

Aufgabe 27 (Faltung mittels Fast-Fourier-Transformation)

Die *zirkuläre diskrete Faltung* $*$ zweier Vektoren $x, y \in \mathbb{C}^n$ liefert wieder einen Vektor aus \mathbb{C}^n und ist definiert als

$$(x * y)_j = \sum_{\ell=0}^{n-1} x_\ell y_{(j-\ell) \bmod n} \quad \text{für } j = 0, 1, \dots, n-1. \quad (1)$$

Wir verwenden zudem folgende Konvention für die diskrete Fourier-Transformation:

$$\mathcal{F} : \mathbb{C}^n \rightarrow \mathbb{C}^n, \quad \mathcal{F}(x) = z \quad \text{mit} \quad z_k = \sum_{j=0}^{n-1} e^{-2\pi i j k / n} x_j \quad \text{für } k = 0, 1, \dots, n-1,$$

mit entsprechender inversen Fourier-Transformation

$$\mathcal{F}^{-1} : \mathbb{C}^n \rightarrow \mathbb{C}^n, \quad \mathcal{F}^{-1}(z) = x \quad \text{mit} \quad x_j = \frac{1}{n} \sum_{k=0}^{n-1} e^{2\pi i j k / n} z_k \quad \text{für } j = 0, 1, \dots, n-1.$$

(a) Rechnen Sie das Faltungstheorem nach: für beliebige $x, y \in \mathbb{C}^n$ gilt

$$x * y = \mathcal{F}^{-1}(\mathcal{F}(x) \cdot \mathcal{F}(y)), \quad (2)$$

wobei \cdot die punktweise Multiplikation bezeichnet. [3 Punkte]

Hinweis: Sie können ohne Beweis die Orthogonalität der Fourier-Vektoren verwenden, d.h.

$$\frac{1}{n} \sum_{k=0}^{n-1} e^{2\pi i (j-\ell)k/n} = \delta_{(j-\ell) \bmod n, 0} \quad \forall j, \ell = 0, 1, \dots, n-1.$$

(b) Schreiben Sie jeweils eine Python-Funktion, die die Faltung zweier Vektoren via Gl. (1) bzw. Gl. (2) mittels FFT berechnet. Überprüfen Sie anhand eines kleinen Beispiels, dass beide Funktionen (bis auf numerische Rundungsfehler) auch tatsächlich dasselbe Ergebnis liefern. Was ist die asymptotische Laufzeit der Funktionen? [2 Punkte]

Hinweis: Die Fourier-Transformationen lassen sich mit `numpy.fft.fft` bzw. `numpy.fft.ifft` berechnen. Die asymptotische Laufzeit der FFT beträgt $\mathcal{O}(n \log(n))$.

(c) Eng verwandt mit der Faltung ist die *Kreuzkorrelation*

$$(x \star y)_j = \sum_{\ell=0}^{n-1} \overline{x_\ell} y_{(j+\ell) \bmod n} \quad \text{für } j = 0, 1, \dots, n-1,$$

also $j+\ell$ anstatt $j-\ell$ in Gl. (1), wobei $\overline{x_\ell}$ die komplexe Konjugation von x_ℓ bezeichnet. Wie muss der Vektor x modifiziert werden, um wieder auf die Form (1) zu kommen, d.h. für welchen Vektor x' gilt $x \star y = x' * y$? [1 Punkt]

Aufgabe 28 (Modulare Implementierung künstlicher neuronaler Netze)

Zur modularen Umsetzung beliebiger Netzwerk-Architekturen, d.h. einer flexiblen Hintereinanderschaltung verschiedener Schicht-Typen ("dense layer", "convolutional layer", ...), soll zunächst der Feedforward-Pass und die Backpropagation als separate Funktion für jeden Schicht-Typ implementiert werden. Wir bezeichnen die Eingabe einer Schicht mit $x = a^{\ell-1}$ und die Ausgabe mit $a = a^\ell$ und lassen zur Vereinfachung der Notation den Index ℓ im Folgenden weg. Wie bisher hängt die Kostenfunktion C direkt von der Ausgabe der letzten Schicht ab. Ausgedrückt durch x und a lauten somit exemplarische Feedforward und Backpropagation-Gleichungen:

- *Dense Layer*
Feedforward:

$$a_j = \sigma(z_j), \quad z_j = \sum_k w_{j,k} x_k + b_j.$$

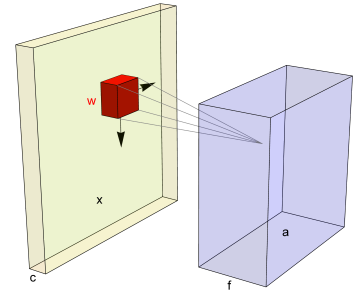
Backpropagation:

$$\frac{\partial C}{\partial x_k} = \sum_j \frac{\partial C}{\partial a_j} \sigma'(z_j) w_{j,k}, \quad \frac{\partial C}{\partial w_{j,k}} = \frac{\partial C}{\partial a_j} \sigma'(z_j) x_k, \quad \frac{\partial C}{\partial b_j} = \frac{\partial C}{\partial a_j} \sigma'(z_j).$$

- Convolutional Layer

$$a_{f,j} = \sigma(z_{f,j}), \quad z_{f,j} = \sum_{c,k} w_{f,c,k} x_{c,j+k} + b_f,$$

wobei $j \in \mathbb{Z}^2$ und $k \in \mathbb{Z}^2$ räumliche (Pixel-)Indizes bezeichnen, c den "Channel" der Eingabe (z.B. den Farbkanal) und f das jeweilige "Feature" der Ausgabe. Wir erlauben zusätzlich ein "zero-padding" der Eingabe x bezüglich der räumlichen (Pixel-)Dimensionen, d.h. eine Umrandung mit Nullen, so dass z.B. für $\text{pad}=2$ und einen 5×5 Gewichts-Filter die Höhe und Breite der Ausgabe gleich der Eingabe ist. Die "stride length" gibt an, um wie viele Pixel das rezeptive Feld jeweils nach rechts bzw. unten verschoben wird; typisch ist $\text{stride}=1$.



In `modular_net.zip` finden sich Python-Programme für eine modulare Implementierung neuronaler Netze¹, wobei `layers.py` die erwähnten Feedforward und Backpropagation-Funktionen für verschiedene Schicht-Typen enthält. Aus Effizienzgründen wird stets ein "mini-batch" aus N Ein- und Ausgabevektoren auf einmal abgearbeitet; diese werden in Matrizen mit erster Dimension N zusammengefasst. Es stellt sich als einfacher heraus, die Aktivierungsfunktion σ als eigene Schicht zu behandeln; im Sinne der Implementierung ist somit σ in den obigen Gleichungen die Identitätsfunktion und die Ableitung σ' stets 1.

- (a) Vervollständigen Sie den Code in `layers.py` an den (mit `#### ... ####`) angegebenen Stellen, und verifizieren Sie Ihre Lösung mittels `python -m unittest test_layers`. [2 Punkte]

Die Klasse `Solver` in `solver.py` übernimmt das Trainieren eines beliebigen Netzwerks, ohne Festlegung auf eine spezielle Architektur. Dieses muss hierzu lediglich eine Methode `loss` zum Berechnen der Kostenfunktion und der entsprechenden Gradienten (mittels Backpropagation) zur Verfügung stellen.

- (b) In `cnn_basic.py` findet sich eine Vorlage für ein einfaches "convolutional neural network". Vervollständigen Sie den Code und verifizieren Sie Ihre Lösung mittels `python -m unittest test_cnn_basic`. [2 Punkte]
- (c) Trainieren Sie nun das Netzwerk anhand eines Ausschnitts des MNIST Datensatzes, indem Sie `run_training.py` laufen lassen. Woran lässt sich Overfitting erkennen? [2 Punkte]
- In zukünftigen Aufgaben soll eine effizientere Implementierung der Convolutional und Max-Pool Layer auch das Trainieren anhand größerer Datensätze ermöglichen.

Tutoraufgabe (Convolutional Layer zur Bildbearbeitung)

Im Allgemeinen berechnet eine Convolutional Layer die Operation

$$a_{f,j} = \sigma(z_{f,j}), \quad z_{f,j} = \sum_{c,k} w_{f,c,k} x_{c,j+k} + b_f$$

für Eingabe x , wobei $j \in \mathbb{Z}^2$ und $k \in \mathbb{Z}^2$ räumliche (Pixel-)Indizes bezeichnen, c den "Channel" der Eingabe (z.B. den Farbkanal) und f das jeweilige "Feature" der Ausgabe.

Durch geeignete Wahl der Gewichte $w_{f,c,k}$ und Bias-Werte b_f können Convolutional Layers als Bildbearbeitungs-Werkzeuge fungieren, wie im Folgenden demonstriert werden soll. Der Einfachheit halber sei σ stets die Identitätsfunktion, also $a_{f,j} = z_{f,j}$. Die Anzahl der Features betrage 3 (interpretiert als Farbkanäle) oder 1 (interpretiert als Graustufen-Wert).

- (a) Erstellen Sie einen *Graustufen-Konvertierer*. Welche Dimension besitzt der Gewichtstensor w ?
- (b) Implementieren Sie einen 5×5 Gauß-Unschärfe-Filter durch Wahl von

$$w_{f,c,k} = \delta_{f,c} \frac{e^{-\frac{1}{4}(k_x^2 + k_y^2)}}{\sum_{k'} e^{-\frac{1}{4}(k_x'^2 + k_y'^2)}} \quad \text{für } |k_x| \leq 2, \quad |k_y| \leq 2.$$

- (c) Ein gebräuchlicher Kantendetektions-Filter ist der *Sobel-Operator*, der (pro Farbkanal) durch Faltung mit

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{bzw.} \quad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

die horizontale bzw. vertikale "Ableitung" der Helligkeitswerte approximiert. Implementieren Sie den Sobel-Operator. Wie lassen sich die beiden Ableitungen für die jeweiligen Farbkanäle zu einem einzelnen Wert kombinieren?

¹basierend auf <http://cs231n.github.io/assignments2018/assignment2>