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

t 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 complex triggering patterns, f.e. a line in a certain direction. This hierarchy can be extended to hypercomplex shapes.

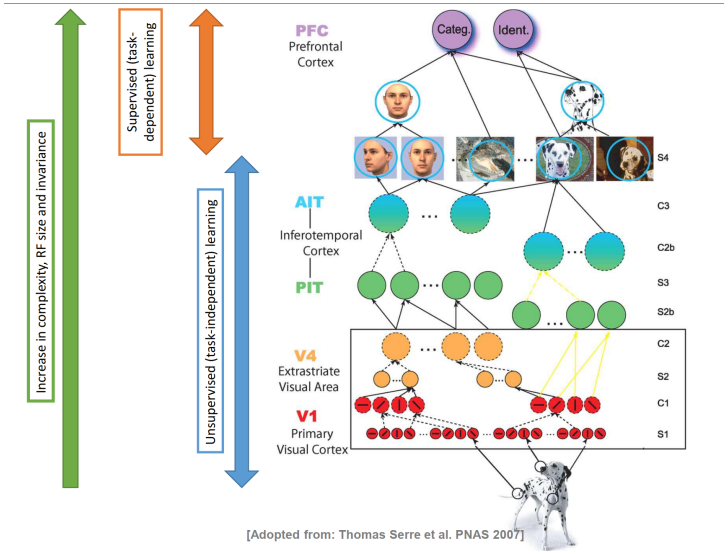

Figure 1: Complexity, unsupervised and supervised learning hierarchy

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.

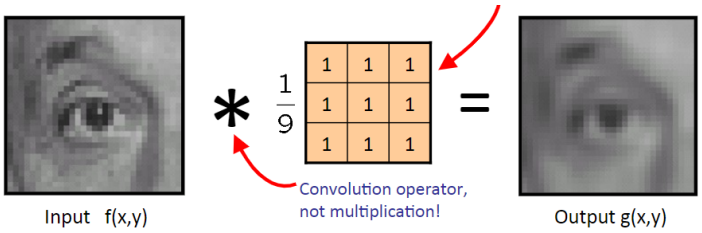

Figure 2: Linear filtering is applied with the convolution operator to compute a new image from the neigbourhood of each pixel of the old image

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 ourselfs?
∇: 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

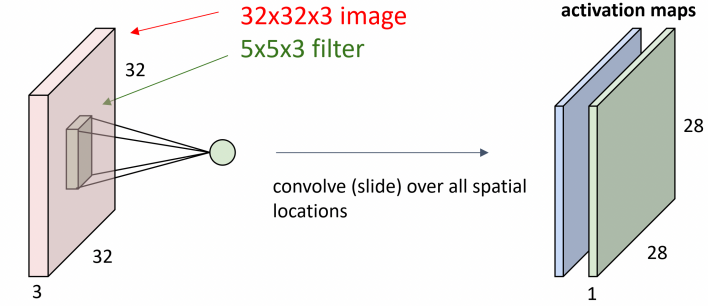

Figure 3: Visualisation applying filters to an image

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

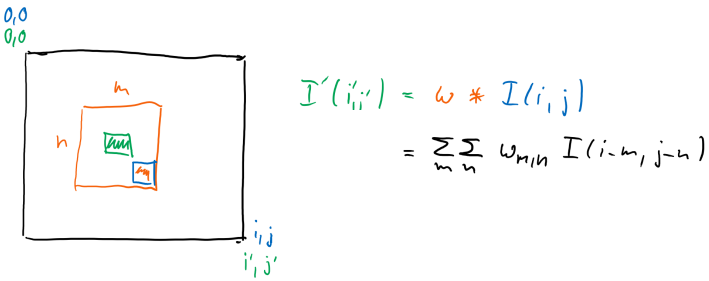

Figure 4: Visualisation forward pass

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

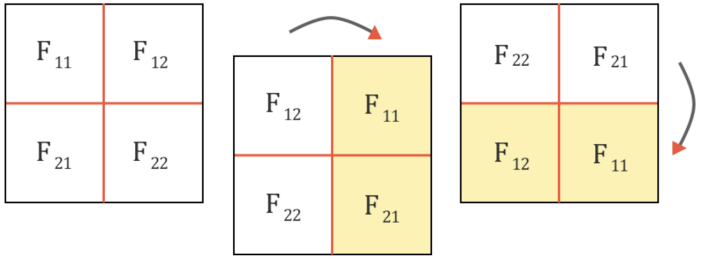

Figure 5: Flipped convolution operator

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.

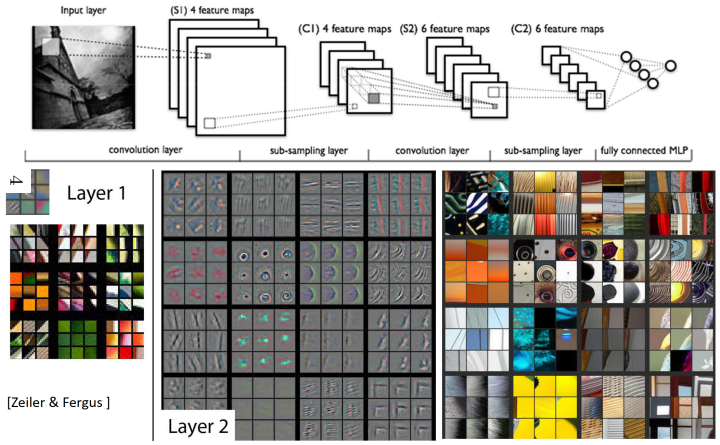

Figure 6: Visualisation of output of cnn layers

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.

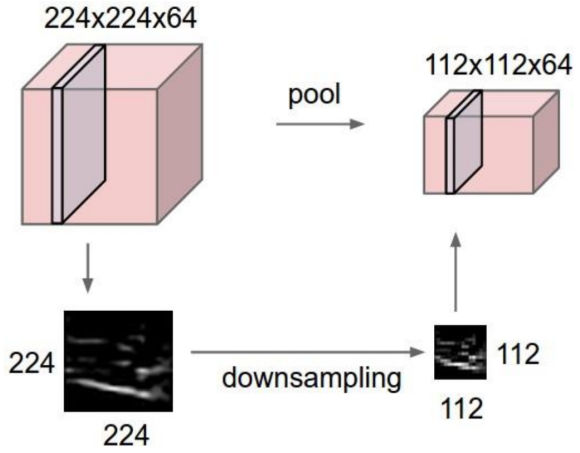

Figure 7: Visualisation of pooling layer

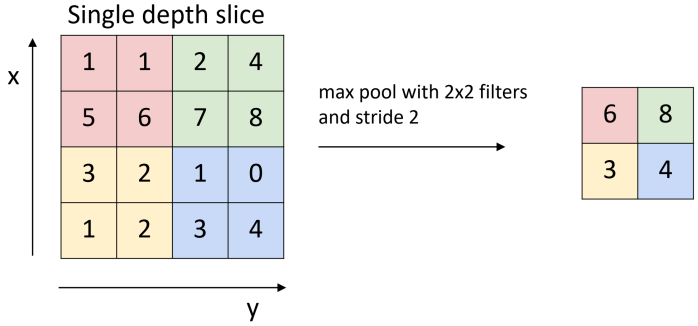

Figure 8: Visualisation of max-pooling layer

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

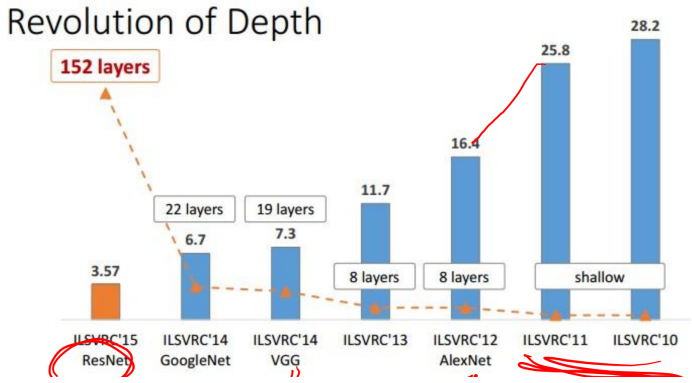

Figure 9: Evolution of NN architecture

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.

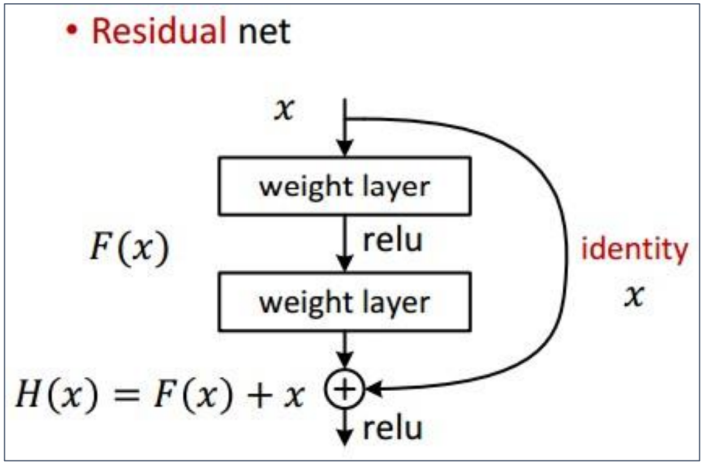

Figure 10: Evolution of NN architecture

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.

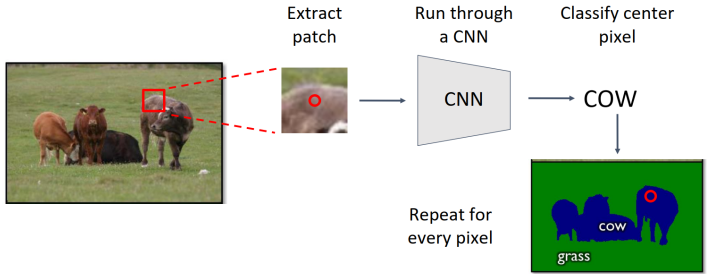

Figure 11: Pixel-wise classification pipeline

Low-res: due to all the pooling and down-sampling
This results in low resolution and fuzzy object boundaries. Lower 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.

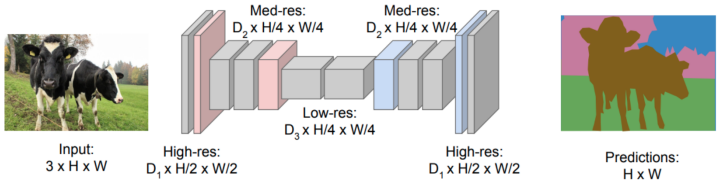

Figure 12: Downsample and than upsample to increase resolution again

Upsampling (NN): copy value for the whole output
Upsampling can be seen as interpolation, increasing the resolution of the signal.

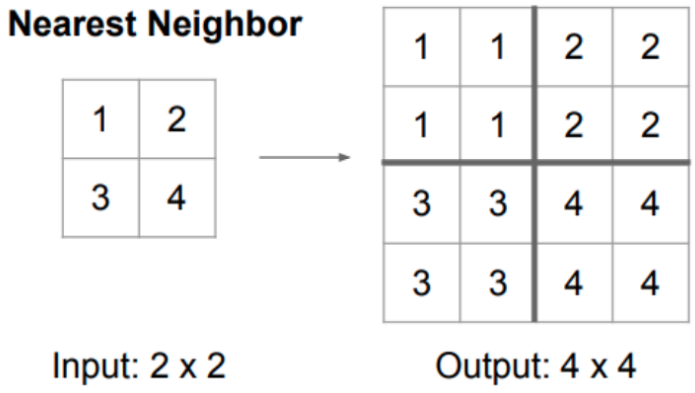

Figure 13: Copy input pixel to all output pixels

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

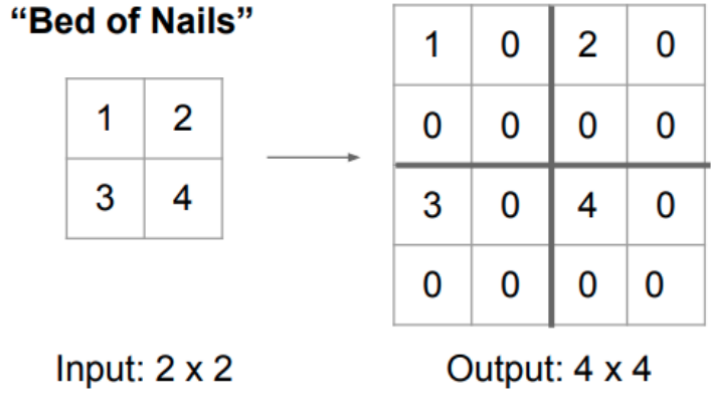

Figure 14: Copy input pixel to one output pixels, rest zero

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

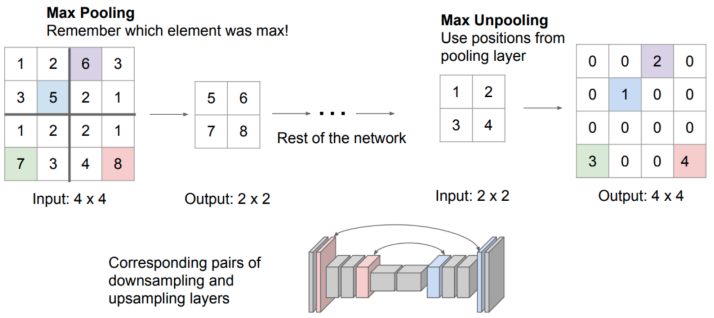

Figure 15: Copy input pixel to location of the max pooling pixel

Transpose Conv: input gives (learnable) weight for filter

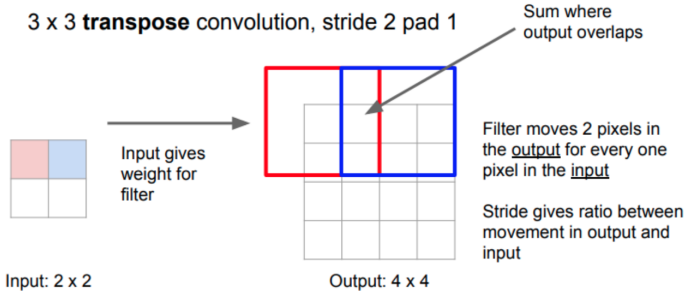

Figure 16: Input pixel gives the weight for a filter

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.

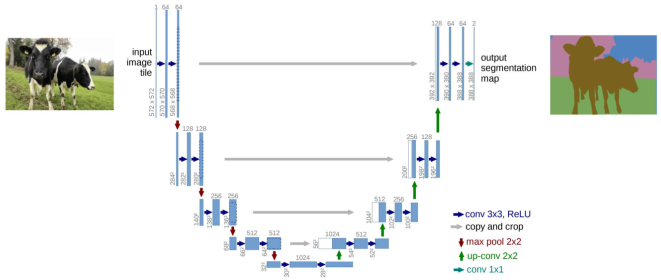

Figure 17: U-Net architecture

7 Recurrent Neural Network

The main advantage of RNN's over vanilla NN's or CNN's is the flexibility when it comes to input size.
NN has a fixed input/output size and computes a fixed amount of computational steps (defined by number of layers).
RNN can work over sequences of data, learning the transistion given the (implicitly stored) past data.

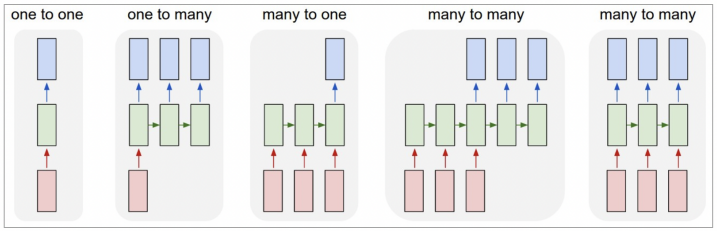

Figure 18: RNN 1. NN, 2. Image captioning, 3. Sentiment analysis, 4. Translation, 5. Video classification

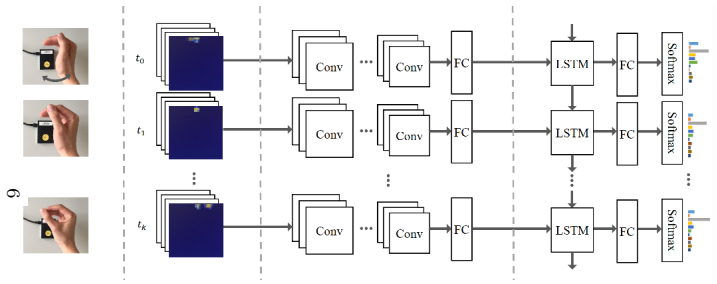

Figure 19: RNN Video classification architecture

Vanilla RNN: $\hat{y} = W_{hy}h^t,\ h^t = tanh(W_{hh}h^{t-1} + W_{xh}x^t)$
Goal: Learn transition function knowing what's worth keeping.
Tanh is important, otherwise we can't forget information.

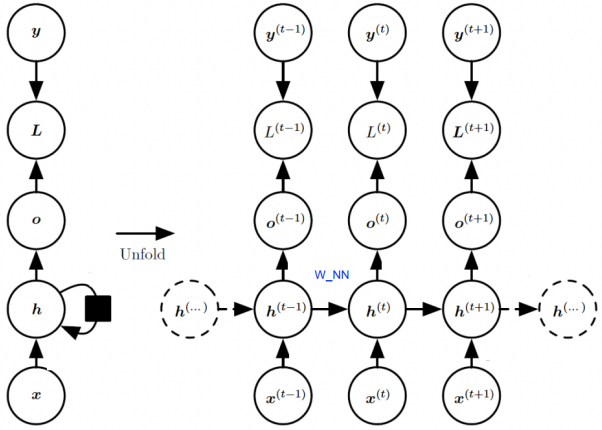

Figure 20: Visualisation of complex RNN

Backprop: $\frac{\delta L}{\delta W} = \sum_{t=1}^{S} \frac{\delta L^t}{\delta W} = \sum_{t=1}^{S} \sum_{k=1}^{t} \frac{\delta L^t}{\delta y^t} \frac{\delta y^t}{\delta h^t} \frac{\delta h^t}{\delta h^k} \frac{\delta^+ h^k}{\delta W}$
TODO: Understand this better

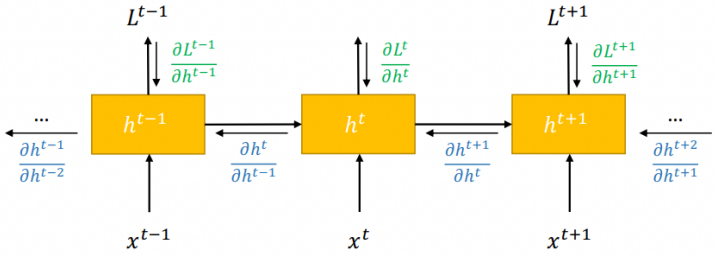

Figure 21: RNN Backprop unrolled

Gradient: $h^t = W^T h^{t-1} = (W^T)^t h^1 = (Q^T \Lambda^t Q) h^1$
The Eigenvalues of the weigh matrix explode or vanish when the sequence length gets to big. Regularizing the Eigenvalues is proven to reduce the learning capabilities of the learning-system drastically, and is thus not an option
Exploding gradients: exploding gradients can be managed by clipping after a certain size
Vanishing gradients: Add memory cell

$$\frac{\partial L^t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L^t}{\partial y^t} \frac{\partial y^t}{\partial h^t} \frac{\partial h^t}{\partial h^k} \frac{\partial^+ h^k}{\partial W}$$

we are interestedin this

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

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

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

Figure 22: Proof that vanishing Gradients are a problem

LSTM: $c_t^l = f \odot c_{t-1}^l + i \odot g,\ h_t = o \odot tanh(c_t^l)$,
$f, i, o, g = [\sigma, \sigma, \sigma, tanh] \odot W^l [h_{t-1}^l h_t^{l-1}]^T$
To mitigate the vanishing gradients, we introduce a long term memory unit. There are three phases to this: forgetting, new information and output generation.
1) Forgetting happens through manipulating c with $f \in [0, 1]$
2) New information is compiled by trying to figure out what to add/ remove to the current memory, as $g \in [-1, 1]$
3) Output generation is done by mixing the short-term input with the long-term memory

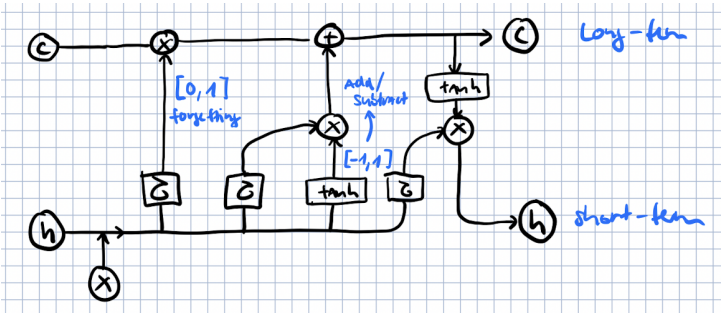

Figure 23: LSTM Visualization

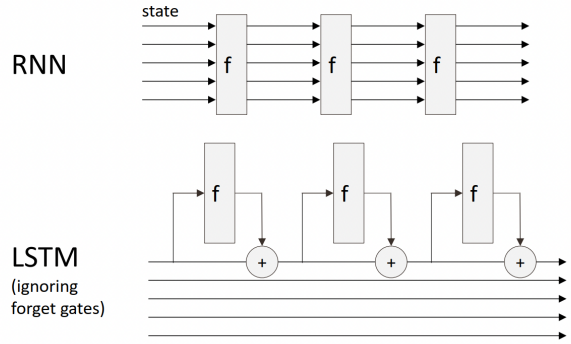

Figure 24: LSTM vs. RNN

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. Learn p_{model} close to p_{data}, generate from p_{model}

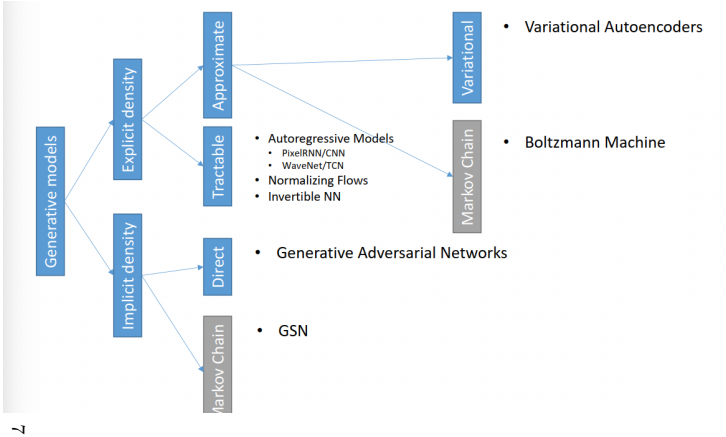

Figure 25: Taxonomy of variational models

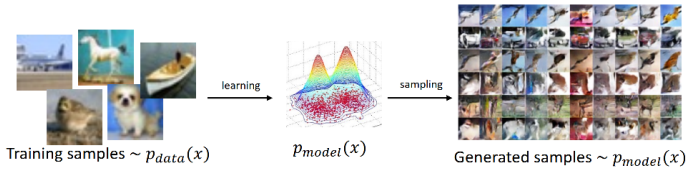

Figure 26: VAE Pipeline

AE: optimize $\theta_f, \theta_g = \arg\min \sum_n^N \|x_n - g_\theta(f_\theta(x_n))\|^2$

The encoder projects the original input to a latent space Z. It has to figure out what are the important features worth keeping (f.e. when dim(Z) < dim(X)).

The decoder learns to construct an output from a sample of Z.

In the optimal case, the encoder-decoder pipeline approximates the identity function of the data.

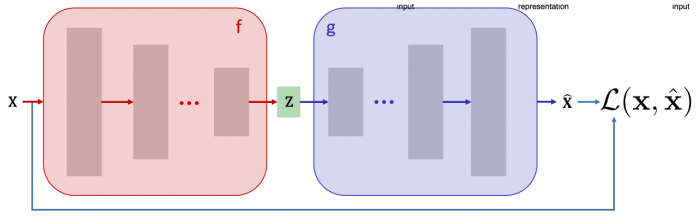

Figure 27: Auto-Encoders Model with loss

PCA: $z = f(x) = Wx + b$, $\widehat{x} = g(z) = W^*z + c$
linear embedding along the principle components.

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

Latent Space: meaningful DOF, optimally continous and interpolatable, under- or overcomplete

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 which does not represent an image.

Undercomplete Z: Compressed representation, learns important features of input. Bad for out-of-distribution samples

Overcomplete Z: Hidden units copies input components, but might not extract meaningful structure.

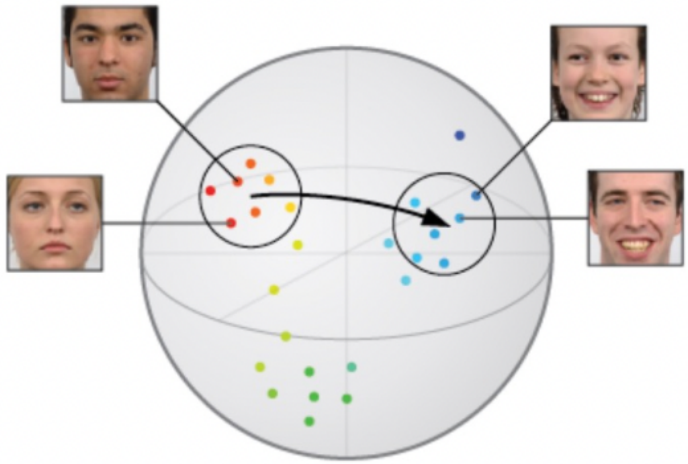

Figure 28: Latent space

Denoising: add gaussian noise to input, reconstruct to original

Note: The Latent space is overcomplet in this instance

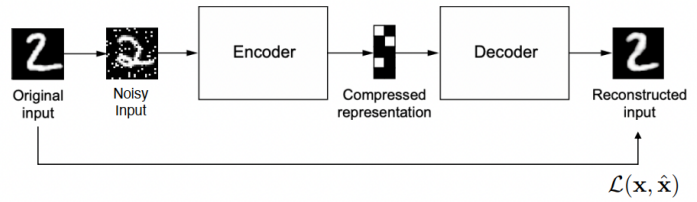

Figure 29: VAE pipeline for denoising

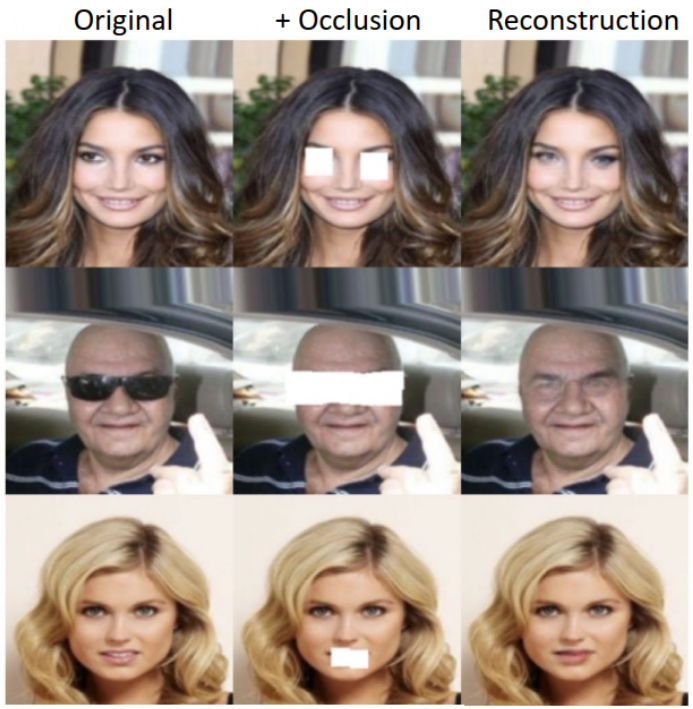

Figure 30: VAE de-noising example

Limit: latent space not interpolatable, unseen samples? Freshly generated samples from latent space Z do not make much sense, as in these regions, the decoder doesn't know what makes sense. Still, for reconstruction, the latent space is very good.

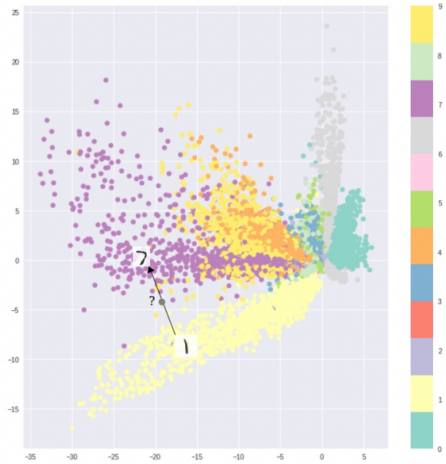

Figure 31: AE MNIST embedding, points in the space

VAE: The latent space Z is approximated by $f(x) \sim \mathcal{N}(\hat{\mu}, \hat{\sigma} I)$

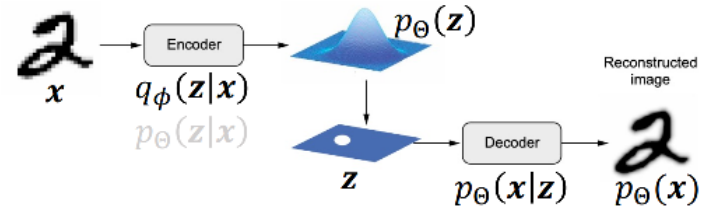

Figure 32: VAE Learned distributions

By design, the latent space is compact and interpolatable. This allows easy sampling and interpolation when generating new samples.

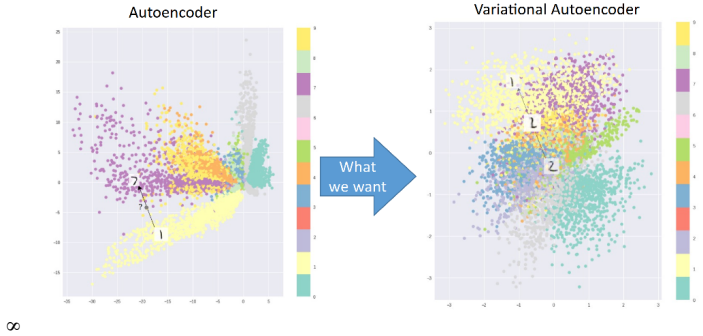

Figure 33: VAE Latent space

Encode: $q_\phi(z|x)$ for $\mu_{(z|x)}, \Sigma_{(z|x)}$, $z|x \sim \mathcal{N}(\mu_{(z|x)}, \Sigma_{(z|x)})$
The Encoder outputs a vector each for μ, σ. This multivariate gaussian can them be sampled to get the latent vector Z depending on x.
Problem: Using only the reconstruction error (MSE), the variance is going to be reduced to zero and the individual classes form clusters around their means. This leads to bad interpolation performance. We want to force the encoder to find features that are as close as possible to each other, while still being distinct.

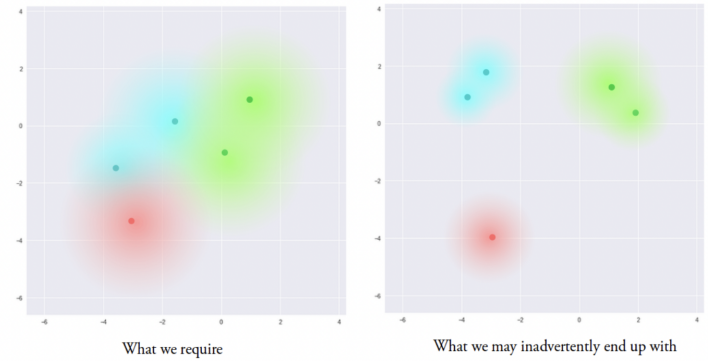

Figure 34: VAE Reconstruction loss only

Decode: $p_\theta(x|z)$ for $\mu_{(\hat{x}|z)}, \Sigma_{(\hat{x}|z)}$, $\hat{x}|z \sim \mathcal{N}(\mu_{(\hat{x}|z)}, \Sigma_{(\hat{x}|z)})$

KL: $-D_{KL}(q_\phi(z|x)||p_\theta(z)) = \int_x q_\phi(z|x) \log(\frac{p_\theta(z)}{q_\phi(z|x)})$
$= \frac{1}{2}\sum_i^J (1 + \log(\sigma_j^2) - \mu_j - \sigma_j^2)$, if $q \sim \mathcal{N}(\mu, \sigma I), p \sim \mathcal{N}(0, I)$
from exercise: $\int p(z) \log q(z) dz =$
$-\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_i^J \log \sigma_{p,j} - \frac{1}{2}\sum_j^J \frac{\sigma_{p,j}^2 + (\mu_{p,j} - \mu_{q,j})^2}{\sigma_{q,j}^2}$
KL meassures how similar two distributions are. We punish the encoder if the final distribution diverges away from the standart normal. **Note:** Assymetric and non-negative

ELBO: $p_\theta(x) \geq E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z))$
The first part of the loss is the reconstruction loss. It meassures the loss from the decoder prediction.
The second part of the loss is the latent code loss. It enforces that the latent space is a gaussian, e.g. enforcing properties such as compactness and good interpolation
Note: We want to maximize the ELBO
Note: If we have a sequence, the ELBO is the sum of all variational lower bounds
Note: The different classes are still entangled, interpolation works good, but features are not distinguishing unique properties.

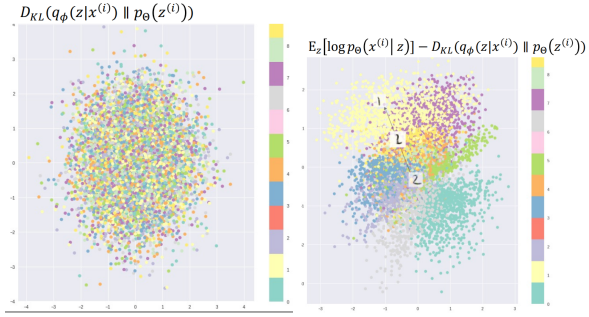

Figure 35: VAE KL-Loss only vs. both losses (entangled)

Goal: Learn features that correspond to distinct factors of variaton e.g., digits and style (or thickness, orienta9on) Statistical independence of the of the features can be used
In the picture below, one can see that the AE might achieve detangled features, but it is bad for interpolation. The KL divergence loss craetes some entaglement, though.

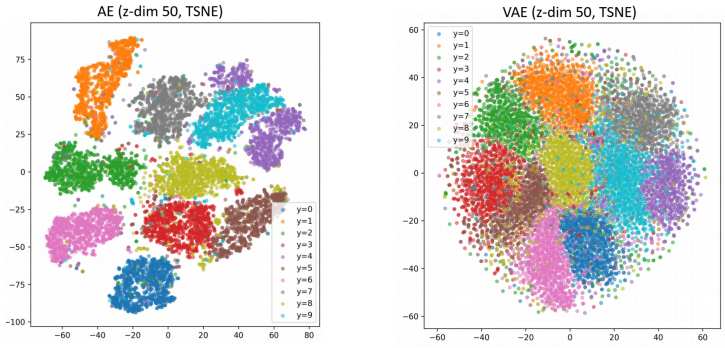

Figure 36: VAE vs. AE

β-**VAE:** $\mathcal{L} = -E_{q_\phi(z|x)}[\log p_\theta(x|z)] + \beta D_{KL}(q_\phi(z|x)||p_\theta(z))$
Goal is to disentangle the different features. The KL-loss tries to create a diagonal multivariate Gaussian, which perfectly seperates the independent features. The reconstruction loss fucks this up, as it fights for the loss as well.
Putting more weight for the KL loss helps solving it in an unsupervised fashion. Alternatively, could condition on features.

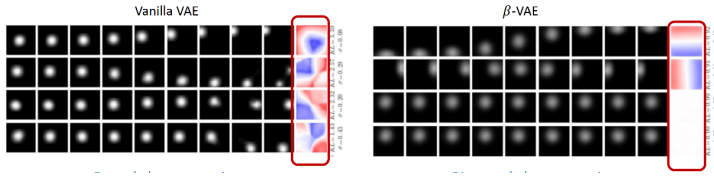

Figure 37: VAE Entangled vs. Disentangled representation

9 Auto-regressive Models

Regressive Property: $x_t = b_0 + b_1 x_{t-1} + b_2 x_{t-2}$

Sequence Model: $p(x) = \prod^N p(x_i|x_1, .., x_{i-1}) = \prod p(x_i|x_{<i})$
We are trying to estimate a distribution here.
Note: Sequence models trivially fulfill the regressive property

Believe Net: $\hat{x}_i = p(x_i = 1|x_{<i}) = Ber(\sigma(\sum_1^{i-1} w_j^i x_j + w_0^i)$
In this fully visible sigmoid belief network, the value of the next element is modelled as a Bernoulli variable.
Note: The weight Matrix $W \in \mathbb{R}^{n \times n}$ is triangular.
Element i at row j contains w_i^j, describing weight of x_i when calculating $\hat{x}_j$.

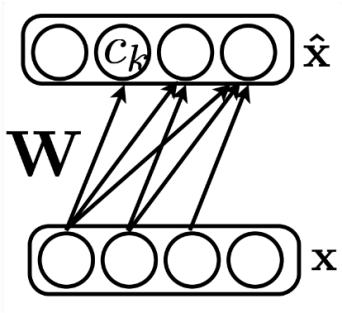

Figure 38: AR Fully Visible Sigmoid Belief Network

NADE: $h_i = \sigma(b + \sum^{i-1} w_j x_j)$ (in $O(H)$, update in $O(1)$)
$\hat{x}_i = p(x_i = 1|x_{<i}) = \sigma(c_i + V_i h_i)$ (N times $\Rightarrow$ total $O(NH)$)
The difference to FVSB-Net is that the weights are shared. Therefore, the number of weights is in $\mathcal{O}(n)$.
Note: $b + W_{<i+1} x_{<i+1} = b + W_{<i} x_{<i} + W_{i+1} x_{i+1}$ in $\mathcal{O}(1)$

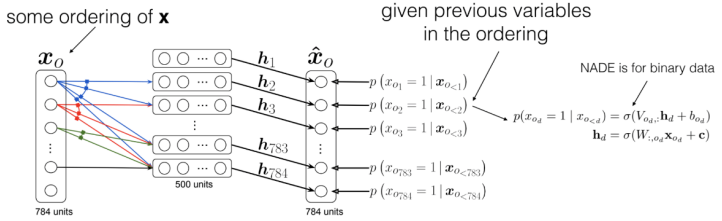

Figure 39: AR NADE Schema

PixelRNN: $p(x) = \prod^{N^2} p(x_i|x_{<i}) = p(x_{i,R}|x_{<i})p(x_{i,G}|x_{<i}, x_{i,R})$
RNN's are autoreg. from nature: h_t summarises inputs $< t$
Learning and generation is slow, because of the explicit pixel dependencies from the pixel ordering.
If we use a LSTM for example, $h_{i,j} = f(h_{i-1,j}, h_{i,j-1}, p_{i,j})$, e.g. no parallelization in training, neither in prediction, can be made. It is very slow, but LSTM and RNN can capture long-term dependencies very well.

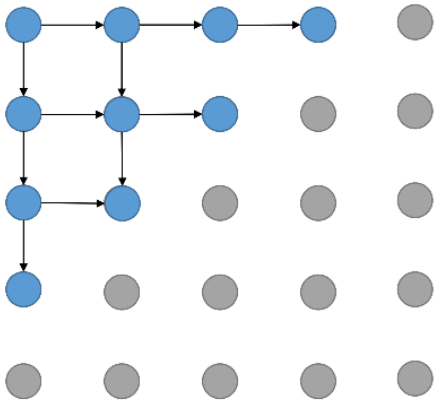

Figure 40: AR PixelRNN dependencies

PixelCNN: Use conv + mask, parallelize training, blind spot
Note: Generation is still sequential
Summary:
Pros are the explicit likelihood p(x) and good evaluation metric through LL of training data.
Cons are slow generation

WaveNet: dilated convolution for large scale temp. dependencies
The issue with sound is that the frequency is at $\sim$ 16'000 samples/second.
To capture speach, convolution must have a very large receptive field.

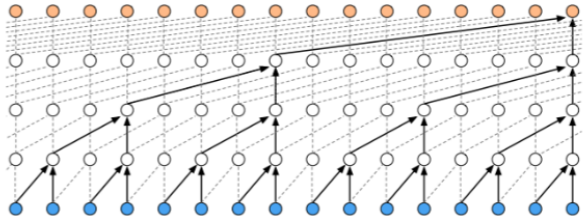

Figure 41: AR WaveNet long term dependencies

Dilated Conv: exponential receptive field size increase

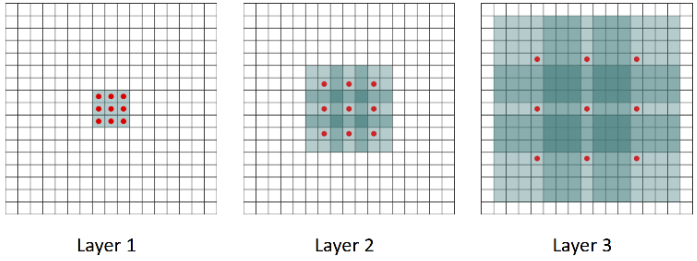

Figure 42: Dilated Conv: Receptive field increase 1

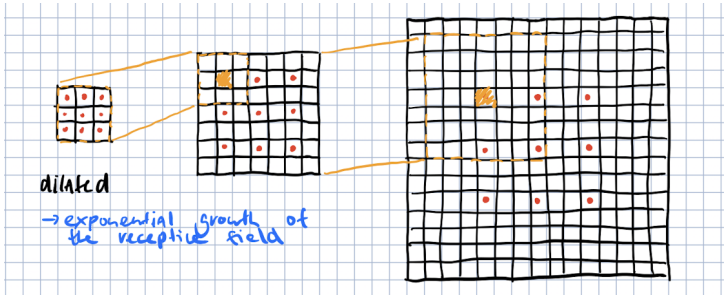

Figure 43: Dilated Conv: Receptive field increase 2

Self-Attention: $K = XW_K, V = XW_V, Q = x_t W_Q (\in \mathbb{R}^D)$,
$X \in \mathbb{R}^{T \times D}, W \in \mathbb{R}^{D \times D}, \alpha = softmx(QK^T/\sqrt{D}), x_{t+1} = \alpha V$
$X = softmax(\frac{(W_Q X)(W_K X)^T}{\sqrt{D}} + M)(W_V X)$
RNN: $h_t = g(x_{t-1}, h_{t-1})$, $x_t = f(x_{t-1}, h_t)$. Learns to encode relevant information for future steps and maintains long-term temporal relations.
Complexity per layer: $\mathcal{O}(TD^2)$, with $\mathcal{O}(T)$ sequential operations
CNN: $x_t = f(x_{t-1}, x_{t-2}, .., x_1)$. Learns to aggregate past information for the next step.
Complexity per layer: $\mathcal{O}(kTD^2)$, with $\mathcal{O}(1)$ sequential operations
Self Attention: $x_t = f(x_{t-1}, x_{t-2}, .., x_1)$. Learns to identify relevant information for the next step.
Very expressive, but computationally complex. Attention is a quadratic operation. Complexity per layer: $\mathcal{O}(T^2 D)$, with $\mathcal{O}(1)$ sequential operations. Can be restricted in how far we look back, reduces complexity of layer

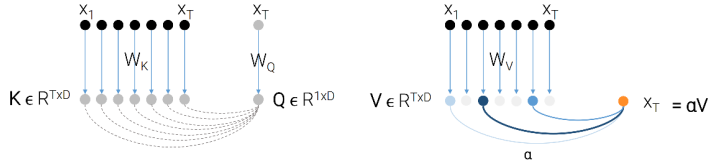

Figure 44: AR Self-attention

TODO: Understand the runtimes better, be able to derive this

10 Normalizing Flows

Goal: What we want is a model that combines the tractable likelihoods from AR models with the latent space of VAE.

Variable change 1D: $p_x(x) = p_z(h(x))\,|h(x)|$
2D: $\int\int f(x,y)dxdy = \int\int f(g(u,v),h(u,v))J(u,v)dudv$
The probabilitstic mass must be preserved!

Normalizing Flow: $f : \mathbb{R} \to \mathbb{R}$, cont. and invertible

$$p_X(x;\theta) = p_Z(f_\theta^{-1}(x))\left|det(\frac{\partial f_\theta^{-1}(x)}{\partial x})\right| = p_z(f^{-1}(x))\left|det(\frac{\partial f(z)}{\partial z})\right|^{-1}$$

Example Linear transformation: $z = Ax$
$\Rightarrow p_x(x) = p_z(A^{-1}x)\,|det(A^{-1})|$
Note: Not every NN works to model f:

- must be differentiable
- must be invertible
- must preserve dimensionality
- jacobian must be computed efficiently

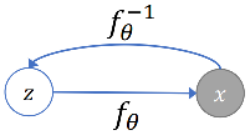

Figure 45: Normalizing Flow mapping

Coupling: $(y_A; y_B)^T = (h(x^A, \beta(x^B)); x^B)^T$ | h:elemnt, β:NN
$(x^A; x^B)^T = (h^{-1}(y^A; \beta(y^B)); y^B)^T$, $J = ((h'; h'f'), (0; 1))$
Note: The Jacobian is triangular. This allows computation of det in $\mathcal{O}(d)$, instead of $\mathcal{O}(d^3)$
Note: β does not have to be invertible

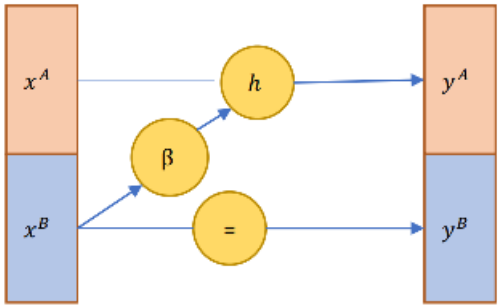

Figure 46: Normalizing Flow coupling layer

Composition: $p_X(x;\theta) = p_Z(f_\theta^{-1}(x))\prod_k \left|det(\frac{\partial f_\theta^{-1}(x)}{\partial x})\right|$
For $x = f(z) = f_k \circ f_{k-1} \circ \circ \circ f_2 \circ f_1(z)$

Train: $logp_x(D) = \sum_x^D (logp_z(f^{-1}(x)) + \sum_k \log\left|det(\frac{\partial f^{-1}(x)}{\partial x})\right|)$

Inference: draw $z \sim p_z(.)$, $\widehat{x} = f(z)$
The whole point of this is that we want to draw $z \sim \mathcal{N}(0, I)$ and then learn the transformation to create really complex x

Model: $x_i \to squeeze \to n.flow \to split \to z_i(\text{repeat } L-1)$ $\to squeeze \to n.flow \to z_L$
The flow step is repeated K times and consists of

- actnorm (like batch normalization)
- 1x1 conv
- affine coupling layer