Neural Network Study Guide

A structured summary of key neural network concepts and practical deep learning insights, based on in-depth Q&A and code-driven learning.

---

1. Activation Functions

What is an activation function?
It introduces non-linearity into the model. Without it, multiple layers collapse into a single linear transformation.

Why is it called "activation"?
Inspired by biological neurons — the function decides whether a neuron "fires."

Common Activation Functions

Name	Description
ReLU	`max(0, x)` — simple, fast, sparse
Sigmoid	S-shaped curve, output in [0, 1]
Tanh	Zero-centered version of sigmoid
GELU	Smooth approximation used in Transformers

Mathematical Forms

Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Tanh:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU:

$$\mathrm{ReLU}(x) = \max(0, x)$$

---

2. Non-Linearity

Why do we need non-linearity?
Without it, a stack of layers behaves like one large linear layer. Non-linear functions allow models to learn complex decision boundaries.

Is ReLU linear?
No — ReLU is piecewise linear but not globally linear. The zeroing-out behavior adds non-linearity.

---

3. Dropout and Weight Decay

What does `dropout = 0.3` mean?
30% of neurons are randomly set to zero during each training pass.

What is weight decay?
Also called L2 regularization. It penalizes large weights to reduce overfitting.

$$\text{Loss}_{\text{total}} = \text{Loss}_{\text{task}} + \lambda \sum w_i^2$$

---

4. Batch Size and Epoch

Batch size: number of training examples used in one forward/backward pass.

Epoch: one full pass over the entire dataset.

---

5. Overfitting and Underfitting

- **Overfitting**: Model fits training data too closely and fails to generalize.
- **Underfitting**: Model is too simple to capture patterns.

Bias-Variance Tradeoff

Condition	Bias	Variance	Generalization
Underfitting	High	Low	Poor
Overfitting	Low	High	Poor
Balanced Fit	Low	Low	Good

Variance Formula:

$$\mathrm{Var}(x) = \mathbb{E}\left[(\hat{y}(x) - \mathbb{E}[\hat{y}(x)])^2\right]$$

---

6. Normalization Techniques

Batch Normalization

- Normalizes each feature across the batch.
- Stabilizes training and reduces internal covariate shift.

Layer Normalization

- Normalizes across features for each individual sample.
- Often used in Transformers and RNNs.

Key Difference: - BatchNorm: uses batch statistics - LayerNorm: uses per-sample statistics

---

7. Mean Pooling

Mean pooling summarizes a variable-length sequence (e.g. word embeddings) into a single fixed-size vector.

Process:

Given an embedded sentence of shape:

$$(\text{sequence length} \times \text{embedding dimension})$$

Mean pooling averages across the sequence:

$$\text{Pooled vector} = \frac{1}{n} \sum_{i=1}^{n} \text{Embedding}_i$$

This produces a single vector of size:

$$(1 \times \text{embedding dimension})$$

---

8. Softmax

Softmax turns raw logits into a probability distribution.

Formula:

$$\mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Properties:

- Outputs are in the range (0, 1)
- Sum of outputs = 1
- Often used in the final layer of classification models

Is softmax a type of normalization?
Yes — it normalizes logits into probabilities, but it is **not** the same as z-scoring (mean-variance normalization).

---

9. Attention Mechanism (Transformers)

Key Concepts:

- **Query (Q)**: what we're looking for
- **Key (K)**: what each word has to offer
- **Value (V)**: what each word shares if selected

Formula (Scaled Dot-Product Attention):

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

This computes attention weights via similarity between query and key, scales it, applies softmax, and then multiplies by value.

---

10. Analogy for Attention

Imagine a dating app:

- You (the query) are looking for certain qualities
- Other people (keys) advertise what they offer
- You score them based on alignment
- Their responses (values) contribute to your final impression

---

11. Why Attention Works

- It allows models to focus on relevant parts of input
- Unlike CNNs or RNNs, attention isn't constrained by distance or order

12. Hyperparameter Tuning Strategy

Recommended Tuning Order

Tune parameters **progressively**, from highest to lowest impact:

1. Learning rate (`lr`)
2. Dropout rate
3. Hidden layer size
4. Batch size
5. Weight decay
6. Embedding dimension
7. Optimizer (Adam, SGD)
8. Activation function (ReLU, GELU, etc.)

Why This Order?

- Learning rate heavily affects convergence.
- Dropout and hidden size control overfitting.
- Batch size affects stability and speed.
- Embedding dimension and optimizer are usually fine-tuned last.

---

13. Best Practices for Optimization

- Always use a **train/validation split** to guide tuning.
- Never tune based on the test set.
- Change 1–2 hyperparameters at a time.
- Use validation **accuracy** or **macro F1** for selection.
- Log each experiment and result.

---

14. Evaluation Metrics

Accuracy

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total predictions}}$$

Macro F1

- Averages F1 scores across all classes equally
- Less biased by class imbalance

When Accuracy   Macro F1?

- Classes are balanced
- Model performs uniformly across all classes

---

15. Unknown Words at Eval Time

If a word wasn't in the training vocab: - It is mapped to the `<unk>` token - Too many unknowns can degrade performance

Track unknown word ratio: “‘python num_unk = tokens.count(“”) unk_ratio = num_unk / total_tokens