First- and Second-Order Expectation Semirings
with Applications to Minimum-Risk Training on Translation Forests[*]

Zhifei Li and **Jason Eisner**

Department of Computer Science and Center for Language and Speech Processing
Johns Hopkins University, Baltimore, MD 21218, USA
`zhifei.work@gmail.com, jason@cs.jhu.edu`

Abstract

Many statistical translation models can be regarded as weighted logical deduction. Under this paradigm, we use weights from the expectation semiring (Eisner, 2002), to compute first-order statistics (e.g., the expected hypothesis length or feature counts) over packed forests of translations (lattices or hypergraphs). We then introduce a novel *second-order* expectation semiring, which computes second-order statistics (e.g., the variance of the hypothesis length or the gradient of entropy). This second-order semiring is essential for many interesting training paradigms such as minimum risk, deterministic annealing, active learning, and semi-supervised learning, where gradient descent optimization requires computing the gradient of entropy or risk. We use these semirings in an open-source machine translation toolkit, **Joshua**, enabling minimum-risk training for a benefit of up to 1.0 BLEU point.

1 Introduction

A hypergraph or "packed forest" (Gallo et al., 1993; Klein and Manning, 2004; Huang and Chiang, 2005) is a compact data structure that uses structure-sharing to represent exponentially many trees in polynomial space. A *weighted* hypergraph also defines a probability or other weight for each tree, and can be used to represent the hypothesis space considered (for a given input) by a monolingual parser or a tree-based translation system, e.g., tree to string (Quirk et al., 2005; Liu et al., 2006), string to tree (Galley et al., 2006), tree to tree (Eisner, 2003), or string to string with latent tree structures (Chiang, 2007).

Given a hypergraph, we are often interested in computing some quantities over it using dynamic programming algorithms. For example, we may want to run the Viterbi algorithm to find the most probable derivation tree in the hypergraph, or the k most probable trees. Semiring-weighted logic programming is a general framework to specify these algorithms (Pereira and Warren, 1983; Shieber et al., 1994; Goodman, 1999; Eisner et al., 2005; Lopez, 2009). Goodman (1999) describes many useful semirings (e.g., Viterbi, inside, and Viterbi-n-best). While most of these semirings are used in "testing" (i.e., decoding), we are mainly interested in the semirings that are useful for "training" (i.e., parameter estimation). The expectation semiring (Eisner, 2002), originally proposed for finite-state machines, is one such "training" semiring, and can be used to compute feature expectations for the E-step of the EM algorithm, or gradients of the likelihood function for gradient descent.

In this paper, we apply the expectation semiring (Eisner, 2002) to a hypergraph (or packed forest) rather than just a lattice. We then propose a novel *second-order* expectation semiring, nicknamed the "variance semiring."

The original first-order expectation semiring allows us to efficiently compute a vector of first-order statistics (expectations; first derivatives) on the set of paths in a lattice or the set of trees in a hypergraph. The second-order expectation semiring *additionally* computes a matrix of second-order statistics (expectations of *products*; second derivatives (Hessian); derivatives of expectations).

We present details on how to compute many interesting quantities over the hypergraph using the expectation and variance semirings. These quantities include expected hypothesis length, feature expectation, entropy, cross-entropy, Kullback-Leibler divergence, Bayes risk, variance of hypothesis length, gradient of entropy and Bayes risk, covariance and Hessian matrix, and so on. The variance semiring is essential for many interesting training paradigms such as deterministic

annealing (Rose, 1998), minimum risk (Smith and Eisner, 2006), active and semi-supervised learning (Grandvalet and Bengio, 2004; Jiao et al., 2006). In these settings, we must compute the gradient of entropy or risk. The semirings can also be used for second-order gradient optimization algorithms.

We implement the expectation and variance semirings in **Joshua** (Li et al., 2009a), and demonstrate their practical benefit by using minimum-risk training to improve Hiero (Chiang, 2007).

2 Semiring Parsing on Hypergraphs

We use a specific tree-based system called Hiero (Chiang, 2007) as an example, although the discussion is general for any systems that use a hypergraph to represent the hypothesis space.

2.1 Hierarchical Machine Translation

In Hiero, a synchronous context-free grammar (SCFG) is extracted from automatically word-aligned corpora. An illustrative grammar rule for Chinese-to-English translation is

$$X \rightarrow \langle X_0 \text{ 的 } X_1, X_1 \text{ of } X_0 \rangle,$$

where the Chinese word 的 means *of*, and the alignment, encoded via subscripts on the nonterminals, causes the two phrases around 的 to be reordered around *of* in the translation. Given a source sentence, Hiero uses a CKY parser to generate a hypergraph, encoding many derivation trees along with the translation strings.

2.2 Hypergraphs

Formally, a hypergraph is a pair $\langle V, E \rangle$, where V is a set of *nodes* (vertices) and E is a set of *hyperedges*, with each hyperedge connecting a *set* of *antecedent nodes* to a single *consequent node*.[1] In parsing parlance, a node corresponds to an *item* in the chart (which specifies aligned spans of input and output together with a nonterminal label). The root node corresponds to the *goal item*. A hyperedge represents an SCFG rule that has been "instantiated" at a particular position, so that the nonterminals on the right and left sides have been replaced by particular antecedent and consequent items; this corresponds to storage of backpointers in the chart.

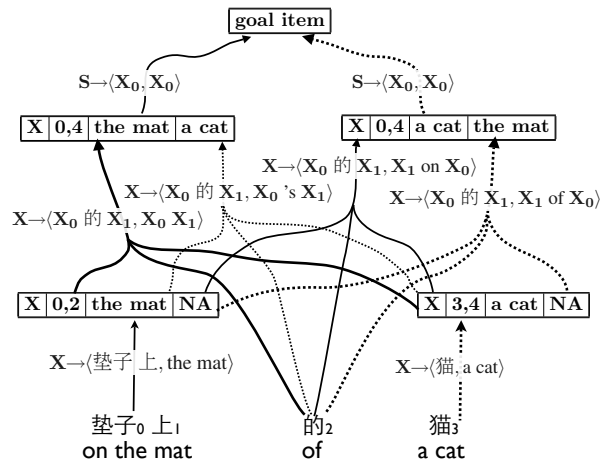We write $T(e)$ to denote the set of antecedent nodes of a hyperedge e. We write $I(v)$ for the

Figure 1: A toy hypergraph in Hiero. When generating the hypergraph, a trigram language model is integrated. Rectangles represent items, where each item is identified by the non-terminal symbol, source span, and left- and right-side language model states. An item has one or more incoming hyperedges. A hyperedge consists of a rule, and a pointer to an antecedent item for each non-terminal symbol in the rule.

set of *incoming hyperedges* of node v (i.e., hyperedges of which v is the consequent), which represent different ways of deriving v. Figure 1 shows a simple Hiero-style hypergraph. The hypergraph encodes four different derivation trees that share some of the same items. By exploiting this sharing, a hypergraph can compactly represent exponentially many trees.

We observe that any finite-state automaton can also be encoded as a hypergraph (in which every hyperedge is an ordinary edge that connects a *single* antecedent to a consequent). Thus, the methods of this paper apply directly to the simpler case of hypothesis lattices as well.

2.3 Semiring Parsing

We assume a hypergraph HG, which compactly encodes many derivation trees $d \in \mathrm{D}$. Given HG, we wish to extract the best derivations—or other aggregate properties of the forest of derivations. Semiring parsing (Goodman, 1999) is a general framework to describe such algorithms. To define a particular algorithm, we choose a semiring K and specify a "weight" $k_e \in K$ for each hyperedge e. The desired aggregate result then emerges as the *total weight* of all derivations in the hypergraph. For example, to simply count derivations, one can assign every hyperedge weight 1 in the semiring of *ordinary integers*; then each derivation also has weight 1, and their total weight is the number of derivations.

We write $\mathbf{K} = \langle K, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ for a semiring with elements K, additive operation $\oplus$, multi-

---

[1] Strictly speaking, making each hyperedge designate a single consequent defines a *B-hypergraph* (Gallo et al., 1993).

plicative operation $\otimes$, additive identity $\mathbf{0}$, and multiplicative identity $\mathbf{1}$. The $\otimes$ operation is used to obtain the weight of each derivation d by *multiplying* the weights of its component hyperedges e, that is, $k_d = \bigotimes_{e \in d} k_e$. The $\oplus$ operation is used to *sum* over all derivations d in the hypergraph to obtain the *total* weight of the hypergraph HG, which is $\bigoplus_{d \in D} \bigotimes_{e \in d} k_e$.[2] Figure 2 shows how to compute the total weight of an acyclic hypergraph HG.[3] In general, the total weight is a sum over exponentially many derivations d. But Figure 2 sums over these derivations in time only linear on the size of the hypergraph. Its correctness relies on axiomatic properties of the semiring: namely, $\oplus$ is associative and commutative with identity $\mathbf{0}$, $\otimes$ is associative with two-sided identity $\mathbf{1}$, and $\otimes$ distributes over $\oplus$ from both sides. The distributive property is what makes Figure 2 work. The other properties are necessary to ensure that $\bigoplus_{d \in D} \bigotimes_{e \in d} k_e$ is well-defined.[4]

The algorithm in Figure 2 is general and can be applied with any semiring (e.g., Viterbi). Below, we present our novel semirings.

3 Finding Expectations on Hypergraphs

We now introduce the computational problems of this paper and the semirings we use to solve them.

3.1 Problem Definitions

We are given a function $p : D \to \mathbb{R}_{\geq 0}$, which decomposes **multiplicatively** over component hyperedges e of a derivation $d \in D$: that is, $p(d) \stackrel{\text{def}}{=} \prod_{e \in d} p_e$. In practice, $p(d)$ will specify a probability distribution over the derivations in the hyper-

---

[2]Eisner (2002) uses *closed semirings* that are also equipped with a Kleene closure operator *. For example, in the real semiring $\langle \mathbb{R}, +, \times, 0, 1 \rangle$, we define $p^* = (1 - p)^{-1}$ ($= 1 + p + p^2 + \ldots$) for $|p| < 1$ and is undefined otherwise. The closure operator enables exact summation over the *infinitely* many paths in a cyclic FSM, or trees in a hypergraph with non-branching cycles, without the need to iterate around cycles to numerical convergence. For completeness, we specify the closure operator for our semirings, satisfying the axioms $k^* = 1 \oplus k \otimes k^* = 1 \oplus k^* \otimes k$, but we do not use it in our experiments since our hypergraphs are acyclic.

[3]We assume that HG has *already* been built by deductive inference (Shieber et al., 1994). But in practice, the nodes' inside weights $\beta(v)$ are usually accumulated *as the hypergraph is being built*, so that pruning heuristics can consult them.

[4]Actually, the notation $\bigotimes_{e \in d} k_e$ assumes that $\otimes$ is commutative as well, as does the notation "for $u \in T(e)$" in our algorithms; neither specifies a loop order. One could however use a non-commutative semiring by ordering each hyperedge's antecedents and specifying that a derivation's weight is the product of the weights of its hyperedges *when visited in prefix order*. Tables 1–2 will not assume any commutativity.

$\text{INSIDE}(\text{HG}, \mathbf{K})$

1 **for** v **in** topological order on HG $\triangleright$ each node
2      $\triangleright$ find $\beta(v) \leftarrow \bigoplus_{e \in I(v)} (k_e \otimes (\bigotimes_{u \in T(e)} \beta(u)))$
3      $\beta(v) \leftarrow \mathbf{0}$
4   **for** $e \in I(v)$ $\triangleright$ each incoming hyperedge
5          $k \leftarrow k_e$ $\triangleright$ hyperedge weight
6   **for** $u \in T(e)$ $\triangleright$ each antecedent node
7              $k \leftarrow k \otimes \beta(u)$
8          $\beta(v) \leftarrow \beta(v) \oplus k$
9 **return** $\beta(\text{root})$

Figure 2: Inside algorithm for an acyclic hypergraph HG, which provides hyperedge weights $k_e \in \mathbf{K}$. This computes all "inside weights" $\beta(v) \in \mathbf{K}$, and returns $\beta(\text{root})$, which is total weight of the hypergraph, i.e., $\bigoplus_{d \in D} \bigotimes_{e \in d} k_e$.

$\text{OUTSIDE}(\text{HG}, \mathbf{K})$

1 **for** v **in** HG
2      $\alpha(v) \leftarrow \mathbf{0}$
3 $\alpha(\text{root}) \leftarrow \mathbf{1}$
4 **for** v **in** reverse topological order on HG
5   **for** $e \in I(v)$ $\triangleright$ each incoming hyperedge
6   **for** $u \in T(e)$ $\triangleright$ each antecedent node
7              $\alpha(u) \leftarrow \alpha(u) \oplus (\alpha(v) \otimes k_e \otimes$
8                  $\bigotimes_{w \in T(e), w \neq u} \beta(w))$

Figure 3: Computes the "outside weights" $\alpha(v)$. Can only be run after $\text{INSIDE}(\text{HG})$ of Figure 2 has already computed the inside weights $\beta(v)$.

graph. It is often convenient to permit this probability distribution to be unnormalized, i.e., one may have to divide it through by some Z to get a proper distribution that sums to 1.

We are also given two functions of interest $r, s : D \to \mathbb{R}$, each of which decomposes **additively** over its component hyperedges e: that is, $r(d) \stackrel{\text{def}}{=} \sum_{e \in d} r_e$, and $s(d) \stackrel{\text{def}}{=} \sum_{e \in d} s_e$.

We are now interested in computing the following quantities on the hypergraph HG:

$$Z \stackrel{\text{def}}{=} \sum_{d \in D} p(d) \tag{1}$$

$$\overline{r} \stackrel{\text{def}}{=} \sum_{d \in D} p(d) r(d) \tag{2}$$

$$\overline{s} \stackrel{\text{def}}{=} \sum_{d \in D} p(d) s(d) \tag{3}$$

$$\overline{t} \stackrel{\text{def}}{=} \sum_{d \in D} p(d) r(d) s(d) \tag{4}$$

Note that $\overline{r}/Z$, $\overline{s}/Z$, and $\overline{t}/Z$ are expectations under p of $r(d)$, $s(d)$, and $r(d)s(d)$, respectively.

More formally, the probabilistic interpretation is that D is a discrete sample space (consisting

INSIDE-OUTSIDE(HG, **K**, X)

1   ▷ Run inside and outside on HG with only k_e weights
2    $\hat{k} \leftarrow$ INSIDE(HG, **K**)   ▷ see Figure 2
3   OUTSIDE(HG, **K**)   ▷ see Figure 3
4   ▷ Do a single linear combination to get $\hat{x}$
5    $\hat{x} \leftarrow \mathbf{0}$
6   **for** v **in** HG   ▷ each node
7   **for** $e \in I(v)$ ▷ each incoming hyperedge
8          $\overline{k_e} \leftarrow \alpha(v)$
9   **for** $u \in T(e)$ ▷ each antecedent node
10            $\overline{k_e} \leftarrow \overline{k_e}\, \beta(u)$
11          $\hat{x} \leftarrow \hat{x} + (\overline{k_e}\, x_e)$
12   **return** $\langle \hat{k}, \hat{x} \rangle$

Figure 4: If every hyperedge specifies a weight $\langle k_e, x_e \rangle$ in some expectation semiring $\mathbb{E}_{\mathbf{K}, X}$, then this inside-outside algorithm is a more efficient alternative to Figure 2 for computing the total weight $\langle \hat{k}, \hat{x} \rangle$ of the hypergraph, especially if the x_e are vectors. First, at lines 2–3, the inside and outside algorithms are run using only the k_e weights, obtaining only $\hat{k}$ (without $\hat{x}$) but also obtaining all inside and outside weights $\beta, \alpha \in \mathbf{K}$ as a side effect. Then the second component $\hat{x}$ of the total weight is accumulated in lines 5–11 as a linear combination of all the x_e values, namely $\hat{x} = \sum_e \overline{k_e} x_e$, where $\overline{k_e}$ is computed at lines 8–10 using α and β weights. The linear coefficient $\overline{k_e}$ is the "exclusive weight" for hyperedge e, meaning that the *product* $\overline{k_e} k_e$ is the total weight in **K** of all derivations $d \in \mathrm{D}$ that include e.

of all derivations in the hypergraph), p is a measure over this space, and $r, s : \mathrm{D} \to \mathbb{R}$ are random variables. Then $\overline{r}/Z$ and $\overline{s}/Z$ give the expectations of these random variables, and $\overline{t}/Z$ gives the expectation of their product $t = rs$, so that $\overline{t}/Z - (\overline{r}/Z)(\overline{s}/Z)$ gives their covariance.

Example 1: $r(d)$ is the length of the translation corresponding to derivation d (arranged by setting r_e to the number of target-side terminal words in the SCFG rule associated with e). Then $\overline{r}/Z$ is the expected hypothesis length. ***Example 2:*** $r(d)$ evaluates the loss of d compared to a reference translation, using some additively decomposable loss function. Then $\overline{r}/Z$ is the risk (expected loss), which is useful in minimum-risk training. ***Example 3:*** $r(d)$ is the number of times that a certain feature fires on d. Then $\overline{r}/Z$ is the expected feature count, which is useful in maximum-likelihood training. We will generalize later in Section 4 to allow $r(d)$ to be a vector of features. ***Example 4:*** Suppose $r(d)$ and $s(d)$ are identical and both compute hypothesis length. Then the second-order statistic $\overline{t}/Z$ is the second moment of the length distribution, so the variance of hypothesis length can be found as $\overline{t}/Z - (\overline{r}/Z)^2$.

3.2 Computing the Quantities

We will use the semiring parsing framework to compute the quantities (1)–(4). Although each is a sum over exponentially many derivations, we will compute it in $O(|\mathrm{HG}|)$ time using Figure 2.

In the simplest case, let $\mathbf{K} = \langle \mathbb{R}, +, \times, 0, 1 \rangle$, and define $k_e = p_e$ for each hyperedge e. Then the algorithm of Figure 2 reduces to the classical inside algorithm (Baker, 1979) and computes Z.

Next suppose **K** is the expectation semiring (Eisner, 2002), shown in Table 1. Define $k_e = \langle p_e, p_e r_e \rangle$. Then Figure 2 will return $\langle Z, \overline{r} \rangle$.

Finally, suppose **K** is our novel *second-order* expectation semiring, which we introduce in Table 2. Define $k_e = \langle p_e, p_e r_e, p_e s_e, p_e r_e s_e \rangle$. Then the algorithm of Figure 2 returns $\langle Z, \overline{r}, \overline{s}, \overline{t} \rangle$. Note that, to compute $\overline{t}$, one cannot simply construct a *first-order* expectation semiring by defining $t(d) \overset{\text{def}}{=} r(d)s(d)$ because $t(d)$, unlike $r(d)$ and $s(d)$, is not additively decomposable over the hyperedges in d.[5] Also, when $r(d)$ and $s(d)$ are identical, the second-order expectation semiring allows us to compute variance as $\overline{t}/Z - (\overline{r}/Z)^2$, which is why we may call our second-order expectation semiring the **variance semiring**.

3.3 Correctness of the Algorithms

To prove our claim about the first-order expectation semiring, we first observe that the definitions in Table 1 satisfy the semiring axioms. The reader can easily check these axioms (as well as the closure axioms in footnote 2). With a valid semiring, we then simply observe that Figure 2 returns the total weight $\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d} k_e = \bigoplus_{d \in \mathrm{D}} \langle p(d), p(d)r(d) \rangle = \langle Z, \overline{r} \rangle$. It is easy to verify the second equality from the definitions of $\oplus$, Z, and $\overline{r}$. The first equality requires proving that $\bigotimes_{e \in d} k_e = \langle p(d), p(d)r(d) \rangle$ from the definitions of $\otimes, k_e$, $p(d)$, and $r(d)$. The main intuition is that $\otimes$ can be used to build up $\langle p(d), p(d)r(d) \rangle$ inductively from the k_e: if d decomposes into two disjoint sub-derivations d_1, d_2, then $\langle p(d), p(d)r(d) \rangle = \langle p(d_1)p(d_2), p(d_1)p(d_2)(r(d_1) + r(d_2)) \rangle = \langle p(d_1), p(d_1)r(d_1) \rangle \otimes \langle p(d_2), p(d_2)r(d_2) \rangle$. The base cases are where d is a single hyperedge e, in which case $\langle p(d), p(d)r(d) \rangle = k_e$ (thanks to our choice of k_e), and where d is empty, in which case

---

[5]However, in a more tricky way, the second-order expectation semiring can be constructed using the first-order expectation semiring, as will be seen in Section 4.3.

Element	$\langle p, r \rangle$
$\langle p_1, r_1 \rangle \otimes \langle p_2, r_2 \rangle$	$\langle p_1 p_2,\ p_1 r_2 + p_2 r_1 \rangle$
$\langle p_1, r_1 \rangle \oplus \langle p_2, r_2 \rangle$	$\langle p_1 + p_2,\ r_1 + r_2 \rangle$
$\langle p, r \rangle^*$	$\langle p^*, p^* p^* r \rangle$
0	$\langle 0, 0 \rangle$
1	$\langle 1, 0 \rangle$

Table 1: **Expectation semiring**: Each element in the semiring is a **pair** $\langle p, r \rangle$. The second and third rows define the operations between two elements $\langle p_1, r_1 \rangle$ and $\langle p_2, r_2 \rangle$, and the last two rows define the identities. Note that the multiplicative identity **1** has an r component of 0.

s_a s_b	$a + b$		$a \cdot b$	
	s_{a+b}	ℓ_{a+b}	$s_{a \cdot b}$	$\ell_{a \cdot b}$
+ +	+	$\ell_a + \log(1 + e^{\ell_b - \ell_a})$	+	$\ell_a + \ell_b$
+ -	+	$\ell_a + \log(1 - e^{\ell_b - \ell_a})$	-	$\ell_a + \ell_b$
- +	-	$\ell_a + \log(1 - e^{\ell_b - \ell_a})$	-	$\ell_a + \ell_b$
- -	-	$\ell_a + \log(1 + e^{\ell_b - \ell_a})$	+	$\ell_a + \ell_b$

Table 3: **Storing signed values in log domain**: each value a ($= s_a e^{\ell_a}$) is stored as a **pair** $\langle s_a, \ell_a \rangle$ where s_a and ℓ_a are the **sign bit** of a and **natural logarithm** of $|a|$, respectively. This table shows the operations between two values $a = s_a 2^{\ell_a}$ and $b = s_b 2^{\ell_b}$, assuming $\ell_a \geq \ell_b$. *Note*: $\log(1 + x)$ (where $|x| < 1$) should be computed by the Mercator series $x - x^2/2 + x^3/3 - \cdots$, e.g., using the math library function log1p.

$\langle p(d), p(d)r(d) \rangle = \mathbf{1}$. It follows by induction that $\langle p(d), p(d)r(d) \rangle = \bigotimes_{e \in d} k_e$.

The proof for the second-order expectation semiring is similar. In particular, one mainly needs to show that $\bigotimes_{e \in d} k_e = \langle p(d), p(d)r(d), p(d)s(d), p(d)r(d)s(d) \rangle$.

3.4 Preventing Underflow/Overflow

In Tables 1–2, we do not discuss how to store p, r, s, and t. If p is a probability, it often suffers from the underflow problem. r, s, and t may suffer from both underflow and overflow problems, depending on their scales.

To address these, we could represent p in the log domain as usual. However, r, s, and t can be positive or negative, and we cannot directly take the log of a negative number. Therefore, we represent real numbers as ordered pairs. Specifically, to represent $a = s_a e^{\ell_a}$, we store $\langle s_a, \ell_a \rangle$, where the $s_a \in \{+, -\}$ is the **sign bit** of a and the floating-point number ℓ_a is the **natural logarithm** of $|a|$.[6] Table 3 shows the "·" and "+"operations.

---

[6]An alternative that avoids log and exp is to store $a = f_a 2^{e_a}$ as $\langle f_a, e_a \rangle$, where f_a is a *floating-point number* and e_a is a sufficiently wide *integer*. E.g., combining a 32-bit f_a with a 32-bit e_a will in effect extend f_a's 8-bit internal exponent to 32 bits by adding e_a to it. This gives much more dynamic range than the 11-bit exponent of a 64-bit double-precision floating-point number, if vastly less than in Table 3.

4 Generalizations and Speedups

In this section, we generalize beyond the above case where p, r, s are $\mathbb{R}$-valued. In general, p may be an element of some other semiring, and r and s may be **vectors** or other algebraic objects.

When r and s are vectors, especially high-dimensional vectors, the basic "inside algorithm" of Figure 2 will be slow. We will show how to speed it up with an "inside-outside algorithm."

4.1 Allowing Feature Vectors and More

In general, for P, R, S, T, we can define the first-order expectation semiring $\mathbb{E}_{P,R} = \langle P \times R, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ and the second-order expectation semiring $\mathbb{E}_{P,R,S,T} = \langle P \times R \times S \times T, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$, using the definitions from Tables 1–2. But do those definitions remain meaningful, and do they continue to satisfy the semiring axioms?

Indeed they do when $P = \mathbb{R}, R = \mathbb{R}^n, S = \mathbb{R}^m, T = \mathbb{R}^{n \times m}$, with rs defined as the outer product $rs^{\mathbf{T}}$ (a matrix) where $s^{\mathbf{T}}$ is the transpose of s. In this way, the second-order semiring $\mathbb{E}_{P,R,S,T}$ lets us take expectations of vectors and outer products of vectors. So we can find *means* and *covariances* of any number of linearly decomposable quantities (e.g., feature counts) defined on the hypergraph.

We will consider some other choices in Sections 4.3–4.4 below. Thus, for generality, we conclude this section by stating the precise technical conditions needed to construct $\mathbb{E}_{P,R}$ and $\mathbb{E}_{P,R,S,T}$:

- P is a semiring
- R is a P-module (e.g, a vector space), meaning that it comes equipped with an associative and commutative addition operation with an identity element 0, and also a multiplication operation $P \times R \to R$, such that $p(r_1 + r_2) = pr_1 + pr_2$, $(p_1 + p_2)r = p_1 r + p_2 r$, $p_1(p_2 r) = (p_1 p_2)r$
- S and T are also P-modules
- there is a multiplication operation $R \times S \to T$ that is bilinear, i.e., $(r_1 + r_2)s = r_1 s + r_2 s$, $r(s_1 + s_2) = rs_1 + rs_2$, $(pr)s = p(rs)$, $r(ps) = p(rs)$

As a matter of notation, note that above and in Tables 1–2, we overload "+" to denote any of the addition operations within P, R, S, T; overload "0" to denote their respective additive identities; and overload concatenation to denote any of the multiplication operations within or between

Element	$\langle p, r, s, t \rangle$
$\langle p_1, r_1, s_1, t_1 \rangle \otimes \langle p_2, r_2, s_2, t_2 \rangle$	$\langle p_1 p_2,\ p_1 r_2 + p_2 r_1,\ p_1 s_2 + p_2 s_1,$ $p_1 t_2 + p_2 t_1 + r_1 s_2 + r_2 s_1 \rangle$
$\langle p_1, r_1, s_1, t_1 \rangle \oplus \langle p_2, r_2, s_2, t_2 \rangle$	$\langle p_1 + p_2,\ r_1 + r_2,\ s_1 + s_2,\ t_1 + t_2 \rangle$
$\langle p, r, s, t \rangle^*$	$\langle p^*,\ p^* p^* r,\ p^* p^* s,\ p^* p^* (p^* rs + p^* rs + t) \rangle$
0	$\langle 0, 0, 0, 0 \rangle$
1	$\langle 1, 0, 0, 0 \rangle$

Table 2: **Second-order expectation semiring** (variance semiring): Each element in the semiring is a **4-tuple** $\langle p, r, s, t \rangle$. The second and third rows define the operations between two elements $\langle p_1, r_1, s_1, t_1 \rangle$ and $\langle p_2, r_2, s_2, t_2 \rangle$, while the last two rows define the identities. Note that the multiplicative identity **1** has r, s and t components of 0.

P, R, S, T. "1" refers to the multiplicative identity of P. We continue to use distinguished symbols $\oplus, \otimes, \mathbf{0}, \mathbf{1}$ for the operations and identities in our "main semiring of interest," $\mathbb{E}_{P,R}$ or $\mathbb{E}_{P,R,S,T}$.

To compute equations (1)–(4) in this more general setting, we must still require multiplicative or additive decomposability, defining $p(d) \stackrel{\text{def}}{=} \prod_{e \in d} p_e$, $r(d) \stackrel{\text{def}}{=} \sum_{e \in d} r_e$, $s(d) \stackrel{\text{def}}{=} \sum_{e \in d} s_e$ as before. But the $\prod$ and $\sum$ operators here now denote appropriate operations within P, R, and S respectively (rather than the usual operations within $\mathbb{R}$).

4.2 Inside-Outside Speedup for First-Order Expectation Semirings

Under the first-order expectation semiring $\mathbb{E}_{\mathbb{R}, \mathbb{R}^n}$, the inside algorithm of Figure 2 will return $\langle Z, \overline{r} \rangle$ where $\overline{r}$ is a vector of n feature expectations.

However, Eisner (2002, section 5) observes that this is inefficient when n is large. Why? The inside algorithm takes the trouble to compute an inside weight $\beta(v) \in \mathbb{R} \times \mathbb{R}^n$ for *each* node v in the hypergraph (or lattice). The second component of $\beta(v)$ is a presumably *dense* vector of all features that fire in all subderivations rooted at node v. Moreover, as $\beta(v)$ is computed in lines 3–8, that vector is built up (via the $\otimes$ and $\oplus$ operations of Table 1) as a linear combination of other dense vectors (the second components of the various $\beta(u)$). These vector operations can be slow.

A much more efficient approach (usually) is the traditional inside-outside algorithm (Baker, 1979).[7] Figure 4 generalizes the inside-outside algorithm to work with any expectation semiring $\mathbb{E}_{\mathbf{K}, X}$.[8] We are given a hypergraph HG whose edges have weights $\langle k_e, x_e \rangle$ in this semiring (so

now $k_e \in \mathbf{K}$ denotes only *part* of the edge weight, not all of it). INSIDE-OUTSIDE(HG, $\mathbf{K}$, X) finds $\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d} \langle k_e, x_e \rangle$, which has the form $\langle \hat{k}, \hat{x} \rangle$.

But, INSIDE(HG, $\mathbb{E}_{\mathbf{K}, X}$) could accomplish the same thing. So what makes the inside-outside algorithm more efficient? It turns out that $\hat{x}$ can be found quickly as a *single* linear combination $\sum_e \overline{k_e} x_e$ of just the feature vectors x_e that appear on *individual* hyperedges—typically a sum of very sparse vectors! And the linear coefficients $\overline{k_e}$, as well as $\hat{k}$, are computed entirely within the cheap semiring $\mathbf{K}$. They are based on β and α values obtained by first running INSIDE(HG, $\mathbf{K}$) and OUTSIDE(HG, $\mathbf{K}$), which use *only* the k_e part of the weights and ignore the more expensive x_e.

It is noteworthy that the expectation semiring is not used at all by Figure 4. Although the return value $\langle \hat{k}, \hat{x} \rangle$ is in the expectation semiring, it is built up not by $\oplus$ and $\otimes$ but rather by computing $\hat{k}$ and $\hat{x}$ separately. One might therefore wonder why the expectation semiring and its operations are still needed. One reason is that the input to Figure 4 consists of hyperedge weights $\langle k_e, x_e \rangle$ in the expectation semiring—and these weights may well have been constructed using $\otimes$ and $\oplus$. For example, Eisner (2002) uses finite-state operations such as composition, which do combine weights entirely within the expectation semiring before their result is passed to the forward-backward algorithm. A second reason is that when we work with a *second-order* expectation semiring in Section 4.4 below, the $\hat{k}, \beta$, and α values in Figure 4 will turn out to be elements of a first-order expectation semiring, and *they* must still be constructed by first-order $\otimes$ and $\oplus$, via calls to Figures 2–3.

Why does inside-outside work? Whereas the inside algorithm computes $\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d}$ in any semiring, the inside-outside algorithm exploits the special structure of an expectation semiring. By that semiring's definitions of $\oplus$ and $\otimes$ (Table 1), $\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d} \langle k_e, x_e \rangle$ can be found as

---

[7] Note, however, that the expectation semiring requires *only* the forward/inside pass to compute expectations, and thus it is more efficient than the traditional inside-outside algorithm (which requires two passes) if we are interested in computing only a small number of quantities.

[8] This follows Eisner (2002), who similarly generalized the forward-backward algorithm.

$\langle \sum_{d\in D} \prod_{e\in d} k_e, \sum_{d\in D} \sum_{e\in d} (\prod_{e'\in d, e'\neq e} k_{e'}) x_e \rangle$. The first component (giving $\hat{k}$) is found by calling the inside algorithm on just the k_e part of the weights. The second component (giving $\hat{x}$) can be rearranged into $\sum_e \sum_{d:\, e\in d} (\prod_{e'\in d, e'\neq e} k_{e'}) x_e = \sum_e \overline{k_e} x_e$, where $\overline{k_e} \stackrel{\text{def}}{=} \sum_{d:\, e\in d} (\prod_{e'\in d, e'\neq e} k_{e'})$ is found from β, α.

The application described at the start of this subsection is the classical inside-outside algorithm. Here $\langle k_e, x_e \rangle \stackrel{\text{def}}{=} \langle p_e, p_e r_e \rangle$, and the algorithm returns $\langle \hat{k}, \hat{x} \rangle = \langle Z, \overline{r} \rangle$. In fact, that $\hat{x} = \overline{r}$ can be seen directly: $\overline{r} = \sum_d p(d) r(d) = \sum_d p(d) (\sum_{e\in d} r_e) = \sum_e \sum_{d:\, e\in d} p(d) r_e = \sum_e (\overline{k_e} k_e) r_e = \sum_e \overline{k_e} x_e = \hat{x}$. This uses the fact that $\overline{k_e} k_e = \sum_{d:\, e\in d} p(d)$.

4.3 Lifting Trick for Second-Order Semirings

We now observe that the second-order expectation semiring $\mathbb{E}_{P,R,S,T}$ can be obtained indirectly by nesting one first-order expectation semiring inside another! First "lift" P to obtain the first-order expectation semiring $\mathbf{K} \stackrel{\text{def}}{=} \mathbb{E}_{P,R}$. Then lift this a second time to obtain the "nested" first-order expectation semiring $\mathbb{E}_{\mathbf{K},X} = \mathbb{E}_{(\mathbb{E}_{P,R}),(S\times T)}$, where we equip $X \stackrel{\text{def}}{=} S \times T$ with the operations $\langle s_1, t_1 \rangle + \langle s_2, t_2 \rangle \stackrel{\text{def}}{=} \langle s_1 + s_2, t_1 + t_2 \rangle$ and $\langle p, r \rangle \langle s, t \rangle \stackrel{\text{def}}{=} \langle ps, pt + rs \rangle$. The resulting first-order expectation semiring has elements of the form $\langle \langle p, r \rangle, \langle s, t \rangle \rangle$. Table 4 shows that it is indeed isomorphic to $\mathbb{E}_{P,R,S,T}$, with corresponding elements $\langle p, r, s, t \rangle$.

This construction of the second-order semiring as a first-order semiring is a useful bit of abstract algebra, because it means that known properties of first-order semirings will also apply to second-order ones. First of all, we are immediately guaranteed that the second-order semiring satisfies the semiring axioms. Second, we can directly apply the inside-outside algorithm there, as we now see.

4.4 Inside-Outside Speedup for Second-Order Expectation Semirings

Given a hypergraph weighted by a *second-order* expectation semiring $\mathbb{E}_{P,R,S,T}$. By recasting this as the *first-order* expectation semiring $\mathbb{E}_{\mathbf{K},X}$ where $\mathbf{K} = \mathbb{E}_{P,R}$ and $X = (S \times T)$, we can again apply INSIDE-OUTSIDE(HG, $\mathbf{K}$, X) to find the total weight of all derivations.

For example, to speed up Section 3.2, we may define $\langle k_e, x_e \rangle = \langle \langle p_e, p_e r_e \rangle, \langle p_e s_e, p_e r_e s_e \rangle \rangle$ for each hyperedge e. Then the inside-outside algorithm of Figure 4 will compute $\langle \hat{k}, \hat{x} \rangle =$ $\langle \langle Z, \overline{r} \rangle, \langle \overline{s}, \overline{t} \rangle \rangle$, more quickly than the inside algorithm of Figure 2 computed $\langle Z, \overline{r}, \overline{s}, \overline{t} \rangle$.

Figure 4 in this case will run the inside and outside algorithms *in the semiring* $\mathbb{E}_{P,R}$, so that $k_e, \hat{k}, \alpha, \beta$, and $\overline{k_e}$ will now be elements of $P \times R$ (not just elements of P as in the first-order case). Finally it finds $\hat{x} = \sum_e \overline{k_e} x_e$, where $x_e \in S \times T$.[9]

This is a particularly effective speedup over the inside algorithm when R consists of scalars (or small vectors) whereas S, T are sparse high-dimensional vectors. We will see exactly this case in our experiments, where our weights $\langle p, r, s, t \rangle$ denote (probability, risk, gradient of probability, gradient of risk), or (probability, entropy, gradient of probability, gradient of entropy).

5 Finding Gradients on Hypergraphs

In Sections 3.2 and 4.1, we saw how our semirings helped find the sum Z of all $p(d)$, and compute *expectations* $\overline{r}, \overline{s}, \overline{t}$ of $r(d)$, $s(d)$, and $r(d)s(d)$.

It turns out that these semirings can *also* compute first- and second-order *partial derivatives* of all the above results, with respect to a parameter vector $\theta \in \mathbb{R}^m$. That is, we ask how they are affected when θ changes slightly from its current value. The elementary values p_e, r_e, s_e are now assumed to implicitly be functions of θ.

Case 1: Recall that $Z \stackrel{\text{def}}{=} \sum_d p(d)$ is computed by INSIDE(HG, $\mathbb{R}$) if each hyperedge e has weight p_e. "Lift" this weight to $\langle p_e, \nabla p_e \rangle$, where $\nabla p_e \in \mathbb{R}^m$ is a gradient vector. Now $\langle Z, \nabla Z \rangle$ will be returned by INSIDE(HG, $\mathbb{E}_{\mathbb{R}, \mathbb{R}^m}$)— or, more efficiently, by INSIDE-OUTSIDE(HG, $\mathbb{R}, \mathbb{R}^m$).

Case 2: To differentiate a second time, "lift" the above weights again to obtain $\langle \langle p_e, \nabla p_e \rangle, \nabla \langle p_e, \nabla p_e \rangle \rangle =$ $\langle \langle p_e, \nabla p_e \rangle, \langle \nabla p_e, \nabla^2 p_e \rangle \rangle$, where $\nabla^2 p_e \in \mathbb{R}^{m\times m}$ is the Hessian matrix of second-order mixed partial derivatives. These weights are in a second-order expectation semiring.[10] Now

---

[9] Figure 4 was already proved generally correct in Section 4.2. To understand more specifically how $\langle \overline{s}, \overline{t} \rangle$ gets computed, observe in analogy to the end of Section 4.2 that
$\langle \overline{s}, \overline{t} \rangle = \sum_d \langle p(d)s(d), p(d)r(d)s(d) \rangle$
$= \sum_d \langle p(d), p(d)r(d) \rangle \langle s(d), 0 \rangle$
$= \sum_d \langle p(d), p(d)r(d) \rangle \sum_{e\in d} \langle s_e, 0 \rangle$
$= \sum_e \sum_{d:\, e\in d} \langle p(d), p(d)r(d) \rangle \langle s_e, 0 \rangle$
$= \sum_e (\overline{k_e} k_e) \langle s_e, 0 \rangle = \sum_e \overline{k_e} \langle p_e, p_e r_e \rangle \langle s_e, 0 \rangle$
$= \sum_e \overline{k_e} \langle p_e s_e, p_e r_e s_e \rangle = \sum_e \overline{k_e} x_e = \hat{x}$.

[10] Modulo the trivial isomorphism from $\langle \langle p, r \rangle, \langle s, t \rangle \rangle$ to $\langle p, r, s, t \rangle$ (see Section 4.3), the intended semiring both here and in Case 3 is the one that was defined at the start of Section 4.1, in which r, s are vectors and their product is defined

$$\langle\langle p_1, r_1\rangle, \langle s_1, t_1\rangle\rangle \oplus \langle\langle p_2, r_2\rangle, \langle s_2, t_2\rangle\rangle = \langle\langle p_1, r_1\rangle + \langle p_2, r_2\rangle, \ \langle s_1, t_1\rangle + \langle s_2, t_2\rangle\rangle$$
$$= \langle\langle p_1 + p_2, \ r_1 + r_2\rangle, \ \langle s_1 + s_2, \ t_1 + t_2\rangle\rangle$$
$$\langle\langle p_1, r_1\rangle, \langle s_1, t_1\rangle\rangle \otimes \langle\langle p_2, r_2\rangle, \langle s_2, t_2\rangle\rangle = \langle\langle p_1, r_1\rangle\langle p_2, r_2\rangle, \ \langle p_1, r_1\rangle\langle s_2, t_2\rangle + \langle p_2, r_2\rangle\langle s_1, t_1\rangle\rangle$$
$$= \langle\langle p_1 p_2, \ p_1 r_2 + p_2 r_1\rangle, \ \langle p_1 s_2 + p_2 s_1, \ p_1 t_2 + p_2 t_1 + r_1 s_2 + r_2 s_1\rangle\rangle$$

Table 4: **Constructing second-order expectation semiring as first-order.** Here we show that the operations in $\mathbb{E}_{\mathbf{K}, X}$ are isomorphic to Table 2's operations in $\mathbb{E}_{P,R,S,T}$, provided that $\mathbf{K} \stackrel{\text{def}}{=} \mathbb{E}_{P,R}$ and $X \stackrel{\text{def}}{=} S \times T$ is a **K**-module, in which addition is defined by $\langle s_1, t_1\rangle + \langle s_2, t_2\rangle \stackrel{\text{def}}{=} \langle s_1 + s_2, t_1 + t_2\rangle$, and left-multiplication by **K** is defined by $\langle p, r\rangle\langle s, t\rangle \stackrel{\text{def}}{=} \langle ps, pt + rs\rangle$.

$\langle Z, \nabla Z, \nabla Z, \nabla^2 Z\rangle$ will be returned by $\text{INSIDE}(\text{HG}, \mathbb{E}_{\mathbb{R}, \mathbb{R}^m, \mathbb{R}^m, \mathbb{R}^{m \times m}})$, or more efficiently by $\text{INSIDE-OUTSIDE}(\text{HG}, \mathbb{E}_{\mathbb{R}, \mathbb{R}^m}, \mathbb{R}^m \times \mathbb{R}^{m \times m})$.

Case 3: Our experiments will need to find expectations and their partial derivatives. Recall that $\langle Z, \bar{r}\rangle$ is computed by $\text{INSIDE}(\text{HG}, \mathbb{E}_{\mathbb{R}, \mathbb{R}^n})$ when the edge weights are $\langle p_e, p_e r_e\rangle$ with $r_e \in \mathbb{R}^n$. Lift these weights to $\langle\langle p_e, p_e r_e\rangle, \nabla\langle p_e, p_e r_e\rangle\rangle = \langle\langle p_e, p_e r_e\rangle, \langle\nabla p_e, (\nabla p_e)r_e + p_e(\nabla r_e)\rangle\rangle$. Now $\langle Z, \bar{r}, \nabla Z, \nabla\bar{r}\rangle$ will be returned by $\text{INSIDE}(\text{HG}, \mathbb{E}_{\mathbb{R}, \mathbb{R}^n, \mathbb{R}^m, \mathbb{R}^{n \times m}})$ or by $\text{INSIDE-OUTSIDE}(\text{HG}, \mathbb{E}_{\mathbb{R}, \mathbb{R}^n}, \mathbb{R}^m \times \mathbb{R}^{n \times m})$.[11]

5.1 What Connects Gradients to Expectations?

In *Case 1*, we claimed that the *same* algorithm will compute either gradients $\langle Z, \nabla Z\rangle$ or expectations $\langle Z, \bar{r}\rangle$, if the hyperedge weights are set to $\langle p_e, \nabla p_e\rangle$ or $\langle p_e, p_e r_e\rangle$ respectively.[12] This may seem wonderful and mysterious. We now show in two distinct ways why this follows from our setup of Section 3.1. At the end, we derive as a special case the well-known relationship between gradients and expectations in log-linear models.

From Expectations to Gradients One perspective is that our semiring fundamentally finds expectations. Thus, we must be finding ∇Z by formulating it as a certain expectation $\bar{r}$. Specifically, $\nabla Z = \nabla \sum_d p(d) = \sum_d \nabla p(d) =$

$\sum_d p(d)r(d) = \bar{r}$, provided that $r(d) = (\nabla p(d))/p(d)$. That can be arranged by defining $r_e \stackrel{\text{def}}{=} (\nabla p_e)/p_e$.[13] So that is why the input weights $\langle p_e, p_e r_e\rangle$ take the form $\langle p_e, \nabla p_e\rangle$.

From Gradients to Expectations An alternative perspective is that our semiring fundamentally finds gradients. Indeed, pairs like $\langle p, \nabla p\rangle$ have long been used for this purpose (Clifford, 1873) under the name "dual numbers." Operations on dual numbers, including those in Table 1, compute a result in $\mathbb{R}$ *along with* its gradient. For example, our $\otimes$ multiplies dual numbers, since $\langle p_1, \nabla p_1\rangle \otimes \langle p_2, \nabla p_2\rangle = \langle p_1 p_2, p_1(\nabla p_2) + (\nabla p_1)p_2\rangle = \langle p_1 p_2, \nabla(p_1 p_2)\rangle$. The inside algorithm thus computes both Z and ∇Z in a single "forward" or "inside" pass—known as automatic differentiation in the forward mode. The inside-outside algorithm instead uses the reverse mode (a.k.a. back-propagation), where a separate "backward" or "outside" pass is used to compute ∇Z.

How can we modify this machinery to produce expectations $\bar{r}$ given some *arbitrary* r_e of interest? Automatic differentiation may be used on any function (e.g., a neural net), but for our simple sum-of-products function Z, it happens that $\nabla Z = \nabla(\sum_d \prod_e p_e) = \sum_d \sum_{e \in d}(\prod_{e' \in d, e' \neq e} p_{e'})\nabla p_e$. Our trick is to surreptitiously *replace* the ∇p_e in the input weights $\langle p_e, \nabla p_e\rangle$ with $p_e r_e$. Then the output changes similarly: the algorithms will *instead* find $\sum_d \sum_{e \in d}(\prod_{e' \in d, e' \neq e} p_{e'})p_e r_e$, which reduces to $\sum_d \sum_{e \in d} p(d)r_e = \sum_d p(d)\sum_{e \in d} r_e = \sum_d p(d)r(d) = \bar{r}$.

Log-linear Models as a Special Case Replacing ∇p_e with $p_e r_e$ is unnecessary if ∇p_e already equals $p_e r_e$. That is the case in log-linear models, where $p_e \stackrel{\text{def}}{=} \exp(r_e \cdot \theta)$ for some feature vector r_e associated with e. So there, ∇Z already equals $\bar{r}$—yielding a key useful property of log-linear

---

to be $rs^{\mathbf{T}}$, a matrix. However, when using this semiring to compute second derivatives (Case 2) or covariances, one may exploit the invariant that $r = s$, e.g., to avoid storing s and to compute $r_1 s_2 + s_1 r_2$ in multiplication simply as $2 \cdot r_1 r_2$.

[11] Or, if $n > m$, it is faster to instead use $\text{INSIDE-OUTSIDE}(\text{HG}, \mathbb{E}_{\mathbb{R}, \mathbb{R}^m}, \mathbb{R}^n \times \mathbb{R}^{m \times n})$, swapping the second and third components of the 4-tuple and transposing the matrix in the fourth component. Algebraically, this changes nothing because $\mathbb{E}_{\mathbb{R}, \mathbb{R}^n, \mathbb{R}^m \times \mathbb{R}^{n \times m}}$ and $\mathbb{E}_{\mathbb{R}, \mathbb{R}^m, \mathbb{R}^n \times \mathbb{R}^{m \times n}}$ are isomorphic, thanks to symmetries in Table 2. This method computes the expectation of the gradient rather than the gradient of the expectation—they are equal.

[12] *Cases 2–3* relied on the fact that this relationship still holds even when the scalars $Z, p_e \in \mathbb{R}$ are replaced by more complex objects that we wish to differentiate. Our discussion below sticks to the scalar case for simplicity, but would generalize fairly straightforwardly. Pearlmutter and Siskind (2007) give the relevant generalizations of dual numbers.

[13] Proof: $r(d) = \sum_{e \in d} r_e = \sum_{e \in d}(\nabla p_e)/p_e = \sum_{e \in d} \nabla \log p_e = \nabla \sum_{e \in d} \log p_e = \nabla \log \prod_{e \in d} p_e = \nabla \log p(d) = (\nabla p(d))/p(d)$.

models, that $\nabla \log Z = (\nabla Z)/Z = \bar{r}/Z$, the vector of feature expectations (Lau et al., 1993).

6 Practical Applications

Given a hypergraph HG whose hyperedges e are annotated with values p_e. Recall from Section 3.1 that this defines a probability distribution over all derivations d in the hypergraph, namely $p(d)/Z$ where $p(d) \overset{\text{def}}{=} \prod_{e \in d} p_e$.

6.1 First-Order Expectation Semiring $\mathbb{E}_{\mathbb{R},\mathbb{R}}$

In Section 3, we show how to compute the **expected hypothesis length** or **expected feature counts**, using the algorithm of Figure 2 with a first-order expectation semiring $\mathbb{E}_{\mathbb{R},\mathbb{R}}$. In general, given hyperedge weights $\langle p_e, p_e r_e \rangle$, the algorithm computes $\langle Z, \bar{r} \rangle$ and thus $\bar{r}/Z$, the expectation of $r(d) \overset{\text{def}}{=} \sum_{e \in d} r_e$. We now show how to compute a few other quantities by choosing r_e appropriately.

Entropy on a Hypergraph The entropy of the distribution of *derivations* in a hypergraph[14] is

$$\text{H}(p) = -\sum_{d \in \text{D}} (p(d)/Z) \log(p(d)/Z) \qquad (5)$$

$$= \log Z - \frac{1}{Z} \sum_{d \in \text{D}} p(d) \log p(d)$$

$$= \log Z - \frac{1}{Z} \sum_{d \in \text{D}} p(d) r(d) = \log Z - \frac{\bar{r}}{Z}$$

provided that we define $r_e \overset{\text{def}}{=} \log p_e$ (so that $r(d) = \sum_{e \in d} r_e = \log p(d)$). Of course, we can compute $\langle Z, \bar{r} \rangle$ as explained in Section 3.2.

Cross-Entropy and KL Divergence We may be interested in computing the cross-entropy or KL divergence between two distributions p and q. For example, in variational decoding for machine translation (Li et al., 2009b), p is a distribution represented by a hypergraph, while q, represented by a finite state automaton, is an approximation to p. The cross entropy between p and q is defined as

$$\text{H}(p, q) = -\sum_{d \in \text{D}} (p(d)/Z_p) \log(q(d)/Z_q) \qquad (6)$$

$$= \log Z_q - \frac{1}{Z_p} \sum_{d \in \text{D}} p(d) \log q(d)$$

$$= \log Z_q - \frac{1}{Z_p} \sum_{d \in \text{D}} p(d) r(d) = \log Z_q - \frac{\bar{r}}{Z_p}$$

---

[14]Unfortunately, it is intractable to compute the entropy of the distribution over *strings* (each string's probability is a sum over several derivations). But Li et al. (2009b, section 5.4) do estimate the gap between derivational and string entropies.

where the first term Z_q can be computed using the inside algorithm with hyperedge weights q_e, and the numerator and denominator of the second term using an expectation semiring with hyperedge weights $\langle p_e, p_e r_e \rangle$ with $r_e \overset{\text{def}}{=} \log q_e$.

The KL divergence to p from q can be computed as $\text{KL}(p \,\|\, q) = \text{H}(p, q) - \text{H}(p)$.

Expected Loss (Risk) Given a reference sentence y^*, the expected loss (i.e., Bayes risk) of the hypotheses in the hypergraph is defined as,

$$\text{R}(p) = \sum_{d \in \text{D}} (p(d)/Z) L(\text{Y}(d), y^*) \qquad (7)$$

where $\text{Y}(d)$ is the target yield of d and $L(y, y^*)$ is the **loss** of the hypothesis y with respect to the reference y^*. The popular machine translation metric, BLEU (Papineni et al., 2001), is not **additively** decomposable, and thus we are not able to compute the expected loss for it. Tromble et al. (2008) develop the following loss function, of which a linear approximation to BLEU is a special case,

$$L(y, y^*) = -(\theta_0 |y| + \sum_{w \in N} \theta_w \#_w(y) \delta_w(y*)) \quad (8)$$

where w is an n-gram type, N is a set of n-gram types with $n \in [1, 4]$, $\#_w(y)$ is the number of occurrence of the n-gram w in y, $\delta_w(y^*)$ is an indicator to check if y^* contains at least one occurrence of w, and θ_n is the weight indicating the relative importance of an n-gram match. If the hypergraph is *already* annotated with n-gram ($n \geq 4$) language model states, this loss function is **additively** decomposable. Using $r_e \overset{\text{def}}{=} L_e$ where L_e is the *loss* for a hyperedge e, we compute the expected loss,

$$\text{R}(p) = \frac{\sum_{d \in \text{D}} p(d) L(\text{Y}(d), y^*)}{Z} = \frac{\bar{r}}{Z} \quad (9)$$

6.2 Second-Order Expectation Semirings

With second-order expectation semirings, we can compute from a hypergraph the expectation and variance of hypothesis length; the feature expectation vector and covariance matrix; the Hessian (matrix of second derivatives) of Z; and the gradients of entropy and expected loss. The computations should be clear from earlier discussion. Below we compute gradient of entropy or Bayes risk.

Gradient of Entropy or Risk It is easy to see that the gradient of entropy (5) is

$$\nabla \text{H}(p) = \frac{\nabla Z}{Z} - \frac{Z \nabla \bar{r} - \bar{r} \nabla Z}{Z^2} \qquad (10)$$

We may compute $\langle Z, \bar{r}, \nabla Z, \nabla \bar{r} \rangle$ as explained in **Case 3** of Section 5 by using $k_e \stackrel{\text{def}}{=} \langle p_e, p_e r_e, \nabla p_e, (\nabla p_e) r_e + p_e \nabla r_e \rangle \stackrel{\text{def}}{=} \langle p_e, p_e \log p_e, \nabla p_e, (1 + \log p_e) \nabla p_e \rangle$, where ∇p_e depends on the particular parameterization of the model (see Section 7.1 for an example).

Similarly, the gradient of risk of (9) is

$$\nabla \mathrm{R}(p) = \frac{Z \nabla \bar{r} - \bar{r} \nabla Z}{Z^2} \qquad (11)$$

We may compute $\langle Z, \bar{r}, \nabla Z, \nabla \bar{r} \rangle$ using $k_e \stackrel{\text{def}}{=} \langle p_e, p_e L_e, \nabla p_e, L_e \nabla p_e \rangle$.

7 Minimum-Risk Training for MT

We now show how we improve the training of a Hiero MT model by optimizing an objective function that includes entropy and risk. Our objective function could be computed with a first-order expectation semiring, but computing it *along with its gradient* requires a second-order one.

7.1 The Model p

We assume a **globally normalized linear model** for its simplicity. Each derivation d is scored by

$$\text{score}(d) \stackrel{\text{def}}{=} \Phi(d) \cdot \theta = \sum_i \Phi_i(d) \, \theta_i \qquad (12)$$

where $\Phi(d) \in \mathbb{R}^m$ is a vector of features of d. We then define the unnormalized distribution $p(d)$ as

$$p(d) = \exp(\gamma \cdot \text{score}(d)) \qquad (13)$$

where the scale factor γ adjusts how sharply the distribution favors the highest-scoring hypotheses.

7.2 Minimum-Risk Training

Adjusting θ or γ changes the distribution p. Minimum error rate training (MERT) (Och, 2003) tries to tune θ to minimize the BLEU loss of a decoder that chooses the most probable output according to p. (γ has no effect.) MERT's specialized linesearch addresses the problem that this objective function is piecewise constant, but it does not scale to a large number of parameters.

Smith and Eisner (2006) instead propose a *differentiable* objective that can be optimized by gradient descent: the Bayes risk $\mathrm{R}(p)$ of (7). This is the *expected* loss if one were (hypothetically) to use a *randomized* decoder, which chooses a hypothesis d in proportion to its probability $p(d)$. If entropy $\mathrm{H}(p)$ is large (e.g., small γ), the Bayes risk

is smooth and has few local minima. Thus, Smith and Eisner (2006) try to avoid local minima by starting with large $\mathrm{H}(p)$ and decreasing it gradually during optimization. This is called **deterministic annealing** (Rose, 1998). As $\mathrm{H}(p) \to 0$ (e.g., large γ), the Bayes risk does approach the MERT objective (i.e. minimizing 1-best error).The objective is

$$\text{minimize } \mathrm{R}(p) - T \cdot \mathrm{H}(p) \qquad (14)$$

where the "temperature" T starts high and is explicitly decreased as optimization proceeds.

7.3 Gradient Descent Optimization

Solving (14) for a given T requires computing the entropy $\mathrm{H}(p)$ and risk $\mathrm{R}(p)$ and their gradients with respect to θ and γ. Smith and Eisner (2006) followed MERT in constraining their decoder to only an n-best list, so for them, computing these quantities did not involve dynamic programming. We compare those methods to *training on a hypergraph containing exponentially many hypotheses*. In this condition, we need our new second-order semiring methods and must also approximate BLEU (during training only) by an additively decomposable loss (Tromble et al., 2008).[15]

Our algorithms require that $p(d)$ of (13) is **multiplicatively** decomposable. It suffices to define $\Phi(d) \stackrel{\text{def}}{=} \sum_{e \in d} \Phi_e$, so that all features are *local* to individual hyperedges; the vector Φ_e indicates which features fire on hyperedge e. Then $\text{score}(d)$ of (12) is **additively** decomposable:

$$\text{score}(d) = \sum_{e \in d} \text{score}_e = \sum_{e \in d} \Phi_e \cdot \theta \qquad (15)$$

We can then set $p_e = \exp(\gamma \cdot \text{score}_e)$, and $\nabla p_e = \gamma p_e \Phi(e)$, and use the algorithms described in Section 6 to compute $\mathrm{H}(p)$ and $\mathrm{R}(p)$ and their gradients with respect to θ and γ.[16]

---

[15]Pauls et al. (2009) concurrently developed a method to maximize the *expected* n-gram counts on a *hypergraph* using gradient descent. Their objective is similar to the minimum risk objective (though without annealing), and their gradient descent optimization involves in algorithms in computing expected feature/n-gram counts as well as expected *products* of features and n-gram counts, which can be viewed as instances of our general algorithms with first- and second-order semirings. They focused on tuning only a small number (i.e. *nine*) of features as in a regular MERT setting, while our experiments involve both a small and a large number of features.

[16]It is easy to verify that the gradient of a function f (e.g. entropy or risk) with respect to γ can be written as a weighted sum of gradients with respect to the feature weights θ_i, i.e.

$$\frac{\partial_f}{\partial_\gamma} = \frac{1}{\gamma} \sum_i \theta_i \times \frac{\partial_f}{\partial_{\theta_i}} \qquad (16)$$

7.4 Experimental Results

7.4.1 Experimental Setup

We built a translation model on a corpus for IWSLT 2005 Chinese-to-English translation task (Eck and Hori, 2005), which consists of 40k pairs of sentences. We used a *5-gram* language model with modified Kneser-Ney smoothing, trained on the bitext's English using SRILM (Stolcke, 2002).

7.4.2 Tuning a Small Number of Features

We first investigate how minimum-risk training (MR), with and without deterministic annealing (DA), performs compared to regular MERT. MR without DA just fixes $T = 0$ and $\gamma = 1$ in (14). All MR or MR+DA uses an approximated BLEU (Tromble et al., 2008) (for training only), while MERT uses the exact corpus BLEU in training.

The first five rows in Table 5 present the results by tuning the weights of *five* features ($\theta \in \mathbb{R}^5$). We observe that MR or MR+DA performs worse than MERT *on the dev set*. This may be mainly because MR or MR+DA uses an approximated BLEU while MERT doesn't. *On the test set*, MR or MR+DA on an n-best list is comparable to MERT. But our new approach, MR or MR+DA *on a hypergraph*, does consistently better (statistically significant) than MERT, despite approximating BLEU.[17]

Did DA help? For both n-best and hypergraph, MR+DA did obtain a better BLEU score than plain MR on the dev set.[18] This shows that DA helps with the local minimum problem, as hoped. However, DA's improvement on the dev set did not transfer to the test set.

7.4.3 Tuning a Large Number of Features

MR (with or without DA) is scalable to tune a large number of features, while MERT is not. To achieve competitive performance, we adopt a *forest reranking* approach (Li and Khudanpur, 2009; Huang, 2008). Specifically, our training has two stages. In the *first* stage, we train a baseline system as usual. We also find the optimal feature weights for the five features mentioned before, using the method of MR+DA operating on a hypergraph. In the *second* stage, we generate a hypergraph for each sentence in the training data (which consists of about 40k sentence pairs), using the baseline

Training scheme	dev	test
MERT (Nbest, small)	42.6	47.7
MR (Nbest, small)	40.8	47.7
MR+DA (Nbest, small)	41.6	47.8
NEW! MR (hypergraph, small)	41.3	48.4
NEW! MR+DA (hypergraph, small)	41.9	48.3
NEW! MR (hypergraph, large)	42.3	**48.7**

Table 5: BLEU scores on the Dev and test sets under different training scenarios. In the "small" model, five features (i.e., one for the language model, three for the translation model, and one for word penalty) are tuned. In the "large" model, 21k additional unigram and bigram features are used.

system. In this stage, we add 21k additional unigram and bigram target-side language model features (cf. Li and Khudanpur (2008)). For example, a specific bigram "the cat" can be a feature. Note that the total score by the baseline system is also a feature in the second-stage model. With these features and the 40k hypergraphs, we run the MR training to obtain the optimal weights.

During test time, a similar procedure is followed. For a given test sentence, the baseline system first generates a hypergraph, and then the hypergraph is reranked by the second-stage model. The last row in Table 5 reports the BLEU scores. Clearly, adding more features improves (statistically significant) the case with only five features. We plan to incorporate more informative features described by Chiang et al. (2009).[19]

8 Conclusions

We presented first-order expectation semirings and inside-outside computation in more detail than (Eisner, 2002), and developed extensions to higher-order expectation semirings. This enables efficient computation of many interesting quantities over the exponentially many derivations encoded in a hypergraph: second derivatives (Hessians), expectations of products (covariances), and expectations such as risk and entropy along with their derivatives. To our knowledge, algorithms for these problems have not been presented before.

Our approach is theoretically elegant, like other work in this vein (Goodman, 1999; Lopez, 2009; Gimpel and Smith, 2009). We used it practically to enable a new form of minimum-risk training that improved Chinese-English MT by 1.0 BLEU point. Our implementation will be released within the open-source MT toolkit **Joshua** (Li et al., 2009a).

---

[17] Pauls et al. (2009) concurrently observed a similar pattern (i.e., MR performs worse than MERT on the dev set, but performs better on a test set).

[18] We also verified that MR+DA found a better objective value (i.e., expected loss on the dev set) than MR.

[19] Their MIRA training tries to favor a specific oracle translation—indeed a specific tree—from the (pruned) hypergraph. MR does not commit to such an arbitrary choice.

References

J. K. Baker. 1979. Trainable grammars for speech recognition. In Jared J. Wolf and Dennis H. Klatt, editors, *Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America*, MIT, Cambridge, MA, June.

David Chiang, Kevin Knight, and Wei Wang. 2009. 11,001 new features for statistical machine translation. In *NAACL*, pages 218–226.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

W. K. Clifford. 1873. Preliminary sketch of biquaternions. *Proceedings of the London Mathematical Society*, 4:381–395.

Matthias Eck and Chiori Hori. 2005. Overview of the iwslt 2005 evaluation campaign. In *In Proc. of the International Workshop on Spoken Language Translation*.

Jason Eisner, Eric Goldlust, and Noah A. Smith. 2005. Compiling comp ling: practical weighted dynamic programming and the dyna language. In *HLT/EMNLP*, pages 281–290.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *ACL*, pages 1–8.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *ACL*, pages 205–208.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *ACL*, pages 961–968.

Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. 1993. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201.

Kevin Gimpel and Noah A. Smith. 2009. Cube summing, approximate inference with non-local features, and dynamic programming without semirings. In *EACL*, pages 318–326.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

Y Grandvalet and Y Bengio. 2004. Semi-supervised learning by entropy minimization. In *NIPS*, pages 529–536.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *IWPT*, pages 53–64.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594.

Feng Jiao, Shaojun Wang, Chi-Hoon Lee, Russell Greiner, and Dale Schuurmans. 2006. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *ACL*, pages 209–216.

Dan Klein and Christopher D. Manning. 2004. Parsing and hypergraphs. *New developments in parsing technology*, pages 351–372.

Raymond Lau, Ronald Rosenfeld, and Salim Roukos. 1993. Adaptive language modelling using the maximum entropy principle. In *Proc. ARPA Human Language Technologies Workshop*, pages 81–86.

Zhifei Li and Sanjeev Khudanpur. 2008. Large-scale discriminative n-gram language models for statistical machine translation. In *AMTA*, pages 133–142.

Zhifei Li and Sanjeev Khudanpur. 2009. Forest reranking for machine translation with the perceptron algorithm. In *GALE book chapter on "MT From Text"*.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar. Zaidan. 2009a. Joshua: An open source toolkit for parsing-based machine translation. In *WMT09*, pages 26–30.

Zhifei Li, Jason Eisner, and Sanjeev Khudanpur. 2009b. Variational decoding for statistical machine translation. In *ACL*.

Yang Liu, Qun Liu, and Shouxun Lin. 2006. Tree-to-string alignment template for statistical machine translation. In *ACL*, pages 609–616.

Adam Lopez. 2009. Translation as weighted deduction. In *EACL*, pages 532–540.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL*, pages 160–167.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. BLEU: A method for automatic evaluation of machine translation. In *ACL*, pages 311–318.

Adam Pauls, John DeNero, and Dan Klein. 2009. Consensus training for consensus decoding in machine translation. In *EMNLP*.

B. A. Pearlmutter and J. M. Siskind. 2007. Lazy multivariate higher-order forward-mode ad. In *Proceedings of the 34th Annual Symposium on Principles of Programming Languages (POPL)*, pages 155–160.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as deduction. In *ACL*, pages 137–144.

Chris Quirk, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: syntactically informed phrasal smt. In *ACL*, pages 271–279.

Kenneth Rose. 1998. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. In *Proceedings of the IEEE*, pages 2210–2239.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1994. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *ACL*, pages 787–794.

Andreas Stolcke. 2002. SRILM—an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904.

Roy Tromble, Shankar Kumar, Franz Och, and Wolfgang Macherey. 2008. Lattice minimum-Bayes-risk decoding for statistical machine translation. In *EMNLP*, pages 620–629.