1) Neural Network

General:

- structure determines function, parallel distributed, generalization
- for each node(=neuron) j at iteration k:
  Input signals (from prev layer) $y_i, i \in [n]$
  $\downarrow$
  Linear weighted sum $v_j(k) = \sum_{i=0}^{n} w_{ji}(k) y_i(k) = \vec{w} \cdot \vec{x}$
  $\downarrow$
  Function (output) signal $y_j(k) = \Phi_j(v_j(k))$
- bias is just set as $w_0(k)$ with constant "input" $y_0 := 1$
- activation function Φ: differentiable, monotone-increasing, bounded (and continuous), e.g.:
  - hardlim/threshold/heaviside: $v \geq 0$
  - sigmoid: logistic $(1+e^{-av})^{-1} \in (0;1)$, signum, tanh $\in (-1;1)$
- layers: input > hidden(s) > output
- **T 3.1 Universal Approximation Theory**: any continuous function can be approximated with DNN

Training:

- **Backpropagation**:
  - **given**: training set $\ell = \{x(k), d(k)\}_{k=1}^{K}$, α, network with reasonble random configs
  - **find**: optimal weights to mimic training set
  - error signal $e_j(k) = d_j(k) - y_j(k)$
  - error energy $E_j(k) = \frac{1}{2} e_j^2(k)$
    whole network $E(k) = \sum_j E_j(k)$
    avg risk: $\bar{E}(k) = \frac{1}{K} \sum_{k=1}^{K} E(k)$
    1. for each weight from i to j (and next layer l) in each layer *backwards*:
       $$\delta_j(k) = -\frac{\partial E(k)}{\partial v_j(k)} = \begin{cases} e_j(k) & \cdot \ \Phi_j'(v_j(k)) \quad \text{if j output layer} \\ \sum_l w_{lj}(k)\delta_l(k) & \cdot \ \Phi_j'(v_j(k)) \quad \text{if j hidden layer} \end{cases}$$
       $\Delta w_{ji}(k) = \alpha \delta_j(k) y_i(k)$
    2. repeat step 1 until $\bar{E}(k)$ below specified threshold
- **with momentum:**
  - combines the advantages of low and high learning rate: stablizes the trajectory in the weight space and speeds up the learning speed towards steady directions:
    $$\Delta w_{ji}(k) = \beta \Delta w_{ji}(k-1) + \alpha \delta_j(k) y_i(k)$$
- *Batch* learning (offline): all K samples is 1 epoch, update weights once per epoch
- *Online* learning: update weights for *each* sample datapoint (stochastic)
- Cross-Validation:
  - training set (90%):
    - estimation subset (90%) for actual training
    - validation subset (10%) for picking best model (tune hyperparameters)
  - test set (10%) for evaluating the *final* model

Binary Classifier (Rosenblatt's Perceptron):

- single-layer (0 hidden layers) with only 1 binary output node
- $\Phi(v) = hardlim(v)$ (1 if v>0, otherwise 0)
- can only classify linearly separable patterns (i.e. there exists a hyperplane that seperates the classes)
- **Perceptron Training Algorithm**:
  - **given**: training set $D = \{(\vec{x}(k), d(k)) | k \in [s]\}$ (with k-th sample input $\vec{x}(k)$, desired output $d(k)$ and $x_0(k) = 1$ for bias) and learning rate $0 < \alpha \leq 1$
  - **find**: weights $\vec{w}(k)$ (with bias $w_0(k)$) at time k that best mimic training set
  - **intuition**: $\vec{w}$ is perpendicular to the boundary, so if output for k is wrong, nudge $\vec{w}$ towards $\aleph_1$ and away from $\aleph_0$ by adding or subtracting $x(k)$:
    1. randomize initial weight $\vec{w}(0)$
    2. for $k \in [1; s]$:
       $y(k) := \Phi(\vec{w}(t) \cdot \vec{x}(k))$
       $w_i(k+1) := w_i(k) + \alpha(d_i(k) - y_i(k)) x_i(k)$
  - this converges always (weights won't change) *if training set is linearly separable* (T2.1 Perceptron Convergence Theorem)!

Radial-Basis Function Networks

- **Radial-basis function:** real-valued function only dependent on the distance from origin c: $\Phi(\vec{x}, c) = \varphi(\|x - c\|)$ where $\|\vec{x}\|_2 := \sqrt{\sum_i x_i^2}$
  - i.e.:
    Multiquadrics $\Phi(x) = \sqrt{x^2 + c^2}$
    Inverse multiquadrics $\Phi(x) = \sqrt{x^2 + c^2}^{-1}$
    Guassian functions $\Phi(x) = exp(-\frac{x^2}{2\sigma^2}), \sigma > 0$
- essentially calculates $F(x) = \sum_{i=1}^{K} w_i \Phi(\|x - x^i\|)$ that *universally* and locally (as opposed to globally) approximate continuous function $f : \mathbb{R}^n \to \mathbb{R}$ given K data points $(\vec{x}^i, d_i)$ on f
- single *hidden* layer with n inputs and 1 output neuron, S many neurons in the hidden layer where $S \leq K$ number of clusters with the centroids on the clusters becoming the interpolation points $\vec{x}^i$
- hidden layer usually non-linear activation function, output layer identity function or linear
- step 1: k-means clustering algorithm to determine S
  step 2: backpropagation to train weights like normal
- **k-means clustering: (1)** choose random S centers $\vec{c}_j(0)$, **(2)** pick random sample $\vec{x}$, **(3)** the closest centroid $j(\vec{x})$ is moved towards it: $\vec{c}_j(k+1) \mathrel{+}= \alpha_c(\vec{x}(k) - \vec{c}_j(k))$, **(4)** repeat from *step (2)* until centers don't move much
- **standard width of gaussian RBFs:** $\sigma = \frac{d_{max}}{\sqrt{2S}}$ where d_{max} max distance between any two clusters

Recurrent Neural Networks:

- dynamic neural network by giving it short/long-term memory through feedback
- local feedback: single neuron inside the network
  global feedback: multiple hidden layers or whole network
- the internal state y_t is put back into the network as input in the next timestep, thus output $h_t = f(y_{t-1}, x_t)$

- **Lyapunov stability:** equilibrium state $\bar{x}$ is *(uniformly) stable*, if $\forall\varepsilon > 0 : \exists\delta > 0 : (\|x(0) - \bar{x}\| < \delta \implies \forall t > 0 : \|x(t) - \bar{x}\| < \varepsilon)$, i.e. for any bound ε, there is the *region of attraction* δ, within which any initial state will stay bounded by ε forever
- **asymptotic stability:** if both stable *and* convergent ($x(t) \to \bar{x}$ as $t \to \infty$) within the ROA δ
- if a **Lyapunov function** $V(x) : \mathbb{R}^n \to \mathbb{R}$ is *positive-definite* in a small neighbourhood of $\bar{x}$, and $\frac{\partial V(x)}{\partial t}$ is *negative semidefinite* in that region, then $\bar{x}$ is stable (even *asymptotically* stable if derivative is *negative definite*)
- **positive-definite**: has continuous partial derivatives of each state variable and $V(\bar{x}) = 0$ and $V(x) > 0$ if $x \neq \bar{x}$ (*semi-*: $\geq$, *negative-*: $<$)
- **Hopfield Network**: $\vec{x}(k+1) = \Phi(W\vec{x}(k) + \vec{b})$, usually $\Phi = satlins$ linear capped in $[-1;1]$, converges always!

2) Fuzzy Logic

Fuzzy Sets and Membership Functions:

- given universe set X, its subset can be defined by its **characteristic function** $A : X \to \{0,1\}$ which, given $x \in X$ returns whether its in A or not
- **fuzzy set** has characteristic function $A : X \to [0,1]$, returns the degree of membership (= **membership function**)
- fuzzy set A is *normal* if $sup_{x\in X} A(x) = 1$, otherwise *subnormal*
- standard operators:
  - $A^c(x) = 1 - A(x)$, does *not* satisfy Law of Contardiction/Excluded Middle
  - $(A \cup B)(x) = max\{A(x), B(x)\} = A(x) \vee B(x)$, commutative, distributive
  - $(A \cap B)(x) = min\{A(x), B(x)\} = A(x) \wedge B(x)$
- alt def (*Yager's connectives*) with $w \in (0,\infty)$:
  - $A^c(x) = (1 - A(x)^w)^{1/w}$
  - $(A \cup_w B)(x) = min\{1, (A(x)^w + B(x)^w)^{1/w}\}$
  - $(A \cap_w B)(x) = 1 - min\{1, ((1 - A(x))^w + (1 - B(x))^w)^{1/w}\}$
- α-cut: fuzzy set to crisp set ${}^{\alpha}A = \{x | A(x) \geq \alpha\}$ or ${}^{\alpha+}A = \{x | A(x) > \alpha\}$
- **decomposition**: define ${}_{\alpha}A(x) = \alpha \cdot {}^{\alpha}A(x)$ which creates a horizontal & vertical slice at α, then $A = \bigcup_{\alpha\in[0,1]} {}_{\alpha}A = sup_{\alpha}\{{}_{\alpha}A(x)\}$

Fuzzy Relations and Propositions:

- a (crisp) relation $R : X \to Y$ can be defined by its **characteristic function** $R : X \times Y \to \{0,1\}$, so a **fuzzy relation** is $R : X \times Y \to [0,1]$
- **proposition**: given linguistic variables U, V (nouns e.g. AGE), values (=fuzzy set) A, B (adj. e.g. Young), and optionally hedges H (adv. e.g. very, not): "U is H A"
- **conjunction** (with U_i ling. var. over domains X_i): $A_1 \times A_2(x_1, x_2) = A_1(x_1) \wedge A_2(x_2)$ (matrix representation possible)
- **condition proposition/implication**: "IF U is A, then V is B"
  Lukasiewicz(Zadeh): $R_Z(x,y) = min(1, 1 - A(x) + B(y))$
  Correlation min: $R_{cm}(x,y) = min(A(x), B(y))$
  Correlation product: $R_{cp}(x,y) = A(x) * B(y)$
- matrix representation: express A and B as row vectors over their domains, then $R = A^T \cdot B$ with operator being $R(x,y)$
- **Inference**: *modus ponens*: $(P \Rightarrow Q) \wedge P = Q$, *modus tollens*: $(P \Rightarrow Q) = (\neg Q \Rightarrow \neg P)$
  - single inference:
    given: Rule "IF U is A, THEN V is B" and Fact "U is A'":
    conclusion: $B'(y) = A'(x) \circ R(x,y) = sup_{x\in X} min\{A'(x), R(x,y)\}$ (matr. mul. with row vector and matrix, $(*) := min$ and $(+) := max$)
  - n antecedents and k rules:
    given: Rules $R_i(x_1, ..., x_n, y)$ and Fact "U_1 is A_1' and ... and U_n is A_n'"
    k **conclusions:** $B_i'(y) = A_{i1}' \times ... \times A_{in}'(x_1, ..., x_n) \circ R_i(x_1, ..., x_n, y)$
    aggregated conclusion: $B'(y) = \sum_i B_i'(y)$ or $max_i\{B_i'(y)\}$
- **Defuzzification:** centroid $\bar{y} = (\sum_y y \cdot B'(y))/(\sum_y B'(y))$

T-S Fuzzy Model:

- universal approximators that model a nonlinear plant $\to$ TSFM $\to$ local-linear fuzzy rules $\to$ defuzzify and control plant
- Continuous FS: "IF $z_1(t)$ is M_{i1} and ... and $z_p(t)$ is M_{ip}, THEN $\dot{x}(t) = A_i x(t) + B_i u(t), y(t) = C_i x(t)$" (for DFS: $\dot{x}(t) \to x(t+1)$)
- disc. lin. system $x(t+1) = Ax(t)$ is locally stable if $\forall\lambda \in eig(A) : |\lambda| < 1$ within unit circle
- cont. lin. system $\dot{x}(t) = Ax(t)$ is locally stable if $\forall\lambda \in eig(A) : Re(\lambda) < 0$
- (eigenvalues: solve $det(A - \lambda I) = (a - \lambda)(d - \lambda) - bc = 0$)
- DFS: globally asymp. stable if a common positive definite matrix P exists such that $A_i^T P A_i - P < 0, i = 1, 2, ..., r$

Evolutionary Algorithms:

- iterative search prodecure that generates a *population* of candidate solutions (rather than 1 point like gradient), select best solutions and generates new random solutions based off of them
- selection:
  - $(\mu + \lambda)$: μ parents create λ offspring and best μ from $\mu + \lambda$ become next parents
  - (μ, λ): best μ from only λ become next parents
  - proportional/roulette: probability of getting picked is their relative fitness
  - tournament: select q random, pick best, repeat until full
- variation:
  - mutation: binary/gaussian mutation
  - 1-point crossover: two parents $x, y \in \mathbb{R}^n$ at point p produce two new offsprings
    $x' = (x_1, x_2, ..., x_{p-1}, y_p, ..., y_n)$
    $y' = (y_1, y_2, ..., y_{p-1}, x_p, ..., x_n)$
  - n-point crossover: at each crossover point swap the rest
  - PMX: pick mappings $x_i \leftrightarrow y_i$ for $i \in \{\text{random indices}\}$, swap values according to mappings, rest stays same
  - blending: average each components