Negative log-likelihood loss: $L = -\sum y_i \cdot \log \hat{y}_i$ — Jonas B.

Sigmoid: 0-1 / like a neuron / get small gradients / not 0 centered
$f(x) = \sigma(x) = 1/(1+e^{-x})$ / $f'(x) = f(x)\cdot(1-f(x))$

Softmax: multiclass / $\Sigma = 1$ / $f(x) = \exp(\eta_i)/\sum_j \exp(\eta_j)$

tanH: -1-1 / zero centered / steeper gradients
$f(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ / $f'(x) = 1 - f(x)^2$

ReLU: $[0,\inf]$ / can blow up / accelerate convergence / negative units get no update
$f(x) = \max(0,x)$ / $f'(x) = \begin{cases} 0 & \text{for } x<0 \\ 1 & \text{for } x\geq 0 \end{cases}$

Leaky ReLU: solves dying ReLU problem / α small positive constant
$f(x) = \begin{cases} \alpha x & \text{for } x<0 \\ x & \text{for } x\geq 0 \end{cases}$ / $f'(x) = \begin{cases} \alpha & \text{for } x<0 \\ 1 & \text{for } x\geq 0 \end{cases}$

Gradient Descent: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$ → $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$

Stochastic Gradient Descent: Minibatches / can jump / risk of overshooting

AdaGrad: Modifies learning rate for each parameter θ_i based on past gradients

Max Likelihood: $L(\theta) = \prod p(x_i|\theta) \Leftrightarrow \log L(\theta) = \log \prod_i p(x_i|\theta) \Leftrightarrow \log L(\theta) = \sum_i \log p(x_i|\theta)$

Feed Forward Network Block: $y = \sigma(w^T x + b)$ / perceptron

XOR: $x = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$ $y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$
- Linear: $J(\theta) = \frac{1}{N=4} \sum_{x \in X} (f^*(x^{(i)}) - f(x^{(i)}, \theta))^2$ $f(x,w,b) = x^T w + b$ $w = \begin{bmatrix} 0 \\ \frac{1}{2} \end{bmatrix}$ $b = \frac{1}{2}$
- ReLu: $f(x, w_1, c, w, b) = w^T \cdot \max(0, xw + c) + b$ / $w = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ $w = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$ $b = 0$

Universal approx Theorem: Feed forward neural network with 1 hidden layer and continuous non-linear activation function can approximate any continuous function with arbitrary precision / leading edge: $-b/w$ / w je grösser steiler / b je negativer desto rechts → $w = 100$, $b = -90$

Simple Cell: CNN / small receptive field / at the beginning / tuned to specific stimuli

Complex Cell: Combine output from simple cells / increase variance and receptive field

Convolution: linear: $T(\alpha u + \beta v) = \alpha T(u) + \beta T(v)$ / Invariant: $T(f(u)) = T(u)$ / equivariant $T(f(u)) = f(T(u))$

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

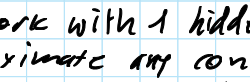
Convolution: $I'(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} K(-m,-n) \cdot I(i+m, j+n)$

$K(i,j) = K(-i,-j)$ → Correlation = Convolution

Semantic Segmentation: Classify each pixel / Downsampling (Max-Pool/Strided) then Upsample again

Upsampling: Unpooling (reverse convolution) / Max Unpooling (remember which element max was)
Learnable Upsampling (learn transpose convolution filter)

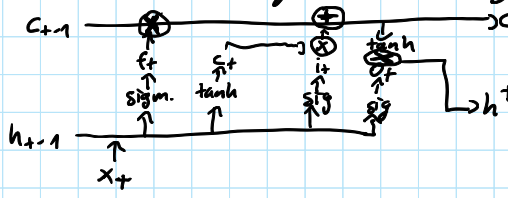
RNN: $h^+ = \tanh(W_{hh} \cdot h^{t-1} + W_{xh} \cdot x^t)$ / $\hat{y}^+ = W_{hy} \cdot h^+$



1:1 = Vanilla / 1:m = Image captioning / m:1 = Sentiment Classification
m:n = Machine Translation / m:n = Video classification for each frame
Parameters shared between time steps / Impossible to capture long-term dependancies
Gradient multiplication for long sequences: vanish → LSTM / explodes → gradient clipping

LSTM: forget gate: what forgotten from previous cell state
input gate: how much input we should take
output gate: how much memory affects output

$f_+ = \sigma_g(W_f x_+ + U_f \cdot h_{+-1} + b_f)$
$i_+ = \sigma_g(W_i \cdot x_+ + U_i \cdot h_{+-1} + b_i)$
$o_+ = \sigma_g(W_o \cdot x_+ + U_o \cdot h_{+-1} + b_o)$
$c_+ = f_+ \circ c_{+-1} + i_+ \circ \sigma_c(W_c \cdot x_+ + U_c \cdot h_{+-1} + b_c)$ / $h_+ = o_+ \circ \sigma_h(c^+)$



Cell state is memory / hidden state is for output

GRU: Cell state stored in hidden state / lacks output gate
because memory already in hidden state

Regularization:
- Data standardization for better training convergence & local minima avoidance → $x_s = \frac{x-\mu}{\sigma}$ / $\mu = \frac{1}{n}\sum x_i$ / $\sigma = \sqrt{\frac{1}{n-1}\sum (x_i - \mu)^2}$
- Data augmentation: Real/synthetic discrepancies
- Parameter norm penalties: $L(\theta) = L(\theta) + \alpha \cdot \Omega(\theta)$ / L1: lasso: sparse
- Dropout: Models share weights / trains part of model L2: ridge: shrinkage
- Bagging: Models are independant / train all models until convergence
- Pretraining with general big dataset or pretrained model

Autoencoder:
- Encoder encodes x into latent space z = f(x) / Decoder decodes z into $\hat{x}$ g(z)
- g of f approximates identity / $\hat{\theta}_f, \hat{\theta}_g = \text{argmin}_{\theta_f, \theta_g} \sum_n^N \|x_n - \hat{x}_n\|^2$
- If f and g linear → optimal ↔ $\hat{\theta}_f, \hat{\theta}_g = \text{argmin}_{\theta_f, \theta_g} \sum_n^N \|x_n - g(f(x_n))\|^2$
  solution given by PCA

Kullback-Leibler divergence: $D_{KL}(P\|Q) = \int_x p(x) \cdot \log \frac{p(x)}{q(x)} \cdot dx$ (How wide apart are dist. p(x) and q(x))

Batch normalization: distribution shifts at each training iteration → weights need to adjust constantly
↳ standardize each layer distribution / Hard to compare models since log &

Spatial modeling: crop/same: pad with 0 → $(W - K + 2P)/S + 1$

Specificity: Subcategories
Invariance: Not only one picture

$\rightarrow E_z[\log p_\theta(x^{(i)}|z)] - D_{KL}(q_\theta(z|x^{(i)}) \| p_\theta(z^{(i)}))$

Blurry images due to noise and inference model

By introducing density model (Gaussian) over latent space z $f(x) \sim N(\hat{\mu}, \hat{\sigma}^2 I)$ where $\hat{\mu}$ and $\hat{\sigma}^2$ are estimated on training data => Blurry images because z discrete

Variational Autoencoder: · To get x: sample from $p_\theta(x|z^{(i)})$ / $p(z)$ is simple prior
We parametrize $p(x|z)$ by neural network since complex / $p(x) = \int_z p(x|z) \cdot p(z) \cdot dz$
Use $p(z|x) = p(x|z) \cdot p(z) / p(x)$ / $p(z|x)$ is approximated by encoder network

β-VAE: · Learn disentangled representation without supervision / Provide framework for automated discovery of interpretable factorised latent representation / modify VAE: add adjustable hyperparameter β balances latent channel capacity and constraints

GAN: · Generator $G: \mathbb{R}^D \to X$ and Discriminator $D: X \to [0,1]$ /
$\min_{\theta_g} \max_{\theta_d} E_{x \sim p_d(x)}[\log D_{\theta_d}(x)] + E_{z \sim p_z(z)}[\log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$
1. Gradient ascend on D: $\max_{\theta_d} E_{x \sim p_d(x)}[\log D_{\theta_d}(x)] + E_{z \sim p_z(z)}[\log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$
2. Gradient ascend on G: $\max_{\theta_g} E_{z \sim p_z(z)}[\log(D_{\theta_d}(G_{\theta_g}(z)))]$
k-steps do sample x_{R1}, x_{G1}, update discriminator $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(\overbrace{x^{(i)}}^{\text{sample}}) + \log(1 - D(G(\overbrace{z^{(i)}}^{\text{generated}})))]$
do one update generator $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_g}(G_{\theta_g}(z^{(i)})))$ / Nash-Equilibrium-2 player-game
Mode collapse: generator collapses and produces limited varieties of samples due focus on one mode

Higher quality images

GAN > VAE: Sharper images / GANs only care about samples / No variational bound needed

Autoregressive property: linear reg. $\hat{y} = b_1 \cdot x_1 + b_0$ / autoreg. used for time series
Tabular approach: chain rule probabilities: $p(x) = \prod p(x_1|x_{n-1}, \dots, x_{i-1}) = \prod p(x_i|x_{<i})$
Fully visible belief network: $\hat{x}_i = p(x_i = 1 | x_{<i}) \Rightarrow$ logistic reg.: $f_i(x_{n-1}, \dots, x_{i-1}) = \sigma(\alpha_0^i + \alpha_1^i x_1 + \dots)$
NADE: Neural Autoregressive Density Estimator / Autoencoder-like neural network
Max. average log-likelihood: $\frac{1}{T} \sum [\log(p(x^t)] = \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{D} \log p(x_i^t | x_{<i}^t)$
Efficient $O(TD)$ / could make use of second order optimizers

MADE: Masked Autoencoder Distribution Estimator / constrain Autoencoder that output can be used as conditionals $p(x_i|x_{<i})$ / No computation path between x_d and input $x_d, \dots, x_D$ must exist / Training same complexity as Autoencoder very large hidden layers necessary / computing $p(x)$ just forward pass

PixelRNN: $p(x) = \prod p(x_i | x_1, \dots, x_{i-1})$ (=likelihood of i-th pixel | given all previous pixels)
Sequential generation is slow due to pixel dependencies

PixelCNN: Dependencies on previous pixels as CNN over context region

Reinforced Learning: · States S / Actions A / reward $r(S,A) \to R$ / transition $p(S,A) \to S$ / Init s_0
Value function of policy π: $V_\pi: S \to R$, expresses expected cumulative reward
Bellman equation: $G_t = \sum_a \pi(a|s) \cdot \sum_{s'} \sum_r p(s', r | s, a) \cdot (r + \gamma \cdot E_\pi(G_{t+1} | S_{t+1} = s'))^*$
↳ Prob. of taking an action given state ↳ joint prob. of next state and reward given current state and action (Transition matrix)
*expected reward given a new state

Dynamic Programming: Value iter: 1. Compute optimal state value function V_*
2. Obtain policy via backtracing
Policy iter: 1. Compute state value function V_π for arbitrary policy π
2. Update policy π given $V_\pi \to \pi'$ => Iterate till $\pi = \pi'$
r(s,a): reward or cost
$V_\pi(p(s,a))$: cost value in neighborhood
↳ Update step for dyn. prog. step: $V'(s) = \text{argmax}(r(s,a) + \gamma \cdot V_\pi(p(s,a))$
+: exact, guaranteed to converge / -: know transition prob. matrix, iterate over whole state space

Temporal difference learning: If state space is too big / Random policy: exploration
greedy policy: exploitation, can get stuck in local minima / $\Delta V(s) = r(s,a) + \gamma V(s') - V(s)$
Update only visited state spaces, use ε-greedy-strategy / $V(s) \leftarrow V(s) + \alpha \cdot \Delta V(s)$ } α: Learning rate
Q-Learning: $\Delta Q(s,A) = R_{t+1} + \gamma \cdot \max\{Q(s',a)\} - Q(s,A)$ / $Q(s,A) \leftarrow Q(s,A) + \alpha \cdot \Delta Q(s,A)$
Off-policy: policy to update Q-function different from policy we use to collect data (= ε-greedy)

Tabular Learning: Learn each state separately / no generalization => use function approx. to learn value func.

Deep Reinforced Learning: Use neural network to learn mapping state-action pairs (s,a) and value
$\text{Loss}(\theta) = (R + \gamma \max\{Q_\theta(s',a')\} - Q_\theta(s,A))^2$ / Q-Networks: samples need i.i.d => Replay Buffer => SGD

Actor critic: Builds on policy gradient methods. Instead of using rewards, a value function from NN is used as an estimate

$\frac{\partial L}{\partial W} = \sum_{t=1}^{t} \frac{\partial L^t}{\partial W}$ On policy: compute Q-value according to policy and agent follow policy
$\frac{\partial L^t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L^t}{\partial y^t} \cdot \frac{\partial y^t}{\partial h^t} \cdot \frac{\partial h^t}{\partial h^k} \cdot \frac{\partial h^k}{\partial W}$
off-policy: Q value greedy but agent different exploration policy

$\frac{\partial \tanh(x)}{\partial x} = (1 - \tanh^2(x)) \cdot x'$ $\quad \frac{\partial \sigma(x)}{\partial x} = (\sigma(x) \cdot (1 - \sigma(x))) \cdot x'$

$\log(x)' = 1/x$