Computer Vision with Deep Learning

Computation Graphs

Fatma Güney

COMP411/511 - Fall 2024

PLAN

Logistic Regression

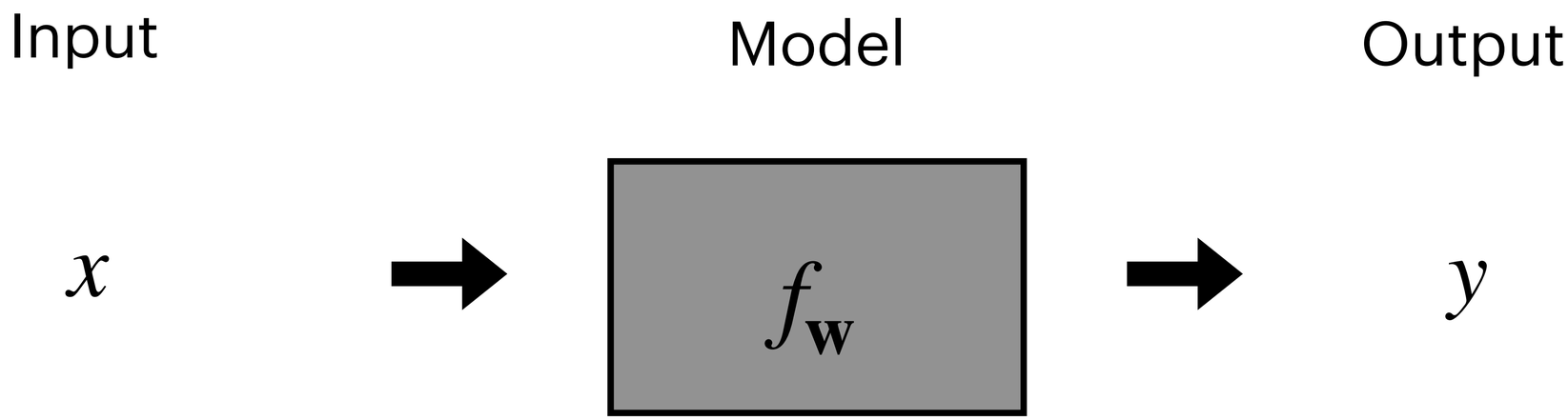
Computation Graphs

Backpropagation

Hands on

Quick Recap

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

$f_{\mathbf{w}}$

Output

143,52 €

- **Mapping:** $f_{\mathbf{w}} : \mathcal{R}^N \mapsto \mathcal{R}$

Classification

Input

Model

$f_{\mathbf{w}}$

Output

“Beach”

- **Mapping:** $f_{\mathbf{w}} : \mathscr{R}^{W \times H} \mapsto \{\text{"Beach", "No beach"}\}$

Logistic Regression

Logistic Regression

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

$$\hat{\mathbf{w}}_{ML} = \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log p_{model}(y_i \mid \mathbf{x}_i, \mathbf{w})$$

- Let's perform binary classification: $y_i \in \{0,1\}$
- How to choose $p_{model}(y \mid \mathbf{x}, \mathbf{w})$ for this problem?
- Answer: Bernoulli distribution

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

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

Logistic Regression

We assumed a Bernoulli distribution

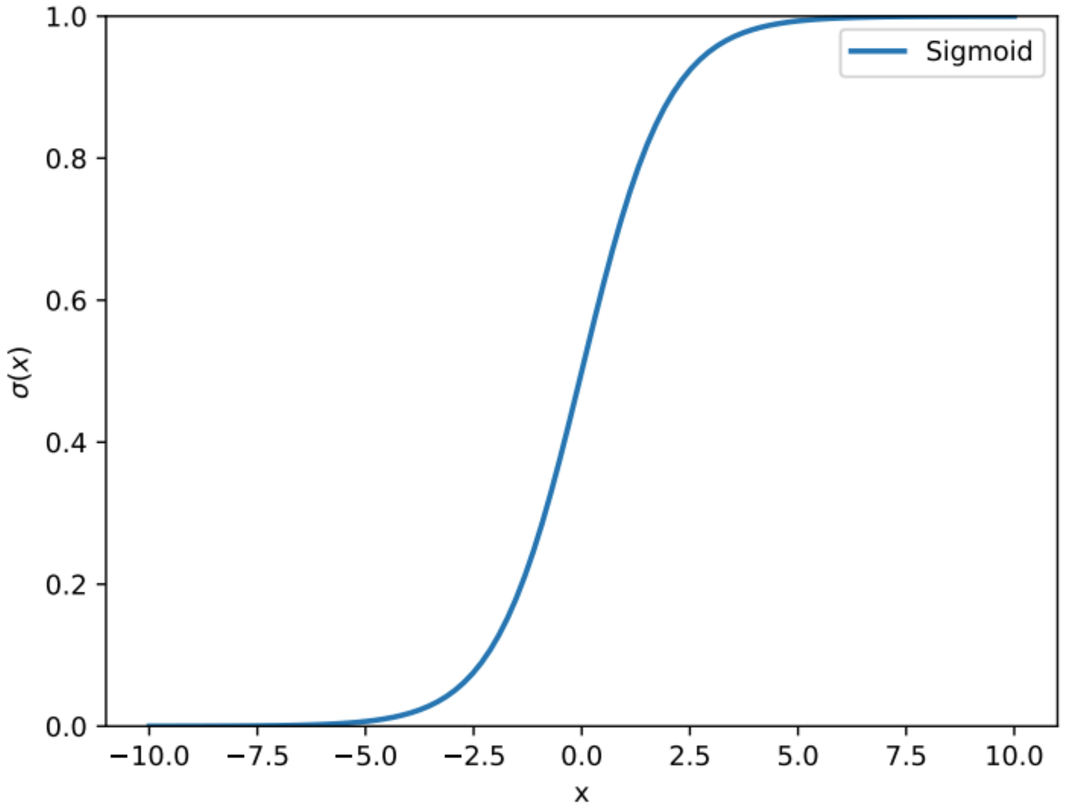
$$p_{model}(y \mid \mathbf{x}, \mathbf{w}) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$$

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

- How to choose $f_{\mathbf{w}}(\mathbf{x})$?
- Requirement: $f_{\mathbf{w}}(\mathbf{x}) \in [0,1]$
- Choose $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$

  where σ is the sigmoid function:

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



Logistic Regression

Putting it together:

$$\hat{\mathbf{w}}_{ML} = \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log p_{model}(y_i \mid \mathbf{x}_i, \mathbf{w})$$

$$= \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log \left[\hat{y}_i^{\,y_i}(1-\hat{y}_i)^{(1-y_i)}\right]$$

$$= \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \boxed{-y_i \log \hat{y}_i - (1-y_i)\log(1-\hat{y}_i)}$$

Binary Cross Entropy Loss

$\mathscr{L}(\hat{y}_i, y_i)$

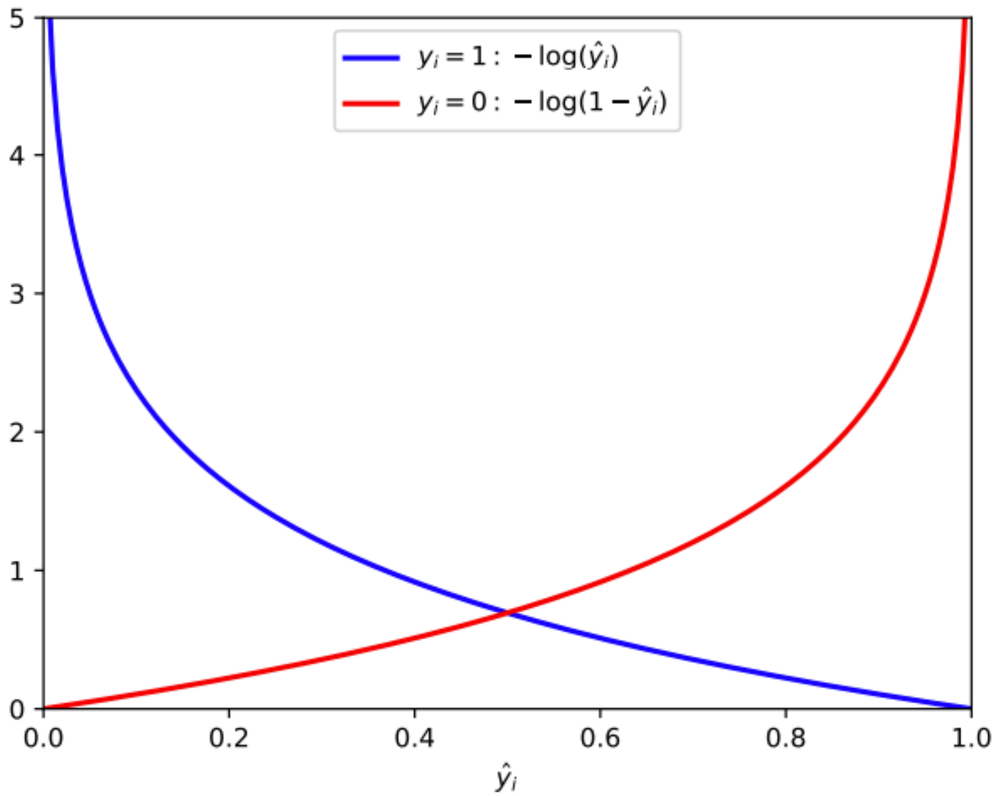
- In ML, we use the more general term “loss function” rather than “error function”
- Interpretation: We minimize the dissimilarity between the empirical data distribution p_{data} (defined by the training set) and the model distribution p_{model}

Logistic Regression

Binary Cross Entropy Loss:

$$\mathscr{L}(\hat{y}_i, y_i) = -\, y_i \log \hat{y}_i - (1 - y_i) \log (1 - \hat{y}_i)$$

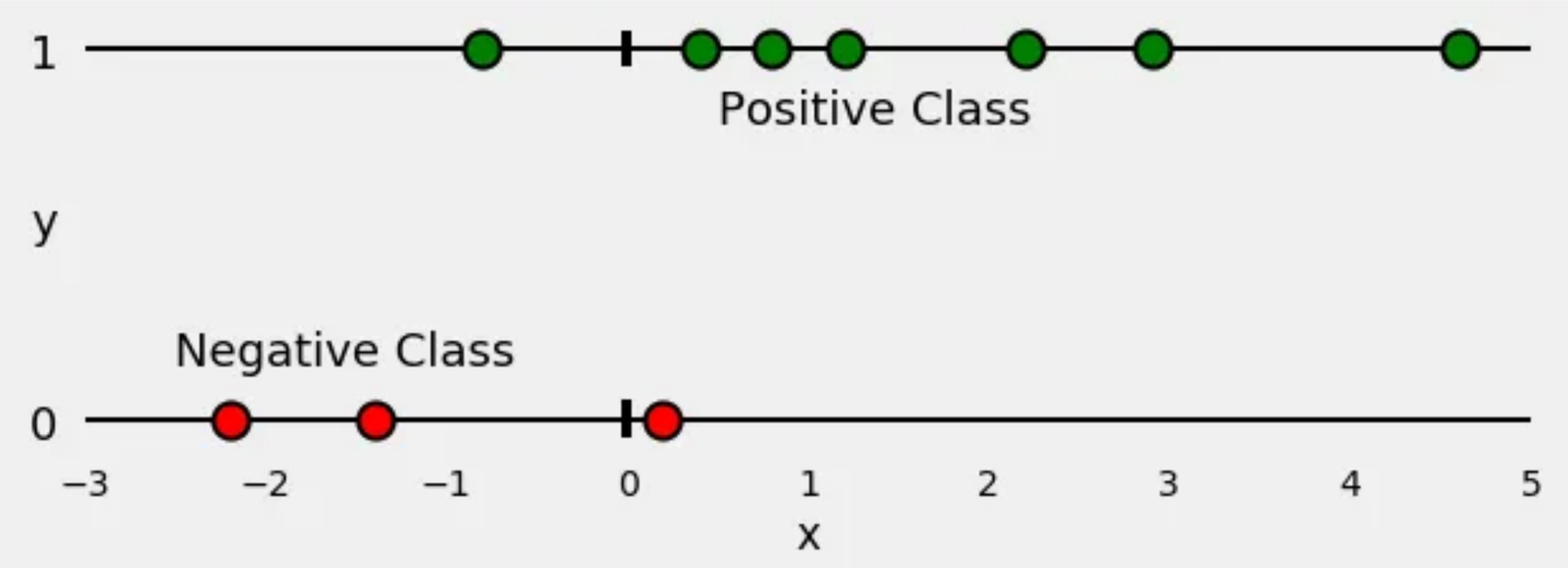
$$= \begin{cases} -\log \hat{y}_i & \text{if } y_i = 1 \\ -\log (1 - \hat{y}_i) & \text{if } y_i = 0 \end{cases}$$



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

Logistic Regression

A simple 1D example:

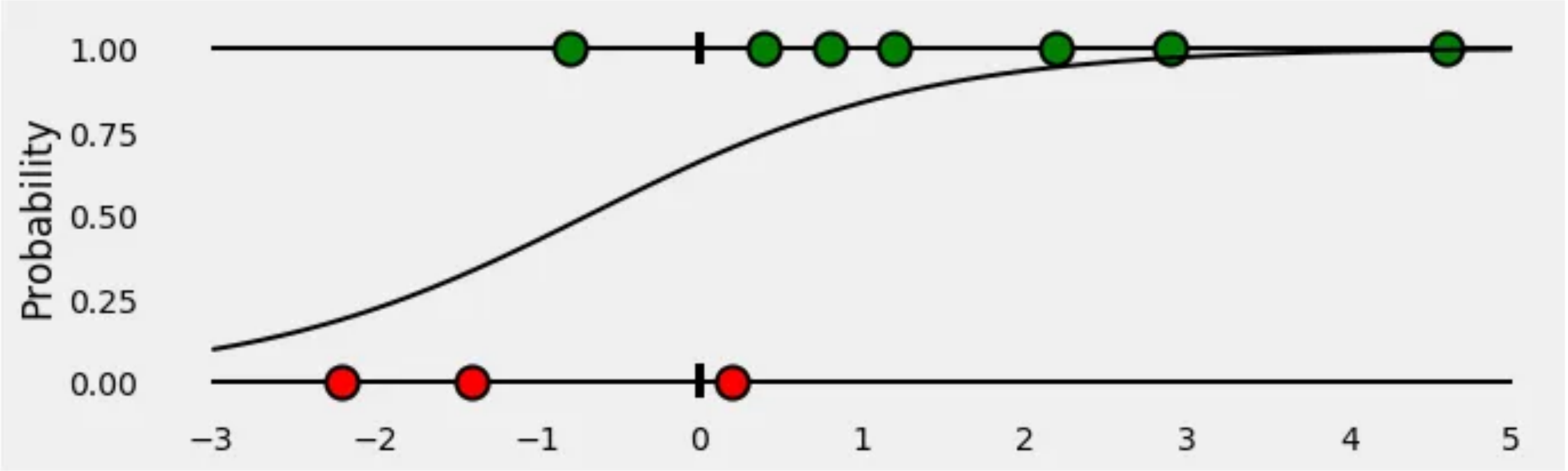


✦ Dataset $\mathcal{X}$ with positive ($y_i = 1$) and negative ($y_i = 0$) samples

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

Logistic Regression

A simple 1D example:

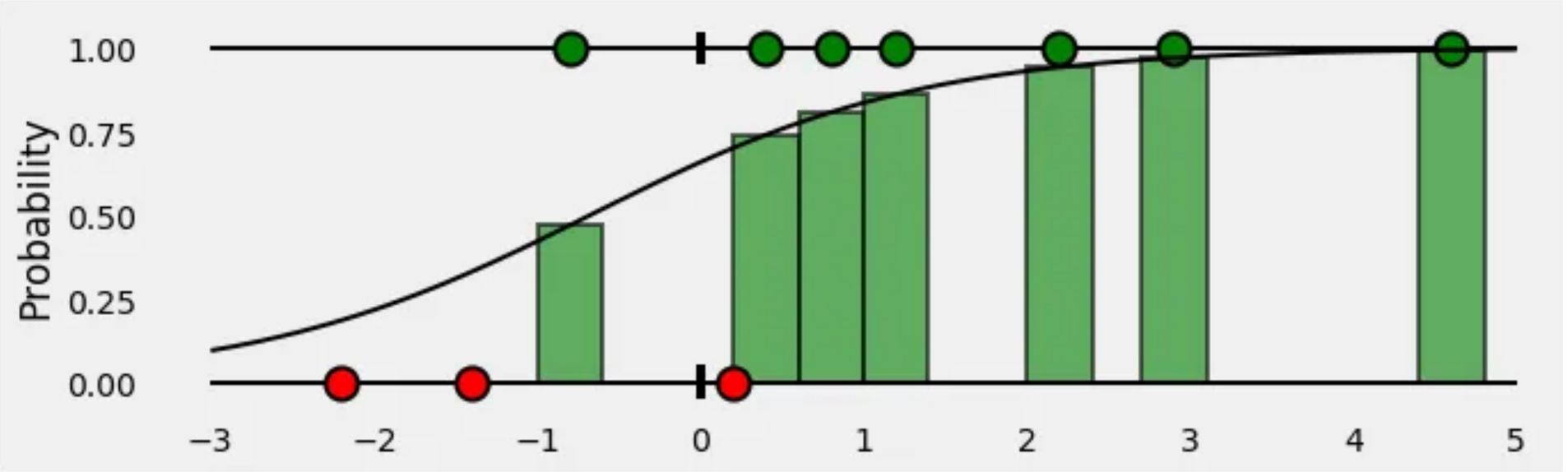


✦ Logistic regressor $f_{\mathbf{w}}(x) = \sigma(w_0 + w_1 x)$ fit to dataset $\mathcal{X}$

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

Logistic Regression

A simple 1D example:

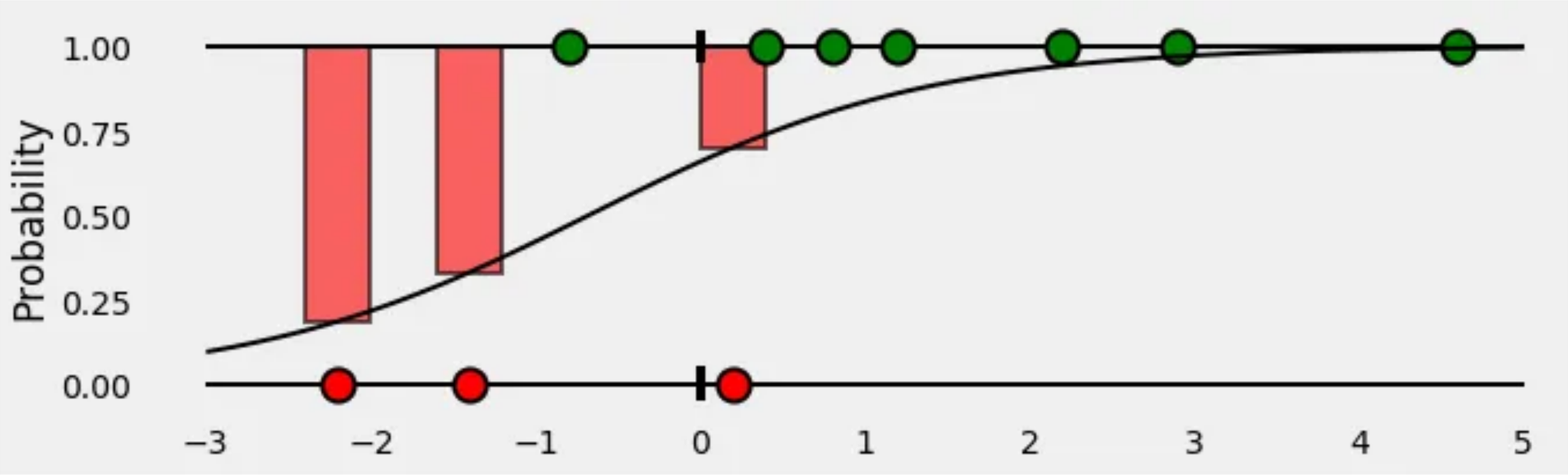


✦ Probabilities of classifier $f_{\mathbf{w}}(x_i)$ for positive samples $(y_i = 1)$

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

Logistic Regression

A simple 1D example:

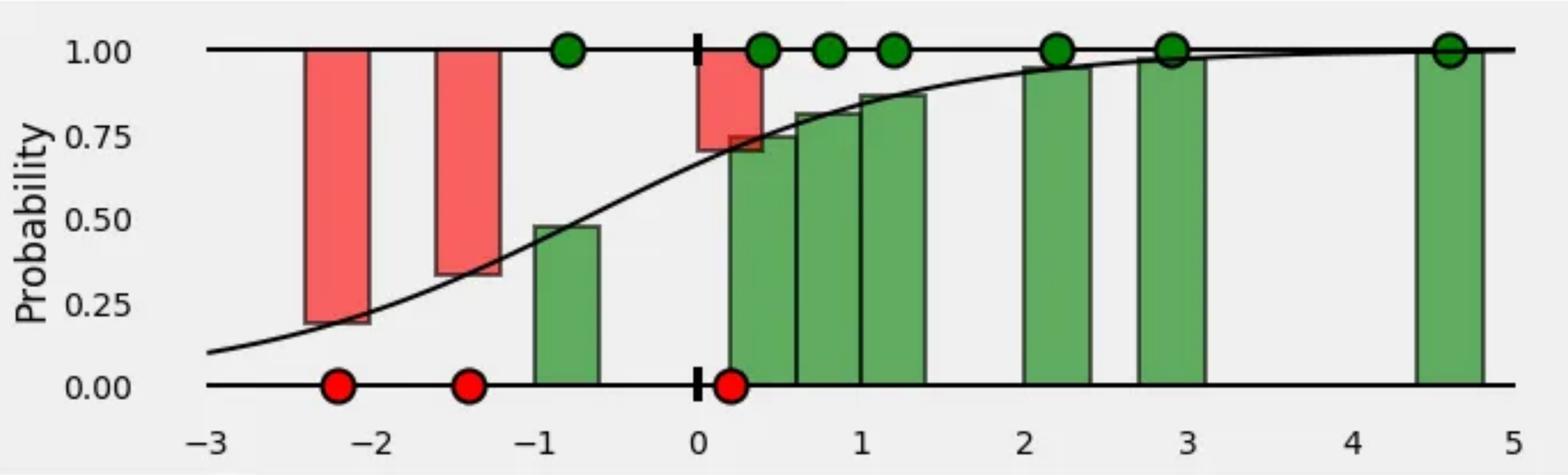


✦ Probabilities of classifier $f_{\mathbf{w}}(x_i)$ for negative samples $(y_i = 0)$

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

Logistic Regression

A simple 1D example:

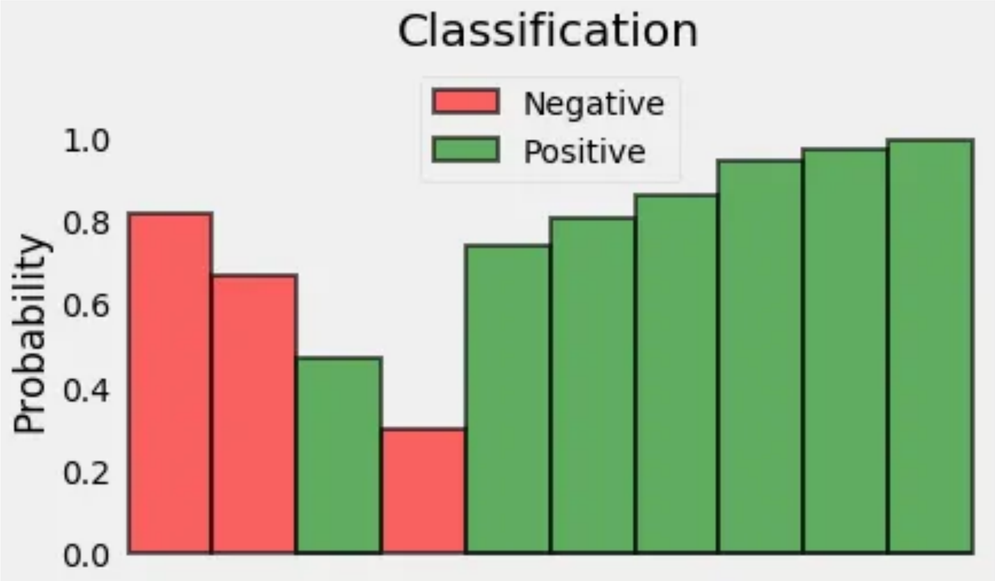


✦ Putting it together

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

Logistic Regression

A simple 1D example:

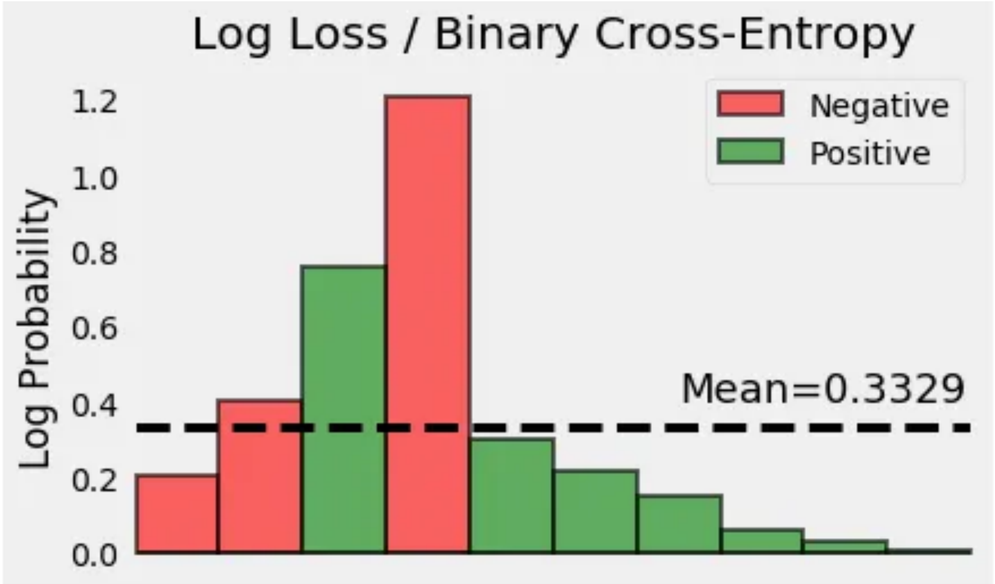


- Let’s get rid of the x axis

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: $-\log\left(f_{\mathbf{w}}(x_i)\right)$

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

Logistic Regression

Maximum Likelihood for Logistic Regression:

$$\hat{\mathbf{w}}_{ML} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \boxed{-y_i \log \hat{y}_i - (1 - y_i) \log (1 - \hat{y}_i)}$$

Binary Cross Entropy Loss $\mathscr{L}(\hat{y}_i, y_i)$

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

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

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

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

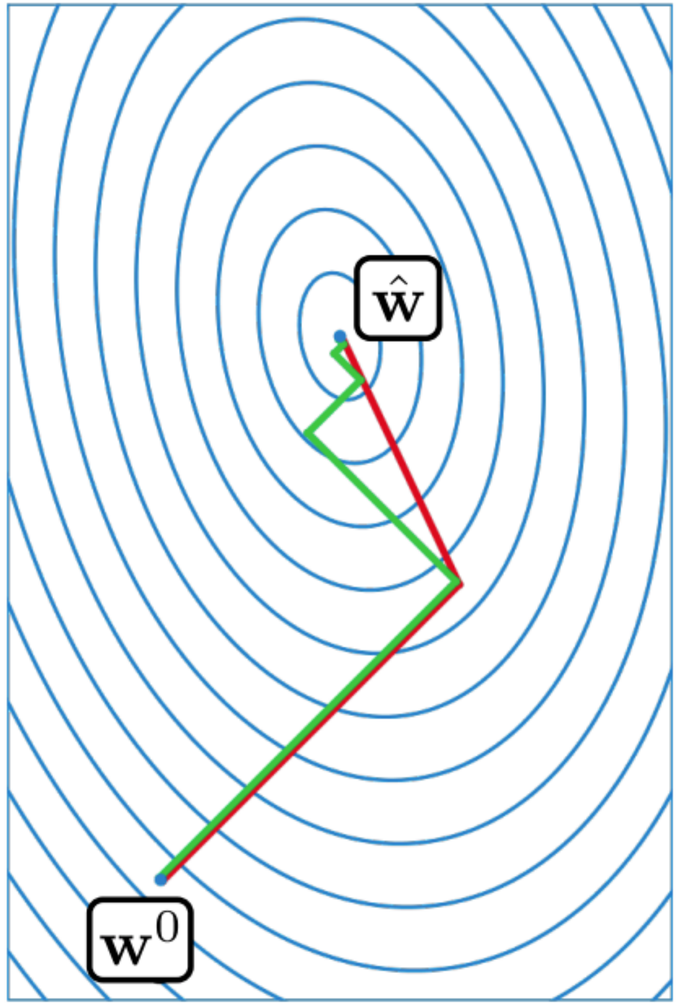
Logistic Regression

Gradient Descent:

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

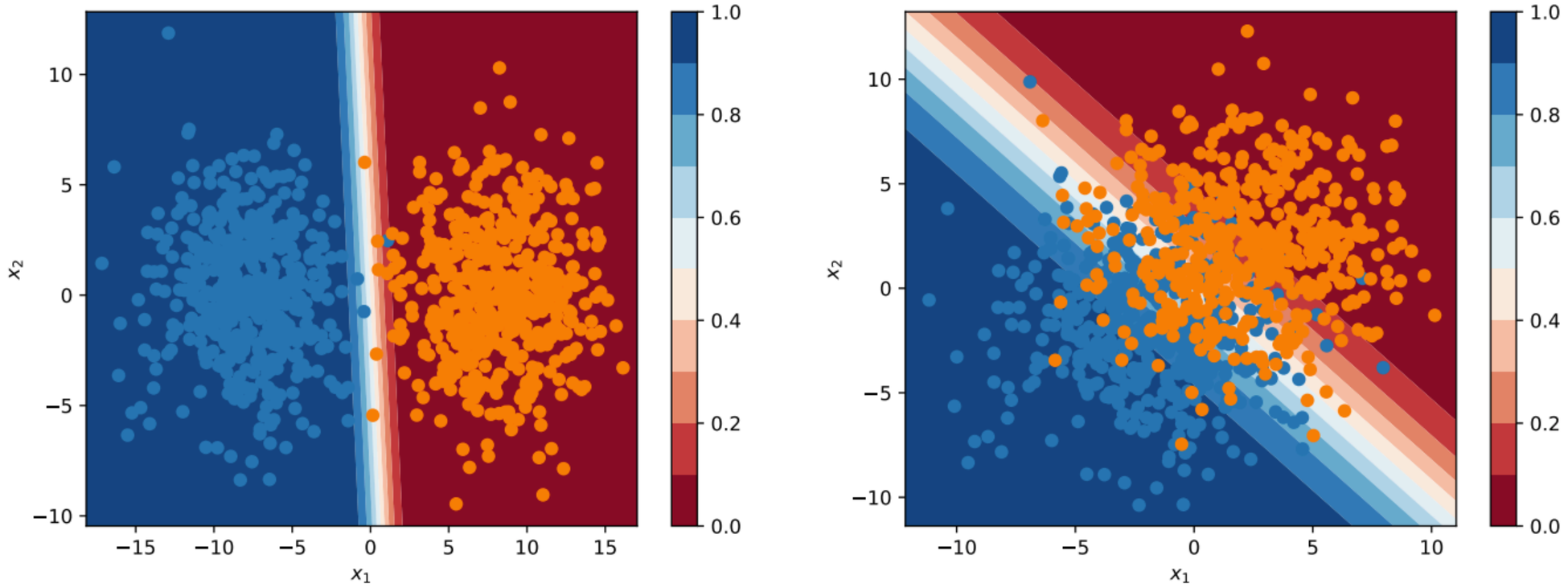
Variants:

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



Logistic Regression

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



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

Logistic Regression

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

$$\hat{\mathbf{w}}_{ML} = \arg\max_{\mathbf{w}} \boxed{\sum_{i=1}^{N} \log p_{model}(y_i \mid \mathbf{x}_i, \mathbf{w})}$$

Log-likelihood

$$= \arg\max_{\mathbf{w}} \mathcal{E}_{p_{data}} \left[\log p_{model}(y \mid \mathbf{x}, \mathbf{w}) \right]$$

$$= \arg\min_{\mathbf{w}} \boxed{- \mathcal{E}_{p_{data}} \left[\log p_{model}(y \mid \mathbf{x}, \mathbf{w}) \right]}$$

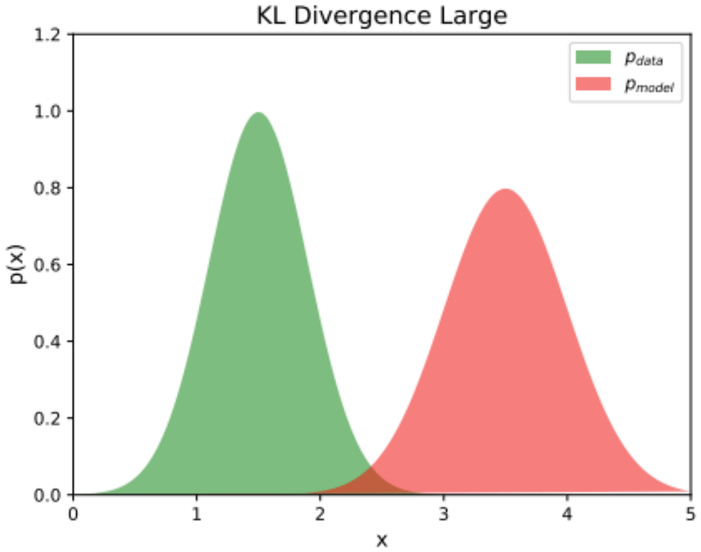
Cross Entropy

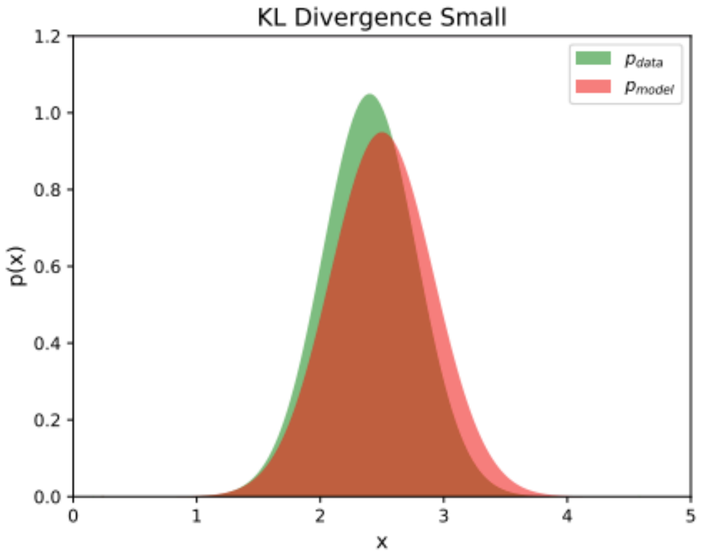
$H(p_{data}, p_{model})$

$$= \arg\min_{\mathbf{w}} \mathcal{E}_{p_{data}} \left[\log p_{data}(y \mid \mathbf{x}) - p_{model}(y \mid \mathbf{x}, \mathbf{w}) \right]$$

$$= \arg\min_{\mathbf{w}} \boxed{D_{KL}(p_{data} \| p_{model})}$$

KL Divergence





Computation Graphs

Logistic Regression

Maximum Likelihood for Logistic Regression:

$$\hat{\mathbf{w}}_{ML} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \boxed{-y_i \log \hat{y}_i - (1 - y_i) \log (1 - \hat{y}_i)}$$

Binary Cross Entropy Loss $\mathscr{L}(\hat{y}_i, y_i)$

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

- Minimization of a **non-linear objective** requires the calculation of gradients $\nabla_{\mathbf{w}}$
- Luckily, in the above case the gradient is simple: $\nabla_{\mathbf{w}} \mathscr{L}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)\mathbf{x_i}$
- 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.:

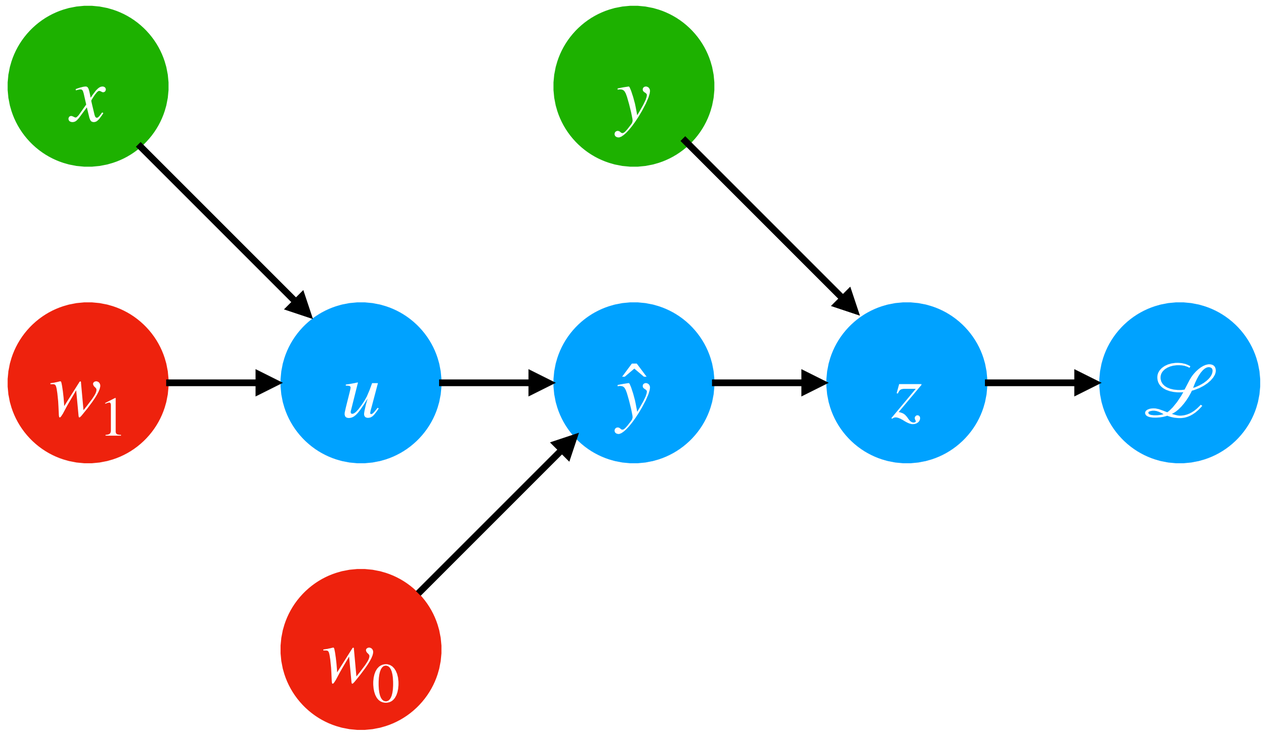
$$\mathscr{L} = -\log p_{model}(y \mid \mathbf{x}, \mathbf{w})$$

- Gradients $\nabla_{\mathbf{w}} \mathscr{L}$ can be computed using a **backward pass**
- Both forward and 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

(1) $u = w_1 x$

(2) $\hat{y} = w_0 + u$

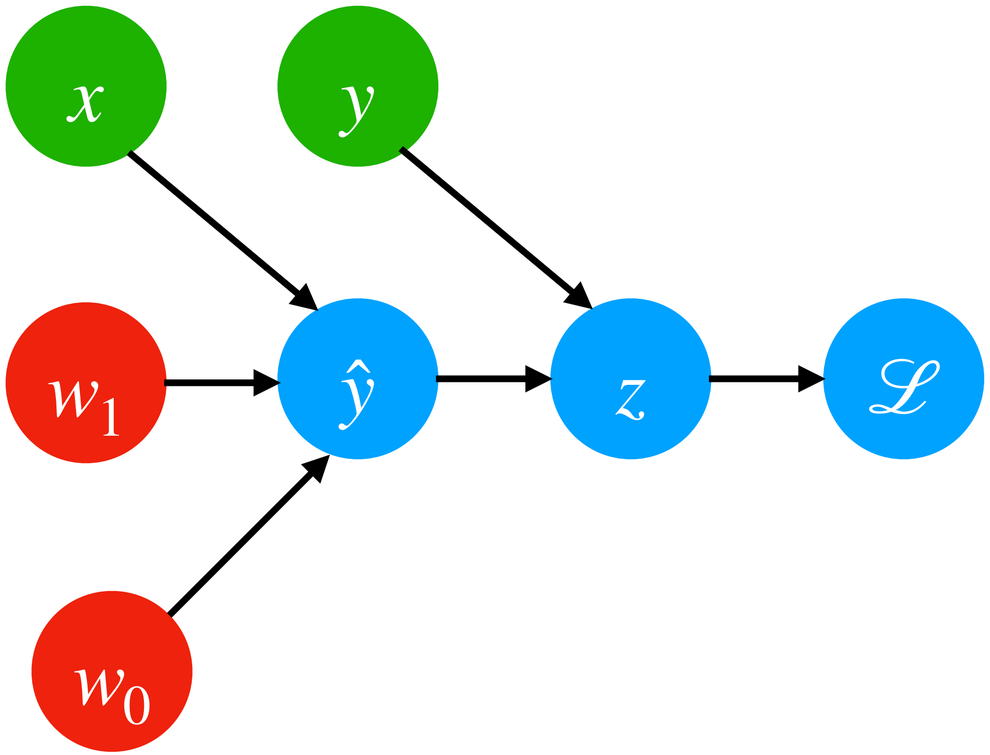
(3) $z = \hat{y} - y$

(4) $\mathscr{L} = z^2$

Computation Graphs

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

- Input nodes
- Parameter nodes
- Compute nodes



Example: Linear Regression

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

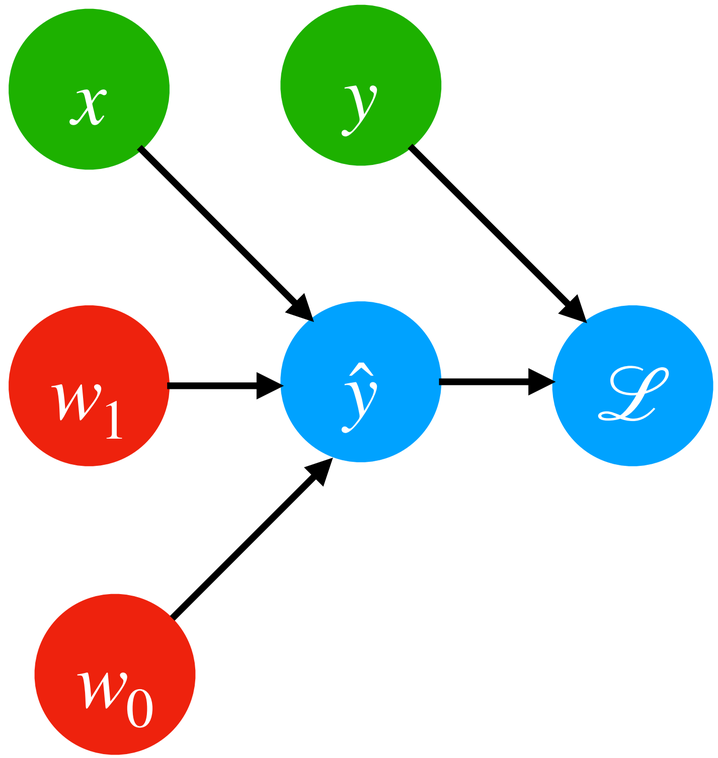
(2) $z = \hat{y} - y$

(3) $\mathscr{L} = z^2$

Computation Graphs

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

- Input nodes
- Parameter nodes
- Compute nodes



Example: Linear Regression

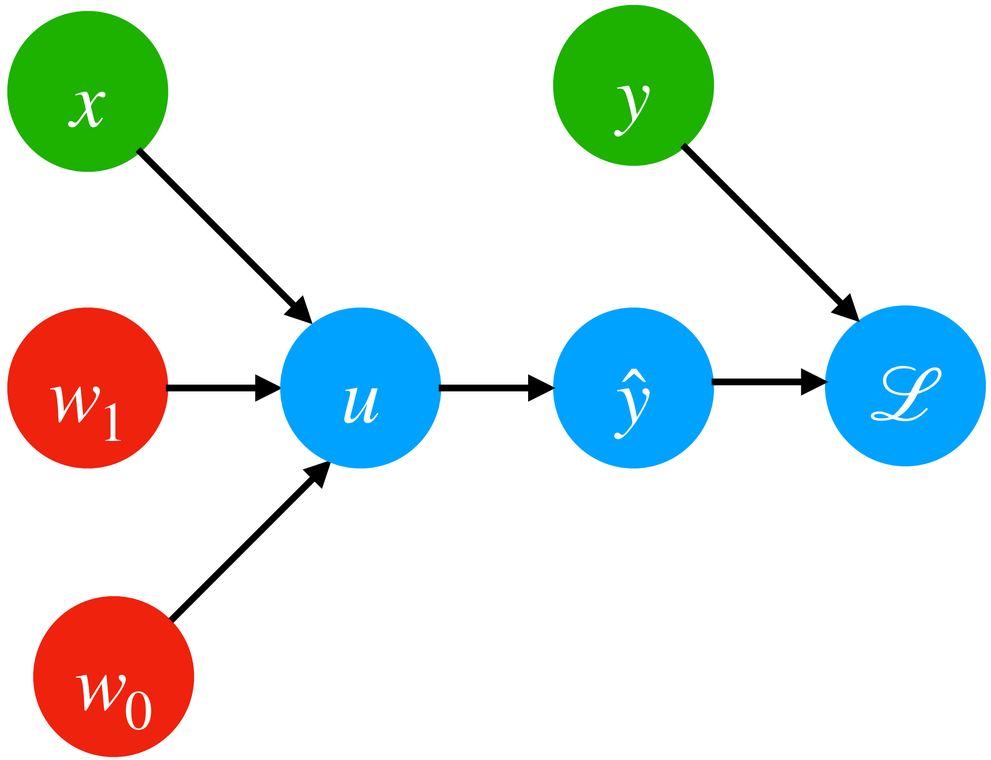
(1) $\hat{y} = w_0 + w_1 x$

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

Computation Graphs

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

- Input nodes
- Parameter nodes
- Compute nodes



Example: Logistic Regression

(1) $u = w_0 + w_1 x$

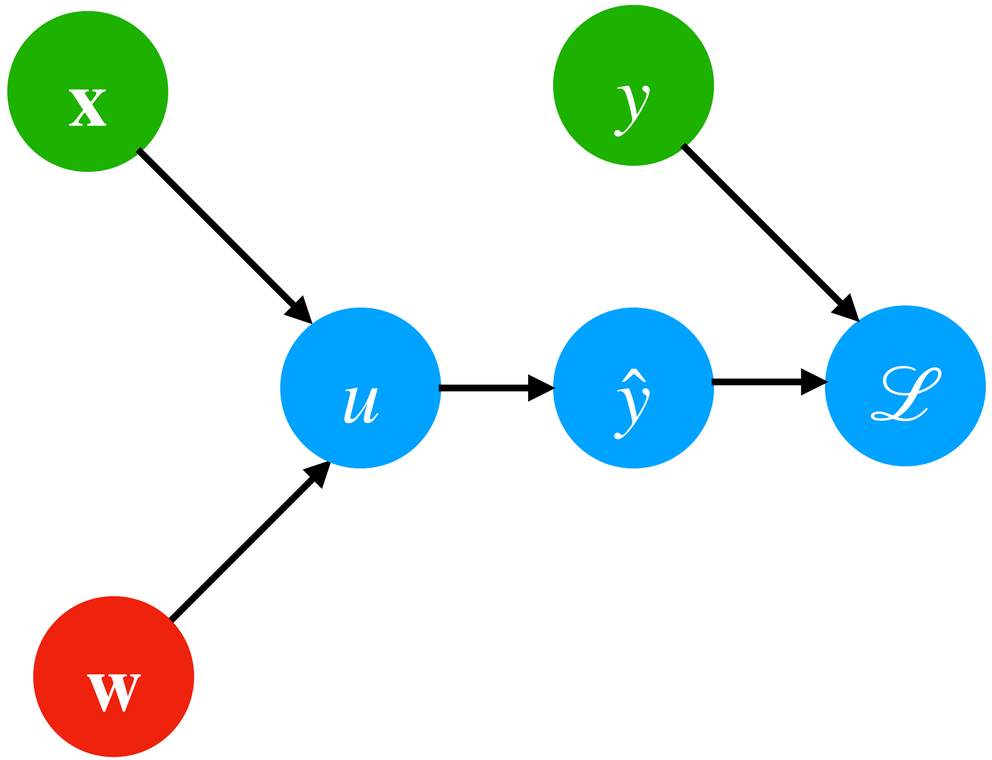
(2) $\hat{y} = \sigma(u)$

(3) $\mathscr{L} = -\,y \log \hat{y} - (1 - y) \log (1 - \hat{y})$

Computation Graphs

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

- Input nodes
- Parameter nodes
- Compute nodes



Example: Logistic Regression

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

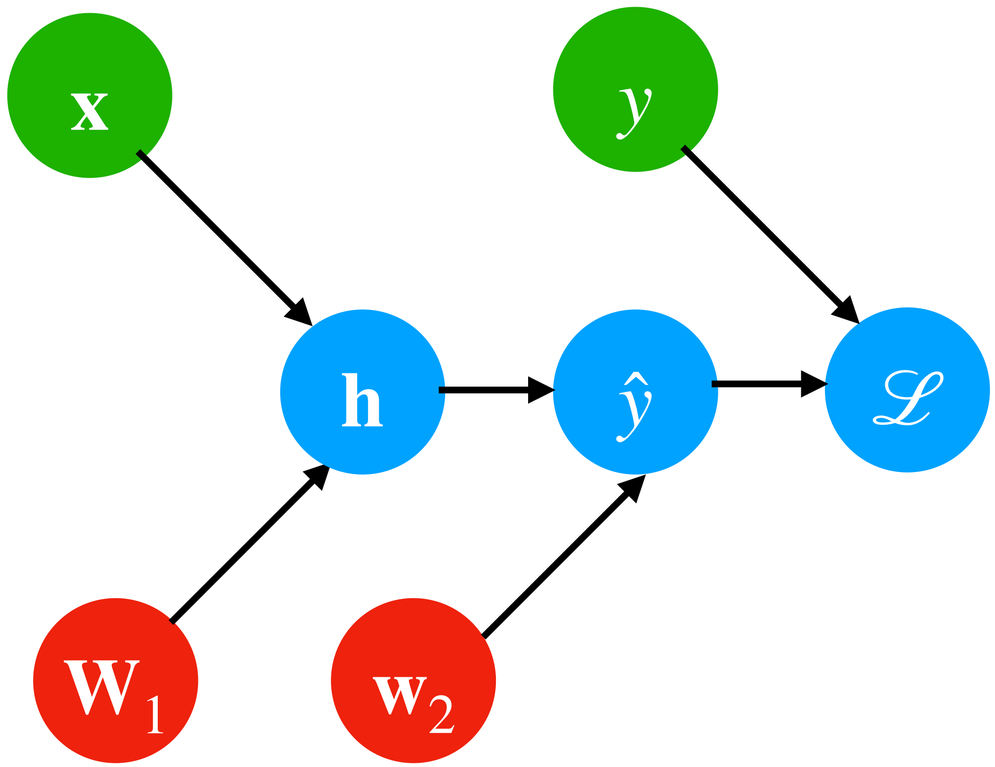
(2) $\hat{y} = \sigma(u)$

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

Computation Graphs

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

- Input nodes
- Parameter nodes
- Compute nodes



Example: Multi-Layer Perceptron

(1) $\mathbf{h} = \sigma\left(\mathbf{W}_1^T \mathbf{x}\right)$

(2) $\hat{y} = \sigma\left(\mathbf{w}_2^T \mathbf{h}\right)$

(3) $\mathscr{L} = -\, y \log \hat{y} - (1 - y) \log (1 - \hat{y})$

Backpropagation

Backpropagation

Goal: Find gradients of negative log likelihood

$$\nabla_{\mathbf{w}} \sum_{i=1}^{N} \boxed{-\log p_{model}(y_i \mid \mathbf{x}_i, \mathbf{w})}$$

$$\mathscr{L}(y_i, \mathbf{x}_i, \mathbf{w})$$

or more generally of a loss function

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

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

Chain Rule

Chain Rule:

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

Multivariate Chain Rule:

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

Backpropagation

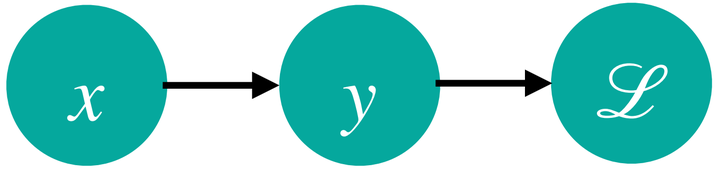
For now: no distinction between node types (input, parameter, compute)

Forward Pass:

(1) $y = x^2$

(2) $\mathscr{L} = 2y$

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



Backpropagation

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

Forward Pass:

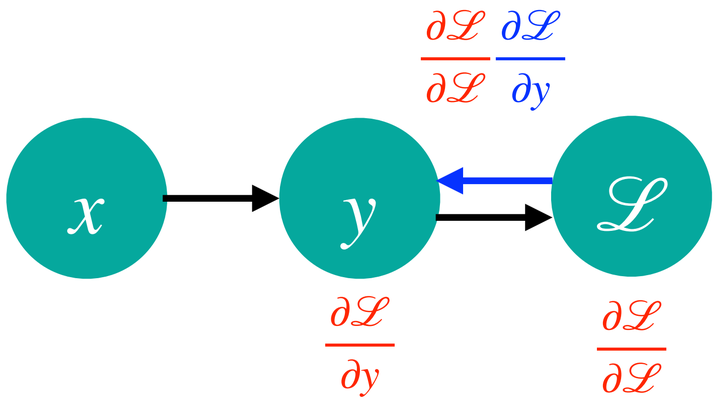
$$(1) \quad y = x^2$$

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

Backward Pass:

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

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



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

Backpropagation

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

Forward Pass:

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

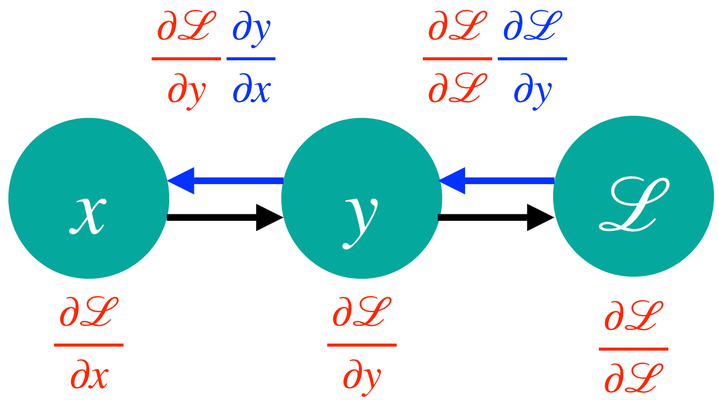
$$(2) \quad \mathscr{L} = 2y$$

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

Backward Pass:

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

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



✦ **Red**: back-propagated gradients

✦ **Blue**: local gradients

Backpropagation

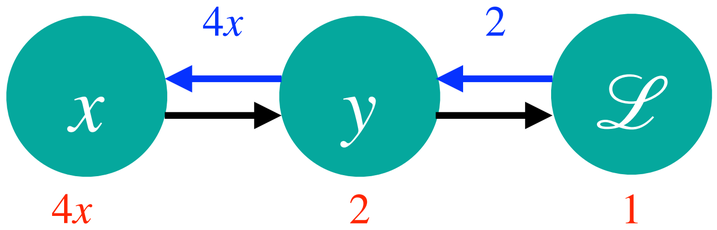
For now: no distinction between node types (input, parameter, compute)

Forward Pass:

(1) $y = x^2$

(2) $\mathscr{L} = 2y$

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



Backward Pass:

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

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

✦ **Red**: back-propagated gradients

✦ **Blue**: local gradients

Backpropagation: Abstract Case

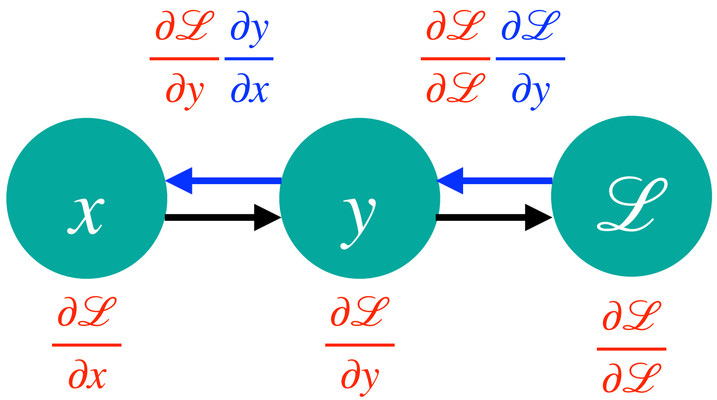
For now: no distinction between node types (input, parameter, compute)

Forward Pass:

Loss: $\mathscr{L}\left(y(x)\right)$

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

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



Backward Pass:

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

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

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

Backpropagation: Fan-Out > 1

Forward Pass:

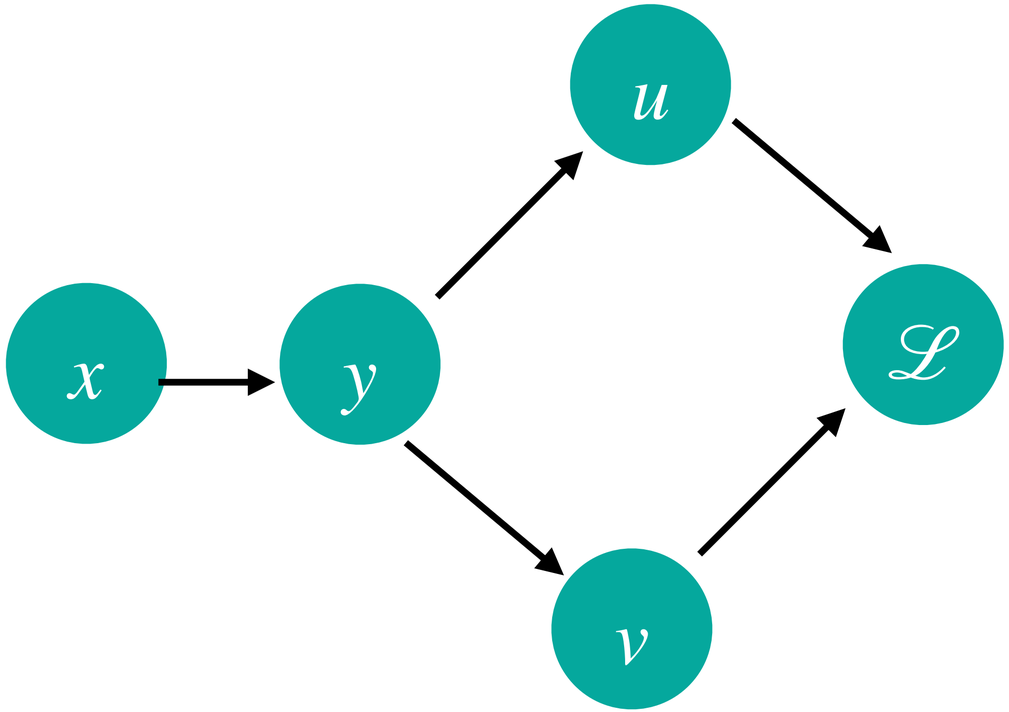
(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



Backpropagation: Fan-Out > 1

Forward Pass:

(1) $y = y(x)$

(2) $u = u(y)$

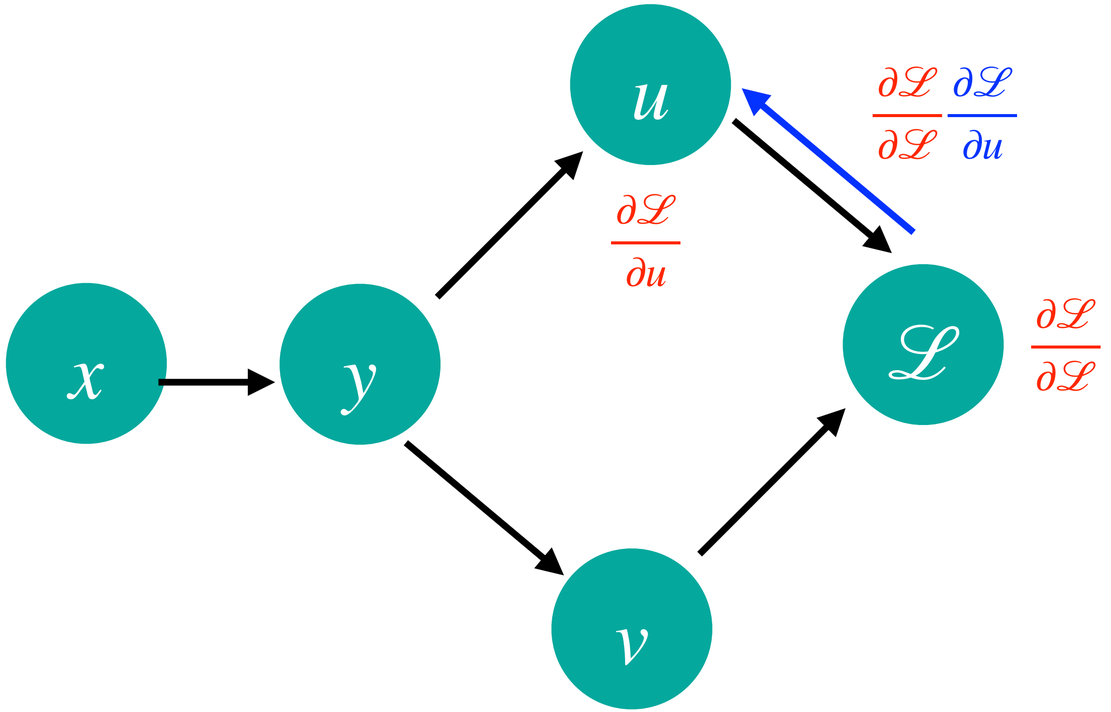
(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



Backpropagation: Fan-Out > 1

Forward Pass:

(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

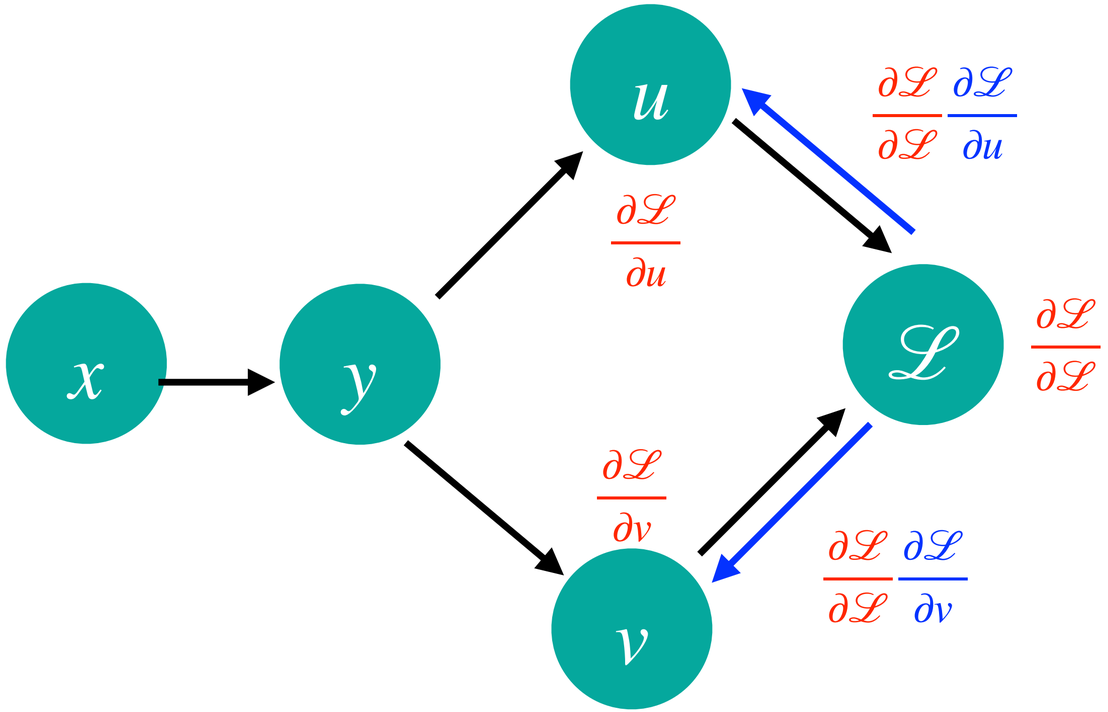
(4) $\mathscr{L} = \mathscr{L}(u, v)$

Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



Backpropagation: Fan-Out > 1

Forward Pass:

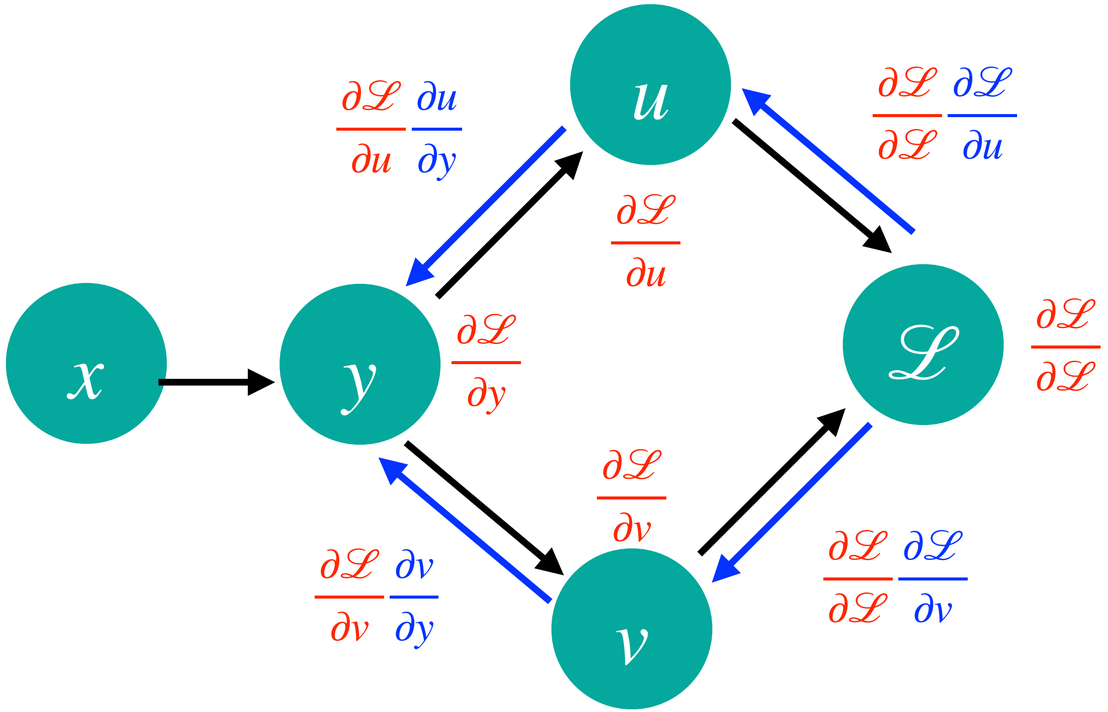
(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}}\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

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

Backpropagation: Fan-Out > 1

Forward Pass:

(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

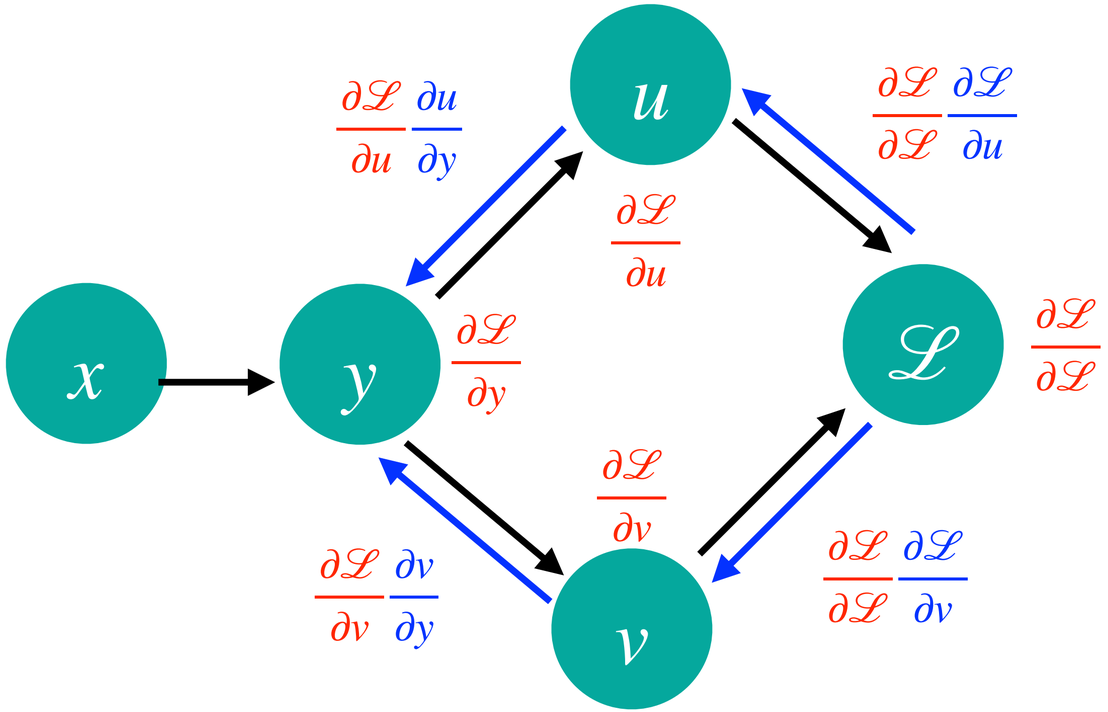
Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

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

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



$$\frac{\mathrm{d}}{\mathrm{dy}}\mathscr{L}\left(u(y), v(y)\right) = \ ?$$

Backpropagation: Fan-Out > 1

Forward Pass:

(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

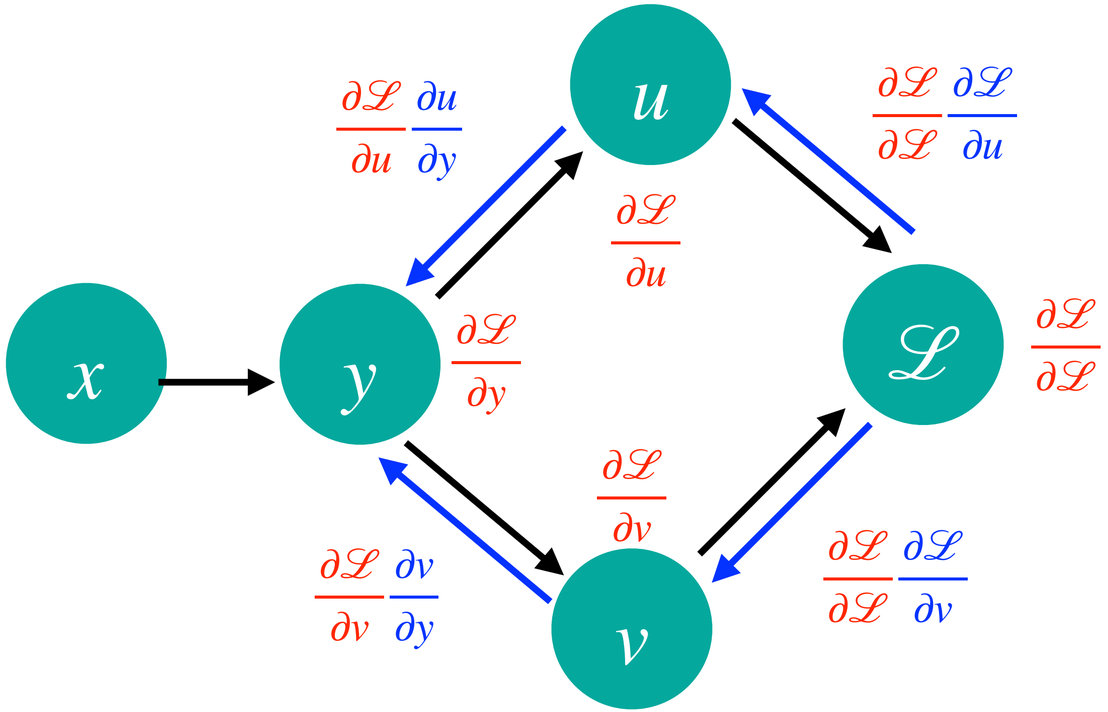
Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

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

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



$$\frac{\mathrm{d}}{\mathrm{dy}}\mathscr{L}\left(u(y), v(y)\right) = \ ?$$

Backpropagation: Fan-Out > 1

Forward Pass:

(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

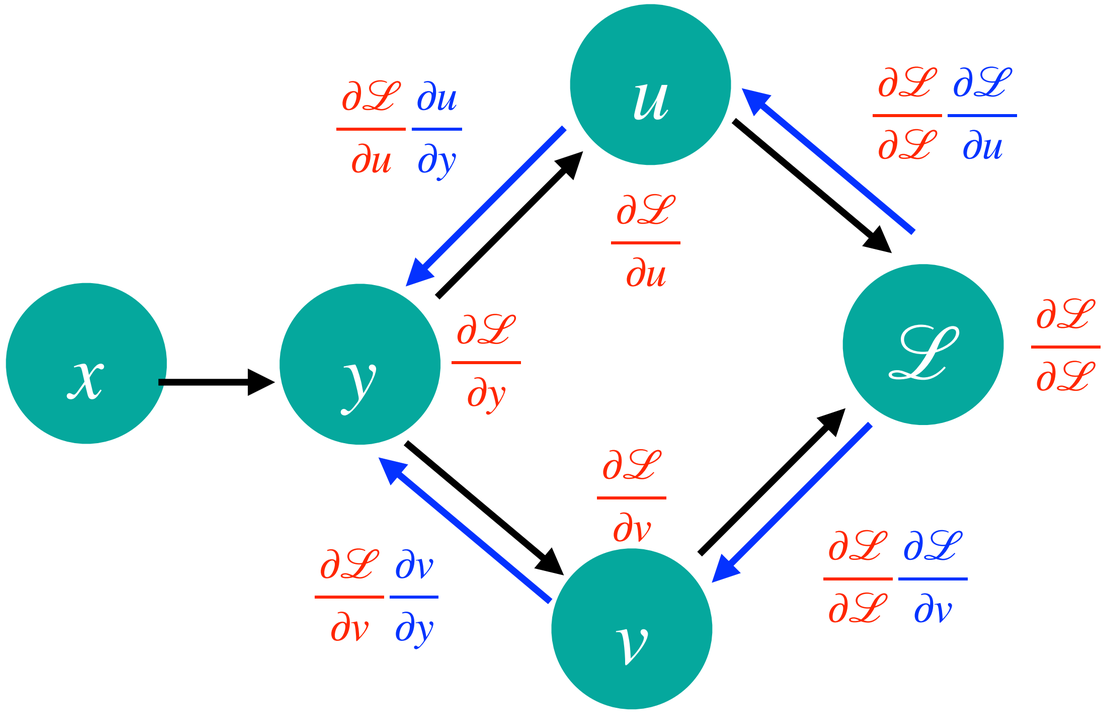
Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

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

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



$$\frac{\mathrm{d}}{\mathrm{dy}} \mathscr{L}\left(u(y), v(y)\right) = \frac{\partial \mathscr{L}}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial \mathscr{L}}{\partial v} \frac{\partial v}{\partial y}$$

All incoming gradients must be summed up!

Backpropagation: Fan-Out > 1

Forward Pass:

(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

Backward Pass:

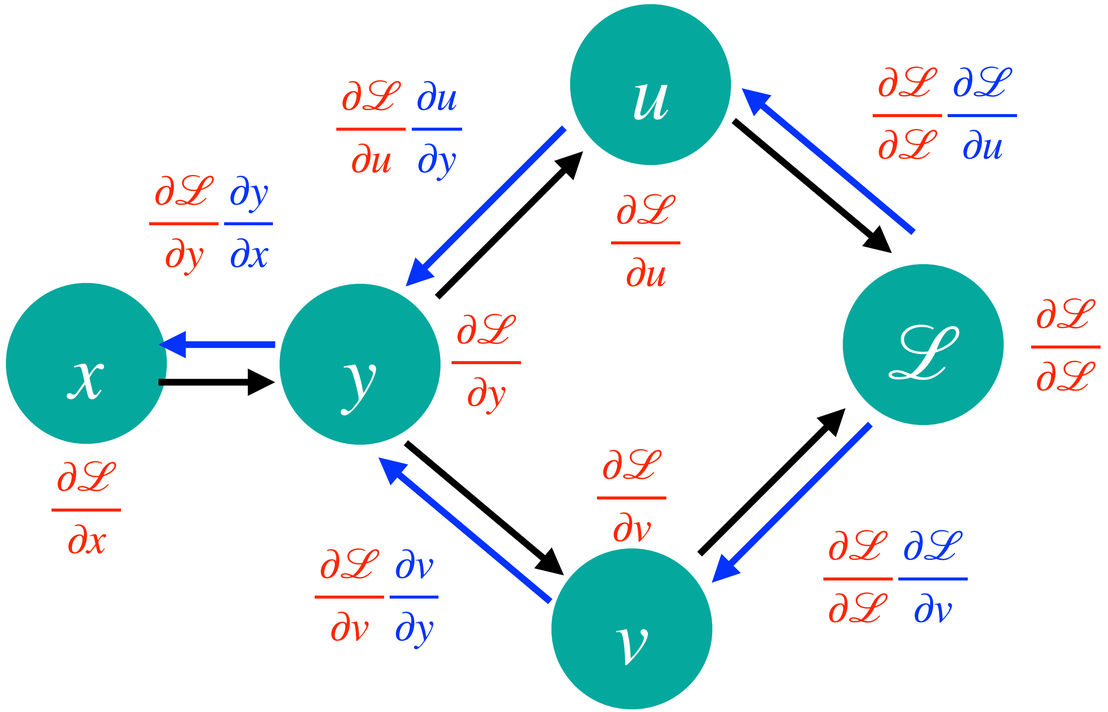
(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

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

(1) $$\frac{\partial \mathscr{L}}{\partial x} = \frac{\partial \mathscr{L}}{\partial y} \frac{\partial y}{\partial x}$$

Loss: $\mathscr{L}\left(u\left(y(x)\right), v\left(y(x)\right)\right)$



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

All incoming gradients must be summed up!

Backpropagation: Fan-Out > 1

Forward Pass:

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

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

$$(3) \quad v = v(y)$$

$$(4) \quad \mathscr{L} = \mathscr{L}(u, v)$$

Backward Pass:

$$(4) \quad \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

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

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

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

```
x.value = Input
y.value = y(x.value)
u.value = u(y.value)
v.value = v(y.value)
L.value = L(u.value, v.value)
```

Backpropagation: Fan-Out > 1

Forward Pass:

(1) $y = y(x)$

(2) $u = u(y)$

(3) $v = v(y)$

(4) $\mathscr{L} = \mathscr{L}(u, v)$

Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial u}$$

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

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

(1) $$\frac{\partial \mathscr{L}}{\partial x} = \frac{\partial \mathscr{L}}{\partial y} \frac{\partial y}{\partial x}$$

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

```
x.grad = y.grad = u.grad = v.grad = 0
L.grad = 1
u.grad += L.grad * (∂ℒ/∂u)(u.value, v.value)
v.grad += L.grad * (∂ℒ/∂v)(u.value, v.value)
y.grad += u.grad * (∂u/∂y)(y.value)
y.grad += v.grad * (∂v/∂y)(y.value)
x.grad += y.grad * (∂y/∂x)(x.value)
```

Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

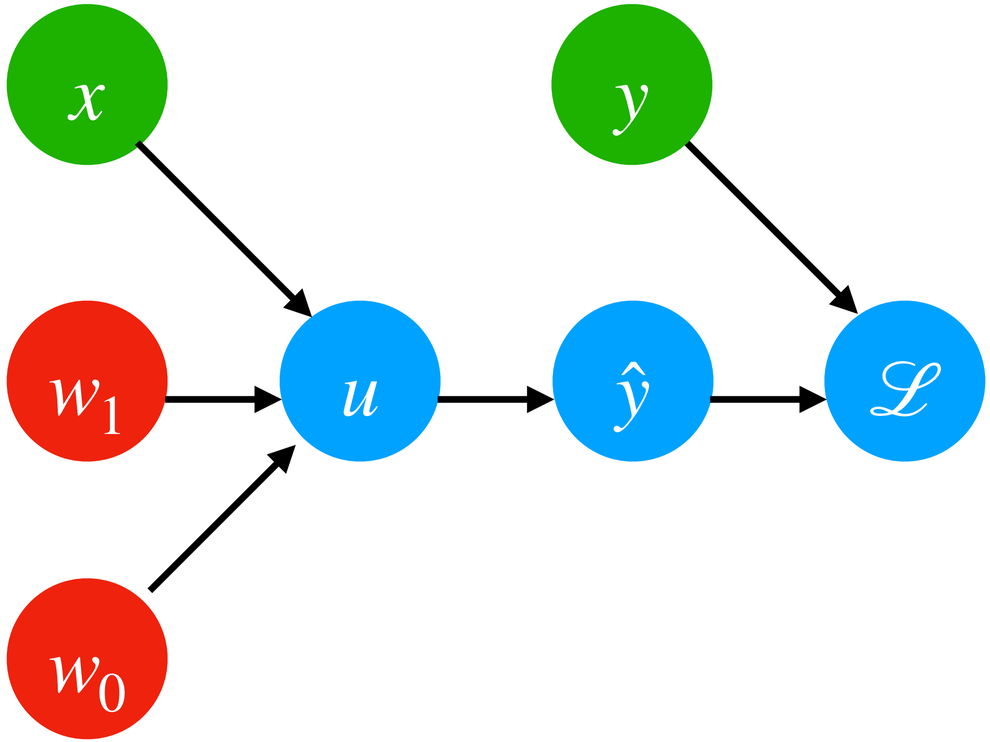
(1) $u = w_0 + w_1 x$

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

(3) $\mathscr{L} = \boxed{-y \log \hat{y} - (1-y) \log (1-\hat{y})}$ $\quad \text{BCE}(\hat{y}, y)$

Backward Pass:

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



Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

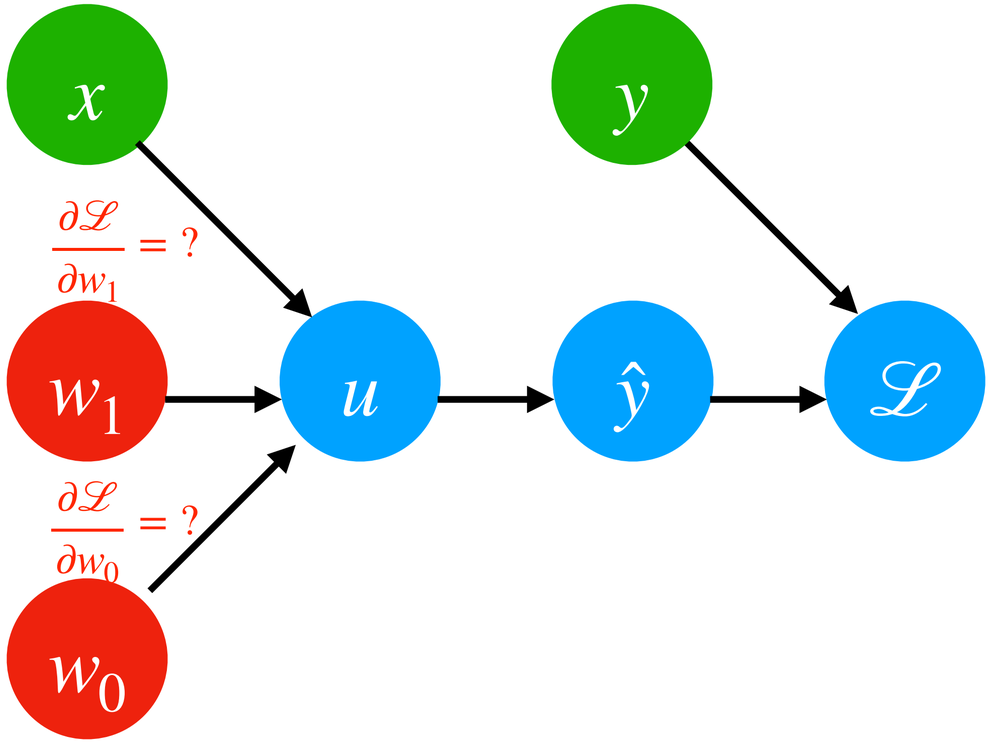
(1) $u = w_0 + w_1 x$

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

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

Backward Pass:

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



Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

(1) $u = w_0 + w_1 x$

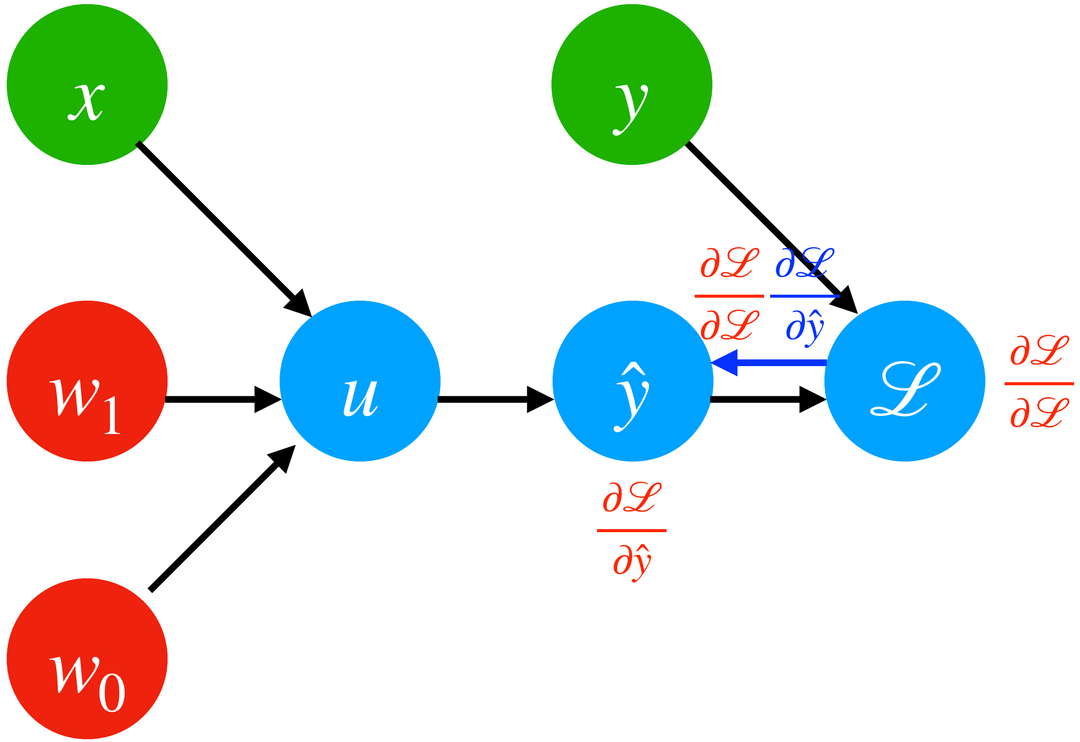
(2) $\hat{y} = \sigma(u)$

(3) $\mathscr{L} = \boxed{-y \log \hat{y} - (1-y) \log (1-\hat{y})}$ $\quad \text{BCE}(\hat{y}, y)$

Backward Pass:

(4) $\dfrac{\partial \mathscr{L}}{\partial \hat{y}} = \dfrac{\partial \mathscr{L}}{\partial \mathscr{L}} \dfrac{\partial \mathscr{L}}{\partial \hat{y}} = \dfrac{\hat{y} - y}{\hat{y}(1-\hat{y})}$

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



Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

(1) $u = w_0 + w_1 x$

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

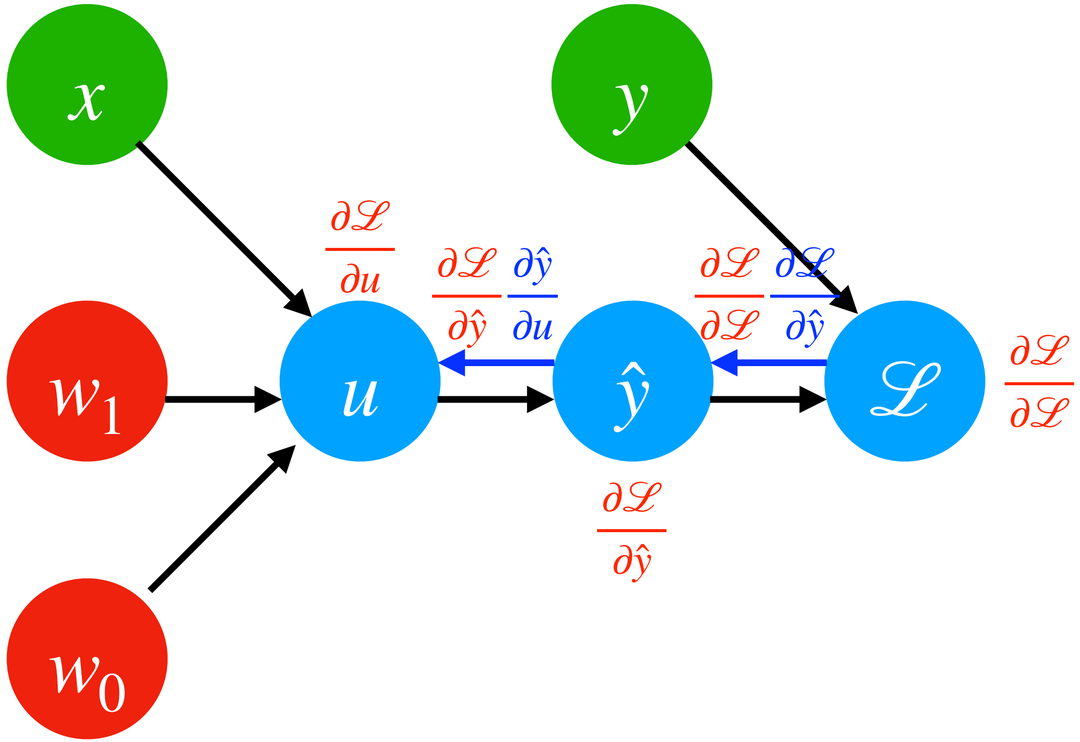
(3) $\mathscr{L} = \boxed{-\, y \log \hat{y} - (1 - y) \log (1 - \hat{y})}$ $\quad \mathrm{BCE}(\hat{y}, y)$

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

Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial \hat{y}} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

(3) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \sigma(u)\left(1 - \sigma(u)\right)$$



Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

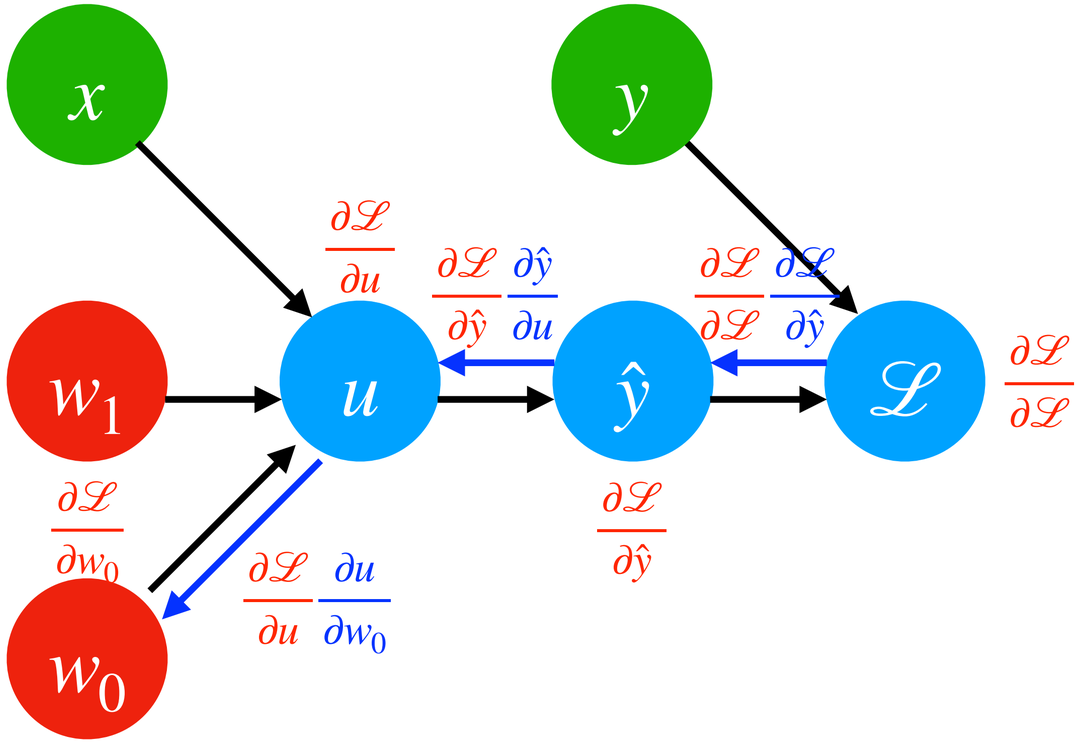
(1) $u = w_0 + w_1 x$

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

$\text{BCE}(\hat{y}, y)$

(3) $\mathscr{L} = \boxed{-\, y \log \hat{y} - (1 - y) \log (1 - \hat{y})}$

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



Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial \hat{y}} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

(3) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \sigma(u)\left(1 - \sigma(u)\right)$$

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

Backpropagation: Logistic Regression with 1D Inputs

Forward Pass:

(1) $u = w_0 + w_1 x$

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

(3) $\mathscr{L} = \boxed{-\, y \log \hat{y} - (1 - y) \log (1 - \hat{y})}$ $\quad \mathrm{BCE}(\hat{y}, y)$

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

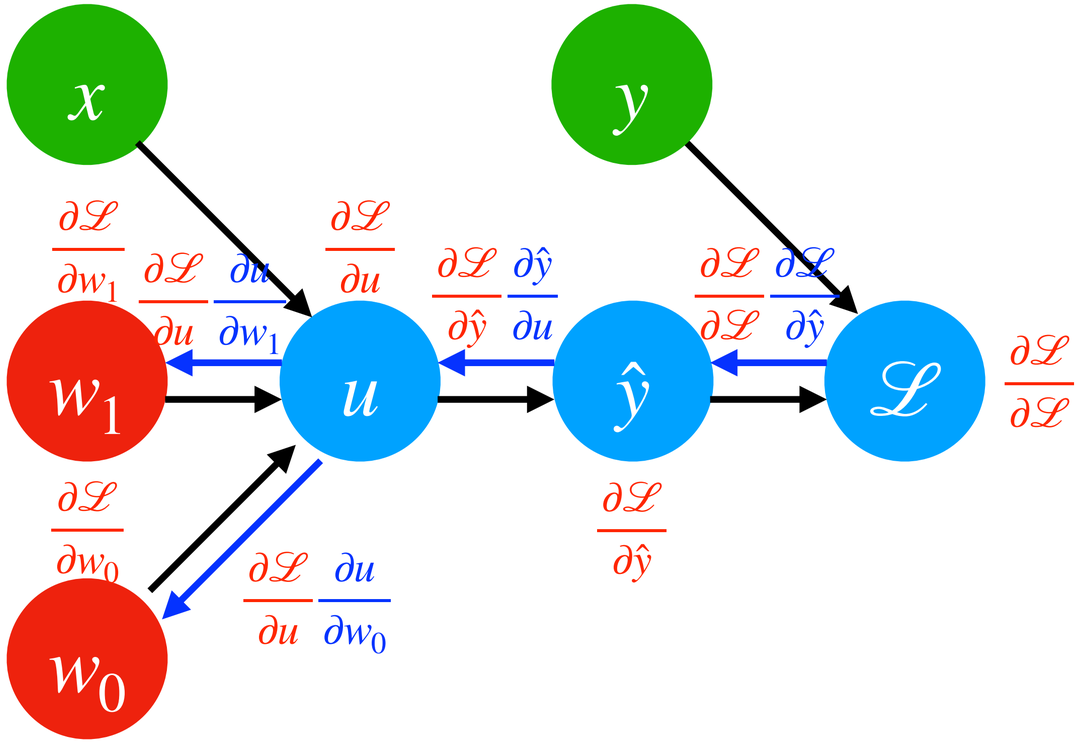
Backward Pass:

(4) $$\frac{\partial \mathscr{L}}{\partial \hat{y}} = \frac{\partial \mathscr{L}}{\partial \mathscr{L}} \frac{\partial \mathscr{L}}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

(3) $$\frac{\partial \mathscr{L}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \sigma(u)\left(1 - \sigma(u)\right)$$

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

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

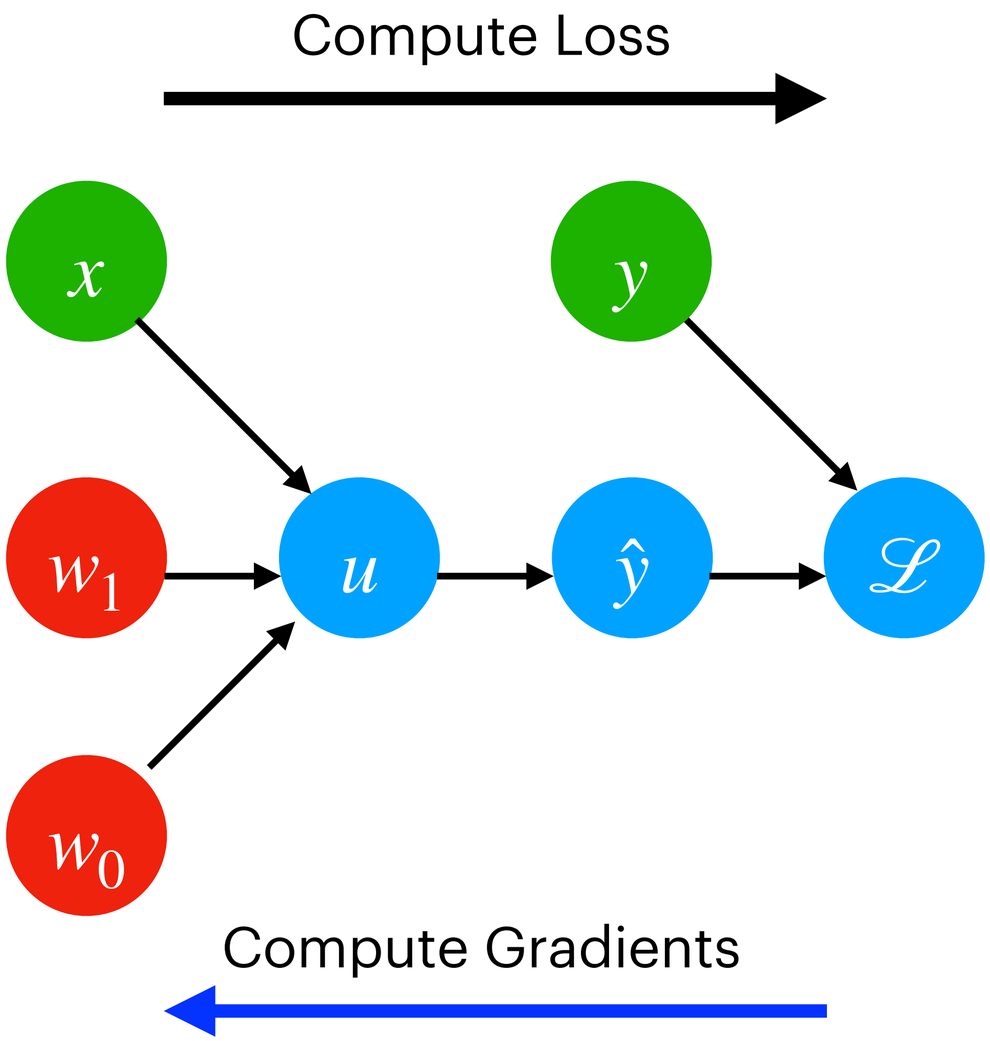


Summary

- 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 w.r.t. its own arguments
- One fw/bw pass per data point:

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

backprop



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