

# 732G12 Data Mining

## Föreläsning 8

---

Josef Wilzén

IDA, Linköping University, Sweden

# Dagens föreläsning

## Agenda:

- Optimering av neurala nätverk
- Regularisering
- Faltade nätverk
- Autoencoder

<http://playground.tensorflow.org>

Lek med olika modeller, se vad som händer etc.

## Projekt:

- Skapa grupper denna vecka
- Datainlämning: instruktioner kommer senare i veckan
- Deadline datainlämning: torsdag kursvecka 6.
- Projektet fortsätter sen kursvecka 7

# Optimering av neurala nätverk

Gradient decent: Hitta minimum på en funktion genom att gå dit den lutar mest!

Vill hitta

$$a^* = \arg \min_a L(a) = \sum_i L_i(f(x^{(i)}, a), y^{(i)}).$$

Löser detta genom sekvensen

$$a_{n+1} = a_n - \gamma \cdot \nabla L(a_n).$$

- Vi behöver gradienter (partiella derivator)
- Backpropagation: kedjeregeln för derivator på neurala nätverk.
- Gradient decent: dyrt när vi har många observationer!

# Stochastic gradient decent (SGD)

Det är dyrt att beräkna  $\nabla L(a_n)$  för alla datapunkter.

Gör istället en väntesvärdesriktig skattning  $\nabla \hat{L}(a_n)$  av gradienten genom att ta ett slumpmässigt sample från data (mini-batch).

- Större batch ger mindre varians i skattningen men blir dyrare att beräkna.
- Kräver fler iterationer och mindre learning rate.
- Kräver att vi har oberoende observationer.
- Funkar bra för neurala nätverk!
- En epoch är en genomgång av all träningsdata.

# Stochastic gradient decent (SGD)

Learning rate  $\gamma$ :

- Viktig parameter för att få konvergens.
- Behöver ofta ändras med antalet iterationer.
  - Schema ger hur learning rate minskar:  $\gamma_1, \gamma_2, \gamma_3, \dots$
  - Vanliga krav på schemat:

$$\sum_{k=1}^{\infty} \gamma_k = \infty \quad \sum_{k=1}^{\infty} \gamma_k^2 < \infty.$$

- Exempel  $\gamma_k = 1/k$ .

**Problem** I backpropagation multiplicerar vi massa derivator med varandra. Detta kan leda till försvinnande eller exploderande gradienter.

- Om derivatorna är nära noll så blir produkten noll.
- Om derivatorna är för stora blir produkten ännu större.
- ReLu fungerar ofta bättre än sigmoid. Finns olika sätt att hantera detta problem på.

- SGD är ofta bra, men ibland för långsamt.
- Idé: Kolla på hur gradient har varit i tidigare steg och använd detta för att ge skjuts åt algoritmen.
- Kallas för momentum.
- Hyperparameter  $\alpha$ , hur mycket av den gamla derivatan man sparar.
- Uppdatera parametrarna med  $v$  som uppdateras som

$$v = \alpha \cdot v - \gamma_k \cdot \widehat{\nabla} L$$

Vilket ger:

$$\theta_k = \theta_{k-1} + v.$$

# Momentum

Kolla på <https://distill.pub/2017/momentum/> för visualisering

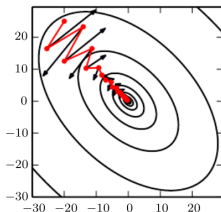


Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian 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



Neurala nätverk har tusentals parametrar som vi måste skatta.

- SGD är iterativt, vi börjar någonstans och ändrar dessa startvärden.
- Dåliga startvärden kan leda till problem.

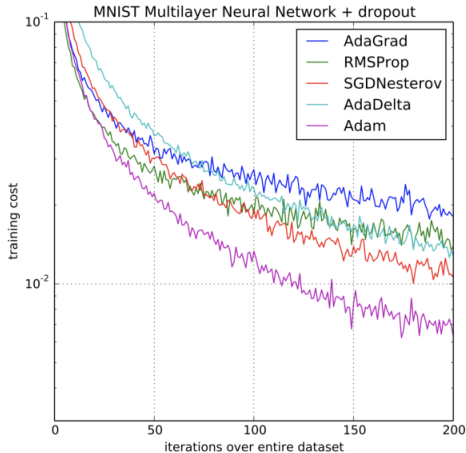
Vad kan vi göra:

- Bryta symmetri mellan noderna, vikter mellan lager kan inte ha exakt samma värden.
- Initiera vikterna slumpmässigt
  - Uniformt
  - Normalfördelat
- Bias sätts för det mesta till något värde.
- Att sätta startparametrar kan ses som en hyperparameter.
- Keras dok: se [här](#)

Kan vara svårt att få till en bra learning rate som passar alla parametrar när vi använder SGD. En idé är att anpassa learning rate för varje parameter.

- RMSProp
  - Skala om gradienten beroende på träningshistorik så att vi
    - tar större steg för de parametrar med små derivator.
    - tar mindre steg för de parametrar med stora derivator.
    - Skala med exponentiellt avtagande glidande medelvärden för tidigare gradienter.
- Adam
  - Kan ses som en blandning av RMSProp och momentum.
- Finns fler varianter, RMSProp och Adam är vanliga standardval för neurala nätverk och finns i de flesta paket.
- Finns hyperparametrar.

# Adam och RMSProp



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

## Batch normalization:

- Metod för att göra träningen av neurala nätverk stabilare och snabbare.
- Funkar bra empiriskt, teorin om varför det funkar är inte klarlagd
- Idén är standardisera alla inputs till ett lager
- Anta att vi använder någon variant av SGD
  - standardiseringen sker per nod i det aktuella lagret
  - standardiseringen sker över den aktuella mini-batchen i ett träningssteg
  - vi beräknar medelvärde och standardavvikelse baserat på aktuell mini-batch

# Batch normalization

Låt  $z_i$  vara ett set av noder i ett lager. Beräkna medelvärde och varians per nod över aktuell mini-batch:

$$\mu_i = \frac{1}{m} \sum_{j=1}^m x_{ij} \quad \text{och} \quad \sigma_i^2 = \frac{1}{m} \sum_{j=1}^m (x_{ij} - \mu_i)^2$$

Normaliserar:

$$z_{i,norm} = \frac{z_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

Skala om  $z_{i,norm}$  med parametarar:

$$\tilde{z}_i = \gamma_i \cdot z_{i,norm} + \beta_i$$

Applicera aktiveringsfunktionen på  $\tilde{z}_i$ .

$\gamma_i$  och  $\beta_i$  är parametarar som vi skattar tillsammans med alla parametrar i nätverket.

Vid prediktioner så används medelvärde och varians beräknat på hela datasetet.

Vad händer med kostandsfunktionen när vi kör Ridge/Lasso?

Vad händer med parametrarna i modellen när vi kör Ridge/Lasso?

# Regularisering genom normstraff

Ridge = OLS +  $\ell^2$ -norm på  $\beta$ :

$$\arg \min_{\beta} L(\beta) = \sum_{i=1}^n (y_i - \hat{y})^2 + \lambda \sum_{j=1}^p \beta_j^2, \quad \lambda \geq 0.$$

Lasso = OLS +  $\ell^1$ -norm på  $\beta$ :

$$\arg \min_{\beta} L(\beta) = \sum_{i=1}^n (y_i - \hat{y})^2 + \lambda \sum_{j=1}^p |\beta_j|, \quad \lambda \geq 0.$$

# Regularisering genom normstraff

Vi kan göra samma sak med neurala nätverk.

Regularisering "tvingar" parametrarna att bli små (nära noll) vilket skapar en enklare funktion och minskar överanpassning.

Vilken norm ska vi använda oss av?



Generellt:

$$\tilde{L}(\theta) = L(\theta) + \lambda\Omega(\theta),$$

$L(\theta)$  kostandsfunktionen

$\Omega(\theta)$  straffterm

$\lambda$  Hyperparameter med  $\lambda \geq 0$ .

Vad händer med  $\tilde{L}(\theta)$  då vi ändrar  $\lambda$ ?

- Använd Frobenius normen:

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

där  $\mathbf{A}$  är en  $p \times q$  matris.

- Låt  $\mathbf{W}_l$  vara viktmatrisen mellan lager  $l$  och  $l+1$ , då får vi kostnadsfunktioner

$$\tilde{L}(\theta) = L(\theta) + \frac{\lambda}{2} \sum_{l=1}^h \|\mathbf{W}_l\|_F^2$$

$$\tilde{L}(\theta) = L(\theta) + \frac{\lambda}{2} \sum_{l=1}^h \|\mathbf{W}_l\|_F^1$$

- Vi kan ha olika  $\lambda$  för olika lager,

$$\frac{1}{2} \sum_{l=1}^h \lambda_l \|\mathbf{w}_l\|_F^2.$$

- Kallas ibland för weight decay (tvingar vikterna att bli mindre).

- I varje iteration när vi tränar vår modell med SGD:
  - Låt varje nod vara noll med en viss sannolikhet.
- Nätverket kan inte "lita på" några specifika kopplingar.
- Nätverket måste distribuera sin kapacitet över hela nätverket, 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 testning.
- Notera likheten med Random forest.

# Dropout

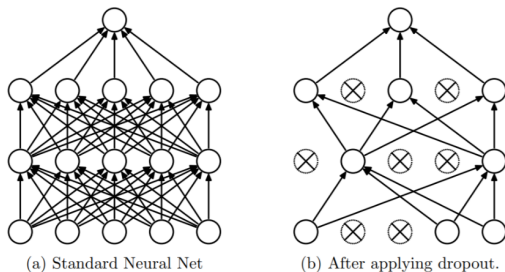


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

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 finns flera andra sätt att regularisera

- Batch normalization: ger viss regularisering
- Skaffa mer träningsdata!
- Stoppa tidigt, använd valideringsdata för att avgöra när det är dags att stanna skattningen.
- Förstärk data (dataset augmentation), skapa mer artificiell data att träna på.
  - Vanligt när det kommer till bilder.
  - Spegla, rotera, beskära, addera brus m.m.
- Ensemblemetoder
  - Generera fler uppsättningar med startvärden och träna flera modeller: aggregera resultatet vid prediktioner.
- Parameter Tying och Parameter Sharing
  - Ha en struktur på modellen som tvingar vissa kopplingar att vara identiska eller glesa m.m.
  - T.ex. CNN, RNN

# Speciella typer av modeller

- Residual nets: Kopplingar som hoppar över ett eller flera lager.
  - Skip layer connections.
  - Tänk: "Modellera residualerna på en linjär modell med ett neuralt nätverk".
- Transfer learning:
  - "Överför kunskap mellan olika problem/uppgifter".
  - Finns olika sorter, vanlig variant: Använd förtränade nätverk på andra problem.
    - Ta ett neuralt nätverk som redan är anpassat för ett problem/dataset.
    - Starta optimering för ett annat dataset med start i de redan tränade parametrarna.
    - Kan ses som ett avancerat sätt att välja startvärden.
    - Ofta effektivt på mindre dataset.

Faltade nätverk, benämns ofta CNN (Convolutional neural networks)

- Speciella nätverk som passar för data som har en "rutnätsstruktur" (grid):
  - 1-D nät: tidsserier
  - 2-D när: bilder
  - Tänk pixlar organiserade i en matris.
  - **Används mycket framgångsrikt!**
  - [Lite historia](#)
- Använder faltning mellan (minst två) lager i nätverket istället för vanliga matrimultiplikation.



- Faltning är en speciell typ av linjär operator.
- Kan defineras på olika sätt, kontinuerliga funktioner:

$$y_t = \int x_\tau h(t - \tau) d\tau = \int x(t - \tau) h(\tau) d\tau$$
$$y(t) = (x * h)(t)$$

- Kan ses som en sammanviktnig av två kontinuerliga funktioner
  - $x(t)$  är vår insignal (input).
  - $h(t)$  är vår kärnfunktion (kernel) eller filter.
  - $y(t)$  är vår utsignal, kan kallas feature map i DL.

Faltning är en vanlig operation inom statistik, matematik, signalbehandling, reglerteknik m.m.

- Används för att filtrera signaler/tidsserier, t.ex. för att ta bort brus.
- Används när vi vill beräkna täthetsfunktionen för **summan av två oberoende slumpvariabler**.
  - Givet  $p(X)$ ,  $p(Y)$ , låt  $Z = X + Y$ , hur beräkna  $p(Z)$ ?

- Om vi har diskret tid

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

$h$  blir här en vektor med värden (observera index i vektorn).

- I två dimensioner har vi

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

- Kan generaliseras till flera dimensioner.
- Vi kan använda faltning som ett sätt att koppla ihop två lager i vårt neurala nätverk.

# Varför faltning?

- Faltning ger glesa kopplingar.
- Spatial information: närliggande pixlar relaterar till varandra mer än pixlar långt borta.

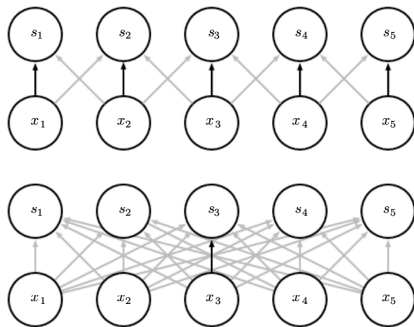
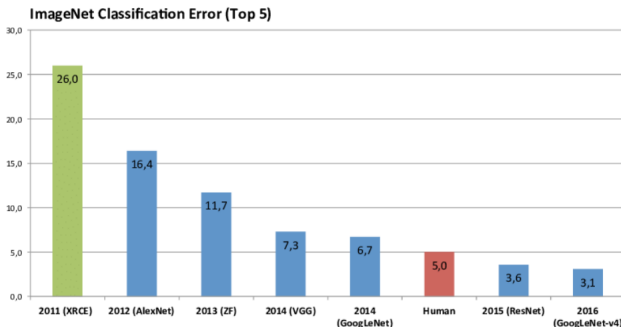


Figure 9.5: Parameter sharing. Black arrows indicate the connections that use a particular parameter in two different models. (Top) The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all 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.

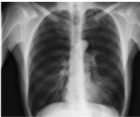
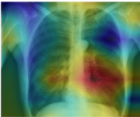
# Exempel: Klassificering av bilder: 2D

- ImageNet dataset / challenge. 1.2 miljoner bilder som träningsdata, 1000 klasser.
- <http://www.image-net.org/>



- <https://devopedia.org/imagenet>

# Exempel: Klassificering av bilder: 2D


<b>Input</b> Chest X-Ray Image
<b>CheXNet</b> 121-layer CNN
<b>Output</b> Pneumonia Positive (85%)


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.

# Vad är en bild?

- Bild: En 2D signal, varje pixel är ljusstyrka.
- Varje pixel har 2D koordinater  $(x, y)$
- Varje pixel kan ha flera olika värden.
  - Färgbilder, t.ex. röd grön och blå ger tre kanaler (channels).
  - Gråskala, en kanal.
- Spatial autokorrelation.



# 2D-faltning av bild

## Visualisering Förklaringar och räkneexempel

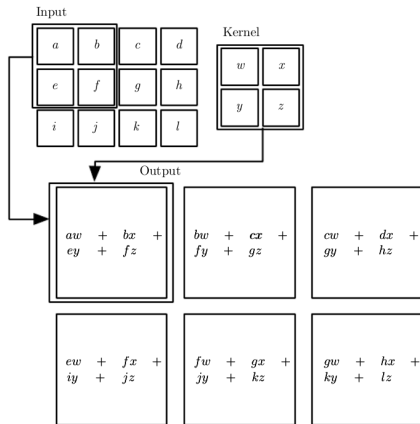


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called “valid” convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by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

- Genomför faltning enligt föregående bild.
  - Vi erhåller nu en ny matris (bild).
- Aktiveringsfunktionen appliceras elementvis.
- Pooling:
  - Tar ett rektangulärt område av bilden och beräkna sammanfattade mått.
  - Gör resultatet från faltning invariant mot små förändringar i input data.
- Notera att vi ofta vill ha fler filter per lager.

- Pooling används ofta för vajre (eller vartannat) faltningslager.
- Hjälper till att reducera storleken på bilden.
  - Betyder att senare lager inte behöver vara så stora.

- Vid faltning av bild blir resultatet mindre.
  - **Padding** lägg till ett eller flera lager med 0 runt bilden.
  - Rätt padding gör att output har samma storlek som input.
- Om vi vill spara beräkningar/parametrar.
  - **Stride** Hoppas över pixlar när vi glider med kernelmatrisen i faltningen.
  - $\text{Stride} = 2$ , hoppar över varannan pixel.

# Padding

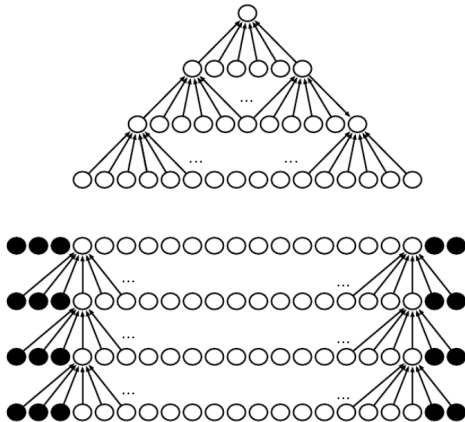


Figure 9.13: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, so only the convolution operation itself shrinks the network size. (*Top*) In this convolutional network, we do not use any 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 even 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.

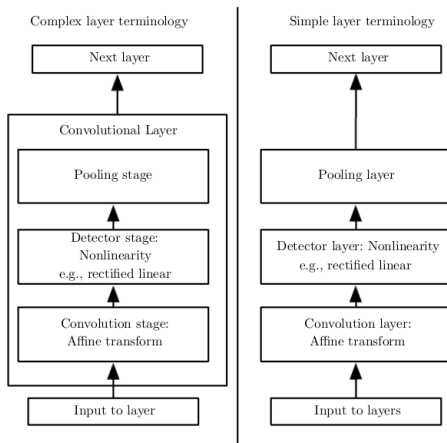
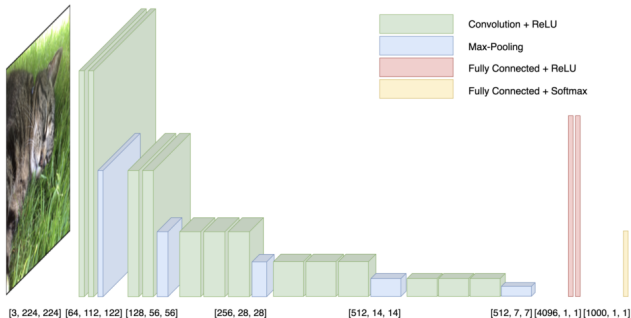


Figure 9.7: The components of a typical convolutional neural network layer. There are two commonly used sets of terminology for describing these layers. *(Left)* In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many “stages.” 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.

- Generellt: Det är svårt att få ut bra variabler/features från bilder.
- Faltning är en lämplig metod!
  - Relativt få parametrar.
  - Använder lokal spatial information.
  - Sparar beräkningar.
- Vilka filter ska vi välja?
  - → Neurala nätverk till undsättning!
  - Vi lär oss parametrarna i ett eller flera filter med SGD (RMSPProp, Adam etc.)
- Förr valdes filter "för hand" för att passa olika problem.

# CNN Arkitektur för Klassificering



**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

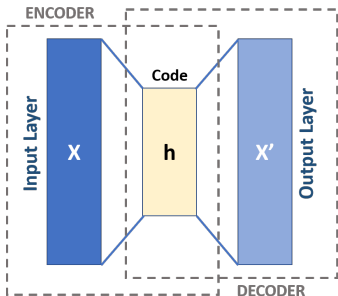


# Autoencoder

## Autoencoder:

- Målet är att lära sig en effektiv kod/representation av en dataset  $X$ .
- Exempel på **oövervakad inlärning**
- Oftast så har koden/representation lägre dimension än  $X$ .
- Nätverk med två delar: encoder och decoder

$$X \rightarrow \text{encoder} \rightarrow Z/\text{code} \rightarrow \text{decoder} \rightarrow X'$$

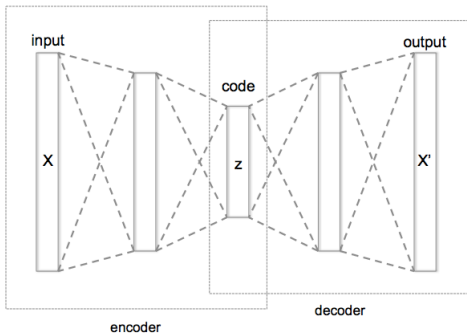


# Autoencoder

Vanligt att använda djupa nätverk: encodern och decodern har noder i flera lager.

Antalet noder brukar minskas successivt tills kod-lagret, för sen successivt ökas till det sista lagret. Antalet noder i kod-lagret är en viktig hyperparameter.

Input och output är lika stora.



# Autoencoder: Träning

- Encoder:  $E_\phi : X \rightarrow Z$
- Decoder:  $D_\theta : Z \rightarrow X$
- Anpassade värden:  $X' = D_\theta(E_\phi(X))$
- $D_\phi()$  och  $E_\theta()$ : kan vara MLP, CNN etc

Låt  $d(x, x')$  vara ett avståndsmått, kostnadsfunktion:

$$L(\theta, \phi) = \mathbb{E}[d(x, x')] = \mathbb{E}[d(x, D_\theta(E_\phi(X)))]$$

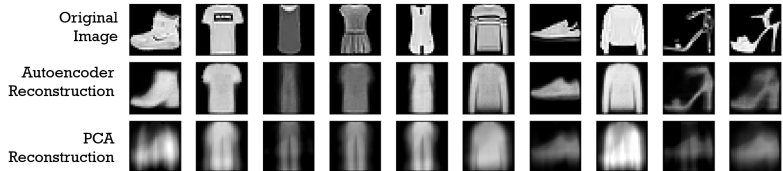
Om vi använder Euklidiskt avstånd och MSE som kostnadsfunktion:

$$\arg \min_{\theta, \phi} L(\theta, \phi) = \frac{1}{N} \sum_{i=1}^N \|x_i - D_\theta(E_\phi(X))\|_2^2$$

Finns olika varianter

- Icke-linjär variabelreduktion/transformation
  - Notera: Om vi har en linjär autoencoder så är det likt PCA
- Sparse autoencoder: koden tvingas att vara nära noll för de flesta noder
- Denoising autoencoder - brusreducering: tränar på ett  $X$  där man lagt till brus och försöker återskapa  $X$  utan brus
- Variational autoencoder (VAE): en generativ modell där koden som man lär sig är en sannolikhetsfördelning  $\rightarrow$  lär sig att sampla från fördelningen för  $X$

# Autoencoder: exempel



“Reconstruction of 28x28pixel images by an Autoencoder with a code size of two (two-units hidden layer) and the reconstruction from the first two Principal Components of PCA. Images come from the Fashion MNIST dataset.” - [källa](#)