Softwareprojektpraktikum
Maschinelle Übersetzung

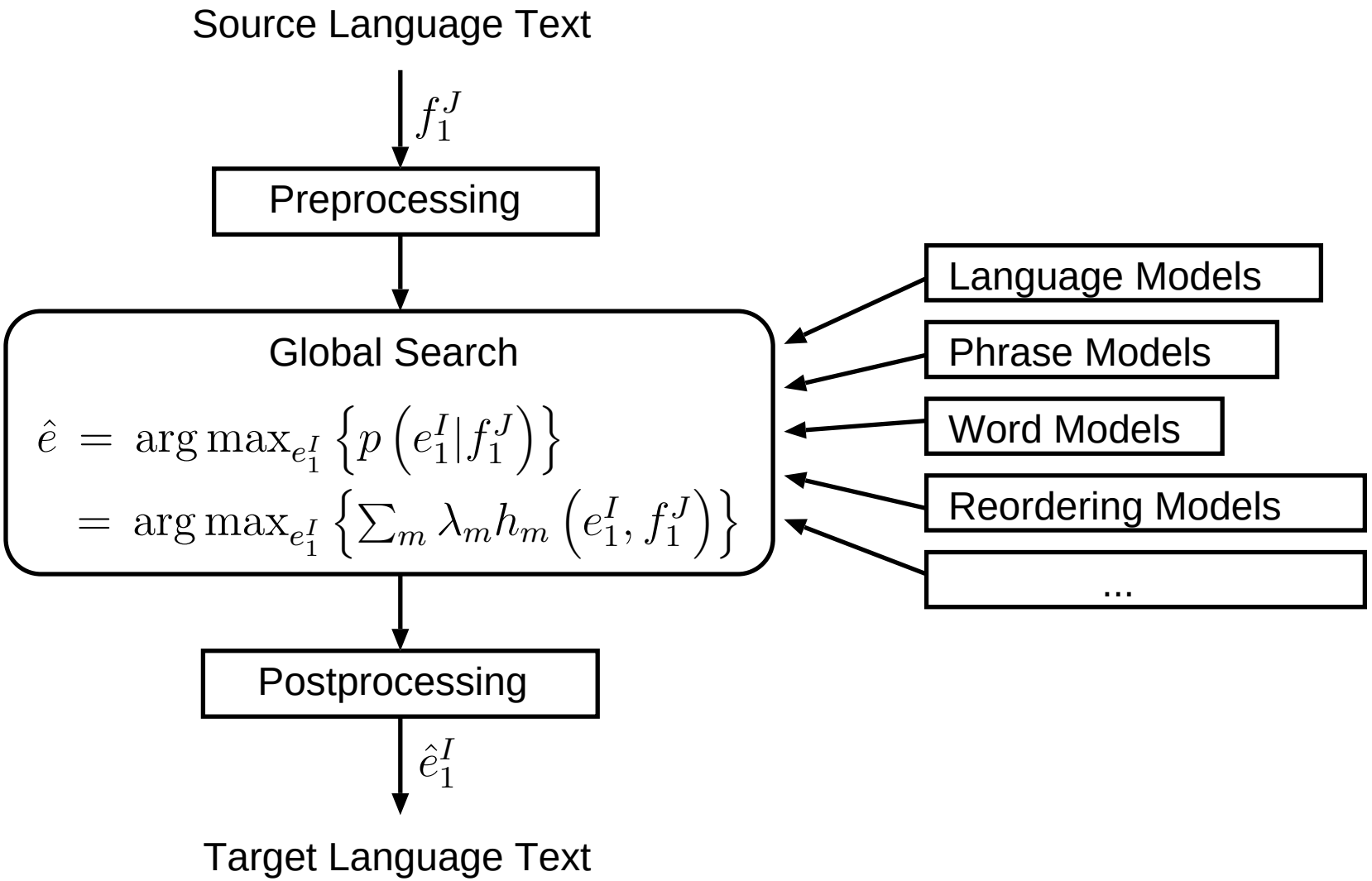
Matthias Huck, Markus Freitag
{huck,freitag}@i6.informatik.rwth-aachen.de

Vorbesprechung 5. Aufgabe 9. Juni 2011

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6
Computer Science Department
RWTH Aachen University, Germany

Contents

1 Log-lineare Modellkombination



Log-lineare Modellkombination

Mathematisch:

$$
\begin{aligned}
\hat{e}_1^{\hat{I}} &= \arg\max_{e_1^I} \left\{ p(e_1^I | f_1^J) \right\} \\
&= \arg\max_{e_1^I} \left\{ \frac{\exp\left(\sum_{m=1}^M \lambda_m h_m(e_1^I, f_1^J)\right)}{\sum_{\tilde{e}_1^I} \exp\left(\sum_{m=1}^M \lambda_m h_m(\tilde{e}_1^I, f_1^J)\right)} \right\} \\
&= \arg\max_{e_1^I} \left\{ \sum_{m=1}^M \lambda_m h_m(e_1^I, f_1^J) \right\}
\end{aligned}
$$

für Skalierungsfaktoren λ_m und Funktionen $h_m(e_1^I, f_1^J)$, $m = 1, \ldots, M$.

Optimierung der Skalierungsfaktoren

$$\hat{e}_1^{\hat{I}} = \arg\max_{e_1^I} \left\{ \sum_{m=1}^{M} \lambda_m h_m(e_1^I, f_1^J) \right\}$$

für Skalierungsfaktoren λ_m und Funktionen $h_m(e_1^I, f_1^J)$, $m = 1, \ldots, M$.

Algorithmen zur Optimierung der Skalierungsfaktoren λ_m

- **ideale Gewichtung der Einzelmodelle $h_m(e_1^I, f_1^J)$**
- **direkte Optimierung bzgl. des verwendeten Fehlermaßes**
- **finde ideale Gewichtung für in den Trainingsdaten nicht enthaltene sog. Development-Daten**
- **evaluiere Ergebnis auf ungesehenen Test-Daten**

Optimieren auf N-Best Listen

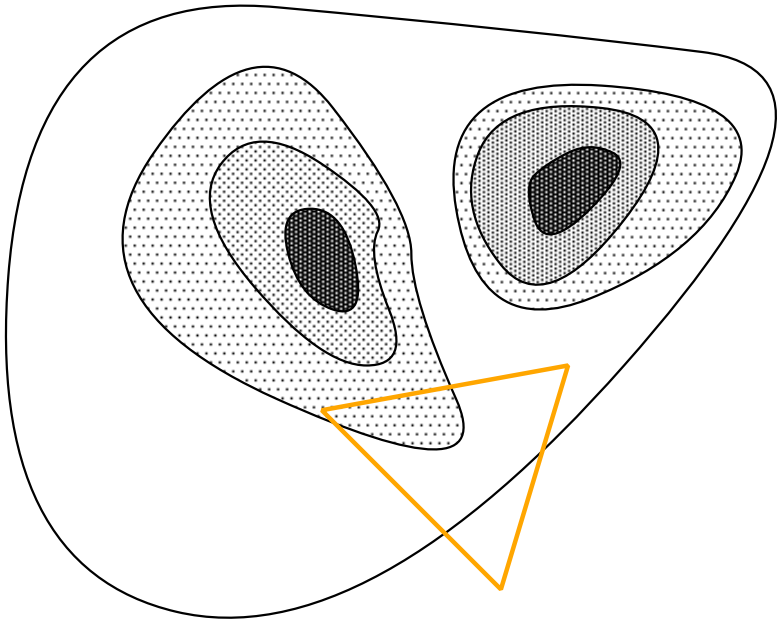
- eine Optimierung benötigt viele Iterationen
  - sehr Zeitaufwendig verschiedene Parameter Konfigurationen immer wieder neu zu Übersetzen
- optimieren auf N-Best Listen
  - wiederhole das folgende (solange neue N-Best Listen Einträge generiert werden):
    - erstelle eine N-Best Liste für jeden source Satz für die aktuellen Parameter
    - füge die neu generierten mit den alten N-Best Listen zusammen
    - optimiere die Parameter auf den zusammengefügten N-Best Listen
    - starte eine neue Iteration mit den neuen Parametern
- Vorteil
  - viel schneller
- Nachteil
  - gute Hypothesen könnten eventuell nicht betrachtet werden

2 Downhill-Simplex-Verfahren

Wie optimiert man nun die verschiedenen Skalierungsfaktoren?

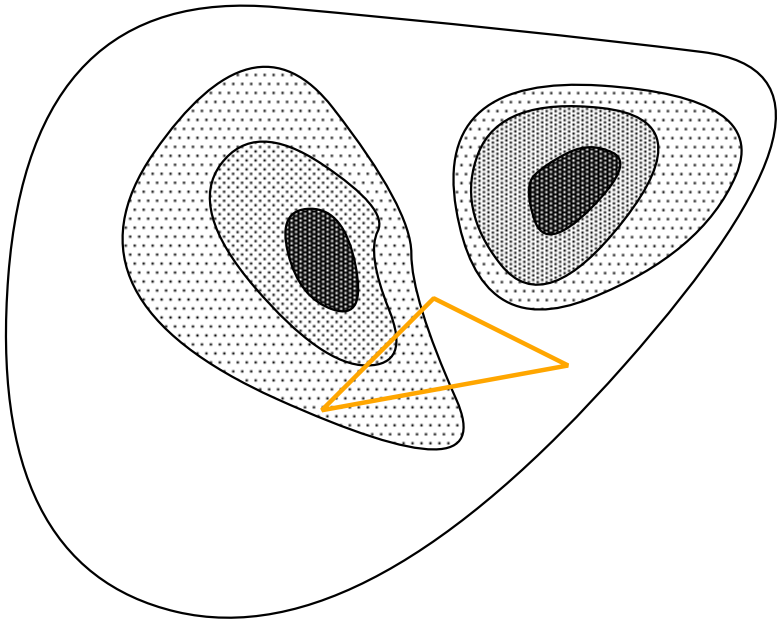
- **Downhill Simplex**
  - **Hillclimbing-Verfahren**
  - **langsame Konvergierung, aber relativ robust**
  - **Aufstellen eines $M+1$-Dimensionalen Vielkants (Simplex)**
  - **Schrittweise Verbesserung des schlechtesten Punktes**
- **zu verbessernde Kosten: z.B. $BLEU$**

Beispiel



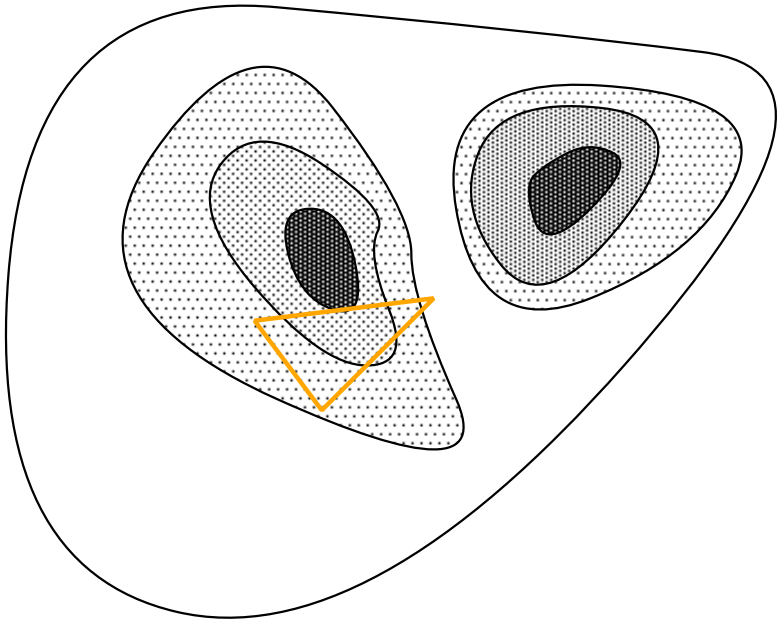
Beispielverlauf einer Simplex-Optimierung

Beispiel



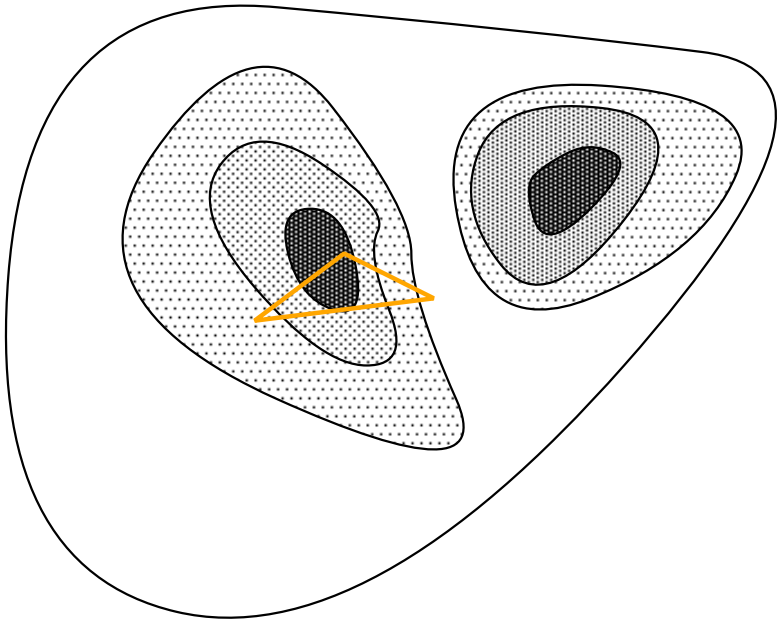
Beispielverlauf einer Simplex-Optimierung

Beispiel



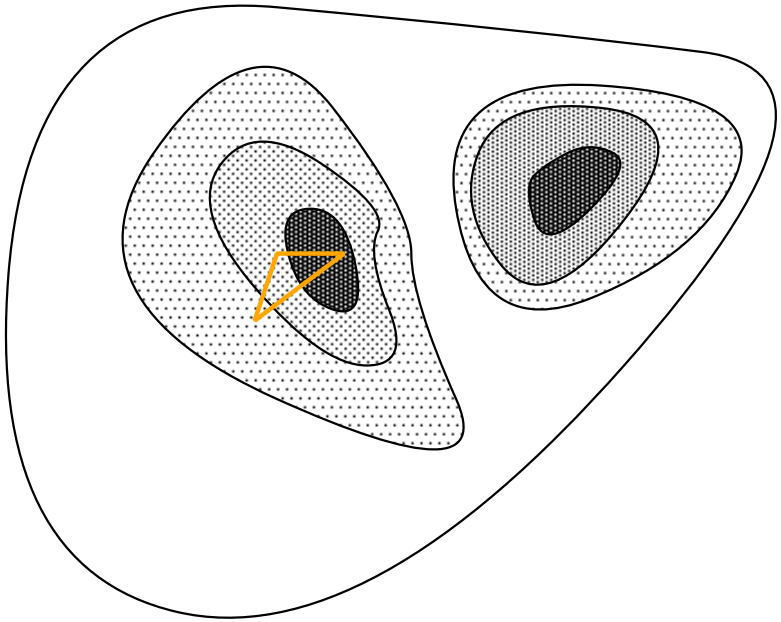
Beispielverlauf einer Simplex-Optimierung

Beispiel



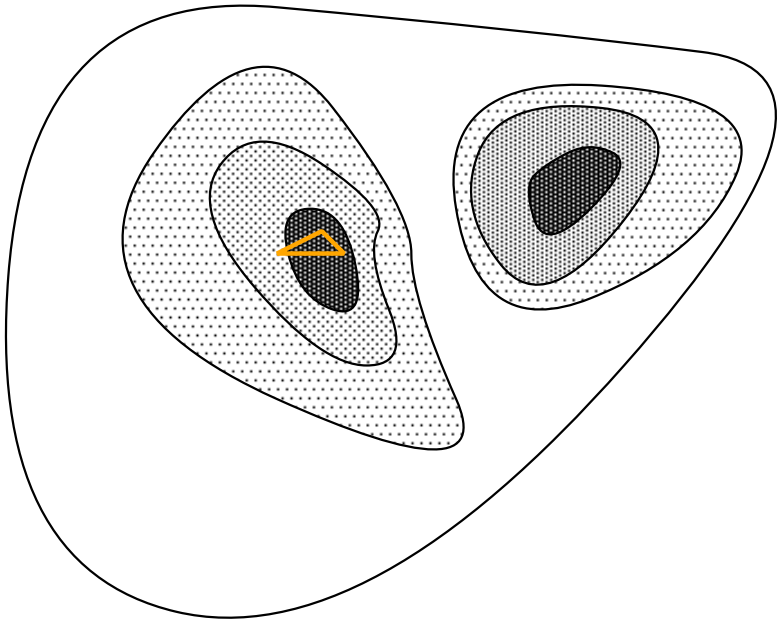
Beispielverlauf einer Simplex-Optimierung

Beispiel



Beispielverlauf einer Simplex-Optimierung

Beispiel



Beispielverlauf einer Simplex-Optimierung

Algorithmus

Initialisierung:

- **Simplex aufspannen: $M + 1$ Werte berechnen**

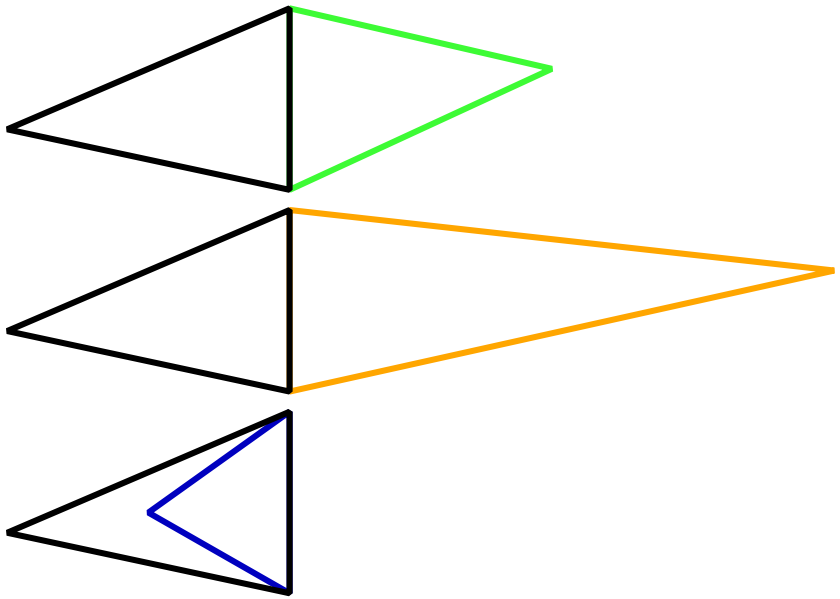
In jeder Schleife:

- **Werte sortieren, Schwerpunkt berechnen**
- **versuchen, den schwächsten Punkt zu verbessern**
- **im Notfall den Simplex zusammenziehen**

Schwächsten Punkt verbessern

- **Berechnung des Simplex-Schwerpunktes** $x_0 = \frac{\sum_{m=1}^{M} x_i}{M}$
- **Verlagerung des schlechtesten Wertes**

- **durch Reflektion:**
  $x_r = x_0 + \alpha(x_0 - x_{worst})$
- **durch Erweiterung:**
  $x_e = x_0 + \gamma(x_0 - x_{worst})$
- **durch Kontraktion:**
  $x_c = x_{worst} + \rho(x_0 - x_{worst})$



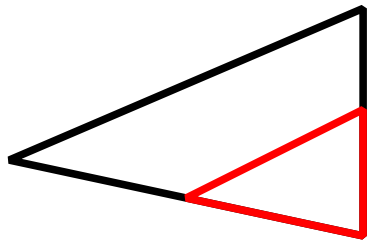
- **typische Werte:** $\alpha = 1.0, \gamma = 2.0, \rho = 0.5$

Schwächsten Punkt verbessern

- nicht möglich den schwächsten Punkt zu verbessern?

$\Rightarrow$ ziehe den ganzen Simplex zusammen

  - $x_i = x_1 + \sigma(x_i - x_1) \qquad \forall i \in \{2, \ldots, M+1\}$



- typische Werte: $\sigma = 0.5$

- die Optimierung endet, wenn
  - abs $(BLEU(x_i) - BLEU(x_j)) \leq \epsilon$
  - oder wenn z.B. eine bestimmte Anzahl an Iterationen erreicht wurde

Downhill Simplex

Algorithm 1: Downhill Simplex

while *Optimierungskriterium nicht erreicht* **do**

SORTIERE Simplexpunkte nach Wert: $\boldsymbol{BLEU}(x_1) \geq \boldsymbol{BLEU}(x_2) \geq \cdots \geq \boldsymbol{BLEU}(x_{M+1})$;

BERECHNE Schwerpunkt x_0 aller Punkte ausser x_{M+1} ;

if $\boldsymbol{BLEU}(x_1) \geq \boldsymbol{BLEU}(x_r) \geq \boldsymbol{BLEU}(x_M)$ **then**

REFLEKTIERE, d.h. ersetze x_{M+1} durch x_r;

else if $\boldsymbol{BLEU}(x_r) > \boldsymbol{BLEU}(x_1)$ **then**

if $\boldsymbol{BLEU}(x_e) \geq \boldsymbol{BLEU}(x_r) \geq \boldsymbol{BLEU}(x_1)$ **then**

EXPANDIERE, d.h. ersetze x_{M+1} durch x_e;

else

REFLEKTIERE, d.h. ersetze x_{M+1} durch x_r;

else // $\boldsymbol{BLEU}(x_r) < \boldsymbol{BLEU}(x_M)$

if $\boldsymbol{BLEU}(x_c) \geq \boldsymbol{BLEU}(x_{M+1})$ **then**

KONTRAHIERE , d.h. ersetze x_{M+1} durch x_c;

else

KOMPRIMIERE Simplex, $x_i = x_1 + \boldsymbol{\sigma}(x_i - x_1) \qquad \forall i \in \{2, \ldots, M+1\}$;

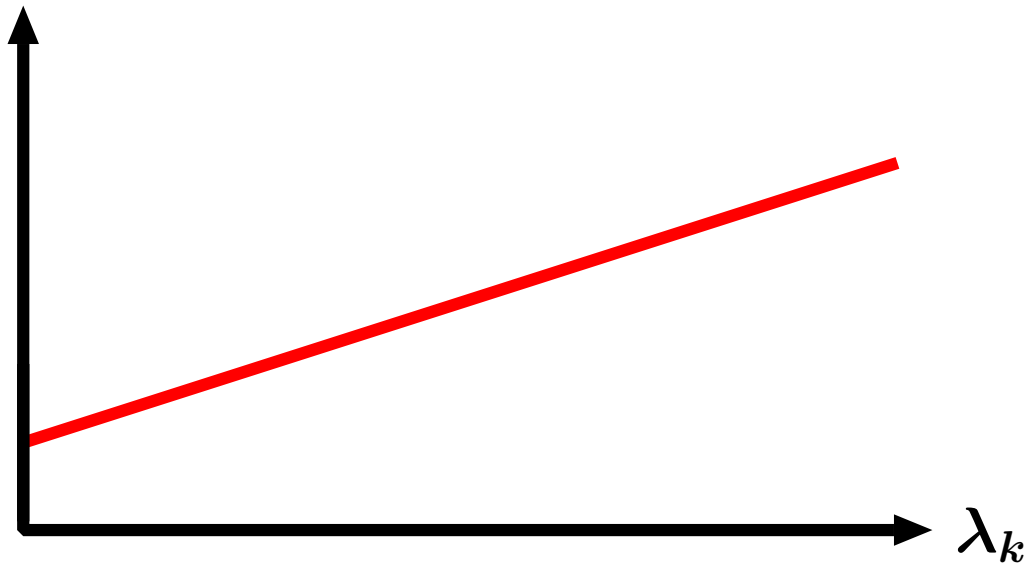
3 Och's Minimum Error Rate Training (MERT)

- Basiert auf der Powell's Methode
- Üblicherweise schneller und stabiler als Downhill Simplex
- Ansatz: es wird jeweils nur ein Parameter λ_k auf einmal verändert
- Idee: nicht jede Position wird berechnet, sondern nur die wesentlichen

$$\sum_m \lambda_m h_m(e_1^I, f_1^J) = \sum_{m \neq k} \lambda_m h_m(e_1^I, f_1^J) + \lambda_k h_k(e_1^I, f_1^J)$$

Beispiel

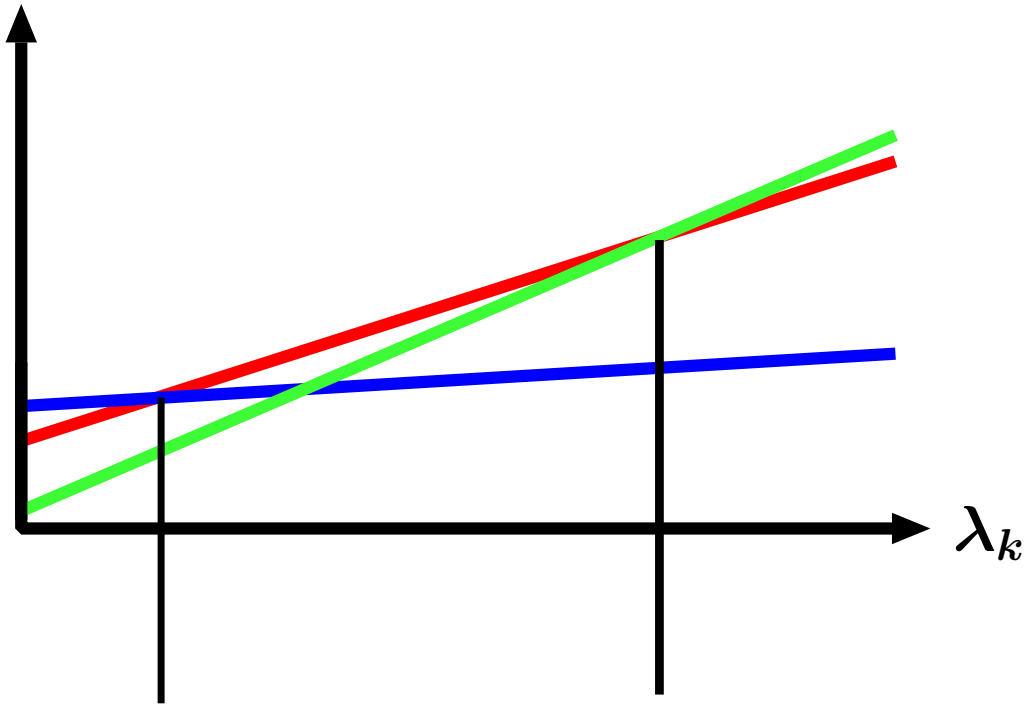
$$\sum_{m=1}^{M} \lambda_m h_m(E_1^I, F_1^J)$$



Gesamtkosten von einer einzigen Übersetzung

Beispiel

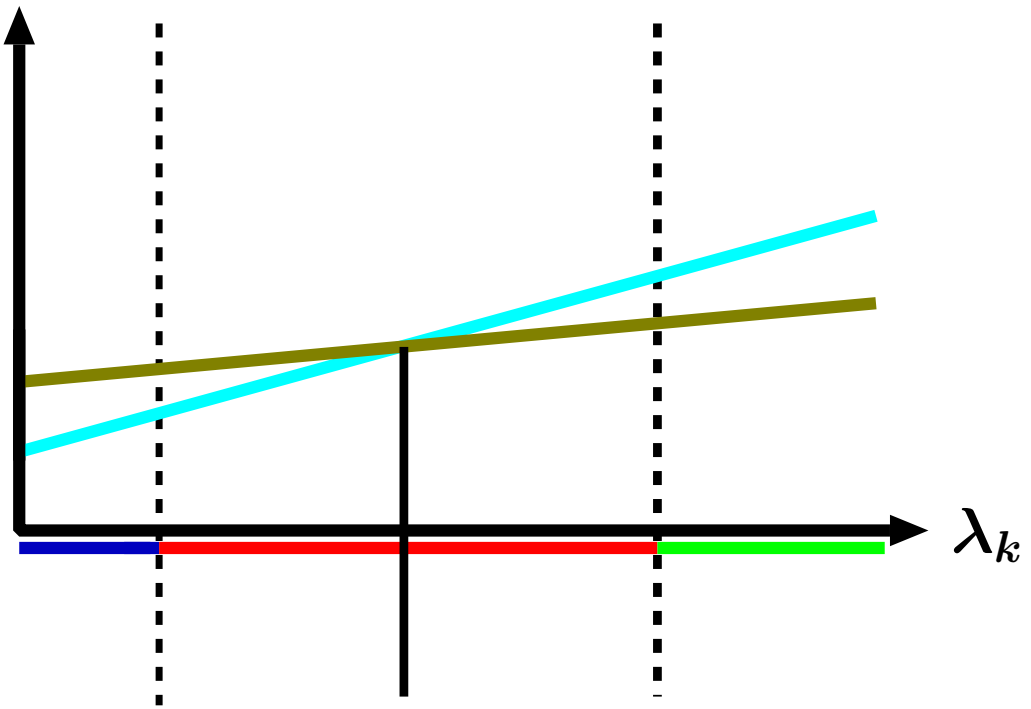
$$\sum_{m=1}^{M} \lambda_m h_m(E_1^I, F_1^J)$$



Schnittpunkte der N-Best Hypothesen für Satz Eins

Beispiel

$$\sum_{m=1}^{M} \lambda_m h_m(E_1^I, F_1^J)$$



Schnittpunkte der N-Best Hypothesen für Satz Zwei

Update eines Parameters

- berechne $BLEU$ an jedem Schnittpunkt (bzw. in der Mitte zwischen 2 Schnittpunkten)
- update den aktuellen Parameter auf den Wert des Schnittpunktes mit dem geringsten $BLEU$
- optimiere den nächsten Parameter

Schnittpunkt-Berechnung

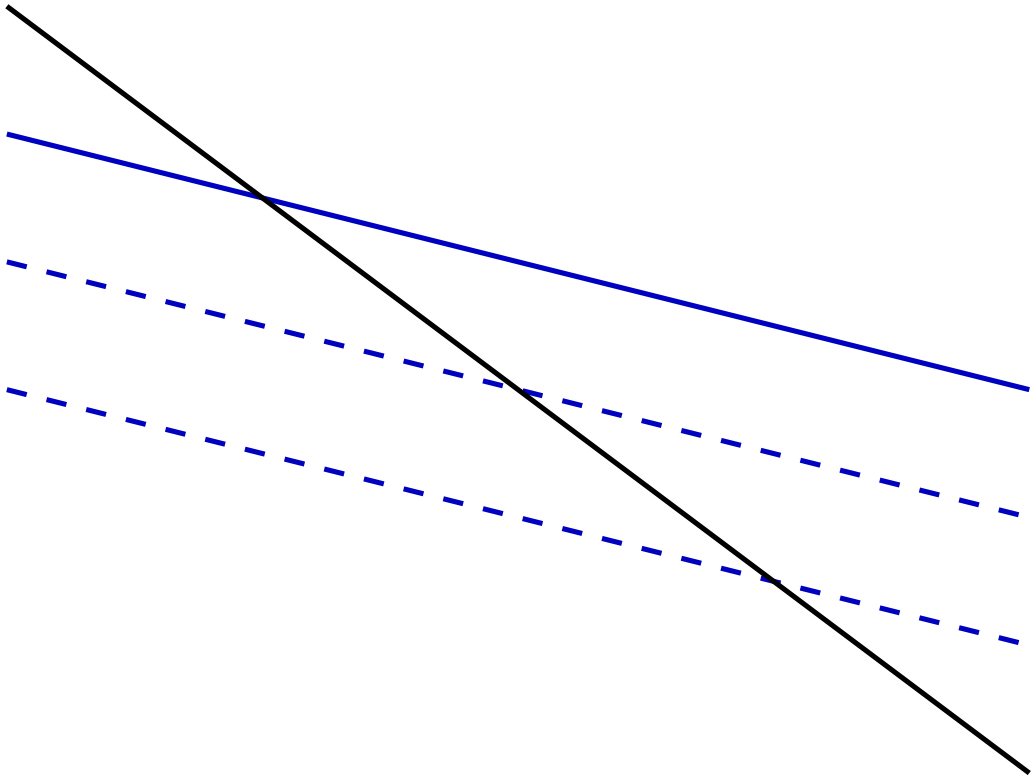
Offset = y-Wert der Geraden bei (z.B.) $x = 0$
Steigung = Differenz der y-Werte bei z.B. $x = 0$ und $x = 1$

$$(\textbf{Offset}(b) - \textbf{Offset}(a))/(\textbf{Steigung}(a) - \textbf{Steigung}(b))$$

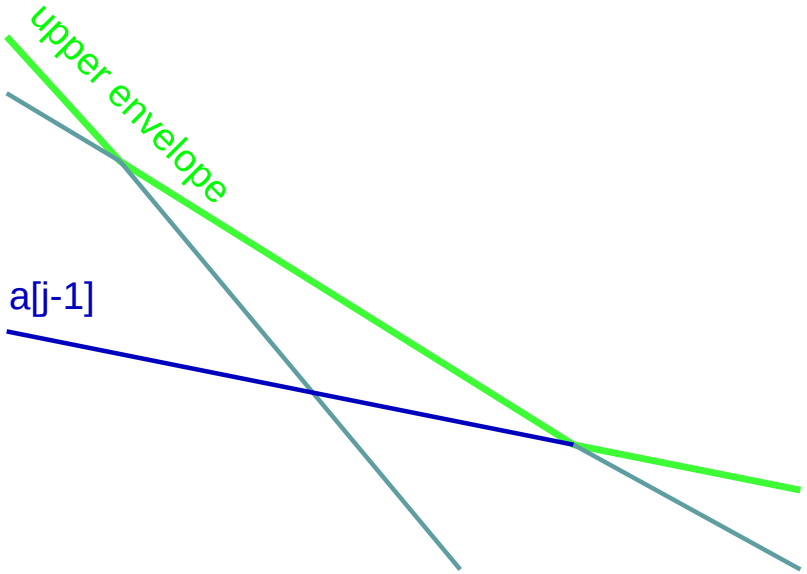
Grundalgorithmus:

- **sortiere Geraden nach Steigung**
- **warum? -> die Gerade mit der geringsten Steigung ist die Gerade bei $-\infty$**
- **berechne Schnittpunkte von links nach rechts**
- **die Gerade mit der größten Steigung ist die Gerade bei ∞**

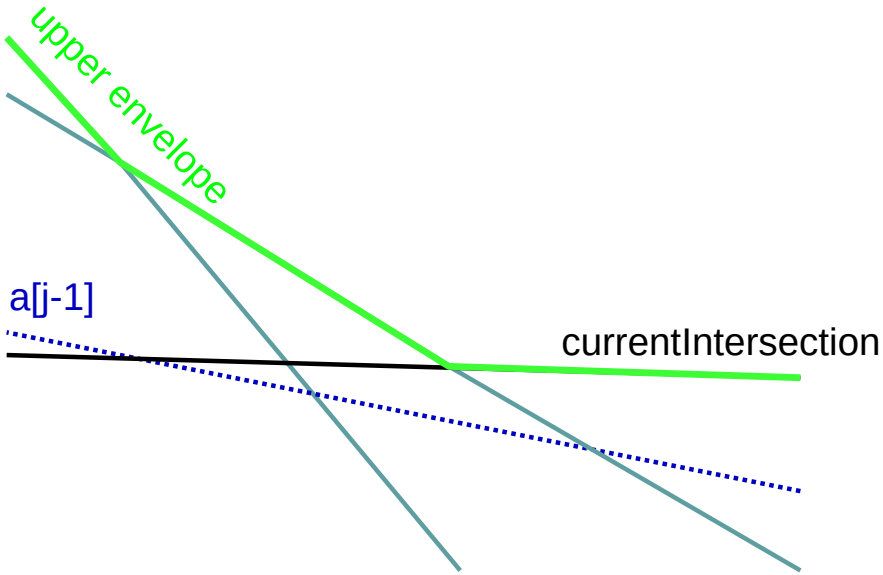
Eliminiere niedriger gelegene Linien



Vergleiche Schnittpunkte



Vergleiche Schnittpunkte



Algorithmus

Algorithm 2: MERT

```
INITIALISIERE;
while Neue Hypothese produziert wird do
    while Parameter ändern sich do
        Berechne neue (zufällige) Parameter-Reihenfolge;
        for Jeden Parameter do
            Berechne alle Schnittpunkte aller N-Best Listen;
            Berechne BLEU neu für alle Schnittpunkte (effizient!);
            Wähle besten Parameter;
    Erzeuge neue N-Best Liste mit neuen Parametern;
    Füge die neuen N-Best Einträge zu der alten Liste hinzu
```

4 Übung 5

- **Optimierung der Skalierungsfaktoren in der log-linearen Modellkombination**
  - **Downhill Simplex**
  - **MERT**
    - **Effiziente Schnittpunktberechnung**
    - **Effiziente $BLEU$ Berechnung**
    - **ist es notwendig an jedem Schnittpunkt $BLEU$ komplett neu zu berechnen?**
    - **Begrenzung der Veränderung eines Parameters?**
  - **Normierung der Parameter?**
  - **N-Best Listen Größe?**
  - **weitere Ideen?**

Berechnung der Konvexen Hülle

Algorithm 3: Line Sweep Algorithmus

input : Array a der Größe K mit Linien sortiert nach Steigung. $j = 0$

for $(i = 0 \ldots K - 1;)$ **do**
- SETZE currentIntersection = $a[i]$;
- SETZE currentIntersection.previous = minimum;
- **if** $0 < j$ **then**
  - **while** *Steigung von* $a[j-1]$ *== Steigung von currentIntersection, aber Offset geringer* **do**
    - VERRINGERE j;
  - **while** $(0 < j)$ **do**
    - SETZE currentIntersection.previous auf Schnittpunkt mit $a[j-1]$;
    - **if** $a[j-1]$*.previous < currentIntersection.previous* **then**
      - break;
    - VERRINGERE j;
  - **if** *0 == j* **then**
    - currentIntersection.previous = minimum;
    - $a[inc(j)]$ = currentIntersection;
- **else**
  - $a[inc(j)]$ = currentIntersection;

GIB $a[0 \ldots j]$ als Lösung;

Fragen?

Viel Erfolg!