1 Core Concepts & Network Structure

1.1 The Neuron

The fundamental unit of a neural network performing a two-step computation:

1. **Linear Step:** $z = \mathbf{w}^\intercal \mathbf{x} + b = \sum_{i=1}^{n_x} w_i x_i + b$
2. **Activation Step:** $a = g(z)$, where $g(z)$ is a non-linear function.

1.2 Activation Functions

- Sigmoid σ - $\frac{1}{1+e^{-z}}$ Used for binary classification (logistic regression) transforms the output to $\in \{0, 1\}$.
- Softmax - Used for multiclass classification. e^z for every output, then sums up the total before dividing each raised number by the total. This results in a "probability" according to how positive they are with $\in \{0, 1\}$. The result is outputted as a vector of each output probability.
- ReLU - $max\{0, z\}$ - This allows for the derivative to always be 0 or 1 making it not only fast to compute but also fast to backpropagate.
- LeakyReLU $max\{0.1 * z, z\}$ - In order to avoid stagnation from a parameter not being updated leaky ReLU swaps out the 0 for z multiplied by some small number so the derivative of the negative case is still small number.
- tanh

1.3 Network Structure & Forward Prop

A multilayer network is defined by its architecture (e.g., [1, 1, 1] for a 3-layer network). The superscript $[l]$ denotes the layer index.

- L: Total number of layers.
- $n^{[l]}$: Number of neurons in the layer l. n is the number of input features.
- **Forward Propagation (Layer l):** $\mathbf{Z}^{[l]} = \mathbf{W}^{[l]}\mathbf{A}^{[l-1]} + \mathbf{b}^{[l]}$; $\mathbf{A}^{[l]} = g^{[l]}(\mathbf{Z}^{[l]})$
- **Parameter Dimensions (Layer l):** $\mathbf{W}^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$, $\mathbf{b}^{[l]} \in \mathbb{R}^{n^{[l]} \times 1}$

2 Backpropagation & Optimization

2.1 Loss Functions

Loss Function	Equation	When Used
Mean Squared Error (MSE)	$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$	Regression tasks where large errors should be penalized heavily.
Mean Absolute Error (MAE)	$L = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}_i\rvert$	Regression tasks where all errors contribute linearly; more robust to outliers.
Binary Cross-Entropy (Log Loss)	$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$	Binary classification (e.g., logistic regression, binary image classification).
Categorical Cross-Entropy	$L = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$	Multiclass classification, with one true class per example.

2.2 Interpreting Back Propagation

- **Exploding/Vanishing Gradient:** In deep networks, gradients can shrink to zero or explode due to repeated multiplications, poor weight initialization, or saturated activations (sigmoid/tanh).
- **Gradient Checking:** Debugging tool to verify backprop gradients. Include regularization in cost, but disable dropout. Use only for debugging, not during training.

2.3 Backpropagation Algorithm

Backpropagation efficiently computes gradients (∇J) for all parameters in a neural network by applying the chain rule in reverse order—from output to input. During training, it measures how each parameter contributes to the output error, then adjusts the parameters to minimize the loss. This process allows gradient-based optimizers (like SGD or Adam) to update weights efficiently even in deep networks.

- **Output Error:** $d\mathbf{Z}^{[L]} = \mathbf{A}^{[L]} - \mathbf{Y}$
- **Hidden Error:** $d\mathbf{Z}^{[l]} = (\mathbf{W}^{[l+1]\intercal} d\mathbf{Z}^{[l+1]}) * g'^{[l]}(\mathbf{Z}^{[l]})$
- **Gradients:** $d\mathbf{W}^{[l]} = \frac{1}{m} d\mathbf{Z}^{[l]}\mathbf{A}^{[l-1]\intercal}$; $d\mathbf{b}^{[l]} = \frac{1}{m}\text{np.sum}(d\mathbf{Z}^{[l]})$
- **Update:** $\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha\, d\mathbf{W}^{[l]}$

2.4 Exponentially Weighted Average

EWA is a technique used to smooth out sequential or noisy data by giving more weight to recent observations while gradually reducing the influence of older ones. This helps reveal the underlying trend over time without being overly sensitive to short-term fluctuations. $v_t = \beta v_{t-1} + (1-\beta)\theta_t$

where β controls smoothness. Higher β means slower adaptation. Approximate averaging window: $\frac{1}{1-\beta}$ steps (e.g., $\beta = 0.9 \Rightarrow$ 10 days, $\beta = 0.98 \Rightarrow$ 50 days).

2.5 Bias Correction

Early EMA estimates are biased toward zero (since $v_0 = 0$). To correct this, divide by the missing weight factor:

$$v_t = \beta v_{t-1} + (1-\beta)g_t, \quad \hat{v}_t = \frac{v_t}{1-\beta^t}$$

Example ($t = 1$, $\beta = 0.9$): $v_1 = 0.1(-0.058) = -0.0058 \Rightarrow \hat{v}_1 = -0.058$

Bias correction removes the initialization bias, making $\hat{v}_t$ an unbiased estimate.

2.6 Advanced Optimization Algorithms

Momentum: Uses an EWA of past gradients to accelerate in consistent directions and dampen oscillations. The hyperparameter β (typically ~ 0.9) controls the "friction."

$$v_{dW} = \beta v_{dW} + (1-\beta)dW$$

$$W := W - \alpha v_{dW}$$

RMSprop: Adapts the learning rate for each parameter individually by dividing by an EWA of squared gradients. This slows learning on steep dimensions and speeds it up on flatter ones.

$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2$$

$$W := W - \alpha\frac{dW}{\sqrt{S_{dW}}+\epsilon}$$

Adam (Adaptive Moment Estimation): Combines Momentum (1st moment) and RMSprop (2nd moment) with a bias correction step to counteract the zero-initialization of the moving averages, which is helpful in early training stages. It is often the default, robust choice.

1. **Momentum:** $v_{dW} = \beta_1 v_{dW} + (1-\beta_1)dW$
2. **RMSprop:** $S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2$
3. **Correction:** $\hat{v}_{dW} = \frac{v_{dW}}{1-\beta_1^t}$, $\hat{S}_{dW} = \frac{S_{dW}}{1-\beta_2^t}$
4. **Update:** $W := W - \alpha\frac{\hat{v}_{dW}}{\sqrt{\hat{S}_{dW}+\epsilon}}$

Default hyperparameters: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.

Learning Rate Decay: Gradually reduces the learning rate α during training to allow larger updates initially for faster convergence and smaller updates later for fine-tuning. Common strategies include:

- **Step Decay:** Reduce α by a factor every k epochs: $\alpha_t = \alpha_0 \cdot \text{drop}^{\lfloor t/k \rfloor}$
- **Exponential Decay:** Multiply α by a constant factor at each step: $\alpha_t = \alpha_0 \cdot e^{-\lambda t}$
- **1/t Decay (Inverse Time Decay):** Decreases α smoothly over time: $\alpha_t = \frac{\alpha_0}{1+\lambda t}$

Here, t is the current epoch or iteration, λ is the decay rate, and α_0 is the initial learning rate.

3 Regularization & Training

Techniques to overfitting and improve generalization.

3.1 L2 Regularization (Ridge Regression)

Penalizes large weights by adding their squared magnitude (Frobenius norm for matrices) to the cost function. This encourages simpler models.

- **Modified Cost:** $J_{reg} = J + \frac{\lambda}{2m}\sum_{l=1}^{L} ||\mathbf{W}^{[l]}||_F^2$
- **Modified Gradient:** $d\mathbf{W}^{[l]}_{reg} = d\mathbf{W}^{[l]} + \frac{\lambda}{m}\mathbf{W}^{[l]}$

3.1 L1 Regularization (Lasso Regression

Adds a penalty proportional to the absolute value of the weights. This encourages sparsity, meaning many weights are driven to exactly zero, which can be useful for feature selection.

- **Modified Cost:** $J_{reg} = J + \frac{\lambda}{m}\sum_{l=1}^{L} ||\mathbf{W}^{[l]}||_1$

3.2 Dropout Regularization

In order to avoid over resliance on subsets of neurons and balance the load during training, dropout randomly deactivates a fraction of neurons per example, forcing the network to learn more robust features. At test time, dropout is turned off. Another alternative to this is keepProb which works identically but instead is the chance of the neuron staying activated.

3.3 Initialization and Normalization

Weight Initialization: Prevents vanishing/exploding gradients.

- **Xavier/Glorot Init (for Tanh):** Variance $\sigma^2 = 1/n^{[l-1]}$.
- **He Init (for ReLU):** Variance $\sigma^2 = 2/n^{[l-1]}$.

Batch Normalization (BN): Normalizes layer inputs ($Z^{[l]}$) for each mini-batch to have zero mean and unit variance. This stabilizes and accelerates training. Solves covariate shift by normalizing the intermediate activations.

3.4 Hyperparameter Importance

1. α - Learning rate
   - Typically a version of 10^{-n} (1, 0.1, ..., 0.00001)
2. β - Momentum hyperparameter
   - Typically $(1 - 10^{-n})$ (0, 0.9, ..., 0.9999999)
3. $n^{[l]}$ - The number of hidden neurons
4. s - mini batch size
   - Typically an order of 2 as it parallelizes better on the GPU
5. L - Number of layers
6. decayRate - The speed at which the learning rate decrease from epoch to epoch
7. Adam $\beta_1, \beta_2, \epsilon$ - momentum, RMS prop, small term to avoid divide by zero
8. Random search is more effective than grid search

4 Convolutional Neural Networks

Specialized architectures for grid-like data, using parameter sharing to learn spatial hierarchies of features efficiently.

4.1 Core Operations & Layers

- **CONV Layer:** A filter (kernel) of size $f \times f$ slides across the input, computing dot products. Key hyperparameters are filter size (f), padding (p), and stride (s).
- **POOL Layer (Max/Avg):** Downsamples feature maps to reduce dimensionality and create invariance to small translations. Has hyperparameters (f, s) but no learnable parameters.

Output Dimension Formulas:

- **CONV:** $\lfloor\frac{n+2p-f}{s}+1\rfloor$
- **POOL:** $\lfloor\frac{n-f}{s}+1\rfloor$

5 Advanced Applications

5.1 CNN Applications

- **Classification** - outputs a predicted class.
- **Classification** with Localization - In addition to classification it outputs the bounding box for a single image.
- **Object Detection** - Expands on previous application by enabling the model to output multiple objects and their bounding boxes.
- **Landmark Detection** - outputs locations of key parts of an image (e.g. points on a face)

5.2 Object Localization

Extends classification to predict a bounding box for a single object.

- **Output Vector y:** Includes object presence (p_c), bounding box coordinates (b_x, b_y, b_h, b_w), and class probabilities ($c_1, \ldots, c_C$).

$$\mathbf{y} = [p_c, b_x, b_y, b_h, b_w, c_1, \ldots, c_C]^\mathsf{T}$$

- **Hybrid Loss:** A conditional loss is used, applying regression loss (e.g., L2) for the bounding box only when an object is present ($p_c = 1$).

5.3 Object Detection (YOLO)

You Only Look Once (YOLO) is a real-time object detection algorithm that frames the task as a single regression problem.

- **Grid System:** Divides the input image into an $S \times S$ grid. Each cell is responsible for detecting objects whose center falls within it.
- **Anchor Boxes:** Each grid cell predicts multiple bounding boxes using pre-defined "anchor boxes" of various shapes, helping the model detect objects with different aspect ratios.
- **Non-Max Suppression (NMS):** A post-processing step that cleans up multiple, overlapping detections of the same object by discarding boxes with low confidence and suppressing redundant boxes with high Intersection over Union (IoU).

6 Le Math

6.1 Fill up the tensor shapes and calculate the number of learnable parameters

1. Shape Equations

Conv: $n_w^{[i]} = \lfloor\frac{n_w^{[i-1]}+2p^{[i]}-f^{[i]}}{s^{[i]}}+1\rfloor$, $n_h^{[i]} = \lfloor\frac{n_h^{[i-1]}+2p^{[i]}-f^{[i]}}{s^{[i]}}+1\rfloor$

Pool: $n_w^{[i]} = \lfloor\frac{n_w^{[i-1]}-f^{[i]}}{s^{[i]}}+1\rfloor$, $n_h^{[i]} = \lfloor\frac{n_h^{[i-1]}-f^{[i]}}{s^{[i]}}+1\rfloor$

2. Parameter Equations

Conv: $\text{Conv}^{[i]} = n_c^{[i]}(f^{[i]})^2 n_c^{[i-1]} + n_c^{[i]}$

Pool: $\text{Pool}^{[i]} = 0$

FC: $\text{FC}^{[i]} = n_h^{[i-1]} n_w^{[i-1]} n_c^{[i-1]} n_{\text{neu}}^{[i]} + n_{\text{neu}}^{[i]}$

Softmax: $\text{Softmax}^{[i]} = n_{\text{neu}}^{[i]} n_{\text{neu}}^{[i-1]} + n_{\text{neu}}^{[i]}$

Total: $\sum_{i=1}^{L_c}\text{Conv}^{[i]} + \sum_{i=1}^{L_f}\text{FC}^{[i]} + \sum_{i=1}^{L_s}\text{Softmax}^{[i]}$

Variables

n_w, n_h	width, height of feature map
p, f, s	padding, filter size, stride
n_c	number of channels (filters)
n_{neu}	neurons in FC/Softmax layer
L_c, L_f, L_s	total conv, FC, softmax layers
$\lfloor\cdot\rfloor$	floor function

6.2 Suppose, in a logistic regression classifier

$z = x * w^T + b$

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

$a = \sigma(z)$

$L = L(x, y) = -(y * \ln(a) + (1-y) * \ln(1-a))$

$dz = a - y$

$dw_1 = x_1 * dz$

$dw_2 = x_2 * dz$

$db = dz$

$w_1 \leftarrow w_1 - \alpha * dw_1$

$w_2 \leftarrow w_2 - \alpha * dw_2$

$b \leftarrow b - \alpha * db$