Radial Basis Function Neural Networks

& Recurrent Neural Networks

Components • Training • Growing Networks • Jordan & Elman Networks

Course Page:

https://www.gcjana.in/courses/shardauniversity/2502/CSA203/

Based on: Haykin (2009) Neural Networks and Learning Machines | Broomhead & Lowe (1988) | Powell (1987)

CONTENTS

Lecture Overview

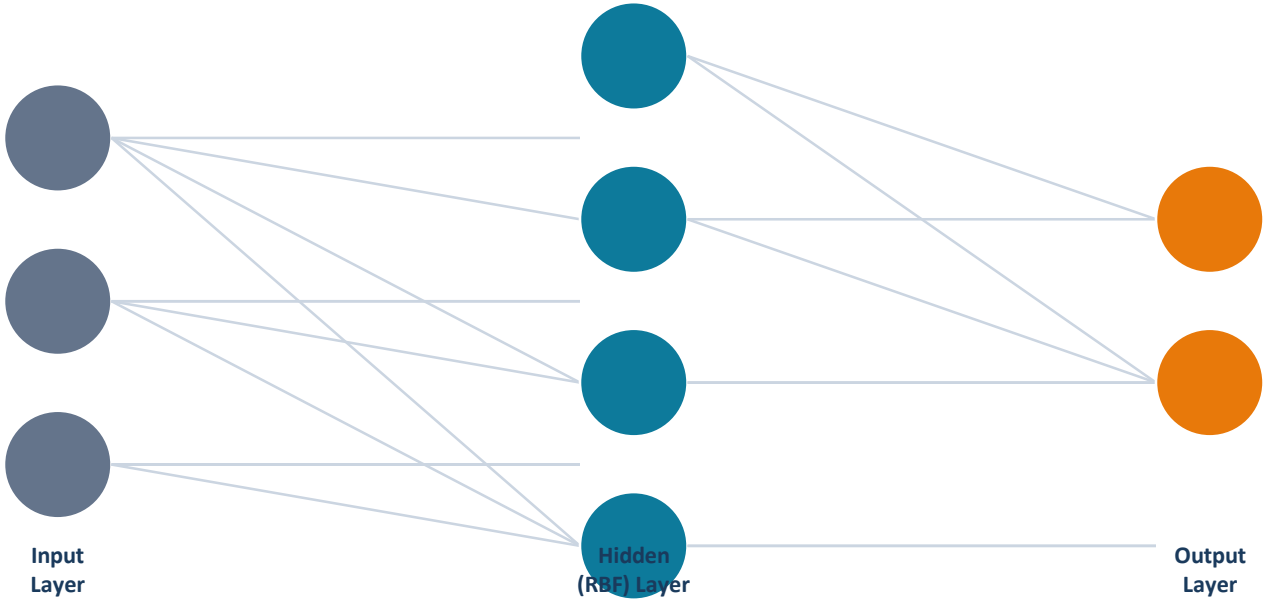
Part I: RBF Neural Networks

Part II: Recurrent Neural Networks

Part I
RBF Neural Networks

Radial Basis Function Networks

Components & Structure of an RBF Network



Three-Layer Architecture

- Input Layer: passes stimuli (no processing)
- Hidden Layer: RBF units, each computes a radial basis function $\phi(||x - c_i||)$
- Output Layer: linear superposition of hidden activations
- Network output: $f(x) = \Sigma\, w_i\, \phi(||x - c_i||) + \text{bias}$

Source: Haykin, S. (2009). Neural Networks and Learning Machines, 3rd Ed. Pearson.

Information Processing of an RBF Network

Two-Stage Processing Pipeline

Stage 1 — Hidden Layer (Nonlinear)

Each hidden unit computes the Euclidean distance between input x and its centre c_i:

$$\phi_i(x) = \phi(||x - c_i||)$$

Activation is a nonlinear function of distance. The mapping from input space to feature space is NONLINEAR and high-dimensional.

Stage 2 — Output Layer (Linear)

The output is a linear combination of the hidden unit activations:

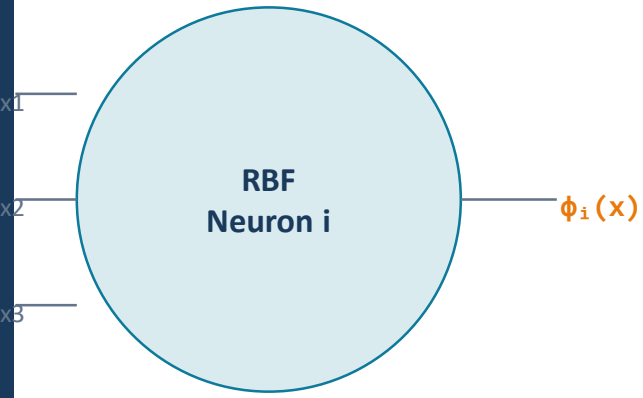
$$F(x) = \Sigma_i w_i \phi_i(x) + w_o$$

The linearity of the output layer means the network is a linear regression in the feature space defined by the RBF units.

Key Insight: The separation of feature extraction (nonlinear hidden layer) from decision-making (linear output layer) enables efficient training — only output weights need gradient-based learning; centres and widths can be set analytically (Broomhead & Lowe, 1988).

Source: Broomhead, D.S. & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. Complex Systems, 2, 321–355.

Information Processing in RBF Neurons



Common Radial Basis Functions

Gaussian (most common) $\phi(r) = \exp(-r^2 / 2\sigma^2)$

Localised response; σ controls width. Analogous to a radial receptive field.

Multiquadric (Powell, 1987) $\phi(r) = \sqrt{(r^2 + c^2)}$

Non-local; grows with distance. Useful for interpolation.

Inverse Multiquadric $\phi(r) = 1 / \sqrt{(r^2 + c^2)}$

Localised; always positive and bounded above by 1/c.

Thin-Plate Spline $\phi(r) = r^2 \ln(r)$

Widely used in scattered data interpolation problems.

$r = ||x - c_i||$ (Euclidean distance from input x to centre c_i); σ, c = shape parameters

Analytical Thoughts Prior to Training

Before training, we must determine: How many RBF units? Where to place centres? What widths?

1. Interpolation Perspective

- Exact interpolation: N centres = N training points → always fits data (Powell, 1987)
- Requires solving NxN linear system: $\Phi w = d$
- Problem: ill-conditioning if data clusters tightly
- Solution: regularisation (Tikhonov) or subset selection

2. Centres Placement Strategies

- Random subset: pick $M < N$ training points as centres
- K-means clustering: centres = cluster centroids (most common)
- Orthogonal Least Squares (OLS): greedily select centres to reduce residual error
- Self-organising maps (SOMs) can also serve as centre finders

3. Width (σ) Heuristics

- P-nearest-neighbour: $\sigma_i = (1/P) \Sigma \, ||c_i - c_j||$
- Uniform width: $\sigma = d\_max / \sqrt{(2M)}$ where d_max = max inter-centre distance
- Width controls overlap; too small → gaps; too large → over-smoothing
- Cross-validation is the gold standard for σ selection

4. Network Size Selection

- Bias-variance trade-off governs choice of M
- More centres → lower bias, higher variance
- Regularisation bridges interpolation and approximation
- Structural Risk Minimisation (Vapnik) offers theoretical guidance

Source: Powell, M.J.D. (1987). Radial basis functions for multivariable interpolation: a review. Algorithms for Approximation, pp. 143–167.

Equation System & Gradient Strategies for Training

Linear System (Output Weights)

Design Matrix: $\Phi \in \mathbb{R}^{N \times M}, \quad \Phi_{ij} = \phi(||x_i - c_j||)$

Least-squares problem: $\min_w ||\Phi w - d||^2$

Normal equation: $w^* = (\Phi^T \Phi)^{-1} \Phi^T d = \Phi^+ d$

Regularised (Tikhonov): $w^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T d$

Ridge param λ: controls smoothness vs. fit trade-off

Gradient Strategies for Full Training

Hybrid Learning (two-phase)

Phase 1: fix output weights, use unsupervised clustering for centres & widths. Phase 2: fix centres, optimise weights with least-squares. Fast convergence.

Gradient Descent (full)

Jointly optimise w, c_i, σ_i via backprop. $\partial E/\partial c_i$ and $\partial E/\partial \sigma_i$ derived from chain rule through ϕ. Slower but more accurate placement.

Gradient wrt Centre (Gaussian)

$\partial \phi/\partial c_i = (2(x - c_i)/\sigma^2) \cdot \phi(||x - c_i||)$. Update: $c_i \leftarrow c_i - \eta \cdot \partial E/\partial c_i$

Regularised Training

Add weight-decay term $\lambda||w||^2$ to error. Prevents overfitting, controls effective complexity.

Source: Moody, J. & Darken, C.J. (1989). Fast learning in networks of locally-tuned processing units. Neural Computation, 1(2), 281–294.

Growing RBF Networks

Dynamic construction: add hidden units incrementally based on the residual error criterion.

Orthogonal Least Squares (OLS) Growing Algorithm

1. **Initialise:** Start with empty network (M=0 centres)
2. **Candidate Set:** All N training points are candidate centres
3. **Select Centre:** For each candidate, compute Error Reduction Ratio (ERR): $ERR_i = w_i^2(g_i^T g_i) / d^T d$, where g_i is Gram-Schmidt orthogonalised basis vector
4. **Add Unit:** Add centre with maximum ERR to network
5. **Update Residual:** Recompute residuals after adding new unit
6. **Stop Criterion:** Stop when cumulative ERR ≥ threshold (e.g., 0.999) or max M reached

$$ERR_i = [w_i\ (g_i^T g_i)^{(1/2)}]^2 / (d^T d) \quad \text{(Error Reduction Ratio)}$$

Advantages of Growing Approach

- Automatically determines M (no pre-specification needed)
- Computational efficiency: forward selection is $O(M^2N)$
- Orthogonalisation prevents near-linear dependencies among basis vectors
- Produces parsimonious networks with good generalisation
- OLS selects most informative centres first
- Well-suited for large datasets with redundant training points

Chen, Cowan & Grant (1991). Orthogonal least squares learning algorithm for radial basis function networks.

Source: Chen, S., Cowan, C.F.N. & Grant, P.M. (1991). IEEE Trans. Neural Networks, 2(2), 302–309.

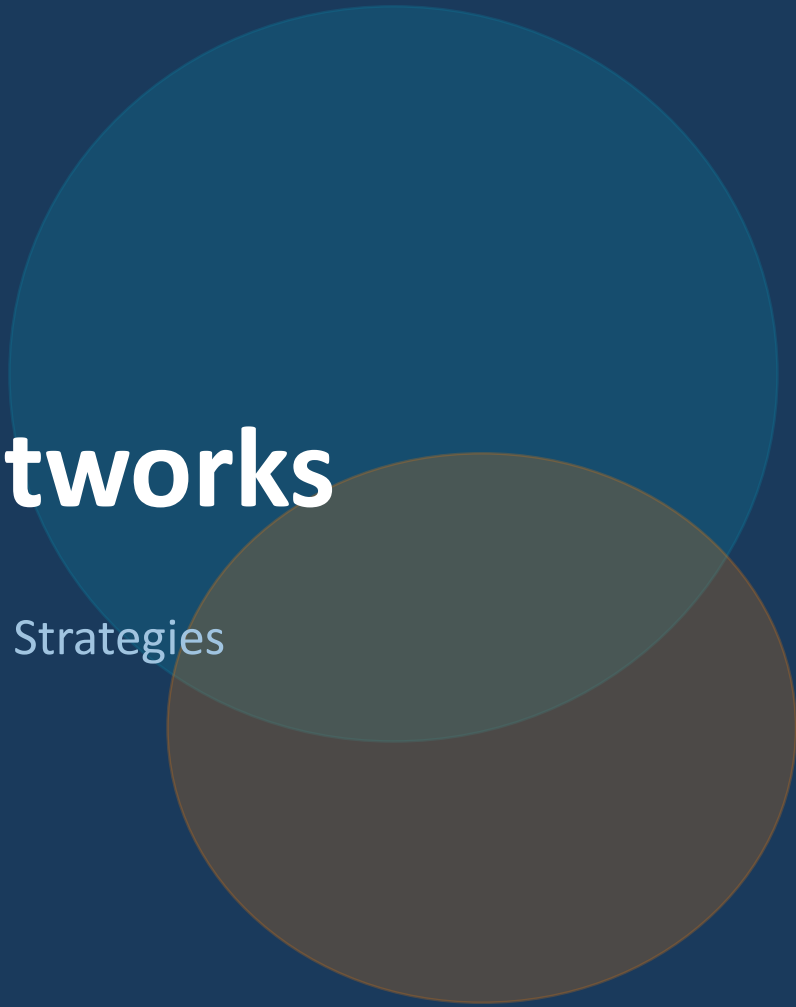
Comparison: RBF Networks vs Multilayer Perceptrons

Criterion	RBF Network	Multilayer Perceptron
Activation Function	Radial (local): $\phi(\|\|x-c\|\|)$	Sigmoidal (global): $\sigma(w^T x+b)$
Locality	Local: each neuron responds to nearby inputs only	Global: each neuron influenced by entire input space
Number of Layers	Always 3 (fixed architecture)	Variable depth; deep nets possible
Output Layer	Linear — analytically solvable	Nonlinear — requires iterative training
Training Speed	Faster: two-phase or one-shot LS	Slower: full backpropagation through all layers
Universal Approximation	Yes (Park & Sandberg, 1991)	Yes (Cybenko, 1989; Hornik, 1991)
Generalisation	Good with regularisation; prone to curse of dimensionality	Typically better in high dimensions
Interpretability	Higher: centres are prototypes in input space	Lower: distributed representations
Hyperparameters	M (centres), σ (widths), λ (regularisation)	Depth, width, learning rate, momentum, etc.
Best For	Low-to-medium dim., sufficient data density	High-dim. data, large-scale tasks, deep learning

Sources: Park, J. & Sandberg, I.W. (1991). Universal approximation using RBF networks. Neural Computation 3(2). | Cybenko, G. (1989). Math. Control Signals Syst., 2, 303–314.

Part II
Recurrent Neural Networks

Jordan Networks • Elman Networks • Training Strategies



Recurrent Neural Networks: Introduction

Feedforward networks map fixed-size inputs to outputs; they have no memory. RNNs process sequences by maintaining an internal state that captures temporal context.

What Makes an RNN 'Recurrent'?

Connections that form directed cycles, allowing information to persist. At each time step t, the network produces both an output and an updated hidden state. The hidden state is a learned compressed representation of all past inputs.

Key Property: Dynamic Memory

Unlike feedforward networks, RNNs have an implicit short-term memory. This allows them to model time series, natural language, speech, and any sequential data where context matters.

Core Equation

Hidden state: $h(t) = f(W_{hh} \cdot h(t-1) + W_{xh} \cdot x(t) + b_h)$
Output: $y(t) = g(W_{hy} \cdot h(t) + b_y)$

W_{hh}: recurrent weights | W_{xh}: input weights

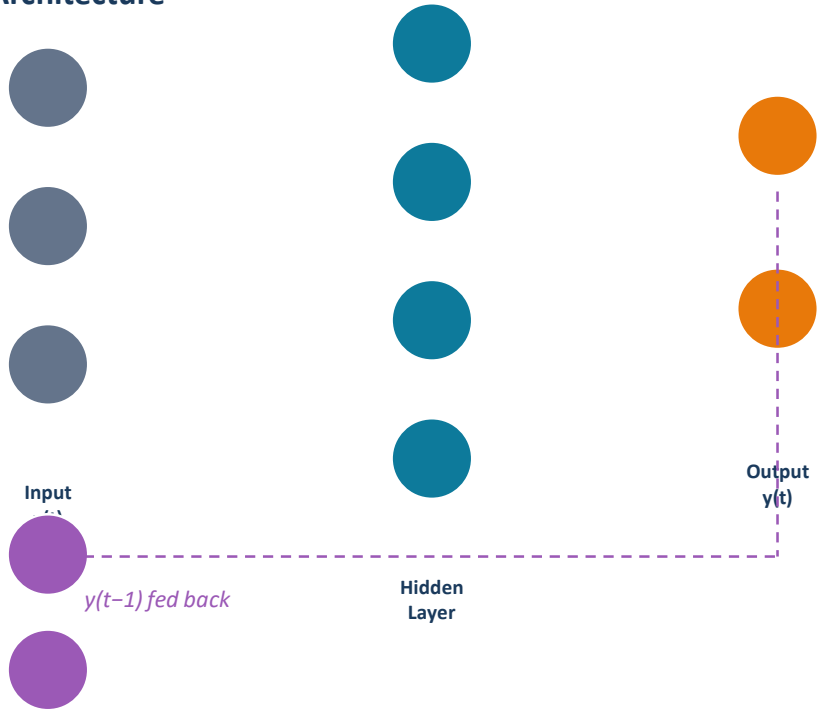
Two Classic Architectures

Jordan Network (1986): output context units — the network's own output is fed back as input at the next time step.
Elman Network (1990): hidden context units — the hidden layer state is fed back.

Source: Haykin, S. (2009). Neural Networks and Learning Machines, 3rd Ed., Ch. 15.

Jordan Networks

Architecture



Jordan Network — Key Features

Proposed by:

Michael I. Jordan (1986) — 'Serial Order: A Parallel Distributed Processing Approach'

Context Units:

Receive copy of previous output y(t−1); act as external memory visible to the network

State equation:

$s(t) = \alpha \cdot s(t-1) + y(t-1)$ (leaky integrator)

$\alpha \in [0,1]$ controls memory decay

Output feedback:

Hidden layer receives both x(t) and context s(t); context captures past outputs, not past hidden states

Limitation:

Context only reflects output history; hidden layer dynamics are not directly recurrent

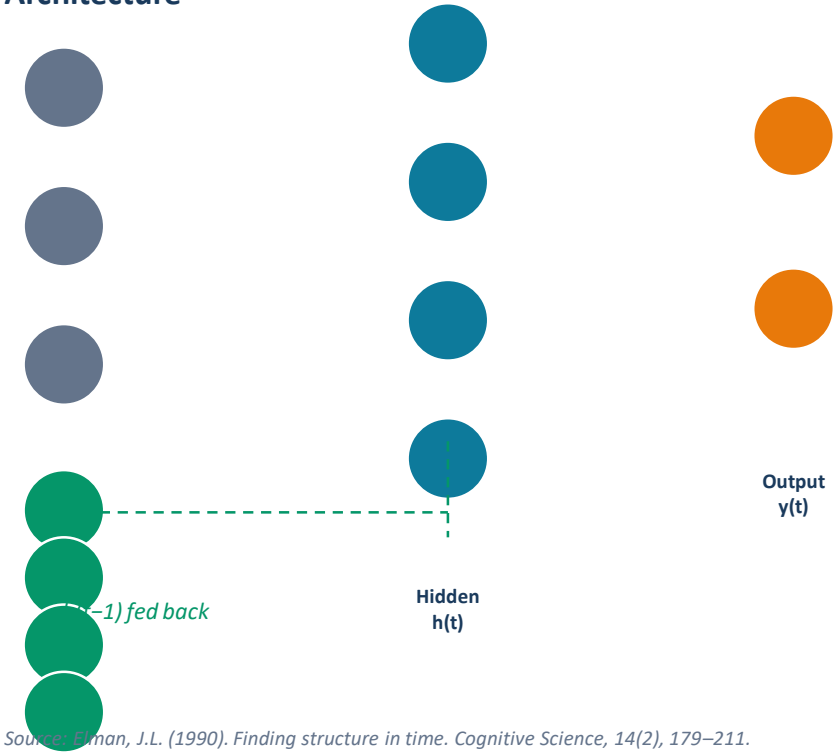
Application:

Sequence generation, motor control, serial order tasks

Source: Jordan, M.I. (1986). Serial order: A parallel distributed processing approach. UCSD Institute for Cognitive Science TR 8604.

Elman Networks

Architecture



Elman Network — Key Features

Proposed by:
Jeffrey L. Elman (1990) — 'Finding Structure in Time'

Context Units:
Store a copy of previous hidden layer h(t−1); this creates a recurrent connection through the hidden layer itself

State equation:
$h(t) = \sigma(W_{xh} \cdot x(t) + W_{hh} \cdot h(t-1) + b_h)$
Direct hidden-to-hidden recurrence

vs Jordan:
Elman feeds back hidden state (richer dynamics); Jordan feeds back only the output (limited memory)

Expressivity:
Elman networks are more powerful: hidden state can encode arbitrary intermediate representations, not just output values

Application:
Language modelling, grammar learning, time-series prediction, speech recognition

Source: Elman, J.L. (1990). Finding structure in time. Cognitive Science, 14(2), 179–211.

Jordan vs Elman: Side-by-Side Comparison

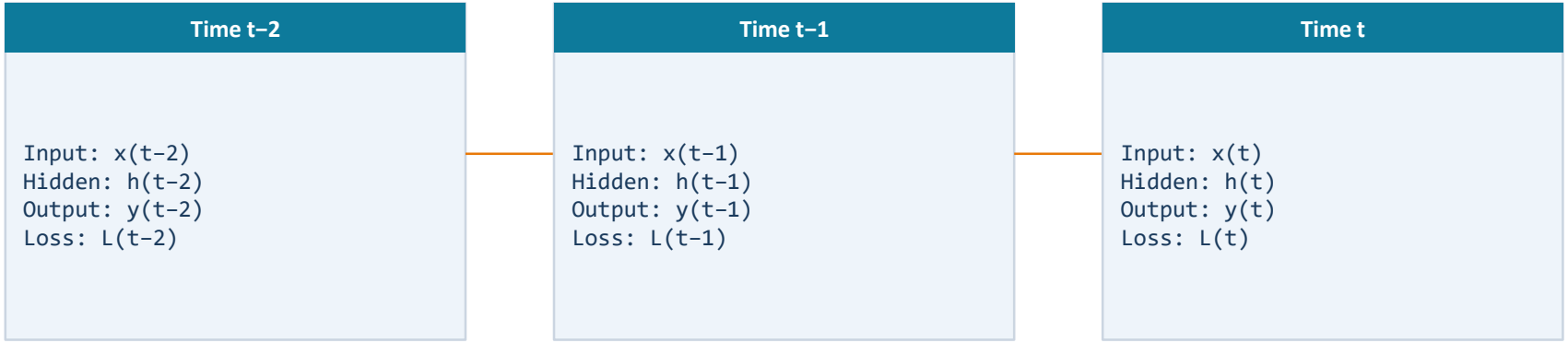
Feature	Jordan Network (1986)	Elman Network (1990)
Context Connection	Output y(t−1) → Context → Hidden	Hidden h(t−1) → Context → Hidden
Feedback Source	Network output (external)	Hidden layer state (internal)
Context Decay	s(t) = α·s(t−1) + y(t−1); α tunable	Fixed copy: c(t) = h(t−1)
Information in Context	Only what the network outputs	Full hidden-layer representational state
Expressiveness	Limited — output may lose details	Higher — hidden state is richer
Training Difficulty	Easier: error at output is direct	Harder: gradient flows back through h(t−1)
Memory Capacity	Determined by output size	Determined by hidden layer size
Typical Use Case	Motor control, sequence generation	Language, grammar, general sequences
BPTT Needed?	Less critical (output context)	Essential (recurrent hidden state)

Source: Haykin, S. (2009). Neural Networks and Learning Machines, 3rd Ed., Prentice Hall.

Training Recurrent Neural Networks: BPTT

Standard backpropagation cannot handle cyclic connections directly. Backpropagation Through Time (BPTT) unfolds the network in time and applies standard gradient rules.

BPTT: Network Unfolded Across Time Steps (t−2, t−1, t)



Core BPTT Gradient Equations

```
δh(t) = (∂L/∂h(t)) ⊙ f'(h(t))  +  W_hh^T · δh(t+1)   [delta at hidden layer t]
∂L/∂W_hh = Σ⃞ δh(t) · h(t−1)^T        [gradient on recurrent weights]
∂L/∂W_xh = Σ⃞ δh(t) · x(t)^T          [gradient on input-to-hidden weights]
```

Source: Werbos, P.J. (1990). Backpropagation through time: What it does and how to do it. Proc. IEEE, 78(10), 1550–1560.

Training RNNs: Vanishing Gradient & Solutions

The vanishing gradient problem is the central challenge in training RNNs: gradients decay exponentially with the number of time steps, preventing learning of long-range dependencies.

THE PROBLEM

Gradient decays exponentially:

$$\left\|\frac{\partial h(t)}{\partial h(k)}\right\| \leq (\lambda_{max})^{(t-k)}$$

If $\lambda_{max} < 1 \rightarrow$ gradients $\rightarrow 0$ (vanish)
If $\lambda_{max} > 1 \rightarrow$ gradients $\rightarrow \infty$ (explode)

Vanishing: network forgets distant inputs
Exploding: training becomes unstable

SOLUTIONS

Gradient Clipping

Limit $||\nabla||$ to threshold θ: if $||g|| > \theta$, $g \leftarrow \theta \cdot g/||g||$. Prevents explosion (Pascanu et al., 2013).

Truncated BPTT

Only unroll T steps back. Reduces computation; may miss very long dependencies.

LSTM (Hochreiter & Schmidhuber, 1997)

Gated cell state carries gradients unchanged. Forget, input, output gates regulate information flow.

GRU (Cho et al., 2014)

Simpler gating: update and reset gates. Nearly as effective as LSTM with fewer parameters.

Echo State Networks

Reservoir computing: fix recurrent weights, train only output. Avoids BPTT entirely.

Sources: Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8). | Pascanu, R. et al. (2013). On the difficulty of training recurrent neural networks. ICML.

Training RNNs: RTRL & Additional Methods

BPTT (Werbos, 1990)

Unroll in time; apply standard backprop. Time complexity: $O(T \cdot N^2)$ per sequence. Requires storing all activations; memory $O(T \cdot N)$. Best for offline/batch training where full sequence is available.

RTRL — Real-Time Recurrent Learning (Williams & Zipser, 1989)

Online algorithm: propagates sensitivity matrices forward in time. Computes $\partial h(t)/\partial W$ incrementally. Time complexity: $O(N^4)$ per step — expensive. Advantage: online, no unrolling needed; suitable for continuous, non-episodic tasks.

Teacher Forcing

During training, feed ground-truth $y(t-1)$ as input instead of network's own $y(t-1)$. Accelerates convergence. Risk: exposure bias — at test time, network must condition on its own (possibly wrong) predictions. Scheduled sampling (Bengio et al., 2015) gradually reduces teacher forcing.

Second-Order Methods

Hessian-free optimisation (Martens & Sutskever, 2011): uses curvature information via conjugate gradient on the Hessian-vector product. Avoids explicit Hessian storage. Particularly effective for deep RNNs with long-range dependencies.

Regularisation for RNNs

Dropout on non-recurrent connections (Zaremba et al., 2014). Zoneout: randomly preserve hidden/cell states. L2 weight decay on recurrent weights. Activity regularisation: penalise $||h(t)||$.

Initialisation Strategies

Identity matrix initialisation for W_hh (Le et al., 2015) with ReLU. Orthogonal initialisation preserves gradient norms. Small random initialisation for feedforward weights; $|\lambda| < 1$ for stability.

Sources: Williams, R.J. & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. Neural Computation, 1(2). | Werbos, P.J. (1990). Proc. IEEE.

SUMMARY

Summary & Key Takeaways

RBF Networks

- Three-layer network: input → RBF hidden → linear output
- Hidden units compute radial distances from learned centres
- Training: set centres (k-means/OLS), then solve linear system for weights
- Growing networks (OLS) automatically determine architecture
- Faster training than MLPs; limited by curse of dimensionality

RNN: Jordan & Elman

- Jordan: output context — feeds y(t−1) back to hidden layer
- Elman: hidden context — feeds h(t−1) back; richer dynamics
- Both can model temporal/sequential patterns unlike feedforward nets
- Elman networks are generally more expressive than Jordan networks

Training RNNs

- BPTT: unfold in time, backpropagate through unrolled network
- RTRL: online forward-mode alternative; $O(N^4)$ but no unrolling
- Vanishing gradients: use LSTM/GRU or gradient clipping
- Teacher forcing accelerates training; scheduled sampling reduces exposure bias

References

Broomhead, D.S. & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. Complex Systems, 2, 321–355.

Chen, S., Cowan, C.F.N. & Grant, P.M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. IEEE Trans. Neural Networks, 2(2), 302–309.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2, 303–314.

Elman, J.L. (1990). Finding structure in time. Cognitive Science, 14(2), 179–211.

Haykin, S. (2009). Neural Networks and Learning Machines, 3rd Ed. Prentice Hall/Pearson Education.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780.

Jordan, M.I. (1986). Serial order: A parallel distributed processing approach. UCSD Institute for Cognitive Science TR 8604.

Moody, J. & Darken, C.J. (1989). Fast learning in networks of locally-tuned processing units. Neural Computation, 1(2), 281–294.

Park, J. & Sandberg, I.W. (1991). Universal approximation using radial-basis-function networks. Neural Computation, 3(2), 246–257.

Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. ICML 2013.

Powell, M.J.D. (1987). Radial basis functions for multivariable interpolation: a review. In Algorithms for Approximation (pp. 143–167).

Werbos, P.J. (1990). Backpropagation through time: What it does and how to do it. Proceedings of the IEEE, 78(10), 1550–1560.

Williams, R.J. & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. Neural Computation, 1(2), 270–280.