Vektorquantisierung

Carsten Isert

21. Januar 1997

1 Einleitung

- Bei der Methode der Vektorquantisierung handelt es sich ebenso wie bei der skalaren Quantisierung um ein verlustbehaftetes Kompressionsverfahren.
- Im allgemeinen benötigt das Codieren bei der Vektorquantisierung viel Zeit, das Decodieren hingegen nur sehr wenig.
- Für verschiedene Anwendungen gibt es viele unterschiedliche Verfahren zur Vektorquantisierung (und für die gleiche Anwendung auch).
- Zu jedem Eingabevektor wird der nächstgelegene Codevektor gesucht und dessen Index aus dem Codebuch übertragen. Der Empfänger kann dann aus dem Index den Ausgabevektor wieder rekonstruieren.
- Das große Problem bei der Vektorquantisierung ist die Erstellung (und anschließende Übertragung) des Codebuchs.

2 Grundlagen der Vektorquantisierung

- L : Dimension der Vektoren, K : Größe des Codebuchs.
  $R := \frac{\log_2 K}{L}$: Anzahl der Bits bei der Übertragung jeder Vektorkomponente.
  B : Anzahl der Bits zur Speicherung jeder Komponente der Codevektoren.
- Man verwendet normalerweise den quadratischen euklidischen Abstand:
  $$d(X,Y) = \|X - Y\|^2 = \sum_{i=1}^{L} (x_i - y_i)^2,$$
  wobei $X = (x_1, x_2, \ldots, x_L)$ und $Y = (y_1, y_2, \ldots, y_L)$.
  Allerdings kann man auch andere Abstände verwenden, so wird z.B. bei der Spracheverarbeitung ein gewichteter quadratischer Abstand benutzt.
- Der Eingabevektor X ist genau dann dem Codebuch-Vektor Y_i am nächsten, d.h. liegt in der Voronoi-Region V_i, wenn gilt:
  $$\|X - Y_i\|^2 \leq \|X - Y_j\|^2 \quad \forall\, Y_j$$
  Bei gleicher Entfernung wird die Region mit dem niedrigeren Index gewählt.

- Die zu übertragenden Daten können aufgrund der Struktur des Codebuchs nicht gleichmäßig über die Codevektoren verteilt sein, daher kann man noch einmal einen Huffman Algorithmus anwenden.

3 Das Codebuch

- Bei der Erstellung des Codebuchs handelt es sich um ein typisches Optimierungsproblem der Informatik: In einer Fitneßlandschaft wird nach dem globalen Minimum/Maximum gesucht.
- Dieses Problem läßt sich nicht effizient lösen. Deshalb verwendet man Algorithmen, die ein möglichst gutes lokales Minimum finden können und nur wenig Zeit benötigen.
- Dazu gibt es viele Ansätze: Gradientenabstieg, Simulated Annealing, Deterministic Annealing, Genetische Algorithmen, ...
- Der Linde-Buzo-Gray Algorithmus (Generalisierter Lloyd Algorithmus) ist ein Gradientenabstiegsverfahren, daß folgendermaßen funktioniert:

  1. Starte mit den Codevektoren $\{Y_i^{(0)}\}_{i=1}^M$. Sei $k = 1, D^{(0)} = 0$ und wähle eine Abbruchgrenze $\epsilon > 0$.
  2. Bestimme die Voronoi-Regionen
  $$V_i^{(k)} = \{X : \|X - Y_i\|^2 \leq \|X - Y_j\|^2 \quad \forall\, j \neq i\} \quad j = 1, 2, \ldots, M$$
  3. Berechne den mittleren quadratischen Fehler:
  $$D^{(k)} = \sum_{i=1}^{M} \int_{V_i^{(k)}} \|X - Y_i^{(k)}\|^2 f_X(X)\, dX$$
  4. Falls $\frac{D^{(k-1)} - D^{(k)}}{D^{(k-1)}} < \epsilon$ dann höre auf.
  5. Berechne $\{Y_i^{(k)}\}_{i=1}^M$, so daß sie die Zentren von $\{V_i^{(k-1)}\}$ sind. $k = k + 1$. Gehe zu Schritt 2.

- Für diesen Algorithmus muß man jedoch die Verteilung kennen und außerdem einige ziemlich üble Integrale berechnen.
- In der Praxis verwendet man besser ein Verfahren, bei dem man einige Trainingsvektoren zur Verfügung hat:

  1. Starte mit den Codevektoren $\{Y_i^{(0)}\}_{i=1}^M$ und Trainingsvektoren $\{X_n\}_{n=1}^M$. Sei $k = 1, D^{(0)} = 0$ und wähle eine Abbruchgrenze $\epsilon > 0$.
  2. Bestimme die Voronoi-Regionen
  $$V_i^{(k)} = \{X_n : \|X_n, Y_i\|^2 \leq \|X_n, Y_j\|^2 \quad \forall\, j \neq i\} \quad j = 1, 2, \ldots, M$$
  Falls eine Region leer ist, dann sollte das entsprechende Y_i umgesetzt werden.

3. Berechne den mittleren quadratischen Fehler:

$$D^{(k)} = \sum_{i=1}^{M} \sum_{X \in V_i^{(k)}} \|X - Y_i^{(k)}\|^2$$

oder falls dieser unabhängig von der Anzahl der Vektoren sein soll:

$$D^{(k)} = \frac{1}{M} \sum_{i=1}^{M} \sum_{X \in V_i^{(k)}} \|X - Y_i^{(k)}\|^2$$

4. Falls $\frac{D^{(k-1)} - D^{(k)}}{D^{(k-1)}} < \epsilon$ dann höre auf.
5. Berechne $\{Y_i^{(k)}\}_{i=1}^{M}$, so daß sie die Zentren von $\{V_i^{(k-1)}\}$ sind. $k = k + 1$. Gehe zu Schritt 2.

- Beobachtung: Bei verschiedenen Startwerten kommen unterschiedliche Ergebnisse heraus. Diese entsprechen den verschiedenen lokalen Minima.

- Das Zentrum (der Schwerpunkt beim euklidischen Abstand) berechnet sich nach dem arithmetischen Mittel:

$$cent(V_j) = \frac{1}{|V_j|} \sum_{i=1}^{|V_j|} x_i$$

  Dabei ist $|V_j|$ die Mächtigkeit der Menge V_j.

- Problem der gleichen Entfernung

- Für eine endliche Menge an Trainingsvektoren konvergiert (terminiert) der LBG Algorithmus offensichtlich

- Um dieses Problem in den Griff zu bekommen verwendet man verschiedene Verfahren:

  - *Rechteckiger Ansatz:* Man verteilt die Codevektoren am Anfang gleichmäßig über das Gebiet.
  - *Splitting:* Man beginnt mit nur einem Codevektor und führt für ihn den LBG-Algorithmus aus. Danach erzeugt man einen zweiten, geringfügig in der Lage veränderten Codevektor und führt den Algorithmus wieder aus. Man verdoppelt solange die Codevektoren, bis man ein Codebuch der gewünschten Größe hat.
  - *Zufällige Auswahl der Startvektoren aus den Trainingsvektoren.* Dies kann man öfters machen und diejenigen Vektoren, die den geringsten Fehler aufweisen als Startvektoren für den Algorithmus verwenden.
  - *PNN (pairwise nearest neighbor):* Man beginnt mit sovielen Codevektoren, wie es Trainingsvektoren gibt und fügt dann die jeweils nächsten Codevektoren zusammen und ersetzt sie durch das Mittel.

- In der Praxis ist zwischen den einzelnen Verfahren kein großer Unterschied.

- *Problem der leeren Zellen:* Entweder man verändert Codevektoren, mit leeren Voronoi-Regionen nicht und läuft somit Gefahr, Codvektoren zu haben, die keinen Ausgabevektor repräsentieren, oder man verschiebt diesen leeren Codevektor einfach in die dichteste Region. Dies kann nun ein zufällig geschehen, oder man verwendet hierzu wieder eine dem Splitting ähnliche Technik.
- Verwendung zur Bildkompression
- Durch eine Verallgemeinerung des LBG-Algorithmus kann man zu den in der Künstlichen Intelligenz verwendeten Kohonen-Netzen gelangen [RMS92].
- Problem des Codebuchs:
  - Wird das Codebuch mit den Daten übertragen, dann müssen zusätzlich noch LBK Bits übertragen werden. Dies kann insbesondere bei kleineren Datenmengen zu einem unverhältnismäßig großen Overhead führen und die ganze Kompression ad absurdum führen.
  - Man kann stattdessen das Codebuch einer ähnlichen Quelle verwenden. Dabei entstehen mit großer Sicherheit Verluste in der Qualität, aber auch nur solange man den andernfalls zu übertragenden Overhead nicht mitbeachtet.
  - Die Größe des Codebuchs steigt exponentiell mit dem Produkt der Anzahl der Bits pro Komponente R und der Dimension L des Vektors. Dies führt zu prinzipiellen Problemen. Deswegen arbeiten die meisten Vektorquantisierer mit niedrigen Bitraten.

4 Baumförmige Vektorquantisierung

- Beim LBG-Algorithmus ist es in Schritt 3 immer notwendig, die Entfernung eines Eingabevektors zu allen anderen Codebuchvektoren zu berechnen.
- Dies führt zu einem „recht hohen" Rechenaufwand $O(N)$. Dies kann man verringern, indem man eine gewisse Struktur ins Codebuch einbringt.
- *Verfahren:* Beginne mit zwei Codevektoren und teile die Trainingsvektoren in zwei Gruppen, so daß die in *Gruppe0* näher an dem einen, und in *Gruppe1* näher an dem anderen Codevektor sind.
- Nun bilde in den beiden Gruppen wieder jeweils zwei Teilgruppen *Gruppe00, Gruppe01, Gruppe10* und *Gruppe11*. Teile die Trainingsvektoren entsprechend der Näherelation in die jeweiligen Gruppen. So wird fortgefahren, bis die gewünschte Anzahl an Gruppen erreicht ist.
- Nachteile: Der Fehler kann gegenüber dem LBG-Algorithmus größer werden.
- Man nimmt ein Anwachsen der Speicherplatzkomplexität in Kauf, da ja die Vektoren der inneren Baumelemente auch gespeichert werden müssen. Dies ist aber nur der Fall, wenn man auf die inneren Knoten wert legt. Übertragen werden müssen sie nicht werden, d.h. der Decoder braucht sie nicht. Ansonsten benötigt man zusätzlich $\frac{m^d-1}{m-1}$ Knoten.

- Vorteile: Bei diesem Verfahren muß man nun nur noch in jeder Gruppe zwei Vergleiche anstellen und kommt insgesamt mit $O(\log N)$ Vergleichen aus.
- Wenn von einer Nachricht nur der Anfang übertragen wird, dann ist eine teilweise Rekonstruktion möglich.
- Dieses Prinzip kann man sich zunutze machen und einige Zweige absichtlich abschneiden (Pruning), so daß weniger Speicher für den Baum benötigt wird, aber die Qualtät sich nur geringfügig verringert, so daß am Ende eine bessere Kompressionsrate resultiert.

5 Strukturierte Vektorquantisierung

- Durch die Gleichung $\sum_{i=1}^{L} |x_i| = const.$ ist eine Hyperpyramide mit konstanter Wahrscheinlichkeit gegeben.
- Außerhalb einer gewissen Hyperpyramide werden alle Codevektoren weggelassen, da der Gewinn an Speicherplatz den Verlust im Fehler aufwiegt (Grenz-Gewinn). Dieser Gewinn wird mit steigender Dimension L immer größer.
- *Pyramiden VQ:* Nach Shannons AEP (asymptotic equipartition property) gilt für entsprechend große L und kleine ϵ:

  $$\left| \frac{\log f_X(X)}{L} + h(X) \right| < \epsilon$$

  Damit liegen fast alle Eingabevektoren innerhalb der Region mit konstanter Wahrscheinlichkeit

  $$-\frac{\log f_X(X)}{L} = h(X)$$

  Nun verteilt man die Codevektoren gleichmäßig in dieser Region.

  Bei der Laplace-Verteilung kann zu dem Wert dieser Pyramide folgendermaßen kommen:

  Verfahren: Berechne $r = \sum_{i=1}^{L} |x_i|$ und übertrage den Wert dieser Hyperpyramide an den Empfänger. Die Eingabevektoren werden normalisiert und anschließend mit dieser einzigen Hyperpyramide quantisiert.
- *Gitter VQ:* Berechnet man den mittleren quadratischen Fehler für Quadrat und Kreis mit gleicher Fläche, dann erhält man folgende Ergebnisse:

  $$\int_Q \|X\|^2 = 0,1\overline{6} \qquad \int_K \|X\|^2 = 1,59$$

  Mit einem Kreis kann man jedoch keine Fläche ohne Überlappung bzw. Lücken ausfüllen, so daß man mit Näherungen auskommen muß.

  In höheren Dimensionen kann man sich diese Formen jedoch nicht mehr veranschaulichen. Man definiert daher bestimmte Ausgabepunkte, die in regelmäßigen Abständen genommen ein Gitter für den Quantisierer bilden.

Weitere Methoden

- Variieren die Eingabevektoren in ihrer Größe sehr stark, dann kann man sie zuerst normalisieren und die beiden Komponenten getrennt quantisieren.
- Bei der Bildkompression treten bei unterschiedlichen Helligkeiten sehr verschiedene Codebücher auf. Dies kann man verringern, indem man von jedem Vektor den Mittelwert entfernt und getrennt überträgt. Dies bringt auch Vorteile, wenn man das Codebuch eines anderen Bildes verwendet.
- Einteilung der Eingabevektoren in verschiedene Klassen (z.B. Kanten bei Bildern) und Verwendung verschiedener Quantisierer.
- *Mehrfache VQ:* Zuerst geringe Auflösung, dann im nächsten Schritt Quantisierung des gemachten Fehlers usw.

  Man kann dieses Verfahren auch teilweise früher abbrechen, wenn der Fehler unter einer gewissen Schranke ϵ liegt.
- *Adaptive VQ:* Ändert sich die Charakteristik der Quelle, kann es nötig sein, den Quantisierer anzupassen. Eine Möglichkeit dazu ist, ein sehr großes Codebuch zu verwenden, daß in verschiedene Teile gegliedert ist, und immer nur auf einem dieser Teile zu operieren. Will man kein so großes Codebuch verwenden, so kann man einen der änderung der Quelle angepaßten Codebuchvektor hinzufügen und dafür einen wenig benutzten löschen.

6 Schluß

- Insgesmat kann man sicher das Gebiet der Vektorquantisierung nicht in einer Stunde umfassend oder tiefgreifend darstellen.
- Die Resultate der allgemeinen Vektorquantisierung sind mindestens so gut, wie die von jedem anderen Verfahren, daß auf Vektoren arbeitet.

  Genauer gilt folgendes **Optimalitätstheorem:** Für ein beliebiges Codierungssystem $\mathcal{C}$, daß einen Signalvektor auf eine von L binären Ausgaben abbildet und einen angenäherten Ausgabevektor aus dem binären Wort rekonstruiert, gibt es einen Vektorquantisierer $\mathcal{Q}$ mit Codebuchgröße L, der exakt das gleiche leistet, d.h. daß er für jeden beliebigen Eingabevektor das gleiche Ergebnis produziert, wie das gegebene Codierungssystem $\mathcal{C}$.

  Beweis: Nummeriere die vom Codierungssystem $\mathcal{C}$ produzierte Menge an Binärwörtern mit Indizes $1, 2, \ldots, L$. Für das $i$te Binärwort sei y_i der von $\mathcal{C}$ dekodierte Ausgabevektor. Nun definiere man das Codebuch $\mathcal{B}$ von $\mathcal{Q}$ als die geordnete Menge der y_i. Damit erreicht $\mathcal{Q}$ das gleiche Ergebnis wie $\mathcal{C}$. □
- Besonders zu empfehlen ist das Buch von Gersho und Gray [GG92].
- Einsatzgebiete: z.B. Bild-, Sprachkompression und Spracherkennung Auf dem Sprachgebiet wird jedoch meistens ein gewichteter quadratischer Fehler verwendet, der in etwa der Kennlinie des menschlichen Gehöhrs entspricht.

- Diesen Artikel gibt es im Internet unter der Adresse:
  `http://www.informatik.tu-muenchen.de/~isert/VQ-Ausarbeitung.html`

Literatur

[GG92] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[RMS92] Helge Ritter, Thomas Martinetz, and Klaus Schulten. *Neuronale Netze.* Addison-Wesley, 2. edition, 1992.

[Say96] K. Sayood. *Data Compression.* Morgan Kaufmann, 1996.