Tree Kernels for Parsing: (Collins & Duffy, 2001)

Advanced Statistical Methods in NLP
Ling 572
February 28, 2012
Roadmap

- Collins & Duffy, 2001
- Tree Kernels for Parsing:
  - Motivation
  - Parsing as reranking
- Tree kernels for similarity
- Case study: Penn Treebank parsing
Motivation: Parsing

- Parsing task:
  - Given a natural language sentence, extract its syntactic structure
    - Specifically, generate a corresponding parse tree
Motivation: Parsing

- Parsing task:
  - Given a natural language sentence, extract its syntactic structure
    - Specifically, generate a corresponding parse tree

- Approaches:
Motivation: Parsing

• Parsing task:
  • Given a natural language sentence, extract its syntactic structure
    • Specifically, generate a corresponding parse tree

• Approaches:
  • “Classical” approach:
    • Hand-write CFG productions; use standard alg, e.g. CKY
Motivation: Parsing

- Parsing task:
  - Given a natural language sentence, extract its syntactic structure
    - Specifically, generate a corresponding parse tree

- Approaches:
  - “Classical” approach:
    - Hand-write CFG productions; use standard alg, e.g. CKY
  - Probabilistic approach:
    - Build large treebank of parsed sentences
    - Learn production probabilities
    - Use probabilistic versions of standard alg
    - Pick highest probability parse
Parsing Issues

- Main issues:
Parsing Issues

- Main issues:
  - Robustness: get reasonable parse for any input
  - Ambiguity: select best parse given alternatives
- “Classic” approach:
Parsing Issues

- Main issues:
  - Robustness: get reasonable parse for any input
  - Ambiguity: select best parse given alternatives

- “Classic” approach:
  - Hand-coded grammars often fragile
  - No obvious mechanism to select among alternatives

- Probabilistic approach:
Parsing Issues

• Main issues:
  • Robustness: get reasonable parse for any input
  • Ambiguity: select best parse given alternatives

• “Classic” approach:
  • Hand-coded grammars often fragile
  • No obvious mechanism to select among alternatives

• Probabilistic approach:
  • Fairly good robustness, small probabilities for any
  • Select by probability, but decisions are local
    • Hard to capture more global structure
Approach: Parsing by Reranking

- Intuition:
  - Identify collection of candidate parses for each sentence
    - e.g. output of PCFG parser
    - For training, identify gold standard parse, sentence pair
Approach: Parsing by Reranking

- Intuition:
  - Identify collection of candidate parses for each sentence
    - e.g. output of PCFG parser
    - For training, identify gold standard parse, sentence pair

- Create a parse tree vector representation
  - Identify (more global) parse tree features
Approach: Parsing by Reranking

- **Intuition:**
  - Identify collection of candidate parses for each sentence
    - e.g. output of PCFG parser
    - For training, identify gold standard parse, sentence pair
  
- Create a parse tree vector representation
  - Identify (more global) parse tree features

- Train a reranker to rank gold standard highest
Approach: Parsing by Reranking

- Intuition:
  - Identify collection of candidate parses for each sentence
    - e.g. output of PCFG parser
    - For training, identify gold standard parse, sentence pair
  - Create a parse tree vector representation
    - Identify (more global) parse tree features
  - Train a reranker to rank gold standard highest
  - Apply to rerank candidate parses for new sentence
Parsing as Reranking, Formally

- Training data pairs: \{ (s_i, t_i) \}
- where \( s_i \) is a sentence, \( t_i \) is a parse tree
Parsing as Reranking, Formally

- Training data pairs: \{(s_i, t_i)\}
  - where \(s_i\) is a sentence, \(t_i\) is a parse tree
Parsing as Reranking, Formally

- Training data pairs: \(((s_i, t_i))\)
  - where \(s_i\) is a sentence, \(t_i\) is a parse tree

- \(C(s_i) = \{x_{ij}\}:\)
  - Candidate parses for \(s_i\)
  - \(wlog, x_{i1}\) is the correct parse for \(s_i\)
Parsing as Reranking, Formally

- Training data pairs: \{ (s_i, t_i) \}
  - where \( s_i \) is a sentence, \( t_i \) is a parse tree
- \( C(s_i) = \{ x_{ij} \} \):
  - Candidate parses for \( s_i \)
  - \textit{wlog}, \( x_{i1} \) is the correct parse for \( s_i \)
- \( h(x_{ij}) \): feature vector representation of \( x_{ij} \)
Parsing as Reranking, Formally

- Training data pairs: \{((s_i,t_i))\}
  - where $s_i$ is a sentence, $t_i$ is a parse tree

- $C(s_i) = \{x_{ij}\}$:
  - Candidate parses for $s_i$
  - wlog, $x_{i1}$ is the correct parse for $s_i$

- $h(x_{ij})$: feature vector representation of $x_{ij}$

- Training: Learn $\vec{w}$
Parsing as Reranking, Formally

- Training data pairs: \( \{(s_i, t_i)\} \)
  - where \( s_i \) is a sentence, \( t_i \) is a parse tree

- \( C(s_i) = \{x_{ij}\} \):
  - Candidate parses for \( s_i \)
  - wlog, \( x_{i1} \) is the correct parse for \( s_i \)

- \( h(x_{ij}) \): feature vector representation of \( x_{ij} \)

- Training: Learn \( \vec{w} \)

- Decoding: Compute \( x^* = \arg\max_{x \in C(s)} \vec{w} \cdot h(x) \)
Parsing as Reranking: Training

- Consider the hard-margin SVM model:
  - Minimize $||w||^2$ subject to constraints
Parsing as Reranking: Training

- Consider the hard-margin SVM model:
  - Minimize $||w||^2$ subject to constraints

- What constraints?
Consider the hard-margin SVM model:
- Minimize $||w||^2$ subject to constraints

What constraints?
- Here, ranking constraints:
  - Specifically, correct parse outranks all other candidates
Parsing as Reranking: Training

- Consider the hard-margin SVM model:
  - Minimize $||w||^2$ subject to constraints

- What constraints?
  - Here, ranking constraints:
    - Specifically, correct parse outranks all other candidates
    - Formally,
      $$\bar{w} \cdot h(x_{i1}) > \bar{w} \cdot h(x_{ij}), \forall i, \forall j \geq 2$$
Parsing as Reranking: Training

- Consider the hard-margin SVM model:
  - Minimize $||w||^2$ subject to constraints

- What constraints?
  - Here, ranking constraints:
    - Specifically, correct parse outranks all other candidates
  - Formally,
    \[ \tilde{w} \cdot h(x_{i1}) > \tilde{w} \cdot h(x_{ij}), \forall i, \forall j \geq 2 \]
    \[ \tilde{w} \cdot (h(x_{i1}) - h(x_{ij})) \geq 0, \forall i, \forall j \geq 2 \]
Parsing as Reranking: Training

- Consider the hard-margin SVM model:
  - Minimize $||w||^2$ subject to constraints

- What constraints?
  - Here, ranking constraints:
    - Specifically, correct parse outranks all other candidates
    - Formally,
      $$\bar{w} \cdot h(x_{i1}) > \bar{w} \cdot h(x_{ij}), \forall i, \forall j \geq 2$$
      $$\bar{w} \cdot (h(x_{i1}) - h(x_{ij})) \geq 0, \forall i, \forall j \geq 2$$
      $$SVM: \bar{w} \cdot (h(x_{i1}) - h(x_{ij})) \geq 1, \forall i, \forall j \geq 2$$
Reformulating with $\alpha$

- Training learns $\alpha_{ij}$, such that
  \[ \tilde{w} = \sum_{ij} \alpha_{ij} (h(x_{i1}) - h(x_{ij})) \]
Reformulating with $\alpha$

- Training learns $\alpha_{ij}$, such that
  \[ \tilde{w} = \sum_{ij} \alpha_{ij} (h(x_i) - h(x_j)) \]
- Note: just like SVM equation, w/different constraint
Reformulating with $\alpha$

- Training learns $\alpha_{ij}$, such that
  \[
  \tilde{w} = \sum_{ij} \alpha_{ij} (h(x_i) - h(x_j))
  \]
- Note: just like SVM equation, w/different constraint
- Parse scoring: $\tilde{w} \cdot h(x)$
Reformulating with $\alpha$

- Training learns $\alpha_{ij}$, such that
  \[ \vec{w} = \sum_{ij} \alpha_{ij}(h(x_{i1}) - h(x_{ij})) \]
- Note: just like SVM equation, w/different constraint

- Parse scoring: $\vec{w} \cdot h(x)$
- After substitution, we have
  \[ f(x) = \vec{w} \cdot h(x) = \sum_{ij} \alpha_{ij}((h(x_{i1}) \cdot h(x)) - (h(x_{ij}) \cdot h(x))) \]
Reformulating with $\alpha$

- Training learns $\alpha_{ij}$, such that
  \[
  \tilde{w} = \sum_{ij} \alpha_{ij} (h(x_{i1}) - h(x_{ij}))
  \]
- Note: just like SVM equation, w/different constraint

- Parse scoring: $\tilde{w} \cdot h(x)$
- After substitution, we have
  \[
  f(x) = \tilde{w} \cdot h(x) = \sum_{ij} \alpha_{ij} ((h(x_{i1}) \cdot h(x)) - (h(x_{ij}) \cdot h(x)))
  \]
- After the kernel trick, we have
  \[
  f(x) = \tilde{w} \cdot h(x) = \sum_{ij} \alpha_{ij} (K(x_{i1}, x) - K(x_{ij}, x))
  \]
Reformulating with $\alpha$

- Training learns $\alpha_{ij}$, such that
  \[ \tilde{w} = \sum_{ij} \alpha_{ij} (h(x_{i1}) - h(x_{ij})) \]
- Note: just like SVM equation, w/different constraint

- Parse scoring: $\tilde{w} \cdot h(x)$
  - After substitution, we have
    \[ f(x) = \tilde{w} \cdot h(x) = \sum_{ij} \alpha_{ij} ((h(x_{i1}) \cdot h(x)) - (h(x_{ij}) \cdot h(x))) \]
  - After the kernel trick, we have
    \[ f(x) = \tilde{w} \cdot h(x) = \sum_{ij} \alpha_{ij} (K(x_{i1}, x) - K(x_{ij}, x)) \]
  - Note: With a suitable kernel $K$, don’t need $h(x)$s
Parsing as reranking: Perceptron algorithm

- Similar to SVM, learns separating hyperplane
Parsing as reranking: Perceptron algorithm

- Similar to SVM, learns separating hyperplane
  - Modeled with weight vector \( w \)
  - Using simple iterative procedure
    - Based on correcting errors in current model
Parsing as reranking: Perceptron algorithm

- Similar to SVM, learns separating hyperplane
- Modeled with weight vector $w$
- Using simple iterative procedure
  - Based on correcting errors in current model

$$f(x) = \sum_{ij} \alpha_{ij} ((h(x_{x1}) \cdot h(x)) - (h(x_{ij}) \cdot h(x)))$$

- Initialize $\alpha_{ij}=0$
Parsing as reranking: Perceptron algorithm

- Similar to SVM, learns separating hyperplane
- Modeled with weight vector \( w \)
- Using simple iterative procedure
  - Based on correcting errors in current model

\[
f(x) = \sum_{i,j} \alpha_{ij} ((h(x_{1i}) \cdot h(x)) - (h(x_{ij}) \cdot h(x)))
\]

- Initialize \( \alpha_{ij} = 0 \)
- For \( i=1,\ldots,n; \) for \( j=2,\ldots,n \)
  - If \( f(x_{i1}) > f(x_{ij}) \): continue
  - else: \( \alpha_{ij} += 1 \)
Defining the Kernel

- So, we have a model:
  - Framework for training
  - Framework for decoding
Defining the Kernel

- So, we have a model:
  - Framework for training
  - Framework for decoding

- But need to define a kernel $K$

$$ f(x) = \vec{w} \cdot h(x) = \sum_{ij} \alpha_{ij} (K(x_{i1}, x) - K(x_{ij}, x)) $$
Defining the Kernel

- So, we have a model:
  - Framework for training
  - Framework for decoding

- But need to define a kernel $K$

$$f(x) = \vec{w} \cdot h(x) = \sum_{ij} \alpha_{ij} (K(x_{i1}, x) - K(x_{ij}, x))$$

- We need: $K: X \times X \rightarrow \mathbb{R}$
Defining the Kernel

• So, we have a model:
  • Framework for training
  • Framework for decoding

• But need to define a kernel $K$

\[ f(x) = \vec{w} \cdot h(x) = \sum_{ij} \alpha_{ij} (K(x_{i1}, x) - K(x_{ij}, x)) \]

• We need: $K: X \times X \rightarrow \mathbb{R}$

• Recall that $X$ is a tree, and $K$ is a similarity function
What’s in a Kernel?

- What are good attributes of a kernel?
What’s in a Kernel?

- What are good attributes of a kernel?
  - Capture similarity between instances
    - Here, between parse trees
What’s in a Kernel?

- What are good attributes of a kernel?
  - Capture similarity between instances
    - Here, between parse trees
  - Capture more global parse information than PCFG
What’s in a Kernel?

- What are good attributes of a kernel?
  - Capture similarity between instances
    - Here, between parse trees
  - Capture more global parse information than PCFG
  - Computable tractably, even over complex, large trees
Tree Kernel Proposal

- **Idea:**
  - PCFG models learn MLE probabilities on rewrite rules
    - NP $\rightarrow$ N vs NP $\rightarrow$ DT N vs NP $\rightarrow$ PN vs NP $\rightarrow$ DT JJ N
    - Local to parent:children levels
Tree Kernel Proposal

- **Idea:**
  - PCFG models learn MLE probabilities on rewrite rules
    - NP $\rightarrow$ N vs NP $\rightarrow$ DT N vs NP $\rightarrow$ PN vs NP $\rightarrow$ DT JJ N
    - Local to parent:children levels

- New measure incorporates all tree fragments in parse
  - Captures higher order, longer distances dependencies
    - Track counts of individual rules + much more
Tree Fragment Example

- Fragments of NP over ‘apple’
- Not exhaustive
Tree Representation

- Tree fragments:
  - Any subgraph with more than one node
    - Restriction: Must include full (not partial) rule productions

- Parse tree representation:
  - \( h(T) = (h_1(T), h_2(T), \ldots, h_n(T)) \)
    - \( n \): number of distinct tree fragments in training data
    - \( h_i(T) \): # of occurrences of \( i^{th} \) tree fragment in current tree
Tree Representation

- Tree fragments:
  - Any subgraph with more than one node
    - Restriction: Must include full (not partial) rule productions

- Parse tree representation:
  - $h(T) = (h_1(T), h_2(T), \ldots, h_n(T))$
    - $n$: number of distinct tree fragments in training data
    - $h_i(T)$: # of occurrences of $i^{th}$ tree fragment in current tree
Tree Representation

- Pros:
Tree Representation

- Pros:
  - Fairly intuitive model
  - Natural inner product interpretation
  - Captures long- and short-range dependencies

- Cons:
Tree Representation

• Pros:
  • Fairly intuitive model
  • Natural inner product interpretation
  • Captures long- and short-range dependencies

• Cons:
  • Size!!!: # subtrees exponential in size of tree
  • Direct computation of inner product intractable
Key Challenge
Key Challenge

• Efficient computation:
  • Find a kernel that can compute similarity efficiently
    • In terms of common subtrees
Key Challenge

• Efficient computation:
  • Find a kernel that can compute similarity efficiently
    • In terms of common subtrees
  • Pure enumeration clearly intractable
Key Challenge

- Efficient computation:
  - Find a kernel that can compute similarity efficiently
    - In terms of common subtrees
  - Pure enumeration clearly intractable

- Compute recursively over subtrees
  - Using a polynomial process
Counting Common Subtrees

- Example:
  - $C(n_1,n_2)$: number of common subtrees rooted at $n_1,n_2$
  - $C(n_1,n_2)$:

\[\text{Due to F. Xia}\]
Calculating $C(n_1, n_2)$

- Given two subtrees rooted at $n_1$ and $n_2$
- If productions at $n_1$ and $n_2$ are different,
Calculating $C(n_1, n_2)$

- Given two subtrees rooted at $n_1$ and $n_2$
  - If productions at $n_1$ and $n_2$ are different,
    - $C(n_1, n_2) = 0$
  - If productions at $n_1$ and $n_2$ are the same,
    - And $n_1$ and $n_2$ are preterminals,
Calculating $C(n_1,n_2)$

- Given two subtrees rooted at $n_1$ and $n_2$
  - If productions at $n_1$ and $n_2$ are different,
    - $C(n_1,n_2) = 0$
  - If productions at $n_1$ and $n_2$ are the same,
    - And $n_1$ and $n_2$ are preterminals,
      - $C(n_1,n_2) = 1$
  - Else:
Calculating $C(n_1,n_2)$

- Given two subtrees rooted at $n_1$ and $n_2$
  - If productions at $n_1$ and $n_2$ are different,
    - $C(n_1,n_2) = 0$
  - If productions at $n_1$ and $n_2$ are the same,
    - And $n_1$ and $n_2$ are preterminals,
      - $C(n_1,n_2) = 1$
    - Else: $C(n_1,n_2) = \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1,j), ch(n_2,j)))$

- $nc(n_1)$: # children of $n_1$:
  - What about $n_2$?
Calculating $C(n_1,n_2)$

- Given two subtrees rooted at $n_1$ and $n_2$
  - If productions at $n_1$ and $n_2$ are different,
    - $C(n_1,n_2) = 0$
  - If productions at $n_1$ and $n_2$ are the same,
    - And $n_1$ and $n_2$ are preterminals,
      - $C(n_1,n_2) = 1$
    - Else:
      - $C(n_1,n_2) = \prod_{j=1}^{\text{nc}(n_1)} (1 + C(\text{ch}(n_1,j),\text{ch}(n_2,j)))$

- $\text{nc}(n_1)$: # children of $n_1$:
  - What about $n_2$? same production $\Rightarrow$ same # children
  - $\text{ch}(n_1,j) \Rightarrow j^{th}$ child of $n_1$
Components of Tree Representation

- Tree representation: $h(T) = (h_1(T), h_2(T), \ldots, h_n(T))$
- Consider 2 trees: $T_1, T_2$
- Number of nodes: $N_1, N_2$, respectively
Components of Tree Representation

- Tree representation: \( h(T) = (h_1(T), h_2(T), \ldots, h_n(T)) \)
- Consider 2 trees: \( T_1, T_2 \)
- Number of nodes: \( N_1, N_2 \), respectively
- Define: \( l_i(n) = 1 \) if \( i^{th} \) subtree is rooted at \( n \), 0 o.w.
Components of Tree Representation

- Tree representation: \( h(T) = (h_1(T), h_2(T), \ldots, h_n(T)) \)
- Consider 2 trees: \( T_1, T_2 \)
- Number of nodes: \( N_1, N_2 \)
- Define: \( I_i(n) = 1 \) if \( i^{th} \) subtree is rooted at \( n \), 0 o.w.
- Then,

\[
\begin{align*}
    h_i(T_1) &= \sum_{n_1 \in N_1} I_i(n_1) \\
    h_i(T_2) &= \sum_{n_2 \in N_2} I_i(n_2)
\end{align*}
\]
Components of Tree Representation

- Tree representation: \( h(T) = (h_1(T), h_2(T), \ldots, h_n(T)) \)
- Consider 2 trees: \( T_1, T_2 \)
- Number of nodes: \( N_1, N_2 \), respectively
- Define: \( I_i(n) = 1 \) if \( i^{th} \) subtree is rooted at \( n \), 0 o.w.
- Then,

\[
\begin{align*}
    h_i(T_1) &= \sum_{n_1 \in N_1} I_i(n_1) \\
    h_i(T_2) &= \sum_{n_2 \in N_2} I_i(n_2) \\
    C(n_1, n_2) &= \sum_i I_i(n_1)I_i(n_2)
\end{align*}
\]
Tree Kernel Computation

\[
K(T_1,T_2) = h(T_1) \cdot h(T_2)
\]
\[
= \sum_i h_i(T_i) h_i(T_2)
\]
\[
= \sum_i \left( \sum_{n_1 \in N_1} I_i(n_1) \right) \left( \sum_{n_2 \in N_2} I_i(n_2) \right)
\]
\[
= \sum_i \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} I_i(n_1) I_i(n_2)
\]
\[
= \sum_{n_1 \in N} \sum_{n_2 \in N_2} \sum_i I_i(n_1) I_i(n_2)
\]
\[
= \sum_{n_1 \in N} \sum_{n_2 \in N_2} C(n_1,n_2)
\]
Tree Kernel Computation

- Running time: $K(T_1, T_2)$
- $O(N_1N_2)$: Based on recursive computation of $C$
Tree Kernel Computation

- Running time: $K(T_1,T_2)$
  - $O(N_1N_2)$: Based on recursive computation of $C$

- Remaining Issues:
Tree Kernel Computation

- Running time: $K(T_1, T_2)$
  - $O(N_1 N_2)$: Based on recursive computation of $C$

- Remaining Issues:
  - $K(T_1, T_2)$ depends on size of $T_1$ and $T_2$
Tree Kernel Computation

- Running time: $K(T_1,T_2)$
  - $O(N_1 N_2)$: Based on recursive computation of $C$

Remaining Issues:
- $K(T1,T2)$ depends on size of $T1$ and $T2$
- $K(T1,T1) >> K(T1,T2)$, $T1 != T2$
  - $10^6$ vs $10^2$: Very ‘peaked’
Improvements

- Managing tree size:
Improvements

- Managing tree size:
  - Normalize!! (like cosine similarity)
Improvements

- Managing tree size:
  - Normalize!! (like cosine similarity)
  
  $K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1)K(T_2, T_2)}}$

- Downweight large trees:
Improvements

- Managing tree size:
  - Normalize!! (like cosine similarity)
  - Downweight large trees:
    - Restrict depth: just threshold
    - Rescale with weight $\lambda$

\[ K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1)K(T_2, T_2)}} \]
Rescaling

- Given two subtrees rooted at n1 and n2
  - If productions at n1 and n2 are different,
    - \( C(n1,n2) = 0 \)
  - If productions at n1 and n2 are the same,
    - And n1 and n2 are preterminals,
      - \( C(n1,n2) = \lambda \)
    - Else: \( C(n_1,n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1,j),ch(n_2,j))) \)
  - \( 0 < \lambda <= 1 \)
Rescaling

- Given two subtrees rooted at n1 and n2
  - If productions at n1 and n2 are different,
    - \( C(n1,n2) = 0 \)
  - If productions at n1 and n2 are the same,
    - And n1 and n2 are preterminals,
      - \( C(n1,n2) = \lambda \)
    - Else:
      - \( C(n_1,n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1,j),ch(n_2,j))) \)
  - \( 0 < \lambda \leq 1 \)
Case Study
Parsing Experiment

- Data:
  - Penn Treebank ATIS corpus segment
  - Training: 800 sentences
    - Top 20 parses
  - Development: 200 sentences
  - Test: 336 sentences
    - Select best candidate from top 100 parses
Parsing Experiment

- **Data:**
  - Penn Treebank ATIS corpus segment
  - Training: 800 sentences
    - Top 20 parses
  - Development: 200 sentences
  - Test: 336 sentences
    - Select best candidate from top 100 parses

- **Classifier:** Voted perceptron
  - Kernelized like SVM, more computationally tractable
Parsing Experiment

- **Data:**
  - Penn Treebank ATIS corpus segment
  - Training: 800 sentences
    - Top 20 parses
  - Development: 200 sentences
  - Test: 336 sentences
    - Select best candidate from top 100 parses

- **Classifier:** Voted perceptron
  - Kernelized like SVM, more computationally tractable

- **Evaluation:** 10 runs, average parse score reported
Experimental Results

- Baseline system: 74%

- Substantial improvement: 6% absolute score

<table>
<thead>
<tr>
<th>Depth</th>
<th>Score</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>73 ± 1</td>
<td>79 ± 1</td>
</tr>
<tr>
<td></td>
<td>−1 ± 4</td>
<td>20 ± 6</td>
</tr>
</tbody>
</table>
Summary

- Parsing as reranking problem

- Tree Kernel:
  - Computes similarity between trees based on fragments
  - Efficient recursive computation procedure

- Yields improved performance on parsing task