Deep Learning and Optimization

Unpacking Transformers, LLMs and Diffusion

Session 2

olivier.koch@ensae.fr

slack #ensae-dl-2025

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.

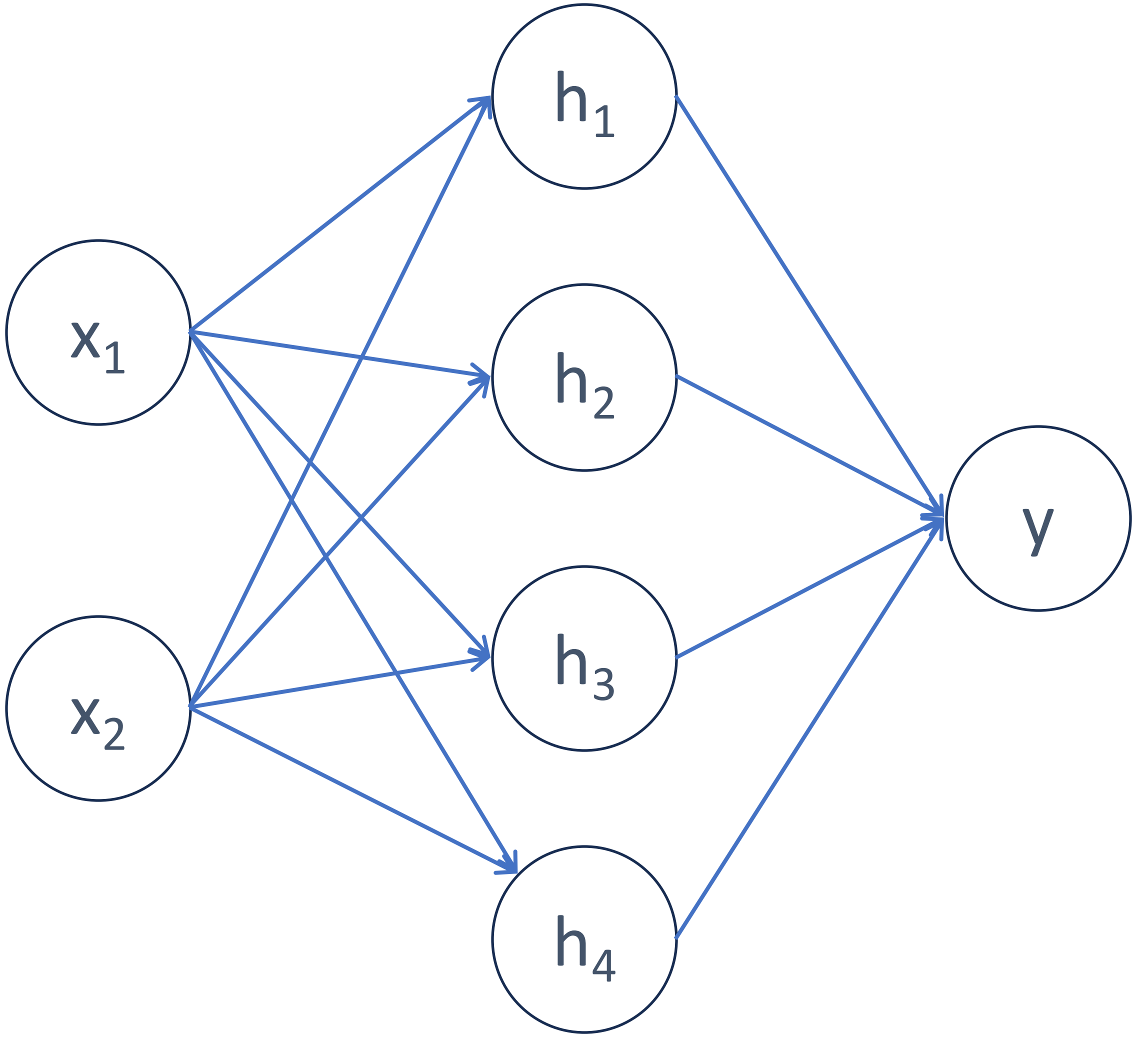
Session	Date	Topic
1	05-02-2025	Intro to Deep Learning Practical: micrograd
2	12-02-2025	DL fundamentals ● Backprop ● Loss functions Practical: bigram, MLP for next character prediction
3	19-02-2025	DL Fundamentals II ● Activation functions ● Regularization ● Initialization ● Residual networks ● Normalization Practical: tensor-based models
	26-02-2025	Pas de cours
4	05-03-2025	Attention & Transformers Practical: GPT from scratch
5	12-03-2025	DL for computer vision Practical: Convnets for CIFAR-10
6	19-03-2025	VAE & Diffusion models Practical: diffusion from scratch Quiz/Exam

There is no reason for deep learning to work

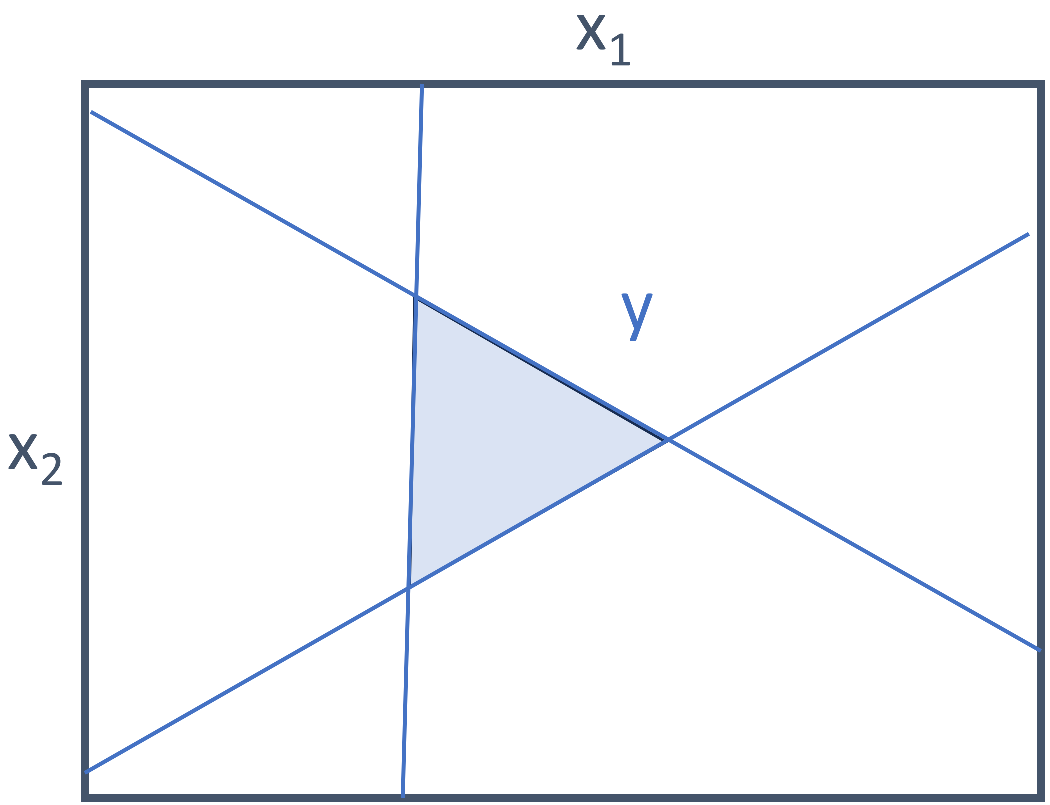
Training: Finding the global optimum of an arbitrary non-convex function is NP-hard (Murty & Kabadi, 1987).

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.

Wolpert, D. H.; Macready, W. G. (1997). "No Free Lunch Theorems for Optimization". *IEEE Transactions on Evolutionary Computation*. 1: 67–82.

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)…

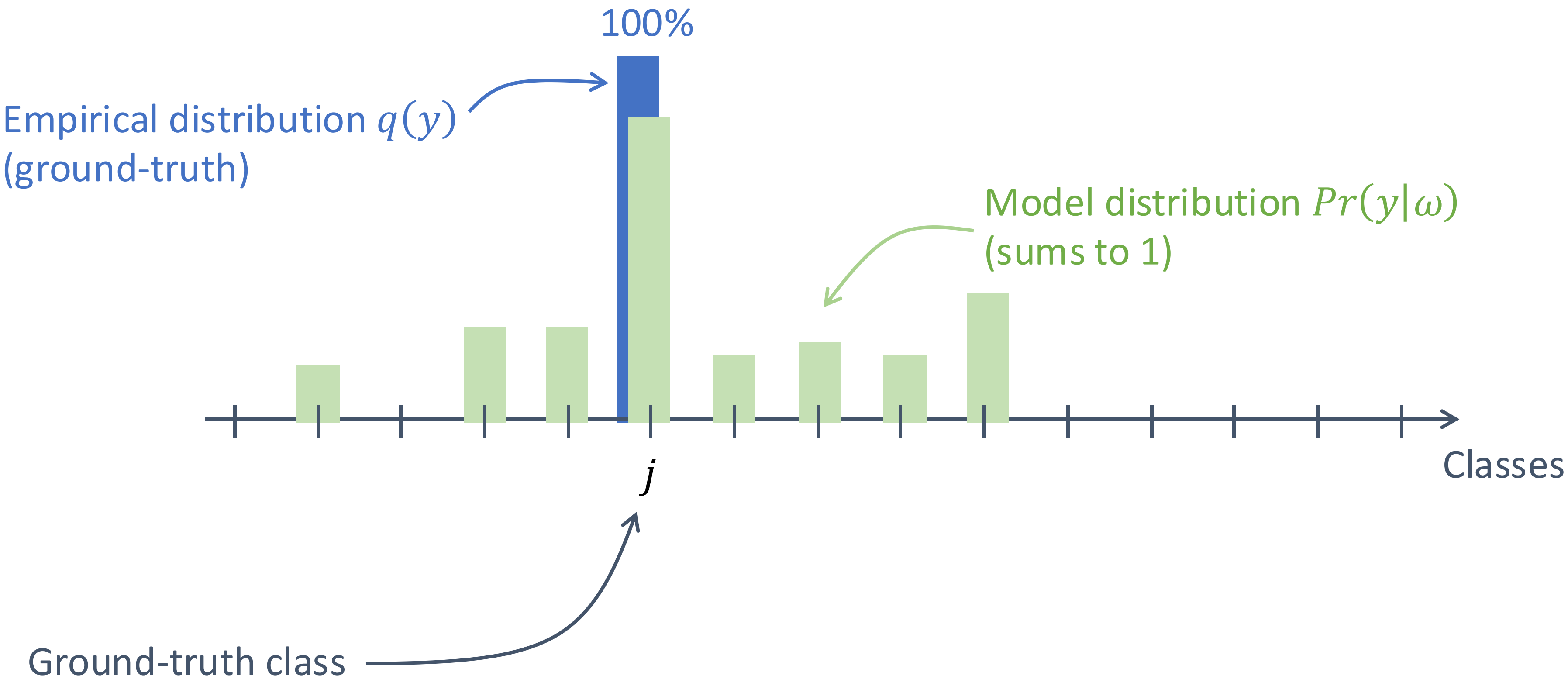
… and reuse it throughout this course!

Let's venture into the variations of a deep networks

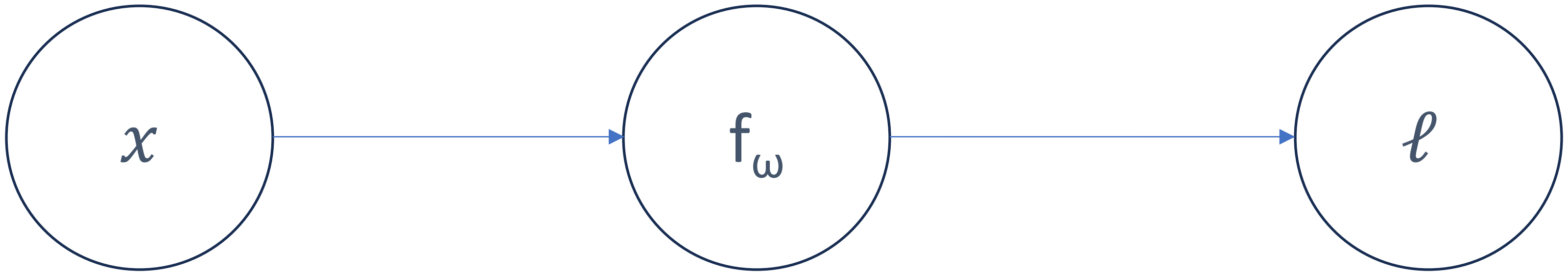
Fundamentally, the loss function expresses the distance between two distributions:

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

Loss functions: example for classification



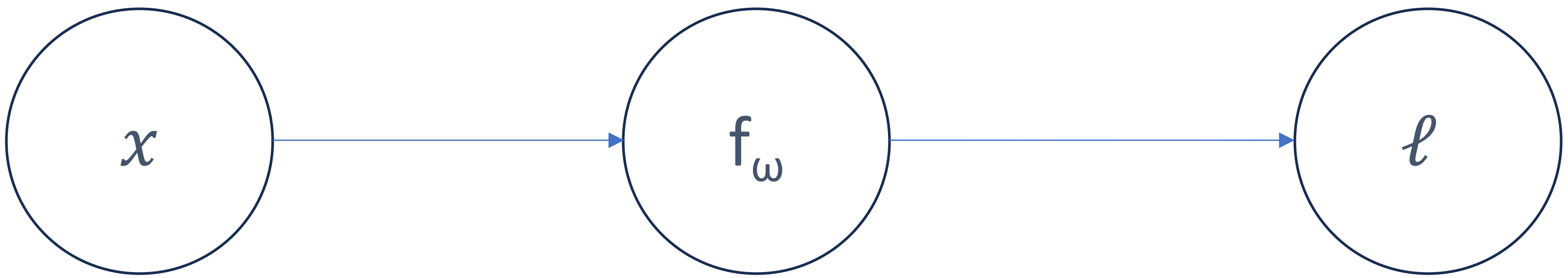
Loss functions



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

ground-truth

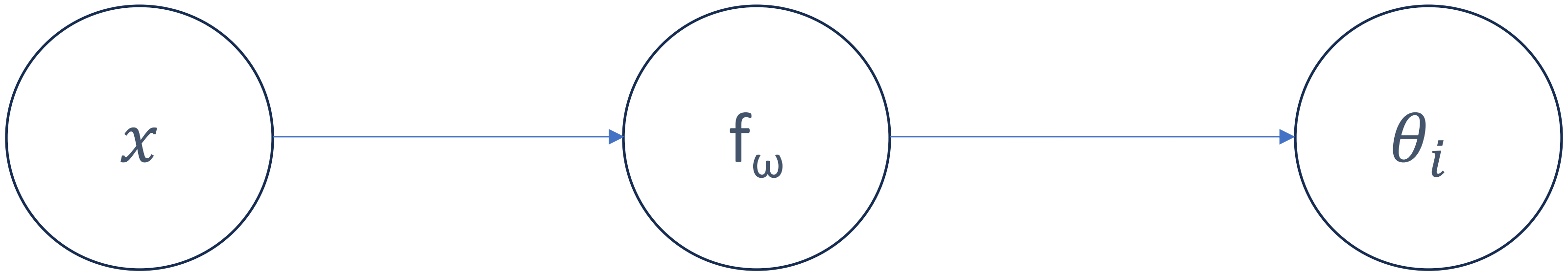
Loss functions



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

Expressed as the distance between two distributions

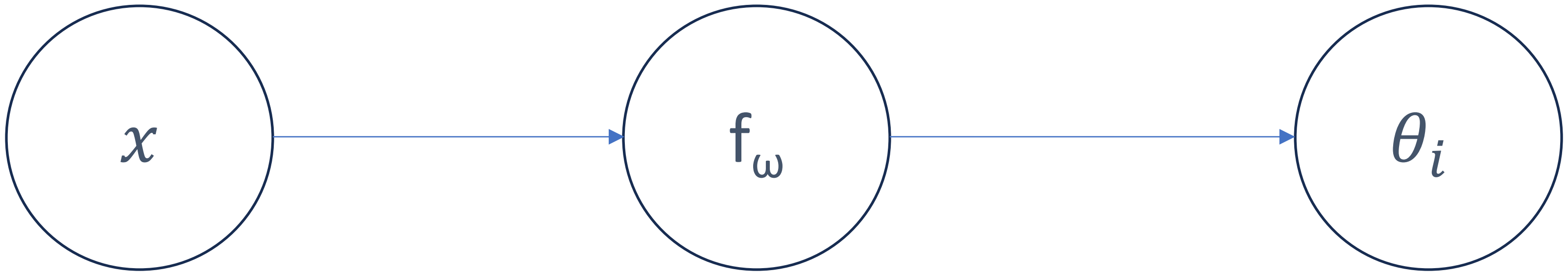
Loss functions



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

Parameters of a distribution

Loss functions



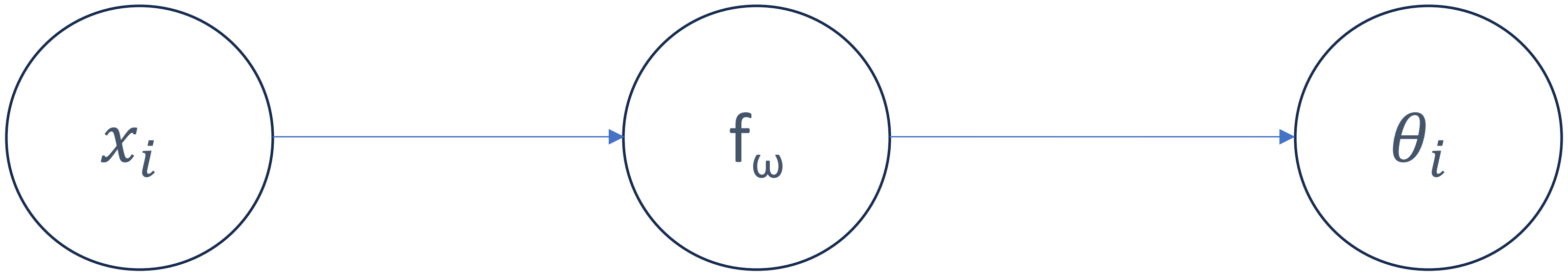
$$f_\omega(x_i) = \theta_i$$

The distribution is chosen based on the domain.

The model computes the optimal θ_i given the data.

Loss functions

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



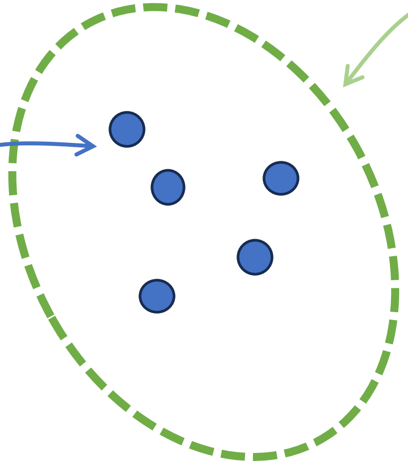
Example: univariate regression

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

θ

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

Given example data points



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

Loss functions

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

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

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

Loss functions

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

$\Pr(y_1, y_2, \ldots, y_N | x_1, x_2, \ldots, x_N)$

Data is assumed i.i.d

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

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

Loss functions

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

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

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

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

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

Negative log likelihood (NLL)

Inference

Given new input data

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

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.

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

Negative log likelihood (NLL)

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

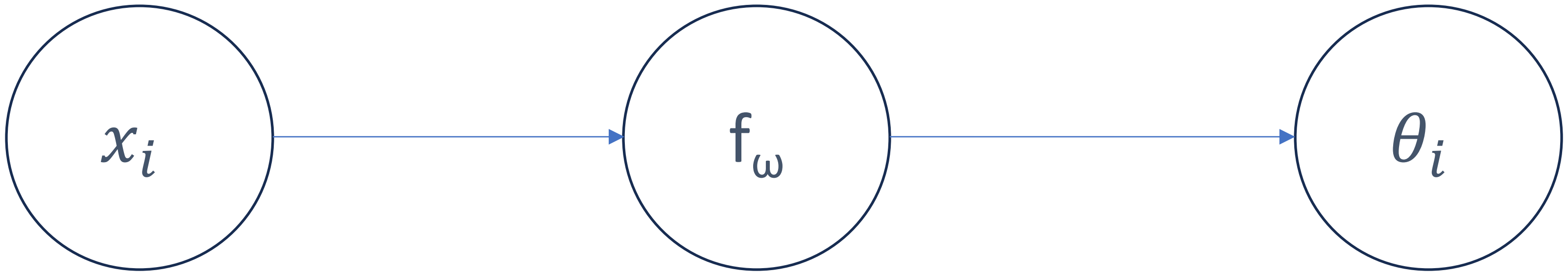
$$\hat{\omega} = \underset{\omega}{\mathrm{argmin}}\left[-\sum_{i=1}^{N}\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y_i-f_\omega(x_i))^2}{2\sigma^2}}\right]\right]$$

$$\hat{\omega} = \underset{\omega}{\mathrm{argmin}}\left[\sum_{i=1}^{N}(y_i - f_\omega(x_i))^2\right]$$

least squares

Loss functions

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



Example: binary classification

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

θ

Loss functions

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

$$x_i \longrightarrow f_\omega \longrightarrow \theta_i$$

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

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

$$x_i \longrightarrow \mathrm{f}_\omega \longrightarrow \theta_i$$

Example: multiclass classification

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

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

Loss functions

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

$$x_i \longrightarrow f_\omega \longrightarrow \theta_i$$

Example: multiclass classification

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

NLL

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

Loss functions

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

$$x_i \longrightarrow f_\omega \longrightarrow \theta_i$$

Example: multiclass classification

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

Wait, can we differentiate softmax?

NLL

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

Yes, we can!

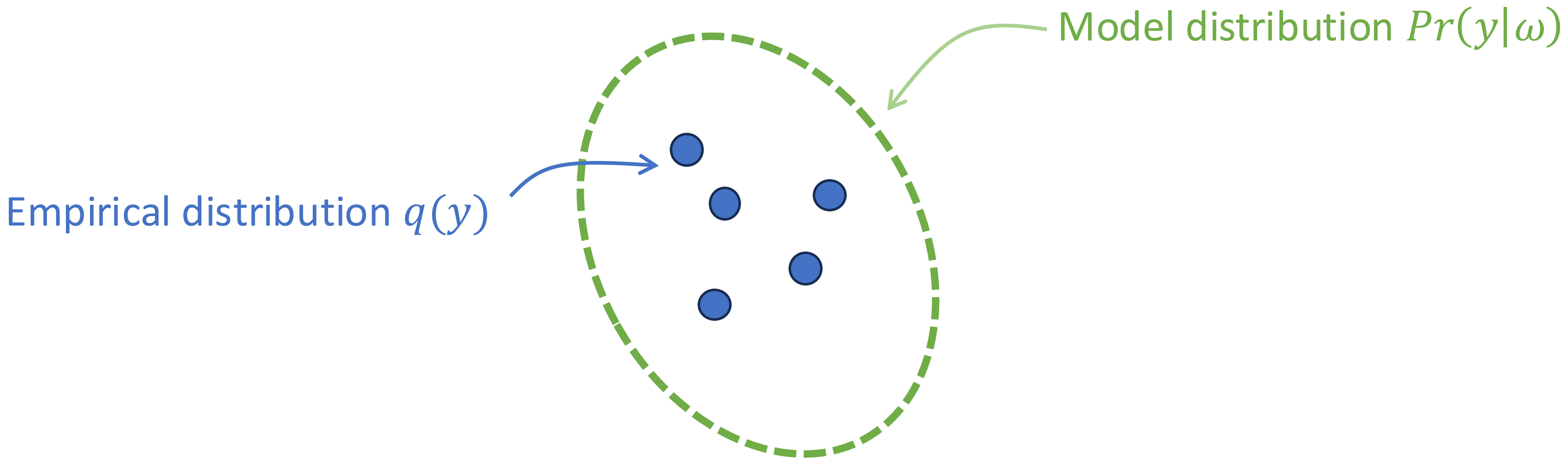
Loss functions

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

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:

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

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

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

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

Loss functions

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

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

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

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

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

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

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

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

cross-entry loss

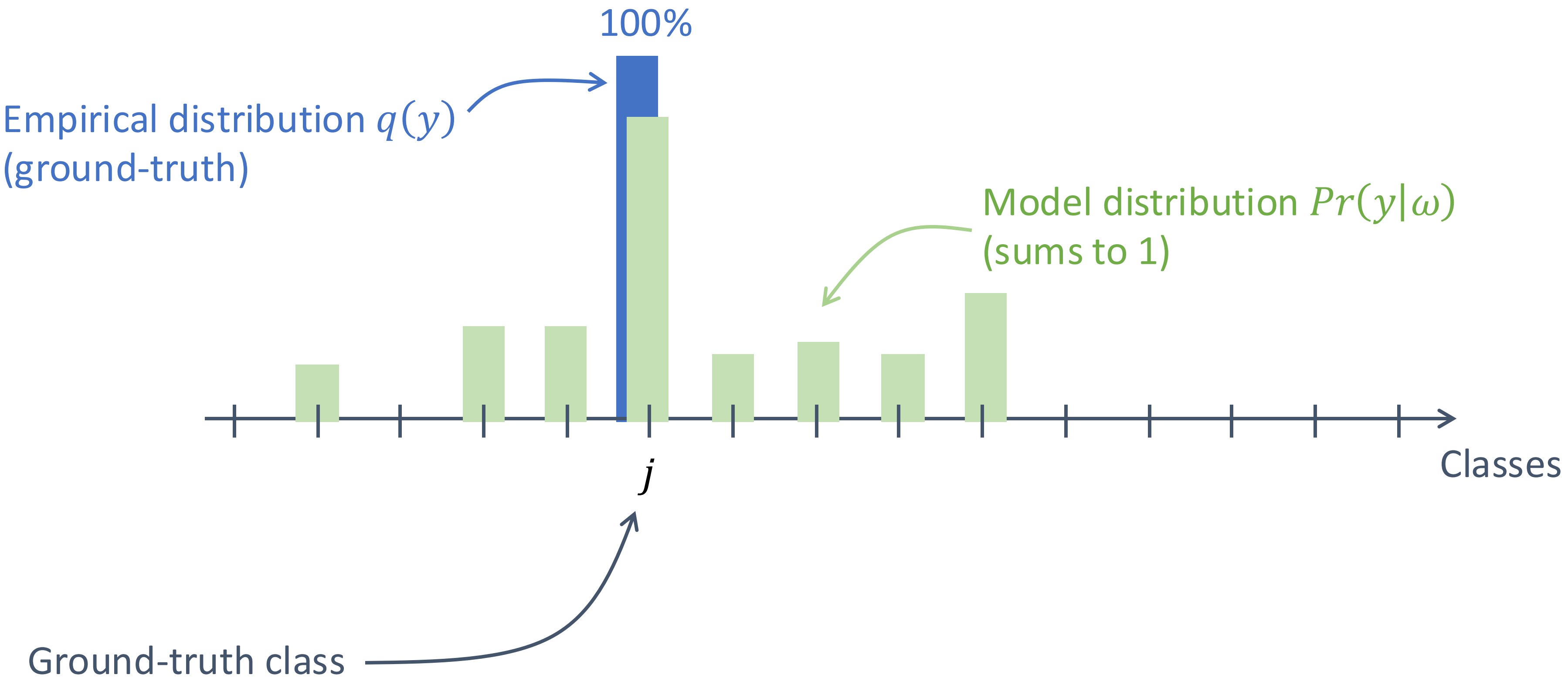
$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}}\left[-\iint_{-\infty}^{\infty}\left(\frac{1}{N}\sum_{i=1}^{N}\delta[y-y_i]\right)\log[\Pr(y|\omega)]dy\right]$$

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

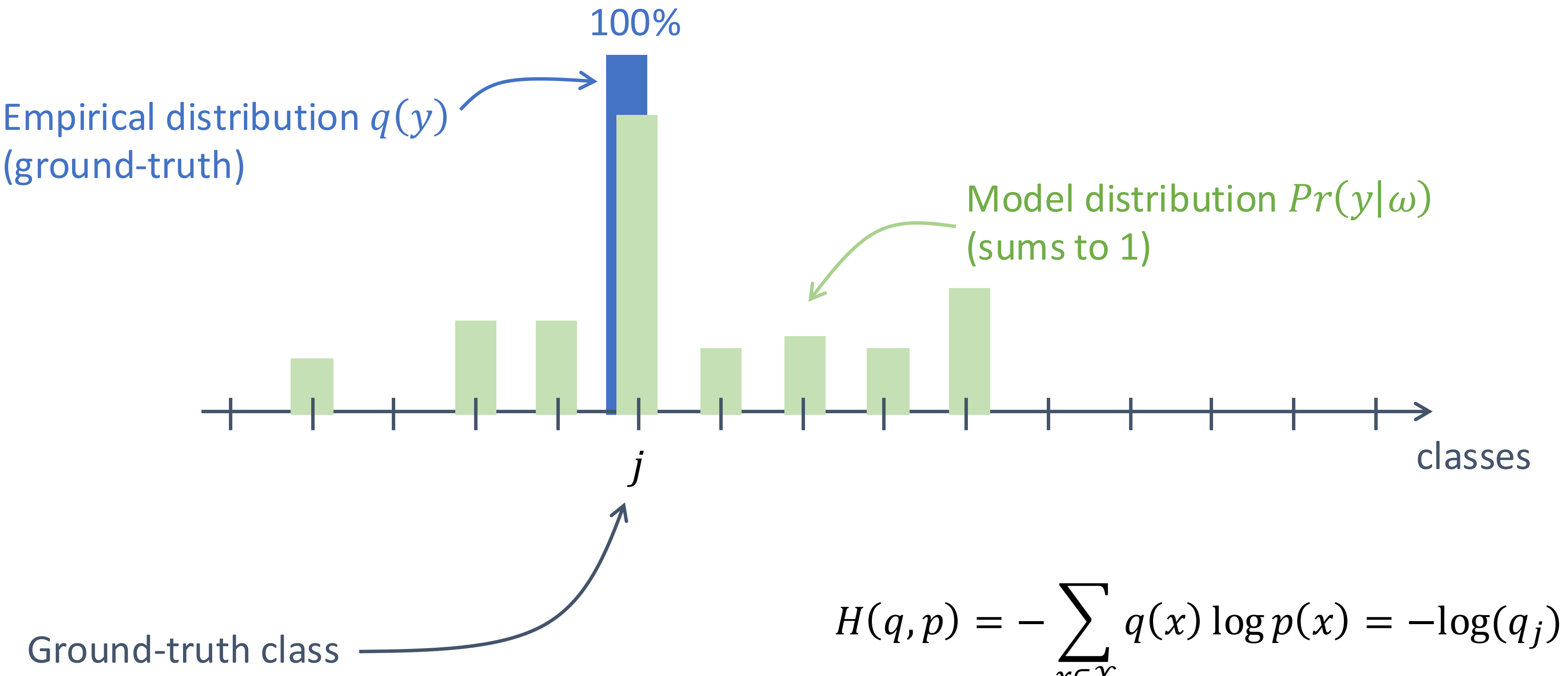
$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}}\left[-\sum_{i=1}^{N}\log[Pr(y_i|\omega)]\right]$$

NLL

Loss functions: example for classification

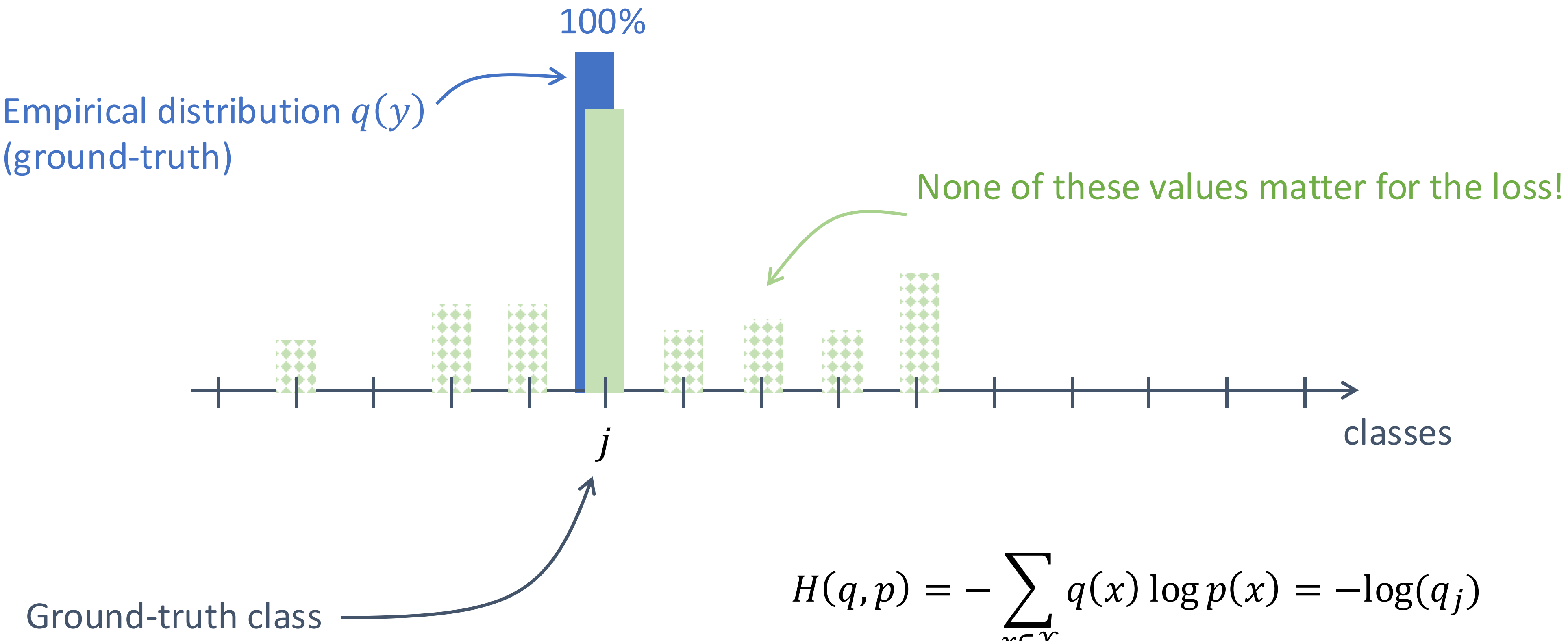


Loss functions: example for classification



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

Loss functions: example for classification



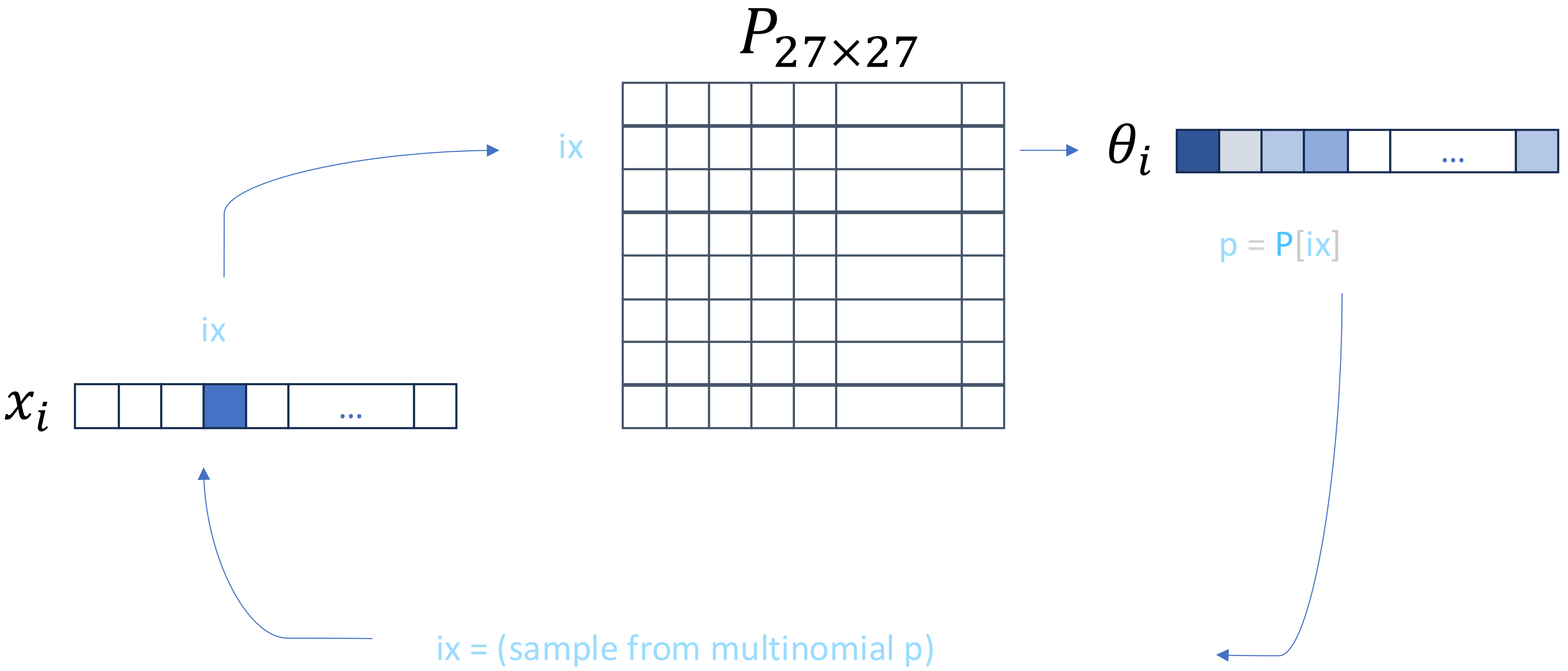
$$H(q,p) = -\sum_{x \in \mathcal{X}} q(x) \log p(x) = -\log(q_j)$$

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”

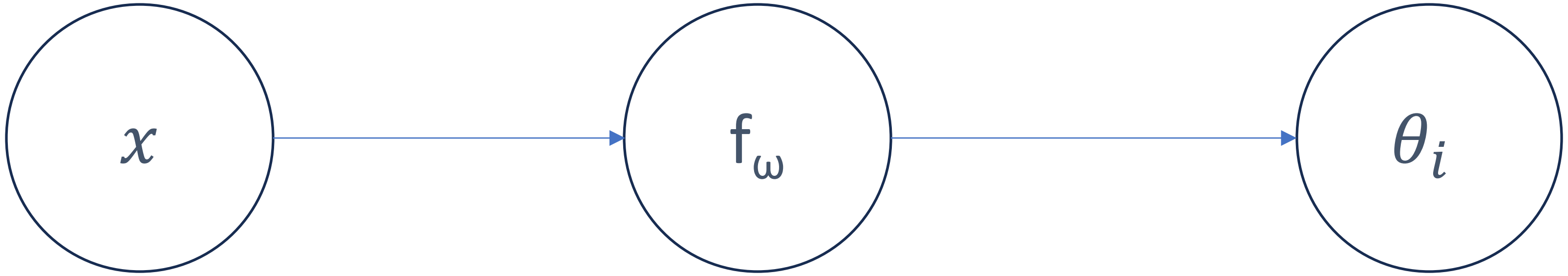
The dot character (.) marks the beginning and end of a word.

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

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



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

One-hot encoding of letter 'd'

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

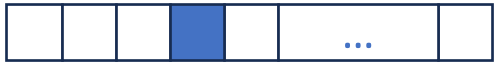
$\theta_i = softmax(x \cdot W)$ is a $N \times 27$ vector

representing the distribution of the next

character for each sample

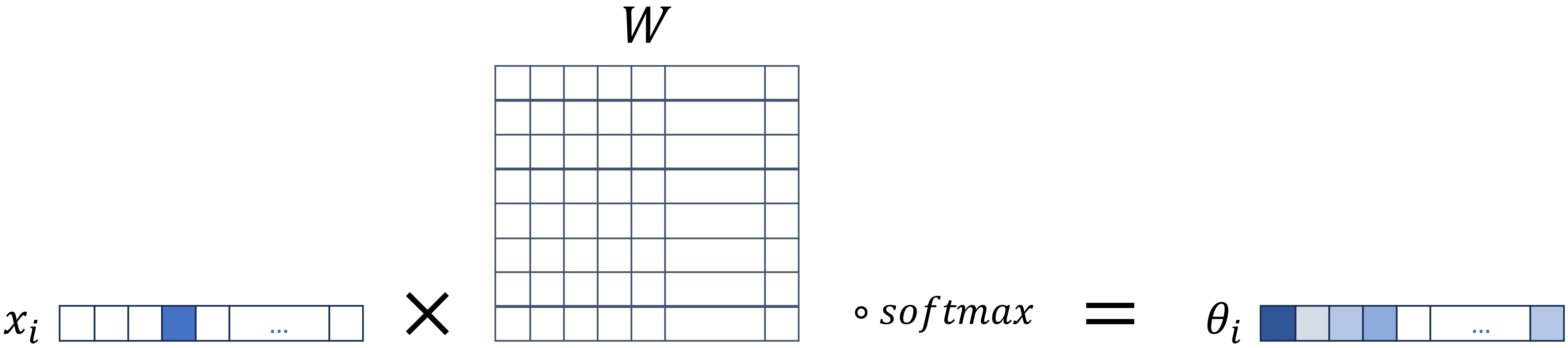
Step 2: bigram as a learnable matrix

x_i



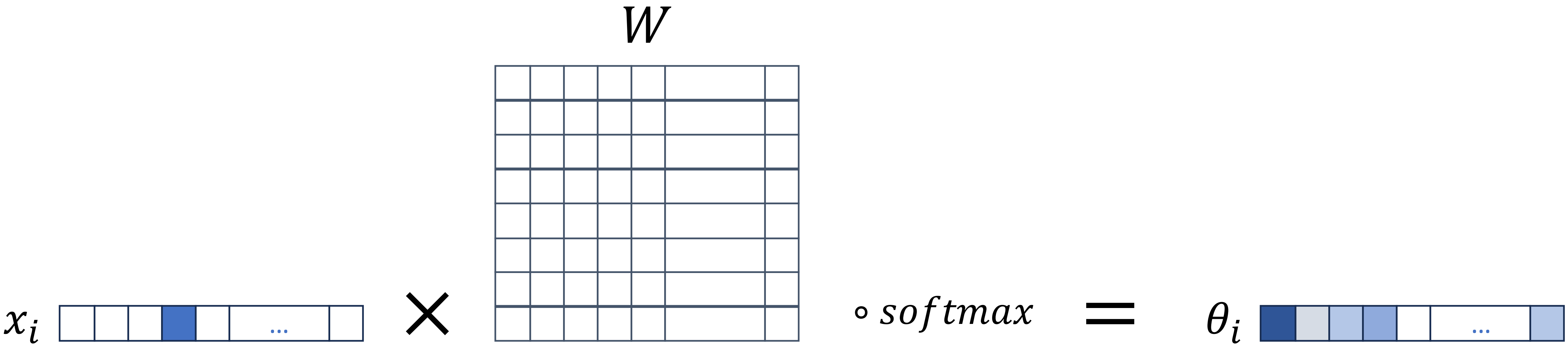
One-hot encoding of letter 'd'

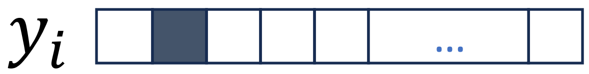
Step 2: bigram as a learnable matrix



One-hot encoding of letter 'd'

Step 2: bigram as a learnable matrix

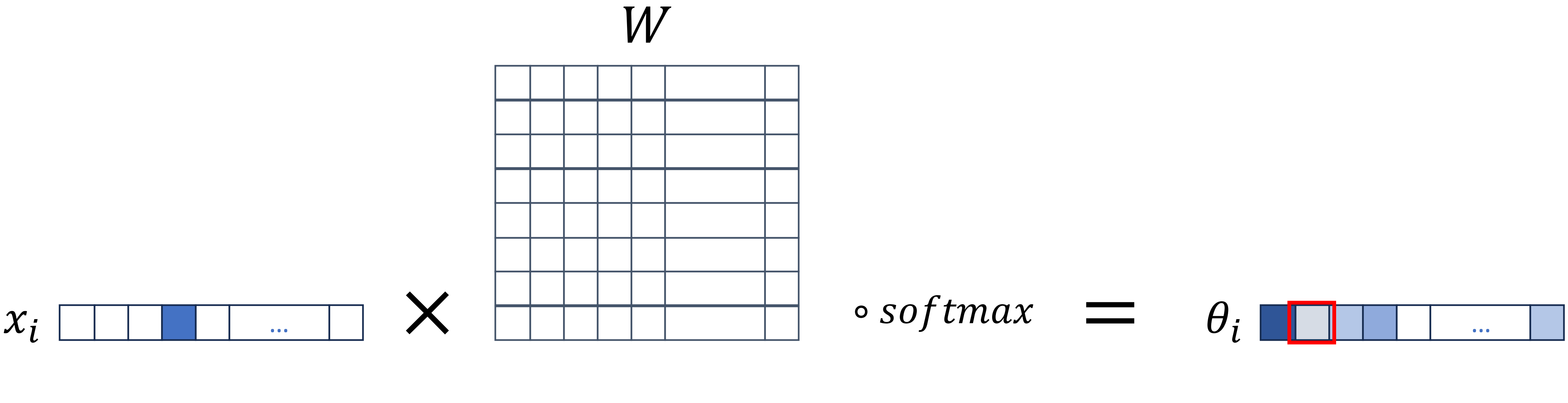




We want to minimize the KL divergence or NLL

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

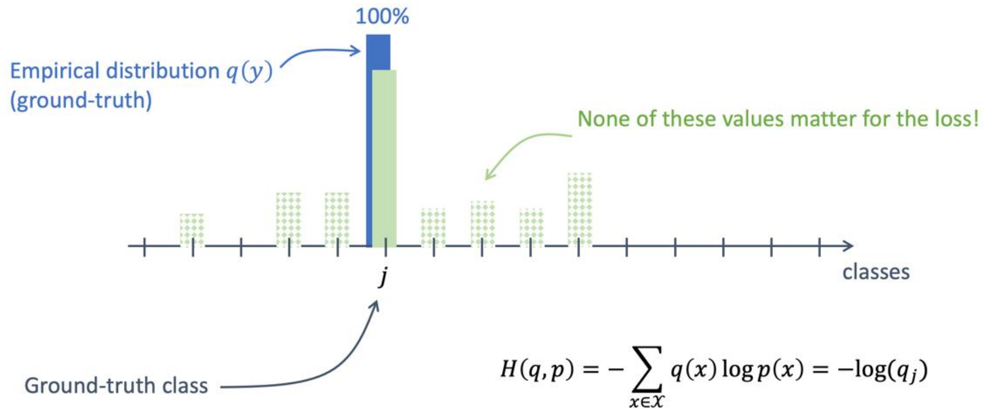
Step 2: bigram as a learnable matrix

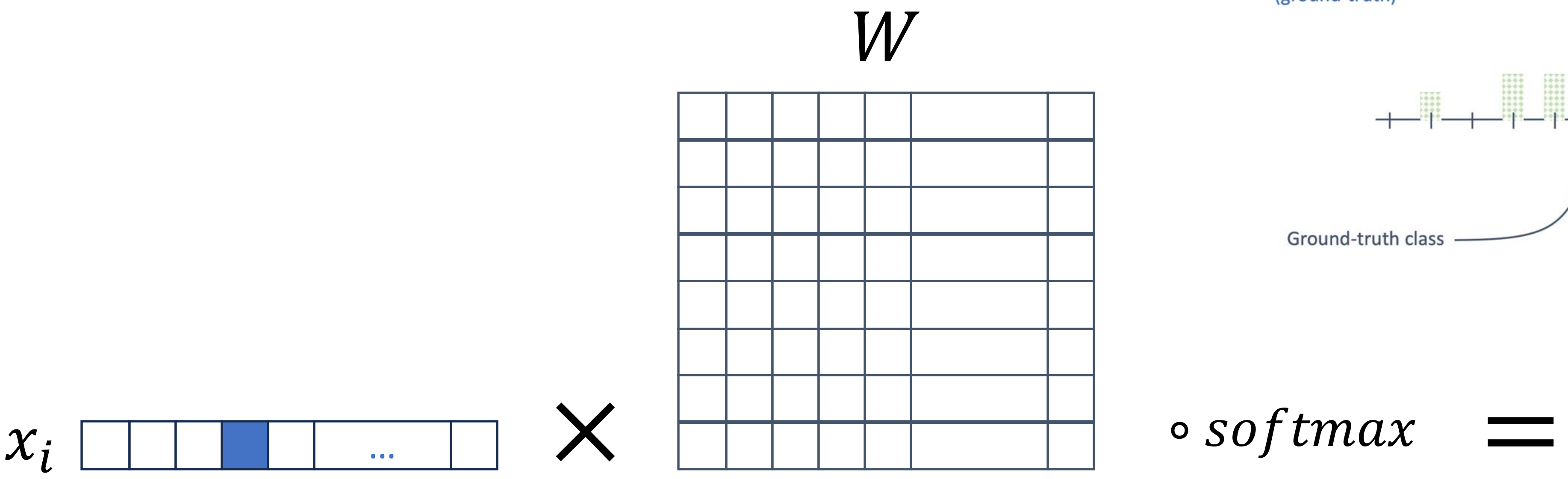


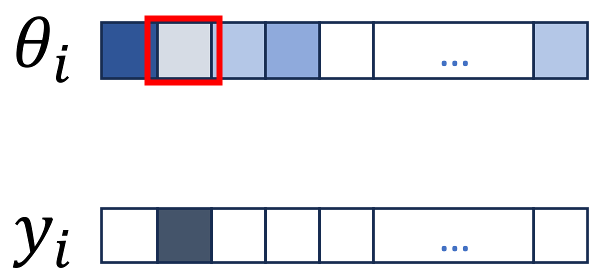
We want to minimize the KL divergence or NLL

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

Step 2: bigram as a learnable matrix





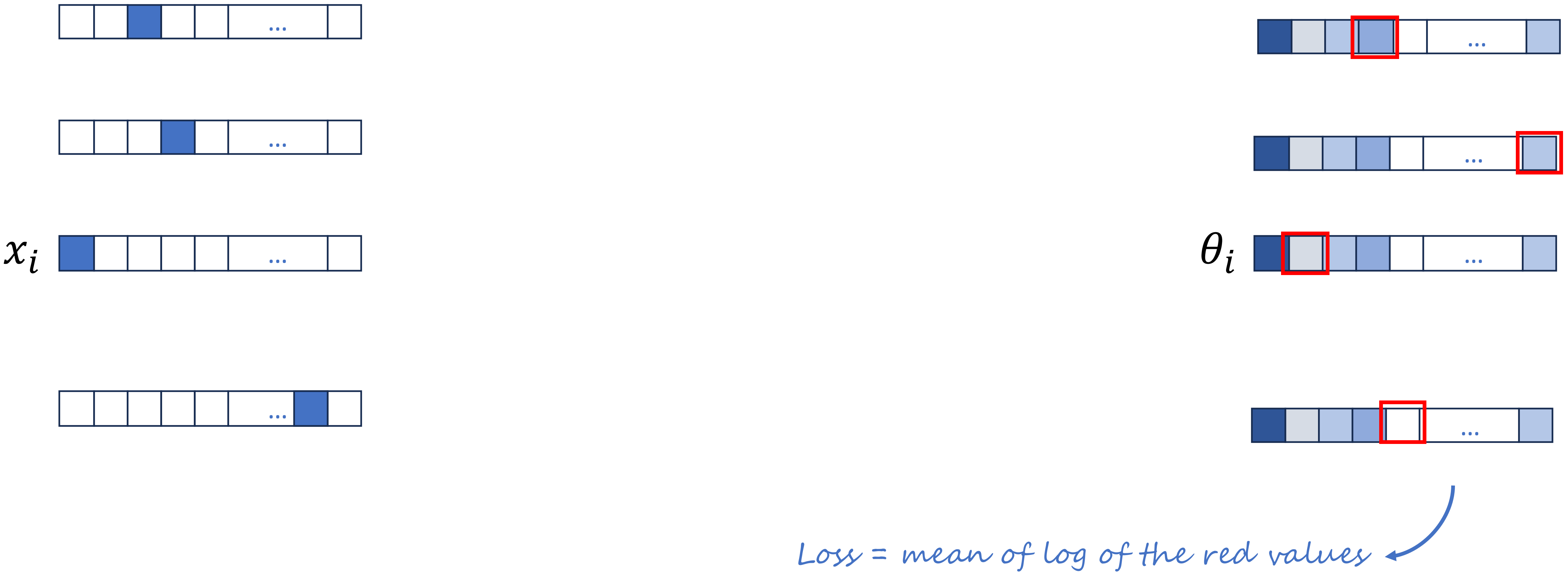


We want to minimize the KL divergence or NLL

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

NLL

Step 2: bigram as a learnable matrix



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 = torch.randn((7,7))
>>> a
tensor([[ 1.2555,  0.6821,  0.9131, -0.7238,  0.5636, -2.8689, -0.4744],
        [ 2.1393, -0.8737,  2.4039,  0.0056,  0.6169, -0.2245, -0.2242],
        [ 0.1821, -0.4250, -0.1115, -0.3568, -2.2182,  0.9574,  1.9415],
        [-0.2646,  1.7013, -2.7297,  0.3786, -1.7883,  0.8484, -0.1894],
        [-0.5430, -0.2352,  0.4820, -0.0737,  0.8632,  0.1648,  1.1864],
        [ 1.3596, -0.6411,  2.9097,  0.9422, -0.0167, -0.1453, -0.6059],
        [-0.4946,  0.2705,  0.5348, -1.8176, -1.3861, -1.0276, -1.0050]])
>>> a.sum(axis=1)
tensor([-0.6527,  3.8434, -0.0306, -2.0437,  1.8446,  3.8025, -4.9256])
>>> a.sum(axis=1, keepdim=True)
tensor([[-0.6527],
        [ 3.8434],
        [-0.0306],
        [-2.0437],
        [ 1.8446],
        [ 3.8025],
        [-4.9256]])
```

Step 3: A Neural Probabilistic Language Model

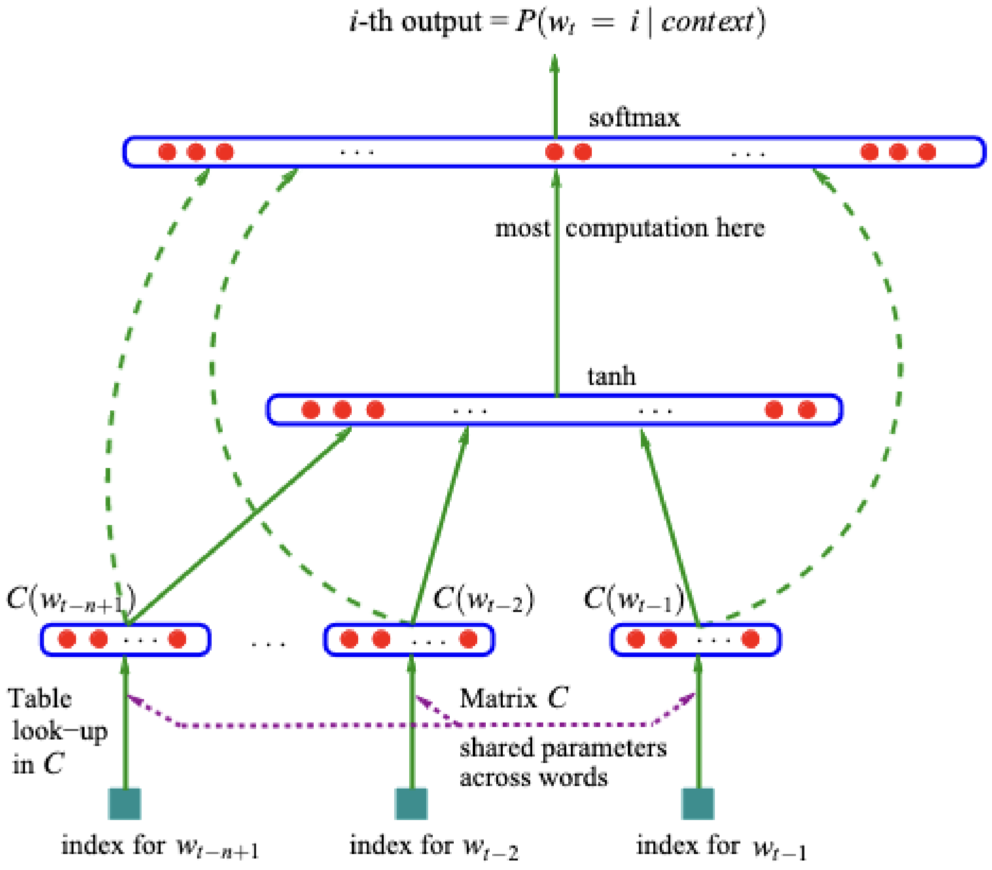


Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

A Neural Probabilistic Language Model, Bengio et al, 2003

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_{27x10}= dictionary

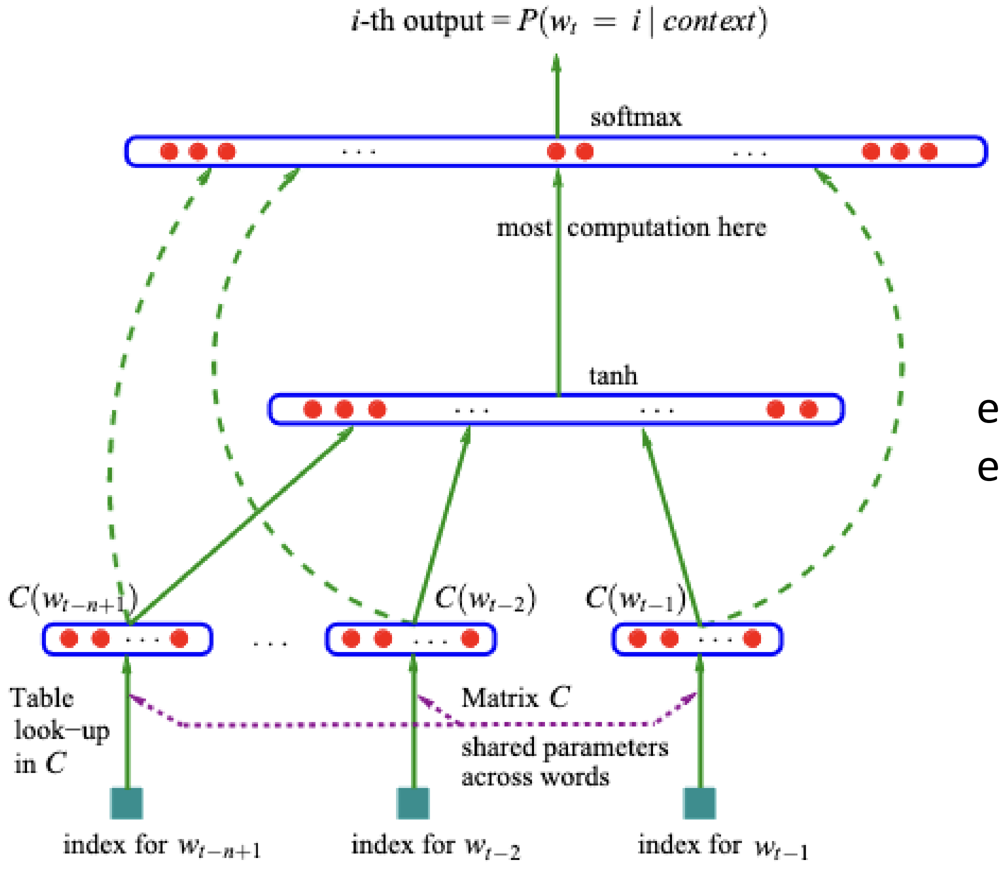


Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

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

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

emb = C[X_train[ix]]
emb = emb.view(...)

A Neural Probabilistic Language Model, Bengio et al, 2003