1 section

1.1 — subsection

1.1.1 — subsubsection

– some intermediate title

sample text blabla

$math : f \to X, x = 43, \forall x.\lambda z$

comment

TODO: todo

code

Table:

c1	c2	c3
c4	c5	c6

List:

+ pro
– con

Tighcenter:

Text in the center

D.Definition XY here comes the definition
T.Theorem XY here comes a theorem
L.Lemma XY here comes the lemma
Ex.Example XY here comes example
Alg.Algo XY here comes algo

Note: here comes a Note
Important: Here comes something important
Proof: Here comes a proof
Intuition: Here comes some intuition
Raised rule:

2 Maths

test test $\{test\}^2$ okok

test test $\{asfa \mid whenever\}^2$ okok

test test $\{test\}^2$ okok

test test $\{asfa \mid whenever\}^2$ okok

3 Introduction

3.1 — Week 1-2: Deep Learning Basics

- MLP
- Fully Connected Networks
- Data, tasks, loss functions
- Backprop
- Activation functions

3.2 — Week 3-4: CNNs, RNNs, co.

- CNNs
- RNNs
- LSTM/GRU/BPTT
- Fully convolutional Networks

3.3 — Week 5-9: Generative Modeling

- Latent variable models
- Implicit Models
- Autoregressive models
- Normalizing FLows/ Invertible Networks

3.4 — Week 10-12: DL 4 CV

- Problems and tasks in human centric computer vision
- DL architectures for CV
- Human body and hand models
- Implicit representations

3.5 — Week 13: Deep RL

Perceptron: `while` $\exists x. w^T x > 0 \neq y$ `do` $w = w + \eta(y - \widehat{y})x$
Note: If the data is separable, the algorithm converges in finite time
ILP: $\forall l \quad x^{(l)} = \sigma((w^{(l)})^T x^{(l-1)} + b^{(l)}), f(x; w, b) = x^{(L)}$

Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$, $\nabla : \sigma(x) \cdot (1 - \sigma(x))$

Softmax: $softmax(x_i) = \exp(x_i) / \sum_j \exp(x_j)$
Requirements: 1) Output must be positive
2) output must be between $[0, 1]$
3) sum of all outputs must be 1, e.g. $\sum_i^M softmax(x_i) = 1$

TODO: Linear activation function (Week 2, Page 28)

MLE: Maximise $\log L(\theta) = \log \prod p(x_i|\theta) = \sum \log p(x_i|\theta)$
1) Write down probability distribution
2) Decompose into per sample probability
3) Minimize negative log likelihood

CE loss: $-\frac{1}{N} \sum y_i \log(\sigma(w^T x_i)) + (1 - y_i) \log(1 - \sigma(w^T x_i))$
Note: Cross-entropy loss is a maximum likelihood estimator We assume the y's to be bernoulli distributed, from there we maximise the weights over the probability $P(D|w) = \prod^N \sigma(w^T x_i)^{y_i} (1 - \sigma(w^T x_i))^{1-y_i}$. Taking the negative log-likelihood results in the given loss.

Universal Approximation: $\exists g(x) = \sum v_i \sigma(w_i^T x + b_i) \approx f(x)$ and $|g(x) - f(x)| < \epsilon$
Note: $\sigma : \mathbb{R} \to \mathbb{R}$ must be non-constant, bounded and continous
Note: $f \in [0, 1]^m$, e.g. must be in the m-dimensional hypercube

4 Deep Learning

5 Convolutional Neural Network

Goal: Good classification performance.
Trade-off: Specifity vs. invariance
To achieve this, we need to trade off specifity and invariance (created by affine transformations of an object and different lightings).
The Impact (?) is the generalization ability of the model.

Receptive Field: Areas triggering firing of sensory neurons
Note: Areas can be on the Retina, Skin, Tounge, etc.
Note: Usually divided into excitatory and inhibitory regions.
From Hubel and Wiesels experiments we learned that stimulus covering the whole cat retina, most neurons didn't fire, since the excitatory and inhibitory stimulus canceled each other out. The light must fall on specific regions to excite, forming specific excitement *patterns*.
Direction and alignment of the light affected the firing, and was used to figure out the alignment of the receptive field.

Hierarchy: Simple cell respond to simple 0-1 input (noisy). Complex cells, connected to multiple simple cells, form more complex patterns, more resistant to invariance.
Note: Size of receptive fields tends to get larger, the more complex the cell becomes.
Sensory receptors connected to cell in the brain form the receptive field of that cell. This receptive field can be very noisy in simple cells.
The connection of such simple cells to complex cells can form ${}^{\omega}$mplex triggering patterns, f.e. a line in a certain direction.
This hierarchy can be extended to hypercomplex shapes.

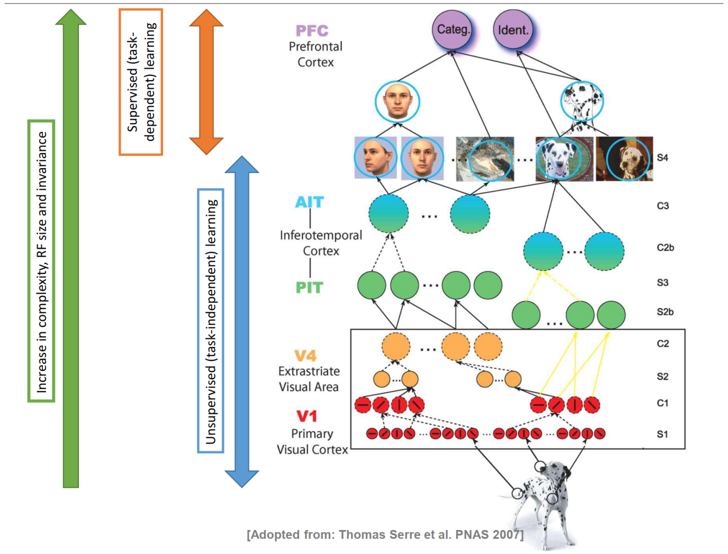


Figure 1: Complexity, unsupervised and supervised learning hierarchy

HMAX: S: $y = \exp(-\frac{1}{2\sigma^2}\sum^{n_{S_k}}(w_j - x_j)^2)$, C: $y = \max_{n_{C_k}} y_j$
x is the input to the cell, w is the weight of the cell. sigma tells us about the size of the receptive fields.

Conv: Modify pixel by some function of surrounding pixels
Note: Any linear, shift-equivariant transform can be written as convolution.
Linear: $T(\alpha u + \beta v) = \alpha T(u) + \beta T(v)$
Invariant: $T(f(u)) = T(u)$
Note: We want this in classification, b.c. a cat in the middle of the picture should still be classified as cat if it is shifted or roted
Equivariant: $T(f(u)) = f(T(u))$
Note: In Edge detection very important, if we shift the edge in the image, we also want the response to shift the same way

Lin. Filtering: $I'(i,j) = \sum_{(m,n)\in N(i,j)} K(m,n)I(i+m, j+n)$
Note: I is the image, K is the Kernel, N(i,j) is the neighbourhood of a pixel.
Shift-invariance: The Kernel is usually parameterized as K(i,j,m,n), e.g. the weights of the Kernel depend on the location in the image. Removing i,j-dependence makes the kernel shift invariant.
1

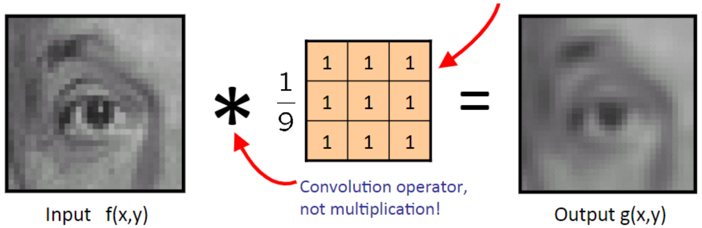


Figure 2: Linear filtering is applied with the convolution operator to compute a new image from the neigbourhood of each pixel of the old image

Correlation: $I'(i,j) = \sum_{m=-k}^{k}\sum_{n=-k}^{k} K(m,n)I(i+m, j+n)$

Conv: $I'(i,j) = \sum_{m=-k}^{k}\sum_{n=-k}^{k} K(m,n)I(i-m, j-n)$
Alternative representation would be
$I'(i,j) = \sum_{m=-k}^{k}\sum_{n=-k}^{k} K(-m,-n)I(i+m, j+n)$
Note: Convolution can be done via matrix multiplication
It acts as a point-spread function, what we want is the weights of this kernel
TODO: Ask how good we must know this, e.g. perform it ourselves?
∇: Conv. with kernel [-1, 1], $\frac{\delta f}{\delta x} \approx \frac{f(x_{n+1},y) - f(x_n,y)}{\Delta x}$
Correlation = Convolution: iff $K(i,j) = K(-i,-j)$
Convolutional Layer: $w^T z^{(l-1)} + b$ (+ activation function)
I' dim: $\frac{I_{height} + 2\cdot\text{padding} - \text{dilation}\cdot(K_{height}-1) - 1}{\text{stride}} + 1$
Note: If we have 6 filters, the produced "new image" would be of size 28x28x6

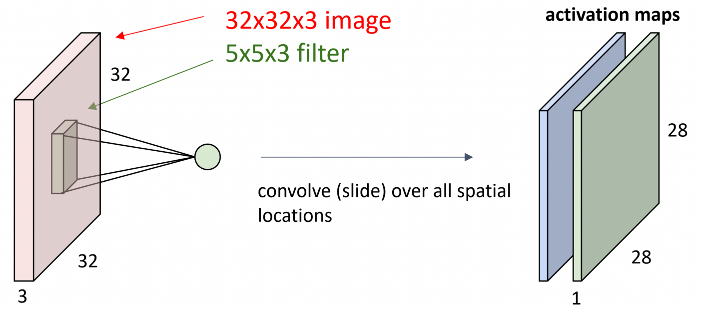


Figure 3: Visualisation applying filters to an image

TODO: Do convolution via matrix on your own!
Weight sharing: Same feature detector on whole image and ↓ weights
Note: This makes the feature detector robust agains affine transformations, as it detects the feature in the whole image once trained
Stride: used to reduce size, replaces pooling layers
Dillation: add "holes" to filter, fast increase of rec. field
It is easy to integrate the global context like this

CNN-fwd: $z_{i,j}^{(l)} = w^{(l)} * z^{(l-1)} + b = (\sum_{m,n} w_{m,n}^{(l)} z_{i-m,j-n}^{(l-1)}) + b$

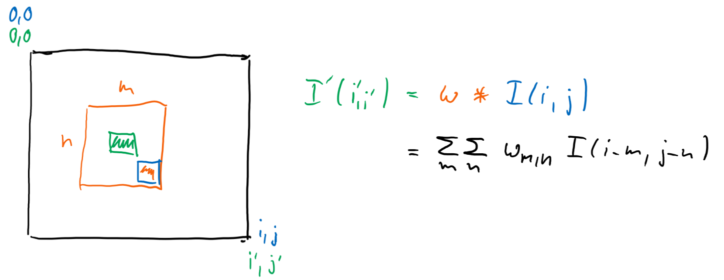


Figure 4: Visualisation forward pass

CNN-bwd (z): $\delta_{i,j}^{(l-1)} = \frac{\delta C}{\delta z_{i,j}^{(l-1)}} = \sum_{i',j'} \frac{\delta C}{\delta z_{i',j'}^{(l)}} \frac{\delta z_{i',j'}^{(l)}}{\delta z_{i,j}^{(l-1)}}$
$= \sum_{i',j'} \delta_{i',j'}^{(l)} w_{i'-i,j'-j}^{(l)} = \delta_z^l * ROT_{180}(w^{(l)})$
If we take the derivative of $\delta z_{i',j'}^{(l)}$, only certain weight terms stick, the rest is going to be zero.
Note: Backward path is just a convolution with the flipped kernel

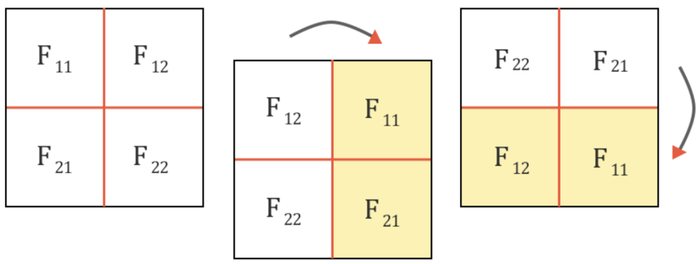


Figure 5: Flipped convolution operator

TODO: Do calculation yourself to see why it is flipped, or better, understand the relation to the "standart" convolution

CNN-bwd (w): $\delta_w^{(l)} * ROT_{180}(z^{(l-1)})$
QUESTION: Is this going to be added or substracted to the original image?
Note: Each layer holds more complex "patterns", this resembles the neuron complexity hierarchy we have seen in nature.

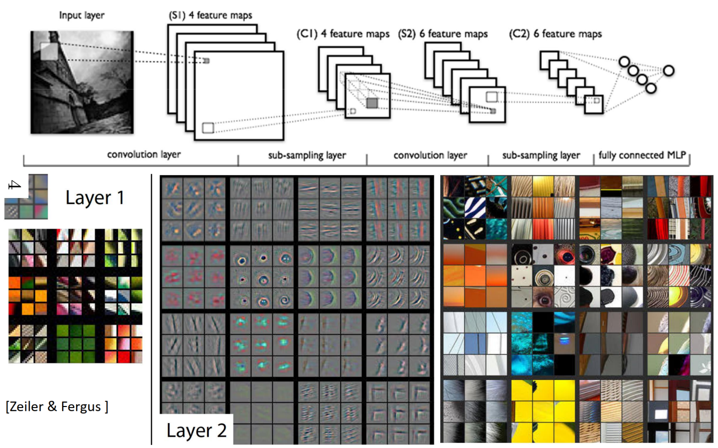


Figure 6: Visualisation of output of cnn layers

Max Pooling: $m \times m$ Filter with m stride, take max
Reduces size of each activation layer, downsampling. This helps in extending the activation region of downstream pixels, as we are putting more info in a smaller region. It also helps in reducing the noise in pictures.
There are other pooling strategies that can be applied, but with the MAX-pooling, we gain robustness to local changes. For example, if a digit rotates a bit, the region probably still has the same max pixel and thus is robust against rotations.

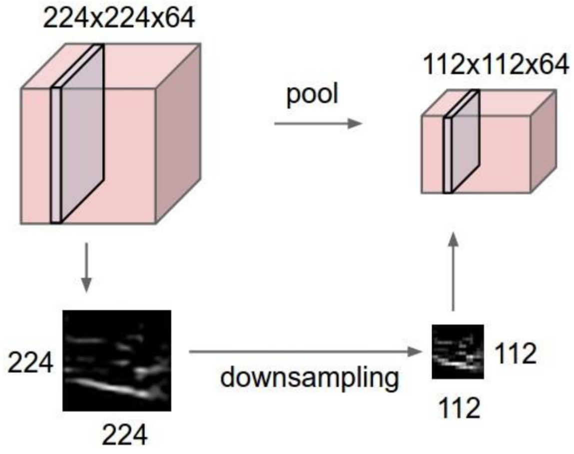


Figure 7: Visualisation of pooling layer

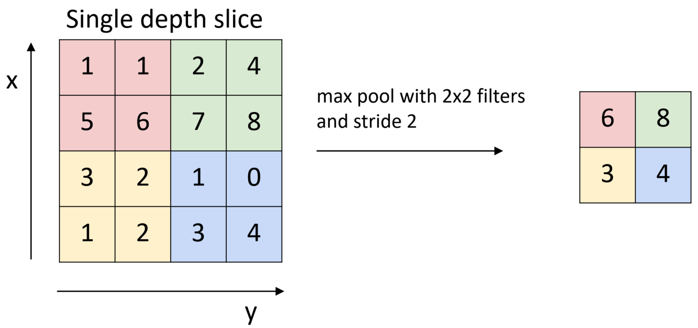


Figure 8: Visualisation of max-pooling layer

fwd: $z^{(l)} = \max\{z_i^{(l-1)}\}$ **bwd:** $\frac{\delta z^{(l)}}{\delta z_i^{(l-1)}} = 1\text{if } i^* = \max\{z_i^{(l-1)}\}$
Only one contributing pixel gets a gradient, all others have gradient zero.

5.1 — Evolution of architectures

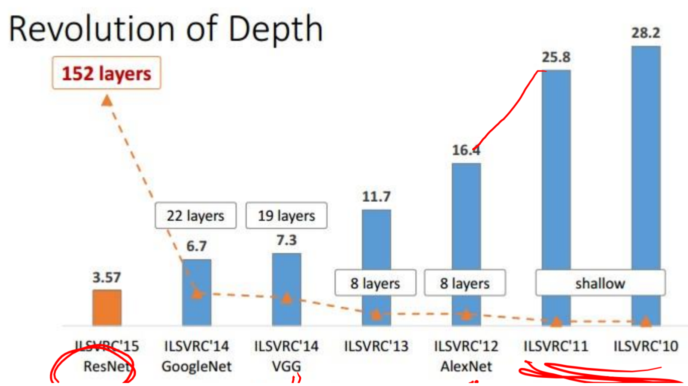


Figure 9: Evolution of NN architecture

5.2 — VGG vs. AlexNet

less kernels/filters (↓ params), more layers (↑ perceptive field)
Larger receptive field means that the net can respond to patterns that are larger spread apart.
Each large filter, f.e. 11x11, can be represented using multiple 3x3 filters. The number of parameters for one 11x11 filter is 121, which is equal to using 5 3x3 filters with 45 parameters.

5.3 — GoogleNet

More layers, removed fully connected layer on the top.
Inception: use 1x1 conv. layers to reduce layer depth
Because of small kernels and deep networks, the number of channels (depth) is going to be huge. Inception modules downsamples the activation maps, reducing the number of channels by convolution, and thus reducing the number of parameters.
Note: In addition, GoogLeNet uses auxiliary classification heads throughout the CNN, to make sure the gradients don't dry out.

5.4 — ResNet

Residual-Con: Skips weight layers with residual connections.
Note: A deep network should at least perform as good as a shallow one (set all additional layers to identity).
This approach lead to the believe that failing to perform is an optimisation issue, not a design issue.
ResNet one (drastic) downsampling and then residual layers.
The "Ultra-deep" version is not used often, in general ResNet18 is used.

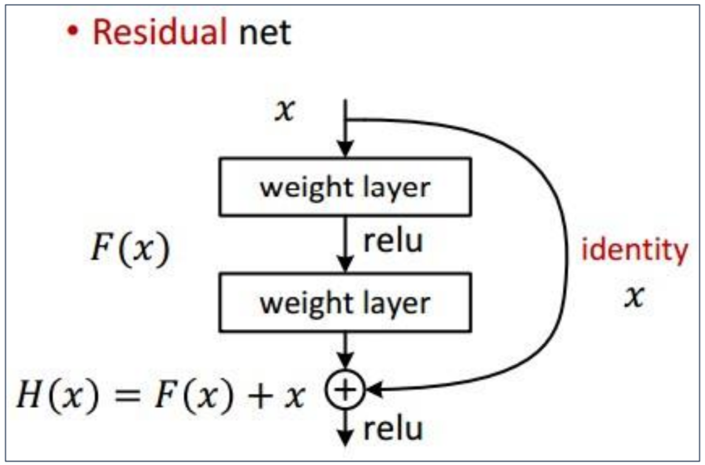


Figure 10: Evolution of NN architecture

6 Fully Convolutional Neural Network

The main difference to classical CNN is that we don't want to only work with fixed sized images as input. This limitation comes from the fully connected layers in the end.

Goal: Pixel-to-pixel classification By removing the size constraints of the input pictures allows us to classify pixels by using their local neighbourhood.

Drawback: We don't get the information of the whole context.

The basic approach is using a standard CNN to learn features of the input image (encoding), which is followed by a decoder who projects the learned features to the higher resolution pixel space.

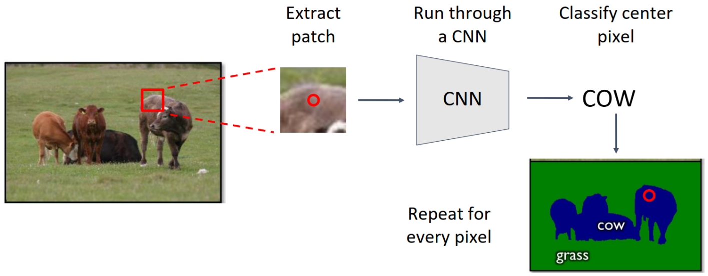


Figure 11: Pixel-wise classification pipeline

Low-res: due to all the pooling and down-sampling

This results in low resolution and fuzzy object boundaries. Fewer architectures can mitigate this by upsampling again. The learned features on each downsampled level are copied to the upsampling (decoding) stream, to give the fcnn a chance to include features of different levels.

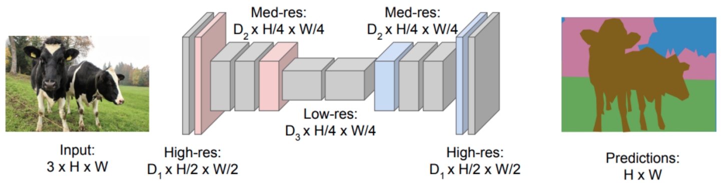


Figure 12: Downsample and than upsample to increase resolution again

Upsampling (NN): copy value for the whole output

Upsampling can be seen as interpolation, increasing the resolution of the signal.

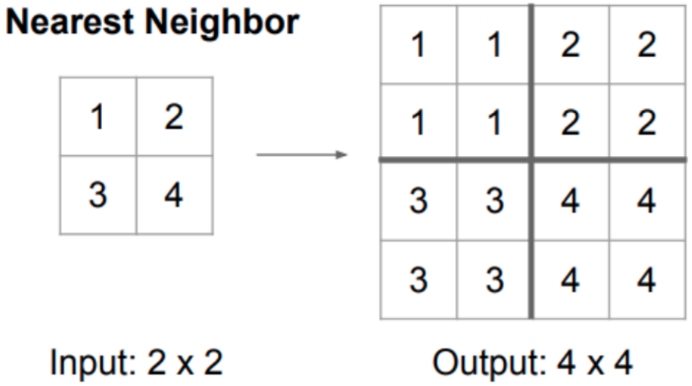


Figure 13: Copy input pixel to all output pixels

Bed-of-nails: zero all outputs but one copy of input

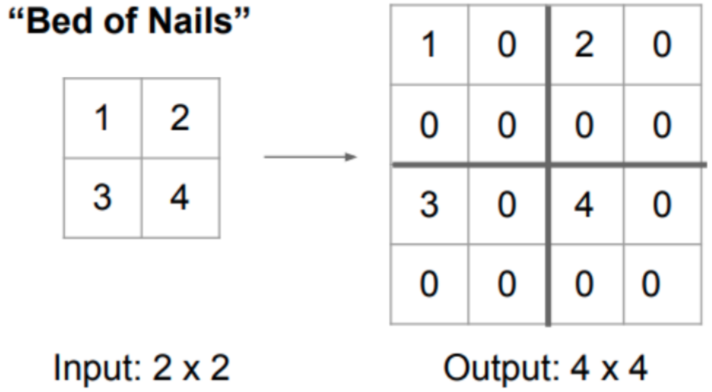


Figure 14: Copy input pixel to one output pixels, rest zero

Max-Unpooling: remember max pooling, BOF to that location

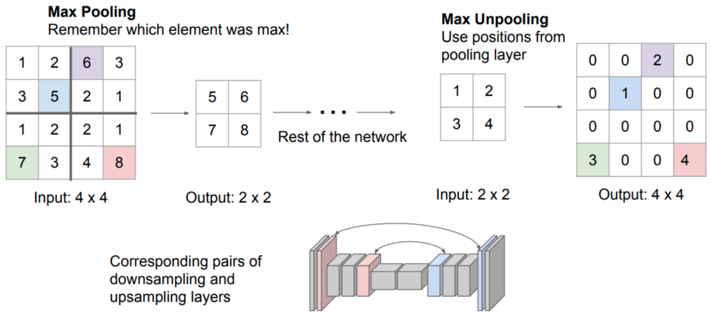


Figure 15: Copy input pixel to location of the max pooling pixel

Transpose Conv: input gives (learnable) weight for filter

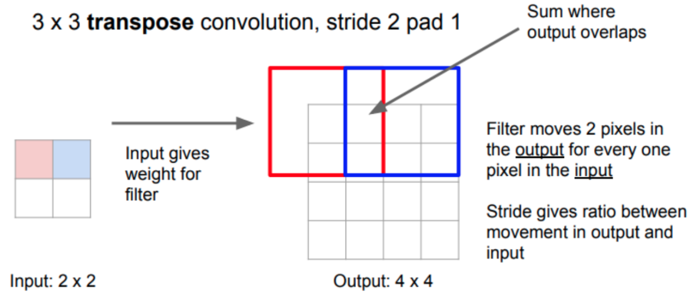


Figure 16: Input pixel gives the weight for a filter

UNet: TransCov + Skip connections

Main idea: combine global and local feature maps by copying corresponding tensors from earlier stages.

Also, the downsampling information is pretty shallow, upsampling performance is improved when including "pre-downsamplin" information.

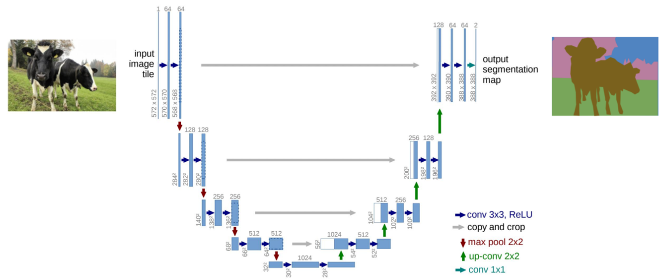


Figure 17: U-Net architecture

7 Recurrent Neural Network

Dyn. System: $h^t = f(h^{t-1}, x^t; \theta)$
The state at time t depends on the state at time t-1.
Note: We assume here that the same transition function and parameters are used for each timestep, also called auto-regressive models

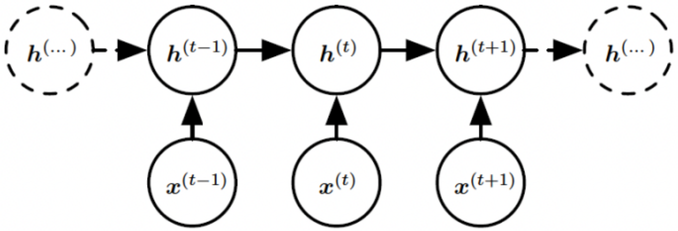


Figure 18: Visualisation unrolled RNN

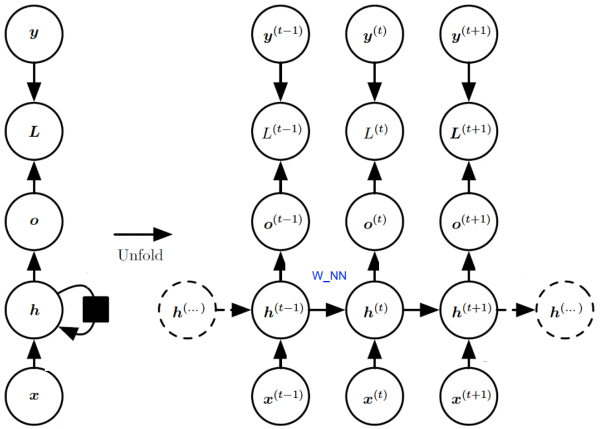


Figure 19: Visualisation of complex RNN

Vanilla RNN: $\hat{y} = W_{hy}h^t$, $h^t = tanh(W_{hh}h^{t-1} + W_{xh}x^t)$

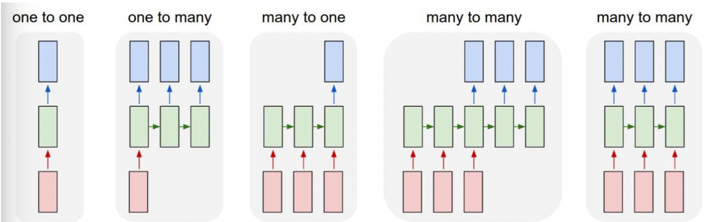


Figure 20: Possibilities of RNN architectures

Backprop: $h^t = f(h^{t-1}, x^t; W), \hat{y}^t = W_{hy}h^t, L^t = \|\hat{y}^t - y^t\|^2, \frac{\delta L}{\delta W} = \sum_{t=1}^{S} \frac{\delta L^t}{\delta W}$
Treat unrolled recurrent modelas multi-layer Network with unbounded number of layers and perform backprop.

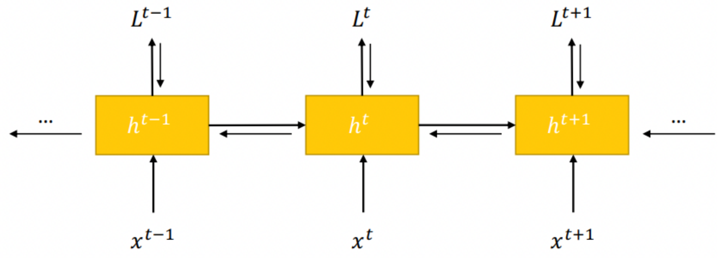


Figure 21: Unrolling of RNN

$\frac{\delta L^t}{\delta W} = \sum_{k=1}^{t} \frac{\delta L^t}{\delta y^t} \frac{\delta y^t}{\delta h^t} \frac{\delta h^t}{\delta h^k} \frac{\delta^+ h^k}{\delta W}$
The δ^+ indicates the immediate loss, since we could take the derivative infinite amount of times (apparently).

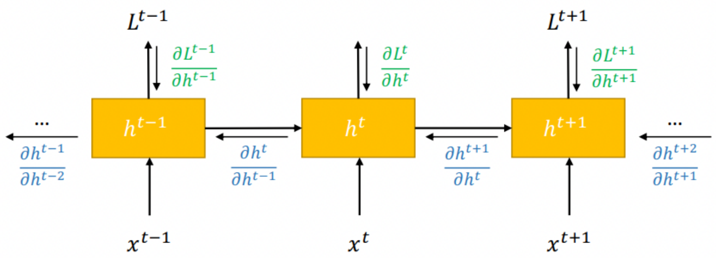


Figure 22: Unrolling of RNN - Individual Loss

Grads: $h^t = W^T h^{t-1} = (W^T)^t h^1 = (Q^T \Lambda^t Q) h^1$
The Eigenvalues explode or vanish, depending on their initial size.
Note: Regularizing the Eigenvalues is proven to reduce the learning capabilities of the learning-system drastically.

$$\frac{\partial L^t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L^t}{\partial y^t} \frac{\partial y^t}{\partial h^t} \underbrace{\frac{\partial h^t}{\partial h^k}}_{\text{we are interestedin this}} \frac{\partial^+ h^k}{\partial W}$$

$$\frac{\partial h^t}{\partial h^k} = \prod_{i=k+1}^{t} \frac{\partial h^i}{\partial h^{i-1}} = \prod_{i=k+1}^{t} W_{hh}^T diag[f'(h^{i-1})]$$

$$\forall_i, \left\|\frac{\partial h^i}{\partial h^{i-1}}\right\| \le \|W_{hh}^T\| \|diag[f'(h^{i-1})]\| < \frac{1}{\gamma}\gamma < 1$$

$$\left\|\frac{\partial h^t}{\partial h^k}\right\| < (\eta)^{t-k}$$

Figure 23: Proof that vanishing Gradients are a problem

Naive LSTM: $c_t = Wc^{t-1} + W_g g^t, h^t = tanh(c^t)$

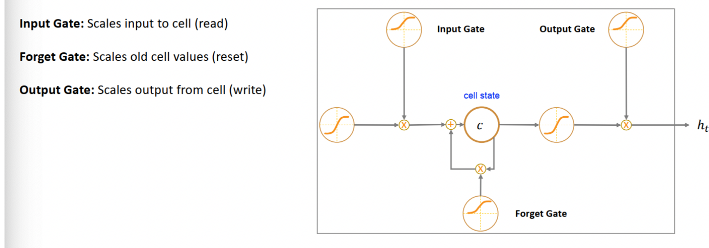


Figure 24: LSTM Abstraction

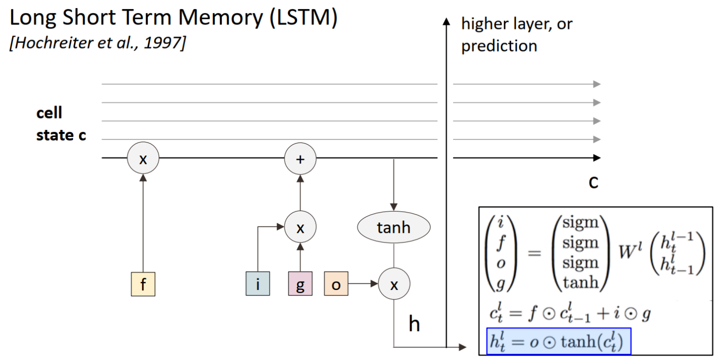


Figure 25: LSTM Visualization

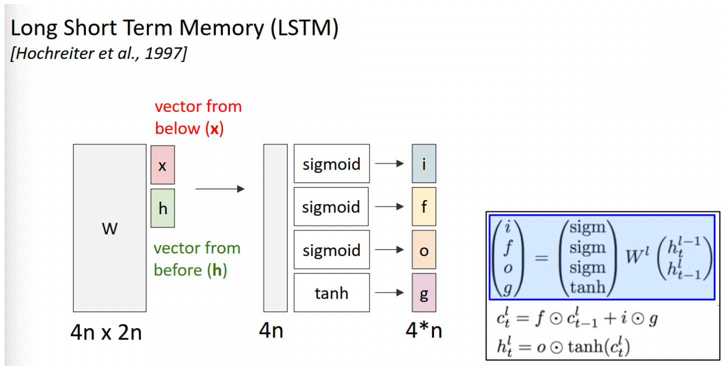


Figure 26: LSTM Visualization

QUESTION: How does this work?
TODO: Calculate through the matrices and functions to see if it works
RNN vs. LSTM: memory cell, adding gradient to stabalize

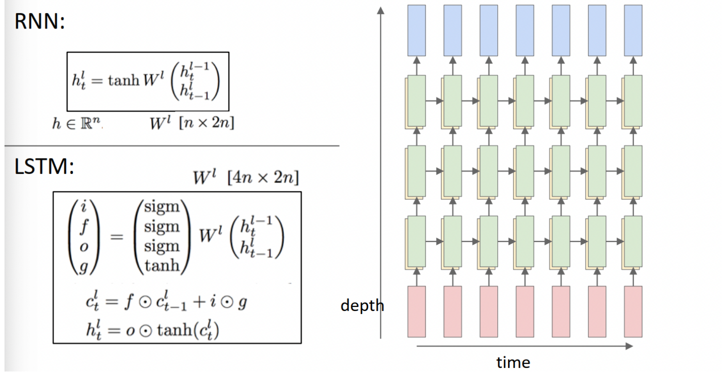


Figure 27: LSTM vs. RNN

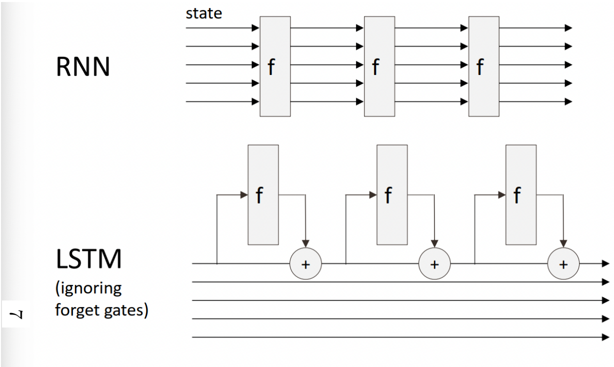


Figure 28: LSTM vs. RNN

Grad. clipping: Clip gradient, don't allow big jumps
When the gradient gets too big the learning gets unstable, because the big gradient allows huge jumping around. Clipping it might reduce the performance, but at least the direction is more stable.

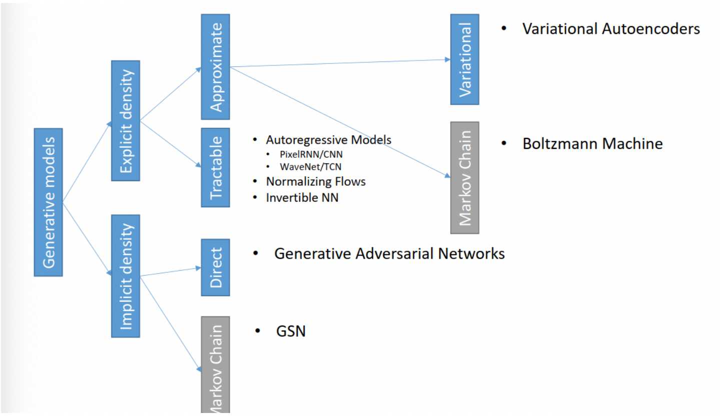
8 Variational Auto Encoders

Supervised vs. Unsupervised learning:

Supervised learning: Tries to learn a function that maps $X \Rightarrow Y$, used in classification, regression, object detection and segmentation.

Unsupervised Learning: The Goal here is to learn the hidden structure of the data, such as in clustering, feature learning, dimensionality reduction or density estimation.

Generative Modeling (our Goal): Given training data, learn a distribution and generate new samples drawn from the learned distribution.



∞ Figure 29: Taxonomy of variational models

TODO: Comment on the different models

Auto-Encoders: optimize $\theta_f, \theta_g = \arg\min \sum_n^N \|x_n - g(f(x_n))\|^2$

The encoder projects the original input to a latent space Z. The decoder samples from Z back to the input space. In the optimal case, the encoder-decoder pipeline approximates the identity function of the data.

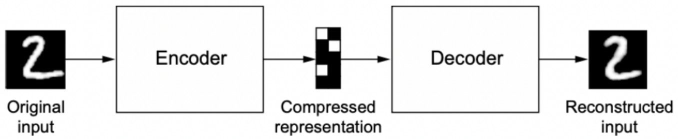


Figure 30: Auto-Encoders Model

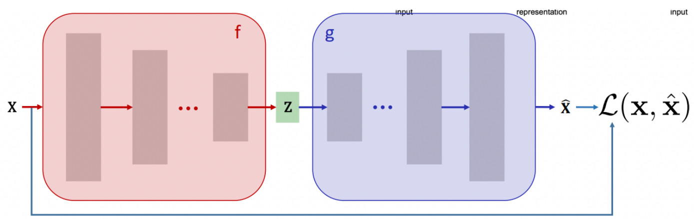


Figure 31: Auto-Encoders Model with loss

Latent Space: find low-res meaningful DOF. Optimally the latent space is continous and interpolatable

Not all dof are important, when we look at the image space (f.e. 256 x 256 x 3), we could as well sample something very random.

Note: The more dimensions the latent space has, the more information can be captured.

Small Z (Undercomplete): Compresses input, learns important features of input. Bad for ood samples

Big Z (Overcomplete): Units copies input components, but hidden units might not extract meaningful structure.

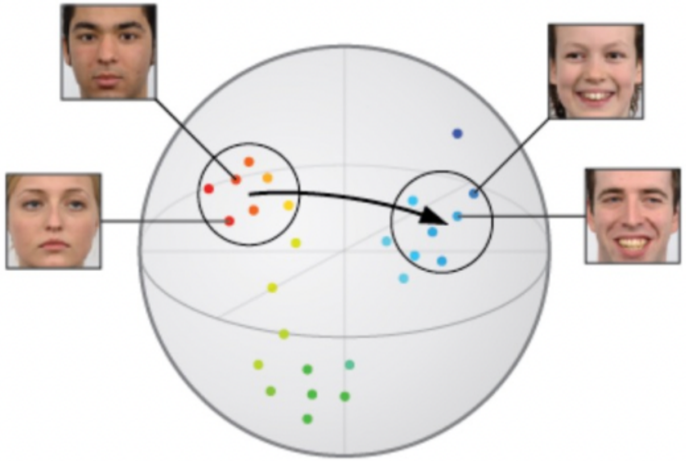


Figure 32: Latent space

PCA: linear embedding along the principle components. This happens if f and g are linear functions.

Denoising: add gaussian noise to input, reconstruct to original

Note: The Latent space is overcomplet in this instance

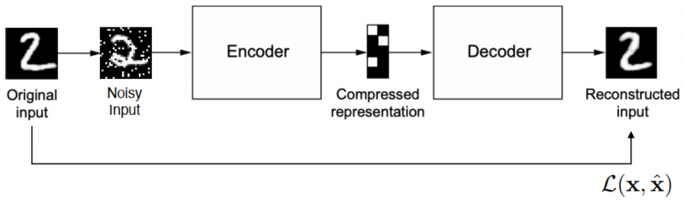


Figure 33: VAE pipeline for denoising

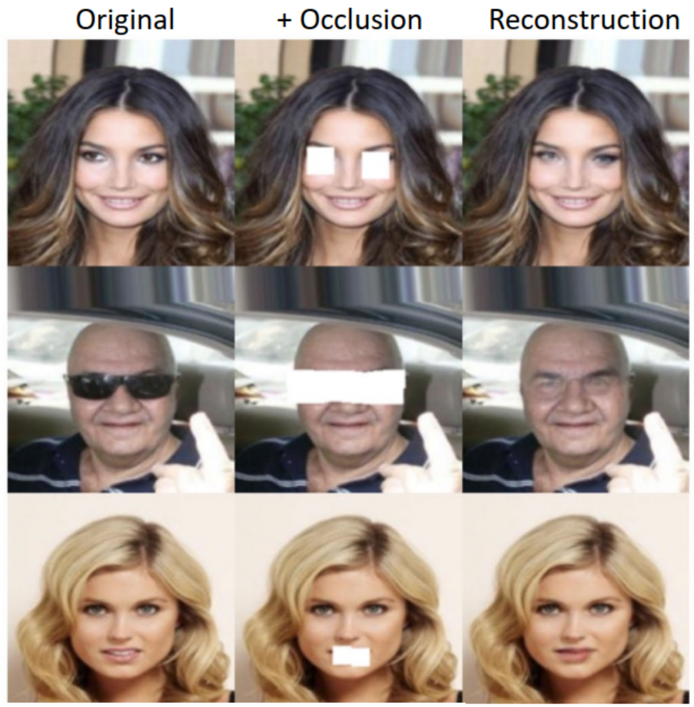


Figure 34: VAE de-noising example

NN: $z = f(a(x)) = \sigma(b + Wx), \widehat{x} = g(\widehat{a}(x)) = \sigma(c + Wz)$

AE limitations: not well-structured. Training samples are easy to reconstruct, but unknown samples can not be generated.

Variational Auto-Encoders use gaussians to model the latent space Z, the decoder can then use the distribution to sample new data.

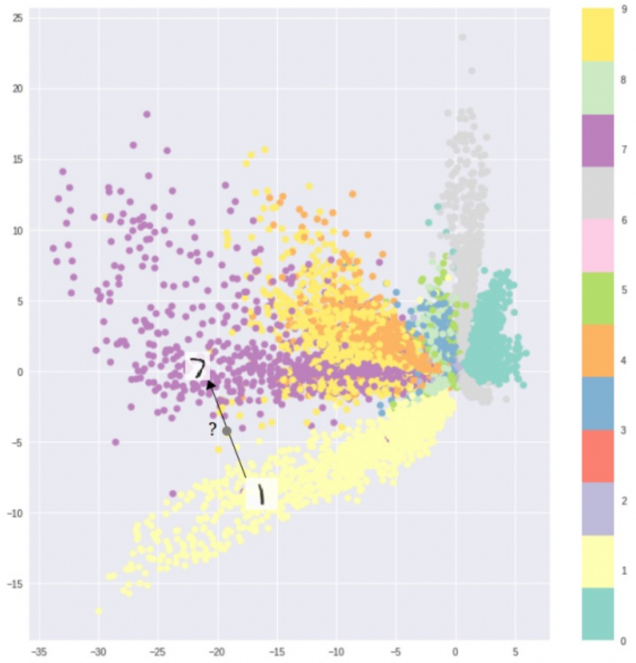


Figure 35: AE MNIST embedding