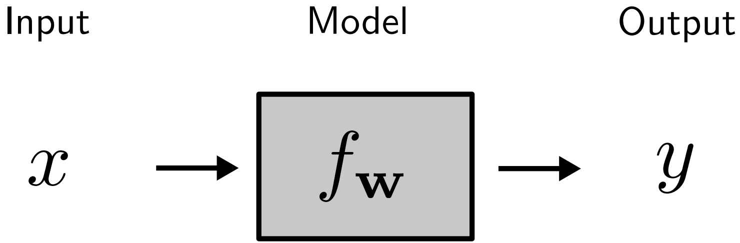
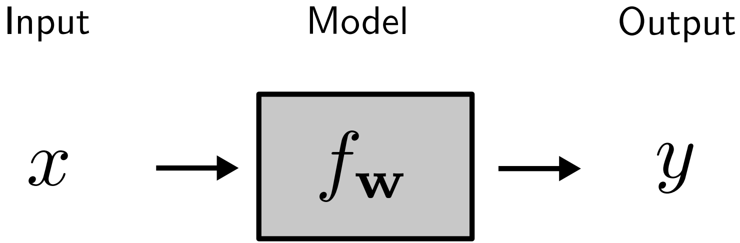
2.1
Logistic Regression

By: Andreas Ginger

Supervised Learning

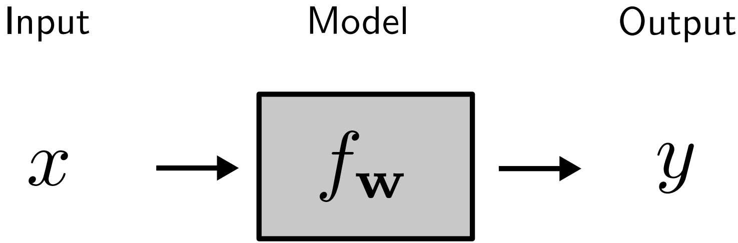


Supervised Learning



- **Learning:** Estimate parameters $\mathbf{w}$ from training data $\{(x_i, y_i)\}_{i=1}^{N}$

Supervised Learning



- **Learning:** Estimate parameters $\mathbf{w}$ from training data $\{(x_i, y_i)\}_{i=1}^N$
- **Inference:** Make novel predictions: $y = f_{\mathbf{w}}(x)$

Regression

Input

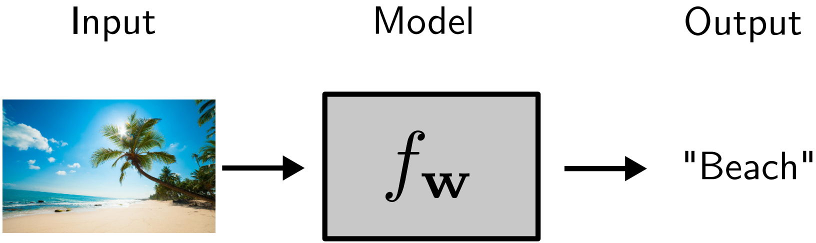
Model

Output

$\rightarrow$ $f_{\mathbf{w}}$ $\rightarrow$ 143,52 €

- **Mapping:** $f_{\mathbf{w}} : \mathbb{R}^N \rightarrow \mathbb{R}$

Classification



- **Mapping:** $f_{\mathbf{w}} : \mathbb{R}^{W \times H} \rightarrow$ {"Beach", "No Beach"}
- Classification will be the topic of today

Logistic Regression

Conditional **Maximum Likelihood Estimator** for $\mathbf{w}$:

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{model}(y_i|\mathbf{x}_i, \mathbf{w})$$

- We now like to perform binary classification: $y_i \in \{0, 1\}$
- How should we choose $p_{model}(y|\mathbf{x}, \mathbf{w})$ in this case?
- Answer: Bernoulli distribution

Predição

Verdade

$$p_{model}(y|\mathbf{x}, \mathbf{w}) = \hat{y}^y \, (1 - \hat{y})^{(1-y)}$$

with $\hat{y}$ predicted by a model: $\hat{y} = f_{\mathbf{w}}(\mathbf{x})$

Logistic Regression

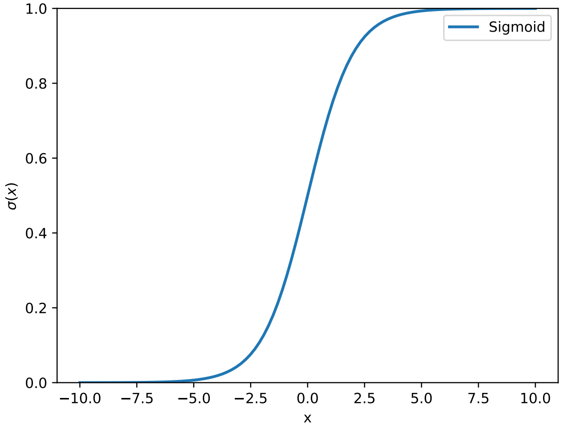
We assumed a Bernoulli distribution

$$p_{model}(y|\mathbf{x}, \mathbf{w}) = \hat{y}^y \, (1 - \hat{y})^{(1-y)}$$

with $\hat{y}$ shorthand for $\hat{y} = f_{\mathbf{w}}(\mathbf{x})$.

- But how to choose $f_{\mathbf{w}}(\mathbf{x})$?
- Requirement: $f_{\mathbf{w}}(\mathbf{x}) \in [0, 1]$
- Choose $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$
  where σ is the sigmoid function:

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



Logistic Regression

Putting it together:

$$
\begin{aligned}
\hat{\mathbf{w}}_{ML} &= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{model}(y_i|\mathbf{x}_i, \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \left[\hat{y}_i^{y_i} \, (1-\hat{y}_i)^{(1-y_i)} \right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{-y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)}_{\text{Binary Cross Entropy Loss } \mathcal{L}(\hat{y}_i, y_i)}
\end{aligned}
$$

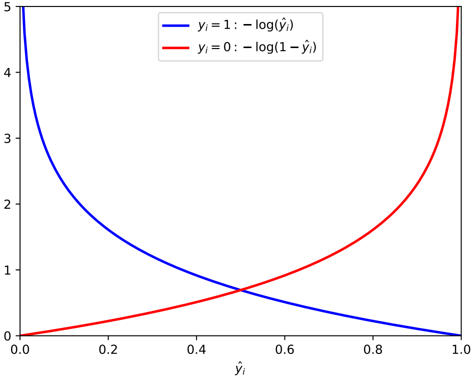
- In ML, we use the more general term “loss function” rather than “error function”
- Interpretation: We minimize the dissimilarity between the empirical data distribution p_{data} (defined by the training set) and the model distribution p_{model}

Logistic Regression

Binary Cross Entropy Loss:

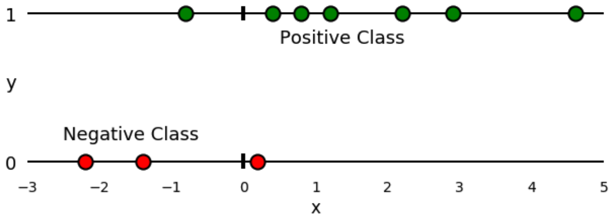
$$\mathcal{L}(\hat{y}_i, y_i) = -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)$$
$$= \begin{cases} -\log \hat{y}_i & \text{if } y_i = 1 \\ -\log(1 - \hat{y}_i) & \text{if } y_i = 0 \end{cases}$$

- For $y_i = 1$ the loss $\mathcal{L}$ is minimized if $\hat{y}_i = 1$
- For $y_i = 0$ the loss $\mathcal{L}$ is minimized if $\hat{y}_i = 0$
- Thus, $\mathcal{L}$ is minimal if $\hat{y}_i = y_i$
- Can be extended to > 2 classes



Logistic Regression

A simple 1D example:

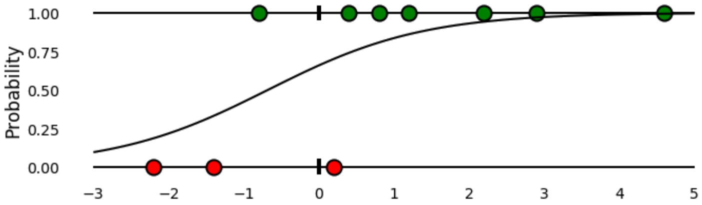


- Dataset $\mathcal{X}$ with positive $(y_i = 1)$ and negative $(y_i = 0)$ samples

Source: `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`

Logistic Regression

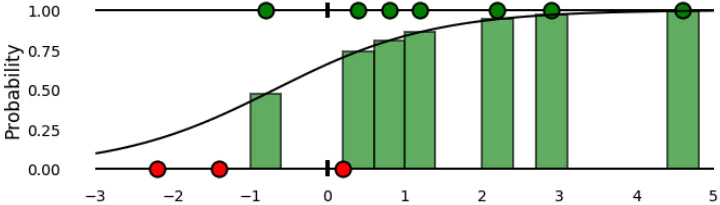
A simple 1D example:



- Logistic regressor $f_{\mathbf{w}}(x) = \sigma(w_0 + w_1 x)$ fit to dataset $\mathcal{X}$

Logistic Regression

A simple 1D example:

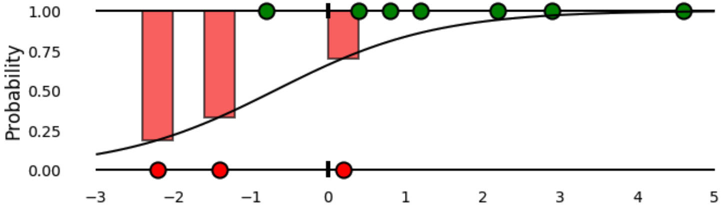


- Probabilities of classifier $f_{\mathbf{w}}(x_i)$ for positive samples $(y_i = 1)$

Source: `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`

Logistic Regression

A simple 1D example:

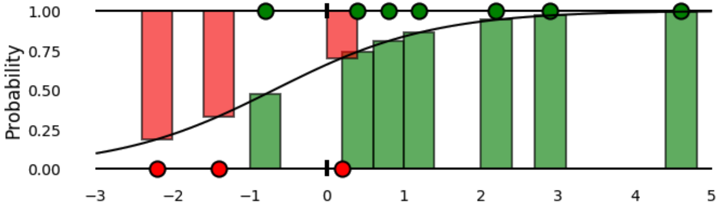


- Probabilities of classifier $f_{\mathbf{w}}(x_i)$ for negative samples $(y_i = 0)$

Source: `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`

Logistic Regression

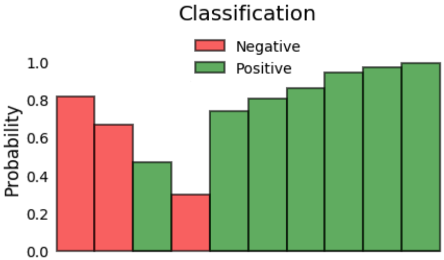
A simple 1D example:



- Putting both together

Logistic Regression

A simple 1D example:

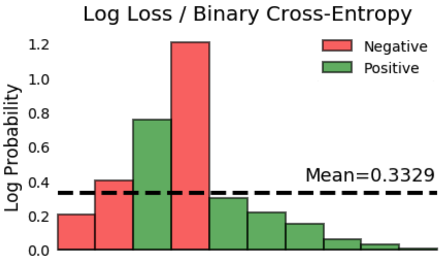


- Let's get rid of the x axis

Source: `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`

Logistic Regression

A simple 1D example:



- And finally compute the negative logarithm: $-\log(f_{\mathbf{w}}(x_i))$

Source: `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`

Logistic Regression

Maximum Likelihood for Logistic Regression:

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)}_{\text{Binary Cross Entropy Loss } \mathcal{L}(\hat{y}_i, y_i)}$$

with $\hat{y} = f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$ and $\sigma(x) = \frac{1}{1 + e^{-x}}$

How do we find the minimizer $\hat{\mathbf{w}}$?

- In contrast to linear regression, the loss $\mathcal{L}(\hat{y}_i, y_i)$ is **not quadratic** in $\mathbf{w}$
- We must apply iterative gradient-based optimization. The gradient is given by:

$$\nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)\mathbf{x}_i$$

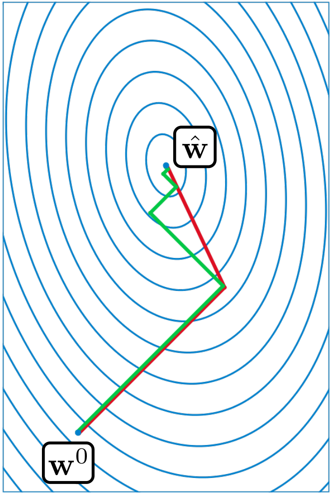
Logistic Regression

Gradient Descent:

- Pick step size η and tolerance ϵ
- Initialize $\mathbf{w}^0$
- Repeat until $\|\mathbf{v}\| < \epsilon$
  - $\mathbf{v} = \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \mathbf{v}$

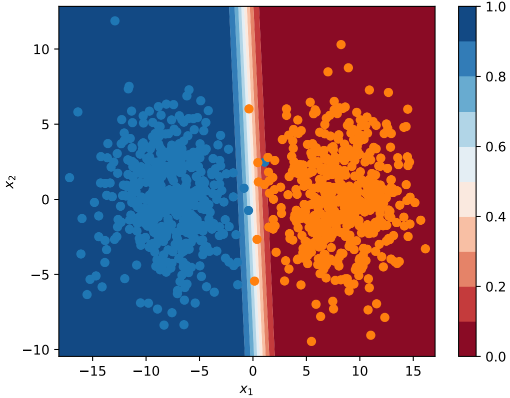
Variants:

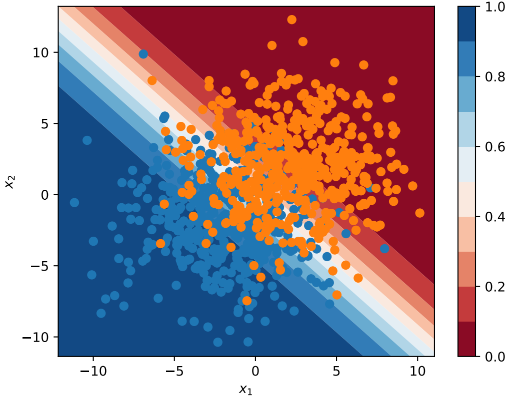
- Line search (green)
- Conjugate gradients (red)
- L-BFGS



Logistic Regression

Examples with two-dimensional inputs $(x_1, x_2) \in \mathbb{R}^2$**:**





- Logistic regression model: $f_{\mathbf{w}}(x_1, x_2) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$

Information Theory

Kullback-Leibler = diferença entre duas distributions

Maximizing the **Log-Likelihood** is equivalent to minimizing **Cross Entropy** or **KL Divergence:**

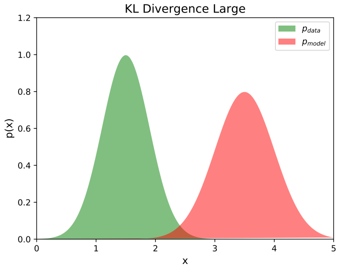
$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w}}{\operatorname{argmax}} \underbrace{\sum_{i=1}^{N} \log p_{model}(y_i|\mathbf{x}_i, \mathbf{w})}_{\text{Log-Likelihood}}$$

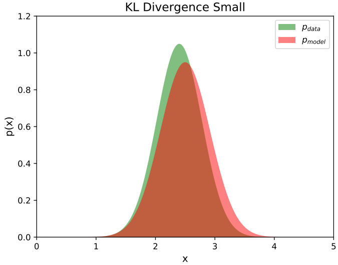
$$= \underset{\mathbf{w}}{\operatorname{argmax}} \ \mathbb{E}_{p_{data}} \left[\log p_{model}(y|\mathbf{x}, \mathbf{w})\right]$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{-\mathbb{E}_{p_{data}} \left[\log p_{model}(y|\mathbf{x}, \mathbf{w})\right]}_{\text{Cross Entropy } H(p_{data}, p_{model})}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \ \mathbb{E}_{p_{data}} \left[\log p_{data}(y|\mathbf{x}) - \log p_{model}(y|\mathbf{x}, \mathbf{w})\right]$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{D_{KL}(p_{data} \| p_{model})}_{\text{KL Divergence}}$$





2.2
Computation Graphs

Logistic Regression

Maximum Likelihood for Logistic Regression:

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)}_{\text{Binary Cross Entropy Loss } \mathcal{L}(\hat{y}_i, y_i)}$$

with $\quad \hat{y} = f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) \quad$ and $\quad \sigma(x) = \dfrac{1}{1 + e^{-x}}$

- Minimization of a **non-linear objective** requires the calculation of gradients $\nabla_{\mathbf{w}}$
- Luckily, in the above case the gradient is simple: $\nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)\mathbf{x}_i$
- But this is not true for more complex models such as deep neural networks
- How can we **efficiently** compute gradients in the general case?

Computation Graphs

Key Idea:

- **Decompose** complex computations into sequence of atomic assignments
- We call this sequence of assignments a **computation graph** or **source code**
- The **forward pass** takes a training point $(\mathbf{x}, y)$ as input and computes a loss, e.g.:

$$\mathcal{L} = -\log p_{model}(y|\mathbf{x}, \mathbf{w})$$

- As we will see, gradients $\nabla_{\mathbf{w}}\mathcal{L}$ can be computed using a **backward pass**
- Both, the forward pass and the backward pass are **efficient** due to the use of dynamic programming, i.e., storing and reusing intermediate results
- This decomposition and reuse of computation is key to the success of the **backpropagation algorithm**, the primary workhorse of deep learning

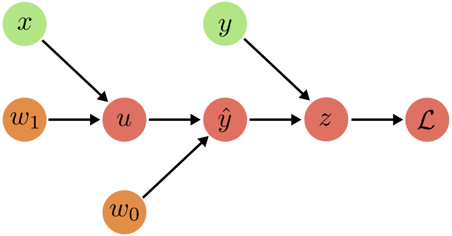
Computation Graphs

A **computation graph** has three kinds of nodes:

- Input nodes
- Parameter nodes
- Compute nodes

Example: Linear Regression

$$
\begin{aligned}
(1) \quad u &= w_1 x \\
(2) \quad \hat{y} &= w_0 + u \\
(3) \quad z &= \hat{y} - y \\
(4) \quad \mathcal{L} &= z^2
\end{aligned}
$$



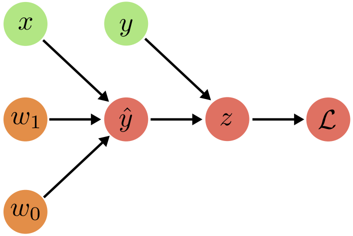
Computation Graphs

A **computation graph** has three kinds of nodes:

- Input nodes
- Parameter nodes
- Compute nodes

Example: Linear Regression

$$
\begin{aligned}
(1)\quad & \hat{y} = w_0 + w_1 x \\
(2)\quad & z = \hat{y} - y \\
(3)\quad & \mathcal{L} = z^2
\end{aligned}
$$



Computation Graphs

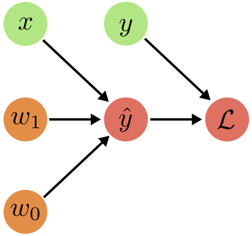
A **computation graph** has three kinds of nodes:

- Input nodes
- Parameter nodes
- Compute nodes

Example: Linear Regression

$$(1) \quad \hat{y} = w_0 + w_1 x$$

$$(2) \quad \mathcal{L} = (\hat{y} - y)^2$$



Computation Graphs

A **computation graph** has three kinds of nodes:

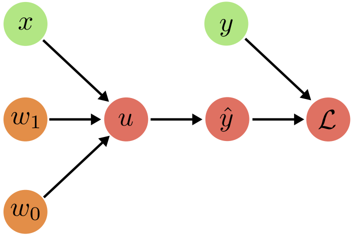
- Input nodes
- Parameter nodes
- Compute nodes

Example: Logistic Regression

$$(1) \quad u = w_0 + w_1 x$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



Computation Graphs

A **computation graph** has three kinds of nodes:

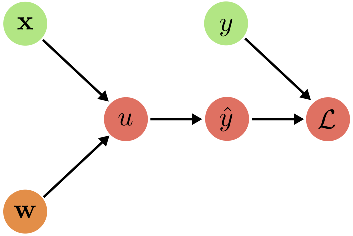
- Input nodes
- Parameter nodes
- Compute nodes

Example: Logistic Regression

$$(1) \quad u = \mathbf{w}^\top \mathbf{x}$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$



Computation Graphs

A **computation graph** has three kinds of nodes:

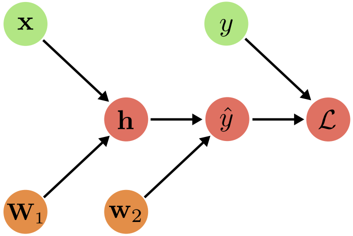
- Input nodes
- Parameter nodes
- Compute nodes

Example: Multi-Layer Perceptron

$$(1) \quad \mathbf{h} = \sigma(\mathbf{W}_1^\top \mathbf{x})$$

$$(2) \quad \hat{y} = \sigma(\mathbf{w}_2^\top \mathbf{h})$$

$$(3) \quad \mathcal{L} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



2.3
Backpropagation

Backpropagation

Goal: Find gradients of negative log likelihood

$$\nabla_{\mathbf{w}} \sum_{i=1}^{N} \underbrace{-\log p_{model}(y_i|\mathbf{x}_i, \mathbf{w})}_{\mathcal{L}(y_i, \mathbf{x}_i, \mathbf{w})}$$

or more generally of a loss function

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{y}, \mathbf{X}, \mathbf{w}) = \nabla_{\mathbf{w}} \sum_{i=1}^{N} \mathcal{L}(y_i, \mathbf{x}_i, \mathbf{w}) = \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathcal{L}(y_i, \mathbf{x}_i, \mathbf{w})$$

given a dataset $\mathcal{X} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ with N elements. In the following, we consider the computation of gradients wrt. a single data point: $\nabla_{\mathbf{w}} \mathcal{L}(y_i, \mathbf{x}_i, \mathbf{w})$. The gradient with respect to the entire dataset $\mathcal{X}$ is obtained by summing up all individual gradients.

Chain Rule

Chain Rule:

$$\frac{\mathrm{d}}{\mathrm{d}x} f(g(x)) = \frac{\mathrm{d}f}{\mathrm{d}g} \frac{\mathrm{d}g}{\mathrm{d}x}$$

Multivariate Chain Rule:

$$\frac{\mathrm{d}}{\mathrm{d}x} f(g_1(x), \ldots, g_M(x)) = \sum_{i=1}^{M} \frac{\partial f}{\partial g_i} \frac{\mathrm{d}g_i}{\mathrm{d}x}$$

Backpropagation

For now: no distinction between node types (input, parameter, compute)

Forward Pass:

$$(1) \quad y = x^2$$

$$(2) \quad \mathcal{L} = 2y$$

Loss: $\mathcal{L} = 2x^2$

Backpropagation

For now: no distinction between node types (input, parameter, compute)

Forward Pass:

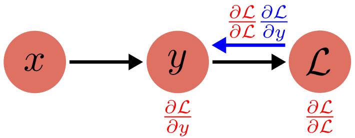
$$(1) \quad y = x^2$$

$$(2) \quad \mathcal{L} = 2y$$

Loss: $\mathcal{L} = 2x^2$

Backward Pass:

$$(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial y} = 2$$



- **Red:** back-propagated gradients
- **Blue:** local gradients

Backpropagation

For now: no distinction between node types (input, parameter, compute)

Forward Pass:

$$(1) \quad y = x^2$$

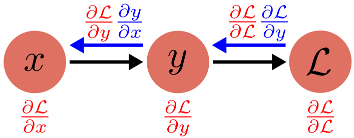
$$(2) \quad \mathcal{L} = 2y$$

Loss: $\mathcal{L} = 2x^2$

Backward Pass:

$$(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial y} = 2$$

$$(1) \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} 2x$$



- **Red:** back-propagated gradients
- **Blue:** local gradients

Backpropagation

For now: no distinction between node types (input, parameter, compute)

Forward Pass:

$$(1) \quad y = x^2$$

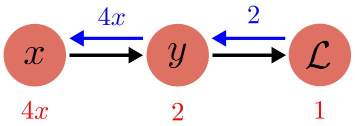
$$(2) \quad \mathcal{L} = 2y$$

Loss: $\mathcal{L} = 2x^2$

Backward Pass:

$$(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial y} = 2$$

$$(1) \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} 2x$$



- **Red:** back-propagated gradients
- **Blue:** local gradients

Backpropagation: A more abstract Example

For now: no distinction between node types (input, parameter, compute)

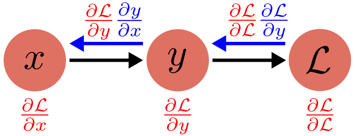
Forward Pass:

$$
\begin{aligned}
(1) \quad y &= y(x) \\
(2) \quad \mathcal{L} &= \mathcal{L}(y)
\end{aligned}
$$

Loss: $\mathcal{L}(y(x))$

Backward Pass:

$$
\begin{aligned}
(2) \quad {\color{red}\frac{\partial \mathcal{L}}{\partial y}} &= {\color{red}\frac{\partial \mathcal{L}}{\partial \mathcal{L}}}{\color{blue}\frac{\partial \mathcal{L}}{\partial y}} = {\color{blue}\frac{\partial \mathcal{L}}{\partial y}} \\
(1) \quad {\color{red}\frac{\partial \mathcal{L}}{\partial x}} &= {\color{red}\frac{\partial \mathcal{L}}{\partial y}}{\color{blue}\frac{\partial y}{\partial x}}
\end{aligned}
$$



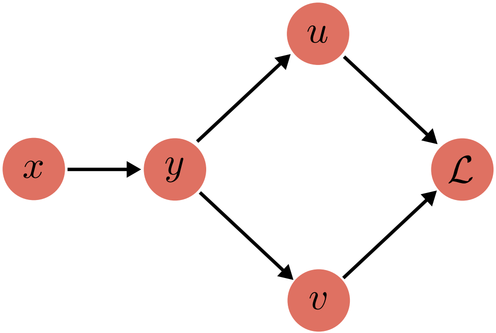
- **Red:** back-propagated gradients
- **Blue:** local gradients

Backpropagation: Fan-Out > 1

Forward Pass:

$$
\begin{aligned}
(1) \quad & y = y(x) \\
(2) \quad & u = u(y) \\
(2) \quad & v = v(y) \\
(3) \quad & \mathcal{L} = \mathcal{L}(u, v)
\end{aligned}
$$

Loss: $\mathcal{L}(\, u(y(x)),\, v(y(x))\,)$



Backpropagation: Fan-Out > 1

Forward Pass:

$$(1) \quad y = y(x)$$

$$(2) \quad u = u(y)$$

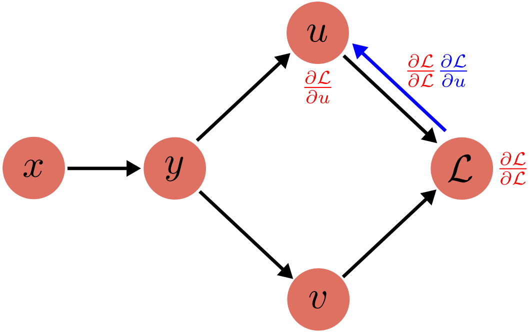
$$(2) \quad v = v(y)$$

$$(3) \quad \mathcal{L} = \mathcal{L}(u, v)$$

Backward Pass:

$$(3) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u}$$

Loss: $\mathcal{L}(\, u(y(x)),\, v(y(x))\,)$



Backpropagation: Fan-Out > 1

Forward Pass:

$$(1) \quad y = y(x)$$

$$(2) \quad u = u(y)$$

$$(2) \quad v = v(y)$$

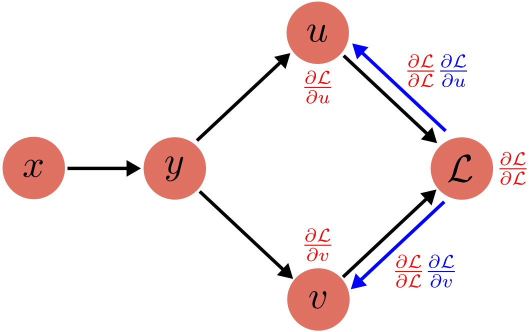
$$(3) \quad \mathcal{L} = \mathcal{L}(u, v)$$

Backward Pass:

$$(3) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u}$$

$$(3) \quad \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v}$$

Loss: $\mathcal{L}(\, u(y(x)),\, v(y(x))\,)$



Backpropagation: Fan-Out > 1

Forward Pass:

$$(1) \quad y = y(x)$$

$$(2) \quad u = u(y)$$

$$(2) \quad v = v(y)$$

$$(3) \quad \mathcal{L} = \mathcal{L}(u, v)$$

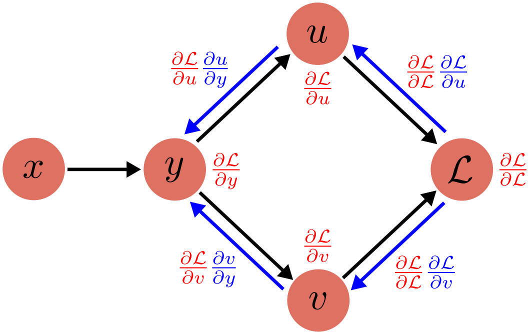
Backward Pass:

$$(3) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u}$$

$$(3) \quad \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v}$$

$$(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial y}$$

Loss: $\mathcal{L}(\, u(y(x)),\, v(y(x))\,)$



Backpropagation: Fan-Out > 1

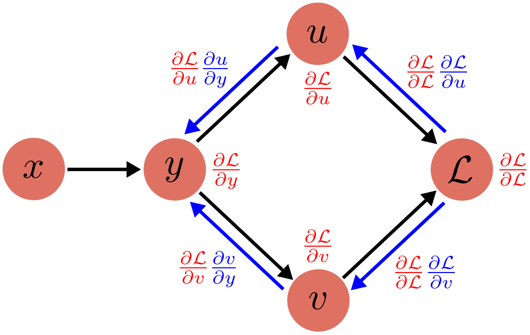
Forward Pass:

$$
\begin{aligned}
(1)\quad y &= y(x)\\
(2)\quad u &= u(y)\\
(2)\quad v &= v(y)\\
(3)\quad \mathcal{L} &= \mathcal{L}(u, v)
\end{aligned}
$$

Backward Pass:

$$
\begin{aligned}
(3)\quad \frac{\partial \mathcal{L}}{\partial u} &= \frac{\partial \mathcal{L}}{\partial \mathcal{L}}\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u}\\
(3)\quad \frac{\partial \mathcal{L}}{\partial v} &= \frac{\partial \mathcal{L}}{\partial \mathcal{L}}\frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v}\\
(2)\quad \frac{\partial \mathcal{L}}{\partial y} &= \frac{\partial \mathcal{L}}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial \mathcal{L}}{\partial v}\frac{\partial v}{\partial y}
\end{aligned}
$$

Loss: $\mathcal{L}(\,u(y(x)),\, v(y(x))\,)$



$$\frac{\mathrm{d}}{\mathrm{d}y}\mathcal{L}(u(y), v(y)) = ?$$

Backpropagation: Fan-Out > 1

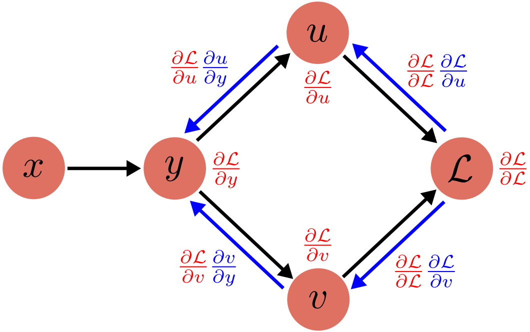
Forward Pass:

$$\begin{aligned} &(1) \quad y = y(x) \\ &(2) \quad u = u(y) \\ &(2) \quad v = v(y) \\ &(3) \quad \mathcal{L} = \mathcal{L}(u, v) \end{aligned}$$

Backward Pass:

$$\begin{aligned} &(3) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u} \\ &(3) \quad \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v} \\ &(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial y} \end{aligned}$$

Loss: $\mathcal{L}(\, u(y(x)),\, v(y(x))\,)$



$$\frac{\mathrm{d}}{\mathrm{d}y} \mathcal{L}(u(y), v(y)) = \frac{\partial \mathcal{L}}{\partial u} \frac{\mathrm{d}u}{\mathrm{d}y} + \frac{\partial \mathcal{L}}{\partial v} \frac{\mathrm{d}v}{\mathrm{d}y}$$

All incoming gradients must be **summed** up!

Backpropagation: Fan-Out > 1

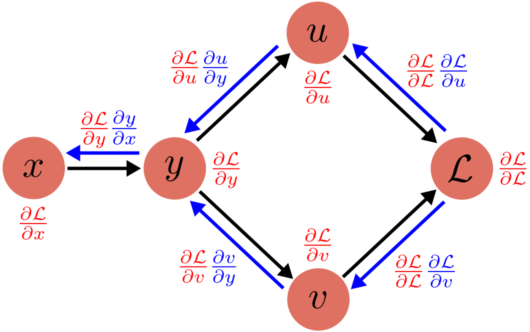
Forward Pass:

$$
\begin{aligned}
&(1) \quad y = y(x) \\
&(2) \quad u = u(y) \\
&(2) \quad v = v(y) \\
&(3) \quad \mathcal{L} = \mathcal{L}(u, v)
\end{aligned}
$$

Backward Pass:

$$
\begin{aligned}
&(3) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u} \\
&(3) \quad \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v} \\
&(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial y} \\
&(1) \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x}
\end{aligned}
$$

Loss: $\mathcal{L}(\, u(y(x)),\, v(y(x))\,)$



$$
\frac{\mathrm{d}}{\mathrm{d}y} \mathcal{L}(u(y), v(y)) = \frac{\partial \mathcal{L}}{\partial u} \frac{\mathrm{d}u}{\mathrm{d}y} + \frac{\partial \mathcal{L}}{\partial v} \frac{\mathrm{d}v}{\mathrm{d}y}
$$

All incoming gradients must be **summed** up!

Backpropagation: Fan-Out > 1

Forward Pass:

$$
\begin{aligned}
(1) \quad & y = y(x) \\
(2) \quad & u = u(y) \\
(2) \quad & v = v(y) \\
(3) \quad & \mathcal{L} = \mathcal{L}(u, v)
\end{aligned}
$$

Backward Pass:

$$
\begin{aligned}
(3) \quad & \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u} \\
(3) \quad & \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v} \\
(2) \quad & \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial y} \\
(1) \quad & \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x}
\end{aligned}
$$

Implementation: Each variable/node is an object and has attributes `x.value` and `x.grad`. Values are computed **forward** and gradients **backward**:

$$
\begin{aligned}
\texttt{x.value} &= \texttt{Input} \\
\texttt{y.value} &= y(\texttt{x.value}) \\
\texttt{u.value} &= u(\texttt{y.value}) \\
\texttt{v.value} &= v(\texttt{y.value}) \\
\texttt{L.value} &= \mathcal{L}(\texttt{u.value}, \texttt{v.value})
\end{aligned}
$$

Backpropagation: Fan-Out > 1

Forward Pass:

$$
\begin{aligned}
&(1) \quad y = y(x) \\
&(2) \quad u = u(y) \\
&(2) \quad v = v(y) \\
&(3) \quad \mathcal{L} = \mathcal{L}(u, v)
\end{aligned}
$$

Backward Pass:

$$
\begin{aligned}
&(3) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial u} \\
&(3) \quad \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial v} \\
&(2) \quad \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial \mathcal{L}}{\partial v} \frac{\partial v}{\partial y} \\
&(1) \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x}
\end{aligned}
$$

Implementation: Each variable/node is an object and has attributes `x.value` and `x.grad`. Values are computed **forward** and gradients **backward:**

$$\texttt{x.grad} = \texttt{y.grad} = \texttt{u.grad} = \texttt{v.grad} = 0$$

$$\texttt{L.grad} = 1$$

$$\texttt{u.grad} \mathrel{+}= \texttt{L.grad} * (\partial \mathcal{L}/\partial u)(\texttt{u.value}, \texttt{v.value})$$

$$\texttt{v.grad} \mathrel{+}= \texttt{L.grad} * (\partial \mathcal{L}/\partial v)(\texttt{u.value}, \texttt{v.value})$$

$$\texttt{y.grad} \mathrel{+}= \texttt{u.grad} * (\partial u/\partial y)(\texttt{y.value})$$

$$\texttt{y.grad} \mathrel{+}= \texttt{v.grad} * (\partial v/\partial y)(\texttt{y.value})$$

$$\texttt{x.grad} \mathrel{+}= \texttt{y.grad} * (\partial y/\partial x)(\texttt{x.value})$$

Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

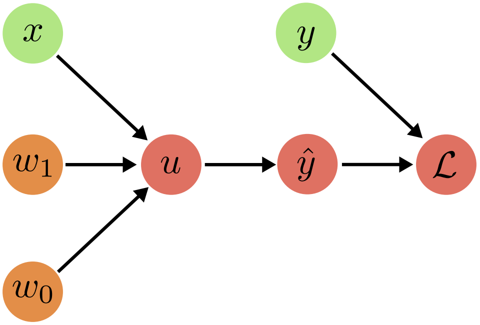
$$(1) \quad u = w_0 + w_1 x$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = \underbrace{-y \log \hat{y} - (1-y)\log(1-\hat{y})}_{\text{BCE}(\hat{y},y)}$$

Backward Pass:

Loss: $\mathcal{L} = \text{BCE}(\sigma(w_0 + w_1 x), y)$



Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

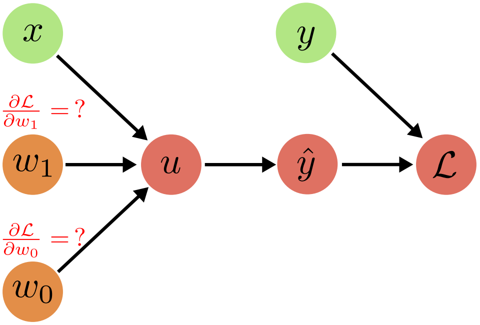
$$(1) \quad u = w_0 + w_1 x$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = \underbrace{-y \log \hat{y} - (1-y)\log(1-\hat{y})}_{\text{BCE}(\hat{y},y)}$$

Backward Pass:

Loss: $\mathcal{L} = \text{BCE}(\sigma(w_0 + w_1 x), y)$



Backpropagation: Logistic Regression with 1D Inputs

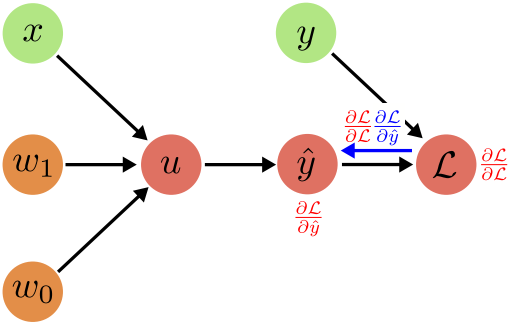
Forward Pass:

$$(1) \quad u = w_0 + w_1 x$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = \underbrace{-y \log \hat{y} - (1-y)\log(1-\hat{y})}_{\text{BCE}(\hat{y},y)}$$

Loss: $\mathcal{L} = \text{BCE}(\sigma(w_0 + w_1 x), y)$



Backward Pass:

$$(3) \quad \frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

Backpropagation: Logistic Regression with 1D Inputs

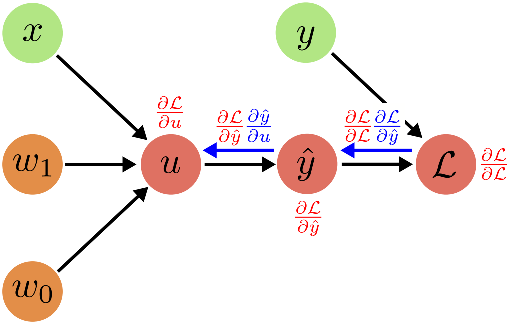
Forward Pass:

$$(1)\quad u = w_0 + w_1 x$$

$$(2)\quad \hat{y} = \sigma(u)$$

$$(3)\quad \mathcal{L} = \underbrace{-y\log\hat{y} - (1-y)\log(1-\hat{y})}_{\mathrm{BCE}(\hat{y},y)}$$

Loss: $\mathcal{L} = \mathrm{BCE}(\sigma(w_0 + w_1 x), y)$



Backward Pass:

$$(3)\quad \frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}}\frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$(2)\quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \hat{y}}\sigma(u)(1-\sigma(u))$$

Backpropagation: Logistic Regression with 1D Inputs

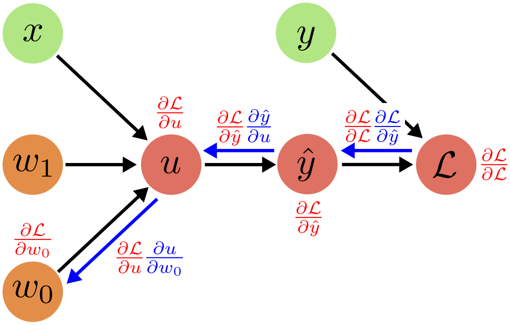
Forward Pass:

$$(1) \quad u = w_0 + w_1 x$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = \underbrace{-y \log \hat{y} - (1 - y) \log(1 - \hat{y})}_{\text{BCE}(\hat{y}, y)}$$

Loss: $\mathcal{L} = \text{BCE}(\sigma(w_0 + w_1 x), y)$



Backward Pass:

$$(3) \quad \frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

$$(2) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \sigma(u)(1 - \sigma(u))$$

$$(1) \quad \frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial \mathcal{L}}{\partial u} \frac{\partial u}{\partial w_0} = \frac{\partial \mathcal{L}}{\partial u}$$

Backpropagation: Logistic Regression with 1D Inputs

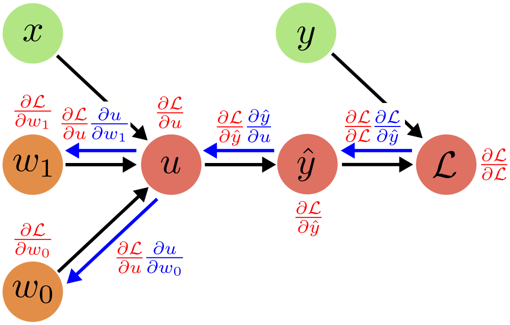
Forward Pass:

$$(1) \quad u = w_0 + w_1 x$$

$$(2) \quad \hat{y} = \sigma(u)$$

$$(3) \quad \mathcal{L} = \underbrace{-y \log \hat{y} - (1-y)\log(1-\hat{y})}_{\text{BCE}(\hat{y},y)}$$

Loss: $\mathcal{L} = \text{BCE}(\sigma(w_0 + w_1 x), y)$



Backward Pass:

$$(3) \quad \frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}}\frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$(2) \quad \frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \hat{y}}\sigma(u)(1-\sigma(u))$$

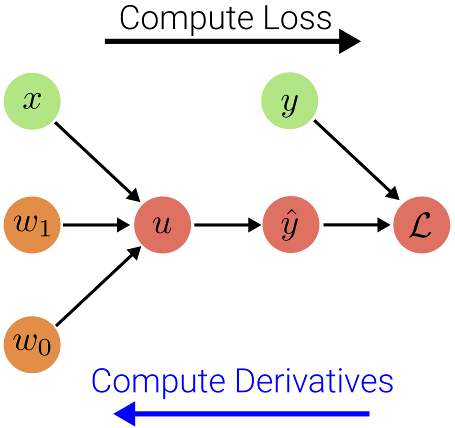
$$(1) \quad \frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial \mathcal{L}}{\partial u}\frac{\partial u}{\partial w_0} = \frac{\partial \mathcal{L}}{\partial u}$$

$$(1) \quad \frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial u}\frac{\partial u}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial u}x$$

Summary

- We can write mathematical expressions as a computation graph
- Values are efficiently computed forward, gradients backward
- Multiple incoming gradients are summed up (multivariate chain rule)
- Modularity: Each node must only “know” how to compute gradients wrt. its own arguments
- One fw/bw pass per data point:

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{y}, \mathbf{X}, \mathbf{w}) = \sum_{i=1}^{N} \underbrace{\nabla_{\mathbf{w}} \mathcal{L}(y_i, \mathbf{x}_i, \mathbf{w})}_{\text{Backpropagation}}$$



Disclaimer: So far we discussed backpropagation only for scalar values. In the next lecture, we will discuss backpropagation with arrays and tensors.

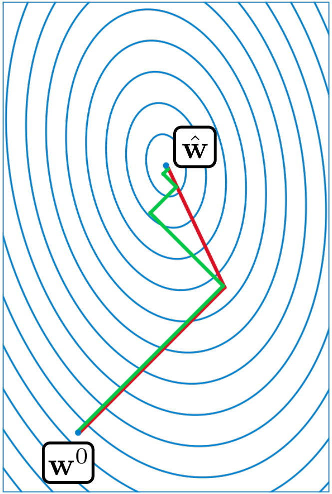
2.4
Educational Framework

Simple Training Recipe

Gradient Descent with Backpropagation:

- Pick step size η and tolerance ϵ
- Initialize $\mathbf{w}^0$
- Repeat until $\|\mathbf{v}\| < \epsilon$
  - For i=1..N
    - Forward Pass $\Rightarrow \mathcal{L}(\hat{y}_i = f_{\mathbf{w}}(\mathbf{x}_i), y_i)$
    - Backward Pass $\Rightarrow \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$
  - Gradient $\mathbf{v} = \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$
  - Update $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \mathbf{v}$

Let us now implement this in Python code ..



Educational Framework

- 150 lines of Python-NumPy code that implement a deep learning framework
- Allows us to understand the inner workings of a deep learning framework in depth
- Variables are bound to objects
  - Parents: `x`, `y`
  - Values: `value`
  - Gradients: `grad`
- Nodes are implemented as classes:
  - `Input`
  - `Parameter`
  - `CompNode`

David McAllester
TTI Chicago

Educational Framework

Computation Graph:

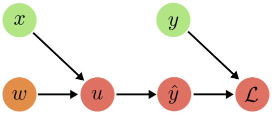
- Input nodes
- Parameter nodes
- Compute nodes

Remark: Specific compute node classes (e.g., `Sigmoid`) inherit from the abstract base class `CompNode`.

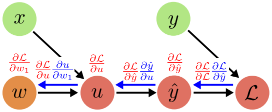
```
class Input:
    def __init__(self):
        pass

    def addgrad(self, delta):
        pass

class Parameter:
    def __init__(self,value):
        self.value = DT(value)
        Parameters.append(self)

    def addgrad(self,delta):
        self.grad += np.sum(delta, axis = 0)

    def UpdateParameters(self):
        self.value -= learning_rate*self.grad

class CompNode:
    def addgrad(self, delta):
        self.grad += delta
```

Educational Framework

Forward Pass:



Backward Pass:



Parameter Update:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$$

```
def Forward():
    for c in CompNodes: c.forward()

def Backward(loss):
    for c in CompNodes + Parameters:
        c.grad = np.zeros(c.value.shape, dtype = DT)
        loss.grad = np.ones(loss.value.shape)/len(loss.value)
    for c in CompNodes[::-1]:
        c.backward();

def UpdateParameters():
    for p in Parameters: p.UpdateParameters()
```

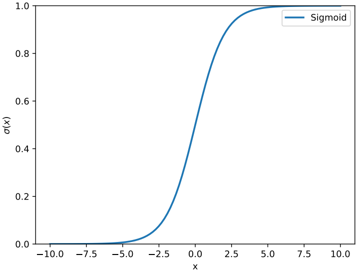
Remark: `Forward()` and `Backward()` compute the forward/backward pass over the entire dataset. Vectorization is more efficient than looping. Parallel computing can be exploited on GPUs.

Educational Framework

Computation Node Sigmoid:

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

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$




```
class Sigmoid(CompNode):
    def __init__(self,x):
        CompNodes.append(self)
        self.x = x

    def forward(self):
        bounded = np.maximum(-10,np.minimum(10,self.x.value))
        self.value = 1 / (1 + np.exp(-bounded))

    def backward(self):
        self.x.addgrad(self.grad * self.value * (1-self.value))
```

Remark: In the backward pass, the gradient is sent to the parent node `self.x`.

Educational Framework

Execution Example:

- Load data $\mathbf{X}$ and labels $\mathbf{y}$
- Initialize parameters $\mathbf{w}^0$
- Define computation graph
- For all iterations do
  - Forward Pass
    $\mathcal{L}(\hat{y}_i = f_{\mathbf{w}}(\mathbf{x}_i), y_i)$
  - Backward Pass
    $\nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$
  - Gradient Update
    $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \sum_{i=1}^N \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i)$

```python
import edf

# data loading
edf.clear_compgraph()
x = edf.Input()
y = edf.Input()
x.value = Load(data)
y.value = Load(labels)

# initialization of parameters
params_1 = edf.AffineParams(nInputs,nHiddens)
params_2 = edf.AffineParams(nHiddens,nLabels)

# definition of computation graph
h = edf.Sigmoid(edf.Affine(params_1, x))
p = edf.Softmax(edf.Affine(params_2, h))
L = edf.CrossEntropyLoss(p, y)

# gradient descent
for i in range(iterations):
    edf.Forward()
    edf.Backward(L)
    edf.UpdateParameters()
```