

Deep Learning and Optimization

Unpacking Transformers, LLMs and Diffusion

Session 2

olivier.koch@ensae.fr

Scoring for this course

50% TPs, 50% quiz.

TPs, you get:

- 0 if you don't return anything by end of class
- 1 if you return a decent notebook
- 2 if you found all answers (or made significant effort towards that goal)

Perfection is not the goal. Learning is.

Summary of Session 1

Neural networks are **compute graphs**.

Gradient descent minimizes a loss function over the network's parameters.

Back-propagation allows **efficient learning** (tuning of the network).

We built a Multi-Layer Perception (MLP) from scratch.

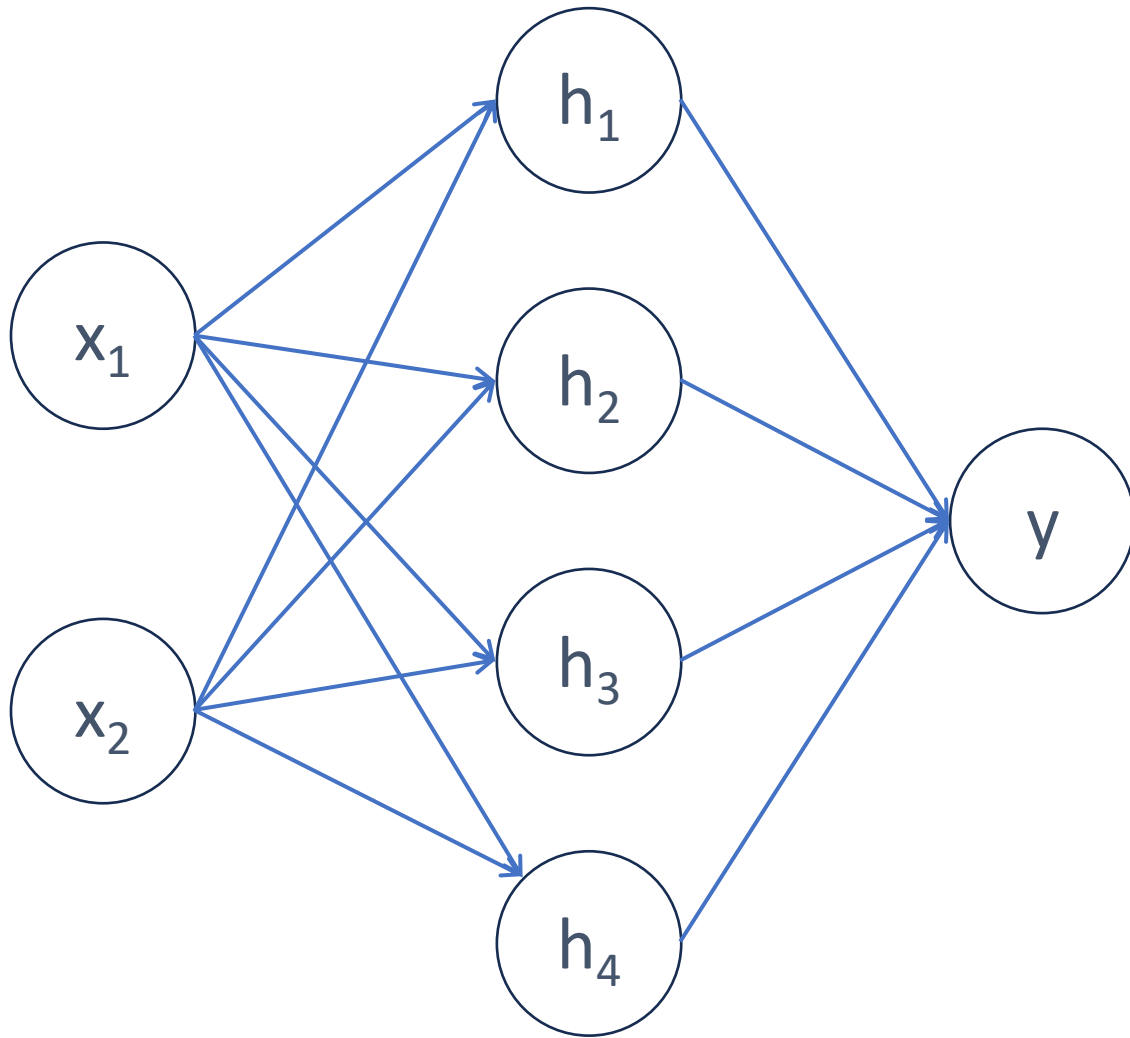
	Session	Date	Content
Foundations	1	Jan, 28	Intro to DL TP: micrograd
	2	Feb, 4	Fundamentals I: inductive bias, loss functions TP: bigram, MLP for next character prediction
	3	Feb, 11	Fundamentals II: DL architectures TP: tensor-based models
Applications	4	Feb, 18	Attention & Transformers TP: GPT from scratch
	5	Feb, 25	DL for Computer vision TP: convnets on CIFAR-10
	6	Mar, 11	VAE and Diffusion TP: diffusion from scratch Quiz / Exam

There is no reason for deep learning to work

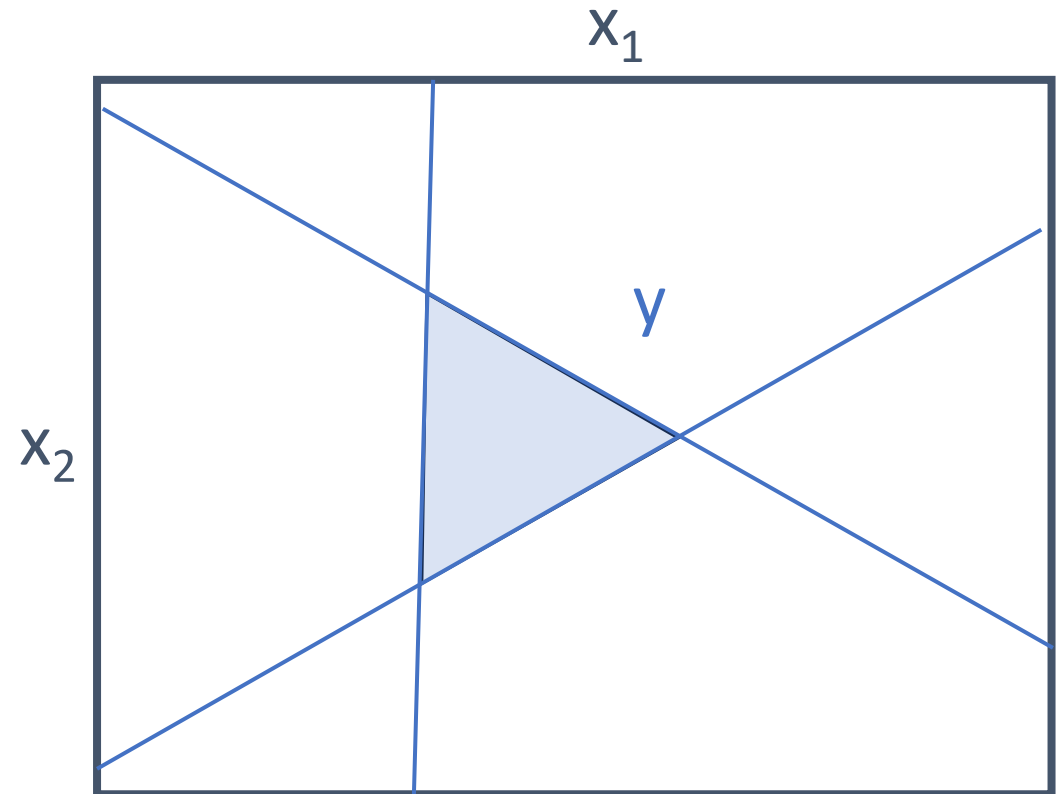
Training: Finding the global optimum of an arbitrary non-convex function is NP-hard (Murty & Khabadi, 1987).

Generalization: deep networks generate way more regions than training samples.

Neural networks generate large number of regions



A neural network generate linear sub-regions in the output space.



Neural networks generate large number of regions

Shallow model

$O(n^d)$ regions

n units, d dim.

Deep model

$O(n^{dL})$ regions

L layers

Deep networks generate **even more** of regions / parameter count

The number of regions grows exponentially with the depth of the network but only polynomially with the width of the hidden layers [1].

→ Deep neural networks create much more complex functions for a fixed parameter budget.

[1] [On the number of linear regions of deep neural networks](#), Montufar et al, NeurIPS 2014.

Deep networks generate **even more** of regions / parameter count

1D input, 5 layers, 10 units / layer → 471 parameters, 161,051 regions

10D input, 5 layers, 50 units / layer → 10,801 parameters, $> 10^{40}$ regions

Number of atoms in the universe: 10^{80}

Let's venture into the variations of deep networks

Network architecture and inductive bias

Loss function

Activation function

Regularization

Initialization

Residual networks

Batch norm, layer norm

Inductive bias

Set of assumptions made by the model about the relationship between input data and output data.

Examples:

- Minimum features
- Maximum margin (SVM)
- Minimum cross-validation error
- Neural net architecture (convnet, transformer)

Inductive bias

No free-lunch theorem

Every learning algorithm is as good as any other when averaged over all sets of problems.

→ You can't just learn « purely from data » without bias.

Do networks have to be deep?

Empirical evidence: shallow networks don't work as well as deeper ones.

Intuition:

1. Deep networks can represent more complex functions with the same parameter count
2. Deep networks are easier to train
3. Deep network impose better inductive bias

The challenges of depth

- Vanishing/exploding gradients
- Shattered gradients

In short, depth is required but comes with challenges that need to be addressed.

Let's venture into the variations of a deep networks

Network architecture and inductive bias

Loss function

Activation function

Regularization

Initialization

Residual networks

Batch norm, layer norm

Let's venture into the variations of a deep networks

Loss functions are a fundamental component of a deep learning.

We will learn about cross-entropy on a simple model (bigram)...

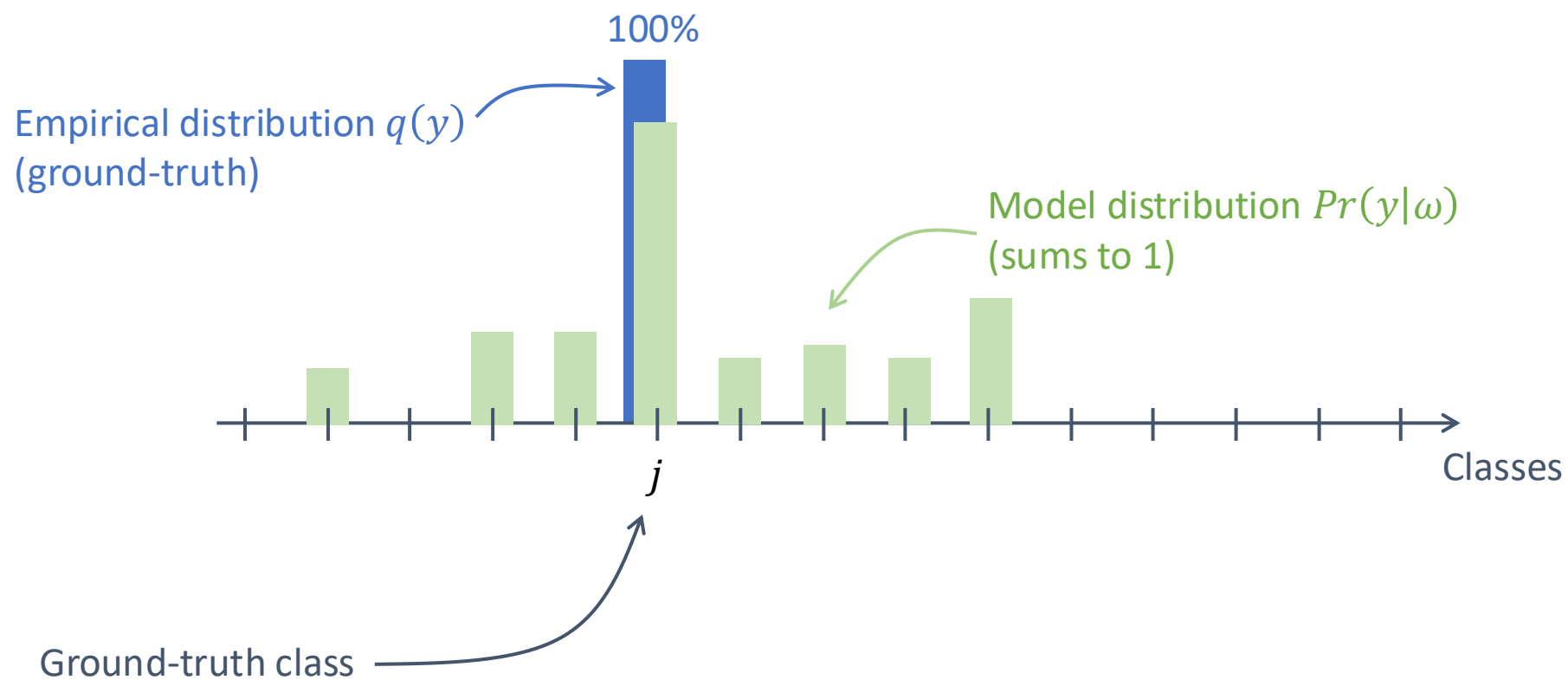
... and reuse it throughout this course!

Let's venture into the variations of a deep networks

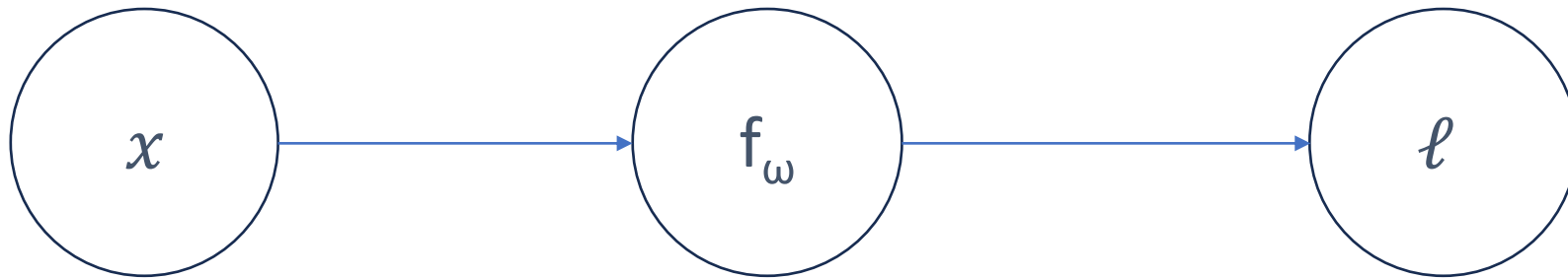
Fundamentally, the loss function expresses the distance between two distributions:

- The distribution of real data
- The distribution of predicted data

Loss functions: example for classification



Loss functions

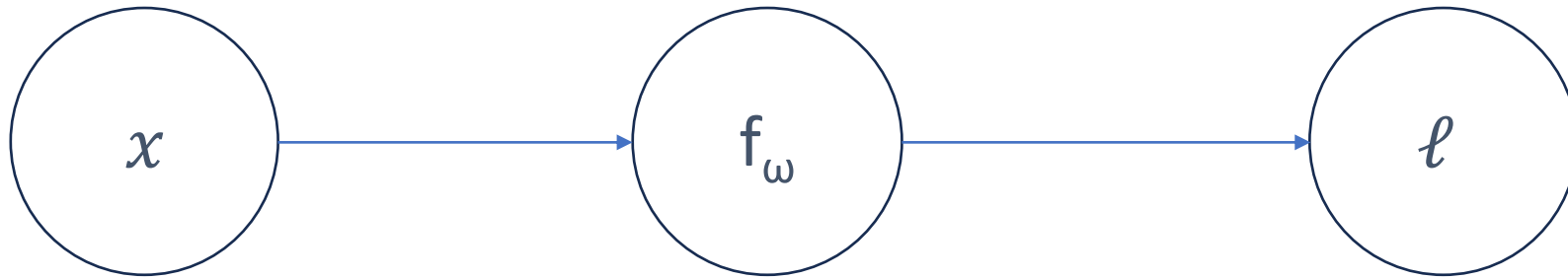


$$\ell = (y - \hat{y})^2$$

ground-truth

A blue arrow originates from the text "ground-truth" and points upwards and to the left, terminating at the variable y in the equation $\ell = (y - \hat{y})^2$ shown in the block above.

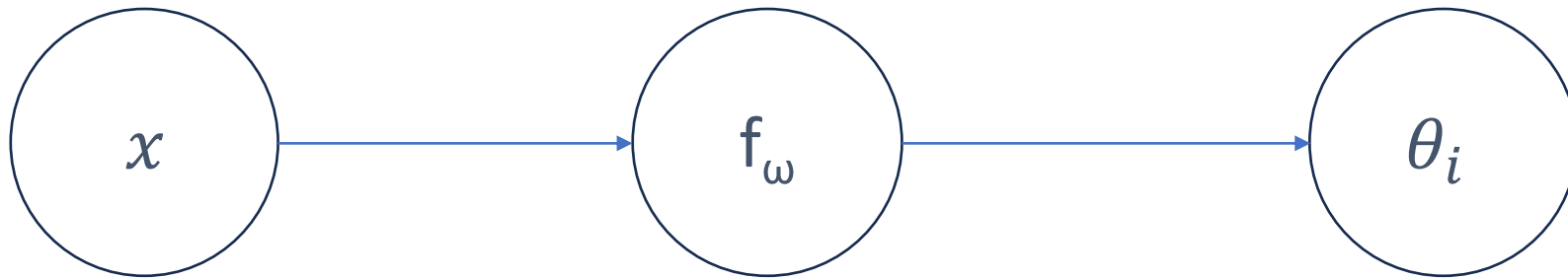
Loss functions



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

Expressed as the distance between two distributions

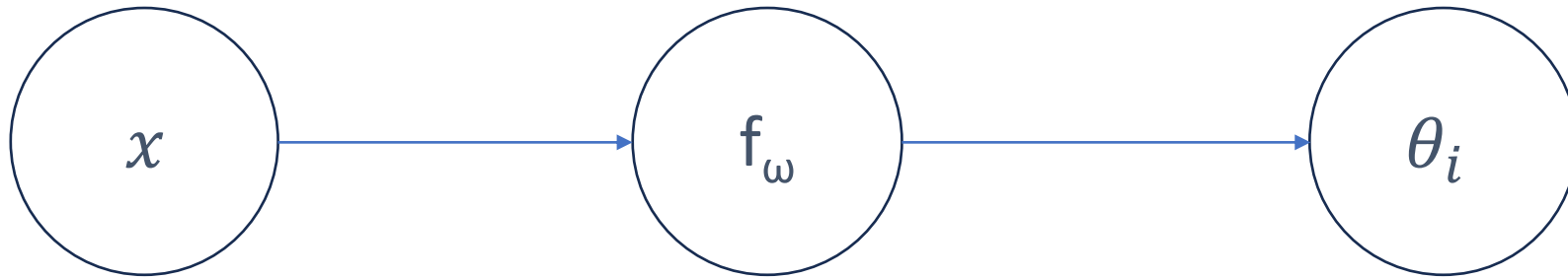
Loss functions



$$f_\omega(x_i) = \theta_i$$

Parameters of a distribution

Loss functions

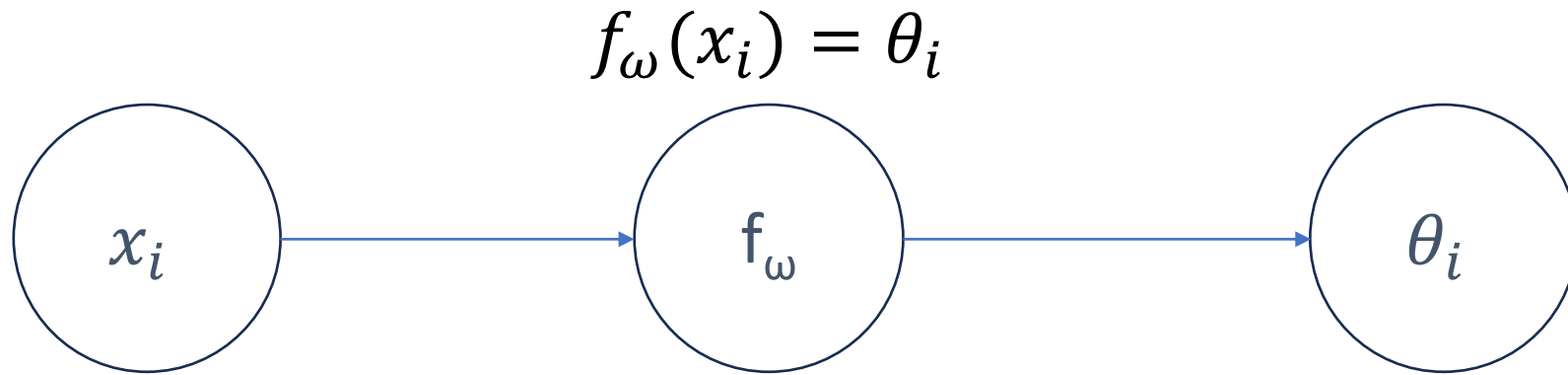


$$f_{\omega}(x_i) = \theta_i$$

The distribution is chosen based on the domain.

The model computes the optimal θ_i given the data.

Loss functions

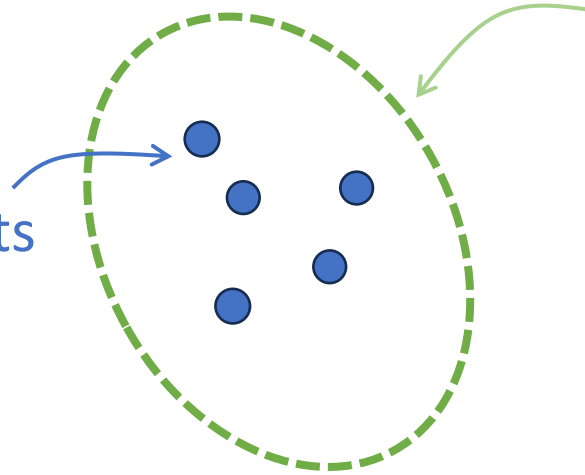


Example: univariate regression

$$\Pr(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}$$

θ ↑

Given example data points



Find the optimal parameters of the distribution $f_{\omega}(x_i) = \theta_i$

The parameters ω are *the same* across all samples x_i

Loss functions

$$\hat{\omega} = \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | x_i) \right]$$

$$= \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | \theta_i) \right]$$

$$= \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | f_{\omega}(x_i)) \right]$$

Loss functions

$$\hat{\omega} = \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | x_i) \right]$$

$\Pr(y_1, y_2, \dots, y_N | x_1, x_2, \dots, x_N)$
Data is assumed i.i.d

$$= \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | \theta_i) \right]$$

$$= \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | f_{\omega}(x_i)) \right]$$

Loss functions

$$\hat{\omega} = \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | x_i) \right]$$

$$= \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | \theta_i) \right]$$

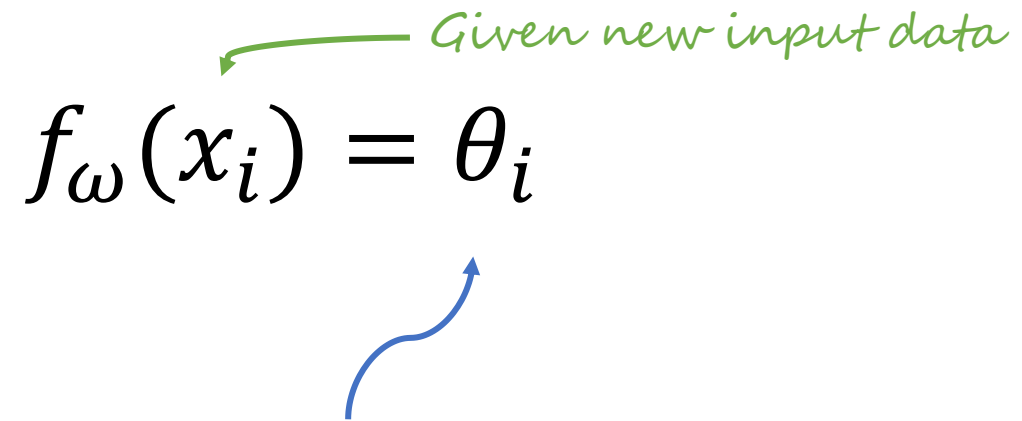
$$= \operatorname{argmax}_{\omega} \left[\prod_{i=1}^N \Pr(y_i | f_{\omega}(x_i)) \right]$$

$$= \operatorname{argmax}_{\omega} \left[\sum_{i=1}^N \log[\Pr(y_i | f_{\omega}(x_i))] \right]$$

$$= \operatorname{argmin}_{\omega} \left[- \sum_{i=1}^N \log[\Pr(y_i | f_{\omega}(x_i))] \right]$$

Negative log likelihood (NLL)

Inference



The diagram shows the equation $f_{\omega}(x_i) = \theta_i$. A green curved arrow points from the text "Given new input data" to the variable x_i in the function. A blue curved arrow points from the text "Optimal choice: maximum of the distribution" to the variable θ_i .

$$f_{\omega}(x_i) = \theta_i$$

Optimal choice: maximum of the distribution

...or sample from the distribution!

Loss functions

In the case of the univariate regression, the NLL is equivalent to least squares.

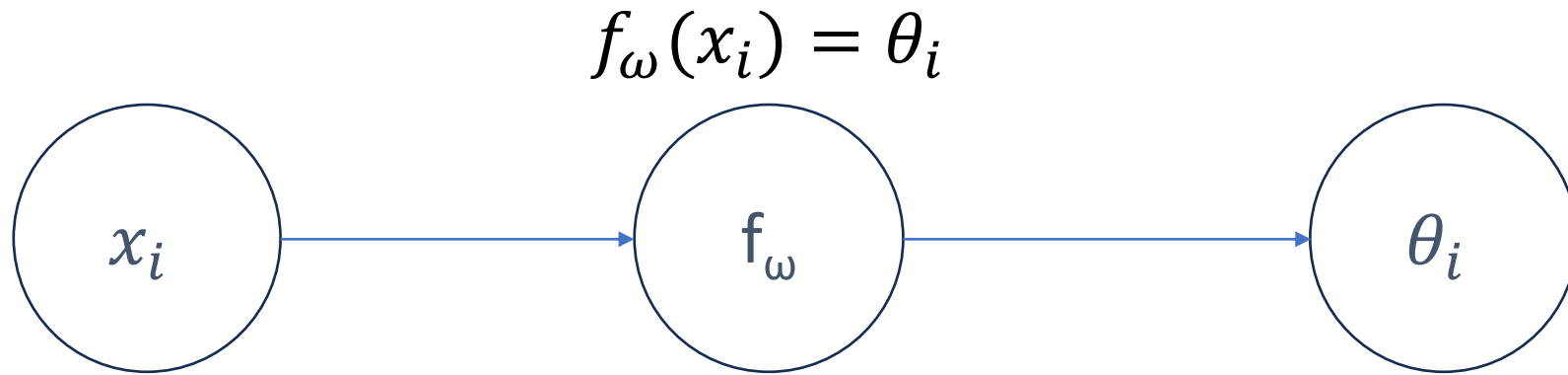
$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \sum_{i=1}^N \log[\operatorname{Pr}(y_i | f_{\omega}(x_i))] \right] \quad \text{Negative log likelihood (NLL)}$$

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \sum_{i=1}^N \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \mu)^2}{2\sigma^2}} \right] \right]$$

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \sum_{i=1}^N \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - f_{\omega}(x_i))^2}{2\sigma^2}} \right] \right]$$

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[\sum_{i=1}^N (y_i - f_{\omega}(x_i))^2 \right] \quad \text{least squares}$$

Loss functions

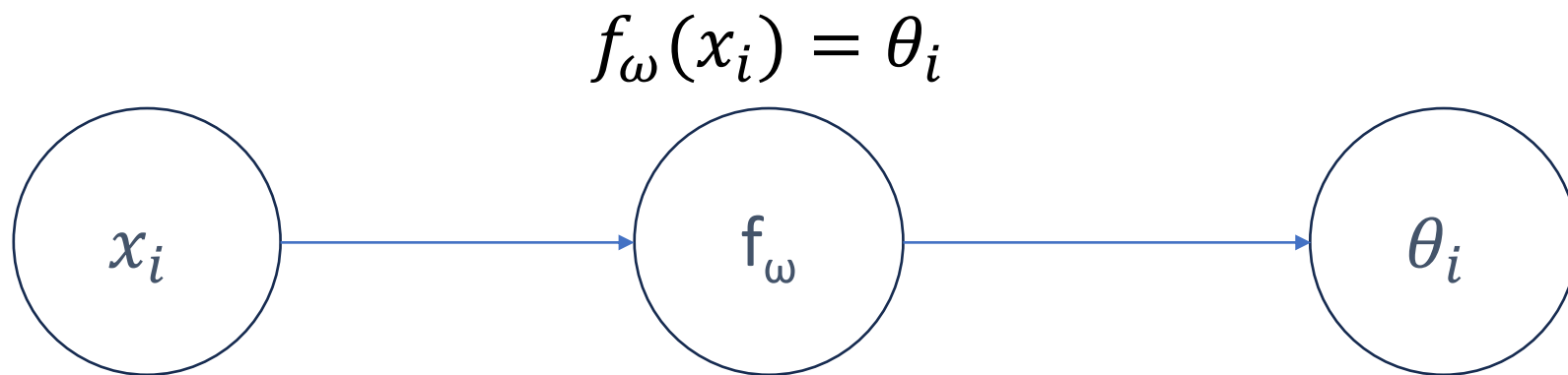


Example: binary classification

$$\Pr(y|\lambda) = \begin{cases} 1 - \lambda & y = 0 \\ \lambda & y = 1 \end{cases}$$

\uparrow
 θ

Loss functions



Example: binary classification

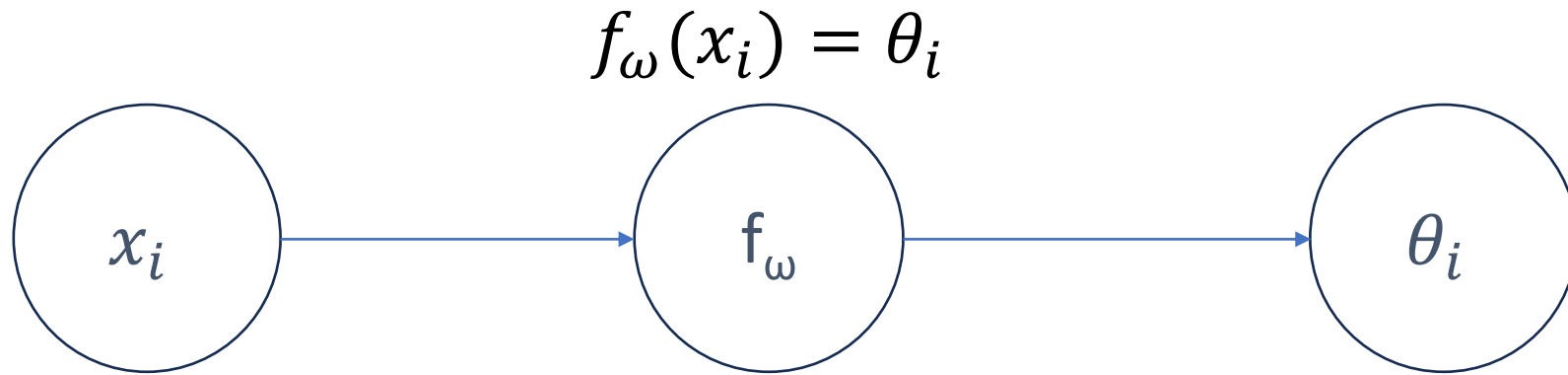
$$\Pr(y|\lambda) = \begin{cases} 1 - \lambda & y = 0 \\ \lambda & y = 1 \end{cases}$$

NLL

$$\ell = \sum_{i=1}^N -(1 - y_i) \log[1 - \sigma(f_{\omega}(x_i))] - y_i \log[\sigma(f_{\omega}(x_i))]$$

σ : sigmoid function

Loss functions

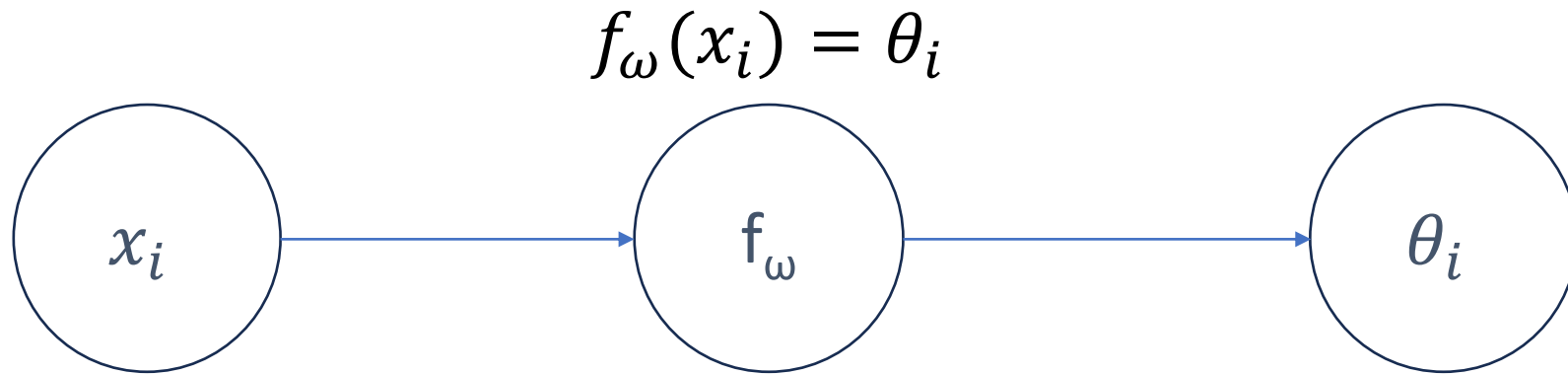


Example: multiclass classification

$$\Pr(y = k) = \lambda_k \qquad \sum \lambda_k = 1 \qquad 0 < \lambda_k < 1$$

$$\Pr((y = k|x)) = \text{softmax}_k[f_{\omega}(x)] \qquad \text{softmax}(\mathbf{z}) = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

Loss functions



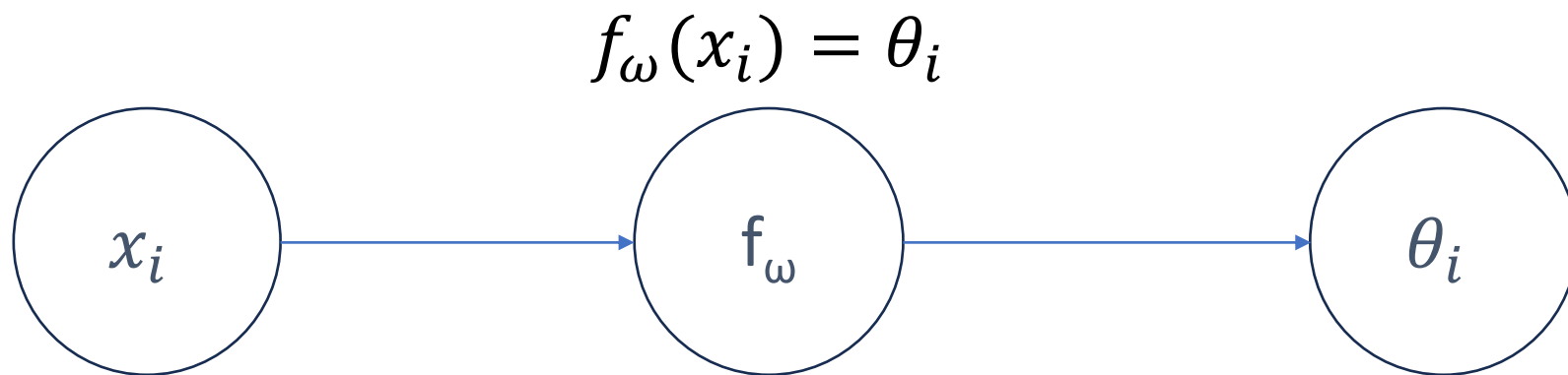
Example: multiclass classification

$$\Pr(y = k) = \lambda_k \quad \sum \lambda_k = 1$$

NLL

$$\ell = - \sum_{i=1}^N \log \left[\text{softmax}_{y_i} [f_{\omega}(x_i)] \right]$$

Loss functions



Example: multiclass classification

$$\Pr(y = k) = \lambda_k \quad \sum \lambda_k = 1$$

NLL

$$\ell = - \sum_{i=1}^N \log \left[\text{softmax}_{y_i} [f_{\omega}(x_i)] \right]$$

Wait, can we differentiate softmax?

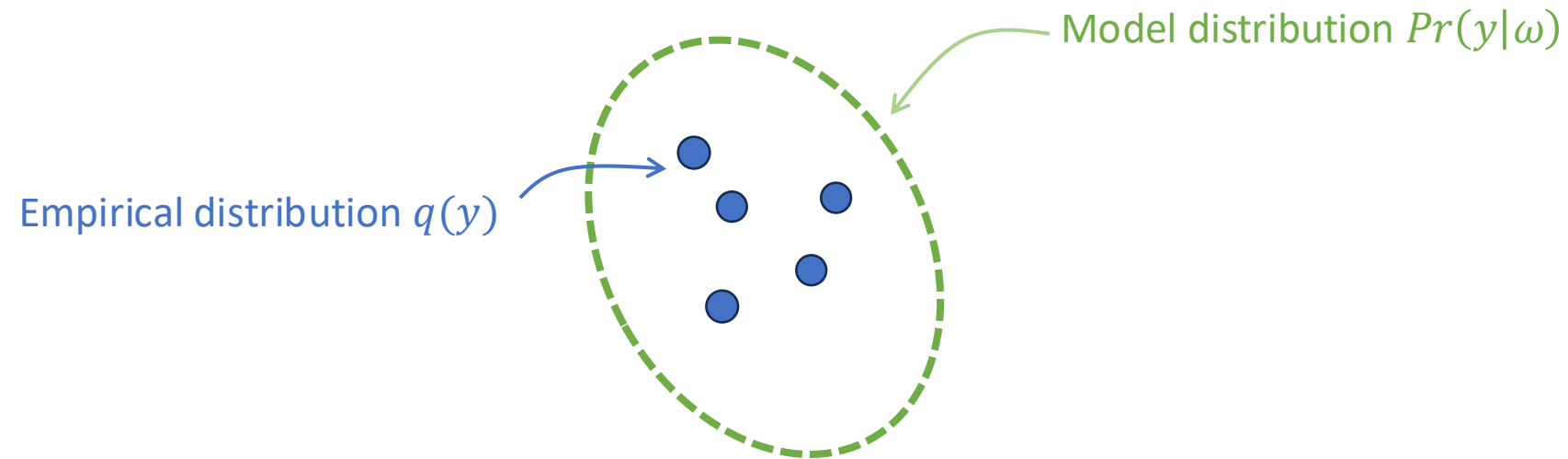
Yes, we can!

Loss functions

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \sum_{i=1}^N \log[\operatorname{Pr}(y_i | f_{\omega}(x_i))] \right] \quad \text{Negative log likelihood (NLL)}$$

is equivalent to the cross-entropy loss

Loss functions



Goal: Minimize divergence between q and p

Loss functions

Given two distributions $q(z)$ and $p(z)$, the distance between the two distributions can be computed with:

$$D_{KL}(q|p) = \int_{-\infty}^{\infty} q(z) \log(q(z)) dz - \int_{-\infty}^{\infty} q(z) \log(p(z)) dz$$

Given an empirical distribution $q(y)$ and a model distribution $Pr(y|\omega)$, we want to minimize the KL divergence:

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[\int_{-\infty}^{\infty} q(y) \log(q(y)) dy - \iint_{-\infty}^{\infty} q(y) \log[Pr(y|\omega)] dy \right]$$

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \iint_{-\infty}^{\infty} q(y) \log[Pr(y|\omega)] dy \right]$$

Loss functions

Given two distributions $q(z)$ and $p(z)$, the distance between the two distributions can be computed with:

$$D_{KL}(q|p) = \int_{-\infty}^{\infty} q(z) \log(q(z)) dz - \int_{-\infty}^{\infty} q(z) \log(p(z)) dz$$

Given an empirical distribution $q(y)$ and a model distribution $Pr(y|\omega)$, we want to minimize the KL divergence:

entropy of q  *divergence between q and p* 

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[\int_{-\infty}^{\infty} q(y) \log(q(y)) dy - \iint_{-\infty}^{\infty} q(y) \log[Pr(y|\omega)] dy \right]$$

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \iint_{-\infty}^{\infty} q(y) \log[Pr(y|\omega)] dy \right]$$

Loss functions

Definition of **cross-entropy loss** of distribution q relative to distribution p over the set \mathcal{X} :

$$H(q, p) = -E_q[\log p]$$

where $E_q[\cdot]$ is the expected value operator with respect to distribution q .

In the continuous case:

$$H(q, p) = - \int_{\mathcal{X}} Q(x) \log P(x) dx$$

In the discrete case:

$$H(q, p) = - \sum_{x \in \mathcal{X}} q(x) \log p(x)$$

Loss functions

Intuition behind cross-entropy $H(q, p) = -E_q[\log p]$

q = the truth, p = the prediction

$\log(p)$ is the "surprise" (low p -> high surprise)

E_q = average surprise over all observations, i.e. $\log(p)$ weighted by q

Loss functions

Given an empirical distribution $q(y)$ and a model distribution $Pr(y|\omega)$, we want to minimize the KL divergence:

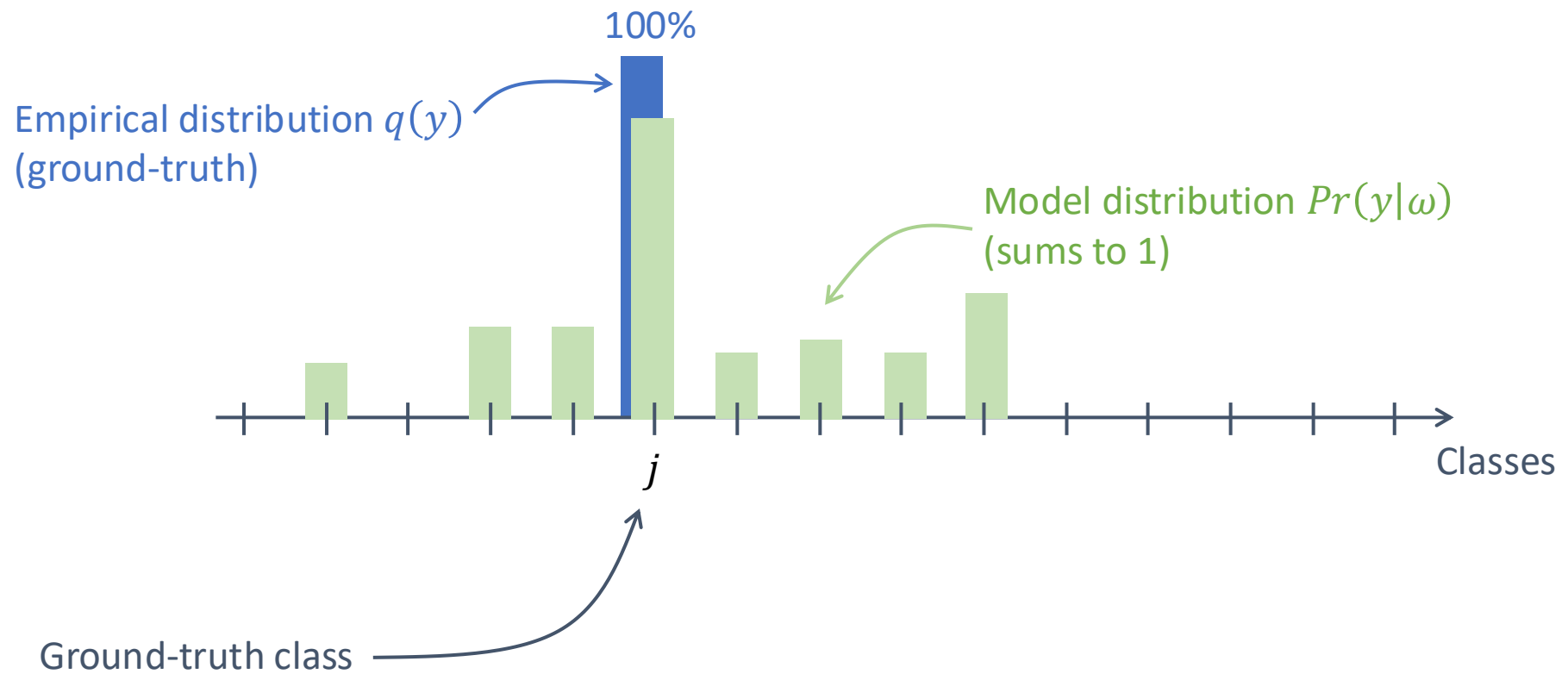
$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \iint_{-\infty}^{\infty} q(y) \log[Pr(y|\omega)] dy \right] \quad \text{cross-entry loss}$$

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \iint_{-\infty}^{\infty} \left(\frac{1}{N} \sum_{i=1}^N \delta[y - y_i] \right) \log[Pr(y|\omega)] dy \right]$$

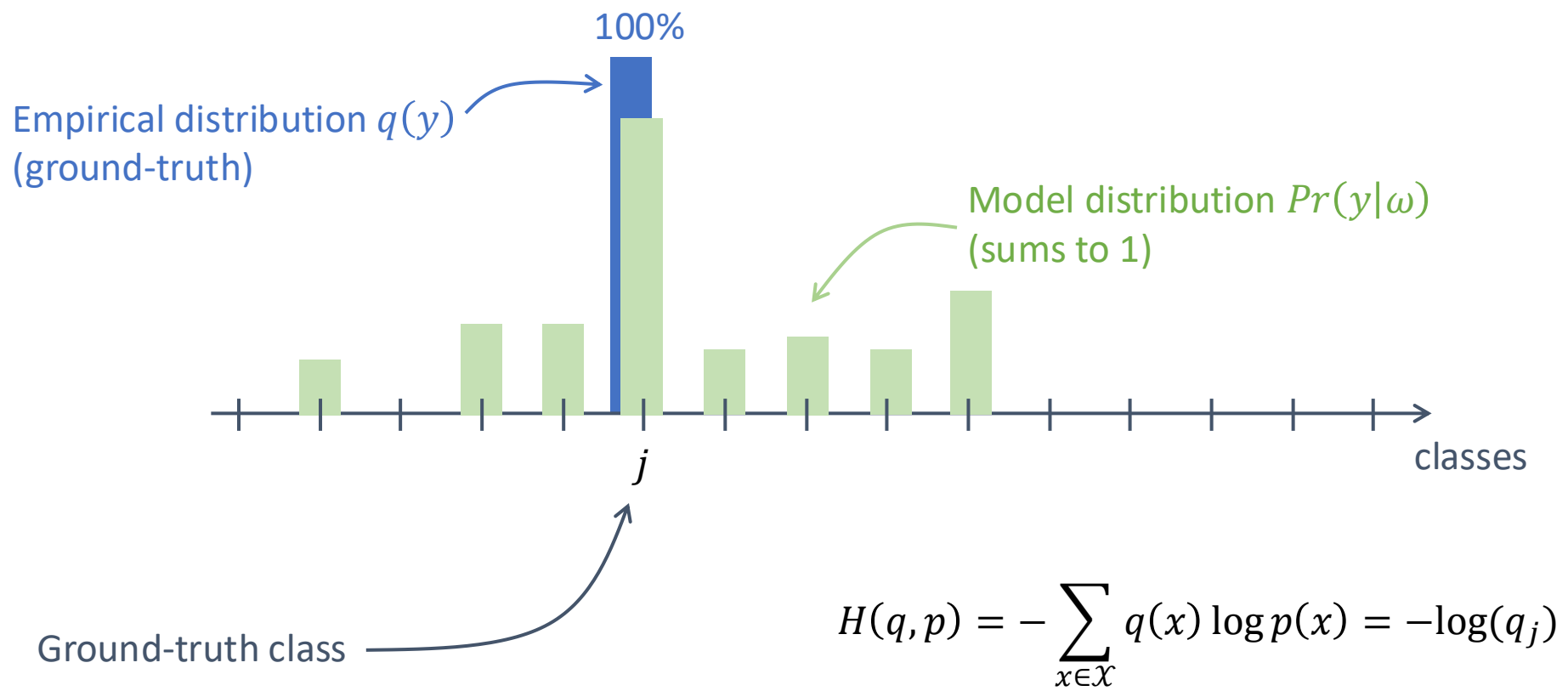
$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \frac{1}{N} \sum_{i=1}^N \log[Pr(y_i|\omega)] \right]$$

$$\hat{\omega} = \operatorname{argmin}_{\omega} \left[- \sum_{i=1}^N \log[Pr(y_i|\omega)] \right] \quad \text{NLL}$$

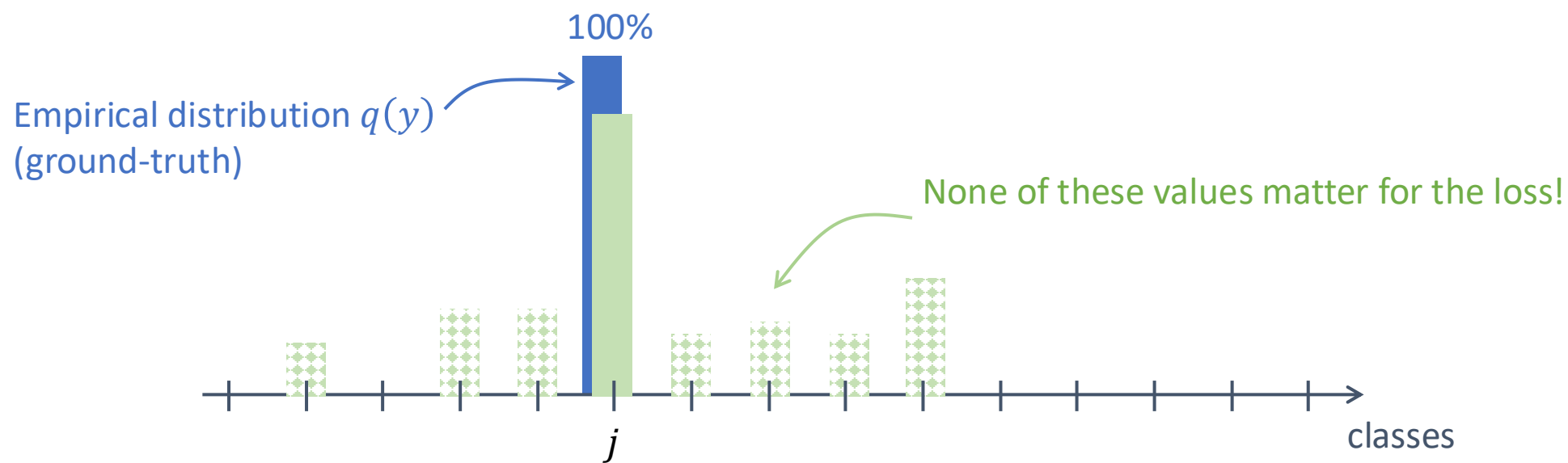
Loss functions: example for classification



Loss functions: example for classification



Loss functions: example for classification



Ground-truth class

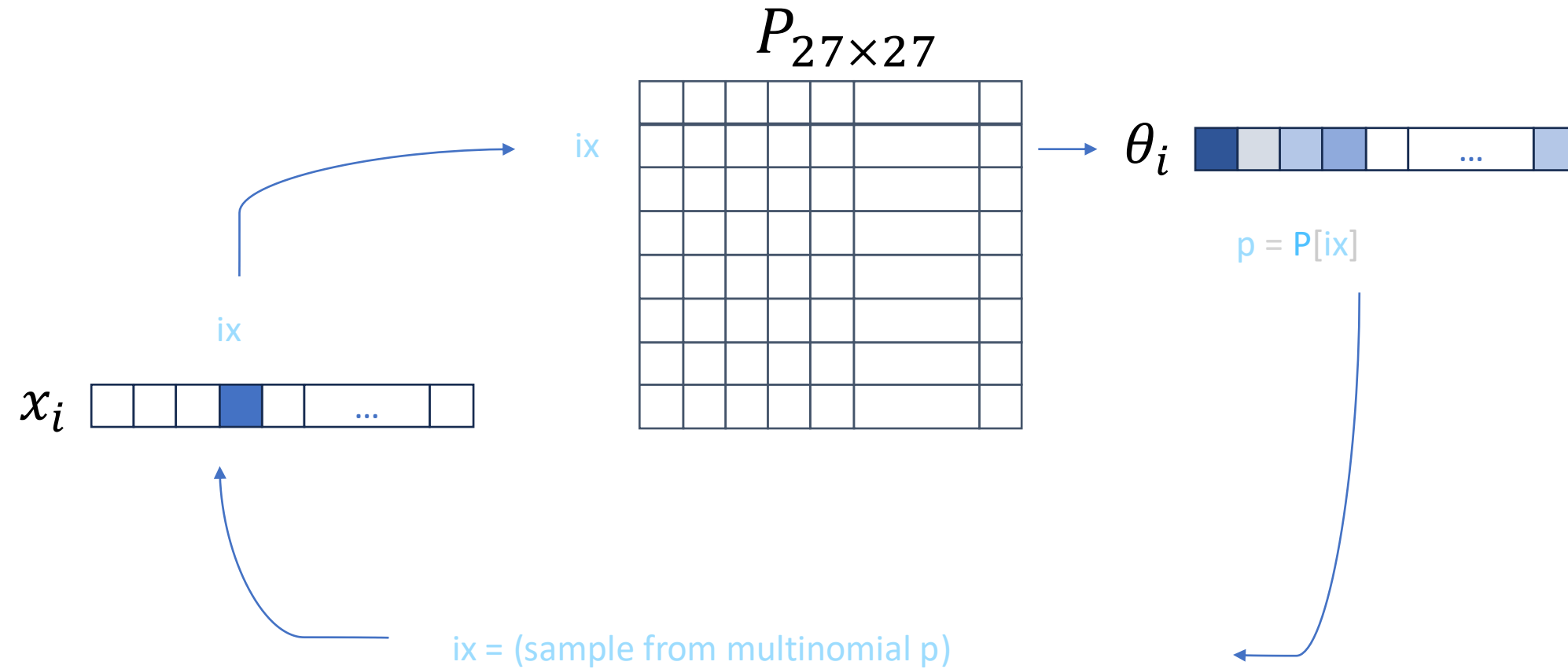
$$H(q, p) = - \sum_{x \in \mathcal{X}} q(x) \log p(x) = -\log(q_j)$$

TP2: makemore

Goal: Given a bunch of names, generate more “name-like” words.

1. Build a simple bigram model for next-character prediction
2. Build the same bigram model using the NLL loss
3. Implement a better model: [[Bengio et al., 2003](#)]

Step 1: bigram “by hand”



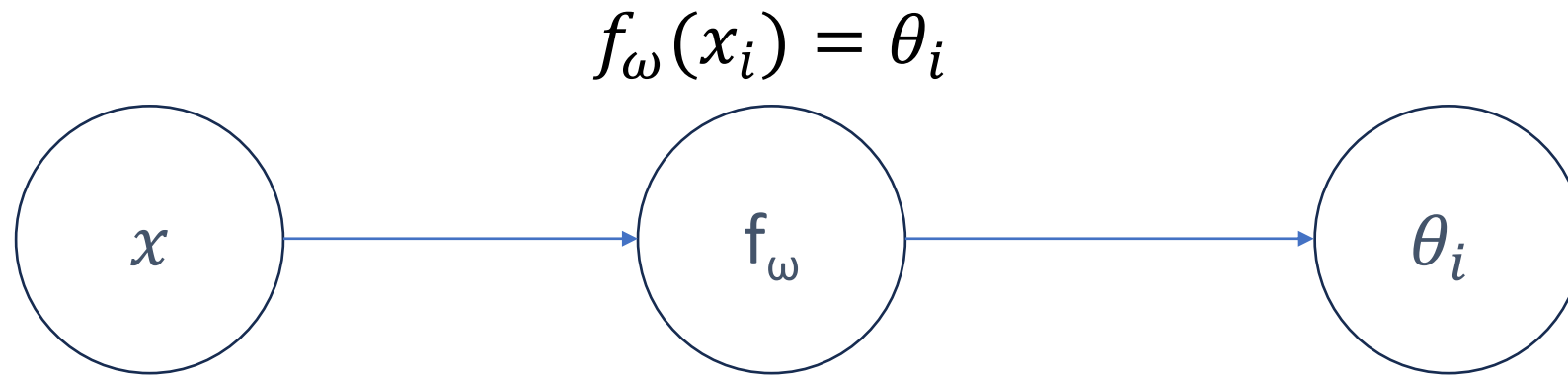
Step 1: bigram “by hand”

The dot character (.) marks the beginning and end of a word.

When sampling, you need to stop when you hit that special character.

How to initialize a torch matrix of size 27x27 containing floats?

Step 2: bigram as a learnable matrix



$$x_i = [0, 0, 0, 1, 0, \dots, 0]$$

One-hot encoding of letter 'd'

ω is a matrix $W_{27 \times 27}$ such that

$\theta_i = \text{softmax}(x \cdot W)$ is a $N \times 27$ vector representing the distribution of the next character for each sample

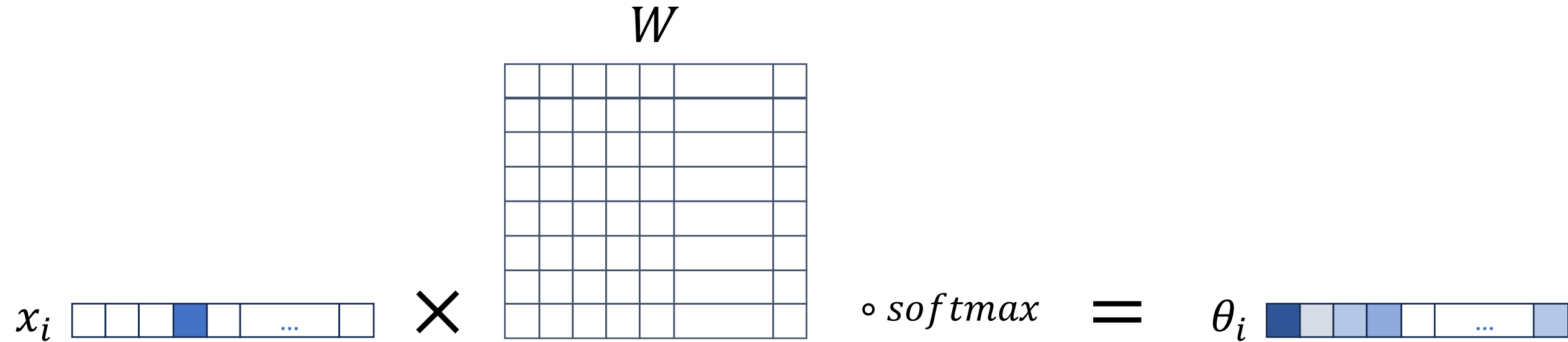
Step 2: bigram as a learnable matrix

x_i

--	--	--	--	--	--	--	--	--	--

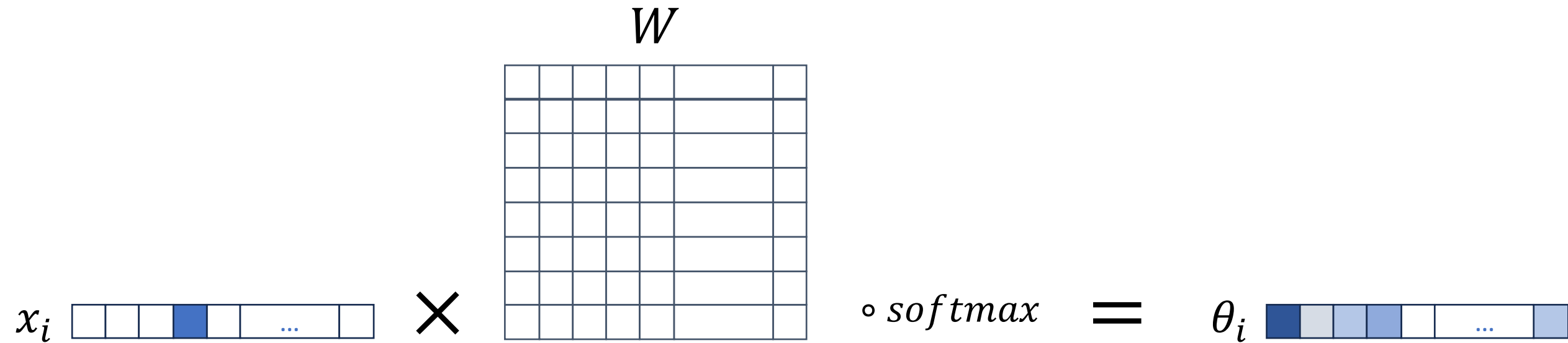
One-hot encoding of letter 'd'

Step 2: bigram as a learnable matrix



One-hot encoding of letter 'd'

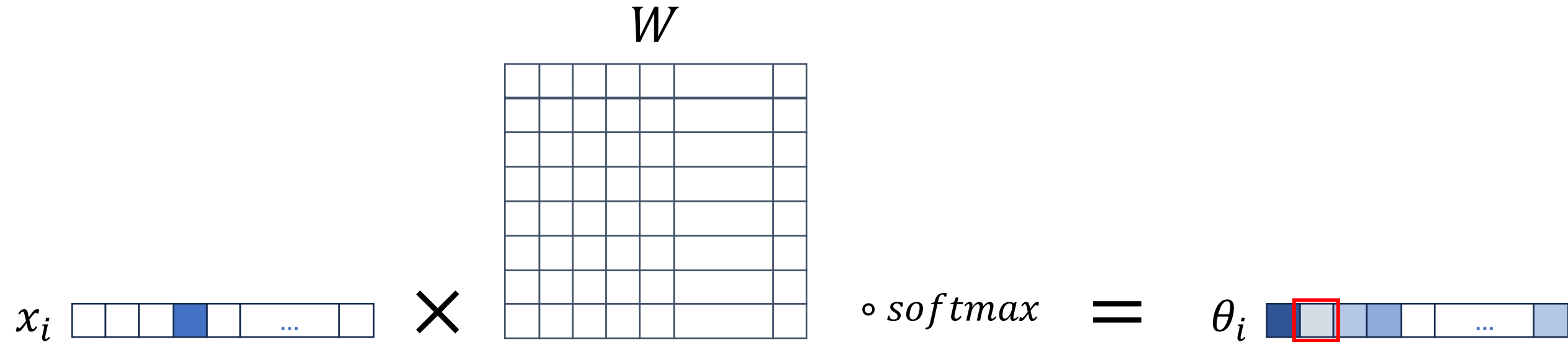
Step 2: bigram as a learnable matrix



We want to minimize the KL divergence or NLL

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \left[- \sum_{i=1}^N \log[Pr(y_i|\omega)] \right] \quad \text{NLL}$$

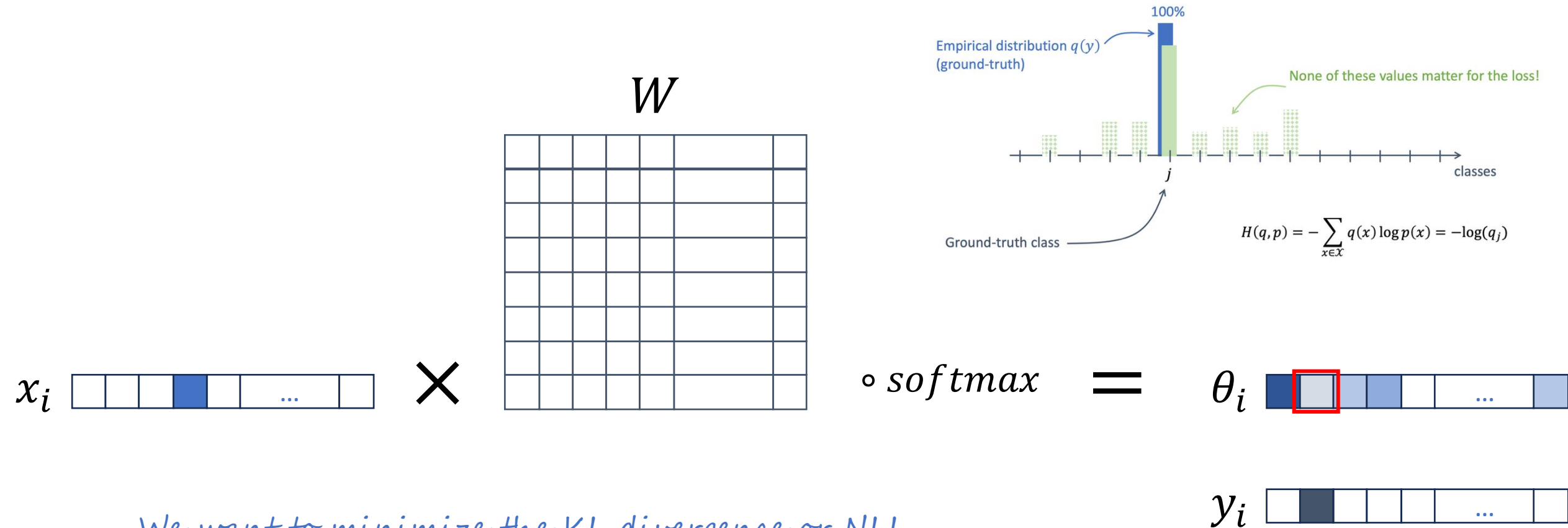
Step 2: bigram as a learnable matrix



We want to minimize the KL divergence or NLL

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \left[- \sum_{i=1}^N \log \boxed{Pr(y_i | \omega)} \right] \quad \text{NLL}$$

Step 2: bigram as a learnable matrix



$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \left[- \sum_{i=1}^N \log \boxed{Pr(y_i | \omega)} \right] \quad \text{NLL}$$

Step 2: bigram as a learnable matrix



x_i



θ_i



Loss = mean of log of the red values

Step 2: bigram as a learnable matrix

#forward pass

xenc = ??? # encode xs with F.one_hot

logits = ??? # multiply by W

counts = ??? # exp(logits)

probs = ??? # softmax

loss = ??? # sum of logs of probs

A few tips...

```
import torch.nn.functional as F
```

$x \cdot W$ is written as `x @ W`

One-hot encoding: `F.one_hot(x, num_classes=...).float()`

For inference `z.multinomial()`

Normalizing a matrix $W_{27 \times 27}$ by row requires the `keepdim` parameter somewhere...

A few tips...

```
>>> a
tensor([[1, 2, 3, 4, 5, 6],
        [1, 2, 3, 4, 5, 6],
        [1, 2, 3, 4, 5, 6],
        [1, 2, 3, 4, 5, 6],
        [1, 2, 3, 4, 5, 6],
        [1, 2, 3, 4, 5, 6],
        [1, 2, 3, 4, 5, 6]])
>>> a.sum(axis=1)
tensor([21, 21, 21, 21, 21, 21, 21])
>>> a.sum(axis=1, keepdim=True)
tensor([[21],
        [21],
        [21],
        [21],
        [21],
        [21],
        [21]])
```

Step 3: A Neural Probabilistic Language Model

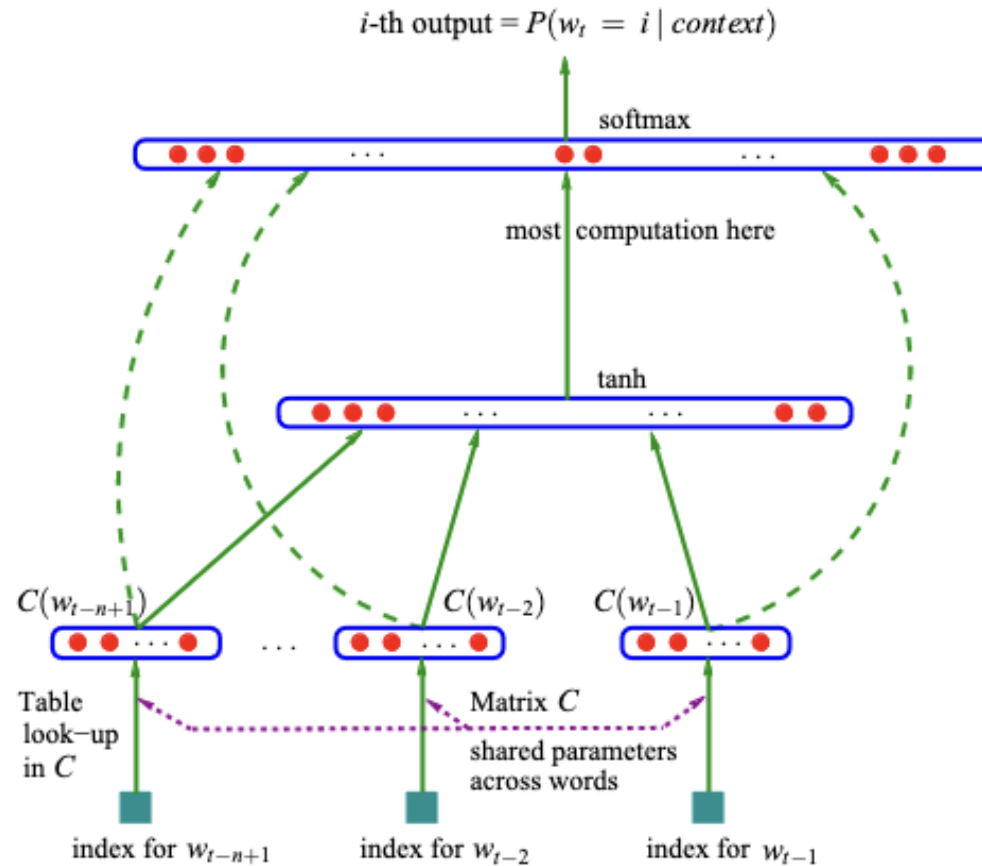
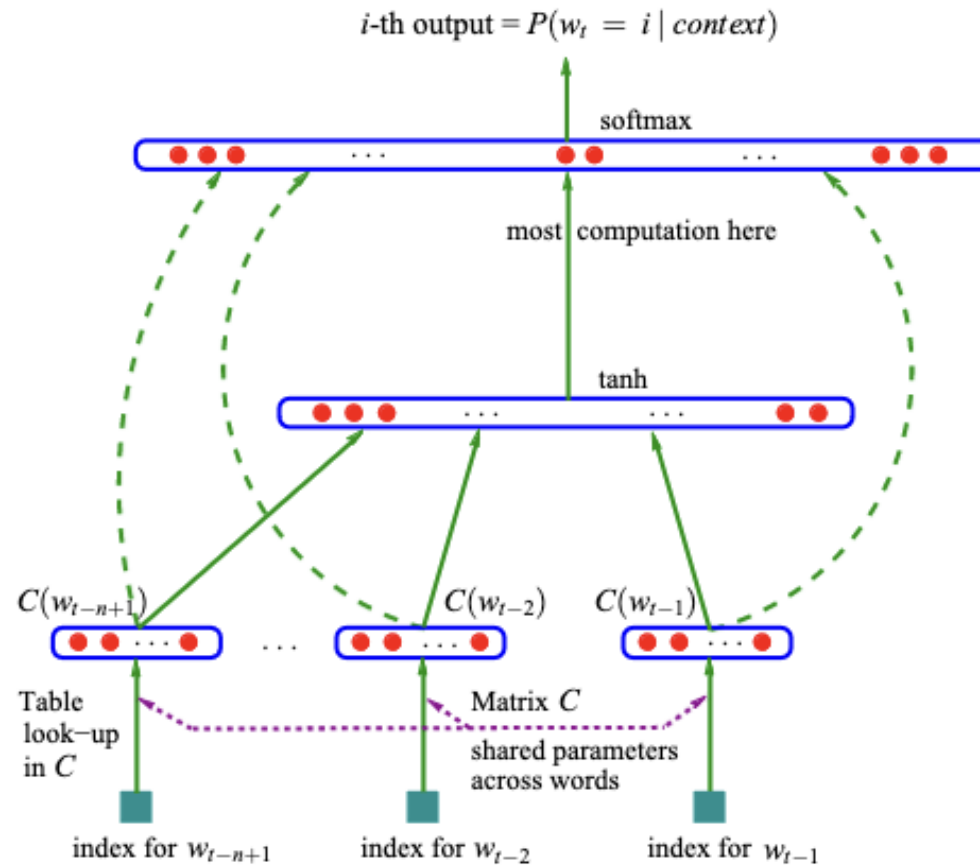


Figure 1: Neural architecture: $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i -th word feature vector.

Step 3: A Neural Probabilistic Language Model

```
X_train = tensor([
  [ 0, 0, 0],
  [ 0, 0, 5],
  [ 0, 5, 2], ...,
  [25, 1, 14],
  [ 1, 14, 14],
  [14, 14, 9]])
```

$C_{27 \times 10}$ = dictionary



$$\text{logits} = \tanh(W_2 \cdot h + b_2)$$

$$h = \tanh(W_1 \cdot \text{emb} + b_1)$$

$\text{emb} = C[X_train[ix]]$
 $\text{emb} = \text{emb.view(...)}$

Figure 1: Neural architecture: $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i -th word feature vector.