DL Introduction

Postulates for PR:
1. Availability representative sample ω of patterns $f(x)$, field pro. Ω; $\omega = \{^1f(x), \dots, ^Nf(x)\} \in \Omega$
2. Simple pattern has features, char. membership in class Ω_k
3. Compact domain same class, separable different class
4. (complex) pattern consists of simpler constituents
5. pattern $f(x) \in \Omega$ has certain structure
6. similar if features/constituents differ only slightly

Perceptron: $\hat{y} = \text{sign}(w^T x)$, $y \in \{-1, 1\}$, $S = \{x_1 y_1, y_1 \dots\}$ training set
M set with $y_i \neq \hat{y}_i = \text{sign}(w^T x_i)$
$\arg\min \{D(w) = -\sum_{x_i \in M} y_i \cdot (w^T x_i)\}$
$\nabla D(w) = -\sum_{x_i \in M} y_i \cdot x_i$, $w^{(k+1)} = w^{(k)} + y_i \cdot x_i$

Feed Forward Neural Networks
XOR problem cannot be solved, a single hidden layer already universal function approx.
Universal approximation theorem: Let $\varphi(\cdot)$ non-constant, bounded, mono. increasing. For any $\varepsilon > 0$, f continuous defined on compact subset, $\exists N$, const. $v_i, b_i \in \mathbb{R}$ and $w_i \in \mathbb{R}^m$ with
$F(x) = \sum_{i=1}^{N} v_i \varphi(w_i^T x + b_i)$ with $|F(x) - f(x)| \le \varepsilon$
→ approx with just one hidden layer

Softmax: $\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^{K} \exp(x_j)}$
Properties: 1. $\sum_{k=1}^{K} \hat{y}_k = 1$ 2. $\hat{y}_k \ge 0 \ \forall \hat{y}_k \in \hat{y}$ Kolmogorov
Cross entropy: $H(p,q) = -\sum_k p_k \log(q_k)$
Loss function: $L(y, \hat{y}) = -\sum_{k=1}^{K} \log(\hat{y}_k)|_{y_k=1}$
Softmax loss: Substitute $\hat{y}$

Optimization network $L(w, x, y)$
$\mathbb{E}_{x,y \sim \hat{p}_{data}(x,y)}[L(w, x, y)] = \frac{1}{M} \sum_{m=1}^{M} L(w, x, y)$
Target: $\min_w \{L(w, x, y)\}$
Gradient descent: $w^{k+1} = w^k - \eta \nabla_w \frac{1}{M} \sum_{m=1}^{M} L(w, x, y)$
Complex network: $L(w, x, y) = L(g(f(x, w_f), w_g), y)$
$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \to 0} \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h}$
Chain rule: $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \cdot \frac{d}{dx} g(x)$

Feedback loop: 
$-\eta \hat{y}$ → pos. feedb. → loss ↑
$-\eta L$ → neg. feed b. → gradient vanishes

Activation functions
- Sign: $f(x) = \begin{cases} +1 & x \ge 0 \\ -1 & x < 0 \end{cases}$, $f'(x) = 2\delta(x)$
+ Normalized output − Gradient vanishes
- Sigmoid: $f(x) = \frac{1}{1+\exp(-x)}$, $f'(x) = f(x) \cdot (1 - f(x))$
+ Normalized output − Gradient vanishes
- ReLU: $f(x) = \max(0, x)$, $f'(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases}$
+ Less vanishing gradient

Fully connected: $\hat{y} = Wx$
$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W} = \frac{\partial L}{\partial \hat{y}} x^T$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial x} = W^T \frac{\partial L}{\partial \hat{y}}$

Loss functions
Classification: Estimate a discrete variable for every input
Regression: [...] continuous [...]
Maximum Likelihood Estimation
Given $X = x_1, \dots, x_M$, $Y = y_1, \dots, y_M$, $p(y|x)$
$Y = \prod_{m=1}^{M} p(y_m|x_m) \to \max_w \{\dots\}$
$\min_w \{-\ln(L(w))\}$
Regression: $\{\sum_{m=1}^{M} \ln(p(y|M|x_m|w))\}$
$p(y|x, w, \beta) = N(\hat{y}(x, w), \frac{1}{\beta})$ (Gaussian: hypothesis model)
$= \frac{\sqrt{\beta}}{\sqrt{2\pi}} \exp(\beta \cdot -\frac{(y_m - \hat{y}(x_m, w))^2}{2})$
$-\ln(L(w)) = \frac{M}{2} \ln(2\pi) - \frac{M}{2} \ln(\beta) + \frac{\beta}{2} \sum_{m=1}^{M} (y_m - \hat{y}(x_m, w))^2$
L^2-loss: $\frac{1}{2} \sum_{m=1}^{M} ||y_m - \hat{y}(x_m, w)||_2^2$ (dep. on w)

Classification
$p(y|p) = \begin{cases} p^y (1-p)^{1-y} & \text{if } y \in \{0,1\} \\ 0 & \text{otherwise} \end{cases}$
$C(y|p) = \begin{cases} \prod_{k=0}^{K} p_k^{y_k} & \text{if } y_k \in \{0,1\} \\ 0 & \text{otherwise} \end{cases}$
$p(y|x, w) = C(y|\hat{y}(x, w))$
$L(w) = -\sum_{m=1}^{M} \sum_{k=0}^{K} y_{k,m} \ln(\hat{y}_k)$
$= \sum_{m=1}^{M} \ln(\hat{y}_k(x_m, w))|_{y_{k,m}=1}$ (cross entropy)

Kullback Leibler:
$KL(p, q) = \int_{-\infty}^{+\infty} p(x) \ln \frac{p(x)}{q(x)} dx$
$= \int_{-\infty}^{+\infty} p(x) \ln p(x) dx - \int p(x) \ln q(x) dx$
−Entropy $H(p)$, Cross Entropy $H(p,q)$
$\Rightarrow -\sum_{k=0}^{K} \ln(\hat{y}_k^{y_k}) = H(y, \hat{y})$

Hinge loss:
$L(w) = \sum \max(0, 1 - y_m \hat{y}(x_m, w))$
Subgradient: $f(x) \ge f(x_0) + \nabla f(x_0)^T (x - x_0)$
Support Vector Machine:
$\min \frac{1}{2} ||w||_2^2 + \gamma \sum \xi_m$, $\forall m: -\xi_m \le 0$
s.t. $\forall m: -(y_m \cdot (w^T x_m) - 1 + \xi_m) \le 0$
→ Lagrangian dual function, $\lambda_m \ge 0$
$L(w) = \frac{1}{2} ||w||_2^2 + \gamma \sum_{m=1}^{M} \max(1 - y_m (w^T x_m))$ (L_2 regularizer, Hinge loss)
Outliers: $L(w) = \sum_{m=1}^{M} (\max(0, 1 - y_m \hat{y}(x_m, w)))^2$
Multi-class: $L(w) = \sum_{m=1}^{M} \sum_{k \neq l} \max(0, 1 - \hat{y}_l(x_m, w) + \hat{y}_k(x_m, w))$

Optimization
- Batch Gradient Descent: all M samples, preferred convex, decrease error
- Stochastic (Online) Gradient Descent: Use 1 sample → no decrease, inefficient
- Mini-Batch SGD: Use $B \ll M$ samples
$g^{(k)} = \nabla L(w^{(k)}) = \frac{1}{B} \sum_{b=1}^{B} \nabla L(w^{(k)}, x_b)$
$w^{k+1} = w^{(k)} - H(L(w^{(k)}))^{-1} \nabla L(w^{(k)})$ (too expensive)

Momentum:
$v^{(k)} = \mu \cdot v^{(k-1)} - \eta \nabla L(w^{(k)})$, $w^{(k+1)} = w^{(k)} + v^{(k)}$ (Momentum)
Nesterov Accelerated Gradient (NAG) / Nesterov Momentum: "look ahead"
$v^{(k)} = \mu v^{(k-1)} + \eta \nabla L(w^{(k)} + \mu v^{(k-1)})$
$w^{(k+1)} = w^{(k)} + v^{(k)}$ (approx. of next position)
Alternative: $w^{(k+1)} = w^{(k)} - \mu v^{(k-1)} + (1+\mu) v^{(k)}$

AdaGrad, RMSProp, Adadelta, Adam
→ Adaptive Gradient, Adaption based on past squared gradients
SGD / Momentum: + often conv. most, + still used, − tuning rate needed
Adam: [...] learning rate [...] → Adaptive moment estimation
Adam: $g^{(k)} = \nabla L(w^{(k)})$
$v^{(k)} = \mu v^{(k-1)} + (1-\mu) g^{(k)}$
$r^{(k)} = \rho r^{(k-1)} + (1-\rho) g^{(k)} \odot g^{(k)}$
$w^{(k+1)} = w^{(k)} - \eta \frac{\hat{v}^{(k)}}{\sqrt{\hat{r}^{(k)}} + \varepsilon}$
BIAS corr.: $\hat{v}^{(k)} = \frac{v^{(k)}}{1-\mu^k}$, $\hat{r}^{(k)} = \frac{r^{(k)}}{1-\rho^k}$
ADAM (−AdaGrad, RMSProp), Momentum + every ordinary
AMSGrad: Adam not conv. → problems w. increasing step size
AMSGrad fix: $g^{(k)} = \max(r^{(k-1)}, r^{(k)})$

Activation functions
- Linear activation function does not introduce non-linearity
- Sigmoid saturates $x \ll 0$ and $x \gg 0$. Not zero-centered
- Mean $\mu = 0$ always shifted to $\mu > 0$ → co-variate shift of successive layers → layers constantly have to adapt shifts → Batch Normalization
- Tanh: $f(x) = \tanh(x)$, $f'(x) = 1 - f(x)^2$; $\tanh(x) = 2\sigma(2x) - 1$
- Large changes in x → minimal changes in y → gradient vanishes

ReLU / Dying ReLU: Negative values for any x → $x \le 0$! No more updates because $f'(x) \to 0$ → "dead"
- Leaky ReLU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$, $f'(x) = \begin{cases} 1 & x > 0 \\ \alpha & \text{else} \end{cases}$
- Exponential Linear Unit: else $= \alpha(\exp(x) - 1)$ | $f'(x)$ else $= \alpha \exp(x)$
- Scaled Exponential Linear Unit: Multiply ELU with λ

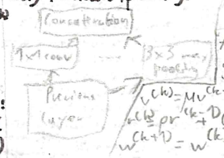
Convolutional Neural Networks
- Convolution: $(f * g)(x) = \int_{-\infty}^{\infty} f(\tau) g(x - \tau) d\tau$
- Cross-correlation: $(f \star g)(x) = \int_{-\infty}^{\infty} \bar{f}(\tau) g(x + \tau) d\tau$
- Cross-correlation is convolution with a flipped kernel g
- Padding: Convolution reduces image size by $2 \cdot \lfloor \frac{n}{2} \rfloor$ (n: kernel size)
- Stride: in $X \times Y \times S$, filter $M \times N \times S$; out: $X \times Y \times H$ (same padding)
- Convolution with Toeplitz matrix → Same formulas as for FCL: $E_{l-1} = W^T E_l$, $\nabla W = E_l X^T$
- Strided convolutions: add offset between measurements
- Dilated / Atrous Convolutions: skip certain pixels
- 1×1 convolution concept: just use fully connected property in depth dimension → decrease size of network, learns dimensionality reduction

Pooling layers
- Max-Pooling: forward simply propagate maximum value / backward only the value contributed to the error
- Average-Pooling: propagate average / backward share error to equal parts

⇒ CNN: Architecture
- Conv. Layer: Feature extraction
- Activation Function: Nonlinearity
- Pooling Layer: Compress and aggregate information, save parameters
- Fully connected for classification

↳ Alternatively and equivalently: Use flatten & 1×1 convolution or N×N convolution → enables arbitrary input size with global pooling

Inception Model: Combine different neighbourhoods / pooling / etc.



BIAS Variance Decomposition
$h = h(x) + \epsilon$, $\epsilon = N(0, \sigma_\epsilon)$
$\hat{y}_D = \hat{f}(x|D)$, $\mathbb{E}_D(L(x)) = \mathbb{E}_D[(h - \hat{y}_D)^2]$
$= (\mathbb{E}_D[\hat{y}_D] - h(x))^2 + \mathbb{E}_D[(\hat{y}_D - \mathbb{E}_D[\hat{y}_D])^2] + \sigma_\epsilon^2$
Bias, Variance, Irred. Error
Bias & Variance → Model capacity
Vapnik-Chervonenkis measure

Variance reduction techniques
- Early stopping: Use parameters with minimal validation loss
- Data augmentation: Artificially enlarge dataset, random spatial transformations (affine, elastic), pixel (resolution, noise, pixel distribution)
- Regularized loss function: Bayesian
$p(w|y, X) = p(w|X) p(y|X, w)$
$p(w|y, X) = \frac{p(y|X, w) p(w)}{p(y|X)} = \frac{p(y|X, w) p(x) p(w)}{p(y|X) p(x)}$
$MAP(w) = \max_w \{p(y|X, w) p(w)\}$ (Maximum a posteriori estimation; Prior)
MLE estimator
$\tilde{L}(w, X, y) = L(w, X, y) + \lambda ||w||_2^2$ | $||w||_1$
$\min \{L(w, X, y)\}$ s.t. $||w||_2^2 \le \alpha$
$w^{(k+1)} = (1 - \eta\lambda) w^{(k)} - \eta \frac{\partial L}{\partial w^{(k)}}$
Shrinkage / $\text{sign}(w^{(k)})$ (L_1 sparsity)
Enforce norm below a maximum: $\min \{L(w, X, y)\}$ s.t. $||w|| \le \alpha$, $\alpha > 0$
→ perform parameter update / project to unit "norm"
→ kind of shrinkage, prohibits / hides exploding

Normalization · Data: MinMax or variance, only training data, Normalization input data / within network
- Batch Normalization:
$\tilde{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma_{B,i}^2 + \epsilon}}$, $\hat{y}_i = \gamma_i \tilde{x}_i + \beta_i$ (Parameters)
μ, σ from training set → co-variate shift
BN improves L-Lipschitz, e.g. $|f(x_1) - f(x_2)| \le L||x_1 - x_2||$
Generalization: $\hat{y} = f(g \frac{x - \mu}{\sigma} + b)$ (weights) calc μ, σ over batch / layer, spatial dims, group, ...
- Self Normalizing Neural Network: addresses stability problem SGD, SeLU + weight
$f(x) = \lambda \begin{cases} x & x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$
drop-out, drop to $-\lambda \cdot \alpha$

Dropout / Dropconnect · Randomly set activations to zero with prob. $1-p$
- Test-time: multiply with p
Dropconnect: Generalizes dropout

Initialization · Simple: Bias to 0 / 0.1 for relu, weights random
- Calibrating: $\hat{y} = Wx = (\sum W_n x_n) + b$
x_n, w_n IID: $Var(\hat{y}) = N Var(w_n) Var(x_n)$
- Xavier: $\sigma = \sqrt{\frac{2}{fan_{in} + fan_{out}}}$ (scales var.)
- He: $\sigma = \sqrt{2 / fan_{in}}$ (+ReLU) $N(0, \sigma)$

Transfer learning · Reuse dataset (e.g. medical) · weight transfer early layers (convolutional)
1. fixed $\eta = 0$ 2. fine-tuned

Multi-task learning: Learn simultaneously, introduces inductive bias, reduces overfitting
- Hard parameter sharing: Several hidden layers shared between tasks, additional task-specific output layers | Soft: Each model own parameters, distance between parameters regularized loss function
- Auxiliary tasks: tasks own purpose or just auxiliary to original task, certain features difficult to learn for one but easy for another

Training strategies
- Keep test data in vault
- Gradient check: centered difference numeric gradients, relative error
- Check initialization and loss, compare chance accuracy / initialization
- Check in general capable by overfitting
- Monitor weights (e.g. dying ReLUs)
- Annealing learning rate (learning rate decay), strategies:
  - stepwise decay (factor/value)
  - Exponential decay: $\eta = \eta_0 e^{-kt}$
  - 1/t-decay: $\eta = \eta_0 / (1 + kt)$

Hyperparameter optimization regarding Architecture, Optimization, Regularization
- Search using log scale, grid search or random search

Ensembling Multiple classifiers independent
→ pred. $1-p$, k errors $\binom{N}{k} p^k (1-p)^{N-k}$
→ prob. majority $k > \frac{N}{2}$: $\sum_{k > N/2}^{N} \binom{N}{k} p^k (1-p)^{N-k}$

Class Imbalance
- Undersampling: Take subset for overrepresented data
- Oversampling: Use sample from underrepresented data / class multiple times
Undersampling causing underfitting can be reduced by taking a different subset, data augmentation can help ↓ overfitting
- Adapt loss function, e.g. $L(y, \hat{y}) = -w_k \log(\hat{y}_k)|_{y_k=1}$

Evaluation · Data annotated / labeled by humans, ambiguous data. Take mean / majority vote
- Performance measures: TP/FP/TN/FN
$ACC = \frac{TP + TN}{P + N}$, precision $= \frac{TP}{TP + FP}$ (pos. predictive value)
recall $= \frac{TP}{TP + FN}$ (true positive value), specificity $= \frac{TN}{TN + FP}$ (true negative value)
$F1 = 2 \cdot \frac{TPV \cdot TNV}{TPV + TNV}$
- Multiclass classification: Top-K error. True class label not in the K classes with highest prediction score
- Cross validation: k-fold cross validation: split data in k folds, use k−1 folds as training data, test on fold k, repeat k times
- Comparing classifiers: Is new method better? Run method multiple times / use student's t-test:
→ means significantly different with respect to a significance level α
→ Bonferroni Correlation: Probability that difference is caused by chance < α
→ Multiple comparisons
→ For n tests n·α, choose α' = α/n for each individual test
→ Assumes independence, pessimistic
→ More accurate: Permutation tests

AdaGrad: $r^{(k)} = r^{(k-1)} + g^{(k)} \odot g^{(k)}$ (individual learning rate)
RMSProp: $r^{(k)} = \rho r^{(k-1)} + (1-\rho) g^{(k)} \odot g^{(k)}$ (still learning rate)
$g^{(2)} = \Delta x$ → $\Delta x = \dots \odot g^{(k)}$
AdaDelta: no learning rate

Datasets Object Detection / Image Class.
- ImageNet dataset (1000 classes, 14 Mio. images, varying size) → 50k test, 10k test
- CIFAR 10/100 (classes, 32×32 img.)

Early architectures
Le-Net · Sequence: Convolution, pooling, non-linearity (tanh), Subsampling via average pooling, MLP as final classifier
Alex-Net · 8 layers, 6 P.U., overlapping max pooling, ReLU, Dropout w. p = 0.5 in first two FC layers, data augmentation
- learning: mini-batch SGD + L2 weight decay, weight initialization $N(0, 0.01)$
Network in Network · 1×1 filters (convolutions), global (spatial) average pooling as last layer (less prone to overfitting FC)
VGG Network · Small kernel sizes in conv.
GoogleNet · Inception models v1

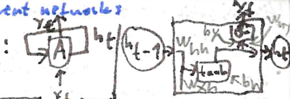
Deeper models · Inception-v2: replace 7×7, 5×5 convolutions with 3×3 conv. grid size reduction, average pooling, softmax
- Inception-v3: ½ RMS Prop, batch-normalization also in FC layers, label-smoothing reg.
- Label-smoothing regularization: Standard label distribution: $q(k|x) = \delta_{k,y}$
→ Softmax predict 0/1, (activation) $|x| \to \infty$
→ Label-smoothing: Exchange label distrib. with $q'(k|x) = (1 - \epsilon)\delta_{k,y} + \epsilon u(k)$ with $u(k) = 1/K$, $\epsilon = 0.1$
+ Prevent hard probabilities without discouraging correct classification

Residual units simplify "identity sol." $H(x) = F(x) - x$ ⟺ $F(x) = H(x) + x$

$x_{l+1} = h(g(x_l) + H_l \sum (x_l, \{w_{l+1,k}\}))$
Original: g: identity, h: ReLU
Better: g and h identity → pre-activation
- Ensemble View: $x_{l+1} = x_l + H_{l+1}(x_l)$
→ 2^n paths, at neuron level many different paths varying lengths
- Representation View
(a) Single layer: direct computation (b) Residual Network: unrolled iterative estimation (c) Classic network: produces new rep. each layer
- Stochastic depths: Layer-wise dropout → short during training → decreased time
- Inception-ResNet: combination of both
- Wide Residual Networks: decrease depth, increase width of ResNet blocks
- ResNeXt: agg. residual transformations
- ResNet-of-ResNets
- DenseNets: Densely connected convolutional networks; layer input: feature-maps of all preceding layers; feature propagation / reuse, alleviates vanishing gradient problem
- SENet: Squeeze-and-Excitation networks: Motivation: Explicitly model channel inter-dependencies
- Idea: Add trainable model that allows rescaling of channels dep. on input
- Squeeze: Compress each channel into one value (global avg. pooling)
- Excitation: FC layers & sigmoid to achieve scaling vector
- Scale: Scale input feature maps

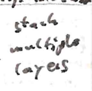
Learning Architectures Goal: Self-developing network structures
Accuracy, FLOPs (optimization)
Grid-search typically too time-consuming
- RNN to generate model descr. of networks, train with reinforcement learning to max. expected accuracy

Recurrent networks
Simple RNN unit



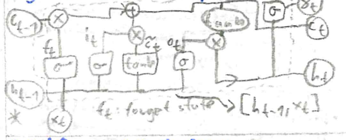
$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_h)$
$\hat{y}_t = \sigma(W_{hy} \cdot h_t + b_y)$
One-to-one: image class. One-to-many: captioning
Many-to-one: sentiment anal. Many-to-many: video class.

Deep RNN: $h^l(t) = \tanh(W_{xh}^l \cdot x_t + W_{hh}^l \cdot h_{t-1}^l + b^l)$ stack multiple layers

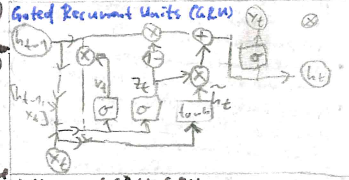


Characters: Encode as one-hot encoded vectors, e.g. $h = (1, 0, 0, 0)$
Backpropagation through time (BPTT)
$L(\hat{y}, y) = \sum_{t=1}^{T} L(\hat{y}_t, y_t)$, $\theta = [\nabla W_{xh}, \nabla W_{hh}, \nabla b_h, \nabla b_y, \nabla W_{hy}]$, $\theta = \theta - \eta \nabla \theta$
$\frac{\partial L}{\partial h_t} = \nabla h_t = \frac{\partial L}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} + \frac{\partial L}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t}$
→ For each gradient simply sum over all time steps
Normal BPTT: backprop. complete sequence
Naive BPTT: split sequence in small batches
T (Truncated) BPTT: Every k_1 steps, run BPTT for k_2 time steps → keep processing whole sequence
(Long dependencies) vanishing gradients problem

Long Short-Term Memory Units (LSTMs)



Gated Recurrent Units (GRU)



Differences LSTM, GRU
LSTM: Separate hidden and cell state, controlled exposure of memory through output gate, memory content independent from gates: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$
GRU: Combined hidden / cell state, full exposure of memory content without control, new memory depends on curr. memory: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

Sampling Strategies for RNNs
- RNN computes prob. distribution of next ele.
- Greedy search: At each point, pick the most likely element (no lookahead)
- Beam search: Select k most likely elements; out of all possible sequences that have one of these k elements as a prefix take k most probable ones
- Random sampling: Sample next word according to output probability distribution / temperature sampling: $\tilde{p}_i = f_\tau(p_i) = \frac{p_i^{1/\tau}}{\sum_j p_j^{1/\tau}}$

* Forget gate: Forgetting old information in cell state, Input gate: Deciding on new input for cell state ($\tilde{C}_t$)
⊗ Reset gate (r_t): Influence of previous hidden state, Update gate (z_t): Influence of newly computed update
$r_t \to$ short term, $r_t \to 0$: ignore prev. hidden state; $z_t \to$ long term, $z_t \to 0$: ignore input

+ Deeper networks: Exponential feature reuse / increasingly abstract features

Visualization – Architecture
- Node-link diagrams: Neurons = nodes, (weighted) connections = edges
- Block diagrams: layer = solid block, single connection between layers = values
- "Others", e.g. Deep Visualization

Training integrating information: input data, parameters, hidden layer data, output data → debugging, improve model design...

Parameters Motivation: confounders, network learns correlated features, need to identify the task, "Learnative" Behavior of Neural Networks
- Adversarial Examples and Optical Illusions ... Kernels ...

– Simple parameter visualization:
- plot learned filter weights directly
- Investigating features via Occlusion (if occlusions causes drop, predicted area important for classification)
- Maximally Activating Images (→ Image patches specific neuron most ...)
- t-SNE visualization of CNN codes, while images regarded as "similar" → t-distributed stochastic Neighbor Embedding → performs dimensionality reduction → 2d-map of images

– Gradient-Based Visualization
- Backpropagation for Visualization: Which pixels are significant for neuron, e.g. for which pixel x_i is $\frac{\partial \text{neuron}}{\partial x_i}$ large? use backpropagation
- Feature visualization: Deconvnet; choose one activation, set other to 0
- Guided Backpropagation: Improve results by "guiding" the backprop. → set negative gradients to zero
- Saliency maps: Investigate impact of pixels on class score

– Parameter visualization via Optimization
- Objectives: Neuron, layer, logits ...
- Google DeepDream: Inceptionism: what "dreams" DNN with images → tweak image towards high activation of a complete layer → find L_2-regularised image x: $\max_x f(x) - \lambda\|x\|_2^2$
- Inversion: attempt construct img from given layer activation $\hat{f}$: $x^* = \min_x (\|f(x) - \hat{f}\|_2^2 + \lambda r(x))$
- Regularizer: High frequency penalization, Transformation robustness, learned priors

Attention Mechanisms Allows follow one thought at the while suppressing irrelevant information to the task
- Implicit attention learned
- Sequence-to-sequence models: hidden state, Encoder network → h → Decoder network, context vector
- Seq2Seq: dynamic context vector as combination of all prev. hidden states
→ Encoder: Bidirectional LSTM with $h_i = [\overrightarrow{h}_i; \overleftarrow{h}_i]$
→ $c_t = \sum_i \alpha_{ti} h_i$
→ $\alpha_{ti} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum \dots}$

$\text{score}(s_t, h_i) = v_a^T W_a [s_t; h_i]$

Alignment: which input is important to which output

- Soft attention: computed over all glimpses
- Hard ": Fixed size glimpse over input
→ Show-attend-tell: image captions
→ Self-attention: attention to itself
- Attention is all you need (AIAYN): Transformer only with attention, representation by self-attention
→ input taken translated into vector same length via embedding alg.
→ compute query q (what token looking for), key k (description for q), value v (info)
→ $\text{attention}(q,k,v) = \text{softmax}\left(\frac{qk^T}{\sqrt{d}}\right)v$
→ Encoder: Multi-Head attention, data representation subspaces, recomb FCL, stacked attention blocks, similar decoder

Sequential Decision Making
- Multi-Armed Bandit Problem: action a_t at time t from set A, a_t has pdf $p(r|a)$ with reward r_t, choosing a as pdf $p(a)$ is policy → find max $E[p(r|a)]$
$E[p(r|a)]$ not known in advance, $Q_t(a)$ action-value function $\frac{1}{t}\sum_{i=1}^t r_i = Q_t(a)$

$Q_{t+1}(a) = Q_t(a) + \frac{1}{t+1}(r_t - Q_t(a))$

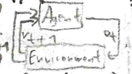
Best policy $\pi(a) = \max Q_t(a)$

Uniform Random: $\pi(a) = 1/|A|$

Epsilon Greedy: $\pi(a) = \begin{cases} 1-\epsilon & \text{if } a = \arg\max Q_t \\ \epsilon/(|A|-1) & \text{else} \end{cases}$

Softmax: $\pi(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{|A|} e^{Q_t(b)/\tau}}$, τ is temperature

Reinforcement learning · Associativity
- State s_t, reward $p(r_t|s_t, a_t)$, actions influence $p(s_{t+1}|s_t, a_t)$
- Markov Decision Process
– action a_t from the s_t



– s_{t+1}, pdf $p(s_{t+1}|s_t, a_t)$
– reward $r_{t+1} \in R \subset \mathbb{R}$ to $p(r_{t+1}|s_t, a_t)$
– action a_t by policy $\pi(a|s)$, finite MDP
- Policy: 1. episodic tasks with end, 2. continuing tasks indefinitely long
→ Unify with terminal state in episodic tasks
– Maximize future return: γ is discount

$\max g_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$, $\gamma \in [0,1]$

- Policy iteration: Use oracle predicting future reward g_t following $\pi(s_t, a_t)$ from s_t
– State-value function $V_\pi(s)$

$V_\pi(s) = E_\pi[g_t|s_t] = E_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \,\middle|\, s_t\right]$

– Action-value-function $Q_\pi(s,a)$

$Q_\pi(s,a) = E_\pi[g_t|s_t, a_t] = E_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \,\middle|\, s_t, a_t\right]$

Only one optimal $V^*(s)$ (in finite MDP)

$V^*(s) = \max_\pi V_\pi(s)$

$Q^*(s,a) = E[r_{t+1} + \gamma V^*(s_{t+1})]$

– Optimal Policies: $\pi \geq \pi' \Leftrightarrow V_\pi(s) \geq V_{\pi'}(s)$
Any policy π with $V_\pi = V^*$ optimal π^*
Only one state-value function, but maybe multiple state-value functions
– Bellman Equation for $V_\pi(s) = \sum \pi(a|s) \sum p(s_{t+1}, r|s, a)[r + \gamma V_\pi(s_{t+1})]$
better: iteratively solve $V_{k+1}(s)$
- Policy Improvement Theorem:
$Q_\pi(s, \pi'(s)) \geq V_\pi(s), \forall s \in S \Rightarrow \pi' \geq \pi$
implies $V_{\pi'}(s) \geq V_\pi(s)$, $Q_\pi(s,a) \geq V_\pi(s)$

Monte Carlo Techniques only episode tasks
learn $V(s)$ by arbitrary $\pi(s|a)$
– Generate episode, loop backwards, g_{t+1}
if not visited, add to returns(s_t)

$V_{s_t} = \frac{1}{N}\sum_{n=1}^N \text{returns}_n(s_t)$

– Temporal difference learning

$V_{t+1}(s) = V_t(s) + \alpha(r_t + \gamma V_t(s_{t+1}) - V_t(s))$ On-policy

Temporal difference: Loop and follow $\pi(s_t, a_t)$

- Policy improvement: If we don't loop over all $s \in S$ for policy evaluation, but update policy directly algorithm called Value iteration

Q learning: $Q_{t+1}(s,a) = Q_t(s_t, a_t) + \alpha[r_t + \gamma \max Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$
- Universal function approximators: policy gradient REINFORCE, $\pi(s_t, a_t, w)$

$w = w + \eta \gamma^t g_t \nabla_w \ln(\pi(a_t|s_t, w))$

Applications · Deep Q learning: Deep network applies a (learning) Q_t to output, second target network fixed weights, random samples for training / replay memory
- AlphaGo: MCTS: Selection, Expansion, Simulation, Backup; Tree expansion and value estimation by neural network
AlphaGo Zero: Solely trained with reinforcement learning, simpler MCTS, shared layers

Unsupervised Deep Learning
- Weakly supervised: label for related task
- Semi-supervised: partial labeled data
- Unsupervised: no labeled data
- Applications: Dimensionality Reduction, Network Initialization, Representation learning, Generative models

To deconvolute: Forward pass is eff. backward pass (without ReLU)

To regularize: (a) degrade reconstruction; (b) Input Invar. Spatial transformations (+ simple, + effective, − orientation suppressed); (c) Enforce prior on x enforcing x look like natural images via GAN

Autoencoder
Encoder: $y = f(x)$; Decoder: $\hat{x} = g(y)$

Loss $= L(x, x') = -\log p(x|x')$

Squared L2 norm: $p(x|x') \sim N(x', \sigma^2)$

$L(x, x') = \|x - x'\|^2$

Cross entropy: $p(x|x') \sim B(x')$

$L(x, x') = -\sum_i x_i \log(x'_i) + (1-x_i)\log(1-x'_i)$

Undercomplete AE (Prevent network to simply copying the input), Sparse AE (more units, bottleneck by enforcing sparsity in the activations)

$L_{SAE}(x, x', w) = L(x, x'; w) + \lambda|y|$

Convolutional AE

- Denoising AE: Corrupt input, $C(\tilde{x}|x) \sim N(x, \sigma^2 I)$

DAE as generative model, implicitly estimate the underlying data generating process $C(\tilde{x}|x)$, DAE learns $p(x|\tilde{x})$ if x typical samples it. applying noise and denoising will reproduce sample very often | Estimator $p(x)$: Markov chain sampling by alternating denoising model and corruption proc. $C(\tilde{x}|x)$, converging MC yields estimator $p(x)$

- Stacked AE: − Option 1: Train layer by layer − Option 2: Train by means of multiple layers

Variational AEs: Traditional AE deterministic − learn latent space / probabilistic manner, sample. Statistical motivation: Approx. (intractable) $p(z|x)$ by $q(z|x)$, Minimize KL

$\min KL(p(z|x), q(z|x)) = \max \mathbb{E}_{q(z|x)}[\log p(x|z)] - KL(q(z|x), p(z))$

reconstr. likelihood, $q(z|x) \approx p(z)$

→ $p(z)$ often Gauss. distrib., $q(z|x)$ μ and σ | $x \to q(z|x) \to z \to p(x|z) \to x'$
– Training: Loss function $L(\theta, \phi, x, z)$
Reparametrization: $\partial f/\partial z$, $z = \mu + \sigma \odot \epsilon$, $\epsilon \sim N(0,1)$
– as given model
– Only models lower bound of likelihood

Generative Adversarial Networks
$D = D(\theta^D; x) \leftarrow G = (\theta^G; z)$

real $x \sim p_{data}$ real: 1 / fake: 0, Noise $z \sim p_z$

1. Train D, minimize $L^D(\theta^D, \theta^G) = -\mathbb{E}_{x\sim p_{data}} \log D(x) - \mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$
2. Train G: minimize $L^G = -L^D$ for every
Optional: k steps of one player for other
→ Equilibrium saddle point of discr. loss

Value function: $V(\theta^D, \theta^G) = -L^D(\theta^D, \theta^G)$

Minimax: $\theta^{G*} = \arg\min_{\theta^G} \max_{\theta^D} V(\theta^D, \theta^G)$

$L^D = 1/2$

Optimal discriminator: both densities $\neq 0$
→ Otherwise some input values never trained → some $D(x)$ undetermined behavior
· Solve for $\partial L^D / \partial D(x) = 0$
· Optimal $D^*(x)$ for $p_{data}(x)$, $p_{model}(x)$:

$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}$

Non-Saturating Games: fights vanishing gradient

$L^G = -\frac{1}{2}\mathbb{E}_{z\sim p_z} \log D(G(z))$

Loss functions: · Feature matching loss / perceptual loss (G trained to match expected value of features $f(x)$ of intermed. layer of D) valid, prevent overtraining

$L^G = \|\mathbb{E}_{x\sim p_{data}} f(x) - \mathbb{E}_{z\sim p_z} f(G(z))\|_2^2$

· Wasserstein: (Train D Lipschitz, max. discrepancy between real and fake and whose gradients remain beyond a limit:

$\max_{\|D\|_L \leq 1} \mathbb{E}_{x\sim p_{data}}(D(x)) - \mathbb{E}_{x\sim p_{model}}(D(x))$

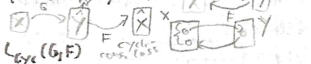
Evaluate GANs: · Inception score: Goals (Gen img recognizable, img diverse), KL dist. between distrib. (higher: better):

$IS(x) = \exp\{\mathbb{E}[KL(p(y|x)\|p(y))]\}$

· Fréchet Inception Distance (FID): use intermed. layer, model data distrib. by multivari. Gaussian (μ, Σ), $FID(x,g) = \|\mu_x - \mu_g\|^2 + Tr(\Sigma_x + \Sigma_g - 2\sqrt{\Sigma_x\Sigma_g})$

Conditional GANs provide y to encode conditioning, G: z & y, D: x & y, obj. function: $\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x|y)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z|y)))]$

Cycle Consistent GANs: Paired data missing: translation mappings $F(G(x)) \approx x$, $G(F(y)) \approx y$
· two discr. D_Y & D_X



$L_{cyc}(G,F) = \mathbb{E}_{x\sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y\sim p_{data}}[\|G(F(y)) - y\|_1]$

$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + \dots + \lambda L_{cyc}(G,F)$

More Tricks · One-sided label smoothing real samples 1 → 0.9, not with fake
· G/D balancing not necessary, when D too good: gradient vanish (→ non-saturating loss), D too large (→ label smoothing)
· Deep GANs: pooling → strided conv; G: ReLU, D: leaky ReLU
· Virtual Batch Normalization: at least two sep. BN for both minibatches, better: − create reference batch R − $\forall x_i$ of batch: draw virtual batch $V_i = R \cup x_i$ − compute mean and st.dev. of V_i − Prop R forward in addition to current batch → Normal BN results strong mini-batch correlation; fixed stat of training

Historical Averaging: Add penalty term for old weights $\|\theta - \frac{1}{t}\sum_{i=1}^t \theta[i]\|^2$

Advanced GAN methods · Mode collapse
G rotates through modes of data distrib., nonconv.: possible reason: $\min_G \max_D V(G,D) \neq \max_D \min_G V(G,D)$
- D-inner loops converge to correct distrib. – G: place all mass on most likely point
→ Solutions: · Minibatch discrimination
– Allow D look at multiple samples (no collapse)
1. Extract features from intermediate layer
2. add minibatch layer (similarity other samples, concatenate sim. vector to each feature)
D still outputs 0/1, but uses side info
- Unrolled GAN: Ideally: $G^* = \arg\min_G \max_D V(G,D)$
→ Essentially, we ignore max in G's gradients
→ Regard $\max_D V(G,D)$ as cost → backpropagate
1. Build comp. graph desc. k steps of learning D
2. Back-propagate through all k steps
- GANs for Semi-supervised learning: Turning problem from K class prob. to K+1 cla. prob.
→ True class = target class; additional: fake
- Multi-scale approaches: Laplacian pyramid of GANs: Subsequently increase resolution and add detail with pyramid of GANs
Input: noise + unsampled output as cond. variable, Output: upscaled image
- StackGANs: Text-to-photo image synthesis (Stage-1: low resolution, Stage-2: upscaling)

Semantic Segmentation
Each pixel gets semantic class / k+1 classes incl. void; p_{ij} amount pixel class $i \to j$ (pii TP) / Eval. Metrics
1. Pixel Accuracy (PA): $PA = \frac{\sum_{i=0}^k p_{ii}}{\sum_{i=0}^k \sum_{j=0}^k p_{ij}}$ (ratio correct classified)
2. Mean PA → per class: $MPA = \frac{1}{k+1}\sum_{i=0}^k \frac{p_{ii}}{\sum_{j=0}^k p_{ij}}$
3. Mean Intersection over Union (MIoU): $PA = \frac{1}{k+1}\sum_{i=0}^k \frac{p_{ii}}{\sum_{j=0}^k p_{ij} + \sum_{j=0}^k p_{ji} - p_{ii}}$ Ratio between intersection / union of two sets
4. Frequency weighted Intersection over Union: $PA = \frac{1}{\sum_{i=0}^k \sum_{j=0}^k p_{ij}}\sum_{i=0}^k \frac{\sum_{j=0}^k p_{ij}\, p_{ii}}{\sum_{j=0}^k p_{ij} + \sum_{j=0}^k p_{ji} - p_{ii}}$

Encoder: learn coarse features; Decoder: decode/map to pixel-wise predictions

Upsampling techniques: · Nearest neighbor upsampling (duplicate value of the element for region) · Bed of nails (one take, rest 0), using Max-Pooling Indices
- Transpose convolution: learnable upsampling (≠ deconvolution), 2 pixels in output for every 1 pixel in input → fractionally strided → transp. conv backward pass of normal convolution
Checkerboard artifacts: uneven overlap → avoid: appropriate kernel size, separate upsampling from conv to compute features

Integrating context knowledge local information crucial to achieve good pixel-level accuracy, integrate local & global info
- Long et al.'s Fully Convolutional Network
– Add links combining the final prediction and previous (lower) layers with finer strides – Additional 1×1 convolution after pooling layer, predictions added up
– DAG with „skip connections"
- SegNet: Upsampling & conv layers followed by softmax, upsampling corresponds to max-pooling layer in encoder
- U-Net: – Encoder: Contracting path to capture context – Decoder: Symmetric expansion path for localization
– Decoder parts: Upsampling of feature maps followed by 2×2 conv halves number of feature channels, concatenation with corr. cropped feature from contr. path

Additional approaches · Dilated convolutions skip input pixels, make receptive field grow exponentially while number of params grows linearly
- Stacked Hourglass Network: Hourglass vs. U-Net: Additional bottleneck layers, learned skip connections, constant feature dim
→ Stacked Network: Refinement cascade of the prediction with losses after each hourglass · Conditional Random Fields: Refine coarse RNN output using CRF, each pixel modeled as node in random field, captures long-range dependencies, fine local information

Advanced Topics · Adversarial networks for segmentation; optimize segmentor with hybrid loss function with two terms: 1. Usual pixel-wise multi-class cross-entropy for semantic segmentation 2. loss based on min-max game with Discriminator
– Given set of N training images x_n and a corresponding label maps y_n the loss function defined as: $l(\theta_s, \theta_a) =$

$\sum_{n=1}^N \underbrace{l_{mce}(s(x_n), y_n)}_{\text{multi-class cross-entropy}} - \lambda[l_{bce}(a(x_n, y_n), 1) + l_{bce}(a(x_n, s(x_n)), 0)]$

adv. loss → binary classific.
→ Multi-task learning with adversarial task

Object Detection
Location + Classification
1. Hypothesize bounding boxes 2. Re-sample selected boxes 3. Apply classifier
Early approaches: Face Detection (Haar Features), Adaboost, cascading classifier for fast rejection), Histogram of Oriented Gradients (HOG), Support Vector Machines

Modern Approaches Sliding window: Slide fixed size window over image + apply trained CNN to multiple scales
- Regional CNN: Proposal Selective Search
– Candidate objects by grouping pixels similar texture or color → Segmentation
– Generate region proposals: Selective Search
2. Classify content + refine bounding box
– For each region proposal: Warp to standard size; pre-trained CNN for feature extraction, linear SVM object classif., linear regr. BB
- Fast R-CNN: R-CNN requires full forward pass for each region → pass full image through network: Feature maps; class. CNN fixed input size, spatial pyramid pooling (ROI pooling) → Use SPP layer with one output map, better sample selection → Sample 128 ROIs uniformly random → instead: Sample from few images, 64 ROIs
- Faster R-CNN: Add regional proposal framework, defines k anchor boxes associated with different scale and aspect ratios, efficient multi-scale ability, 4k box parameters + 2k scores

Single-Shot-Detectors · YOLO (You only look once) algorithm: Combined bounding box prediction and classification in one network; each cell predicts
– B boun. boxes + confidence score
– Class confidence → CNN predicts $S \times S \times (5B + C)$ vals for C classes
- YOLO9000 improved version of YOLO
- Single-Shot Multi-Box Detector: similar to YOLO
- RefineNet Detector: Similar to U-Net:
→ Encoding: ResNet feature extraction
→ Decoding: FPN trained multiscaling
→ Skip Connections: Residual conns.
– Stages of Feature Pyramid Network (FPN) represents different scaling / uses Focal loss

Focal loss · Objectness binary, Bernoulli, usual cross-entropy: $CE(p_t) = -\log(p_t)$
- Modify to: $FL(p_t) = \alpha_t(1-p_t)^\gamma \cdot \log(p_t)$
- α_t imbalance weights, imbalance class freq., γ hyperparameter decreasing infl. of easy examples

Instance Segmentation detect different instances of the same class
- Mask R-CNN: 1. Region proposal 2. Classification, bounding-box regression and segmentation in parallel
→ Multi-task-loss: $L = L_{cls} + L_{box} + L_{class}$
– Segmentation independent of classification
→ Only binary loss computed for correct class
→ Second-stage network computed arbitrarily

Exercise
FC layer: $Wx = \hat{y}$

Backwards: $E_{n-1} = W^T E_n$ Transposed for more efficient memory layout

$W^{t+1} = W^t - \eta \cdot E_n x^T$

Softmax: $(\tilde{x}_k = x_k - \max(x))$

$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^N \exp(x_j)}$ $\quad E_{n-1} = \hat{y}\left(E_n - \sum_{j=1}^N E_{n,j}\hat{y}_j\right)$

Crossentropy-loss $\sum_{k=1}^B -\ln(\hat{y}_k + \epsilon)|_{y_k=1}$ $\quad E_n = -\frac{y}{\hat{y} + \epsilon}$

Conv: backward pass also conv but with spatial-flipped filters → instead cross correlation can be used

$[a,b,c][x,y] = [a_y a x + b_y b x + c_y c x]$

$\to \partial E/\partial l = (\partial E/\partial O)(\partial O/\partial l)$ $\quad \frac{\partial O}{\partial b} \to [0\ y\ x\ 0]$

$\frac{\partial L}{\partial b} = \sum_{b,w,h}^{B,W,H} E_{b,w,i,h}$ → Bias: Channel h of E_{n-1} depends only on the H kernels $K_{S,N,M}$ where $h = S$

The channels of the $H_i K_{S,N,M}$ kernels can be combined to a $K_{H,N,M}$ one (resp. for lower layers) – resp. to weights

We have to correlate the channels H of $E_{h,n}$ with those of X → Pad X with half the kernel's width. Compute for $s \in (1,\dots,S)$: $X_s \star E_{h,n}$ to rec. $\nabla K_{h,S,N,M}$

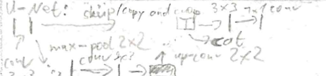
Stride = convolution followed by subsampling
backw.: upsampling followed by trans. conv.

⊙-○	Input ten.	Conv./Corr.	Output ten.
Forw. pass	(S,X,Y)	H×(S,N,M)	(H,X,Y)
∇ weights	(S,X,Y)	H×(X,Y)	(H,S,N,M)
∇ lower lay.	(H,X,Y)	S×(H,N,M)	(S,X,Y)

Conv. Batch Normalization $B \times H \times M \times N$
$\to B \times H \times M \cdot N \to B \times M \cdot N \times H \to B \cdot M \cdot N \times H$

RNN: Interpret Batch as Time Dimension
→ TBPTT: Implement by passing overlapping parts as a batch, memory between states, store last hidden state, impl. method switching whether this state is reused in subsequent forward passes] Backprop every k_1 time steps for only k_2 time steps

Binary cross entropy loss: $L(y_t, \hat{y}_t) = -y_t \ln(\hat{y}_t) - (1-y_t)\ln(1-\hat{y}_t)$



Target network deep Q learning: $y = \max_{a} Q(s,a)$ with function of w_t → oscillation → target network → C steps copy after update

Numpy: arr.copy(), np.argmax(arr, axis), np.expand(arr), np.sum(arr, axis), arr.shape, arr.T, & (bool matrix), np.log(arr), flatten: arr.reshape(batch_size, size), np.unravel_index (6, (3,4)) → 1, 2, np.zeros(shape), spsig.correlate / convolve, np.swapaxes(arr, 1, 2), np.pad(arr)

PyTorch: nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding), nn.BatchNorm2d(out_channels), nn.ReLU(), nn.DropOut(), nn.AvgPool2d(kernel_size), nn.Flatten(), nn.Linear(in_features, out_features), nn.Sigmoid(), nn.Module
ret = self..., ret += residual
x.cuda(), x.cpu(), y.reshape(x, ...), torch.flatten(y, 1), shape[0], −1)
nn.ConvTranspose2d(in_channels, out_channels, kernel_size)

⊙-○ – Forward: correlation, 'same' padding
– ∇ weights: padding $X+N-1$, $Y+M-1$, down 'valid' correlation
– ∇ lower layers: Swap dimension 1 and 2 (batch & channel), convolution, 'same'

+ Conv: Translation invariant, grid-like structure, Concept of „shared weights", „local connectivity"
+ Regularization: counter overfitting, Bias/variance tradeoff, generalizing on unseen data, enforcing prior