Formulas

Definitions

- X: vector of input features, where x_i denotes the i-th input feature
- y: true or target output associated with the input data
- $\hat{y}$: predicted output produced by the neural network
- W: weight matrix with dimension $n \times m$, where m is the number of neurons in the dense layer
- b: bias vector with dimension m
- z: weighted sum of inputs plus the bias term (the logits or pre-activations)
- $h = \sigma(z)$: layer output vector after applying activation function σ element-wise over z

Forward Propagation

$$\hat{y} = f_L((X \cdot W_1 + b_1) \cdot W_2 + b_2 \cdot \ldots \cdot W_L + b_L)$$

Where:

- L: output layer
- f_L: activation function applied at the output layer L
- X: vector of input features
- W_l: weight matrix of layer l
- b_l: bias vector of layer l

Layers

Dense

The formula for the operation performed by a single dense layer is a neural network is:

$$z = W \cdot x + b$$

Where:

- z: weighted sum of inputs plus the bias term
- W: weight matrix of the dense layer with dimension $n \times m$, where m is the number of neurons in the dense layer
- x: input vector to the dense layer with dimension n
- b: bias vector of the dense layer with dimension m

Or in matrix notation:

$$z_j = \sum_{i=1}^{n} w_{ij} \cdot x_i + b_j$$

Where:

- z_j: j-th element of the output vector z
- n: number of neurons in the layer
- w_{ij}: the weight connecting the i-th input neuron to the j-th neuron in the dense layer
- x_i: the i-th element of input vector x
- b_j: the bias term for the j-th neuron in the dense layer

Dropout

A binomial experiment consists of a fixed number n of statistically independent Bernoulli trials, each with a probability of success p, and counts the number of successes. A random variable corresponding to a binomial experiment denoted by $B(n,p)$ has a binomial distribution. The probability of exactly k successes in experiment $B(n,p)$ is given by:

$$P(k) = B(n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Where:

- $\binom{n}{k}$: binomial coefficient
- n: number of trials
- k: number of successes
- p: probability of success
- $(1-p)$: probability of failure

The Bernoulli distribution is a special case of the binomial distribution with $n = 1$ and $k \in 0, 1$:

$$B(1,p) = p^k (1-p)^{1-k}$$

Or equivalently:

$$B(1,p) = \begin{cases} p & \text{if } x = 1 \\ 1-p & \text{if } x = 0 \end{cases}$$

Draw a random binary mask from the Bernoulli distribution described above with probability p to generate a sequence of binary values (0s and 1s). Then the formula for the operation of a dropout layer is given as:

$$z = x \cdot B(1,p) \frac{1}{1-p}$$

Where:

- z: output after dropout
- x: input vector
- $B(1,p)$: Bernoulli distribution with probability p
- p: probability of success

When dropout is applied during training, a fraction p of neuron outputs are randomly set to zero. Thus, the remaining fraction $1-p$ of neuron outputs are scaled up by a factor of $\frac{1}{1-p}$ to compensate for the dropped neurons. During inference, dropout is not applied.

Recurrent

Placeholder

Convolutional

Placeholder

Activation Functions

Linear

$$\sigma(x) = x$$

Where:

- $\sigma(x)$: linear activation function
- x: input vector

Sigmoid

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

Where:

- $\sigma(x)$: sigmoid activation function
- e: Euler's number, or the base of the natural logarithm
- x: input vector

Rectifier

$$\sigma(x) = \max(0, x)$$

Where:

- $\sigma(x)$: rectifier or Rectified Linear Unit activation function
- $\max(0, x)$: function that returns 0 or x depending on which one is larger
- x: input vector

Softmax

$$\sigma_i(z) = \frac{\mathrm{e}^{z_i}}{\sum_{j=1}^{N} \mathrm{e}^{z_{ij}}}$$

Where:

- $\sigma(z)_i$: probability assigned to the i-th class after applying the Softmax function
- z_i: logit corresponding to the i-th class
- e: Euler's number, or the base of the natural logarithm
- N: total number of classes

Loss

Mean Absolute Error

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Where:

- $\hat{y}_i$: predicted value for the i-th sample
- y_i: true value for the i-th sample
- $|\cdot|$: absolute value

Mean Squared Error

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Where:

- $\hat{y}_i$: predicted value for the i-th sample
- y_i: true value for the i-th sample

Binary Cross-Entropy

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \, \log(\hat{y}_i) + (1 - y_i) \, \log(1 - \hat{y}_i))$$

Where:

- N: number of classes
- y_i: truth label for the i-th sample
- $\hat{y}_i$: predicted value (0 or 1) for the i-th sample

Categorical Cross-Entropy

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \, \log(\hat{y}_i)$$

Where:

- N: number of samples
- y_i: truth label for the i-th sample
- $\hat{y}_i$: predicted value (0 or 1) for the i-th sample

When y is expressed as one-hot vectors, the equation simplifies to:

$$\mathcal{L} = -\frac{1}{N}\log(\hat{y}_{i,k})$$

Where k is the index of the target label.

Backward Propagation

The backpropagation process involves the following steps:

1. Compute the error signals δ for each layer l. An error signal is the gradient of the loss function with respect to the activations a and its formula thus depends on the chosen activation and loss functions.

   For the final layer L the error signal is given as:

$$\begin{aligned}\delta_L &= \frac{\partial \mathcal{L}}{\partial a_L} \odot \sigma'_L(z_L) \\ &= \mathcal{L}'_L \odot \sigma'_L(z_L)\end{aligned}$$

   Where:

   - $\odot$ denotes element-wise multiplication
   - $\frac{\partial \mathcal{L}}{\partial a_L}$ or $\mathcal{L}'_L$: partial derivative of the loss function with respect to the activations of layer L
   - a_L: activations of layer L
   - σ'_L: derivative of activation function σ of layer L
   - z_L: pre-activation matrix or logits of layer L ($z_L = W_L \cdot a_L + b_L$)

   The error signal at hidden layers is computed based on the error signals of the subsequent layers and the weights connecting the current layer to the following ones. The formula involves backpropagating the error signal from the subsequent layer (δ_{l+1}) through the weights, combined with the derivative of the activation function in the current layer σ'_l:

$$\delta_l = (W_l^T \cdot \delta_{l+1}) \odot \sigma'_l(z_l)$$

   Where:

   - $\cdot$ denotes the dot product
   - W_l: weight matrix of layer l
   - σ'_l: derivative of activation function σ of layer l
   - z_l: pre-activation matrix or logits of layer l ($z_l = W_l \cdot a_{l+1} + b_l$)

2. Compute the parameters (weights and biases) for each layer using the error signal and the activations of the previous layer:

$$\begin{aligned}\nabla W_l &= \frac{\partial \mathcal{L}}{\partial W_l} = a_{l+1} \cdot \delta_l \\ \nabla b_l &= \frac{\partial \mathcal{L}}{\partial b_l} = \sum_{i=1}^{n} \delta_{l(ij)}\end{aligned}$$

   Where $\delta_{l(ij)}$ denotes the error signal (gradient) for the j-th neuron in layer l for the i-th sample in the batch. The summation sums up all the rows of the error signal matrix along the columns.

3. Update the parameters using an optimizer such as gradient descent:

$$\begin{aligned}W_l &\leftarrow W_l - \alpha \cdot \nabla W_l \\ b_l &\leftarrow b_l - \alpha \cdot \nabla b_l\end{aligned}$$

   Where α is the learning rate. For another optimizer this equation will be more complex.

Activation

In the backward step for an activation function σ its derivative with respect to the logits z (pre-activation values of a layer) must be calculated and is represented as:

$$\frac{\partial \mathcal{L}}{\partial \sigma} = \sigma'(z)$$

Linear

$$\sigma'(z) = 1$$

The backward pass for a linear activation function simply involves passing the error signal backward unchanged through the layer.

Sigmoid

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$$

Rectified Linear Unit

$$\sigma'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Softmax

$$\sigma'(z) = \frac{\partial \hat{y}_i}{\partial z_j} = \begin{cases} \hat{y}_i - \hat{y}_i \hat{y}_j & \text{when } i = j \\ -\hat{y}_i \hat{y}_j & \text{when } i \neq j \end{cases}$$

Where:

- $\hat{y}_i$: i-th element of the Softmax function predictions $\hat{y}$
- z_j: j-th element of the Softmax function input vector z

Loss

Mean Absolute Error

$$\mathcal{L}'(\hat{y}_i) = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{1}{n} \text{sign}(y_i - \hat{y}_i)$$

Where

- $\hat{y}_i$: i-th element of the Softmax function predictions $\hat{y}$
- y_i: i-th element of the target label vector y
- sign(x): -1 if $x < 0, 0$ if $x = 0,$ and 1 if $x > 0$

Mean Squared Error

$$\mathcal{L}'(\hat{y}_i) = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = -\frac{1}{n} 2(\hat{y}_i - y_i)$$

Where:

- $\hat{y}_i$: i-th element of the Softmax function predictions $\hat{y}$
- y_i: i-th element of the target label vector y

Binary Cross-Entropy

$$\mathcal{L}'(\hat{y}_i) = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = -\frac{1}{n} \left[\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i} \right]$$

Where:

- $\hat{y}_i$: i-th element of the Softmax function predictions $\hat{y}$
- y_i: i-th element of the target label vector y

Categorical Cross-Entropy

$$\mathcal{L}'(\hat{y}_i) = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = -\frac{1}{n}\frac{y_i}{\hat{y}_i}$$

Where:

- $\hat{y}_i$: i-th element of the Softmax function predictions $\hat{y}$
- y_i: i-th element of the target label vector y

Softmax Categorical Cross-Entropy

$$\mathcal{L}'(z_i) = \frac{\partial \mathcal{L}}{\partial z_i} = \hat{y}_i - y_i$$

Where:

- z_i: i-th element of the Softmax function input vector z
- $\hat{y}_i$: i-th element of the Softmax function predictions $\hat{y}$
- y_i: i-th element of the target label vector y

Optimization

Stochastic Gradient Descent

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t)$$

Where:

- θ_t: value of the parameter at time step t
- η: learning rate
- $\nabla \mathcal{L}(\theta_t)$: gradient of the loss function $\mathcal{L}$ with respect to parameter θ at time step t

Momentum

$$\begin{aligned} v_{t+1} &= \gamma \cdot v_t + (1-\gamma)\nabla \mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t - \eta \cdot v_{t+1} \end{aligned}$$

Where:

- θ_t: value of the parameter at time step t
- η: learning rate
- $\nabla \mathcal{L}(\theta_t)$: gradient of the loss function $\mathcal{L}$ with respect to parameter θ at time step t
- v_t: moving average of the gradients at time step t
- γ: momentum hyperparameter (value between 0 and 1)

AdaGrad

$$\begin{aligned} v_{t+1} &= v_t + \nabla \mathcal{L}(\theta_t)^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon} \nabla \mathcal{L}(\theta_t) \end{aligned}$$

Where:

- θ_t: value of the parameter at time step t
- η: learning rate
- $\nabla \mathcal{L}(\theta_t)$: gradient of the loss function $\mathcal{L}$ with respect to parameter θ at time step t
- v_t: sum of squares of past gradients up to time step t
- ϵ: small constant (usually on the order of 10^8) to avoid division by zero

AdaDelta

$$\begin{aligned} v_{t+1} &= \rho v_t + (1-\rho)\nabla \mathcal{L}(\theta_t)^2 \\ \Delta\theta_{t+1} &= -\frac{\sqrt{\Delta\theta_t} + \epsilon}{\sqrt{v_{t+1}} + \epsilon} \nabla \mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t + \Delta\theta_{t+1} \end{aligned}$$

Where:

- θ_t: value of the parameter at time step t
- $\nabla \mathcal{L}(\theta_t)$: gradient of the loss function $\mathcal{L}$ with respect to parameter θ at time step t
- v_t: exponentially decaying moving average of squared gradients at time step t
- ρ: decay rate for the moving average, typically close to 1 (e.g. 0.95)
- ϵ: small constant to avoid division by zero

- $\Delta\theta_{t+1}$: update term for the parameter at time step $t+1$, computed based on the root mean square of the parameter updates
- $\Delta\theta_t$: exponentially decaying moving average of squared parameter updates at time step t

RMSProp

$$\begin{aligned} v_{t+1} &= \rho\, v_t + (1-\rho)\nabla\mathcal{L}(\theta_t)^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_{t+1}}+\epsilon}\nabla\mathcal{L}(\theta_t) \end{aligned}$$

Where:

- θ_t: value of the parameter at time step t
- η: learning rate
- $\nabla\mathcal{L}(\theta_t)$: gradient of the loss function $\mathcal{L}$ with respect to parameter θ at time step t
- v_t: exponentially decaying moving average of squared gradients at time step t
- ρ: decay rate, typically set to a value close to 1 (e.g. 0.9)
- ϵ: small constant to avoid division by zero

Adam

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1)\nabla\mathcal{L}(\theta_t) \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2)\nabla\mathcal{L}(\theta_t)^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^t} \\ \hat{v}_{t+1} &= \frac{v_{t+1}}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}}+\epsilon}\hat{m}_{t+1} \end{aligned}$$

Where:

- θ_t: value of the parameter at time step t
- η: learning rate
- $\nabla\mathcal{L}(\theta_t)$: gradient of the loss function $\mathcal{L}$ with respect to parameter θ at time step t
- m_t and v_t are the first and second moments (moving averages) of the gradients at time step t
- $\hat{m}_{t+1}$ and $\hat{v}_{t+1}$ are bias-corrected estimates of the first and second moments
- β_1 and β_2 are the decay rates for the moving averages, typically close to 1 (e.g. 0.9 and 0.999, respectively)
- ϵ: small constant to avoid division by zero