Rapid Frontend Prototyping with Deep Learning

Simon Deussen

7. Oktober 2018

Generierung von HTML/CSS Code anhand von pixelbasierten Screenshots und -designs. Aufbauend auf dem Pix2Code Paper [4], wird diese Arbeit eine eigene Implementation mit einem anderen Framework sowie einer ausführlichereren DSL erstellen. Durch automatisches Erstellen von Frontend-Code kann diese Anwendung rapide Entwicklungszyklen realisieren.

Inhaltsverzeichnis

1 Einleitung

Viele client-basierte Anwendungen brauchen ein schönes Frontend. Dieses soll gleichzeitig funktional und übersichtlich sein, sowie die Firma durch gutes Design repräsentieren. Seit Apple & Co ist es unglaublich wichtig gutes Design im Frontend zu haben, da sich die Kunden sonst schnell eine Alternative suchen. Um nun ein großartiges Frontend zu realisieren benötigt man ein enge Kooperation von Designern und Entwicklern, da hier zwei, sich kaum überschneidende, Skillsets gebraucht werden. Im der klassichen Entwicklung sieht diese Zusammenarbeit folgendermaßen aus:
Ein Designer macht einen grafischen Entwurf, dieser wird vom Kunde abgenommen, dann geht er zu dem Entwickler, der nun zu aller erst Markup für den Content und anschließend das Design und die richtige Darstellung nach bauen muss. Für jede grafische Veränderung muss dieser Prozess wieder ausgeführt werden. Für die meisten Entwickler, ist die Markup und CSS erstellung der widrigste Part der ganzen Arbeit, da er recht zeit-aufwendig, repetetiv und langweilig ist. Es gab bisher viele Ansätze diese Arbeit zu automatisieren, zB durch Tools in dem man gleichzeitig Designen und den Markup exportieren kann. Leider sind diese Tools entweder nicht besonders gut die Designs zu erstellen oder darin den Markup zu exportieren.

Eine Abhilfe soll diese Arbeit liefern: Sie ermöglicht, dass der Designer mit seinem bevorzugten Tools das Design baut und der Entwickler mit einem Mausklick das fertige Markup bekommt. So kann sich der Entwickler vollends auf die Realisierung des Verhaltens und der Logik der Anwendung konzentrieren.

2 Ähnliche Arbeiten

Diese Arbeit basiert auf dem Pix2Code Paper von Tony Beltramelli [4] . Er war der erste der Code anhand von visuellen Input generieren kann. Anderen Ansätze wie DeepCoder [3] benötigen komplizierte DSL als Input und schränken so die Benutzbarkeit stark ein. Visuelle Versuche mit Android-GUIs von Nguyen [5] benötigt ebenfalls unpraktische von Experten erstellte Heuristiken. Pix2Code ist so das erste Paper das einen allgemeinen Input hat, und daraus momentan drei verschiedene Targetsprachen hat. Zum einen kann es HTML/CSS Code erstellen, zum anderen aber auch Android- und iOS-Markup. Siehe Orginal Code auf Github [2]

3 Motivation

Computergenierte Programme werden die Zukunft der Software Entwicklung sein und diesen Bereich auch grundsätzlich verändern.

Ich denke das sich die Webtechnologien auch in der Desktop Umgebung durchsetzen. Da Plattform unabhängig und sehr stark optimiert. Sehr einfach zu lernen, weit verbreitet. Zum Beispiel Electron [1] ermöglich den einfachen Einsatz durch einen eingebetten Browser.

4 Benutzte Technologien

In dem folgenden Abschnitt werden die benutzten Technologien beschrieben. Diese Erklärungen sind recht generell und gehen zunächst nicht auf die Verwendung der Technolgien in dem Projekt ein, dies wird aber im Abschnitt Überblick genauer beleuchtet.

4.1 Neuronale Netzwerke

Neuronale Netzwerke sind einfach zu benutzende Modelle, welche nicht-lineare Abhängigkeiten mit vielen latenten Variablen stochastisch abbilden können [6]. Im einfachen Sinne, sind sie gerichtete Graphen, deren Knoten oder Nodes aus ihren Inputs Werte errechnen und die an die folgenden Nodes weitergeben. Hierbei werden zwischen 3 verschiedenen Arten von Nodes unterschieden:

Input Nodes Über diese Nodes bekommt das Netzwerke die Input Parameter.

Hidden Nodes Nodes, welche das netzwerke-interne Modell repräsentieren.

Output Nodes Diese Nodes bilden die Repräsentation des Ergebisses ab.

Nachdem die Node aus den Inputs einen Wert errechnet hat, geht dieser durch eine Aktivierungsfunktion. Diese Funktion stellt den Zusammenhang zwischen dem Input und dem Aktivitätslevel der Node her. Man unterscheidet zwischen folgenden Aktivitätsfunktionen

Lineare Aktivitätsfunktion Der einfachste Fall, linearer Zusammenhang zwischen Inputs und Output.

Lineare Aktivitätsfunktion mit Schwelle Linearer Zusammenhang ab einem Schwellwert. Sehr nützlich um Rauschen herauszufiltern. Ein häufig genutzte Abhandlung davon:

ReLU Hier werden nur der positive Werte weitergeleitet: $f_x = x^+ = max(0, x)$

Binäre Schwellenfunktion Nur zwei Zustände möglich: 0 oder 1 (oder auch -1 oder 1)

Sigmoide Aktivitätsfunktion Benutzung entweder einer logistischen oder Tangens-Hyperbolicus Funktion. Diese Funktionen gehen bei sehr großen Werten gegen 1 und bei sehr negativen Werten gegen 0 (logistische Funktion) oder -1 (Tangens-Hyperbolicus Funktion). Diese Funktion bietet den Vorteil das sie das Aktivitätlevel begrenzt.

Jede der Nodes hat eine bestimmte Anzahl an Verbindungen, diese hängt von der Art der Nodes und deren Zweck ab. Wichtig ist jedoch, das jede Node mit mehreren anderen Nodes verbunden ist, dies soll heißen, den Output mehrerer Nodes als Input zu bekommen und den eigenen Output als Input für die folgenden Nodes weiterzuleiten. Die Stärke der Abhängigkeit zwischen zwei Nodes wird als Gewicht ausgedrückt. Jede Verbindung in einem Neuronalen Netzwerk hat ein Gewicht, welches mit dem Output der vorangegangen Node multipliziert wird bevor es als Input weiter verwendet wird. TODO BIAS

Das netzwerk-interne Modell wird in diesen Gewichten abgespeichert. Es repräsentiert also das Wissen, dass durch das Training entstanden ist.

Das Training eines Netzwerkes, ist das schrittweiße Anpassen der Gewichte bis es ein gutes Modell des Problems gelernt hat. Die Stärke von Neoronalen Netzen liegt darin, aus großen Mengen von Daten Gesetzmäßigkeiten oder Patterns zu erkennen. Ein einfaches Beispiel ist die Objekterkennung. Wenn ein Netzwerk Alltagsgegenstände erkennen soll, lernt es die Pixelgruppen welche ein Tisch von einem Bett unterscheiden. Damit dies funktioniert braucht man eine große Menge an Daten. Zunächst diwr das Training in zwei verschieden Arten unterteilt:

Supervised learning Innerhalb des Trainingsdatensets, hat jeder Datensatz einen vorgegeben Output Label. Zum Beispiel ein Bild von einem Auto ist auch so gekennzeichent. Nun werden so lange die Gewichte des Netzwerkes optimiert, bis ein jeweiliger Input auch den richtigen Output erzeugt.

Unupervised learning Hier hat das Trainingsdatenset keine Label. Die Gewichtsveränderungen erfolgen im Bezug zur der Ähnlichkeit von Inputs. Das soll heisen, wenn es viele verschiedene Bilder bekommt, werden Bilder mit ähnlichen Inhalten eine

hohe Nähe aufweißen, ein Bild von einem PKW wird näher an dem Bild von einem LKW sein als an dem Bild von einem Apfel.

4.2 Convolutional Neural Network - CNN

CNN sind Tiefe Neuronale Netzwerke mit einer bestimmten Architektur und spezialisiert auf die Verarbeitung von Bildern. Da man die Anwendungsdomäne eingeschränkt hat, kann man bestimmte Annahmen treffen, welche die Anzahl der Verbindungen und damit Rechenoperationen verringert und somit das Netz effektiver macht. Um aus Bildern, Informationen zu gewinnen, müssen die Ebenen des Netzwerkes nicht vollständig verbunden sein. Stattdessen werden Filter (Convolutions) und Sub-Sambling genutzt. Filter sind kleine Matrixen, die bestimmte Features entdecken, zum Beispiel Kanten mit bestimmter Ausrichtung. Durch das Erlernen der Filter im Training kann das Netzwerk aus den Pixel Werten, schrittweiße Abstaktere Feaures errechnen. Diese gehen von einfachen Kanten, zu komplexeren Umrissen, und schließlich zu vollständigen Teil-Objekten. Zum Beispiel werden aus vielen Kanten ein Kreis, dann kommen noch mehr dazu bis eine Feautere Map ein Auge abbildet. Aus dem Auge und der biologischen Signal Verarbeitung ist diese Architektur inspiriert [**?**]Hubel68). Einzelne Neuronen des celebralen Kortex reagieren auf Reize nur in einem beschränkten Bereich. Da diese Bereiche leicht überlappen können so diese Neuronen den gesamten Sichtbereich erkennen.

4.3 Recurrent Neural Network - RNN

Als RNN bezeichnet man neuronale Netze, welches Verbindungen, im Gegensatz zu FeedForward-Netzen, zu Neuronen der selben oder vorhergehenden Schichten besitzt. Dadurch kann es zeitliche Abhängigkeiten in den Input Daten detektieren. Diese Art von Netzen wird in der Spracherkennung, Maschineller Übersetzung und auch Handschrifterkennung eingesetzt.

4.3.1 Long short-term Memory - LSTM

Ein LSTM ist eine bestimmte Form der RNNs. Ein RNN kann durch dessen starre Sturktur immer nur eine bestimmte Anzahl von Schritten abspeichern. Zum Beispiel bei Videoanalysen jeweils die letzten 5 Frames. Das LSTM kann sich dynamisch Daten speichern, wodurch es irrelevante Daten aus vorherigen Schritten verwirft, relevantere jedoch länger abspeichert. Während des Trainings eines LSTMs, erlernt dieses auch das Speichern und Löschen. Dadurch kann es sehr viel effizienter als reine RNNs Daten mit temporaler Dimension auswerten.

4.4 Image Captioning Techniken

4.4.1 CNN-Part

4.4.2 RNN-Part

4.5 Training

Als Training wird der Prozess bezeichnet, während dem ein Neurales Netzwerk Wissen aus vorliegenden Daten extrahiert. Genau dieser Vorgang sorgt für den großen Erfolg von Neuralen Netzen. Anders als bei herkömlischen Statistischen Methoden können NNs aus riesigen Datenmengen Patterns und Sachverhältnisse lernen. Damit dies funktioniert, muss eine Kostenfunktion gebildet werden können, anhand bestimmt wird, wie weit das genutzte Modell von der Optimalen Lösung entfernt ist. Anhand diesem Abstand können die Parameter des gewählten Modells angepasst werden um dem Optimum näher zu kommen.

4.6 Sampling

Während dem Sampling, wird ein fertig trainiertes Neurales Netzwerk genutzt um aus Daten schlussfolgrungen abzuleiten. In dem Fall dieser Arbeit, wird probiert aus einem Screenshot, also einem Array von Pixeln ein deutungsvolle DSL-Sequenz zu erstellen.

4.7 Keras

Um die in dieser Arbeit genutzen Tiefen Neuronalen Netzwerke zu definieren wir das Framework Keras genutzt.

4.8 Domain-specific language - DSL

Eine Programmiersprache, die auf ein einzelne Problem-Domäne spezialisiert ist, wird DSL genannt. Im Gegensatz zur DSL steht die General Purpose Language, welches ein Programmiersprache ist, die sehr breit, für viele verschiedene Anwendungen, benutzt werden kann. Die Trennung zwischen DSL und GPL ist nicht immer klar, es kann zum Beispiel Teile einer Sprache geben die hoch spezialisiert für eine bestimmte Aufgabe sind, aber andere Teile von ihr können allgemeinere Aufgaben lösen. Auch historisch bedingt kann sich die Einordnung einer Sprache ändern. JavaScript wurde ursprünglich für ganz einfache Steuerung von Websites eingeführt, kann aber inzwischen für alles mögliche eingesetzt werden - vom Traininieren von CNNs im Browser, zu klassischen Backend-Jobs. In dieser Arbeit, wird eine hochspezialisierte, eigens erstellte Sprache aus Token, die eine Kombination aus HTML und CSS sind, benutzt.

4.8.1 Hypertext Markup Language - HTML

HTML ist die Standart Programmiersprache um Websites zu erstellen. Mit einzelnen HTML Elementen beschreibt es den semantischen Zusammenhang des Contents von Websites.

4.8.2 Cascading Style Sheets - CSS

CSS beschreibt die Präsentation, also das Aussehen, des Content einer Markup-Language (zum Beispiel HTML). Klassische Inhalte, sind Farben, Positionen und Effekte von User Interface Elementen.

5 Überblick

6 Vision Model

7 Language Model

8 Decoder

9 Daten Synthese

Da im Zuge dieser Arbeit eine Erweiterung der DSL des Original Papers implementiert wurde, ist es erforderlich, neue Trainingsdaten zu synthetisieren. Das DataCreationTool ist ein Programm, welches nach vorgegebenen Regeln einen Token-Baum erzeugt und diesen anschließend abspeichert. Dieser Token-Baum hat immer einen body-Token als Wurzel und der gesamten Inhalt liegt als dessen Kinder vor. Dafür wurde eine Helfer Klasse geschrieben, die ein Element in dem Token Baum abbildet. Diese kann zum einen Parameter wie den Token-Name, Inhalt und Kinder speichern, zum anderen enthält sie Funktionen zum Konvertieren des Baumes zu einer String-Representation sowie zum Rendering nach HTML/CSS.

9.1 Generieren der Token-Bäume

Die Token-Bäume werden in der Datei `createAllTokens.py` generiert. Diese Datei erzeugt alle möglichen Token-Kombination anhand der folgenden Regeln:

9.1.1 Gramatik

$$start \rightarrow [H, C] \tag{1}$$

$$H \rightarrow [Ml|Mr|S] \tag{2}$$

$$Ml \rightarrow [logoLeft, buttonWhite|logoLeft, buttonWhite, buttonWhite|...] \tag{3}$$

$$Mr \rightarrow [buttonWhite, logoRight|buttonWhite, buttonWhite, logoRight|...] \tag{4}$$

$$S \rightarrow [sidebarHeader, sidebarItem|sidebarHeader, sidebarItem, sidebarItem|...] \tag{5}$$

$$C \rightarrow [R|R, R|R, R, R] \tag{6}$$

$$R \rightarrow [S|D, D, |Q, Q, D|Q, D, Q|D, Q, Q|Q, Q, Q, Q] \tag{7}$$

$$S, D, Q \rightarrow [smallTitle, text, contentButton] \tag{8}$$

$$contentButton \rightarrow [buttonBlue, buttonGrey, buttonBlack] \tag{9}$$

Regeln 3 - 5 sind gekürzt. Es können bis zu 5 Buttons auftreten.

9.1.2 Zeichenerklärung

H Header der Website, enthält eins der folgenden Elemente:

- **Ml** Menue mit Logo auf der linken Seite
- **Mr** Menue mit Logo auf der rechten Seite
- **S** Sidebar

C Content der Website, besteht aus ein bis drei Wiederholungen dieses Elements:

- **R** Row, die aus einem oder mehreren Row Elementen bestehen kann:
  - **S** Single Row Element, die ganze Row ist mit diesem ausgefüllt.
  - **D** Double Row Element, ist so breit wie eine Hälfte der Row
  - **Q** Quadruble Row Element, ist so breit wie ein Viertel der Row

  Jedes dieser Elemente enthält den gleichen Inhalt:
- **smallTitle** Überschrift
- **text** Text-Inhalt
- **contentButton** Ein Button der entweder Blau, Grau oder Schwarz ist

9.1.3 DSL Mapping

Zu jedem dieser Token, existiert ein Mapping nach HTML/CSS. In einer extra Datei, `dsl-mapping.json` ist dies abgebildet:

```
{
  "opening-tag": "{",
  "closing-tag": "}",
  "body": "<!DOCTYPE html>\n <head>\n <meta charset=\"utf-8\">\n ...
  "header": "<nav class=\"menue\">\n    <ul class=\"nav nav-pills...
  "btn-active": "<li class=\"active\"><a href=\"#\">[]</a></li>\n...
  "btn-inactive-blue": "<button type=\"button\" class=\"btn btn-p...
  "btn-inactive-black": "<button type=\"button\" class=\"btn btn-...
  "btn-inactive-white": "<button type=\"button\" class=\"btn btn-...
  "btn-inactive-grey": "<button type=\"button\" class=\"btn btn-p...
  "row": "<div class=\"container\"><div class=\"row\">{}</div></d...
  "single": "<div class=\"col-lg-12\">\n{}\n</div>\n"
  "double": "<div class=\"col-lg-6\">\n{}\n</div>\n",
  "quadruple": "<div class=\"col-lg-3\">\n{}\n</div>\n",
  "big-title": "<h2>[]</h2>",
  "small-title": "<h4>[]</h4>",
  "text": "<p>[]</p>\n",
  "logo-left": "<a class=\"logo-left\">RFP</a>\n",
  "logo-right": "<a class=\"logo-right\">RFP</a>\n",
  "sidebar": "<div class=\"wrapper\">\n    <div id=\"sidebar\">\...
  "sidebar-element": "<li><a href=\"#\">[]</a><li>"
}
```

Um somit aus einem Token-Baum HTML/CSS Markup zu erzeugen, startet der Wurzelknoten eine rekursive Rendering-Funktion. Diese nutzt den zu den Knoten gehörigen Code und traversiert den gesamten Baum. Jeder Mapping String eines Tokens, der auch Kinderknoten haben kann, enthält einen Platzhalter, hier `{}`, mit dem signalisiert wird, wo der Code der Kinderknoten hingehört. Ähnlich gibt es ebenso einen Platzhalter für Text-Content, nämlich die Zeichen: `[]`.

10 Vergleich zu dem Original

11 Experimente

12 Zusammenfassung

13 Fazit

Literatur

[1] Electron. `https://www.electronjs.org/`.

[2] pix2code. `https://www.github.com/tonybeltramelli/pix2code`.

[3] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. *CoRR*, abs/1611.01989, 2016.

[4] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *CoRR*, abs/1705.07962, 2017.

[5] Tuan Anh Nguyen and Christoph Csallner. Reverse engineering mobile application user interfaces with remaui (t). *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 248–259, 2015.

[6] Günter Daniel Rey and Fabian Beck. Neuronale netze - eine einführung. `http://www.neuronalesnetz.de/downloads/neuronalesnetz_de.pdf`.