Incremental Natural Language Processing: Challenges, Strategies, and Evaluation

Arne Köhn
Natural Lanuguage Systems Group
Department of Informatics
Universität Hamburg
koehn@informatik.uni-hamburg.de

Abstract

Incrementality is ubiquitous in human-human interaction and beneficial for human-computer interaction. It has been a topic of research in different parts of the NLP community, mostly with focus on the specific topic at hand even though incremental systems have to deal with similar challenges regardless of domain. In this survey, I consolidate and categorize the approaches, identifying similarities and differences in the computation and data, and show trade-offs that have to be considered. A focus lies on evaluating incremental systems because the standard metrics often fail to capture the incremental properties of a system and coming up with a suitable evaluation scheme is non-trivial.

Title and Abstract in German

Inkrementelle Sprachverarbeitung:
Herausforderungen, Strategien und Evaluation

Inkrementalität ist allgegenwärtig in Mensch-Mensch-Interaktiton und hilfreich für Mensch-Computer-Interaktion. In verschiedenen Teilen der NLP-Community wird an Inkrementalität geforscht, zumeist fokussiert auf eine konkrete Aufgabe, obwohl sich inkrementellen Systemen domänenübergreifend ähnliche Herausforderungen stellen. In diesem Überblick trage ich Ansätze zusammen, kategorisiere sie und stelle Ähnlichkeiten und Unterschiede in Berechnung und Daten sowie nötige Abwägungen vor. Ein Fokus liegt auf der Evaluierung inkrementeller Systeme, da Standardmetriken of nicht in der Lage sind, die inkrementellen Eigenschaften eines Systems einzufangen und passende Evaluationsschemata zu entwickeln nicht einfach ist.

1 Introduction

Interaction using language is incremental in many forms: In a dialogue, understanding takes place continuously and participants are able to interrupt each other or signal whether they understand the currently ongoing utterance. Simultaneous interpreters translate speeches as they are spoken. Texts are (partially) understood before they are fully read. Incremental processing exploits this property by starting to compute before all input is available, allowing a system to already act on partial input. Without incremental processing, an NLP system is unable to perform any of these tasks as it is bound to wait for complete utterances or a finished text and only afterwards it can compute how to (re-)act, leading to deficiencies in human-computer interaction.

First, I want to delimit this notion of incrementality from other notions: Systems that work on a complete input but generate output layer by layer, e.g. shallow to deep syntax, are sometimes called incremental, e.g. by Aït-Mokhtar et al. (2002). These systems are not discussed in this survey. Anytime algorithms are incremental in the sense that they iteratively improve their output for a fixed input, given more and more processing time. They can play an important role in incremental systems as they allow to

perform a trade-off between processing time – i.e. system responsiveness – and quality. The question of whether to employ anytime algorithms is however orthogonal to the approaches discussed in this survey[1].

In an incremental system, all processors need to work incrementally. A prime example is the Verbmobil project, which set out to develop a portable simultaneous interpreter (Kay et al., 1994). The project developed speech recognition and synthesis components, syntactic and semantic parsers, self-correction detection, dialogue modeling and of course machine translation, showing that incrementality is an aspect that touches nearly all topics of NLP. This project also exemplifies that building incremental systems is not easy, even with massive funding[2]: Only one of the many components ended up being incremental and the final report makes no mention of *simultaneous* interpretation (Wahlster, 2000).

The incremental nature is more obvious in speech than in written language because language processing often processes already-written text in bulk whereas interactive systems are a major research focus for speech-based applications. As the benefits of incrementality are more prominent in processing dialogues compared to text, research on incrementality in NLP has been primarily driven by speech-based research questions, such as understanding based on partial speech recognition (Sagae et al., 2009), determining when to respond during ongoing utterances (DeVault et al., 2009), training actor policies for rapid task-based dialogue (Paetzel et al., 2015), continuous understanding and acting (Stoness et al., 2005), incremental repair detection (Hough and Purver, 2014), or incremental reference resolution (Schlangen et al., 2009).

2 Psycholinguistic evidence

Humans still pose the gold standard for language processing, especially if the processing does not happen on large scale but in an interactive setting. When speaking, they perform several tasks incrementally and in parallel, from conceptualization to articulation (Levelt, 1989). Machines mimicking human behavior need to be able to perform similar computations as humans in such settings to be seen as a competent partner, and psycholinguistic research gives insight into the processes that humans perform. Psycholinguistic research is often carried out by timing experiments, i.e. they make use of the incremental processing by humans to gain insights into linguistic processes, mostly by eye-tracking (Tanenhaus et al., 1995; Sturt and Lombardo, 2005; von der Malsburg and Vasishth, 2011), but also by timing word-by-word reading of sentences (Gibson and Warren, 2004).

Language is perceived incrementally and this incremental processing is influenced by other modalities even while speech is perceived (Tanenhaus et al., 1995). Humans seem to create fully connected structures for sentence prefixes, even for coordinate structures, and a continuation of a sentence that does not match the predicted structure can be measured by increased reading times (Sturt and Lombardo, 2005). They perform syntactic re-analysis and use various strategies to rescan sentences (von der Malsburg and Vasishth, 2011).

3 A typology for incremental problems

When working on an incremental processor, it is helpful to know which other already existing processors had to deal with similar challenges despite being designed for completely different NLP tasks. In this section, I will examine properties relevant for classifying processors. These properties describe the input and output data as well as the relation between input and output and help classifying concrete tasks in Section 4. For an in-depth discussion of properties relevant to incremental processing, see Chapter 5 of Guhe (2007)[3].

3.1 Data types

Data can be *structured* (e.g. syntactic or semantic structures) or *sequential*. Sequential data can be *discrete* (e.g. words) or *continuous* (e.g. speech signals) along the time axis[4]. Structured data is always discrete on

[1] An anytime algorithm could be seen as a simply another non-monotonic processor, as described in Section 3.4.

[2] Verbmobil had a funding of 116 million DM, (~60 million €, or 78 million € when adjusted for inflation).

[3] The properties discussed by Guhe (2007) are mostly complementary to the ones discussed here.

[4] Data that is discrete along the time axis can of course include continuous data such as word embeddings.

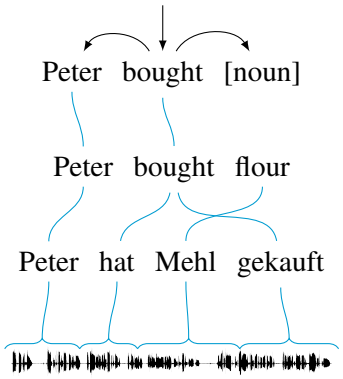
Process	Data	**Data** / *alignment* properties
		**structured, including prediction**
Parsing		*vertices partially grounded, edges not grounded*
		discrete, sequential
Machine translation		*reordering, fuzzy mapping*
		discrete, sequential
Speech recognition		*order-preserving, clear mapping*
		continuous

Figure 1: Different types of data, groundings, and processors. Note that for parsing only the prefix "Peter bought" is processed to exemplify intermediate structure generated for incomplete input. Incremental systems can take many forms, and this example is not meant to perform a specific task.

the time axis. A processor can take one type of data as an input and create another one as output; some examples are depicted in Figure 1.

Sequential data can usually be subdivided along a time axis, i.e. there exists a total ordering between the elements of the input (or output respectively). Structured data does not exhibit this property: Given, for example, the dependency tree produced by parsing in Figure 1, the words are ordered, but the dependencies between the words can't be clearly attributed to a word: They could be attributed to the head or the dependent; this poses an additional challenge for evaluation.

3.2 Granularity

The *granularity* determines the size into which the input and output is subdivided. Assuming coarse enough granularity, every system can be seen as incremental: A normal syntax parser works non-incremental inside a sentence, but incrementally if processing a paragraph with the basic units being sentences. Grapheme to phoneme conversion is incremental when processing text with the basic unit of words. When considering a larger incremental system, a processor usually seen as non-incremental might be incremental enough: If a language generation system works on the level of words, the grapheme to phoneme conversion does not need to be able to process sub-word input. On the other hand, if input typed by a user should be vocalized, sub-word granularity might be needed. The granularity of a pipeline is determined by its most coarse-grained component. In general, fine-grained processing is harder than coarse-grained processing; a system can always process data fine-grained internally while having coarse-grained interfaces, whereas the opposite is not possible.

3.3 Grounding

Grounding describes the alignment from the generated output to the elements of the input that yielded evidence for this output (Schlangen and Skantze, 2009). Grounding allows to reason about which part of the output can be reasonably generated given only partial input. In some cases, this alignment is explicit in both test and training data, e.g. in sequence labeling tasks where each element of the input is assigned a label. In other cases such as machine translation, there is no gold standard word alignment[5] and even a human-generated alignment would not create a one-to-one mapping between input and output. In addition, the alignments may or may not be *order-preserving*: A tagging task preserves the ordering, whereas in translation reordering takes place (cmp. "hat Mehl gekauft" → "bought flour" in Figure 1).

[5] At least not on the word level, but alignments can be generated automatically, see e.g. Och and Ney (2003)

3.4 Monotonicity

A system is *non-monotonic* if it is allowed to retract output it has previously produced. For example, an incremental sequence labeler that is free to re-assign labels can change its mind about every element for a sentence once the sentence is complete. A *monotonic system* on the other hand is required to only extend previously generated output without retracting information. Some components are inherently monotonic because their output is not fed to another processor of the system but to the outside world; e.g. a speech synthesizer is inherently monotonic as it cannot retract sound waves realized through a loudspeaker.

Monotonicity limits the quality a component can produce as it can not revert a decision that turns out to be wrong later on in light of additional available input. In contrast, a non-monotonic component can always achieve the same non-incremental output as a non-incremental component by simply replacing all intermediate output with the one of the non-incremental component once all input is available.

The precise meaning of monotonicity needs to be defined for each component. For sequential output, the most common definition is to only allow appending to the output. For structured output, the structure of an increment could be required to be a super-set of its predecessor.

Non-monotonic output can only be generated sensibly if the consumers of the output can deal with non-monotonic input. Otherwise these consumers might ignore the revisions made to previous output and end up with an inconsistent input or have to restart their computation in light of new input.

3.5 Timeliness

Each NLP processor has to optimize *what* to output given an input. An incremental system also needs to decide *when* to provide output. Discrete input provides specific anchors for this decision, continuous input does not and new output can be generated continuously[6]. Such decisions also need to be made by human interpreters while performing simultaneous translation; they need to buffer input until they are able to produce additional output. The characteristics of this (human) process varies by language, e.g. the delay is relatively long when translating from German to English because the verb in the input tends to be delayed (Goldman-Eisler, 1972).

3.6 Trade-Off between properties

Incremental components have to make a trade-off between timeliness (i.e. the amount of delay introduced between input and output), output quality, and the amount of non-monotonicity (Beuck et al., 2011a; Baumann et al., 2009). High-quality, monotonic output can be obtained by delaying all output. This strategy taken to the extreme results in a non-incremental system that only produces one complete output once all input is available. To reduce the delay, non-monotonicity via output revisions can be allowed, or compromises with respect to the quality of the output can be made. Gradual trade-offs can also be performed: allowing infrequent revisions and/or mild delays can lessen the negative impact on accuracy. These trade-offs are universal to all incremental processors.

4 Incremental processors

This section discusses three tasks that can be performed incrementally to exemplify strategies to "incrementalize" a processor: first, incremental speech recognition, a continuous sequence labeling problem that is usually implemented as a non-monotonic processor, then incremental machine translation, which is a sequence to sequence task with reordering implemented monotonically, and third parsing, a sequence to structure task, which is implemented both monotonic an non-monotonic.

4.1 Speech recognition

Speech recognition lends itself to incremental processing because it is used in all interactive spoken dialogue systems and the decoding happens incrementally even for non-incremental use-cases. It is therefore possible to look into a speech recognizer (SR) to obtain the most probable hypothesis at each point in time without modifying the recognizer (Baumann et al., 2009). Because the SR is not changed

[6]While continuous input is made discrete before reaching a processor, the discretization is usually in the order of milliseconds and can be seen as continuous for all practical purposes.

from the non-incremental one, the only optimization point is when to let new output through, i.e. to implement a *gatekeeper*. Its policy can be guided by observing the time that a hypothesis survived without being discarded (Baumann et al., 2009) or based on the internal state of the recognizer (Selfridge et al., 2011). McGraw and Gruenstein (2012) show that even sophisticated stability estimation only slightly improves upon the simple age-based estimation by Baumann et al. (2009). Given that the SR is the same for the incremental case as for the non-incremental one, no trade-off is performed against the accuracy. In addition, Selfridge et al. (2011) noted that if during decoding all beams in the beam search pass through the same state, the prefix up to that state can be declared stable because future decoding will not change the most probable path up to that state; this observation essentially makes use of the Markov property and is primarily useful if grammar-based recognition is performed.

4.2 Machine translation

Translation can be performed as batch processing (e.g. for websites) or incrementally, to facilitate human-human interaction. Modern approaches to machine translation, i.e. neural machine translation, employ a sequence to sequence model where the input sequence is encoded into a representation and then decoded again. This can be performed using recurrent networks which represent the input as a single fixed-length vector and possibly modeling attention to the input when generating the output sequence (Bahdanau et al., 2014) or even without a recurrent network, using only attention to model the influences from the input sequence to the output sequence to generate (Vaswani et al., 2017). In all these cases, all input is consumed before translation happens.

4.2.1 Incremental Neural Machine Translation (NMT) by using a gatekeeper

As already discussed, machine translation is a sequence to sequence problem where the output ordering does not conform to the input ordering and the ground truth for the grounding often is missing. As the incremental machine translation systems found in the literature are monotonic, the systems need to decide at which point they have enough information from the input to generate the next output token with high confidence. Gu et al. (2017) propose a system where an NMT processor repeatedly proposes an output token based on the currently available input and the generated output to a gatekeeper. The gatekeeper either accepts this output, resulting in a write operation to the output, or rejects it, resulting in a read operation on the input. The gatekeeper can work based on different policies, yielding different trade-offs between timeliness and accuracy: rejecting all output until the input is complete means falling back to a non-incremental behavior; performing alternating read and write actions eliminates delay but results in bad translations. Gu et al. (2017) train the policy using reinforcement learning with the reward based on the resulting BLEU score and the delay incurred; weighting them differently results in different trade-offs. Humans have different preferences regarding this trade-off, depending on whether the output is speech or subtitles (Mieno et al., 2015). The underlying NMT model is trained on complete sentences and therefore not adapted to incremental processing. An example of an incremental translation can be seen in Figure 2.

4.2.2 Reordering output

Reordering phenomena are a major hindrance to timely translation, as exemplified in Figure 2. Grissom II et al. (2014) deal with this by training a classifier to predict verbs needed for the output but not yet seen in the input, allowing the MT system to produce a verb without having seen its counterpart on the input. He et al. (2015) instead propose to edit the gold standard translations to better fit the source ordering for training the MT system by trying to imitate the transformations a human translator would perform. This approach tackles the problem that the training data is only available for the non-incremental case, which is not optimal for simultaneous translations. Human simultaneous translators produce sentences that systematically deviate from the "normal" target language but such data is not readily available for training (He et al., 2016). The training data is adapted by generating phrase-structure trees for the target sentences and applying (manually written) syntax-based reordering rules. The system then checks whether the reordering has reduced the delay based on an automatically computed alignment and if so uses the reordered version instead of the original one. As in the approach by Gu et al. (2017), the average delay induced by the system can be tuned, see Figure 2. Because the processor was not trained on gold-standard

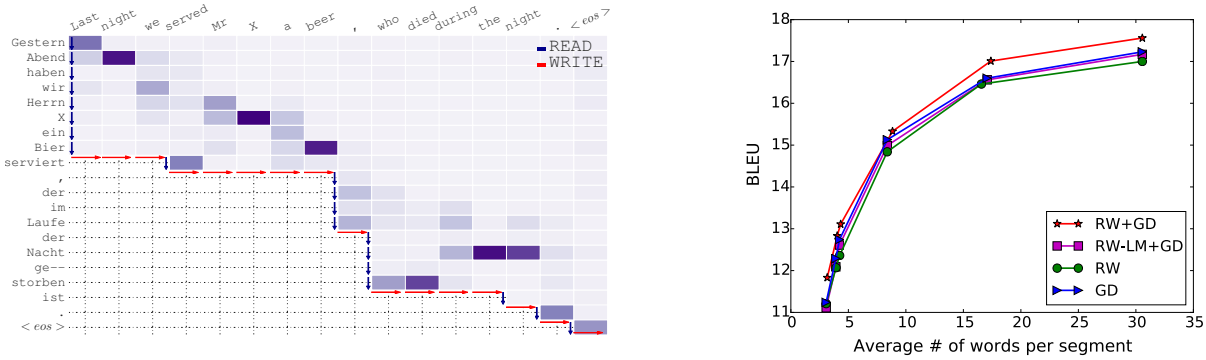


Figure 2: Left: Incremental NMT with attention, from Gu et al. (2017). Colors denote the attention given to each input token when generating an output token. Note the delay induced by the reordering for "serviert" (*served*) and "gestorben" (*died*). Right: trade-off decisions between delay (x-axis, measured in words translated at once) and accuracy (y-axis, measured in BLEU) from He et al. (2015).

data, it might yield sub-optimal results on gold-standard test data. He et al. (2015) evaluate on both gold-standard and transformed data and show that the system in fact performs better on both targets.

4.3 Parsing

Many state-of-the-art parsers work incrementally internally, both for semantic and for syntactic parsing: they use a transition system with a scorer and optionally combine that with beam search to find the best parse (Nivre, 2008; Huang and Sagae, 2010; Dyer et al., 2015; Swayamdipta et al., 2016; Kiperwasser and Goldberg, 2016; Zhou et al., 2016; Damonte et al., 2017, *inter alia*). While they build a structure incrementally going from left to right, they don't produce intermediate structures meant for incremental consumption; the intermediate states consist of several unconnected sub-structures. In addition, they usually employ look-ahead which delays the processing; approaches using Bi-LSTMs (such as Kiperwasser and Goldberg (2016)) effectively make use of the whole sentence during each step, making the computation non-incremental as it depends on the complete input being available. Noji and Miyao (2014) propose a transition system based on the ones discussed by Nivre (2008) that introduces dummy nodes which denote an expectation of upcoming words. This transition system still produces disconnected trees for sentence prefixes but is able to predict processing difficulties humans have when reading sentences with center embeddings. The other approaches described in this section all produce connected structures for sentence prefixes.

4.3.1 Incremental parsing with monotonic expansions

One approach to incremental parsing that produces connected structures at each step is to monotonically extend a connected structure and to employ a beam of possible structures to prevent being stuck with a structure that does not fit the continuation of the sentence. Using beam search, a parser does not provide monotonic output but still guarantees that one of the beam entries will be selected for a continuation.

Hassan et al. (2009) show why either a beam or delay is necessary if performing incremental parsing with monotonic extensions: They experiment with a parser based on Combinatory Categorial Grammar (Steedman, 2000). Their parser achieves an accuracy of 86% when using lookahead and performing greedy parsing (i.e. it does not use a beam). This accuracy drops significantly to 56% without lookahead because the parser often commits to a structure incompatible with the continuation of a sentence.

Roark (2001) presents a top-down phrase structure parser that performs beam-search to generate connected intermediate structures for every sentence prefix. As the parser is based on a probabilistic generative model, it can be used for language modeling and beats trigram models on the Penn Treebank (Marcus et al., 1994) (but not other sequence models, see e.g. Merity et al. (2017)). Demberg and Keller (2008) describe the PLTAG formalism, which is based on Tree Adjoining Grammar (Joshi and Schabes, 1997) and strives for psycholinguistic plausibility. It not only predicts upcoming structure needed for connectedness, but also structure required by e.g. valencies not yet filled in the prefix. The

predictions of the second type are automatically extracted from the treebank during lexical induction by learning the distinction between modifiers and arguments. The PLTAG formalism provides better predictions for reading-time experiments than predictions based on Roark's parser (Demberg et al., 2013). The combination of only extending entries in the beam and prediction leads to some sentences being unparseable for both parsers because the structures stored in the beam can all become incompatible with the input to be consumed. A larger beam reduces the number of unparseable sentences at the cost of additional memory and processing cost, but can not eliminate the problem.

4.3.2 Incremental gold standards for parsing

Köhn and Menzel (2014) go in the opposite direction and perform incremental parsing using restart-incrementality, i.e. performing a complete new parse for each sentence prefix without reusing previously generated output. This approach gives no guarantees that a certain structures will still be present in subsequent outputs but on the other hand is able to react on any upcoming input without constraints, yielding – in contrast to the approaches described in § 4.3.1 – the same accuracy for complete sentences as its non-incremental counterpart. The underlying parser performs a graph-based optimization (Martins et al., 2013) and has no notion of incrementality; instead, the gold standard the parser is trained on is adapted to consist of syntactic structures for sentence prefixes which contain prediction nodes as stand-ins for upcoming words. The partial dependency structure for a prefix is created by a rule based system that works on the complete dependency structure for the sentence. It keeps all dependencies between words in the prefix and delexicalizes words that are needed to connect the words in the prefix to the sentence root as well as arguments that are deemed to be predictable. Words outside the prefix that are neither needed for connection nor deemed to be predictable are deleted. The downside of this approach (in contrast to Demberg and Keller (2008), but similar to the reordering of output discussed in § 4.2.2) is that a rule set needs to be manually created for each language and that the linguistic intuition encoded into this rule set might be a bottleneck for the parser as structures not envisioned by the rule writer might be a better fit than the one created by the rules. Köhn and Baumann (2016) show that using the delexicalized predictions to augment 5-gram language models improves perplexity on the Billion Word Corpus (Chelba et al., 2013).

4.4 Natural language generation

Skantze and Hjalmarsson (2013) compare a non-incremental dialogue system and an incremental one, with which language learners interact to buy items at a flea market. The speech recognition component was performed manually, i.e. a human manually transcribed the speech. As the manual transcription takes time, the system response is noticeably delayed in a non-incremental system where the dialogue system only starts to plan its response once transcription is complete. In contrast, the incremental system constructs a response as soon as possible, based on partial input. The response is recomputed on changed input, which can have three effects: If the update is consistent with what has been said already, the continuation of what to say is simply changed (a *covert change*). If the system has to retract information already uttered, an explicit repair has to be produced (an *overt change*). If not all information to produce a complete utterance is available, fillers are inserted to avoid silence. The system is faster and preferred by users even though it has to explicitly correct itself, resulting in longer responses. Even though its output is monotonic – as it cannot erase information from the hearer's ears – its explicit repairs allow to act with low delay (a similar strategy to the one employed by human speakers (Levelt, 1989, ch. 12)). As the NLG component is rule-based, it is not constrained by the (non-) availability of data suitable for learning incremental language generation.

5 Evaluating incremental systems

An incremental system can be evaluated just as a non-incremental one. Evaluation schemata exist for all established tasks such as speech recognition (quality measured in word error rate), PoS tagging (measured in accuracy), phrase structure parsing (measured in precision/recall), or machine translation (BLEU). Additional evaluation allows to examine the behavior more closely, such as compiling error confusion matrices. These schemata enable a comparison between components in a standardized way.

	non-monotonic (1)	non-monotonic (2)	delayed (3)	erroneous (4)
input: a	a/y	a/y		a/y
b	a/x b/y	a/y b/y	a/x b/y	a/y b/y
c	a/x b/y c/z	a/x b/y c/z	a/x b/y c/z	a/y b/y c/z
inc_acc(i)	2: 1/1; 1: 2/2; 0: 2/3	2: 1/1; 1: 1/2 ; 0: 2/3	2: 1/1; 1: 2/2 ; 0: 2/3	2: 0/1; 1: 1/2 ; 0: 2/3
EO	1/4	1/2	0	0
accuracy	3/3	3/3	3/3	2/3

Table 1: Examples for characteristics of incremental output that need to be captured for evaluation. Correct output: a/x b/y c/z. (1), (2): output for *a* changed; (3): output for *a* held back until input "b" is available; (4): incorrect assignment to "a" stays in output. Dashed: exemplifies output used to compute inc_acc (incremental accuracy). EO: edit overhead. Accuracy: measured on complete output.

The downside of standardized evaluation is the lack of insight for incremental properties. When building incremental systems, not only the final output but also the intermediate behavior is of importance but using evaluations tailored to non-incremental systems fails to give insight into the incremental properties of the system. An incremental system should provide as much correct information as early as possible but using non-incremental evaluation hides all differences in this respect.

Table 1 shows abstract input/output patterns for a sequence labeling task, where the correspondence between input and output is given and no reordering effects take place. A standard accuracy-based evaluation on the complete output would yield a perfect score for the three first systems although their behavior over time is quite different: In addition to the non-incremental evaluation for the complete output, the non-monotonicity in (1) and (2) as well as the delay in (3) need to be covered. It is impossible to boil down all these differences to a single number. We will therefore have a look at distinct metrics for timeliness, monotonicity, and quality.

5.1 Measuring timeliness

Cho and Esipova (2016) propose to measure timeliness by counting for each output element t of an output sequence Y how many input elements from the input sequence X have been consumed before its production ($s(t)$). $\tau(X, Y)$ then computes the translation delay:

$$0 < \tau(X, Y) = \frac{1}{|X||Y|} \sum_{t=1}^{|Y|} s(t) \leq 1$$

This computation is helpful when there is no gold standard alignment between output and input which one could use to obtain the output timing that could be achieved under optimal conditions. $\tau = 0$ means all output was made without consuming input, $\tau = 1$ means all input was read before generating output.

Grissom II et al. (2014) introduce latency-BLEU, a metric that averages the BLEU scores of the outputs corresponding to each input prefix. The complete translation is weighed higher than all other partial translations to penalize incorrect translation. If the resulting translation is the same, a processor with less delay will obtain a higher score. It has to be noted that due to the averaging the sentence-initial output has more influence on the score than output created near the end of a sequence. It is also not possible to completely distinguish between the quality and the timeliness because quality and timeliness are measured in a single metric. In the MT system by He et al. (2015), the timeliness can be (indirectly) tuned by adjusting a threshold at which all yet untranslated words should be translated. Figure 2 (right) shows a plot of the resulting BLEU score against the average number of words translated at once, i.e. the delay. This way, potential users can see the trade-offs that can be made using a system (RW+GD is the proposed architecture, beating the other approaches at each trade-off point).

If an explicit alignment exists between the input and the output – such as in speech recognition – the difference between when an output is made (i.e. which amount of input data has been consumed) and the timing of the corresponding input can be measured to obtain an anchored timeliness measure (Baumann

et al., 2011). Both the relation to the first occurrence of an output (FO) and the relation to the last change of an output (final decision, FD) can be measured. E.g. if a word ends at 2.5 seconds of the input audio, was first recognized after consuming 3 seconds and was part of all outputs produced after consuming 4 seconds, its FO would be 0.5 seconds and its FD would be 1.5 seconds. To separate the timeliness from the quality, these measures can be computed against the final output of the system instead of the gold standard, but then a reliable alignment between the input and the generated output is needed. The advantage to other methods is its interpretability: A FO of 100ms for a speech recognizer means that it produces, on average, an output 100ms after it has consumed input that carries evidence for this output. Obviously, FO and FD only differ for non-monotonic processors; FD measures what delay is necessary on average to rely on an output produced by the processor.

5.2 Measuring incremental quality

When dealing with monotonic output, the incremental quality can be assessed using the non-incremental quality metrics as the processor can't correct previously made output. If a processor can be tuned to provide more or less timely output, the quality can be plotted against delay, as in Figure 2.

If a system is non-monotonic, it makes sense to measure the accuracy not only based on the final output but also on the intermediate output. Averaging the quality for each increment has two downsides: Output for early input is weighed more heavily than for later input, and it is unclear how the non-monotonicity affects the quality. Beuck et al. (2011b) perform an evaluation for incremental dependency parsing by computing the accuracy for the *n*-th word to the right of the frontier in every prefix, with n between zero (measuring the newest words) and five (the accuracy of the sixth-newest word). This way, an accuracy curve relative to the age of the input is generated, Table 1 shows incremental accuracy (*inc_acc*) measures relative to the age of the input; the examples (1) and (2) obtain different incremental accuracy measures even though the non-incremental accuracy is the same.

Incremental structured output might include predictions, which are not accounted for in the metrics discussed up to now. If an incremental gold standard is available (such as the one used for training discussed in § 4.3.2), the precision and recall of those predictions can be computed with respect to the predictions in the gold standard (Beuck et al., 2013).

5.3 Measuring the degree of non-monotonicity

Evaluating non-monotonicity can be viewed from two (similar) standpoints: first, how much intermediate output will be retracted again? And second: how sure can we be that a certain output is reliable, i.e. will also be part of the final output of a processor?

Baumann et al. (2009) and Baumann (2013) tackle the first question by defining the *edit overhead* generated by a non-monotonic processor producing sequential output by defining three edit operations on an output sequence: *add* (append an element to the output), *revoke* (remove the last element from the output), and *substitute* (revoke and then append). The difference $\mathit{diff}(o_i, o_j)$ between two outputs o_i and o_j can then be defined as the minimal number of edits needed to change o_i into o_j. Note that to change the first element of an output of length n, $2(n-1)+1$ operations are needed whereas changing the last element only yields a difference of one. This notation allows to compute the edit overhead: Let $N_{optimal} = |\mathit{diff}(o_0, o_{tmax})|$ be the number of edits necessary to obtain the final output and $N_{actual} = \sum_{t=1}^{tmax} \mathit{diff}(o_{t-1}, o_t)$ be the amount of edit operations actually performed. The edit overhead is then defined as the proportion of unnecessary edits produced by the processor:

$$EO = (N_{actual} - N_{optimal})/N_{actual}$$

Table 1 shows that EO can distinguish between the different levels of non-monotonicity. The edit operations can be adapted to the problem at hand, e.g. if structured output is produced or edits at the start of the sequence should not be penalized heavier than edits at the end.

Regarding the second question, Beuck et al. (2011b) propose to compute the accuracy against the output generated based on the complete input instead of computing against the gold standard to obtain a stability measure. Using the gold standard as reference measures consistency with the ground truth,

i.e. the quality, and using the complete output measures the consistency with that, i.e. the stability. This approach obviously only works if an incremental accuracy is defined for the problem at hand and that accuracy can be measured with respect to a non-incremental gold standard.

6 Combining multiple incremental processors

A NLP system usually consist of several processors. In a non-incremental system, all processors can simply form a pipeline with each processor working on the output of the previous one. This is non-trivial in an incremental system because for a given input, each processor may produce several outputs which may even be contradictory due to non-monotonicity. Therefore, a system either needs to use restart-incrementality throughout the pipeline or track changes to perform partial recomputation. Wirén (1992, ch. 5) introduces the notion of dependencies to track which parts of the input a certain output is based on in a chart parser. This way, only parts of the chart need to be recomputed given a non-monotonic change. Schlangen and Skantze (2009) vastly extend this notion to a general computation model in which multiple processors work on data which is organized in incremental units (IU). An IU stands for a minimal unit of information, such as a recognized word. IUs are grounded in other IUs from a previous level, and linked to other IUs on the same level, e.g. to describe a sequential relationship. Processors create new IUs based on their input and may revoke IUs already generated. A processor can commit to an IU to signify that this IU will not be revoked and other processors may rely on it.

7 Conclusion and outlook

Building incremental processors poses problems that are similar regardless of the domain. Decisions have to be made regarding the amount of delay acceptable, whether non-monotonic output is acceptable for downstream processors, and if so, to what extent. For all these decisions, the relation between input and output is important: Is there a clear mapping between input and output, maybe even a one-to-one mapping, and does reordering happen? Several techniques have been discussed for implementing incremental processors: Training a *gatekeeper* to either perform a trade-off between non-monotonicity and delay (§ 4.1) or – for monotonic processors – between delay and quality (§ 4.2.1), *Beam search* to create non-monotonic output with monotonic extension (§ 4.3.1), and *restart incrementality* for unrestricted non-monotonicity (§ 4.3.2). To evaluate an incremental system, ideally three characteristics should be measured: timeliness (§ 5.1), quality (§ 5.2), and non-monotonicity (§ 5.3). If a system can be tuned in regard to these characteristics, different trade-off points between these properties can be measured (§ 3.6).

There are still open problems for building incremental NLP systems: Processors generating connected structured output need incremental training data to learn intermediate structures not visible in the non-incremental gold standards; as the quality of the training data generation influences the quality of the processor, data-driven approaches producing high-quality incremental training data are needed. Explicit corrections in spoken output are preferred by users to delay (§ 4.4), which could transfer to incremental MT. However, lack of automatic evaluation likely requires an expensive human end-to-end evaluation due to the large deviation from non-incremental gold standard translations. All data-driven processors discussed are able to produce non-monotonic output, but are not able to consume non-monotonic input, a problematic discrepancy for building incremental systems out of multiple processors. To bridge the gap between certainty (monotonic output) and uncertainty (non-monotonic output), the likelihood of an output being stable could be attached to the output (see Selfridge et al. (2011) § 5). Modern NLP processors heavily rely on sub-symbolic representation and use attention mechanisms to obtain the relevant information. With this approach, an explicit grounding for partial recomputation as discussed in Section 6 does not work anymore and would need to be replaced with some notion of soft grounding.

Acknowledgments

I would like to thank Christine Köhn, Sebastian Beschke, and Timo Baumann for valuable feedback, as well as the anonymous reviewers for helpful remarks.

References

S. Aït-Mokhtar, J.-P. Chanod, and C. Roux. 2002. Robustness beyond shallowness: incremental deep parsing. *Natural Language Engineering*, 8(2-3):121–144.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Timo Baumann, Michaela Atterer, and David Schlangen. 2009. Assessing and improving the performance of speech recognition for incremental systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 380–388, Boulder, Colorado, June. Association for Computational Linguistics.

Timo Baumann, Okko Buß, and David Schlangen. 2011. Evaluation and optimisation of incremental processors. *Dialogue & Discourse*, 2(1):113–141. Special Issue on Incremental Processing in Dialogue.

Timo Baumann. 2013. *Incremental Spoken Dialogue Processing: Architecture and Lower-level Components*. Ph.D. thesis, Universität Bielefeld, Germany.

Niels Beuck, Arne Köhn, and Wolfgang Menzel. 2011a. Decision strategies for incremental pos tagging. In Bolette Sandford Pedersen, Gunta Nešpore, and Inguna Skadina, editors, *Proceedings of the 18th Nordic Conference of Computational Linguistics NODALIDA 2011*, volume 11 of *NEALT Proceedings*, pages 26–33. Northern European Association for Language Technology (NEALT).

Niels Beuck, Arne Köhn, and Wolfgang Menzel. 2011b. Incremental parsing and the evaluation of partial dependency analyses. In *Proceedings of the 1st International Conference on Dependency Linguistics*. Depling 2011.

Niels Beuck, Arne Köhn, and Wolfgang Menzel. 2013. Predictive incremental parsing and its evaluation. In Kim Gerdes, Eva Hajičová, and Leo Wanner, editors, *Computational Dependency Theory*, volume 258 of *Frontiers in Artificial Intelligence and Applications*, pages 186 – 206. IOS press.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

Kyunghyun Cho and Masha Esipova. 2016. Can neural machine translation do simultaneous translation? *CoRR*, abs/1606.02012.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546, Valencia, Spain, April. Association for Computational Linguistics.

Vera Demberg and Frank Keller. 2008. A psycholinguistically motivated version of tag. In *Proceedings of the Ninth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+9)*, pages 25–32, Tübingen, Germany, June.

Vera Demberg, Frank Keller, and Alexander Koller. 2013. Incremental, predictive parsing with psycholinguistically motivated tree-adjoining grammar. *Computational Linguistics*, 39(4):1025–1066.

David DeVault, Kenji Sagae, and David Traum. 2009. Can i finish? learning when to respond to incremental interpretation results in interactive dialogue. In *Proceedings of the SIGDIAL 2009 Conference*, pages 11–20, London, UK, September. Association for Computational Linguistics.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July. Association for Computational Linguistics.

Edward Gibson and Tessa Warren. 2004. Reading-time evidence for intermediate linguistic structure in long-distance dependencies. *Syntax*, 7(1):55–78.

Frieda Goldman-Eisler. 1972. Segmentation of input in simultaneous translation. *Journal of Psycholinguistic Research*, 1(2):127–140, Jun.

Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. 2014. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1342–1352, Doha, Qatar, October. Association for Computational Linguistics.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. 2017. Learning to translate in real-time with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1053–1062, Valencia, Spain, April. Association for Computational Linguistics.

Markus Guhe. 2007. *Incremental Conceptualization for Language Production*. Lawrence Erlbaum Associates, Inc.

Hany Hassan, Khalil Sima'an, and Andy Way. 2009. Lexicalized semi-incremental dependency parsing. In *Proceedings of the International Conference RANLP-2009*, pages 128–134, Borovets, Bulgaria, September. Association for Computational Linguistics.

He He, Alvin Grissom II, John Morgan, Jordan Boyd-Graber, and Hal Daumé III. 2015. Syntax-based rewriting for simultaneous machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 55–64, Lisbon, Portugal, September. Association for Computational Linguistics.

He He, Jordan Boyd-Graber, and Hal Daumé III. 2016. Interpretese vs. translationese: The uniqueness of human strategies in simultaneous interpretation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 971–976, San Diego, California, June. Association for Computational Linguistics.

Julian Hough and Matthew Purver. 2014. Strongly incremental repair detection. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 78–89, Doha, Qatar, October. Association for Computational Linguistics.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 69–123. Springer Berlin Heidelberg, Berlin, Heidelberg.

Martin Kay, Jean Mark Gawron, and Peter Norvig. 1994. *Verbmobil: a translation system for face-to-face dialog*. Number 33 in CSLI lecture notes. CSLI.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Arne Köhn and Timo Baumann. 2016. Predictive incremental parsing helps language modeling. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 268–277. The COLING 2016 Organizing Committee.

Arne Köhn and Wolfgang Menzel. 2014. Incremental predictive parsing with turboparser. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 803–808, Baltimore, Maryland, June. Association for Computational Linguistics.

Wilem J. M. Levelt. 1989. *Speaking: From Intention to Articulation*. The MIT Press.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA. Association for Computational Linguistics.

André Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August.

Ian McGraw and Alexander Gruenstein. 2012. Estimating word-stability during incremental speech recognition. In *Interspeech*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing LSTM language models. *CoRR*, abs/1708.02182.

Takashi Mieno, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Speed or accuracy? a study in evaluation of simultaneous speech translation. In *16th Annual Conference of the International Speech Communication Association (InterSpeech 2015)*, Dresden, Germany, September.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Hiroshi Noji and Yusuke Miyao. 2014. Left-corner transitions on dependency parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2140–2150, Dublin, Ireland, August. Dublin City University and Association for Computational Linguistics.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Maike Paetzel, Ramesh Manuvinakurike, and David DeVault. 2015. "so, which one is it?" the effect of alternative incremental architectures in a high-performance game-playing agent. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 77–86, Prague, Czech Republic, September. Association for Computational Linguistics.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Kenji Sagae, Gwen Christian, David DeVault, and David Traum. 2009. Towards natural language understanding of partial speech recognition results in dialogue systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 53–56, Boulder, Colorado, June. Association for Computational Linguistics.

David Schlangen and Gabriel Skantze. 2009. A general, abstract model of incremental dialogue processing. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 710–718, Athens, Greece, March. Association for Computational Linguistics.

David Schlangen, Timo Baumann, and Michaela Atterer. 2009. Incremental Reference Resolution: The Task, Metrics for Evaluation, and a Bayesian Filtering Model that is Sensitive to Disfluencies. In *Proceedings of SigDial 2009*, London, UK.

Ethan Selfridge, Iker Arizmendi, Peter Heeman, and Jason Williams. 2011. Stability and accuracy in incremental speech recognition. In *Proceedings of the SIGDIAL 2011 Conference*, pages 110–119, Portland, Oregon, June. Association for Computational Linguistics.

Gabriel Skantze and Anna Hjalmarsson. 2013. Towards incremental speech generation in conversational systems. *Computer Speech & Language*, 27(1):243 – 262. Special issue on Paralinguistics in Naturalistic Speech and Language.

Mark Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

Scott C. Stoness, James Allen, Greg Aist, and Mary Swift. 2005. Using real-world reference to improve spoken language understanding. In *AAAI Workshop on Spoken Language Understanding*, pages 38–45.

Patrick Sturt and Vincent Lombardo. 2005. Processing coordinated structures: Incrementality and connectedness. *Cognitive Science*, 29:291–305.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Greedy, joint syntactic-semantic parsing with stack lstms. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 187–197. Association for Computational Linguistics, June.

MK Tanenhaus, MJ Spivey-Knowlton, KM Eberhard, and JC Sedivy. 1995. Integration of visual and linguistic information in spoken language comprehension. *Science*, 268(5217):1632–1634.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6000–6010. Curran Associates, Inc.

Titus von der Malsburg and Shravan Vasishth. 2011. What is the scanpath signature of syntactic reanalysis? *Journal of Memory and Language*, 65(2):109 – 127.

Wolfgang Wahlster, editor. 2000. *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer-Verlag Berlin Heidelberg.

Mats Wirén. 1992. *Studies in Incremental Natural-Language Analysis*. Ph.d. thesis, Linköping University.

Junsheng Zhou, Feiyu Xu, Hans Uszkoreit, Weiguang QU, Ran Li, and Yanhui Gu. 2016. Amr parsing with an incremental joint model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 680–689, Austin, Texas, November. Association for Computational Linguistics.