Föreläsning 6 - Neurala Nätverk

Josef Wilzen

2020-09-08

Outline

Intro

http://playground.tensorflow.org/
Lek runt med olika modeller här!

Gradient descent

Kostnadsfunktion:

$$h(\theta_n)$$

Gradient descent: hitta minimum på en funktion

$$\theta_{n+1} = \theta_n - \varepsilon \cdot \nabla h(\theta_n)$$

- Vi behöver gradienten (partiella derivator) $\nabla h(\theta_n)$
- Backpropagation: kedjeregeln för derivator på neurala nätverk
- Gradient descent: dyrt när vi har många obs!
- Vanligt i statistik: (oberoende obs)

$$h(\theta) = \sum_{i=1}^{N} L\left(f\left(x^{(i)}, \theta\right), y^{(i)}\right)$$

Stochastic gradient descent (SGD)

SGD:

- Dyrt att beräkna $\nabla h(\theta)$ för alla datapunkter
- Gör en väntevärdesriktig skattning av $\nabla \hat{h}(\theta)$ genom att ta ett slumpmässigt sample från data (mini-batch)
- Det finns varians i $\nabla \hat{h}(\theta)$, större batch ger mindre varians men blir dyrare att beräkna.
- Kräver många iterationer och liten learning rate (ε)
  - För liten: tar väldigt lång tid!
  - För stor: värdet på kostnadsfunktionen varierar kraftigt!
- Kräver att vi har oberoende observationer i likelihoodfunktionen.
- Väldigt vanlig inom maskininlärning
  - Speciellt inom deep learning

Stochastic gradient descent (SGD)

Learning rate ε (inlärningstakt)

- Mycket viktig parameter
- Ofta behöver vi sakta minska ε med iterationer i SGD
  - Schema (schedule) för minskningen: $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k, \ldots$
  - Gradienten kommer alltid att ha lite brus, även i en lokal minimipunkt
  - Krav för teoretisk konvergens:

$$\sum_{k=1}^{\infty} \varepsilon_k = \infty$$

$$\sum_{k=1}^{\infty} \varepsilon_k^2 < \infty$$

- **Problem:** försvinnande eller exploderande gradienter $\rightarrow$ i backproagation multiplicerar vi många derivator med varandra
  - Om derivatorna är för nära noll: Produkten blir noll
  - Om derivatorna är för stora: Produkten $\rightarrow \infty$
  - Nätverk med ReLu fungerar ofta bättre än tex sigmoid

Stochastic gradient descent (SGD)

Require: Initial parameter θ, Learning rate schedule: $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k, \ldots$

- $k \leftarrow 1$
- **while** stopping criterion not met **do**
  - Sample a minibatch of m examples from the training set $\left(x^{(1)}, \ldots, x^{(m)}\right)$, with corresponding y values.
  - Compute gradient estimate:

$$\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L\left(f\left(x^{(i)}, \theta\right), y^{(i)}\right)$$

  - Apply update:

$$\theta \leftarrow \theta - \varepsilon \cdot \hat{g}$$

  - $k \leftarrow k + 1$
- **end while**

Momentum

- SGD är ofta bra, men ibland för långsamt
- Momentum: metod att snabba upp och göra SGD mer effektiv
- Idé:
  - Ta glidande medelvärde över elementen i gradientvektorn
  - Vi vill inte ha hela "historien" med gradienter $\rightarrow$ exponentiellt avtagande glidande medelvärde
- Hyperparameter: α
  - Ju större α är i relation till ε desto mer påverkar tidigare värden på gradienten.
  - Ofta sätts α till 0.5, 0.9 eller 0.99

Stochastic gradient descent with momentum

Require: Initial parameter θ, initial velocity v, Learning rate schedule: ε

- $k \leftarrow 1$
- **while** stopping criterion not met **do**
  - Sample a minibatch of m examples from the training set $\left(x^{(1)}, \ldots, x^{(m)}\right)$, with corresponding y values.
  - Compute gradient estimate:

$$\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L\left(f\left(x^{(i)}, \theta\right), y^{(i)}\right)$$

  - Compute velocity:

$$v \leftarrow \alpha \cdot v - \varepsilon \cdot \hat{g}$$

  - Apply update:

$$\theta \leftarrow \theta + v$$

  - $k \leftarrow k + 1$
- **end while**

Momentum

Visualisering: här

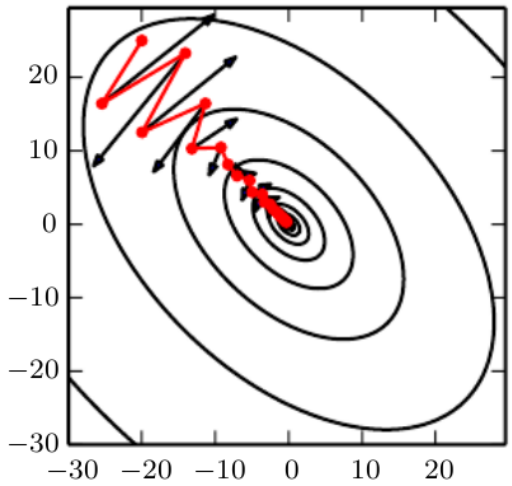


Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function. At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point. We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides. Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon. Compare also figure 4.6, which shows the behavior of gradient descent without momentum.

Från: Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016, länk

Startvärden

- Neurala nätverk: tusentals parametrar som vi behöver skatta
  - SGD är iterativt: vi börjar med några **initiala värden** på alla parametrar som vi sedan ändrar i varje iteration av algoritmen
- Startvärden: Svårt problem!
- Dåliga startvärden kan leda till:
  - SGD konvergerar inte till en lösning (lokalt optima)
  - Numeriska problem som gör att beräkningarna inte går att utföra

Startvärden

Startvärden

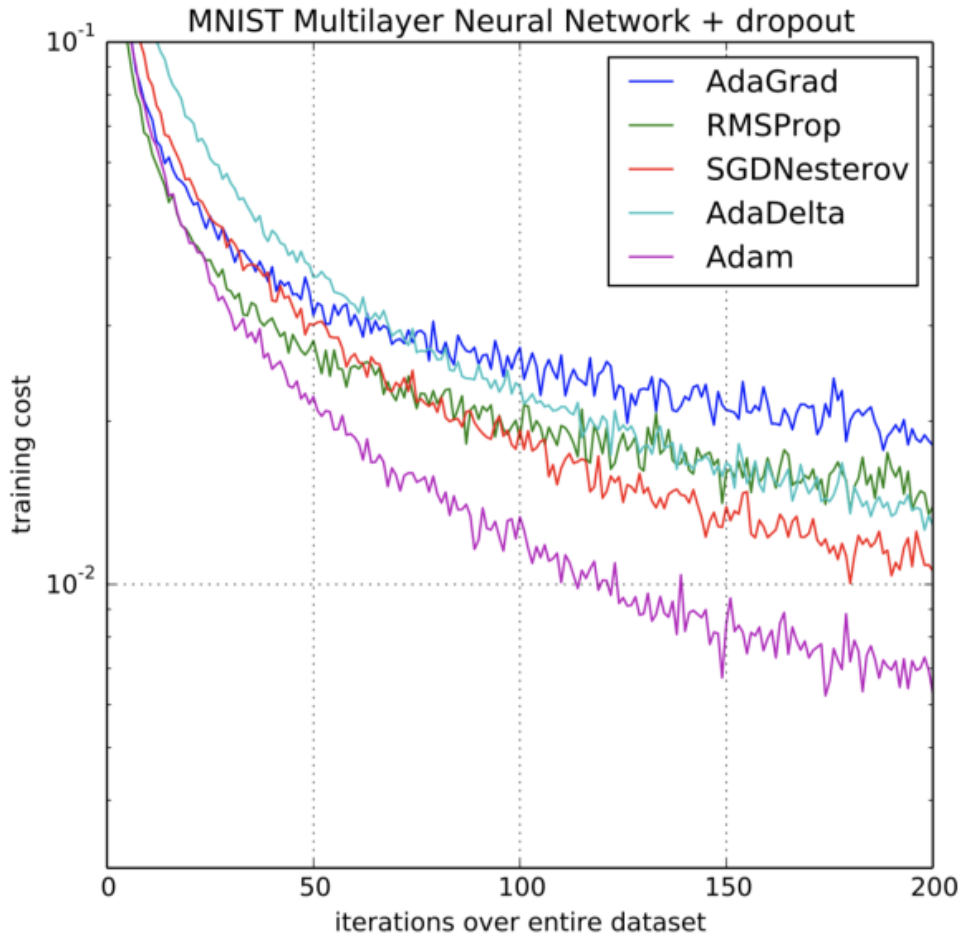
- Bryta symmetri mellen noderna: vikterna mellan två lager kan inte start på exakt samma värden
- Vikter brukar sättas till slumpmässiga tal, som är relativt nära noll
  - Uniformfördelning
  - Normalfördelning
- Bias: brukar sättas till något värde
- Hur vi specificerar startvärden kan ses som en hyperparameter
- Se dok för Keras: länk, under Initializers.
- Svårt område att forska på

Adaptiv inlärningstakt

Kan vara svårt att specificera en learning rate som passar alla våra parametrar i ett neuralt nätverk när vi använder SGD
Algoritmer med adaptiv learning rate:

- RMSProp
  - Skala om gradienten beroende på träningshistorian, gör att vi
    - tar större steg för de parametrar som har små derivator: mer flacka områden, som vi vill passera snabbare
    - tar mindre steg för de parametrar som har stora derivator: här vill vi vara mer försiktiga
    - Skapa med exponentiellt avtagande glidande medelvärde för tidigare gradientvärden (elementvis)
- Adam
  - Kan ses som en blanding av RMSProp och momentum
- **RMSProp** och **Adam** är vanliga standardval av optimeringsmetoder inom deep learning
- Båda har några hyperparametrar

Adam och RMSProp



Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Regularisering genom normstraff

Ridge regession:
Ridge= OLS + l^2-norm på β:

$$\underset{\beta}{arg\,min}\, L(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \qquad \lambda \geq 0$$

Lasso regression:
Lasso= OLS + l^1-norm på β:

$$\underset{\beta}{arg\,min}\, L(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \qquad \lambda \geq 0$$

Neurala nätverk + normstraff

Linjär regression:

- Notera: Ridge/Lasso tvingar parameterarna att vara relativt nära noll $\rightarrow$ detta gör att de funktionen de anpassar blir enklare

Vi kan använda samma idé inom deep learning/neurala nätverk!

- Genom att tvinga parameterarna att vara nära noll så tvingas nätverket att skapa en enklare funktion
- Detta minskar överanpassning

Neurala nätverk + normstraff

- Generellt:

$$\tilde{h}(\theta|X,y) = h(\theta|X,y) + \alpha\Omega(\theta)$$

  - $h(\theta|X,y)$: vanliga kostnadsfunktionen
  - $\Omega(\theta)$ straffterm baserad på vektor/matrisnorm, dvs l^2-norm eller l^1-norm.
  - α: hyperparameter, med $\alpha \geq 0$
  - Vad innebär stora värden på α? Små?

Neurala nätverk + normstraff

- Använder Frobenius normen:

$$||A||_F^2 = \sum_{i=1}^{q}\sum_{j=1}^{p} a_{ij}^2 \qquad ||A||_F^1 = \sum_{i=1}^{q}\sum_{j=1}^{p} |a_{ij}|$$

  där A är en $p \times q$ matris
- Låt W_l bara en matris med vikter mellan lager l och $l+1$
- Kostnadsfunktioner:

$$\tilde{h}(\theta|X,y) = h(\theta|X,y) + \frac{\alpha}{2}\sum_{l=1}^{h} ||W_l||_F^2$$

$$\tilde{h}(\theta|X,y) = h(\theta|X,y) + \frac{\alpha}{2}\sum_{l=1}^{h} ||W_l||_F^1$$

Neurala nätverk + normstraff

- Vi kan ha olika α för olika lager

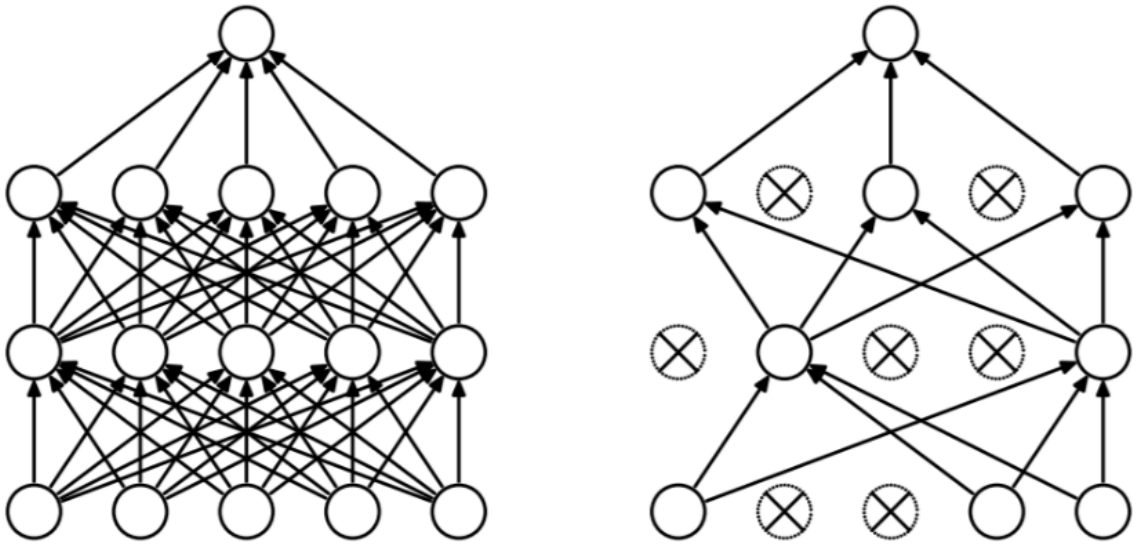
$$\frac{1}{2}\sum_{l=1}^{h}\alpha_l||W_l||_F^2$$

- $||W_l||_F^2$ kallas ibland för weight decay (tvingar vikterna att bli mindre)

Dropout

- I varje iteration när vi tränar vår modell med SGD:
  - låt varje nod vara noll med en viss sannolikhet
  - Skapa indikatorfunktion för varje nod i de lager med dropout: $I_i \sim p(I_i = 1) = r$
- Nätverket tvingas att inte "lita på" specifika kopplingar mellan lager
- Nätverket måste distribuera sin kapacitet över hela nätverket, vilket motverkar överanpassning
- När värdena på vikterna sprids ut över många kopplingar så blir de mindre
  - Liknande effekt som vid normstraff.
- Dropout används vid träning, inte vid testning
- Notera likheten med Random forest

Dropout



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

Mer regularisering

Det flera andra sätt att regularisera:

- Skaffa mer träningsdata!
- Early stopping: Stanna träningen innan maximala antal iterationer är uppnått.
  - Ofta används ett valideringsdata för att avgöra när det är dags att stanna skattningen
- Dataset Augmentation: Skapa mer artificiell data att träna på
  - Ofta i samband med bilddata
  - Speglingar, rotationer, beskärningar, addera brus mm
- Ensamplemetoder/Bagging
  - Generera flera uppsättningar med startvärden och träna flera modeller: aggregera resultatet vid testdata
- Parameter Tying and Parameter Sharing
  - Sätt en viss struktur på arkitekturen som tvingar vissa kopplingar att vara identiska/glesa mm $\rightarrow$ CNN

Specialla typer av modeller

- Residual nets: Kopplingar som hoppar över lager
  - Skip layer connections
  - Tänk: "modellera residualerna på en linjär modell med ett neuralt nätverk"
- Transfer learning:
  - "Överföra kunskap mellan olika problem/uppgifter"
  - Finns olika sorter, vanlig variant: Använda förtränade nätverk (pretrained models) på andra problem.
    - ★ Ta ett neuralt nätverk med skattade vikter som används till ett dataset/problem
    - ★ Ta modellen och starta optimieringen från förtränade vikterna på ett **annat** dataset
    - ★ Kan ses som ett avancerat sätt att specificera startvärden till optimiering
    - ★ Är ofta effektivt på mindre dataset

Faltade nätverk

Faltade nätverk: Convolutional networks, convolutional neural networks, ConvNet, CNN

- Specialla nätverk som passar för data som har en rutnätsstruktur (grid):
  - 1-D nät: tidserier
  - 2-D nät: bilder (pixlar)
  - Tänk pixlar organiserade i en matris = rutnät
  - **Mycket framgångsrik modellklass!!!!!**
  - Revolutionerade bildbehandlingen 2012
  - Lite historia här: länk
- Använder faltning mellan minst två lager i nätverket, istället för vanlig matrismultiplikation.

Faltning

- Faltning är en speciell typ av linjär operator
- Kan defineras lite olika inom olika fält

$$y(t) = \int x(\tau)\, h(t-\tau)\, d\tau = \int x(t-\tau)\, h(\tau)\, d\tau$$

$$y(t) = (x * h)(t)$$

- Kan ses som en sammanviktning av två kontinuerliga funktioner $x(t)$och $h(t)$
  - $x(t)$: insignal (input) och $h(t)$ kärnfunktion (kernel, kernel funciton) eller filter
  - $y(t)$: utsingal, inom DL: **feature map**

Faltning

- Vanlig inom statistik, matematik, singalbehandling mm
  - Används för att filtrera signaler/tidserier, tex för att ta bort brus
  - Används bla när vi vill beräkna täthetsfunktionen för summan av två oberoende slumpvariabler
  - $p(X), p(Y)$, $X + Y = Z$, vad är $p(Z)$?

Faltning

- Om vi har signaler som är samplade i diskret tid

$$y(t) = (x * h)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)\, h(t-\tau)$$

- $h(t)$ är här en vektor

Faltning

- 2D:

$$y(i,j) = (x * h)(i,j) = \sum_m \sum_n x(m,n)\, h(i-m, j-n)$$

- Kan generaliseras till högre dimensioner
  - tensor = en matris/tabell med mer än 2 dimensioner, tänk `array()` i R.
  - TensorFlow
- Vi kan använda faltning som ett sätt att koppla ihop två lager

Varför faltning?

- Glesa kopplingar
- Spatial information: närliggande pixlar relaterar till varandra mer än pixlar långt borta

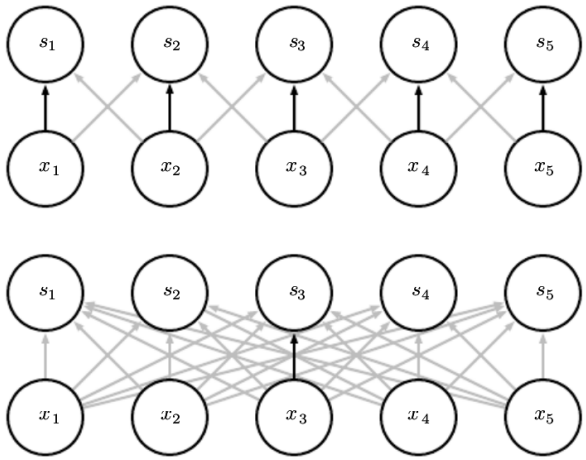
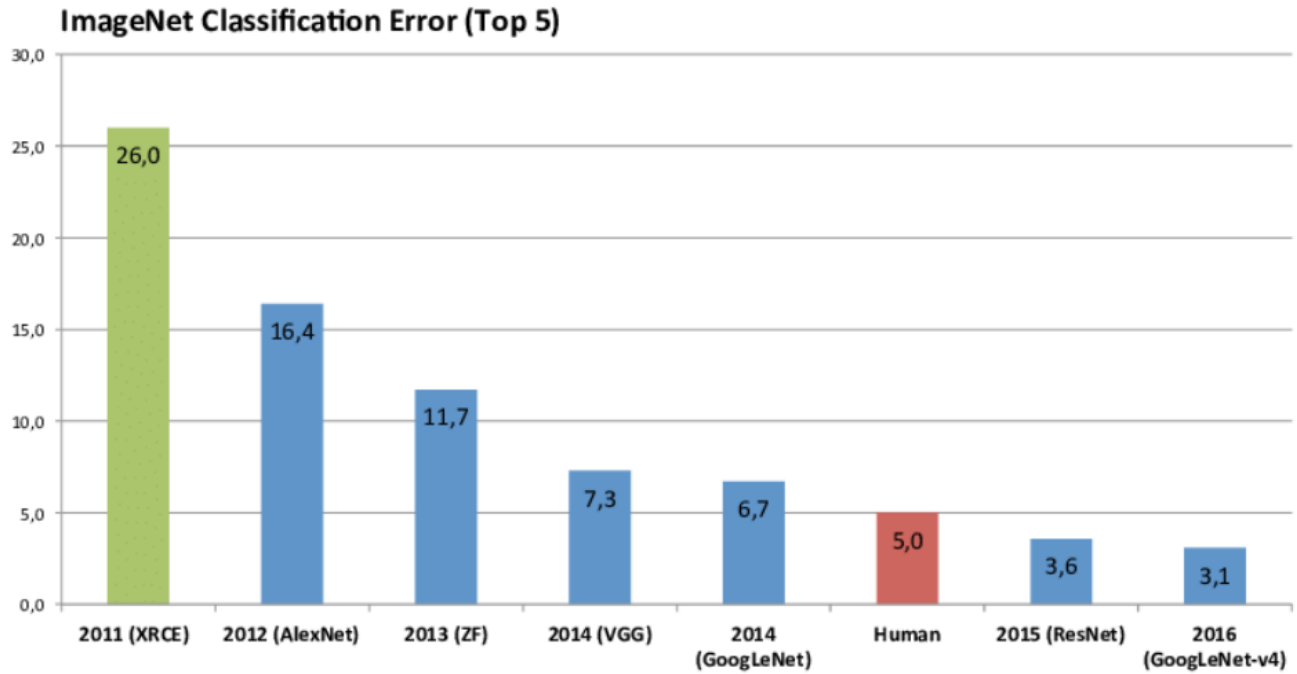


Figure 9.5: Parameter sharing. Black arrows indicate the connections that use a particular parameter in two different models. *(Top)*The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all input locations. *(Bottom)*The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing, so the parameter is used only once.

Från: Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016, länk

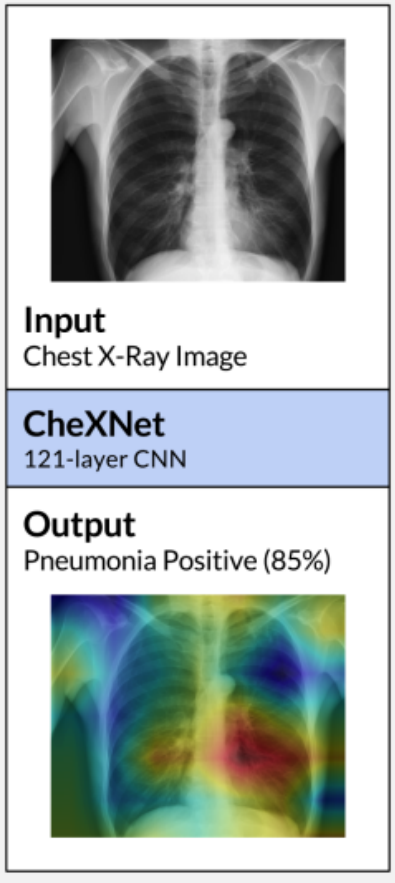
Klassificering av bilder: 2D

- ImageNet dataset / challenge, 1.2 million training images, 1000 classes
- http://www.image-net.org



- https://devopedia.org/imagenet

Klassificering av bilder: 2D



Our model, CheXNet, is a 121-layer convolutional neural network that inputs a chest X-ray image and outputs the probability of pneumonia along with a heatmap localizing the areas of the image most indicative of pneumonia.

We train CheXNet on the recently released ChestX-ray14 dataset, which contains 112,120 frontal-view chest X-ray images individually labeled with up to 14 different thoracic diseases, including pneumonia. We use dense connections and batch normalization to make the optimization of such a deep network tractable.

Rajpurkar et al. (2017). Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225.

Bildanalys/Datorseende

- Stora dataset
- Lätt att dela data
- 2D/3D data
- ImageNet
- Många förtränade modeller för 2D-data

Vad är en bild?

- Bild: 2D signal, varje pixel är ljusstyrka
- Digitala bilder vanligast
- Varje pixel har 2D koordinater (x,y)
- Varje pixel kan flera olika värden
  - Färgbilder: red, grön och blå $\rightarrow$ tre kanaler (channels)
  - Gråskala: en kanal
- Spatial autokorrelation

2D-faltning av bild

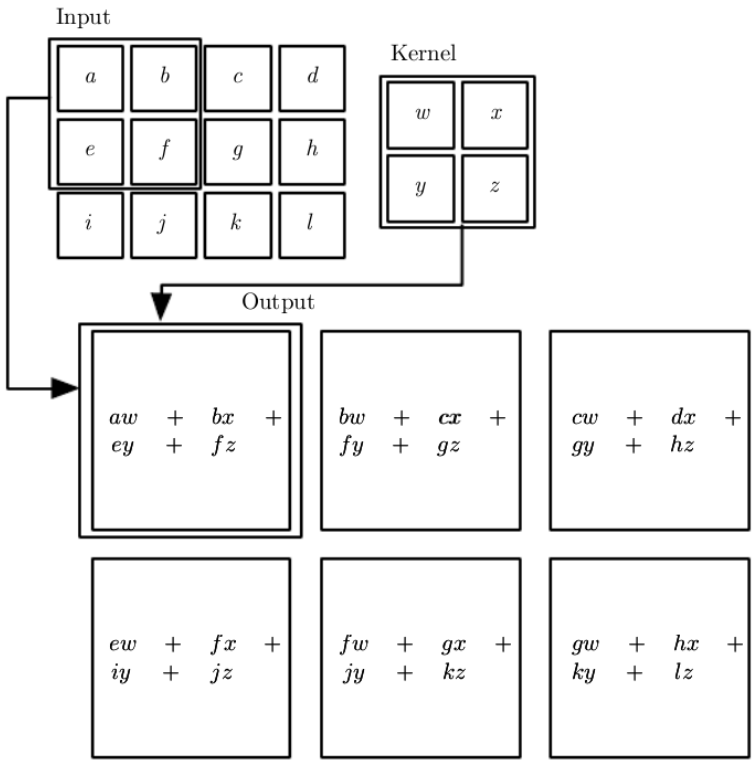


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Från: Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016, länk

Faltade lager

1. Genomför faltning enligt föregående bild
   - Nu erhålls en ny matris (bild)
2. Aktiveringsfunktion appliceras elementvis (detector stage), ofta ReLu.
3. Pooling:
   - Tar ett rektangulärt område av bilden och beräknar sammanfattade mått
     - ★ Medelvärde, maxvärde
   - Gör resultatet från faltningen invariant till små förändringar i input data

- Notera: vi vill oftast ha flera filter per lager!
  - `layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu", input_shape = c(32,32,3))`

Pooling

- Pooling används ofta för varje (eller varanan) faltningslager
- Hjälper till att reducera storleken på bilden (input) $\rightarrow$ variabelselektion
  - Detta gör att senare lager inte behöver vara så stora

Padding och Stride

- Hur hantera kanter på bilder?
  - Om vi vill att input och output ska ha samma storlek:
  - **Padding**: lägg ett eller flera lager med 0 (oftast) runt bilden
- Om vi vill spara beräkningar/parametrar:
  - **Stride**: Hoppa över pixlar när vi glider med kernelmatrisen över inputdata (i faltningen)
  - Stride=2, hoppa över varanan pixel

Padding

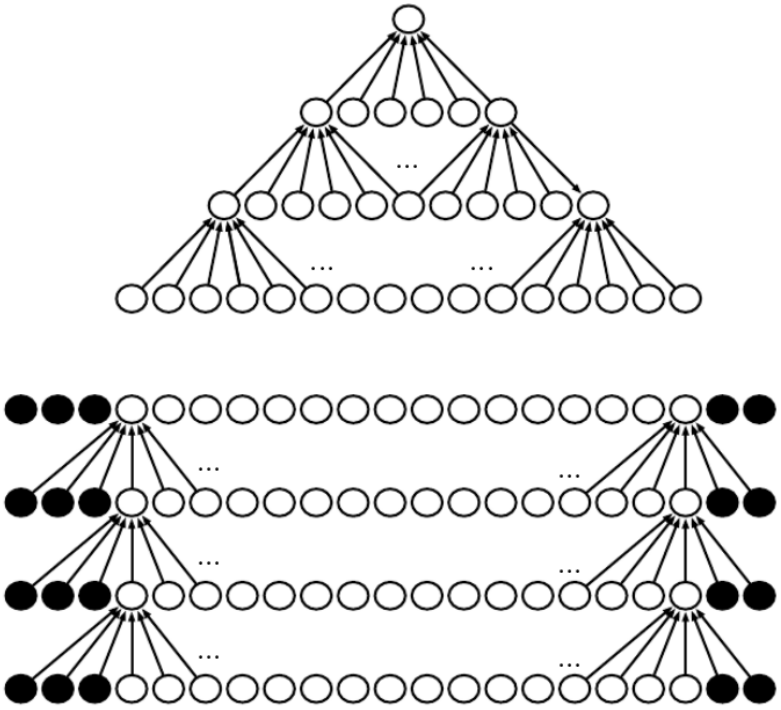


Figure 9.13: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, so only the convolution operation itself shrinks the network size. *(Top)*In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive, and some shrinking is inevitable in this kind of architecture. *(Bottom)*By adding five implicit zeros to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Från: Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016, länk

Faltning + aktiveringsfunktion + pooling

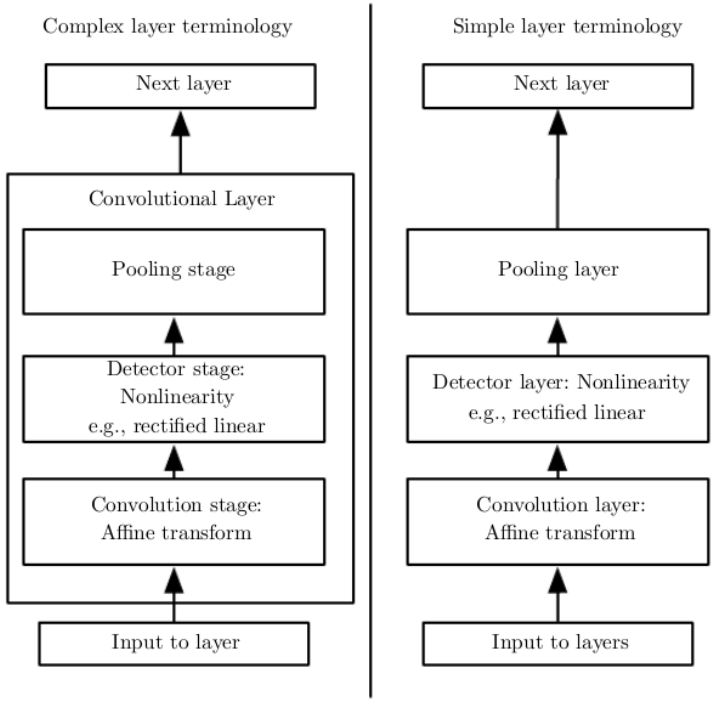


Figure 9.7: The components of a typical convolutional neural network layer. There are two commonly used sets of terminology for describing these layers. *(Left)*In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many "stages." In this terminology, there is a one-to-one mapping between kernel tensors and network layers. In this book we generally use this terminology. *(Right)*In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every "layer" has parameters.

Från: Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016, länk

Faltning + bilder = sant

- Generellt: Det är svårt att att få ut bra variabler/features från bilder
- Faltning är lämplig metod!
  - Relativt få parametrar
  - Använda lokal spatial information
  - Spara beräkningar
- Vilka filter ska vi välja?
  - Neurala nätverk to the rescue!
  - Vi lär oss alla parametrarna i ett eller flera filter med SGD
- Förr: valdes filter "för hand" för att passa olika problem $\rightarrow$ ofta svårt!

CNN ARCHITECTURE - CLASSIFICATION

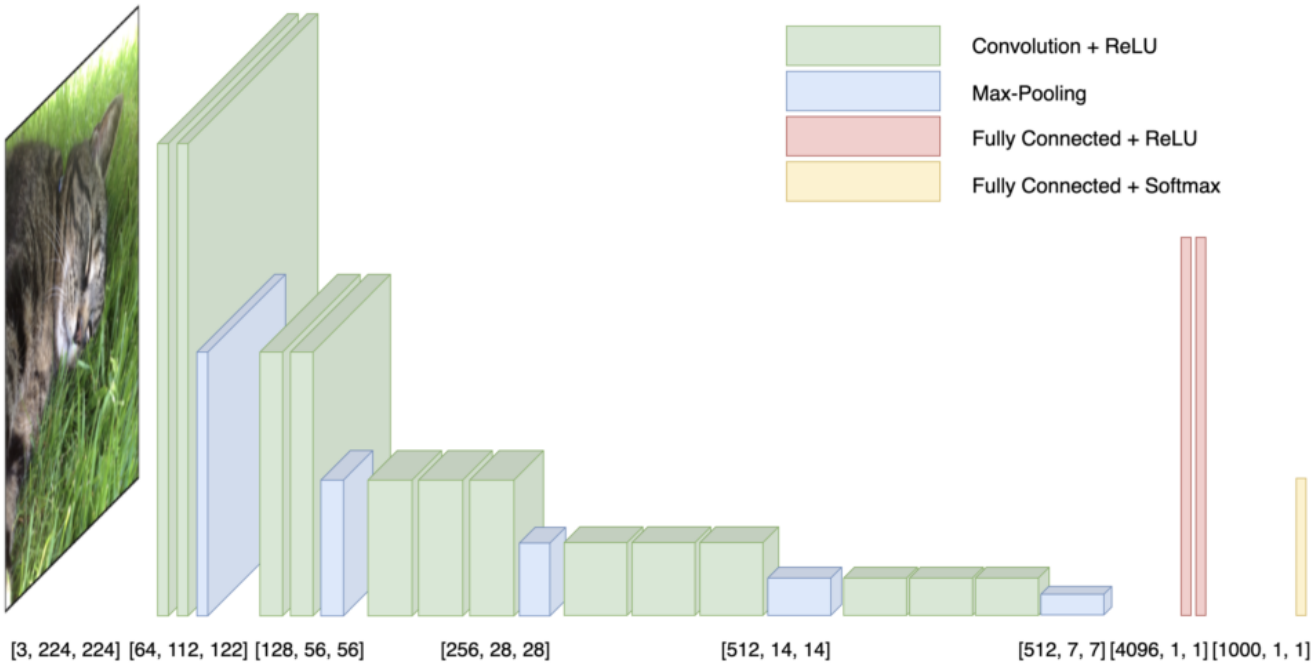


Figure 2.4: *Illustration of the data flow through the network VGG16. Data size, of the format* $[c, h, w]$*, is shown for the input image, output of each max-pooling layer, output after the first two fully connected layers, and the final network output.*

Jesper Westell, Multi-Task Learning using Road Surface Condition Classification and Road Scene Semantic Segmentation, LIU-IMT-TFK-A–19/570–SE

Avslut

- Frågor? Kommentarer?
- Kurshemsidan
- Labben