Neural Network Basics Cheat Sheet

Traditional Feedforward Architectures - Prof. Laio Oriel Seman

🧠 Single Neuron (Building Block)

Step-by-Step Calculation:

1. Linear Combination:

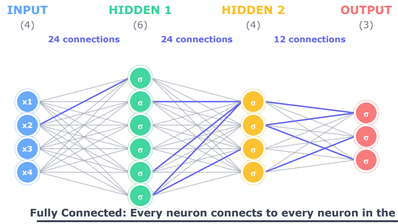
```
z = 0.7×0.5 + (-0.4)×(-0.3) + 0.9×0.8 + 0.2
z = 0.35 + 0.12 + 0.72 + 0.2 = 1.39
```

2. Activation Function:

```
output = σ(1.39) = 1/(1+e^-1.39) ≈ 0.73
```

💡 **Each neuron is a weighted sum followed by a non-linear transformation**

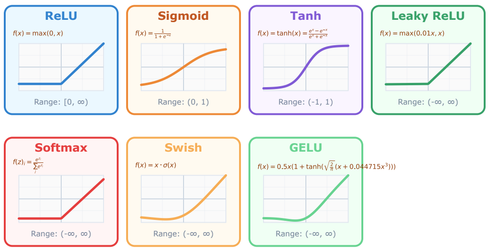
🏛️ Feedforward Architecture



Fully Connected: Every neuron connects to every neuron in the next layer

Raw features — *Pattern detection* — *Feature combination* — *Final prediction*

ACTIVATIONS



📊 Loss Functions

Regression

Mean Squared Error
Penalizes large errors more heavily
`MSE = (1/n)Σ(y - ŷ)²`
Use when: Regression problems, outliers are important

Mean Absolute Error
Robust to outliers
`MAE = (1/n)Σ|y - ŷ|`
Use when: Regression with outliers

Classification

Cross Entropy
Measures difference between probability distributions
`CE = -Σy_i log(ŷ_i)`
Use when: Multi-class classification

Binary Cross Entropy
Specialized for binary classification
`BCE = -y log(ŷ) - (1-y)log(1-ŷ)`
Use when: Binary classification problems

🚀 Optimization

Batch Processing

SGD
Batch: 1 (sample)
✓ Simple and fast
✗ Noisy updates

Mini-Batch SGD
Batch: 32–128
✓ Good balance
✗ Requires tuning

Full Batch
Batch: All data
✓ Stable gradients
✗ Very slow on large datasets

Popular Optimizers

Adam
$m^{-} = \beta_1 m + (1-\beta_1)g$, $v^{-} = \beta_2 v + (1-\beta_2)g^2$
✓ Adaptive learning rate
✓ Combines momentum & RMS
✓ Works well out of the box
Default lr: 0.001
Default choice

RMSprop
$v = \gamma v + (1-\gamma)g^2$, $w \mathrel{-}= \alpha g/\sqrt{v}$
✓ Good for RNNs
✓ Handles non-stationary loss
Default lr: 0.001
RNNs, streaming tasks

AdaGrad
$w \mathrel{-}= \alpha \times g / (\sqrt{G} + \epsilon)$
✓ Good for sparse data
✓ Automatically adjusts lr
Default lr: 0.01
NLP, sparse features

🧱 Layer Types

Dense Layer
(Fully Connected)
`z = Wx + b`
Every neuron connects to all neurons in next layer
• Feature transformation • Classification • Regression

Dropout
(Regularization)
`y = x * mask (training), y = x * p (inference)`
Randomly zeros neurons during training
• Prevent overfitting • Regularization
• Improve generalization

Batch Normalization
(Normalization)
`y = γ((x-μ)/σ) + β`
Normalizes inputs to stabilize training
• Faster training • Higher learning rates
• Reduce internal covariate shift

Layer Normalization
(Normalization)
`y = γ((x-μ)/σ) + β`
Normalizes across features instead of batch
• RNNs, Transformers
• Stabilize training in sequential models

🛡️ Regularization

L1 Regularization
`Loss + λ||w||₁`
Encourages sparsity (some weights become 0)
💡 **Feature selection**
sparse models

L2 Regularization
`Loss + λ||w||₂²`
Reduces weight magnitudes evenly
💡 **Prevent overfitting**
smooth weights

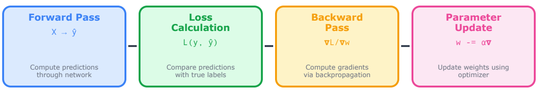
Early Stopping
`Stop when validation loss increases`
Prevents over-training
💡 **Automatic regularization**

⚖️ Weight Initialization

Xavier/Glorot Initialization
`Var(W) = 1/n_in`
Best for: Tanh, Sigmoid activations
Maintains variance across layers

He Initialization
`Var(W) = 2/n_in`
Best for: ReLU family activations
Accounts for ReLU killing half the neurons

🔄 Training Pipeline



Repeat for epochs

Mathematical Foundation:

$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$ $\quad a^{(l)} = \sigma(z^{(l)})$ $\quad \partial L/\partial W^{(l)} = \partial L/\partial z^{(l)} \times (a^{(l-1)})^T$

📉 Learning Rate Schedulers

Step Decay
`lr = lr₀ × γ^{epoch/s}`
Drops lr at fixed steps
💡 **Plateaued loss**
Scheduled decay

Cosine Annealing
`lr = η_min + 0.5(η_max - η_min)(1`
Smooth cyclical decay
💡 **Training from scratch**
Vision tasks

Reduce LR on Plateau
`if val_loss ↑: lr *= factor`
Adaptive on stagnation
💡 **Validation-driven decay**
Auto-tuning

🔗 Skip Connections

`y = F(x) + x`

Key Benefits:

- ✓ Better gradient flow
- ✓ Enables very deep networks
- ✓ Easy to learn identity mapping

Skip connections solve the vanishing gradient problem by providing shortcut paths for gradient flow during backpropagation.

📊 Evaluation Metrics Guide

Classification Metrics

Confusion Matrix

Actual \ Predicted	Pos	Neg
Pos	TP	FP
Neg	FN	TN

Key Metrics:

Accuracy: `(TP + TN) / (TP + TN + FP + FN)`
"Overall correctness"

Precision: `TP / (TP + FP)`
"When I predict positive, how often am I right?"

Recall: `TP / (TP + FN)`
"How many actual positives did I catch?"

F1-Score: `2 × (Precision × Recall) / (Precision + Recall)`
"Harmonic mean of precision and recall"

Roc-Auc: `Area under ROC curve (TPR vs FPR)`
"Measures separability across all thresholds"

Choose Your Metric:
Medical Diagnosis → High Recall (don't miss diseases) | Spam Detection → High Precision (avoid false positives) | Balanced Dataset → Accuracy

Regression Metrics

Mean Absolute Error
$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
✓ Easy to interpret (same units as target) ✓ Robust to outliers

Mean Squared Error
$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
✓ Penalizes large errors more heavily ✓ Smooth gradient

Root Mean Squared Error
$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
✓ Same units as target ✓ Penalizes large errors

R² Score (Coefficient of Determination)
$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$
✓ Scale-independent ✓ Easy interpretation

✨ Best Practices Quick Reference

Architecture Design

- Start simple: 1-2 hidden layers first
- Layer sizes: Powers of 2 (64, 128, 256, 512)
- Activations: ReLU for hidden, task-specific for output
- Add skip connections for networks deeper than 5 layers

Training Tips

- Use Adam optimizer with lr=0.001 as default
- Batch size: 32-128 for most problems
- Monitor both training and validation metrics
- Use early stopping to prevent overfitting

Debugging Guide

- Check learning curves for overfitting/underfitting
- Monitor gradient magnitudes
- Visualize weight distributions
- Start with smaller learning rates if loss explodes

🔧 Debugging & Troubleshooting

⚠️ Vanishing Gradients
Symptoms Deep networks don't train • Early layers learn slowly
Solutions: Use ReLU activation • Add skip connections • Proper weight initialization

⚠️ Exploding Gradients
Symptoms Loss becomes NaN • Weights grow very large
Solutions: Gradient clipping • Lower learning rate • Better weight initialization

⚠️ Overfitting
Symptoms High training accuracy, low validation accuracy • Large gap between training and validation loss
Solutions: Add dropout • L2 regularization • More training data

🧠 Remember: Start simple, validate early, iterate often

Keep experimenting and learning! 🚀