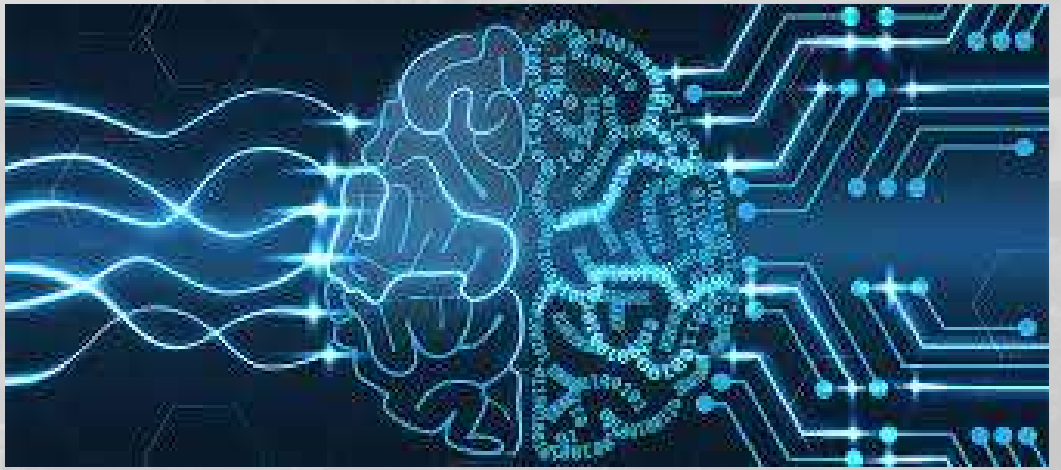
14620323
DEEP LEARNING



Numerical Computation for Deep Learning

PENGAMPU

Dr. Fajar Astuti Hermawati, S.Kom.,M.Kom.

Elsen Ronando, S.Si.,M.Si

Bagus Hardiansyah, S.Kom.,M.Si

Andrey Kartika Widhy H., S.Kom., M.Kom.

Capaian Pembelajaran

- Mampu mengidentifikasi konsep matematika dan mesin pembelajaran dasar untuk algoritma deep learning. [C2, A3]

Bahan Kajian

- Linear Algebra
- Probability and InformationTheory
- **Numerical Computation**

Numerical Computation

Why Numerical Computation?

- Algoritma pembelajaran mesin biasanya membutuhkan perhitungan numerik yang tinggi.
- Ini biasanya merujuk pada algoritma yang memecahkan masalah matematika dengan metode yang memperbarui perkiraan solusi melalui proses iteratif, daripada menurunkan rumus secara analitik untuk memberikan ekspresi simbolis untuk solusi yang tepat.
- Operasi umum termasuk **optimisasi** (menemukan nilai argumen yang meminimalkan atau memaksimalkan suatu fungsi) dan menyelesaikan sistem persamaan linier.

Numerical concerns for implementations of deep learning algorithms

- Algoritma sering ditentukan dalam bentuk **bilangan real**; bilangan real tidak dapat diimplementasikan dalam komputer yang terbatas
  - Apakah algoritma masih berfungsi saat diimplementasikan dengan **jumlah bit yang terbatas**.
- Apakah perubahan kecil pada input ke suatu fungsi menyebabkan perubahan besar pada output?
  - Kesalahan pembulatan, noise, kesalahan pengukuran dapat menyebabkan perubahan besar
  - Pencarian iteratif untuk input terbaik sulit dilakukan

Overflow and Underflow

- Kesulitan mendasar dalam melakukan matematika kontinu pada komputer digital adalah kita perlu **merepresentasikan bilangan real tak terhingga banyaknya dengan pola bit yang jumlahnya terhingga**.
- Ini berarti bahwa untuk hampir semua bilangan real, kita mengalami beberapa **kesalahan perkiraan** saat merepresentasikan bilangan di komputer. Dalam banyak kasus, ini hanyalah **kesalahan pembulatan**.
- **Galat pembulatan bermasalah**, terutama jika digabungkan di banyak operasi, dan dapat menyebabkan algoritma yang bekerja secara teori gagal dalam praktiknya jika tidak dirancang untuk **meminimalkan akumulasi kesalahan pembulatan**.

Overflow and Underflow

- Salah satu bentuk kesalahan pembulatan yang sangat merusak adalah ***underflow***.
- ***Underflow*** terjadi ketika angka mendekati nol dibulatkan menjadi nol.
- Banyak fungsi berperilaku berbeda secara kualitatif ketika argumen mereka nol daripada angka positif kecil.
- Misalnya, kita biasanya ingin menghindari pembagian dengan nol(dicide by zero) atau mengambil logaritma nol
  - ini biasanya diperlakukan sebagai $-\infty$ (infinite), yang kemudian menjadi bukan angka (NaN) jika digunakan untuk banyak operasi aritmatika selanjutnya

Overflow and Underflow

- Bentuk lain dari kesalahan numerik yang sangat merusak adalah Overflow.
- Overflow terjadi ketika **angka dengan magnitudo besar** didekati sebagai ∞ atau $-\infty$.
  - Aritmatika lebih lanjut biasanya akan mengubah nilai tak terhingga ini menjadi nilai bukan angka (NaN)

Overflow and Underflow

- Salah satu contoh fungsi yang harus distabilkan terhadap underflow dan overflow adalah **fungsi softmax**.
- Fungsi softmax sering digunakan untuk memprediksi probabilitas yang terkait dengan distribusi multinoulli.
- Fungsi softmax didefinisikan sbg:

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}.$$

Overflow and Underflow

- Pertimbangkan apa yang terjadi ketika semua xi sama dengan beberapa konstanta *c*. Secara analitis, kita dapat melihat bahwa semua keluaran harus sama dengan 1/*n*
  - Secara numerik, hal ini mungkin tidak terjadi jika *c* bermagnitudo besar.
  - Jika c sangat negatif, maka *exp*(*c*) akan berkurang.
  - Ini berarti penyebut softmax akan menjadi 0, sehingga hasil akhirnya tidak terdefinisi.
  - Ketika *c* sangat besar dan positif, *exp*(*c*) akan overflow, sekali lagi menghasilkan ekspresi secara keseluruhan menjadi tidak terdefinisi.
- Kedua kesulitan ini dapat diatasi dengan mengevaluasi **softmax**(*z*) di mana $z = x - \max_i x_i$.
  - Aljabar sederhana menunjukkan bahwa nilai fungsi softmax tidak diubah secara analitik dengan menambahkan atau mengurangkan skalar dari vektor masukan.
  - Mengurangi $\max_i x_i$ menghasilkan argumen terbesar untuk *exp* menjadi **0**, yang mengesampingkan kemungkinan overflow.
  - Demikian juga, setidaknya satu suku dalam penyebut memiliki nilai 1, yang mengesampingkan kemungkinan underflow pada penyebut yang mengarah ke pembagian dengan nol.

Poor Conditioning

- Pengkondisian mengacu pada **seberapa cepat suatu fungsi berubah** sehubungan dengan perubahan kecil pada inputnya.
- Fungsi yang **berubah dengan cepat ketika inputnya sedikit terganggu** dapat **menimbulkan masalah bagi perhitungan** ilmiah karena **kesalahan pembulatan dalam pindaian input menghasilkan perubahan besar pada output**.

Poor Conditioning

- Misalkan fungsi $f(x) = \boldsymbol{A}^{-1}x$. Ketika $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ memiliki dekomposisi nilai eigen, **nomor kondisinya (*condition number*)** adalah

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|.$$

- Ini adalah **rasio besarnya nilai eigen terbesar dan terkecil**.
- Ketika **angka ini besar**, **inversi matriks sangat sensitif terhadap kesalahan input**

Poor Conditioning

- Sensitivitas ini adalah **properti intrinsik** dari matriks itu sendiri, **bukan hasil kesalahan pembulatan** selama inversi matriks.
- Matriks yang dikondisikan dengan buruk **memperkuat kesalahan** yang sudah ada sebelumnya **saat kita mengalikan dengan invers** matriks yang sebenarnya.
- Dalam prakteknya, kesalahan tersebut akan **diperparah** lebih lanjut dengan **kesalahan numerik dalam proses inversi** itu sendiri

Gradient-Based Optimization

- Sebagian besar algoritma pembelajaran mendalam melibatkan semacam pengoptimalan.
- **Optimasi mengacu pada tugas meminimalkan atau memaksimalkan beberapa fungsi $f(x)$ dengan mengubah x.**
  - Kita biasanya mengutarakan sebagian besar masalah pengoptimalan dalam hal **meminimalkan $f(x)$**.
  - **Maksimalisasi** dapat dicapai melalui algoritma minimalisasi dengan **meminimalkan $-f(x)$**.

Gradient-Based Optimization

- Fungsi yang ingin kita minimalkan atau maksimalkan disebut **fungsi tujuan, atau kriteria**.
- Saat kita meminimalkannya, kita juga bisa menyebutnya sebagai **fungsi biaya, fungsi kerugian, atau fungsi kesalahan**.
- Kita sering menyatakan nilai yang meminimalkan atau memaksimalkan suatu fungsi dengan superskrip $*$.
  - Sebagai contoh, kita dapat mengatakan $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$

Gradient-Based Optimization

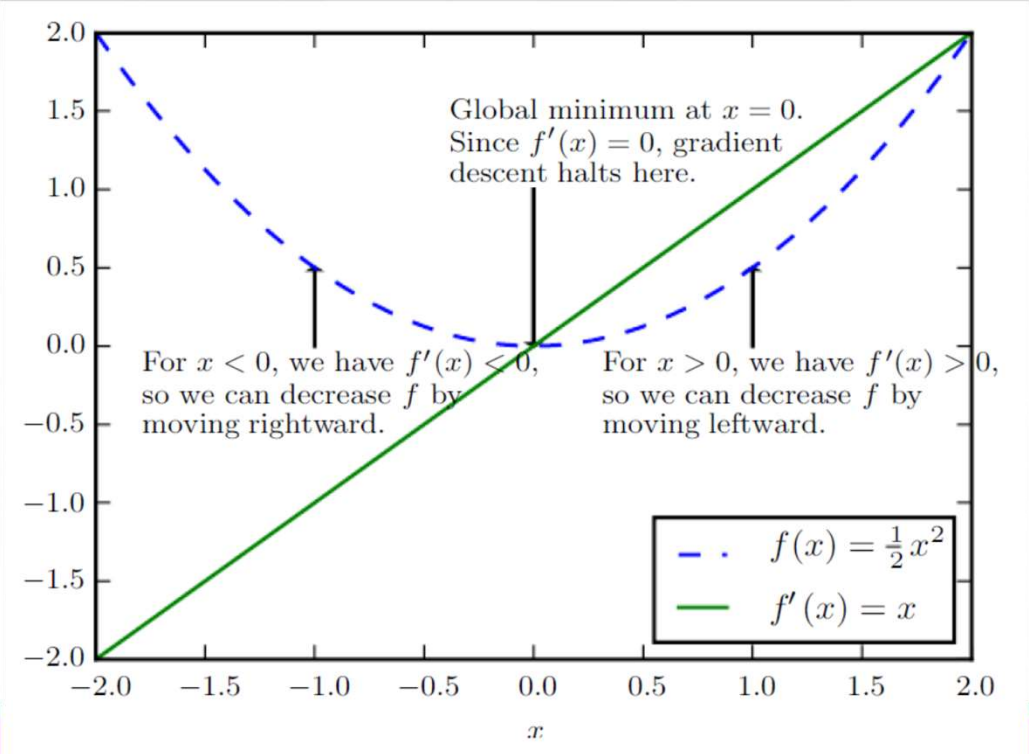
- Misalkan kita memiliki function $y=f(x)$, di mana x dan y adalah bilangan real.
- **Turunan dari fungsi** ini dinotasikan sebagai $f'(x)$ atau sebagai $\frac{dy}{dx}$.
- Turunan $f'(x)$ memberikan kemiringan $f(x)$ di titik x.
- Dengan kata lain, ini menentukan bagaimana menskalakan perubahan kecil pada input untuk mendapatkan perubahan yang sesuai pada output:

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x).$$

Gradient-Based Optimization

- **Derivatif**, oleh karena itu, **berguna untuk meminimumkan suatu fungsi** karena ia memberitahu kita bagaimana mengubah *x* untuk membuat perbaikan kecil pada *y*.
- Misalnya, kita tahu bahwa $f(x - \epsilon\, sign(f'(x))$ kurang dari $f(x)$ untuk ϵ cukup kecil.
- Dengan demikian kita dapat mengurangi $f(x)$ dengan menggerakkan *x* dalam langkah-langkah kecil dengan tanda turunan yang berlawanan.
- Teknik ini disebut **penurunan gradien (gradient descent)** (Cauchy, 1847).

Gradient-Based Optimization



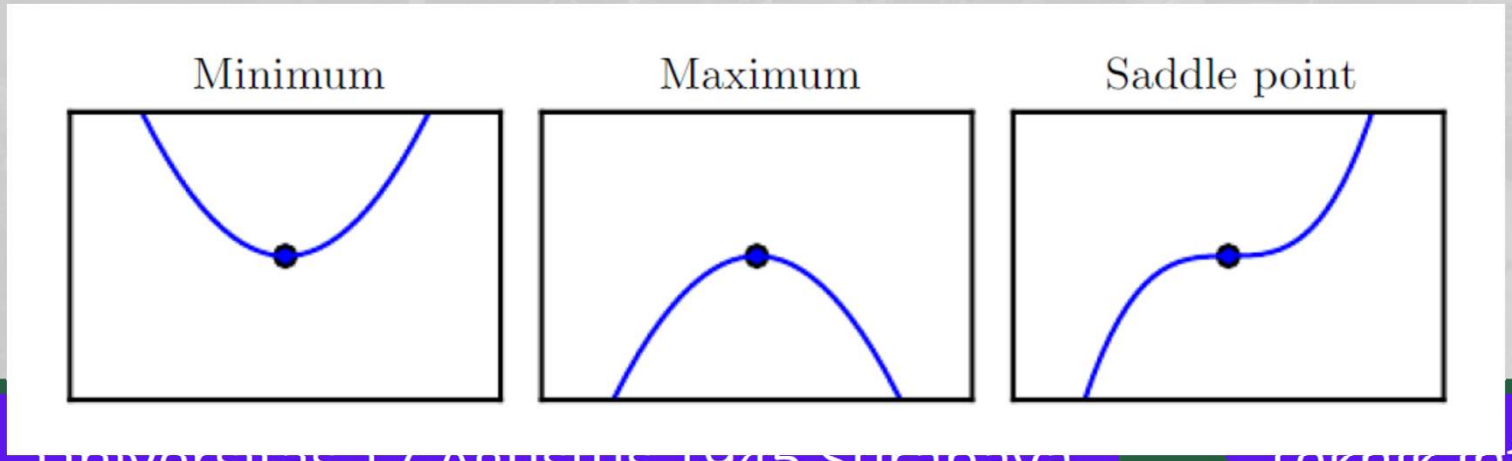
Sebuah ilustrasi tentang bagaimana algoritma ***gradient descent*** menggunakan turunan dari suatu fungsi untuk mengikuti fungsi menurun ke minimum

Gradient-Based Optimization

- Ketika $f'(x) = 0$, turunannya tidak memberikan informasi tentang ke arah mana harus bergerak. Titik dimana $f'(x) = 0$ dikenal sebagai **titik kritis atau titik stasioner**.
- **Minimum lokal** adalah titik di mana $f(x)$ lebih rendah daripada semua titik tetangganya, sehingga tidak mungkin lagi menurunkan $f(x)$ dengan membuat langkah-langkah tak terhingga.
- **Maksimum lokal** adalah titik di mana $f(x)$ lebih tinggi dari pada semua titik tetangga sehingga tidak mungkin untuk meningkatkan $f(x)$ dengan membuat langkah-langkah yang sangat kecil.
- Beberapa **titik kritis bukanlah maksima atau minima**. Ini dikenal sebagai **titik sadel (*saddle points*)**.

Gradient-Based Optimization

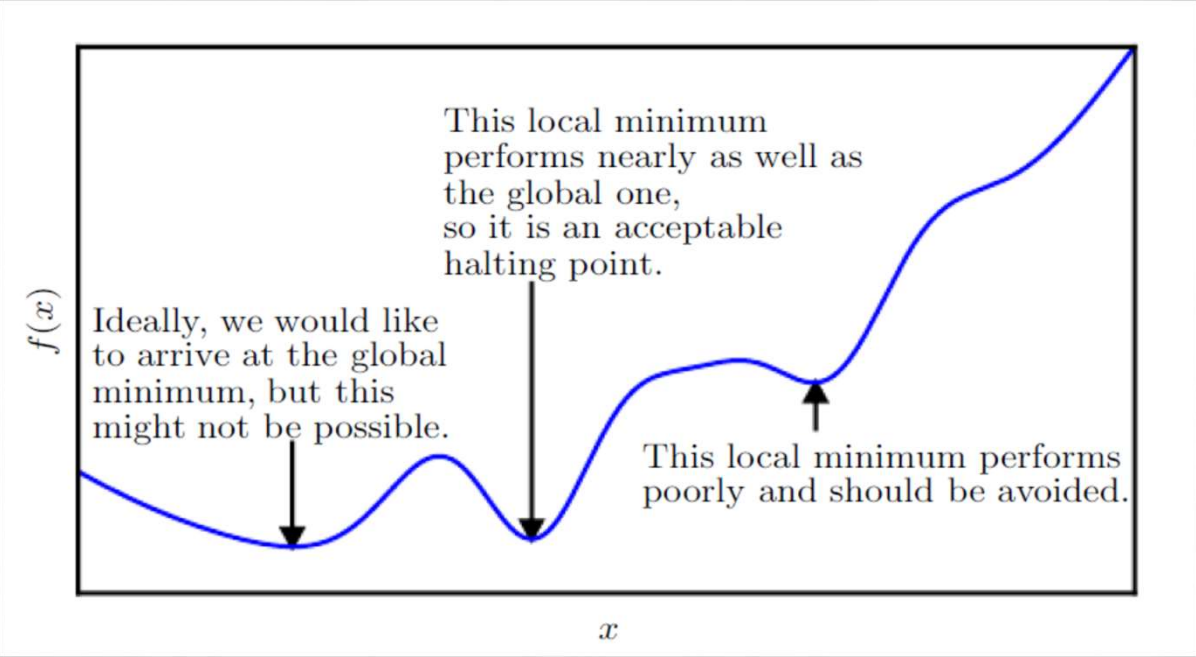
- **Jenis titik kritis**. Contoh tiga jenis titik kritis dalam satu dimensi. Titik kritis adalah titik dengan kemiringan nol. Titik seperti itu bisa berupa minimum lokal, yang lebih rendah dari titik tetangga; maksimum lokal, yang lebih tinggi dari titik tetangga; atau titik sadel, yang memiliki tetangga yang lebih tinggi dan lebih rendah dari titik itu sendiri



Gradient-Based Optimization

- Suatu titik yang memperoleh nilai $f(x)$ terendah absolut adalah **minimum global**.
  - Hanya ada satu minimum global atau beberapa minimum global dari fungsi tersebut.
  - Mungkin juga ada minima lokal yang tidak optimal secara global.
- Dalam konteks deep learning, kita mengoptimalkan fungsi yang mungkin memiliki **banyak minima lokal yang tidak optimal** dan banyak titik pelana (*saddle point*) yang dikelilingi oleh daerah yang sangat datar.
- Semua ini membuat pengoptimalan menjadi sulit, terutama jika masukan ke fungsi bersifat multidimensi.

Gradient-Based Optimization



- **<u>Perkiraan minimalisasi</u>**.
- Algoritma pengoptimalan mungkin gagal untuk menemukan minimum global ketika ada beberapa minima lokal atau dataran tinggi (*plateau*) yang ada.
- Dalam konteks pembelajaran mendalam, kita umumnya menerima solusi seperti itu meskipun tidak benar-benar minimal, asalkan sesuai dengan nilai fungsi biaya yang rendah secara signifikan.

Gradient-Based Optimization

- Kita sering meminimalkan fungsi yang memiliki banyak input: $f: \mathbb{R}^n \longrightarrow \mathbb{R}$.
- Agar konsep "**minimisasi**" masuk akal, harus tetap ada hanya satu output (skalar)
- Untuk fungsi dengan banyak input, kita harus menggunakan konsep turunan parsial.
- **Turunan parsial** $\frac{\partial}{\partial_{x_i}} f(x)$ mengukur bagaimana *f* berubah karena hanya variabel x_i yang bertambah pada titik *x*.

Gradient-Based Optimization

- **Gradien** menggeneralisasikan gagasan turunan ke kasus di mana turunannya sehubungan dengan vektor: **gradien** f adalah vektor yang berisi semua turunan parsial, dilambangkan $\nabla_x f(x)$**.**
- **Elemen *i* dari gradien** adalah **turunan parsial dari terhadap** x_i.
- Dalam banyak dimensi, **titik kritis adalah titik di mana setiap elemen gradien sama dengan nol.**

Gradient-Based Optimization

- Turunan arah dalam arah *u* (vektor satuan) adalah kemiringan fungsi *f* dalam arah *u*.
- Dengan kata lain, turunan arah adalah turunan dari fungsi $f(x + \alpha u)$ terhadap α, dievaluasi di $\alpha = 0$.
- Dengan menggunakan aturan rantai, kita dapat melihat bahwa $\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u})$ bernilai $u^{\top} \nabla_x f(x)$ ketika $\alpha = 0$.

Gradient-Based Optimization

- Untuk meminimumkan *f*, kita ingin mencari arah di mana *f* berkurang paling cepat. Kita dapat melakukannya dengan menggunakan turunan arah:

$$\min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

$$= \min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} ||\boldsymbol{u}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\theta$$

- di mana θ adalah sudut antara *u* dan gradien.
- Mensubstitusikan $\|u\|_2 = 1$ dan mengabaikan faktor-faktor yang tidak bergantung pada *u*, ini disederhanakan menjadi $min_u \cos\theta$. Ini diminimalkan ketika kita menunjuk ke arah yang berlawanan sebagai gradien.

Gradient-Based Optimization

- Dengan kata lain, **titik gradien langsung menanjak**, dan **titik gradien negatif langsung menurun**.
- Kita dapat menurunkan *f* dengan bergerak ke arah gradien negatif.
- Ini dikenal sebagai **metode penurunan paling curam (steepest descent)** atau **penurunan gradien (gradient descent)**.

Gradient-Based Optimization

- Steepest descent mengusulkan sebuah titik baru:

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

- di mana ϵ adalah tingkat pembelajaran (***learning rate***), skalar positif yang menentukan ukuran langkah.
  - Kita dapat memilih ϵ dengan beberapa cara berbeda.
  - Pendekatan yang populer adalah menyetel ϵ ke konstanta kecil.
  - Kadang-kadang, kita dapat memecahkan ukuran langkah yang membuat turunan arah menghilang.
  - Pendekatan lain adalah mengevaluasi $f(x - \epsilon \nabla_x f(x))$ untuk beberapa nilai ϵ dan memilih salah satu yang menghasilkan nilai fungsi objektif terkecil. Strategi terakhir ini disebut **pencarian garis** (***line search***).

Gradient-Based Optimization

- Penurunan tercuram (***Steepest descent***) menyatu ketika setiap elemen gradien adalah nol (atau, dalam praktiknya, sangat mendekati nol).
- Dalam beberapa kasus, kita mungkin dapat menghindari menjalankan algoritma iteratif ini dan langsung melompat ke titik kritis dengan menyelesaikan persamaan $\nabla_x f(x) = 0$ untuk x.
- Meskipun penurunan gradien (*gradient descent*) terbatas pada optimisasi dalam ruang kontinu, konsep umum pembuatan berulang kali sebuah langkah kecil (yang kira-kira merupakan langkah kecil terbaik) menuju konfigurasi yang lebih baik dapat digeneralisasikan ke ruang diskrit. Menaikkan fungsi objektif dari parameter diskrit disebut ***hill climbing***.

Jacobian and Hessian Matrices

- Terkadang kita perlu mencari **semua turunan parsial dari suatu fungsi** yang masukan dan keluarannya adalah vektor.
- **Matriks yang mengandung semua turunan parsial** tersebut dikenal sebagai **matriks Jacobian**.
- Secara khusus, jika kita memiliki fungsi $f: \mathbb{R}^m \longrightarrow \mathbb{R}^n$, maka matriks Jacobian $\boldsymbol{J} \in \mathbb{R}^{nxm}$ dari f didefinisikan sedemikian sehingga $J_{i,j} = \frac{\partial}{\partial x_i} f(x_i)$

Jacobian and Hessian Matrices

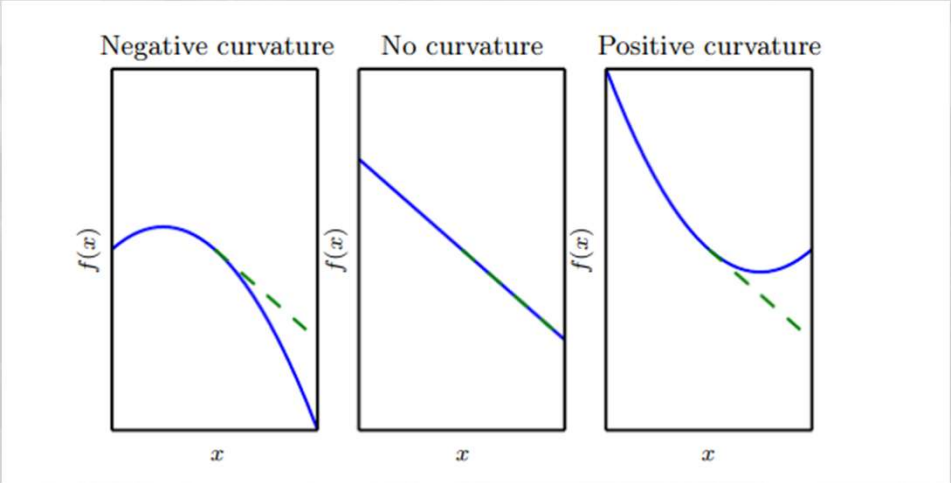
- Kita juga terkadang tertarik pada turunan dari turunan. Ini dikenal sebagai **turunan kedua**.
- Misalnya, untuk fungsi $f: \mathbb{R}^n \longrightarrow \mathbb{R}$, turunan terhadap x_i dari turunan f terhadap x_j dinotasikan sebagai $\frac{\partial^2}{\partial x \partial y} f$.
- **Turunan kedua memberi tahu** kita **bagaimana turunan pertama akan berubah** saat kita memvariasikan input.
- Ini penting karena memberi tahu kita **apakah langkah gradien akan menyebabkan peningkatan** sebanyak yang kita harapkan berdasarkan gradien saja.

Jacobian and Hessian Matrices

- Kita dapat memikirkan **turunan kedua sebagai mengukur kelengkungan**.
- Misalkan kita memiliki fungsi kuadrat (banyak fungsi yang muncul dalam praktiknya tidak kuadrat tetapi dapat didekati dengan baik sebagai kuadrat, setidaknya secara lokal).
  - Jika fungsi tersebut **memiliki turunan kedua dari nol**, maka **tidak ada kelengkungan**. Ini adalah **garis datar sempurna**, dan **nilainya hanya dapat diprediksi menggunakan gradien**. **Jika gradien adalah 1**, maka kita dapat membuat langkah ukuran ϵ sepanjang gradien negatif, dan fungsi biaya akan berkurang sebesar ϵ.
  - Jika **turunan keduanya negatif**, fungsinya **melengkung ke bawah**, sehingga fungsi **biaya sebenarnya akan berkurang lebih dari ϵ**.
  - Akhirnya, jika **turunan keduanya positif**, fungsi tersebut **melengkung ke atas**, sehingga **fungsi biaya dapat menurun kurang dari** ϵ.

Jacobian and Hessian Matrices

- Garis putus-putus menunjukkan nilai fungsi biaya yang kita harapkan berdasarkan informasi gradien saja saat kita membuat langkah gradien menurun.
- Dengan kelengkungan negatif, fungsi biaya sebenarnya menurun lebih cepat daripada prediksi gradien.
- Tanpa kelengkungan, gradien memprediksi penurunan dengan benar.
- Dengan kelengkungan positif, fungsi menurun lebih lambat dari yang diharapkan dan akhirnya mulai meningkat, sehingga langkah yang terlalu besar sebenarnya dapat meningkatkan fungsi secara tidak sengaja.



Jacobian and Hessian Matrices

- Ketika fungsi kita memiliki **beberapa dimensi masukan**, ada **banyak turunan kedua**.
- Derivatif ini dapat dikumpulkan bersama menjadi matriks yang disebut **matriks Hessian**.
- Matriks Hessian ***H***(*f*)(*x*) didefinisikan :

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}).$$

- **Hessian adalah Jacobian dari gradien**

Jacobian and Hessian Matrices

- Turunan parsial kedua kontinu, operator diferensial bersifat komutatif;

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\boldsymbol{x}).$$

- Ini menyiratkan bahwa $\boldsymbol{H}_{i,j} = \boldsymbol{H}_{j,i}$, sehingga **matriks Hessian simetris pada titik-titik tersebut**.
- Sebagian besar fungsi yang kita temui dalam konteks deep learning **memiliki Hessian simetris** hampir di semua tempat.
- Karena matriks Hessian adalah riil dan simetris, kita dapat menguraikannya (decompose) menjadi **satu set nilai eigen** nyata dan **basis vektor eigen ortogonal**.

Jacobian and Hessian Matrices

- **Turunan kedua dalam arah tertentu** yang dinyatakan dengan vektor satuan ***d*** diberikan oleh $\boldsymbol{d}^T\boldsymbol{H}\boldsymbol{d}$.
- Ketika ***d*** adalah **vektor eigen dari *H***, **turunan kedua** dalam arah tersebut diberikan oleh **nilai eigen yang sesuai**.
- Untuk arah lain dari ***d***, turunan kedua arah adalah **rata-rata terboboti** (*weighted average*) dari **semua nilai eigen**, dengan bobot antara 0 dan 1, dan **vektor eigen yang memiliki sudut lebih kecil** dengan ***d*** menerima lebih banyak bobot.
- **Nilai eigen maksimum menentukan turunan kedua maksimum**, dan **nilai eigen minimum menentukan turunan kedua minimum**

Jacobian and Hessian Matrices

- **Derivatif kedua** (terarah) memberi tahu kita **seberapa baik kita dapat mengharapkan langkah penurunan gradien** untuk bekerja.
- Kita dapat membuat **pendekatan deret Taylor orde kedua** ke fungsi $f(x)$ di sekitar titik saat ini $x^{(0)}$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)}),$$

- di mana $\boldsymbol{g}$ adalah gradien dan $\boldsymbol{H}$ adalah Hessian di $\boldsymbol{x}^{(0)}$

Jacobian and Hessian Matrices

- Jika kita menggunakan laju pembelajaran ϵ, maka titik baru $\boldsymbol{x}$ akan diberikan oleh $\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}$. Mengganti ini ke perkiraan , diperoleh:

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2} \epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}.$$

- Ada tiga istilah di sini:
  - **nilai asli fungsi**,
  - **peningkatan yang diharapkan karena kemiringan fungsi**, dan
  - **koreksi yang harus kita terapkan untuk memperhitungkan kelengkungan fungsi**

Jacobian and Hessian Matrices

- Ketika **koreksi** terlalu besar, langkah penurunan gradien sebenarnya bisa bergerak menanjak.
- Ketika $\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}$ adalah **nol atau negatif**, pendekatan deret Taylor memprediksi bahwa **peningkatan ϵ selamanya** akan **menurunkan f selamanya**.
- Ketika $\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}$ **positif**, penyelesaian untuk ukuran langkah optimal yang **menurunkan hampiran deret Taylor** dari fungsi yang paling banyak menghasilkan:

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}}.$$

Jacobian and Hessian Matrices

- Dalam kasus terburuk, ketika ***g* sejajar dengan vektor eigen** dari ***H*** yang sesuai dengan **nilai eigen maksimal** $\lambda_{\max}$, maka **ukuran langkah optimal ini** diberikan oleh $\frac{1}{\lambda_{max}}$.
- Sejauh fungsi yang kita minimalkan dapat didekati dengan baik oleh fungsi kuadrat, **nilai eigen dari Hessian** dengan demikian **menentukan skala laju pembelajaran**.

Jacobian and Hessian Matrices

- Turunan kedua dapat digunakan untuk menentukan apakah suatu titik kritis merupakan maksimum lokal, minimum lokal, atau titik sadel.
- Ingatlah bahwa pada titik kritis, $\boldsymbol{f}'(x) = 0$.
- Ketika turunan kedua $\boldsymbol{f}''(x) > 0$, turunan pertama $\boldsymbol{f}'(x)$ **bertambah saat kita bergerak ke kanan** dan **berkurang saat kita bergerak ke kiri**.
- Ini berarti $\boldsymbol{f}'(x - \epsilon) < 0$ dan $\boldsymbol{f}'(x + \epsilon) > 0$ untuk $\boldsymbol{\epsilon}$ **yang cukup kecil**.
- Dengan kata lain, saat kita bergerak ke kanan, lereng (***slope***) mulai menanjak ke kanan, dan saat kita bergerak ke kiri, lereng (***slope***) mulai mengarah ke kiri menanjak.

Jacobian and Hessian Matrices

- Jadi, ketika $f'(x) = 0$ dan $f''(x) > 0$, kita dapat menyimpulkan bahwa ***x*** adalah **minimum lokal**.
- Demikian pula, ketika $f'(x) = 0$ dan $f''(x) < 0$, kita dapat menyimpulkan bahwa ***x*** adalah **maksimum lokal**.
- Ini dikenal sebagai uji turunan kedua (***second derivative test***).
- Sayangnya, ketika $f''(x) = 0$, tes tersebut tidak meyakinkan. Dalam hal ini ***x*** dapat berupa **titik pelana** atau **bagian dari daerah datar**.

Jacobian and Hessian Matrices

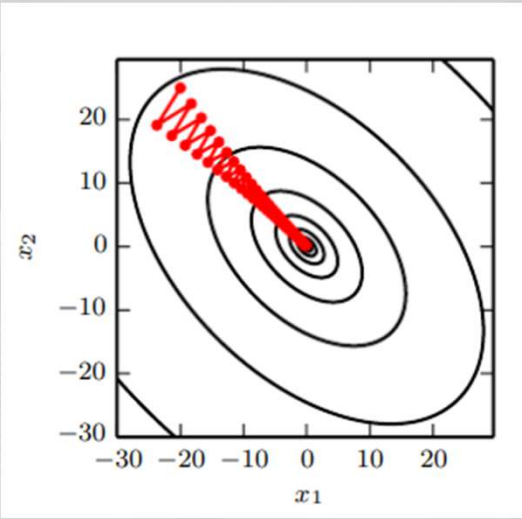
- Dalam **dimensi lebih dari satu**, terdapat **turunan kedua yang berbeda untuk setiap arah pada satu titik**.
- Jumlah **kondisi Hessian pada titik ini** mengukur **seberapa besar perbedaan turunan kedua satu sama lain**.
- Ketika Hessian memiliki **sejumlah kondisi yang buruk** (*poor conditioning*), **penurunan gradien berperforma buruk**.
- Hal ini karena dalam satu arah, turunannya meningkat dengan cepat, sementara di arah lain, meningkat dengan lambat.

Jacobian and Hessian Matrices

- Turunan **gradien tidak menyadari perubahan turunan** ini, sehingga tidak mengetahui bahwa ia perlu menjelajahi secara istimewa ke arah di mana turunannya tetap negatif lebih lama.
- Jumlah langkah yang buruk juga **mempersulit pemilihan ukuran langkah** yang baik.
- Ukuran ***step* harus cukup kecil** untuk menghindari melampaui minimum dan menanjak ke arah dengan kelengkungan positif yang kuat.
- Ini biasanya berarti bahwa ukuran langkah terlalu kecil untuk membuat kemajuan yang signifikan ke arah lain dengan kelengkungan yang lebih sedikit

Jacobian and Hessian Matrices

- Perhitungan gradien tidak dapat menggunakan informasi kelengkungan yang terkandung dalam matriks Hessian.
- Di sini perhitungan gradien digunakan untuk meminimalkan fungsi kuadrat $f(x)$ di mana bilangan kondisi matriks Hessian adalah 5.
- Artinya, kelengkungan arah yang paling melengkung adalah lima kali lebih besar daripada kelengkungan yang arahnya paling tidak melengkung.
- Penurunan gradien menuruni dinding ngarai beberapa kali membutuhkan waktu karena ini adalah fitur yang paling curam.
- Karena ukuran anak tangga agak terlalu besar, cenderung melebihi bagian bawah fungsi, sehingga harus memanjat sepanjang dinding ngarai yang berlawanan di iterasi berikutnya.



Jacobian and Hessian Matrices

- Masalah ini dapat diatasi dengan **menggunakan informasi dari matriks Hessian** untuk memandu pencarian.
- Metode paling sederhana untuk melakukannya dikenal sebagai **metode Newton**.
- Metode Newton didasarkan pada penggunaan deret Taylor orde dua untuk mendekati $\boldsymbol{f}(x)$ di dekat beberapa titik $x^{(0)}$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)}).$$

Jacobian and Hessian Matrices

- Jika kita kemudian memecahkan titik kritis dari fungsi ini, kita dapatkan:

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}).$$

  - Ketika ***f*** adalah **fungsi kuadrat definit positif**, metode Newton terdiri dari menerapkan persamaan tersebut **satu kali** untuk melompat ke fungsi minimum secara langsung.
  - Ketika ***f*** **tidak benar-benar kuadratik** tetapi dapat didekati secara lokal sebagai kuadratik pasti positif, metode Newton terdiri dari penerapan persamaan diatas **beberapa kali**.

Jacobian and Hessian Matrices

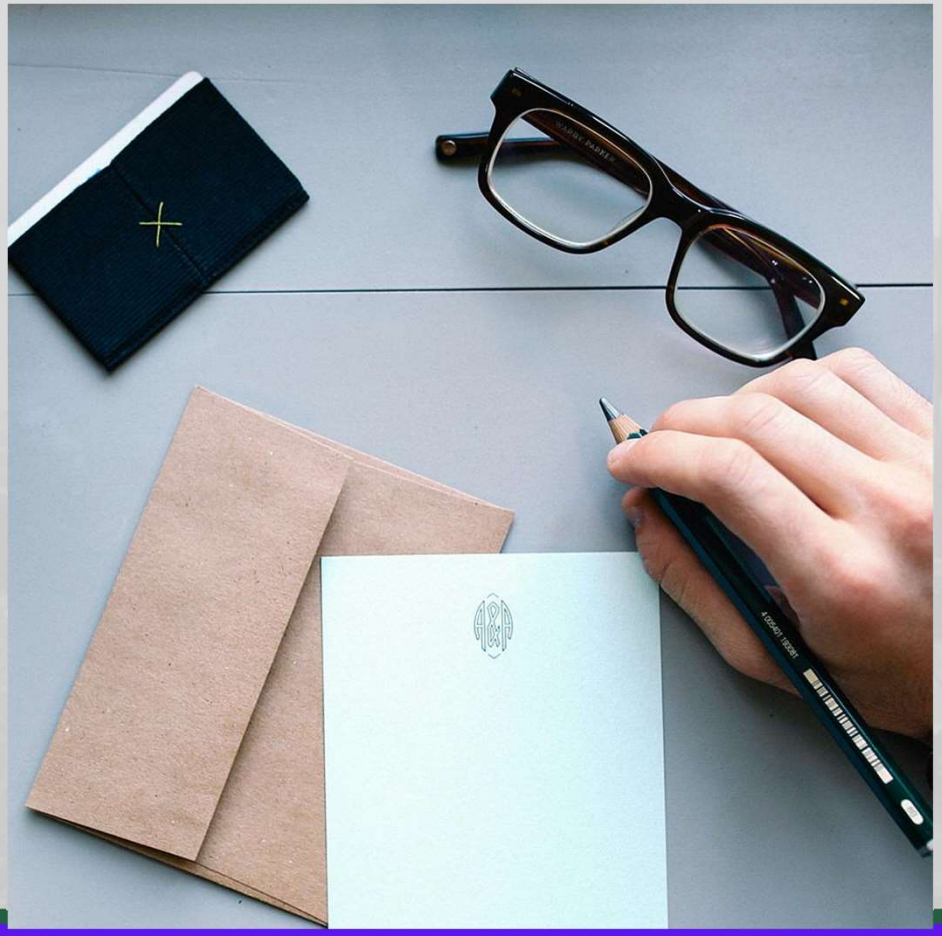
- Algoritma pengoptimalan yang hanya menggunakan gradien, seperti penurunan gradien, disebut **algoritma pengoptimalan orde pertama**.
- Algoritma pengoptimalan yang juga menggunakan matriks Hessian, seperti metode Newton, disebut **algoritma pengoptimalan orde kedua.**

Jacobian and Hessian Matrices

- Dalam konteks pembelajaran mendalam, terkadang kita mendapatkan beberapa jaminan dengan membatasi diri pada **fungsi yang kontinu Lipschitz** atau memiliki **turunan kontinu Lipschitz**.
- Fungsi kontinu Lipschitz adalah fungsi f yang laju perubahannya dibatasi oleh **konstanta Lipschitz** $\mathcal{L}$:

$$\forall \boldsymbol{x}, \forall \boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L}||\boldsymbol{x} - \boldsymbol{y}||_2.$$

- Properti ini berguna karena memungkinkan kita untuk mengukur asumsi kita bahwa perubahan kecil pada input yang dibuat oleh algoritma seperti penurunan gradien akan memiliki perubahan kecil pada output.



TERIMA KASIH