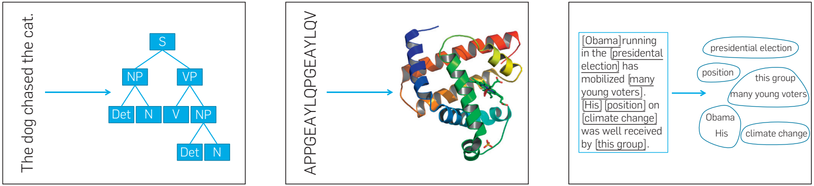
Structured Output Prediction

Andrea Passerini
andrea.passerini@unitn.it

Advanced Topics in Machine Learning and Optimization

Structured Output Prediction: the task



The task

- The input is (typically) a structured object
- The output is also a structured-object (rather than a scalar) e.g.:
  - A sequence (part-of-speech tagging, protein secondary structure prediction)
  - A tree (parse-tree prediction)
  - A graph (link detection, protein 3D structure prediction)

Image from Joachims et al, 2009

Structured Output Prediction: the issue

The issue

- Standard supervised learning learns a function

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

- However the space of candidate outputs is huge (exponential in the number of output variables, or even infinite)
- The problem cannot be formalized as multiclass classification

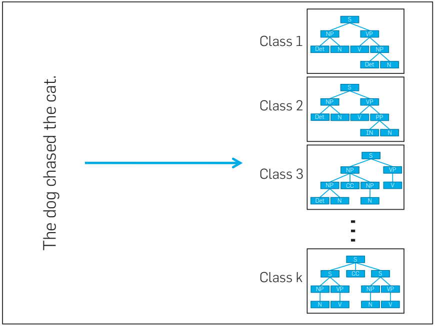
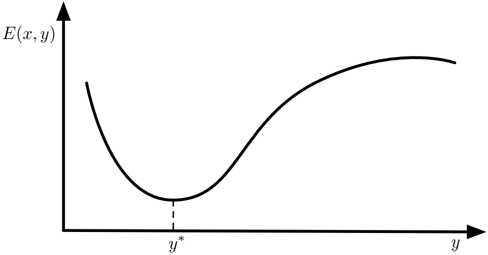


Image from Joachims et al, 2009

Structured Output Prediction: approaches

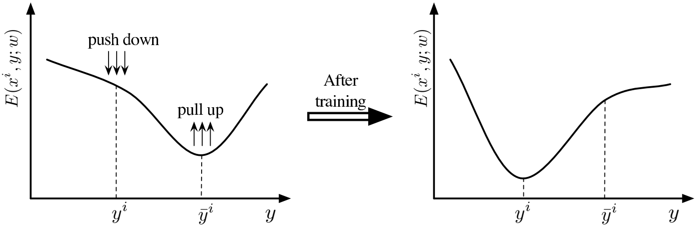


Energy-based models

$$y^* = \text{argmin}_{y \in \mathcal{Y}} E(x, y)$$

- An energy function predicts the energy of each input-output pair
- Prediction is achieved by getting minimal energy output for a given input
- Inference methods are needed to solve the argmin problem (*learning with inference*)

Energy-based models



Learning

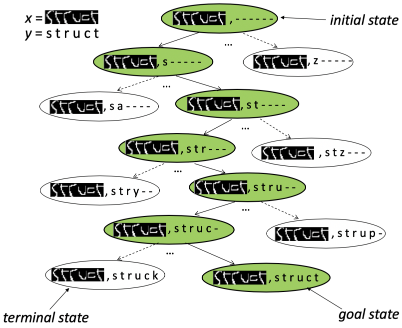
- Adjust weights of energy function to drive correct output to have minimal energy
- Based on loss functions between correct output and incorrect ones
- Typically focus on *most offending incorrect answer*:

$$\bar{y}^i = \operatorname{argmin}_{y \in \mathcal{Y}, y \neq y^i} E(x^i, y^i; w)$$

Structured Output Prediction: approaches

Search-based models

- State-space search process
- Initial state with empty output
- Heuristic function to choose next state (partial output)
- Terminal states are states with complete output
- No need for global inference algorithm (*learning for inference*)

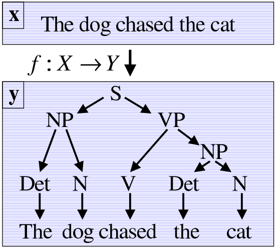


Search-based models

learning

- Adjust weights of heuristic function to have high score for correct moves given current state
- *on-trajectory* training, current state is always a correct one.
- *off-trajectory* training, current state is highest scoring state even if incorrect

Energy-based models: Structured SVM



$$\Psi(\mathbf{x},\mathbf{y}) = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \\ \vdots \\ 0 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{matrix} S \to NP\,VP \\ S \to NP \\ NP \to Det\ N \\ VP \to V\ NP \\ \\ Det \to dog \\ Det \to the \\ N \to dog \\ V \to chased \\ N \to cat \end{matrix}$$

Joint input-output feature map

$$f(x, y) = \mathbf{w}^T \Psi(x, y) = -E(x, y)$$

- Joint input-output feature map $\Psi(x, y)$
- Features capture interaction between input and output variables and between output variables among themselves
- Energy function is a linear function of the feature map
- The function can be kernelized

Structured SVM: learning

$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$$

subject to:

$$\mathbf{w}^T\Psi(x_i, y_i) - \mathbf{w}^T\Psi(x_i, y') \geq \Delta(y_i, y') - \xi_i$$
$$\forall i, y' \neq y_i$$

Max-margin formulation

- $\Delta(y_i, y')$ is the cost for predicting y' instead of y_i (structured-output loss)
- The formulation aims at separating correct predictions from incorrect predictions with a large margin
- Hard to solve directly (exponential number of constraints!!)

Structured SVM: learning

Cutting plane algorithm

1. Initialize weights and constraints $S_i = \emptyset \ \forall i$
2. While constraint added
   1. For each example i

$$\begin{aligned} \xi_i &= \max_{y' \in S_i} \Delta(y_i, y') + \mathbf{w}^T \Psi(x_i, y') - \mathbf{w}^T \Psi(x_i, y_i) \\ \xi_i^{new} &= \max_{y' \neq y_i} \Delta(y_i, y') + \mathbf{w}^T \Psi(x_i, y') - \mathbf{w}^T \Psi(x_i, y_i) \end{aligned}$$

   2. If $\xi_i^{new} - \xi > \epsilon$
   3. Add constraint and update S_i
   4. retrain

Alternatives

- Stochastic subgradient descent
- Block-coordinate Frank-Wolfe optimization

Structured SVM: inference

(Loss augmented) argmax inference

- inference at prediction time

$$y^* = \text{argmax}_{y \in \mathcal{Y}} \mathbf{w}^T \Psi(x, y)$$

- loss augmented inference at training time (most offending incorrect answer)

$$\bar{y}' = \text{argmax}_{y' \neq y_i} \Delta(y_i, y') + \mathbf{w}^T \Psi(x_i, y') - \mathbf{w}^T \Psi(x_i, y_i)$$

Approaches

- Viterbi algorithm for sequence labelling
- CYK algorithm for parse tree prediction
- Loopy belief propagation (approximate)
- Amortized inference (use previous solutions to speed up related inference tasks)

Structured SVM: PROs and CONs

PROs

- Max-margin approach
- Guarantees on number of iterations (depends on ϵ, independent on number of output structures)
- Can deal with arbitrary constrains on output structure

CONs

- Inefficient, (loss augmented) inference required at every training iteration
- The function to be learned is complex, high-order feature typically required (making inference even more expensive)

Search-based models: ordered vs unordered

Ordered search space

- Fixed ordering of decisions (e.g., left-to-right decisions in sequences)
- Classifier-based structured prediction (reduction to multi-class classification task)

Unordered search space

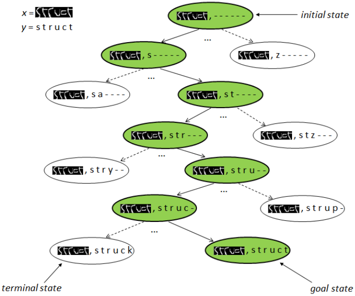
- Learner dynamically orders decisions
- Easy-first approach (make easy decisions first)

Search-based models: classifier-based

Setting

- Ordered search space
- Reduction to multi-class classification on next decision
- Training examples:
  - input is set of outputs up to position t
  - output is correct output for position $t + 1$
- *imitation learning* (training examples as expert demonstrations)

Classifier-based structured prediction: exact imitation



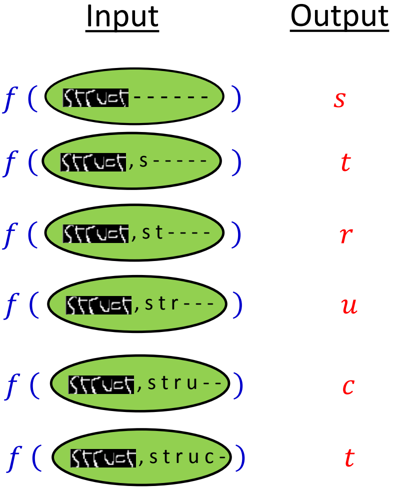
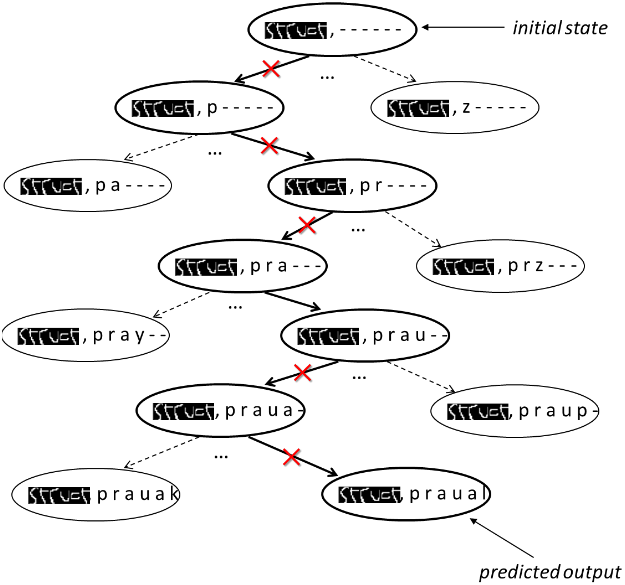


Image from Fern et al., 2016

Exact imitation problem: error propagation



Error propagation

Problem

- Errors in early decisions propagate to down-stream ones
- System is not trained to deal with decisions given incorrect states

Solution

- Generate trajectories using current policy
- Use optimal policy to generate optimal next states given states visited by current policy

DAgger (Dataset Aggregation)

The algorithm

1. Collect training set $\mathcal{D}$ of N trajectories using ground-truth policy π^*
2. Repeat
    1. $\pi \leftarrow \textsc{LearnClassifier}(\mathcal{D})$
    2. Collect set of states $\mathcal{S}$ along trajectories computed using π
    3. For each $s \in \mathcal{S}$
        1. $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, \pi^*(s))\}$
3. Return π

Search-based models: easy-first approach

CONs of classifier-based approaches

- Need to define an ordering over output variables
- Some decision are harder than others $\rightarrow$ fixed ordering can be suboptimal

Easy-first approach: rationale

- Make easy decisions first to constraint harder ones
- Learn to dynamically order decisions
- Analogous to constraint satisfaction algorithms

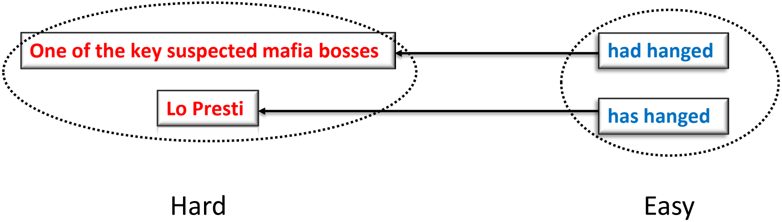
Example: Cross-document coreference

One of the key suspected mafia bosses arrested yesterday **had hanged** himself.

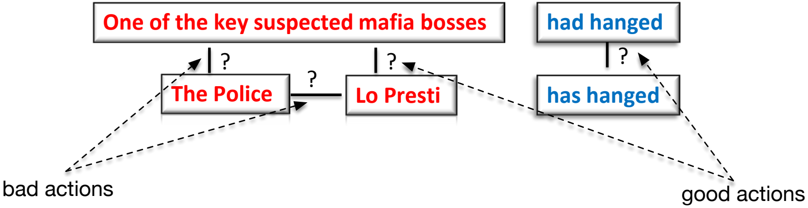
Doc 1

Police said **Lo Presti** **has hanged** himself.

Doc 2



Easy-first approach: inference



Easy action first

- State s is partial solution
- Set of possible actions $a \in A(s)$ from a state (no ordering)
- Action scoring function $f(s, a) = \mathbf{w}^T \Psi(s, a)$
- Proceed making highest scoring (most-confident) action first

Easy-first approach: learning

Easy-first policy learning

while not termination condition **do**
  for $(x, y) \in \mathcal{D}$ **do**
    $s \leftarrow I(x)$
    while not $\textsc{isTerminal}(s)$ **do**
      $a_p \leftarrow \max_{a \in A(s)} w^T \Psi(s, a)$
      if $a_p \in B(s)$ **then**
        $\textsc{update}(w, G(s), B(s))$
      end if
      $a_c \leftarrow \textsc{chooseAction}(A(s))$
      $s \leftarrow$ Apply a_c on s
    end while
  end for
end while

Easy-first policy learning

$$\text{UPDATE}(w, G(s), B(s))$$

Variants

- Highest scoring good action better than highest scoring bad action (perceptron update)
- Highest scoring good action better than all bad actions

$$a_c \leftarrow \text{CHOOSEACTION}(A(s))$$

Variants

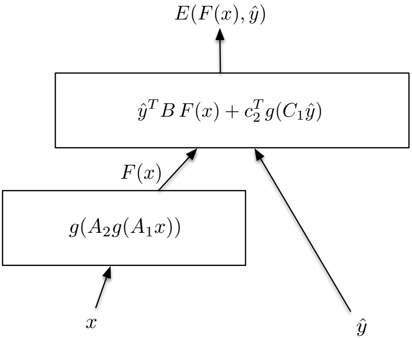
- Choose highest scoring *good* action ($a_c \in G(s)$, on-trajectory training)
- Choose highest scoring action ($a_c \in G(s) \cup B(s)$, off-trajectory training)

Combining energy-based and search-based approaches

HC-search framework

- Generate high-quality candidate complete outputs with search-based approach (H = search heuristic)
- Score candidates with energy function and select minimal energy output (C = cost/energy function)

Deep energy-based methods



Structured Prediction Energy Networks (SPEN)

- Energy function modelled as a deep network
- Replaces outputs $y \in \{0, 1\}^L$ with relaxations $\hat{y} \in [0, 1]^L$
- Training by gradient descent over weights using structured loss (e.g. as in structured SVM)
- Inference by gradient descent over $\hat{y}$ (+ rounding if needed)

SPEN

PROs

- Efficient inference by gradient descent
- No need to pre-specify input-output features (input-output representation learning)

CONs

- No algorithmic guarantees (local optimization of energy)
- No management of explicit constraints
- No support for hard constraints

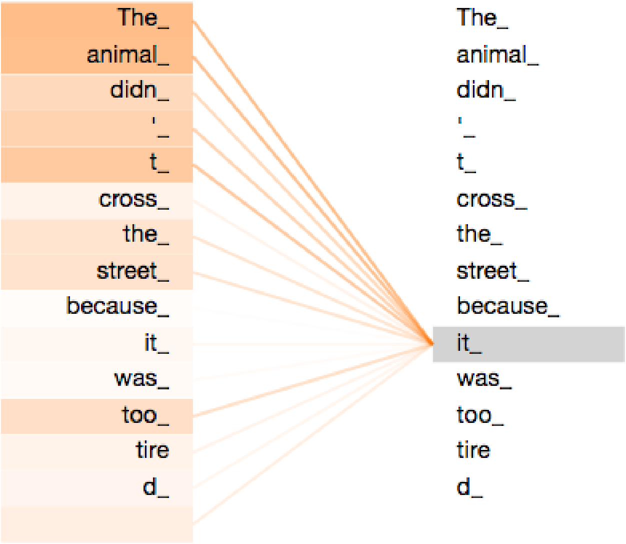
Deep search-based methods

Transformers for machine translation

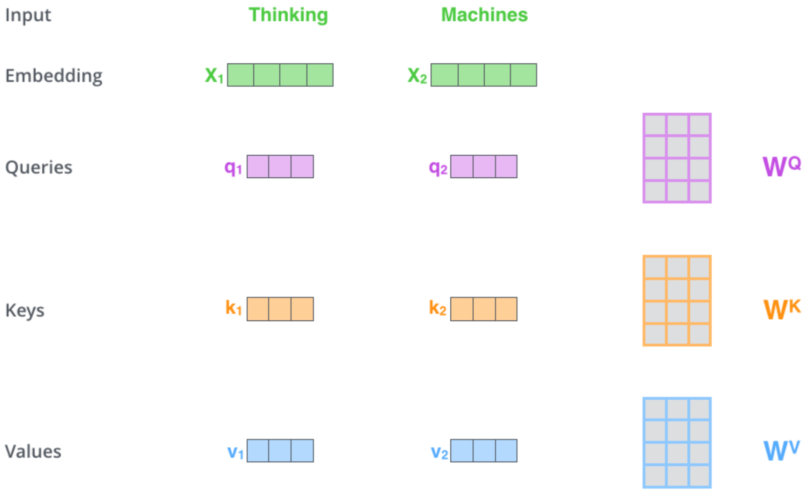
- Use attention mechanism to learn input word encodings that depend on other words in the sentence
- Use attention mechanism to learn output word encodings that depend on input word encodings and previously generated output words
- Predict output words sequentially stopping when the "word" end-of-sentence is predicted

Images and animations from Jay Allamar's "The Illustrated Transformer"

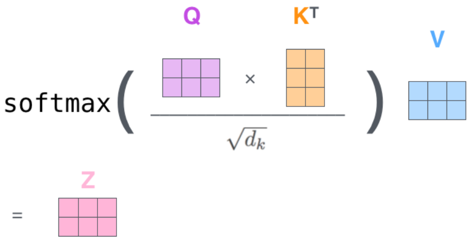
Tranformer: self-attention (concept)



Tranformer: self-attention (vectors)



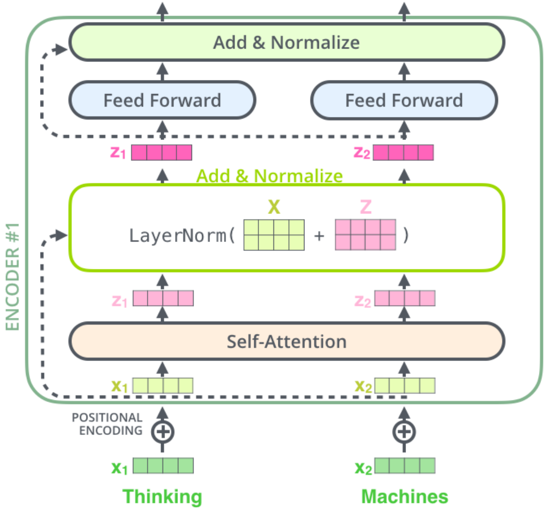
Tranformer: self-attention (computation)



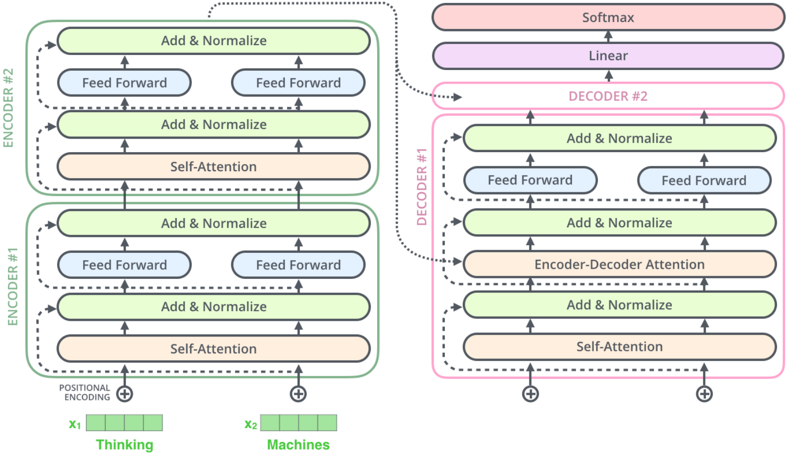
Steps

- Query vector q_1 times key vector k_2 gives importance of word 2 for encoding word 1
- Softmax normalizes importances over all words in the sentence ($\sqrt{d_k}$ helps numerical stability)
- Result z_1 is combination of values v_i for all words, each weighted by its normalized importance for 1

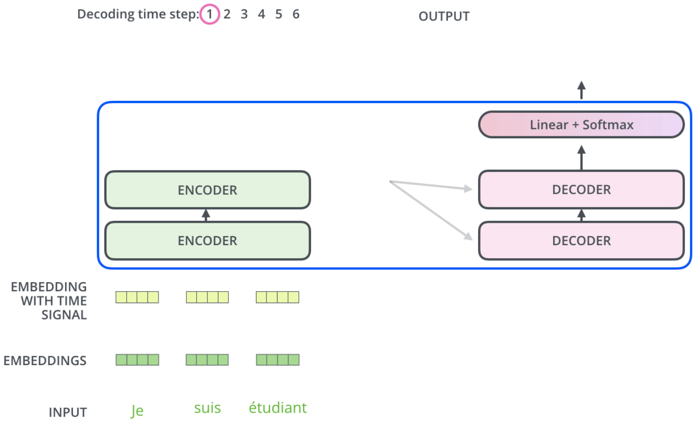
Tranformer: encoder layer



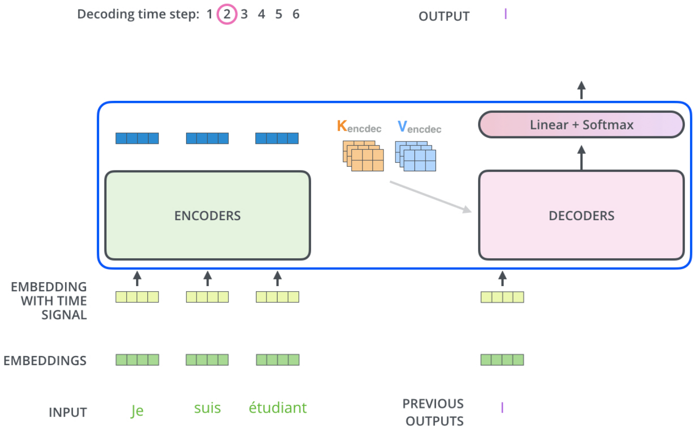
Tranformer: encoder-decoder architecture



Tranformer: predicting the first word



Tranformer: predicting the following words



References

Bibliography

- Deshwal, A.; Doppa, J. R.; and Roth, D., *Learning and inference for structured prediction: A unifying perspective*, in IJCAI 2019.
- LeCun, Y.; Chopra, S.; Hadsell, R.; Huang, F. J.; and et al., *A tutorial on energy-based learning*, in Predicting Structured Data, MIT Press.
- Joachims, T.; Hofmann, T.; Yue, Y.; and Yu, C.-N., *Predicting structured objects with support vector machines*, in Communications of the ACM, 2009.
- Daumé, H.; Langford, J.; and Marcu, D., *Search-based structured prediction*, in Machine Learning, 2009.
- Ross, S.; Gordon, G.; and Bagnell, D., *A reduction of imitation learning and structured prediction to no-regret online learning*, in AISTATS, 2011.
- Belanger D. and McCallum, A., *Structured prediction energy networks*, in ICML 2016.
- Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A., Kaiser L., and Polosukhin I., *Attention is all you need*, in NIPS 2017.

References

Software Libraries

- PyStruct - Structured prediction in Python (PyStruct) [http://pystruct.github.io/]
- Torch-Struct: Structured Prediction Library (Torch-Struct) [https://github.com/harvardnlp/pytorch-struct]
- PyTorch-Transformers: PyTorch implementations of NLP Transformers [https://pytorch.org/hub/huggingface_pytorch-transformers/]