Issues in Training Neural Networks

Seth J. Chandler, with help from AI

September 10, 2025

Introduction

Neural networks are one of the most powerful tools in modern artificial intelligence. They have been used to achieve remarkable results in image recognition, language translation, medical diagnosis, and countless other fields. Yet, for all their power, training a neural network is not a straightforward process. It is not enough simply to feed data into a model and hope for the best. Training involves a series of decisions, each of which affects the success or failure of the network. These decisions touch on data preparation, model architecture, methods for avoiding overfitting, optimization algorithms, hyperparameter tuning, evaluation metrics, and the human expertise that guides the process.

In this lecture, we will examine the key issues that arise when training neural networks. Our goal is not to provide a detailed mathematical manual, but rather to highlight the essential concepts in a way that is accessible to an audience with a college education and some prior exposure to the basic ideas of neural networks. We will emphasize both the "how" and the "why" behind each choice, showing how the pieces fit together into a coherent whole.

1 Data Preparation and Partitioning

1.1 Why Data Matters

The old adage "garbage in, garbage out" is especially true for neural networks. The quality of a model is limited by the quality of its data. A network cannot learn distinctions that are absent from the dataset, nor can it correct for systematic biases present in the examples. Thus, the very first issue a researcher must confront is: where does the data come from, and what does it represent?

1.2 Splitting the Dataset

When training a neural network, the dataset is typically divided into three parts: the training set, the validation set, and the test set. Each serves a distinct purpose.

- **Training set:** This is the portion of the data used directly to teach the network. The network adjusts its internal parameters (weights) based on this data.
- **Validation set:** This is used to monitor progress during training. By measuring how the model performs on data it has not explicitly seen, we can gauge whether it is genuinely learning or merely memorizing.
- **Test set:** This is reserved until the very end. It is used for the final evaluation of the model's performance. Because it has been untouched during training and validation, it provides the fairest measure of generalization.

1.3 How Splits are Done

The simplest method is random sampling. If you have 10000 examples, perhaps 70% are allocated to training, 15% to validation, and 15% to testing. However, this naive approach can cause problems. If the data is imbalanced (say, 95% of the samples are of one category), then random splitting may create skewed sets. For classification tasks, *stratified sampling* is often used: each split maintains the same proportion of classes as in the full dataset.

For time-series data, where order matters, we cannot simply shuffle. In that case, the training set is taken from earlier time periods, and validation and test sets from later ones, mimicking the real-world scenario where future data must be predicted.

1.4 Why This Matters

The goal of splitting is not just bookkeeping. It is a method for dealing with one of the most fundamental issues in artificial intelligence.

A crucial distinction in both human and machine learning is the difference between *memorizing* and *learning*. Memorizing refers to storing specific examples exactly as they were presented, with little or no ability to apply that knowledge to new situations. Learning, by contrast, involves extracting general principles or patterns that extend beyond the specific examples encountered. This distinction has broad application: a student who memorizes past exam questions may excel only if those identical questions reappear, whereas a student who has truly learned the material can tackle novel problems. In neural networks, as in human education, the goal is generalization—the ability to apply what has been acquired in

one context to fresh and unfamiliar circumstances. Testing, therefore, should always aim to measure learning rather than mere memorization.

A key symptom of memorization rather than learning is "overfitting." The model is complex enough to effectively memorize the examples before it but collapses into massive inaccuracy when it confronts data it has not seen before. The idea of spitting data is to artificially create data the model has not seen before by holding back data that in fact we already have. This may seem "cruel," but it is generally essential.

Without a validation set, we cannot know if the model is overfitting while we are training it. Without a test set, we cannot know how it will perform on genuinely new data. These distinctions are vital: they protect us from fooling ourselves about the network's capabilities.

2 Model Architecture and Design Choices

2.1 Layers as Building Blocks

A neural network is composed of layers. Each layer takes in numbers, transforms them in some way, and passes them to the next. There are different types of layers suited to different problems:

- **Dense layers:** Every input connects to every output. Suitable for general tasks.
- **Convolutional layers:** Specialized for images and spatial patterns. They learn filters that detect edges, shapes, and higher-level structures.
- **Recurrent layers:** Suitable for sequences, such as text or audio, because they can retain information from earlier steps.
- **Attention layers / Transformers:** Currently dominant in language models, they allow the network to selectively focus on different parts of the input.
- **Softmax layers:** Transform raw outputs into probability distributions, often used at the final stage of classification tasks.
- **Dropout layers:** Randomly disable a fraction of connections during training to reduce overfitting and encourage more robust learning.
- **Loss layers:** While not always described as "layers" in the same sense as dense or convolutional ones, they define how error is computed. Common examples include mean squared error (for regression tasks), cross-entropy (for classification), and contrastive loss (for similarity or embedding problems).

There are, in fact, dozens of other layer types developed for specialized purposes, ranging from normalization layers to embedding layers and beyond. The central limitation is practical: each layer must support efficient computation of gradients (to enable learning by backpropagation) and be implementable in parallel on modern hardware. Not every mathematical function can be directly used as a layer, although, in principle, any function can be approximated by combining existing differentiable layers into a larger network.

2.2 Depth and Width

The *depth* of a network refers to the number of layers. The *width* refers to the number of units in each layer. Deeper and wider networks can represent more complex patterns. But greater size comes at a cost: more data is needed to train them, more computational power is required, and the risk of overfitting increases.

2.3 Activation functions

Activation functions introduce non-linearity. Without them, a neural network would collapse into a single linear function, regardless of how many layers it had. Common choices include:

- **ReLU (Rectified Linear Unit):** Outputs zero for negative inputs and the input itself for positive inputs. Simple and effective.
- **Sigmoid:** Squeezes outputs into the range $[0, 1]$. Useful for probabilities, but can suffer from vanishing gradients.
- **Tanh:** Outputs between -1 and 1, with similar issues to sigmoid.
- **SELU (Scaled Exponential Linear Unit):** A self-normalizing activation that automatically keeps outputs in a stable range during training, reducing the need for explicit normalization layers.
- **GELU (Gaussian Error Linear Unit):** Smoothly weights inputs by their value and probability under a Gaussian curve, often improving performance in deep models. Widely used in modern transformer architectures.

While ReLU remains the most common, SELU and GELU have gained popularity in recent years, particularly in deep or very large-scale networks. The choice of activation can subtly affect stability, convergence speed, and ultimately accuracy.

The choice of activation affects both learning speed and stability.

3 Overfitting and Generalization

3.1 The Problem of Overfitting

Overfitting occurs when the network memorizes the training data rather than learning general rules. The result is high accuracy on training data but poor performance on new data. This is one of the central challenges in training neural networks.

3.2 Techniques to Combat Overfitting

Dropout. During training, some connections are randomly disabled. This prevents the network from becoming overly reliant on specific pathways, encouraging more distributed learning.

Regularization. Penalties are added to the training objective to discourage overly complex models. L1 regularization encourages sparsity (many weights close to zero), while L2 discourages excessively large weights.

Data Augmentation. For images, this might mean rotating or flipping pictures. For text, it might mean paraphrasing. The idea is to give the network more variety, so it learns general patterns.

Early Stopping. If the validation loss begins to rise while training loss continues to fall, training is halted. This ensures the model does not drift into overfitting territory.

4 Optimization and Training

4.1 Gradient Descent Basics

Training is the process of adjusting the network's weights so that its predictions are closer to correct. This adjustment is done through *gradient descent*: we compute the slope of the error function with respect to each weight and adjust in the opposite direction.

4.2 Variants of Gradient Descent

Stochastic Gradient Descent (SGD). Instead of computing gradients on the whole dataset, we use small random subsets (mini-batches). This makes training faster and introduces helpful randomness.

Momentum. Adds a memory of past gradients, smoothing the path and helping to escape shallow traps.

Adam. An adaptive method that adjusts learning rates for each parameter individually. It is widely used because it works well out of the box.

4.3 Learning Rate

The learning rate is the step size in gradient descent. It must be chosen carefully:

- Too high: training oscillates wildly and may never converge.
- Too low: training creeps along painfully slowly.
- Just right: training converges smoothly and efficiently.

Often, schedules are used: start with a higher learning rate, then gradually reduce it.

5 Hyperparameters: The Researcher's Dials

Hyperparameters are the settings the researcher chooses before training begins. They are not learned by the network itself. Examples include the number of layers, the dropout rate, and the learning rate.

Tuning hyperparameters is both art and science. One can perform systematic searches (grid search or random search), or rely on experience and intuition. Bayesian optimization and evolutionary algorithms offer more sophisticated approaches, but trial and error remains common.

6 Evaluation and Metrics

6.1 Loss Functions

The loss function is the mathematical measure of error that the network tries to minimize. For classification tasks, cross-entropy loss is common. For regression, mean squared error is typical. The choice of loss function defines the goal of learning.

6.2 Performance Metrics

Loss functions measure error, but metrics express success in more interpretable terms. For classification tasks, accuracy is common, but can be misleading in imbalanced datasets. Other metrics include precision (how many predicted positives are correct), recall (how many actual positives are caught), and the F1 score (a balance of precision and recall). For probabilistic tasks, ROC curves and AUC scores are informative.

7 Practical Constraints

7.1 Computational Resources

Training modern networks requires significant hardware. Graphics processing units (GPUs) and tensor processing units (TPUs) accelerate the necessary calculations. For very large models, training may require multiple machines and distributed strategies.

7.2 Time and Efficiency

Even with specialized hardware, training can take hours, days, or weeks. Efficient coding, good batch sizes, and careful resource management are necessary.

7.3 Reproducibility

For scientific and practical reliability, results must be reproducible. Researchers set random seeds, log experiments, and save checkpoints of trained models.

8 Human Expertise and Intuition

8.1 The Role of Domain Knowledge

Knowing the subject area can guide data collection and model design. For example, in medical imaging, a researcher must understand what kinds of artifacts or features are clinically meaningful.

8.2 Heuristics and Rules of Thumb

With experience, researchers develop an eye for patterns: rising validation loss signals overfitting, unstable gradients suggest learning rate problems, and so forth. These intuitions often save time compared to blind trial and error.

8.3 Iterative Mindset

Training is rarely successful on the first try. Researchers adjust, retrain, and refine. Success comes from patience, careful observation, and willingness to adapt.

Conclusion

Training neural networks is a complex process that combines data preparation, architectural choices, regularization strategies, optimization methods, hyperparameter tuning, evaluation, and human intuition. Each step involves decisions that can profoundly affect the outcome. There is no single formula for success. Instead, effective training is a blend of science and craft: understanding the principles, applying them judiciously, and learning from experience.

By appreciating these issues, even those from non-technical backgrounds can grasp why training neural networks is as much about human judgment as it is about algorithms. It is this combination of rigorous technique and informed intuition that allows neural networks to achieve their remarkable results.