Michel Krebs

Phonem-Graphem Konversion für deutsche Wörter

Studienarbeit SA-2004-15
Wintersemester 2003/2004

Betreuer: René Beutler

Verantwortlicher:
Prof. Dr. Lothar Thiele

6.2.2004

Inhalt:

1. Problemstellung

Das Ziel der Arbeit war es, ein Verfahren zu implementieren, welches aus einer gegebenen Sequenz von Phonemen ohne Lexikonwissen das dazugehörende Wort generieren sollte. Hierbei sollten die Abbildungen von Phonemen auf Grapheme, d.h. Buchstaben, betrachtet werden.
Nun ist allerdings die Zuordnung von Phonemen auf Grapheme nicht immer Eindeutig. So kann ein /i:/ im Wort "mir" auf ein i, im Wort "die" auf ein ie, im Wort "ihn" auf ein ih und im Falle von "ziehen" auf ein ieh abgebildet werden.. Die Zuordnungen sind also abhängig vom Kontext.
Das erste Problem ist nun herauszufinden welche Grapheme es gibt, und von welchen Phonemen sie abgebildet werden können.
Sobald die gültigen Abbildungen bekannt sind, muss eine Methode gefunden werden, die das Problem der Wortfindung möglichst gut lösen kann, in diesem Fall eine statistische Methode, da das System ohne Kenntnis von existenten Worten arbeiten soll.

2. Mögliche Phonem-Graphem Abbildungen finden

2.1 Die Grapheme

Zu Beginn der Arbeit stand mir ein Lexikon zur Verfügung, dessen Einträge von der Form [<Wort> <Phonemsequenz>] waren, wobei die Phonemsequenz aus einer nicht getrennten Aneinanderreihung von Phonemen bestand. Das Lexikon umfasste rund 300`000 Einträge.

Die einzelnen Phoneme aus der Phonemsequenz zu bekommen war einfach, da die verschiedenen Phoneme eine Länge von nur ein oder zwei Zeichen haben, konnte ich durch einfaches Überprüfen ob der Substring der Länge zwei ein gültiges Phonem ist, direkt bestimmen welches das nächste Phonem im Wort ist und so die Phoneme zuverlässig finden und Fälle wie /i:/ und /i/ auf einfache Weise unterscheiden, vor allem da keine Phoneme existieren die geteilt auch wieder zwei eigene Phoneme bilden.

Mit den Graphemen war das schwieriger weil sie abhängig von den zugehörigen Phonemen sind. Um ein Beispiel zu geben, kann "ie" als /i:/ aber auch als Folge /I/,/@/ interpretiert werden, was dann aus dem "ie" zwei Grapheme "i" und "e" ergibt.

Ein weiteres Problem war der Sonderfall "z", der phonetisch aus zwei Teilen /t/,/s/ besteht. Das Problem ist, dass man dem /t/ und dem /s/ ein Graphem zuordnen soll, aber nur ein Graphem für zwei Phoneme da ist. Es gibt für die Lösung dieses Problems zwei Möglichkeiten, erstens, man fasst /t/ und /s/ in ein Phonem /ts/ zusammen und gibt ihm die Möglichkeit auf "z" und "ts" abgebildet zu werden, oder man lässt zu, dass einer der beiden Phoneme auf nichts Abgebildet wird. Hierbei wird dann im Graphem ein Sonderzeichen eingefügt, welches am Schluss wieder aus dem Wort gelöscht werden kann. Ich wählte die zweite Variante, indem ich die Abbildung von /t/ auf "z" und von /s/ auf "_" erlaubte, und in jedem Wort, in dem diese Abbildungen vorkamen nach dem "z" das Sonderzeichen einfügte.

2.2 Die Editierdistanz (ED)

Um die Phonem-Graphem-Abbildungen zu finden implementierte ich erst eine Methode, mit der die Editierdistanz zwischen einer Phonemsequenz und einer Sequenz von Graphemen berechnet werden konnte. Nun, um die Editierdistanz für eine Phonemsequenz und eine Graphemsequenz berechnen zu können, müssen die erlaubten Abbildungen der Phoneme auf die Grapheme bekannt sein, also gab ich hier die offensichtlichen Abbildungen bereits in die Bibliothek der gültigen Abbildungen ein. Das half mir auch die Ergebnisse der Methode einfacher zu überprüfen.
Bei der Berechnung der ED setzte ich für die Strafen 3 für das Einfügen oder Weglassen von Graphemen, 5 für eine nicht erlaubte Abbildung und Null wenn es eine gültige Abbildung war. Wichtig waren hierbei allerdings nicht die Werte die ich gewählt hatte, denn da ich nach allen gültigen Abbildungen suchte die im Lexikon vorkamen, war nur die Tatsache wichtig, dass ein Match Null Strafpunkte und jeder andere Fall einen Wert grösser Null zugeordnet bekam. Bei einem Wort im Lexikon das eine Editierdistanz von Null erhielt, konnte ich sicher sein, für dieses Wort alle gültigen Abbildungen gefunden zu haben. Für den Fall, dass die ED aber nicht Null war, mussten noch weitere Abbildungen definiert werden.

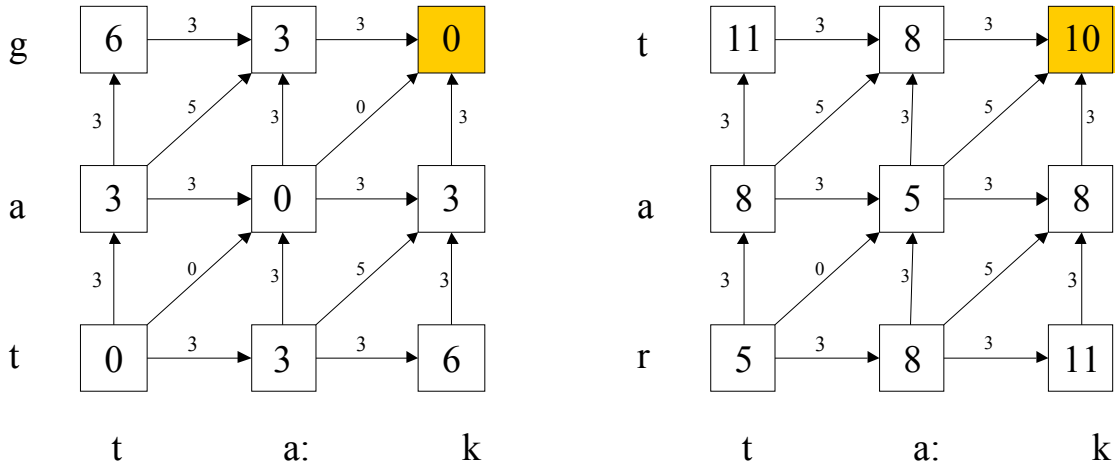


Abb. 1

Vergleich zweier Editierdistanzen für gleiche Phoneme mit verschiedenen Graphemen

Mein erster Ansatz für die Methode zur Berechnung der Editierdistanz war prinzipiell einfach, aber ein wenig zu aufwendig in der Ausführung. Die Idee war, in einem 2-dimensionalen Gitter, in dem Phoneme Graphemen zugeordnet werden einen Pfad zu finden so, dass die Editierdistanz minimal wird.
Hierbei ging ich folgndermassen vor: Da von einem Punkt aus nur drei Richtungen möglich sind um weiterzugehen nämlich waagrecht, senkrecht und diagonal, habe ich in jedem Punkt erst einmal drei Werte definiert, die Kosten um in jede der drei beschriebenen Richtungen weiter zu gehen. Diese Werte berechnete ich, indem ich für die Bewegung in die waagrechte und die senkrechte Richtung zuerst einmal die Kosten drei gab. Dann musste in jedem Punkt überprüft werden, ob die Abbildung im jeweiligen Feld erlaubt ist, wenn nein, mussten die Kosten um diagonal auf dieses Feld zu kommen auf fünf gesetzt werden (was bedeutet, im Feld diagonal darunter, da die Werte sich auf die Kosten beziehen wenn man weiter geht). Dann überprüfte ich jeweils, ob das zusammenfügen von Graphemen oder Phonemen zu einem eigenen Symbol jeweils erlaubt war. Das geschah, indem ich jeweils in die betreffende Richtung ging und nach gültigen Symbolen suchte (da ich zu diesem Zeitpunkt noch nicht sicher war, ob ich später zusammengefasste Phoneme verwenden wollte, suchte ich ebenfalls nach zusammengefassten Phonemen). Wurde ein oder mehrere Grapheme oder Phoneme auf diese Weise gefunden, sollten die dazugehörenden Kosten ebenfalls Null werden. Hierfür durfte allerdings nicht vergessen werden, ebenfalls den Pfad, der diagonal zum ersten Phonem oder Buchstaben des Symbols führt Null zu setzen (auch wenn genau dieser eventuell selber einer nicht gültigen Abbildung entspricht) und die Pfade, welche an Orten vom Symbol diagonal weg führen an denen es nicht zu ende sein kann, mit Kosten für falsche Zuordnungen zu versehen.

Nachdem die Kostenmatrizen berechnet waren, ging ich mit Hilfe von dynamischem Programmieren an die Berechnung der minimalen Editierdistanz. Hierfür ging ich einmal durch

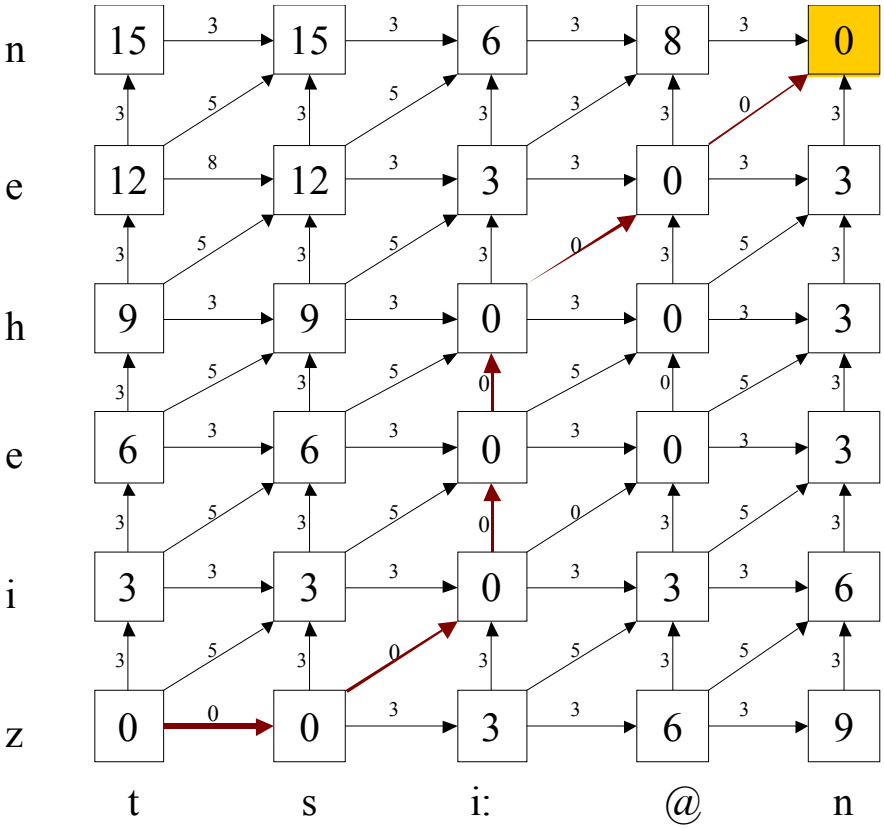


Abb. 2: Zusamenfassen von Graphemen und Phonemen

das gesamte Gitter, jeweils von einem Phonem alle Grapheme dann vom nächsten Phonem aus alle Grapheme und so weiter. In jedem Punkt wurden für die nächsten drei möglich folgenden Punkte die bisher minimale Strafe berechnet indem die für den momentan besetzten Punkt minimale Strafe mit der Strafe des Pfads zu dem jeweiligen Knoten addiert wurde. Da ich durch den gut gewählten Weg durch das Gitter immer nur auf Knoten landete, deren minimale Strafe bereits berechnet war, musste ich mir keine Gedanken darüber machen, was geschehen müsste, wenn ein schon betretener Knoten eine neue minimale Strafe erhielte. Die Editierdistanz konnte dann im letzten Feld des Gitters abgelesen werden.
Um aus dem geprüften Wort nun die jeweiligen Abbildungen zu finden war dann auch nur noch etwas Rechnerei nötig. Aus den minimalen Strafen in den Knoten und den Strafen die über die Bewegung von Knoten zu Knoten entstehen, liess sich der Pfad für die minimale Editierdistanz rückwärts berechnen. Da für jeden Knoten gelten musste, dass die Summe der minimalen Strafe des Vorgängerknotens addiert mit der Strafe des Weges zum aktuellen Knoten mit der minimalen Strafe im aktuellen Knoten übereinstimmt.

Eine kleine Bemerkung wäre noch zu den Möglichen Pfaden zu machen, denn ein gültiger Pfad kann niemals einen rechten Winkel haben, sprich von unten her in einen Knoten kommen und dann horizontal wieder hinaus oder horizontal in einen Knoten hinein und nach oben hinaus, da in diesem Fall das Graphem und das Phonem die zu diesem Konoten gehören in zwei verschiedenen Abbildungen vorkommen würden, was natürlich nicht sein kann. Dieses Phänomen ist im bezug auf das Berechnen der Editierdistanz nicht unbedingt wichtig, allerdings muss man beim Rückrechnen der Editierdistanz damit rechnen, dass ein möglicher Pfad einen rechten Winkel haben und dieser Pfad dann ignoriert werden muss.

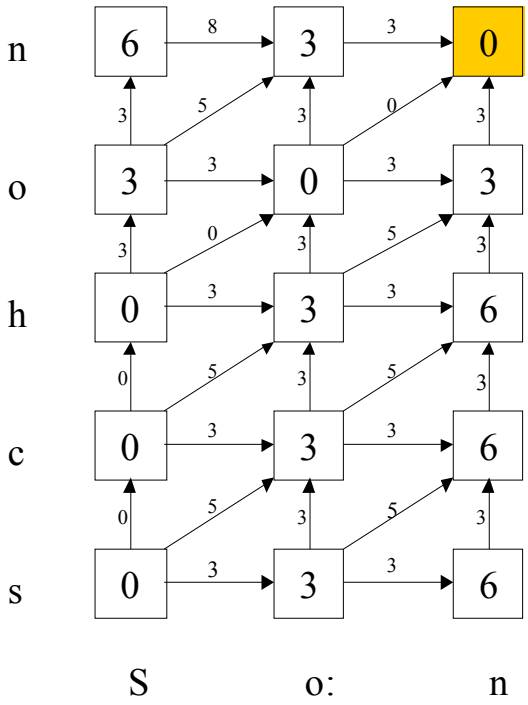


Abb. 3: Zusamenfassen von Graphemen

Ursprünglich wollte ich das Finden neuer Abbildungen automatisieren, aber leider traten bei diesen Versuchen zu viele Fehler auf. Das Problem war, dass bei einer Einfügung oder Löschung verschiedene Fälle auftreten konnten. Selbst wenn alle Abbildungen bis auf eine in einem Wort bekannt wären, bliebe die Frage, ob man einen Buchstaben zuviel zum vorhergehenden oder zum folgenden Graphem hinzunehmen soll.
Wenn ein Graphem zuviel erkannt wird, z.B. /h/ → h, /a:/ → a, /l/ → l und /o:/ → o sind bereits bekannt, und der Eintrag im Lexikon lautet: "hallo" für das Wort und "halo" für die Phonemsequenz. Nun, wohin mit dem l das zuviel ist? Die neue Abbildung muss /l/ → ll lauten, aber aus Sicht der Maschine sind ebenfalls /a:/ → al und /o:/ → lo möglich.
Nun für verschiedene Fälle habe ich Regeln eingeführt, zum einen eine Regel, die Verdoppelungen erkennen soll, also hier zum Zuge käme, eine andere war das Anhängen von stummen hs, da ein stummes h immer hinten an ein Graphem (meistens ein Vokal) gehängt wird.
Leider reichten diese Regeln nicht ganz aus, da überhaupt erst einmal Abbildungen bekannt sein müssen, um ein einigermassen anständiges Finden neuer Abbildungen zu ermöglichen. Also habe ich die offensichtlichen Abbildungen vorgegeben und einen Versuch gestartet und geschaut, welche Abbildungen gefunden wurden. Das ganze hat nicht schlecht aber auch nicht völlig überzeugend funktioniert.
Das Lexikon Fehler in manchen Einträgen, hauptsächlich waren es Umlaute die fehlten oder zuviel waren, vor allem bei Worten die sowohl mit als auch ohne Umlaut existieren. Also waren selbst dann falsche Abbildungen zu erwarten, wenn die Methode perfekt funktioniert hätte. Also habe ich mich dann dazu entschlossen die Abbildungen von Hand vorzugeben und zu schauen welche Einträge mit welchen Abbildungen eine ED ungleich Null erhielten, um die Liste der gültigen Abbildungen dann zu ergänzen.

Für dieses Vorgehen verwendete ich dann aber eine andere Methode die Grapheme zu erkennen , da die Wartezeiten beim Berechnen der Editierdistanzen über alle Einträge enorm waren (etwa 10 Minuten). Diese neue Methode war vom Prinzip als auch vom programmieren her einfacher, ich ging davon aus, dass Phoneme nicht zusammengefasst werden können, was das ganze vereinfachte, allerdings dazu führte, dass ich nun den Fall der Abbildung /ts/ auf z berücksichtigen musste und dann jeweils nach dem z ein „_“ einfügen. Betrachtet wurde dabei jeweils das vorderste nicht zugewiesene Phonem, mit dem ich die längste gültige Abbildung auf der Graphemsequenz suchte. Hatte ich diese gefunden, wurde das nächste Phonem betrachtet und so weiter. Fand nun ein Phonem keine gültige Abbildung, ging ich zum vorhergehenden Phonem zurück und suchte dort die nächst kleinere gültige Abbildung, bis allen Phonemen gültige Abbildungen zugeordnet waren, oder klar war, dass die Editierdistanz mit den gegebenen Abbildungen nicht Null werden kann.

Nachdem ich die Abbildungen gefunden hatte, musste ich aus den gewonnenen Daten über die Abbildungen ein Trainingsset und ein Testset extrahieren. Das geschah über Wortstämme. Ich hatte ein Lexikon bekommen, das in jedem Eintrag eine Zahl enthielt, die den Worten jeweils einen Wortstamm zuordnete. Ich wählte nun die Worte so aus, dass Worte mit gleichem Wortstamm ins gleiche Set geschrieben wurden, um zu ermöglichen, dass das Testset möglichst unabhängig vom Trainingsset war. Da das Lexikon etwa 300'000 Einträge hatte und ein Testset der Grösse 10'000 ausreichend sein sollte, wurde jeder Eintrag, dessen Wortstamm durch 30 Teilbar ist ins Testset geschrieben, alle anderen ins Trainingsset.

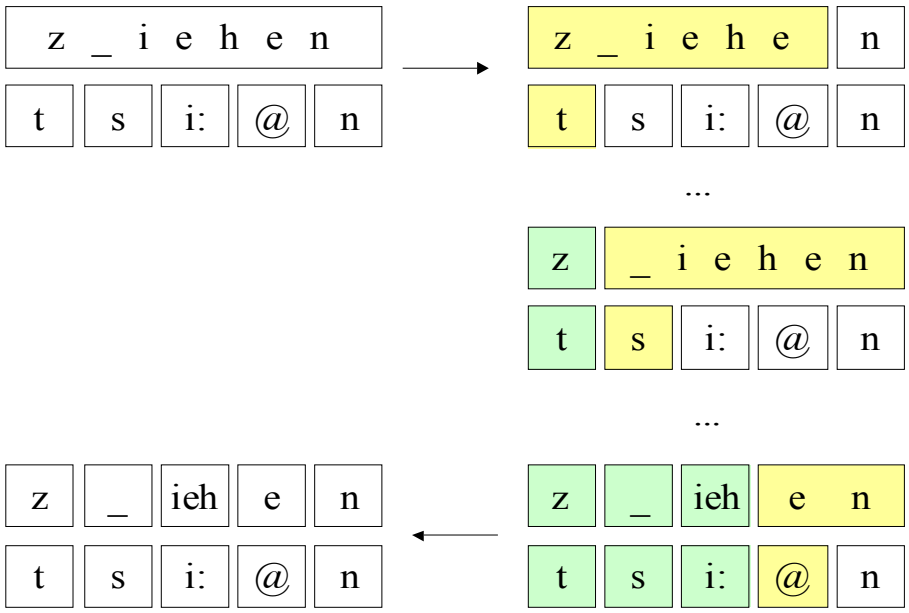


Abb. 4: Grapheme finden durch geeignete Suche

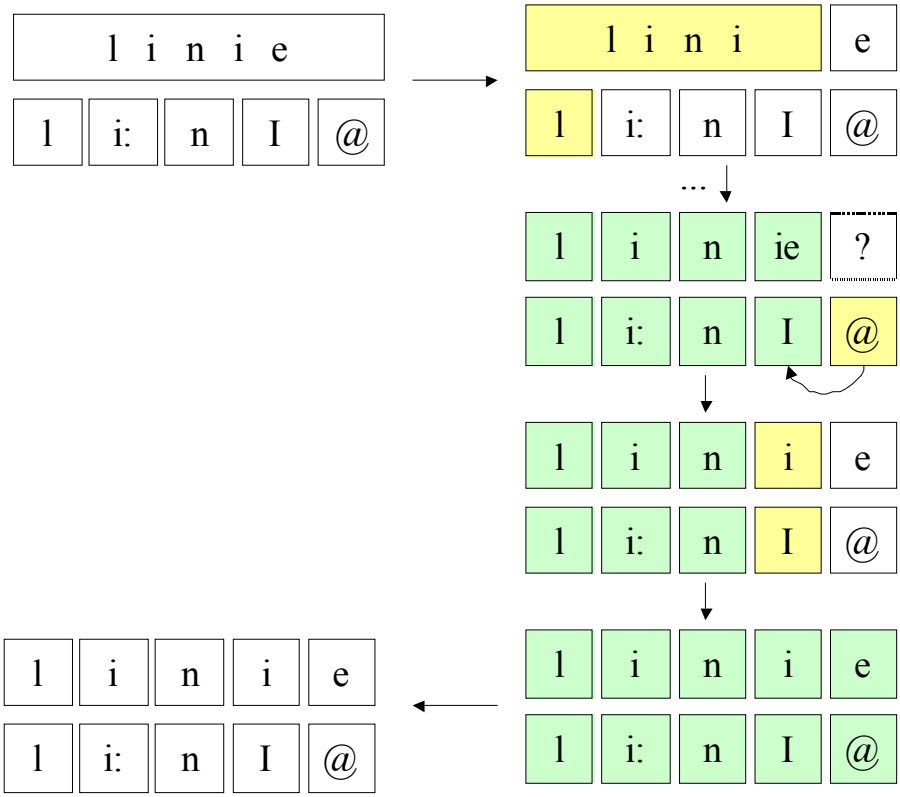


Abb. 5: Zurückgehen bei ungültigen Abbildungen

3. Welche Abbildung ist die Richtige?

Die gewonnenen Daten enthielten nun die eindeutigen Phonem-Graphem Abbildungen für die im Lexikon enthaltenen Worten. Nun musste erst einmal ein Verfahren gewählt werden, mit dem man die gestellte Aufgabe, die richtigen Abbildungen zu finden, am besten lösen kann. Dies liess sich mathematisch als ein Problem schreiben, bei dem die Wahrscheinlichkeit des Resultats, abhängig von den gegebenen Informationen maximiert wird.

$$\max_i P(G_i \mid L)$$

Wobei G_i einer Graphemsequenz und L einer gegebenen Lautsequenz entspricht.

3.1 Wahrscheinlichkeit

Um das Ziel der Arbeit zu erreichen habe ich mich dafür entschlossen ein Verfahren zu benutzen, bei dem die notwendigen Wahrscheinlichkeiten durch Auszählen der benötigten Grössen approximiert werden. Die so gewonnenen Statistiken sollten in der Lage sein, die Wahrscheinlichkeit des erzeugten Wortes abhängig von der Phonemsequenz als Input zu maximieren.
Was also gemacht werden soll, ist *P(G|L)* zu maximieren, wobei *G* eine Sequenz von Graphemen $\{g_1, g_2,... ,g_n\}$ ist und L eine Sequenz von Phonemen $\{l_1, l_2,... ,l_n\}$. Betrachten wir aber erst einmal *P(G)*:

$$\begin{aligned} P(g_1,...,g_n) &= P(g_1 \mid g_2,...,g_n)P(g_2,...,g_n) \\ &= P(g_1 \mid g_2,...,g_n)P(g_2 \mid g_3,...,g_n)P(g_3,...,g_n) \\ &= \prod_{i=1}^{n} P(g_i \mid g_{i+1},...,g_n) \end{aligned}$$

Für *P(G|L)* kann man nun, da die Bedingung der Phonemsequenz nichts anderes als ein anderer, kleinerer Raum in dem die Menge *G* sich befinden kann bedeutet, auf die gleiche Art schreiben:

$$\begin{aligned} P(G \mid L) = P(g_1,...,g_n \mid L) &= P(g_1 \mid g_2,...,g_n,L)P(g_2,...,g_n,L) \\ &= \prod_{i=1}^{n} P(g_i \mid g_{i+1},...,g_n,L) \end{aligned}$$

Da die Wahrscheinlichkeiten durch auszählen von vorkommenden Sequenzen berechnet wird, und die Anzahl dieser Sequenzen mit der Länge exponentiell steigt, macht es keinen Sinn die ganzen Worte mit zugehörender Phonemsequenz zu zählen, also zählt man besser kleinere Teile

von Worten. Nun, welche Sequenzlänge wäre geeignet um nicht zu viele Möglichkeiten zu erhalten, aber trotzdem möglichst wenig Information zu verlieren. Wobei natürlich lange nicht alle Möglichkeiten die existieren auch tatsächlich vorkommen, wenn man zum Beispiel Sequenzen der Länge drei betrachtet, kommt wohl nie eine Sequenz von Graphemen vor, die drei mal hintereinander das selbe Graphem enthält.
Bei einer Zahl von 39 benutzten Phonemen und einer Länge N entspricht 39^N der Anzahl Möglichkeiten. Schon bei einer Länge von N = 3 kommen hier also schon fast 60`000 Sequenzen in Frage, und wenn man die Graphemsequenzen betrachtet, sind das bei etwa 80 verschiedenen Graphemen über 500`000 Möglichkeiten. Wenn man davon ausgeht, dass nicht alle diese möglichen Kombinationen Sinn machen, ist 3 für N eine geeignete Wahl, da N=4 bestimmt zu gross wäre mit über 2Mio. Möglichkeiten für die Phoneme und über 40Mio. für die Grapheme. Das ergibt dann:

$$P(G \mid L) \approx \prod_{i=1}^{n} P(g_i \mid g_{i+1}, g_{i+2}, l_i, l_{i+1}, l_{2+i},)$$

Da sprachlich die späteren Grapheme vielleicht eher von den frühen abhängen, kann man das ganze ohne einen prinzipiell grösseren Fehler zu machen auch mit den vorhergehenden Abbildungen schreiben.

$$P(G \mid L) \approx \prod_{i=1}^{n} P(g_i \mid g_{i-1}, g_{i-2}, l_i, l_{i-1}, l_{i-2},)$$

Hier habe ich die länge der Phonem- und der Graphemsequenzen gleich lang und übereinander gewählt, es wäre aber durchaus möglich verschiedene Längen zu wählen. Es ist hier nicht klar, ob die Lautsequenz wirklich mit der Graphemsequenz übereinstimmen soll oder nicht, ich habe sie so gewählt, da mir das am vernünftigsten schien um den entstehenden Fehler klein zu halten.
Wenn man nun die Formel der bedingten Wahrscheinlichkeiten anwendet, ergibt sich für die gesuchte Wahrscheinlichkeit

$$P(G \mid L) = \frac{P(L \mid G)P(G)}{P(L)}$$

wobei in unserer Anwendung *P(L)* jeweils konstant ist und daher nur eine Normierung darstellt, die für das finden des jeweils grössten Werts keinen Einfluss haben sollte und daher wegfallen kann. *P(L|G)* kann analog wie *P(G|L)* weiter oben berechnet werden. Und *P(G)* wurde bereits weiter oben hergeleitet.

$$P(L \mid G) \approx \prod_{i=1}^{n} P(l_i \mid l_{i-1}, l_{i-2}, g_i, g_{i-1}, g_{i-2},)$$

Wie man die jeweiligen Wahrscheinlichkeiten Ausrechnet ist relativ einfach, die Auftritte der verschiedenen Ereignisse müssen aufsummiert und dividiert werden. Zum Beispiel ist

$$P(g_1 \mid g_2, g_3) = \frac{C(g_1, g_2, g_3)}{C(g_2, g_3)}$$

Wobei $C(g_1,g_2,g_3)$ die Anzahl der vorkommenden Sequenzen $\{g_1,g_2,g_3\}$ und $C(g_2,g_3)$ die Anzahl der vorkammenden Sequenzen $\{g_2,g_3\}$ ist.

3.2 Das Verfahren

Um die jeweilige Entscheidung, welche der Möglichen Abbildungen zu wählen ist, treffen zu können, müssen die nötigen Informationen erst aus dem Trainingsset gewonnen werden. Die Frage war nun: 1. Welche Daten brauche ich überhaupt? und 2. Wie kann ich diese Daten gewinnen?
Zum ersten Punkt, welche Informationen denn gewonnen werden sollten, war aus der obenstehenden Gleichung für die bedingten Wahrscheinlichkeiten und der Frage welche Wahrscheinlichkeiten gebraucht wurden zu klären. Für die Wahrscheinlichkeiten
$P(g_i, |g_{i+1},g_{i+2},l_i,l_{i+1},l_{i+2})$ waren demzufolge die Sequenzpaare $\{g_i, ,g_{i+1},g_{i+2},l_i,l_{i+1},l_{i+2}\}$ und $\{g_{i+1},g_{i+2},l_i,l_{i+1},l_{i+2}\}$ zu zählen waren. Weil ich zudem noch ein Schätzverfahren, das weiter unten erklärt wird, anwedete um aus den Trainingsdaten nicht bekannte Sequenzen schätzen zu können, musste noch weitere für dieses Verfahren wichtige Sequenzen gezählt werden. Dies waren das zu den Trigramm gehörende Bigramm und Unigramm, das bedeutet ich musste $C(g_i,g_{i+1},g_{i+2},l_i,l_{i+1},l_{i+2})$, $C(g_{i+1},g_{i+2},l_i,l_{i+1},l_{i+2})$, $C(g_i,g_{i+1},l_i,l_{i+1})$, $C(g_{i+1}l_i,l_{i+1})$ und $C(g_i,l_i)$, $C(l_i)$ berechnen.
Eine zweite Variante war $P(L|G)$ und $P(G)$ zu schätzen und dann daraus $P`(G|L)$ zu schätzen. Wobei sich diese Schätzung von $P(G|L)$ lediglich in einem Faktor $P(L)$ unterscheidet.
Für das Berechnen von $P(L|G)$ waren dann die analogen Werte zu $P(G|L)$ zu berechnen und um $P(G)$ zu berechnen wurden $C(g_i,g_{i+1},g_{i+2})$, $C(g_{i+1},g_{i+2})$,$C(g_i,g_{i+1})$,$C(g_{i+1})$, $C(g_i)$ und *C(alle Grapheme)* benötigt. Zu beginn war ich mir nicht sicher, welche der beiden Varianten $P(G|L)$ direkt zu berechnen oder $P'(G|L)$ über $P(L|G)$ und $P(G)$ die besseren Ergebnisse ergeben würden. $P(L|G)$ war die direkte Methode, würde und deshalb weniger Rechenaufwand sowie nur eine mögliche Fehlerquelle enthalten. Hingegen wäre es bei der indirekten Methode möglich, dass $P(G)$ aufgrund der Komplexität von $P(L|G)$ und $P(G|L)$ weniger Fehler beinhalten würde als die bedingten Wahrscheinlichkeiten. Deshalb implementierte ich zu Beginn beide (ein paar wenige Zeilen Unterschied) und entschied mich dann, für die zweite Methode $P`(G|L)$ zu benutzen, da die Ergebnisse bei ersten Versuchen leicht besser abschnitt.

Das gewinnen der gesuchten Werte war prinzipiell einfach, da ich alle Sequenzen die im Trainingsset vorkamen nur Aufaddieren musste. Es gab allerdings ein paar Fragen die ich mir bezüglich des Summierens stellen musste und zwar war das Problem was an den Wortanfangen beziehungsweise Wortenden geschehen sollte. Ich löste das Problem derart, dass ich und nach dem Wort jeweils Sonderzeichen einfügte. Es sollte zwar keinen unterschied machen, aber ich wählte für den Wortanfang zweimal „*“ und für das Wortende zweimal „%“ um den Unterschied zwischen Anfang und Ende zu markieren.

3.3 Kneser-Ney Smoothing

Weil ich nicht davon ausgehen konnte, im Trainingsset bereits alle überhaupt vorkommenden Trigramme gefunden zu haben, musste ich, um die Wahrscheinlichkeit aller auftretenden

Trigramme abschätzen zu können, ein Verfahren benutzen, welches es ermöglicht die Wahrscheinlichkeit nicht bekannter Trigramme zu schätzen. Hierfür verwendete ich das Verfahren von Kneser und Ney. Die Mathematik, die hinter dieser Methode steckt ist kompliziert und ich will hier nicht auf die Details eingehen aber trotzdem einen kurzen Einblick in die Funktionsweise des Verfahrens geben, für ein besseres Verständnis weise ich auf [3] von Goodman hin, auf den Seiten 64-66 ist dort ein geeignetes Beispiel der Implementation in Pearl gegeben.
Das Verfahren ist erklärt für nicht bedingte Wahrscheinlichkeiten, aber es lässt sich auch für bedingte Wahrscheinlichkeiten benutzen, ea in beiden Fällen die Berechnung der Wahrscheinlichkeiten auf die selbe Weise ausgeführt wird, durch die Division zweier Werte die durch Aufsummieren entstanden sind.
Im Grunde genommen geht es um eine Schätzung von *P(z)*, wobei $z = \{z_0,z_1,z_2\}$ in unserem Falle eine Graphemsequenz der Länge drei ist. Das Problem ist, dass z nicht unbedingt in den Trainingsdaten vorkommt. Nun wird in diesem Verfahren erst einmal die Wahrscheinlichkeit von z_0 berechnet, von der man ausgehen darf, dass diese tatsächlich in den Trainingsdaten vorkam. Dann wird die Wahrscheinlichkeit der Sequenz der Länge zwei, also des Bigramms, mit Kneser-Ney berechnet, sofern $C(z_1)$ existiert, ansonsten wird die Wahrscheinlichkeit für die Unigramme verwendet. Der Schritt vom Bigramm zum Trigramm geschieht dann Analog zum Schritt Unigramm zu Bigramm.
Wie sieht so ein Schritt aber aus, man berechnet zuerst die Wahrscheinlichkeit des Bigramms ohne Kneser-Ney. Dann zieht man dieser Wahrscheinlichkeit einen Wert $\lambda / C(z_1)$ ab, wobei λ ein frei wählbarer Parameter ist der der grösser Null saein sollte. Allerdings wird dieser Wert nicht abgezogen, wenn $C(z_0,z_1)$ gleich Null ist, da es hier nichts abzuziehen gäbe. Dann wird dem Ergebniss aber wieder ein Wert hinzugefügt, der

$$\lambda * [\text{der Anzahl verschiedener Bigramme } \{z0,z1| z1 \}] * P_{KN}(\text{Unigramm}) / C(z1)$$

entspricht und die Kneser-Ney-Wahrscheinlichkeit für das Bigramm ist berechnet. Das Verfahren macht also nichts anderes als den vorkommenden Sequenzen ein Wenig Wahrscheinlichkeit zu nehmen, um sie den nicht auftretenden Sequenzen zu geben.

3.4 Implementation

Bei der Implementation stellte sich relativ schnell das Problem des
Speicherns der Trigramme und der dazugehörenden Counts, immerhin mussten die relevanten Daten aus über 300`000 Worten erfasst werden.
Für das Speichern der Bigramme und Trigramme verwendete ich deshalb einen Binärbaum, in dem man neue Trigramme einfügen konnte, so dass der benötigte Speicher abhängig von den tatsächlich auftretenden und nicht von den überhaupt möglichen Trigrammen war.
Diese Struktur war in zwei Ebenen vorhanden, wobei in der oberen, die Graphem-Trigramme und in einer tieferen Ebene dann die dazugehörenden Phonem-Trigramme gespeichert wurden.
Wobei in den Graphemtrigrammen jeweils ein Bigramm und mehrere zusätzliche Unigramme gespeichert wurden, mit denen dann das Trigramm zusammengesetzt werden konnte. Das bedeutet, in den Graphemtrigrammen waren $C(g_i,g_{i+1},g_{i+2})$ zu finden und in den Phonemtrigrammen $C(l_i|l_{i+1},l_{i+2},g_i,g_{i+1},g_{i+2})$ und $C(l_{i+1},l_{i+2},g_i,g_{i+1},g_{i+2},)$.

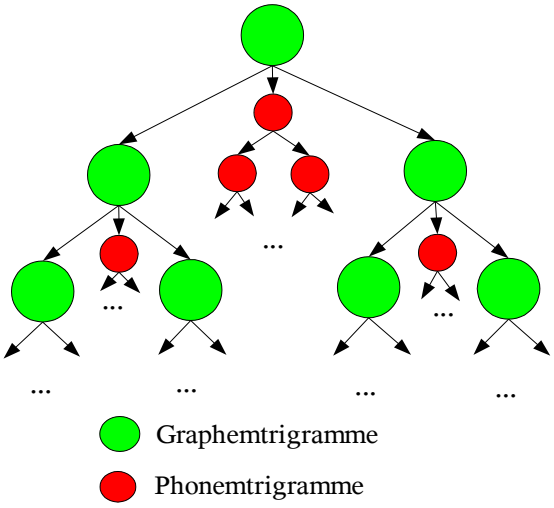


Abb. 6 Speicherstruktur Phonem-Graphem-Trigramme

Für die Bigramme benutzte ich die gleichen Methoden, da vom Prinzip her bei den Bigrammen alles gleich funktionierte bis auf die Länge der verwendeten Sequenzen, konnte ich hierfür die gleichen Klassen benutzen wie für die Trigramme und einfach eine andere Sequenzlänge benutzen. Für die Unigramme benutzte ich dann die Klasse die ich für das speichern der gültigen Abbildungen schon beim Berechnen der Editierdistanzen verwendet hatte, indem ich die Klasse so änderte, dass sie nun die Abbildungen auch zählen konnte.

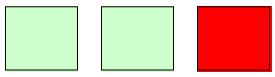
Der Entscheidungsalgorithmus hat mich viel Zeit und Nerven gekostet, mein Ziel war es die Wahrscheinlichkeit jeweils über das ganzeWort zu maximieren. Allerdings stellte sich hier das Problem, dass alle Möglichkeiten auszuprobieren sehr zeitintensiv war und bei einem Grossen Testset und langen Worten eine Testreihe wohl eine rechte Weile gedauert hätte, bis alle Ergebnisse berechnet wurden. Ich versuchte deshalb über einen dynamischen Algorithmus möglichst viele der falschen und nicht relevanten Graphemsequenzen auszuschliessen. Nun, ich hatte zuerst keine Ahnung wie ich hier ansetzen sollte, also habe ich zuerst einfach angefangen und einfache Entscheidungen betrachtet in denen ich für jede Entscheidung der Abbildung ein Trigramm betrachtete. Dann bin ich dazu übergegangen drei Zeitschritte zu betrachten und für diese Situationen die beste Entscheidung zu fällen. Leider hatte ich zu wenig Zeit dann weiter zu gehen und einen Algorithmus zu finden und zu schreiben der immer das beste Wort findet. Ein Ansatz hier wäre aber sicher ein Trellis-Algorithmus, da hier die Anzahl der zu betrachtenden vorherigen Abbildungen jeweils von der Anzahl der Möglichen Abbildungen zu jenem Zeitpunkt abhängt.

4. Auswertungen

4.1 Messungen

Um zu testen wie gut das Verfahren das ich implementiert hatte funktioniert, habe ich mich beim Messen der Ergebnisse auf drei Grössen konzentriert: Die im Verhältnis richtig gefundenen Worte, die Genauigkeit der gefundenen Worte und die in Prozent richtig entschiedenenen Abbildungen.
Das Verhältnis richtig gefundener Worte berechnete sich natürlich durch die Division von richtigen Worten durch alle Worte.
Die Genauigkeit der Worte wurde über ein Verfahren ähnlich dem für die Editierdistanz verwendet, man zählte über alle Buchstaben, wobei hier Identitäten, Einfügungen, Löschungen und Substitutionen gezählt wurden. Die Genauigkeit berechnete sich dann als Verhältnis von identischen Buchstaben zu allen gezählten Buchstaben.
Die Prozent richtig entschiedener Abbildungen berechnete sich über das Verhältnis von richtigen Abbildungen zu allen gemachten Abbildungen. Diese Grösse hatte ich zusätzlich zur Genauigkeit gewählt, um zu sehen, ob auf der Ebene der Abbildungen oder der Ebene der Buchstaben, die sich ja sehr ähnlich sind, mehr Fehler zu entdecken sind.
Das Testset das ich verwendete enthielt etwa 10'000 Worte und etwa 100'000 Abbildungen die zu entscheiden waren.
Um ein Gefühl für die Zusammenhänge zu bekommen hatte ich verschiedene Möglichkeiten für die Berechnung der richtigen Abbildungen untersucht. Zu Beginn habe ich für jede Abbildung die zu entscheiden war jeweils ein Trigramm angeschaut und diese auf zwei verschiedene Arten, einmal vorausschauend und einmal zurückschauend.
Die Ergebnisse für diese beiden Varianten waren:

Rückwärtsschauend:

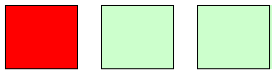


Prozent richtige Worte: 0,13 %

Genauigkeit der Worte: 57,43 %

Prozent richtige Abbildungen: 42,57 %

Vorwärtsschauend:



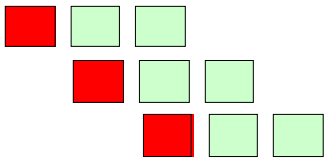
Prozent richtige Worte: 0,18 %

Genauigkeit der Worte: 65,21 %

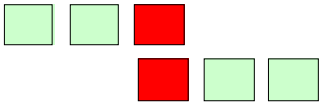
Prozent richtige Abbildungen: 57,48 %

Hier ist klar zu sehen, dass man mit dieser Methode eindeutig zu wenig Informationen gibt um einigermassen befriedigende Ergebnisse zu erzielen. Die klare Differenz zwischen der Genauigkeit der Worte und der Anzahl richtiger Worte ist dadurch zu erklären, dass ein Wort,

sobald eine Abbildung im Wort falsch gewählt wird ebenfalls falsch ist. Dadurch wird die Wahrscheinlichkeit eines Wortes richtig erkannt zu werden exponentiell von der Länge des Wortes und der Wahrscheinlichkeit die richtige Abbildung zu wählen abhängig
Die Differenz der Beiden Verfahren kann darin liegen, dass bei dem vorausschauenden Algorithmus die nachfolgenden Grapheme noch nicht bestimmt sind, beim rückschauenden allerding gesetzt sind.
Um einen grösseren Kontext betrachten zu können und berücksichtigt werden musste, dass schon gefällte Entscheidungen auf das Resultat einen Einfluss haben, habe ich dann jeweils zwei weitere Trigramme betrachtet um eine Abbildung zu entscheiden, um somit dann insgesamt einen Kontext der Länge fünf zu betrachten, ausserdem habe ich mir überlegt, ob es unter Umständen hier genügen würde einmal nach vorne und einmal zurück zu schauen:



Prozent richtige Worte: 20,47 %
Genauigkeit der Worte: 85,34 %
Prozent richtige Abbildungen: 85,58 %



Prozent richtige Worte: 0,35 %
Genauigkeit der Worte: 62,89 %
Prozent richtige Abbildungen: 56,45 %

Wenn man nun die richtigen Abbildungen mit der Genauigkeit der Worte vergleicht, sieht man, dass aufgrund des Ansatzes die Genauigkeit der Worte in schlechteren Methoden zum Teil markant grösser ist als die richtigen Abbildungen, hingegen bei besseren Methoden die beiden Werte ungefähr gleich gross sind. Überraschend ist für mich, dass die Methode, die gleichzeitig vor und zurück schaut im Vergleich eher schlechter ist als bei der nur Vorwärtssehenden aber besser als bei der nur Rückwärtssehenden Methode ist. Das kann man aber dadurch erklären, dass diese Methode eigentlich das Zusammenpappen zweier verschiedener Entscheidungen ist, die vom ursprünglichen Konzept her so nicht gedacht war und es wäre bestimmt genauer die Wahrscheinlichkeit direkt über eine Sequenz der Länge fünf zu berechnen.
Während den Messungen habe ich immer wieder Fehler im Code entdeckt, wie zum Beisiel Divisionen durch Null, wobei allerdings nach der Behebung dieser Fehler zum Teil die Messungen schlechtere Ergebnisse abgaben als vor der Behebung, wie das zustande kam, kann ich nicht erklären, aber die Genauigkeit der Worte erreichte nie einen Wert der die Grenze von 85% überschritt.

4.2 Was ich gelernt habe

Während der Semesterarbeit konnte ich viele wichtige Erfahrungen sammeln. Zum einen gab sie mir einen Einblick in die Probleme und Lösungsansätze die in der Sprachverarbeiung vorkommen. Auch wenn ich beim durchlesen des Skriptes „Sprachverarbeitung I“ nur wenig vorfand, was ich für meine tatsächliche Arbeit verwenden konnte, war es doch eine interessante Lektüre.
Ein zweites war das Auffrischen meiner Programmierkenntnisse in Java, da ich den gesamten Code in Java geschrieben habe und seit dem ersten Vordiplom kein Java mehr verwendet habe.
Ich lernte einiges über Arbeitsplanung, und dass ich mir meistens für eine Aufgabe mehr Zeit einteilen sollte, als mein Gefühl mir sagt.
Eine weitere sehr wichtige Erkenntnis war, niemals ein funktionierendes Stück Code zu erweitern, ohne nicht irgendwo eine Sicherheitskopie des Codes zu haben, insbesondere bei komplizierten Änderungen die in mehr als einer Klasse auswirkungen haben.
Was das Messen von Genauigkeiten eines Verfahrens angeht habe ich auch einiges lernen können, vor allem, dass man sich zu Beginn der Messung gründlich überlegt was man wissen will und was nicht, zum Messresultat schreibt auf welche Weise es zustande kam und wann die Messung stattfand, wenn man die Messung nicht wiederholen möchte. Am Besten speichert man die Resultate in ein Logfile.

4.3 Zeitplan

Ich hatte mir für die Semesterarbeit in der dritten Woche einen Zeitplan aufgestellt, der zwar sehr grob gehalten war, aber bis auf ein paar wenige Termine konnte ich alle einhalten, da ich mit der Planung eher vorsichtig war um am Ende nicht in einen übermässigen Stress zu kommen. Hier ein Überblick, wieviel Zeit ich etwa für welche Aufgaben verwendet habe.

1. Woche 20. – 24. Okt.	Material lesen und versuchen zu verstehen
2. Woche 27. – 31. Okt.	Material lesen
3. Woche 3. – 7. Nov.	Methode zur Berechnung der Editierdistanzen
4. Woche 10. – 14. Nov.	Methode zur Berechnung der Editierdistanzen
5. Woche 17. – 21. Nov.	Rückwärtsrechnen um Abbildungen zu finden und Symbole erzeugen und speichern können
6. Woche 24. – 28. Nov.	Verfahren (ohne ED) um Abbildungen zu finden und Entscheid die Abbildungen selber vorzugeben
7. Woche 1. – 5. Dez.	Fehlersuche und Abschluss des ersten Aufgabenteils die Abbildungen zu finden
8. Woche 8. – 12. Dez.	Modell für statistisches Lernverfahren, Implementation
9. Woche 15. – 19. Dez	Implementation

--- Weihnachtsferien ---	Implementation ohne Smoothing für erste Versuche bereit
10. Woche 5. – 9. Jan.	Kneser-Ney Smoothing implementieren, Verbesserungen überlegen
11. Woche 12. - 16. Jan.	Fehler beheben
12. Woche 19. – 23. Jan.	Auswertung der Methoden, Fehler suchen
13. Woche 26. – 30. Jan.	Bericht beginnen zusammenzufassen, Präsentation
14. Woche 2. – 6. Jan.	Letzte Messungen, Bericht und Arbeit abschliessen

5. Ausblick

In diesem Kapitel möchte ich kurz Gedanken und Konzepte vorstellen, die ich mir während der Semesterarbeit überlegt hatte, aber leider keine Zeit sie zu überprüfen oder zu implementieren hatte. Zum einen gäbe noch einiges, was man bezüglich meiner implementierten Methode tun könnte, zum einen wäre es sicher interessant zu erfahren ob das Verfahren bei anderen Konstellationen des Kontextes besser funktionieren würde oder ob das Verfahren bei einem grösseren Trainingsset besser funktionieren würde, sprich ob die Grenzen des Verfahrens, so wie ich es angewendet habe schon erreicht wurden oder nicht.
Ein weiteres wäre dann natürlich auch die Abbildungsentscheidung durch einen dynamischen algorithmus zu finden.
Eine andere Frage wäre, wie man auf effektive Weise einen grösseren Kontext betrachten kann, da für verschiedene Beispiele ein Kontext der Länge drei zu wenig ist, zum Beispiel kann das Wort „fortgehen“ und das Wort „vortreten“ in den ersten vier Phonemen nicht unterschieden werden, aber trotzdem unterscheiden sie sich im ersten Buchstaben.

Eine weitere Idee von mir ist, nicht Abbildungen von Phonemen auf Grapheme, sondern die Abbildungen von Silben wie: „ver“, „ent“, „ge“ , „se“ oder „hen“ zu betrachten, da diese automatisch einen grösseren Kontext bereitstellen. Allerdings würde es auch sehr schwierig sein, diese Silben zu finden. In meiner Arbeit, waren die Phoneme bereits bekannt, und ich konnte die dazugehörenden Grapheme bereits suchen. Bei den Silben aber, müsste man zuerst die phonetischen Silben finden und von den Silben dann die möglichen Abbildungen finden. Der einzige Ansatz, der mir bisher für die Suche nach diesen Silben eingefallen ist, ist die Tatsache, dass jeweils nur ein Vokal in einer Silbe vorkommt was das ganze nicht wesentlich einfacher macht.

Ein weiterer Versuch könnte das Problem mit Hilfe von neuronalen Netzen zu lösen versuchen, ich selber habe kaum Erfahrungen mit dem Entwurf und den Möglichkeiten dieser Hilfsmittel, aber ich könnte mir vorstellen, dass durch die Kombination verschiedener neuronaler Netze das Problem relativ einfach über eine Klassifikation gelöst werden kann, wobei die Grapheme in den Phonem-Graphem-Abbildungen die jeweiligen Klassen sind, die zu bestimmen sind, wobei für jedes Phonem ein eigenes Klassierungsproblem zu lösen ist.
Man könnte ein Neuronales Netz erzeugen, das eine bestimmte Anzahl von Phonemen rund um ein bestimmtes Phonem oder eine Phonemgruppe und eventuell zusätzlich noch die bereits entschiedenen Grapheme als Eingang hat, und die Pseudowahrscheinlichkeiten für die verschiedenen möglichen Graphemklassen des Phonems oder der Phonemgruppe ausgibt. Natürlich müsste für jedes Phonem ein eigenes Netz trainiert werden, da die Phoneme auf verschiedene Grapheme abgebildet werden können. Ausserdem müsste dann für eine Phonemsequenz der Länge N auch mindestens N mal ein neuronales Netz für das Berechnen der jeweiligen Worte benutzt werden, wobei man sich dann die Frage stellen sollte, ob man das alles nicht doch in ein einziges Neuronales Netz packen könnte, und das so, dass pro Wort nur eine Abfrage des Netzes nötig ist?

6. Quellen

[1] B. Pfister und H.-P. Hutter. Sprachverarbeitung I. Vorlesungsskript für das Wintersemester 2001/2002, Departement ITET, ETH Zürich, 2001.

[2] F.Yvon. Self-learning techniques for grapheme-to-phoneme conersion, 1994.

[3] J. T. Goodman. A Bit of Progress in Language Modeling, 2001.