Joint Inference is not Always Optimal

Hal Daumé III
Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
hdaume@isi.edu

Abstract

It is becoming increasingly popular to solve several NLP tasks simultaneously, rather than in the more classical pipelined manner. We distinguish two styles of such *joint inference* problems. In the first, multi-output joint inference, one cares about the outputs of all the subproblems. In the second, single-output joint inference, one only cares about the output of one of the tasks. Our claim is that joint inference should only be considered in the multi-output case, all other things being equal (eg., training data). We show this theoretically and demonstrate it empirically.

1 Introduction

Common wisdom in natural language processing states that pipelining multiple systems should be avoided. This has led to a burgeoning of *joint inference* techniques that solve multiple problems simultaneously. We argue theoretically that joint inference is not always the appropriate solution. This argument holds whenever only one of the joint outputs is desired. Counter-intuitively, several papers in diverse areas demonstrate the opposite: joint inference helps even when only one output is desired. We suggest reasons why this might be the case together with remedies. We give experimental results that verify the theory and show that joint inference is not always optimal.[1]

When faced with a new problem, one must take a simple pipelined approach or a complex joint inference approach. Consider the task of part of speech tagging and parsing. In the pipelined approach, one would first build a tagger, then build a parser on top. In the joint approach, one would perform tagging and parsing simultaneously. Joint approaches fall into two categories, hinging on the *loss function* (or *evaluation criterion*). In the first type, *multi-output joint inference*, the loss function depends on the outputs from *all* tasks: we care about both the syntactic parse tree and the POS tags. In the second type, *single-output joint inference*, the loss function depends only on the output of *one* of the tasks: we only care about the syntactic tree, irrespective of the POS tags. Both types of joint inference have been explored recently in diverse tasks, including joint sequence labeling (Sutton et al., 2004), syntactic role labeling (Toutanova et al., 2005; Punyakanok et al., 2005a) and entity detection and tracking (Daumé III & Marcu, 2005a), among others.

Our claim is that joint inference is *only* appropriate in the multi-output case, and can hurt performance in the single-output case. None of our theory is based on complex analysis and follows straightforwardly from definitions (Section 2). Our goal is to raise awareness that joint inference is not optimal in all cases and to provide empirical evidence (Section 4) in well controlled experiments to back up this claim. Our results hold because in single-output learning, we do not care about the secondary task. Intuitively, while the secondary task may provide additional information if solved perfectly, a solution to the second task has access to no more information than a solution to the primary task. This makes solving the secondary task useless.

[1] By *joint inference* we mean *training* to minimize a joint loss function. We do not consider learning independent classifiers and combining them with joint inference, as advocated by Punyakanok et al. (2005b).

2 Theoretical Analysis

All joint inference problems can be generically considered in the machine learning framework as structured prediction problems. We are given an input space $\mathcal{X}$ and two output spaces $\mathcal{Y}_1$ and $\mathcal{Y}_2$, corresponding to the multiple outputs desired. The formal machine learning problem is defined by a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}_1 \times \mathcal{Y}_2$ and a given loss function $l : (\mathcal{Y}_1 \times \mathcal{Y}_2) \times (\mathcal{Y}_1 \times \mathcal{Y}_2) \to [0, \infty)$, where $l((y_1, y_2), (\hat{y}_1, \hat{y}_2))$ is the loss incurred for predicting $(\hat{y}_1, \hat{y}_2)$ when the correct answer is (y_1, y_2). Our goal is to find a function $f : \mathcal{X} \to \mathcal{Y}_1 \times \mathcal{Y}_2$ that minimizes the expected joint loss, Eq (1).

$$L_{\mathcal{D},l}(f) = \mathcal{E}_{(x,y_1,y_2)\sim D}\ l\big((y_1, y_2), f(x)\big) \tag{1}$$

Suppose that l is independent of task 2: $l((y_1, y_2), (\hat{y}_1, \hat{y}_2)) = l_1(y_1, \hat{y}_1)$ for some l_1. In this case, the joint loss (Eq (1)) can be rewritten as a marginal loss (Eq (2)), where we write $\text{fst}(a, b) = a$ and D_1 is the marginal distribution of D after integrating out y_2.

$$L_{\mathcal{D},l}(f) = \mathcal{E}_{(x,y_1)\sim D_1}\ l_1\big(y_1, \text{fst}\ f(x)\big) \tag{2}$$

In single-output joint learning, our goal is to minimize Eq (2); in multi-output joint learning, our goal is to minimize Eq (1). It is immediately clear from Eq (2) that we may as well ignore any information from task 2 to solve this problem.

We can obtain a stronger result. Not only is trying to learn task 2 simultaneously *not beneficial*, but it is often *harmful*. In order to learn task 2, we must invent a loss function for it. This is because our *true* loss function l can be expressed purely in terms of l_1 (which does not depend on task 2), it is not a priori clear what is the best loss to minimize for task 2. Suppose we cook up some $l_2 : \mathcal{Y}_2 \times \mathcal{Y}_2 \to [0, \infty)$. Since we must, at the end, seek to optimize a single function, we must find a way to combine this new l_2 with our true l_1. An easy way to do this is by interpolation. Let $\lambda \in [0, 1]$ and define:

$$l((y_1, y_2), (\hat{y}_1, \hat{y}_2)) = \lambda l_1(y_1, \hat{y}_1) + (1 - \lambda) l_2(y_2, \hat{y}_2) \tag{3}$$

In the extreme case when $\lambda = 0$, we ignore l_1, which is clearly the wrong thing to do. However, for any $\lambda < 1$, our optimization is suboptimal. To see this, suppose that $\lambda < 1$. In this case, our model will be willing to make one error on task 1 in exchange for $\lambda/(1-\lambda)$ errors on task 2. But we do not care about task 2 in our true loss function. If we *do not* set $\lambda = 1$, then our model will *allow itself errors on task 1* in order to save errors on task 2. This will lead to *worse* task 1 performance. One might argue that this is an artifact of choosing to linearly interpolate l_1 and l_2. Perhaps there is a better way of combining them that is advantageous. Or perhaps we can concoct a new loss function l that depends on both tasks. Unfortunately, this will never work, for essentially the same reason as before. Anything that cares about performance on task 2 *must* trade good task 2 performance for good task 1 performance. This inevitably leads to worse results.[2]

3 Why has Single-Output Joint Inference been Reported to Work?

Given the foregoing analysis, any use of joint inference for a single-output problem is theoretically unjustified. Despite this, several papers have been published that show that single-output joint learning improves the performance over a baseline pipelined system. Our goal is not to "point fingers," so we will discuss no paper specifically. Instead, we will list some general reasons why single-output joint learning might be successful, and how to avoid these decisions that lead to this counterintuitive and theoretically unjustified results.

3.1 Optimizing an Incorrect Loss Function

For single-output learning, our optimization goal is that of minimizing $L_{\mathcal{D},l}$, as defined in Eq (2). The criterion optimized when a probabilistic model is used (Bayes net, HMM, CRF, etc.) is rarely

[2] In fact, for a linearly interpolated loss, there is no reason to perform joint inference. This is because there is no trade-off between the two losses: we simply want to minimize l_1 and l_2 independently.

the same. For instance, in sequence labeling we often wish to minimize Hamming loss or F-score. However, a model like a CRF optimizes a conditional log loss *over the entire sequence*. This criterion is sufficiently far from the true loss that it is difficult to analyze whether a joint log loss model or multiple single log loss models are more appropriate. A related issue is that of regularization. Forcing a single statistical model to account for multiple outputs may require a greater amount of generalization than in the single output case.

The solution to the optimization problem is to optimize a criterion as close to the true loss as possible. This has previously been argued very well theoretically (Vapnik, 1995) and experimentally (Bottou et al., 1997). There are several techniques that allow us to optimize something better for structured models, and we might want to consider using one of those (Daumé III & Marcu, 2005b; Daumé III et al., 2005; Taskar et al., 2005). The solution to the regularization problem is simply to use a better regularized learning system.

3.2 Using Insufficiently Many Features

Often, by using joint learning, we use a larger feature space than we might realize. This is easiest to see in the case of sequence labeling. Suppose both of our tasks are simple sequence labeling tasks: part of speech tagging and syntactic chunking. We will likely use features to predict the label of word i based on a window around i. Suppose for simplicity that we use a window of size two: features are based on two positions to the left and right. We will also use the POS tag within this window to predict the chunk label. This is strictly more information than ignoring the part of speech labels entirely. The chunk label at position i depends on the POS label at position $(i-2)$. But the POS label at $(i-2)$ depends on the word at $(i-2)-2 = i-4$. Thus, we have effectively doubled our window size, leading to better performance.

There is a second source of additional features. Suppose, as might be typical, we condition the chunk label at position i on the *pair* of POS tags at positions $i-1$ and $i-2$. These POS tags serve as a *summary* of the features at those positions. By using the pair of tags, we introduce a cross-product effect over the features. At a more extreme level, for predicting the chunk label at position i, we will often pair the POS tag at $i-1$ with the whole feature set at position i. Again, we are artificially introduce a form of cross-product feature space and giving the learning system significantly more information.

We may alleviate both of these problems by simply giving the learner more features: Modern learning techniques are resilient to large feature sets.

3.3 Using Different Knowledge Sources

This reason is obvious, but deserves comment for completeness. Consider again the POS/chunking task described above. For identifying syntactic chunks, we will often use dictionaries/ontologies that describe names and common collocations. Similarly, for assigning POS labels, we will often use lexical resources such as WordNet or distributional clusters (for alternative tasks, such as gene name identification, the differences in knowledge sources are more pronounced). When we do joint learning, we are giving both sides an advantage by allowing them access to resources/features that were not otherwise available to them. This also explains the reverse phenomenon: when joint decoding actually helps performance in task 2. That is, by finding chunks as well as POS labels, we get a better POS tagger. But finding chunks gives us more information, since it uses ontologies, so the fact that this helps the POS tagger is not surprising. Again, the solution is just to give the learner all the available features.

3.4 Poor Control over Submodels

We often compare a jointly system with a pipelined version of that system. Unfortunately, building a high-performance pipelined system may be difficult when the output of the first step is used as input in the second step. Continuing the POS/chunking task, suppose we build a POS tagger on the training data. We now with to build a chunker on the training data that uses POS tags as input features. A question arises: where should these POS tags come from. If we use the true POS tags, our model may be overconfident in their correctness, and severely overfit. If we use our learned POS tagger's output as input, we may also overfit (because the POS tagger is tested on its training data).

The best solution to this problem is to not pipeline: simply build a syntactic chunker without POS tags as input. If pipelining is truly desired (perhaps for computational reasons), the best approach is likely to perform cross-validation of the POS tagger on the training data. Build a tagger on 99% of the training data and test it on the last 1%. By doing this 100 times, we obtained a noisily tagged version on the training data. Usually 100 folds would be too expensive computationally, but the more folds, the better. There still will exist a bias in these tags, but this approach is significantly better than either using the true tags or the tags output by a tagger trained on the whole training set.

4 Experimental Results

We consider the following simple joint learning task: syntactic chunking and part of speech tagging from CoNLL 2000. We chose this task for its simplicity, ease of implementation and the readiness of a common data set. We have based all our experiments on the Learning as Search Optimization framework (Daumé III et al., 2005), for which it is trivial to change our loss function. To remove any possibility that our results are affected by search errors, we use a beam size of 100 in all experiments. In all the experiments we describe, we will consider five different learning setups:

1. Joint learning, where our loss function is *joint accuracy*. In this loss function, we get one point per word if we predict both its POS tag and chunk tag correctly.
2. Joint learning, where our loss function is linearly interpolated accuracy for POS tagging and F-score for chunking. We use an interpolation weight of 0.5.
3. POS label learning, where we simply learn a POS tagger and do not use any chunk information. Our loss function is POS accuracy.
4. Chunk learning, where we simply learn a chunk model to optimize F-score. We do not use *any* part of speech information.
5. Chunk learning with true POS input, where we learn a chunk model to optimize F-score, using the true POS tags as features.
6. Chunk learning with noisy POS input, where we learn a chunk model to optimize F-score (20-fold cross validation to produce noisy POS tags).

We will give test data scores on the following five metrics: (1) Joint accuracy; (2) POS accuracy; (3) Chunk accuracy; (4) Chunk F-score; and (5) Linearly interpolated POS accuracy and Chunk F-Score ($\lambda = 0.5$).

The first experiment shows what happens when things are done properly. The remaining experiments show what happens when each of the three best practices mentioned earlier are not followed. In all the experiments, we use a standard feature set: word identity, stem, case pattern, position, prefixes and suffixes up to length 4, word length and word membership in a set of gazetteers of people, companies and locations. As structured features, we use second-order Markov features; in the appropriate tasks combining both POS and chunk-level tag information into the Markov features.

For the purpose of most clearly demonstrating the performance differences for the different experiments, we have run on a subset of the CoNLL 2000 data set. We use the first 4000 sentences from the CoNLL 2000 training set as training data, the next 500 as development data, and the full test set. Similarly scaled results are obtained by training on the full data set, but the numbers are so close that many differences are not statistically significant (see Section 4.4).

4.1 Correct Implementation

Our first experiment is set up to avoid the three pitfalls described earlier. No windows are used: features are applied at all positions at all times. We apply all features for both the POS and Chunking problems. The results are shown in Table 1. We can note several things from these results.

Table 1: Results of the experiments for the correct implementation. The accuracy metrics are along the top and the systems are along the left.

	Jnt Acc	POS Acc	Chn Acc	Chn F	Lin Int
Joint (Acc)	**92.7**	97.6	95.0	*91.3*	94.5
Joint (Lin Int)	*92.4*	97.9	94.3	*91.9*	94.9
POS Only		97.9			
Chunk Only			94.2	**92.1**	
Chunk w/ True POS	*91.0*	97.9	*93.1*	*90.9*	*94.4*
Chunk w/ Noisy POS	*92.3*	97.9	94.2	**92.1**	94.9

Table 2: Results of the experiments for the badly windowed implementation. The accuracy metrics are along the top and the systems are along the left.

	Jnt Acc	POS Acc	Chn Acc	Chn F	Lin Int
Joint (Acc)	91.9	97.3	94.4	91.0	94.2
Joint (Lin Int)	91.7	97.5	94.1	**91.9**	**94.7**
POS Only		97.5			
Chunk Only			93.4	*91.3*	
Chunk w/ Noisy Input	91.3	97.5	93.6	*91.6*	*94.5*

- The highest chunk F-score is obtained by the non-joint models (92.1): The joint models perform at best with a score of 91.9 and at worst at 91.3.
- Part of speech tags do not help the chunking task when computed noisily through cross-validation. As argued before, this is not surprising. The chunk accuracy and chunk F-score are unchanged by the use of the predicted POS labels.
- When the true POS tags are used, the model significantly overfits on these features and performs poorly on the test data (90.9 versus 92.1 chunk F-score and 91.0 versus 92.3 joint accuracy).
- If our goal is to have a *joint model* that performs well according to joint accuracy, we should optimize one as such (to get 92.7 versus 92.4 or 92.3).
- If our goal is to have a joint model that performs well on linearly interpolated accuracy/F-score, it makes no difference if we optimize jointly or not.

In summary: by implementing everything correctly, we have obtained results that agree with the theory. The best chunk F-score is obtained by completely ignoring the POS task (if we solve POS tagging separately, it does not hurt but it also does not help). If our goal is a low joint loss then we should optimize jointly. It is also worth noting that there is no difference between jointly optimizing for the linearly interpolated metric and optimizing separately. This is also predicted by the theory.

4.2 Badly Windowed Implementation

Our second experiment is set up to show the effect of the window sizes on the performance. That is, we only apply the features within a window of 2 on each side. The results are shown in Table 2. We can note several things from these results and those presented previously:

- Overall, the restriction to windows drops performance by half a point.
- For Chunk F-score, the linearly interpolated joint model (91.9) outperforms the separate models (91.6). This is an artifact of artificially doubling the window size.
- According to linearly interpolated accuracy/F-score, we are better off optimizing jointly (94.7) than separately (94.5).

In summary, if we do not give our chunker access to all the features it needs, it does worse. We have actually "shown" that joint learning helps. However, as discussed previously, this is not because joint learning is the appropriate thing to do, but rather because it artificially increases the size of the feature set. This gives the learner more features *implicitly*. An alternative is to explicitly feed the learning algorithm more features. The latter is usually preferable because it is often computationally more efficient.

Table 3: Results of the experiments for the badly featured implementation. The accuracy metrics are along the top and the systems are along the left.

	Jnt Acc	POS Acc	Chn Acc	Chn F	Lin Int
Joint (Acc)	**92.5**	97.4	95.0	91.3	94.4
Joint (Lin Int)	92.0	**97.6**	94.3	*91.9*	**94.9**
POS Only		*96.7*			
Chunk Only			94.2	**92.1**	
Chunk w/ Noisy Input	*91.1*	96.7	94.2	**92.1**	*94.4*

4.3 Badly Featured Implementation

Our third experiment is set up to show the effect of applying different feature sets to the two problems: we do not use the gazetteer features for the tagging problems. The results are shown in Table 3. We can note several things from these results and those presented previously:

- There are few performance differences between this and the correct implementation: the chunk scores are identical; the differences are in the POS accuracies.
- For the single-output case of chunk F-score, we do better by optimizing the chunker separately (92.1 versus 91.9).
- For POS accuracy, we do better with joint learning (97.6 versus 96.7), as would be expected since this gives more features to the POS tagger.
- We get better joint accuracy (92.5 versus 91.1) and better linear interpolated scores (94.9 versus 94.4) for doing joint inference versus pipelined inference.

Again, here we have "shown" that superior performance is obtained though single-output joint learning (for POS accuracy). However, as before, this is not because joint learning is appropriate in this context. Rather, it is because we have supplied more information to the POS tagger in the form of additional features. In summary, if we want a good POS tagger, we should give it more features. If we want a good chunker, we need not build a POS tagger, so long as the chunker has all the needed features.

4.4 Experiments on the Full Dataset

Here, we report some results for running on the full CoNLL 2000 data set. We use the first 90% of the CoNLL training data for training and the last 10% for development. The scores are higher than before and the differences between techniques are significantly less pronounced. However, some results are listed below for completeness:

- We obtain a 94.4 chunk F-score by direct optimization.
- When optimized jointly, chunk F-score drops to 94.2.
- We obtain a 96.8 joint accuracy by optimizing for joint accuracy; when optimized separately, we also obtain 96.8.
- When windowed features are used, we obtain better chunk F-score by optimizing jointly (94.1) than chunking alone (94.0).
- When not all features are used, we obtain a better POS accuracy doing joint inference (99.1) versus not (98.7).

These results track the results described in the previous sections, though the differences are not as substantial. Part of the reason joint inference work so well on this problem is that very high POS accuracies are possible. We suspect that for harder problems, performing unjustified joint learning will result in significantly worse results.

5 Conclusion

Joint inference is a powerful learning technique when dealing with problems where multiple types of outputs are desired. In this case, learning jointly is the correct thing to do. However, when one of the outputs is not desired and is used only as an intermediary, we are better off directly optimizing only the output we want. Not only does this often make the optimization and prediction easier computationally, but it also leads to better theoretical guarantees and empirical performance. These results of course do not hold when, for instance, extra annotated data is available for the "unwanted" task. However, in this case, a model for the unwanted task should be learned *separately* and then *pipelined*, rather that learned jointly.

Acknowledgments

I wish to express my thanks to John Langford for helpful discussions; also thanks to Daniel Marcu, Ben Taskar, Ryan McDonald and Alex Fraser for constructive comments on an earlier draft of this paper. This work was partially supported by DARPA-ITO grant N66001-00-1-9814, NSF grant IIS-0097846, NSF grant IIS-0326276, and a USC Dean Fellowship.

References

Bottou, L., LeCun, Y., & Bengio, Y. (1997). Global training of document processing systems using graph transformer networks. *Proceedings of Computer Vision and Pattern Recognition* (pp. 490–494). Puerto-Rico: IEEE.

Daumé III, H., Langford, J., & Marcu, D. (2005). Search-based structured prediction as classification. *NIPS Workshop on Advances in Structured Learning for Text and Speech Processing.* Whistler, Canada.

Daumé III, H., & Marcu, D. (2005a). A large-scale exploration of effective global features for a joint entity detection and tracking model. *Proceedings of the Joint Conference on Human Language Technology Conference and Empirical Methods in Natural Language Processing (HLT/EMNLP)* (pp. 97–104). Vancouver, Canada.

Daumé III, H., & Marcu, D. (2005b). Learning as search optimization: Approximate large margin methods for structured prediction. *Proceedings of the International Conference on Machine Learning (ICML).*

Punyakanok, V., Roth, D., & Yih, W.-T. (2005a). The necessity of syntactic parsing for semantic role labeling. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1117–1123). Edinburgh, Scotland.

Punyakanok, V., Roth, D., Yih, W.-T., & Zimak, D. (2005b). Learning and inference over constrained output. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1124–1129). Edinburgh, Scotland.

Sutton, C., Rohanimanesh, K., & McCallum, A. (2004). Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 783–790). Banff, Canada.

Taskar, B., Chatalbashev, V., Koller, D., & Guestrin, C. (2005). Learning structured prediction models: A large margin approach. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 897–904). Bonn, Germany.

Toutanova, K., Haghighi, A., & Manning, C. (2005). Joint learning improves semantic role labeling. *Proceedings of the Conference of the Association for Computational Linguistics (ACL)* (pp. 589–596). Ann Arbor, Michigan.

Vapnik, V. N. (1995). *The nature of statistical learning theory.* Springer.