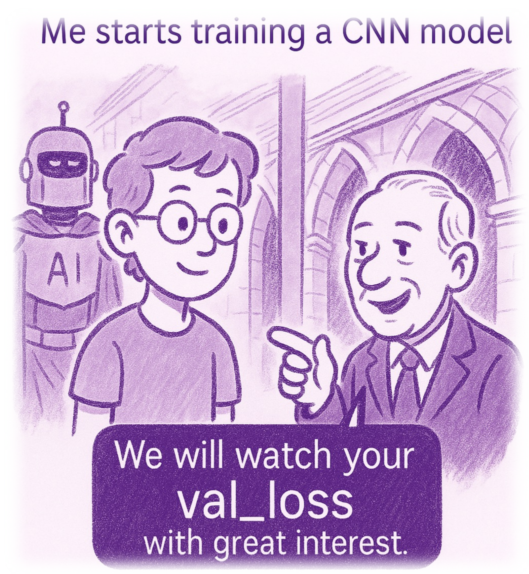
Efficient BackProp: Classic Tricks and Modern Insights

Introduction to Backpropagation Training

Backpropagation is the core algorithm for training neural networks. It uses the chain rule to compute gradients and updates weights with gradient descent. Despite its simplicity and efficiency, fast and stable convergence remains challenging in practice.



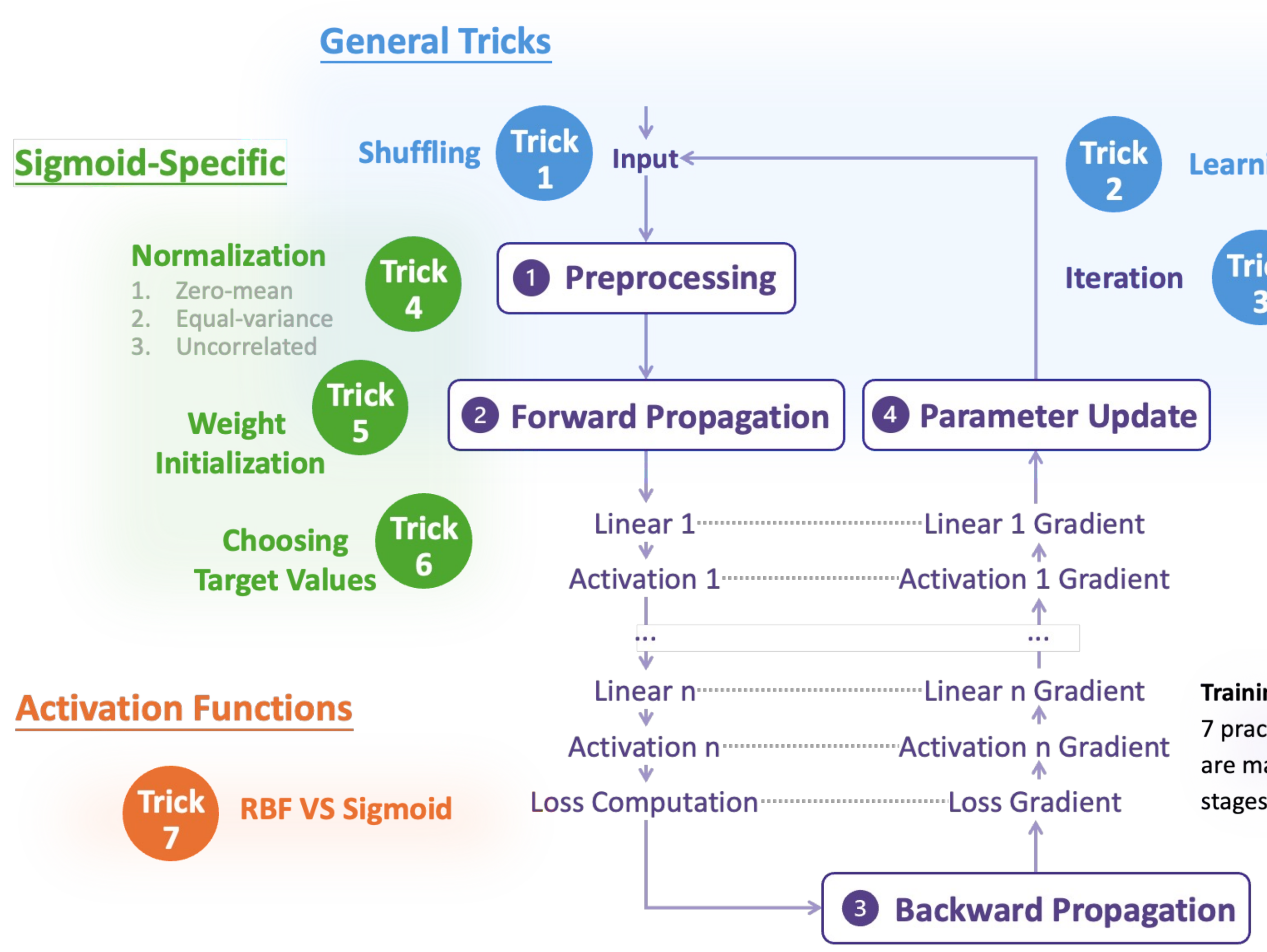
Common challenges in backpropagation:

- Overfitting or order bias — Trick 1
- Slow or unstable convergence — Trick 2
- Noisy or oscillating updates — Trick 2-3
- Vanishing gradients (saturation) — Trick 4-6
- Uneven learning speed across layers — Trick 2,5
- Limited expressiveness of activations — Trick 7

Why Training Is Hard:

- Non-convex, high-dimensional **error surface** with local minima & flats
- Small changes in **initialization** or **learning rate** → very different results
- **Gradients** are propagated layer-by-layer → may vanish or explode
- **Training** is sensitive to input scale, architecture, and optimizer settings

In their influential 1998 paper ***Efficient BackProp***, LeCun et al. analyzed common backpropagation challenges and proposed practical tricks to improve training efficiency and stability.



Training Flow & Optimization Tricks
7 practical tricks from *Efficient BackProp* are mapped to their corresponding stages in the training process.

Objective of This Work

- **Highlight** challenges in training via backpropagation
- **Map** *Efficient BackProp* tricks to training stages
- **Contrast** classic and modern approaches with recent studies
- **Show** experimental results validating or replacing old tricks

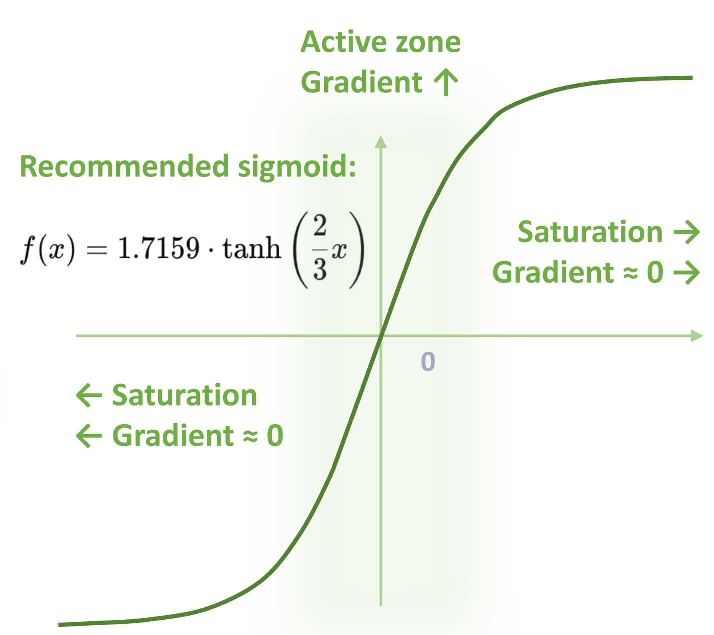
Classic Tricks from *Efficient BackProp*

General Tricks

- **Trick 1: Shuffling**
  1. Shuffle to avoid similar-class samples consecutively
  2. Emphasize high-error samples more often
  Risk: May overemphasize outliers
- **Trick 2: Learning Rate Selection**
  Set wisely:: Per-weight & layer-wise rates, scaled for shared weights
  Update smartly: Momentum, Adaptive Learning Rates
- **Trick 3: Stochastic VS Batch Learning:** Speed ⟵ balance ⟶ Accuracy
  Choose stochastic, mini-batch, batch based on task and data scale
  main stream now

Sigmoid-Specific

Sigmoid functions, especially **tanh**, are recommended as activation functions for their nonlinearity, smoothness, and zero-mean output that promotes faster convergence.



Properties of tanh:

- **Sensitive** to input and initialization
- **Saturates** at large/small inputs
- **Strongest gradient** around input = 0

Keep inputs in tanh's active zone:

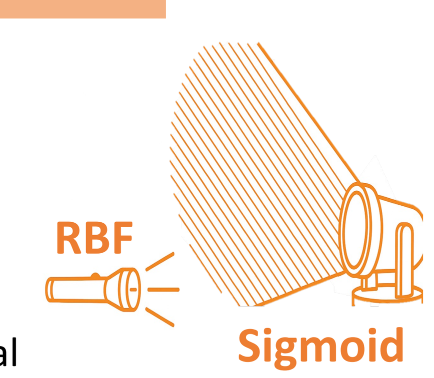
- **Trick 4: Normalization** (global)
  Zero-mean, equal-variance, uncorrelated
- **Trick 5: Weight Initialization** (Xavier Init)
  Random weights with $\sigma_w = m^{-1/2}$
- **Trick 6: Target Values**
  Use ±1 targets instead of 0/1

Now largely replaced by softmax + cross-entropy

Activation Functions

Trick 7: Sigmoid VS Radial Basis Function (RBF)

Feature	Sigmoid	RBF
Activation	Dot product → sigmoid	Distance → Gaussian
Coverage	Global (whole space)	Local (near center only)
Flexibility	Smooth, general	Focused, adaptive
Training	Gradient descent	Gradient + clustering
Best for	Deep, high-dimensional	Shallow, low-dimensional



- **Sigmoid:** used in lower layers to capture broad structure.
- **RBF:** used in upper layers or shallow networks for localized decisions.

Hard to scale to deep or high-dimensional networks; rarely used today

Modern Insights from Research

Learning Rate Selection

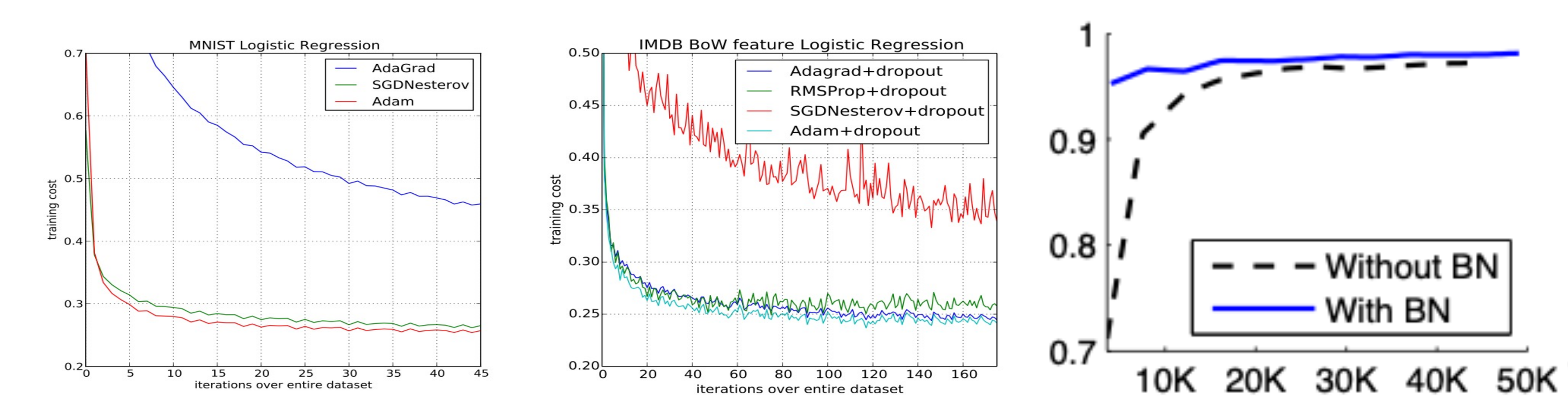
Per-weight learning rates improve convergence speed → adopted by adaptive methods such as **Adam [Kingma & Ba, 2015]**.
Momentum (esp. Nesterov) speeds up training on curved surfaces→ widely used in **SGD variants [Sutskever et al., 2013].**

Normalization

Classic: Normalize only inputs, rely on static preprocessing, and use costly methods like PCA → impractical during training.

BatchNorm: Normalizes layer activations using batch statistics → speeds up training and reduces internal covariate shift **[Ioffe & Szegedy, 2015]**.

LayerNorm: Normalizes per sample, independent of batch size → effective for RNNs and Transformers **[Ba et al., 2016]**.



Learning Rate Selection (Momentum / Adaptive) — Normalization (BatchNorm)

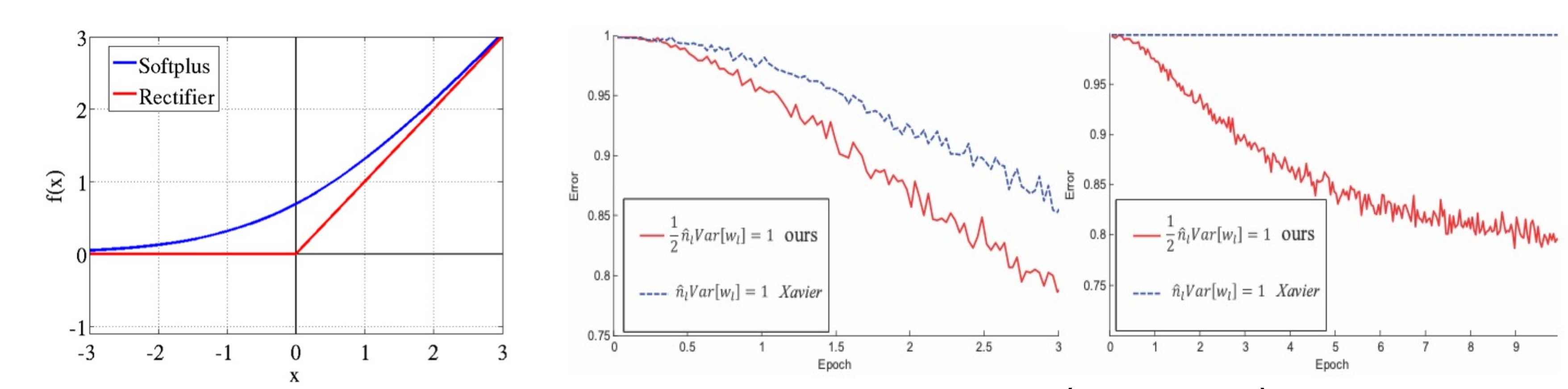
Activation Functions

Classic: Sigmoid → easily saturate at extremes
Modern: ReLU & variants (Leaky ReLU, PReLU, Swish) → avoid saturation, allow gradients to pass through zero **[Glorot et al., 2011]**

Weight Initialization

Classic: Xavier Init → good for tanh/sigmoid, but may cause vanishing gradients in deep ReLU nets

Modern: He Init [He et al., 2015] → designed for ReLU, preserves variance, improves convergence



Activation Functions (ReLU vs. Smooth variants) — Weight Initialization (He Init vs. Xavier)

Reference

1. Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. ICLR.
2. Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the Importance of Initialization and Momentum in Deep Learning. ICML.
3. Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ICML.
4. Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer Normalization. arXiv:1607.06450.
5. Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. AISTATS.
6. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. ICCV.