MNIST Digit Classification with Neural Net

Computing Science

University of Alberta

Nilanjan Ray

Agenda

- Building a neural net with PyTorch for MNIST digit recognition
- How does backpropagation work?
- Softmax function
- Cross-entropy loss

MNIST dataset

Classify images into digits

Each image is **28x28**

10 labels

55,000 training images

5,000 validation images

10,000 test images.

MNIST classification problem

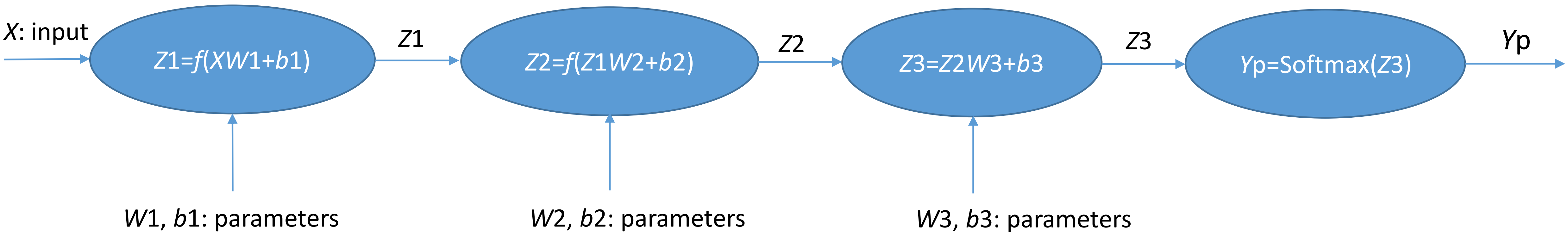
Small 28 pixels-by-28 pixels images of hand written digits

The visual recognition problem definition:
to recognize the digit from an image

Pixel values (feature) Digit: 1-hot vector

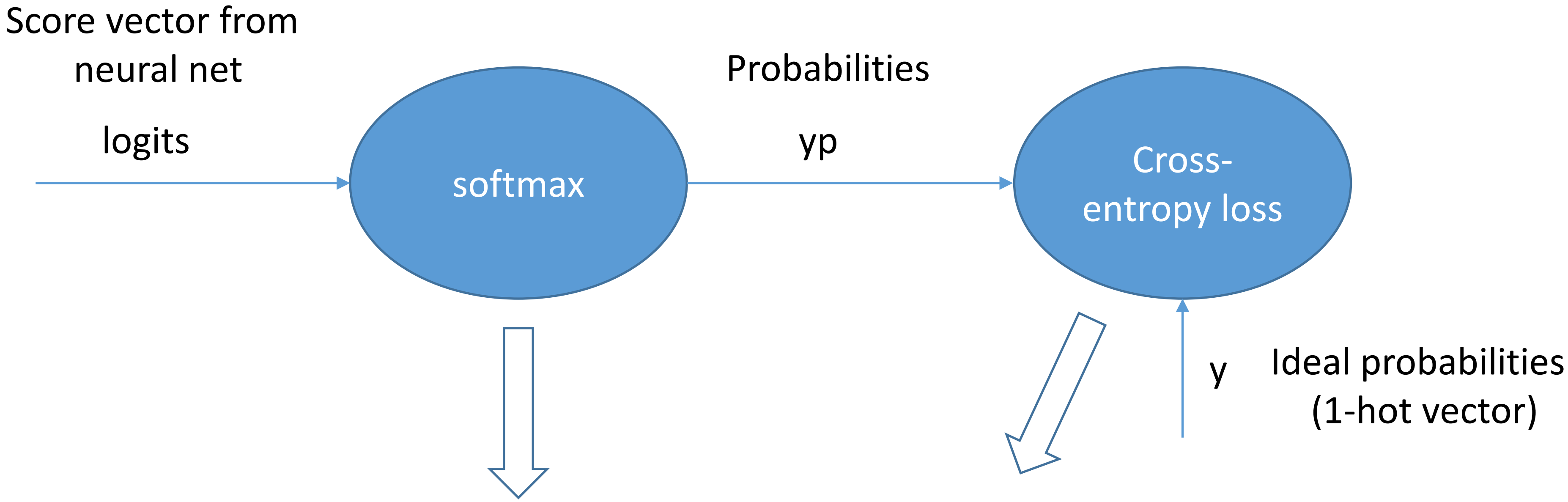
x_1	x_2	...	x_{784}	y_1	...	y_{10}
0.1	0.3	...	0.0	0	...	1
0.2	0.1	...	0.5	1		0
...	...	...	...	...	...	...
...	...	...	...	...	...	...
0.0	0.98	...	0.8	0	...	1
0.5	0.25	...	0.36	?	...	?
0.1	0.95	...	0.1	?	...	?

NN Architecture for MNIST Classification



Activation function, *f* is ReLU in our implementation

Softmax and cross-entropy loss

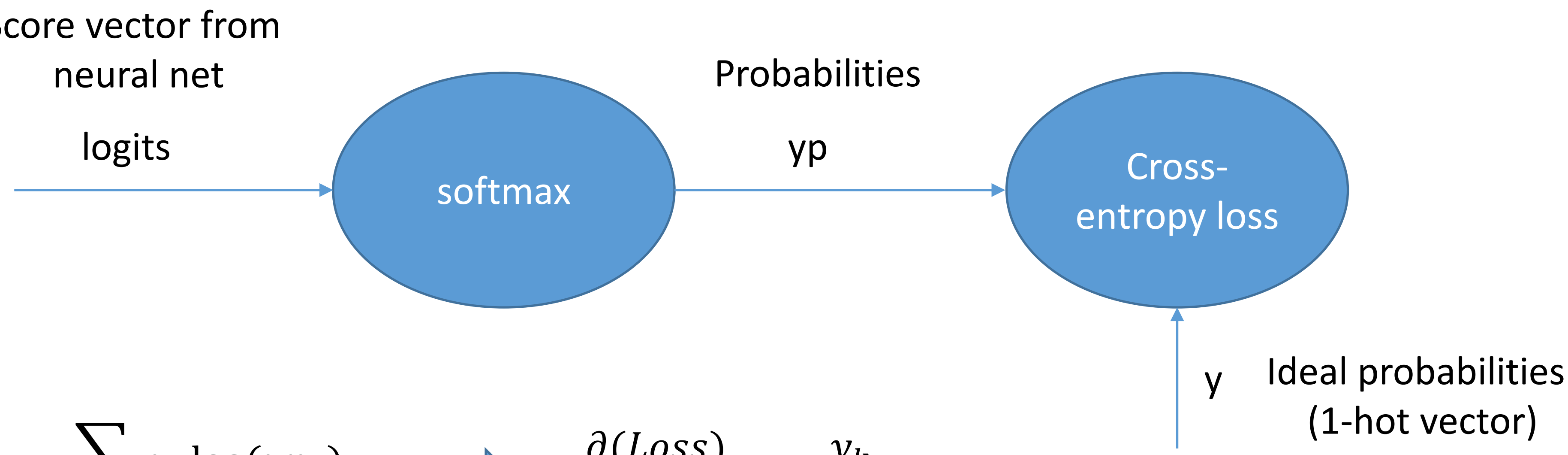


$$yp_i = \frac{\exp(logits_i)}{\sum_k \exp(logits_k)}$$

$$Loss = -\sum_k y_k \log(yp_k)$$

To backpropagate error, we need to compute: $\delta(logits)_i \equiv \frac{\partial(Loss)}{\partial(logits)_i}$

Softmax and cross-entropy loss: backprop



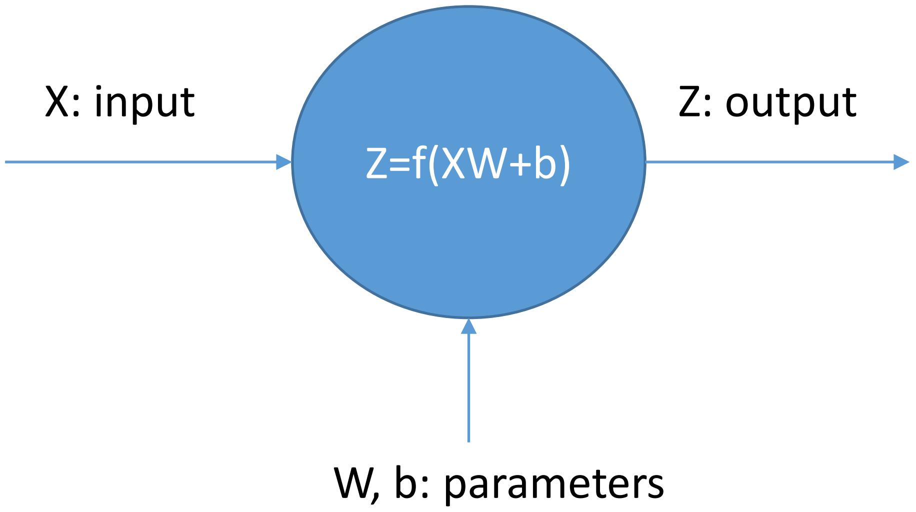
$$Loss = -\sum_k y_k \log(yp_k) \Rightarrow \frac{\partial(Loss)}{\partial(yp)_k} = -\frac{y_k}{yp_k}$$

$$yp_i = \frac{\exp(logits_i)}{\sum_k \exp(logits_k)} \Rightarrow \frac{\partial(yp)_k}{\partial(logits)_i} = \begin{cases} yp_i(1 - yp_i), & \text{if } i = k, \\ -yp_i yp_k, & \text{otherwise.} \end{cases}$$

Using the above two results in the chain rule, $\delta(logits)_i \equiv \frac{\partial(Loss)}{\partial(logits)_i} = \sum_k \frac{\partial(yp)_k}{\partial(logits)_i} \frac{\partial(Loss)}{\partial(yp)_k} = yp_i - y_i$

What if, instead of cross-entropy, we used L2 loss along with softmax?

Backprop across a neural net layer



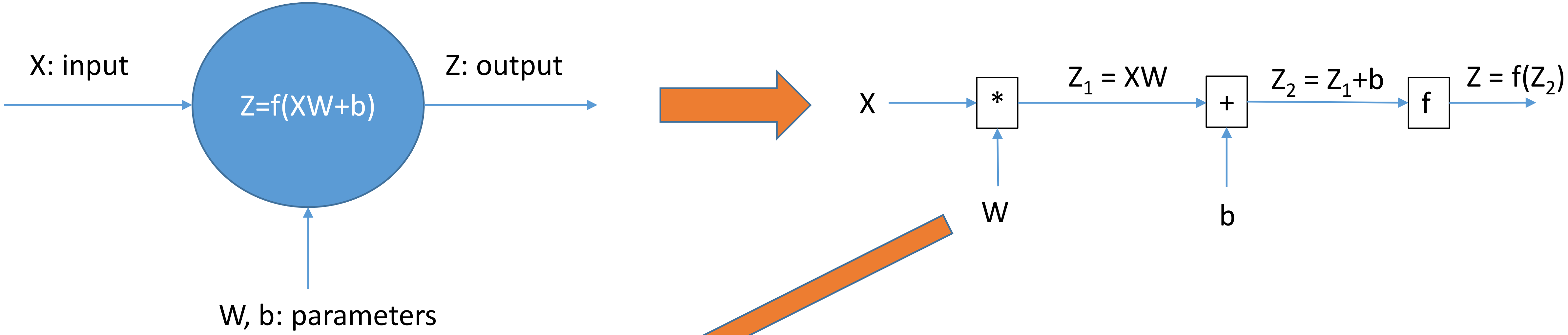
Pointwise multiplication

Backprop rules:

$$\delta X = [f'(XW+b) \cdot \delta Z]W^T \qquad \delta W = X^T[f'(XW+b) \cdot \delta Z] \qquad \delta b = \sum_l [f'(XW+b) \cdot \delta Z]_{l,:}$$

Matrix multiplication

Backprop across a neural net layer: derivation

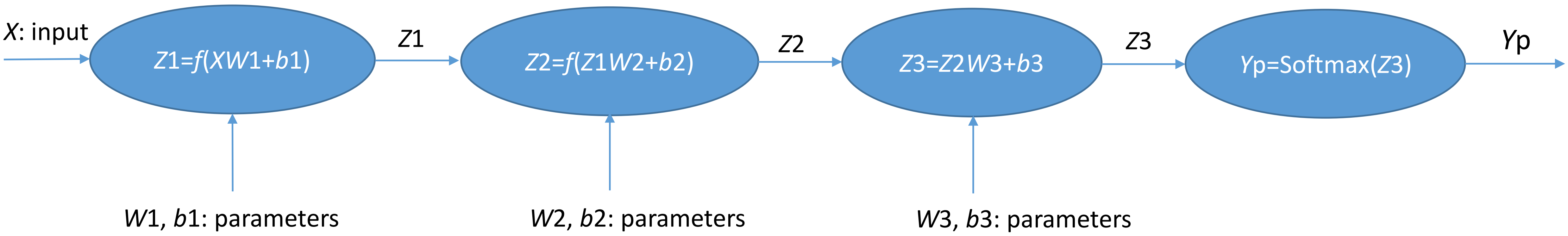


Apply chain rule to computational nodes:

$$\left.\begin{aligned} \delta Z_2 &= f'(Z_2)\delta Z \\ \delta Z_1 &= \delta Z_2 \\ \delta X &= \delta Z_1 W^T \end{aligned}\right\} \Rightarrow \delta X = [f'(XW + b) \cdot \delta Z]W^T$$

Similarly, we can derive backprop rules for δW and δb

NN for MNIST Classification: Gradients



Backprop:

$\delta Z3 = Yp - Y$

$\delta Z2 = (\delta Z3) W3^T$

$\delta Z1 = [f'(Z1W2+b2) . \delta Z2] W2^T$

$$\delta W3 = (Z2^T)\delta Z3$$

$$\delta W2 = Z1^T[f'(Z1W2 + b2) \cdot \delta Z2]$$

$$\delta W1 = X^T[f'(XW1 + b1) \cdot \delta Z1]$$

$$\delta b3 = \sum_l [\delta Z3]_{l,:}$$

$$\delta b2 = \sum_l [f'(Z1W2 + b2) \cdot \delta Z2]_{l,:}$$

$$\delta b1 = \sum_l [f'(XW1 + b1) \cdot \delta Z1]_{l,:}$$

Learning MNIST NN with Backprop and SGD

Initialize all parameters of the neural network
Initialize learning rate variable *lr*

Iterate:

(Load Data): Get training data batch X

(Forward pass): Compute $Z1$, $Z2$, Z3, Yp

(Backward pass): Compute gradients $\delta Z3$, $\delta Z2$, $\delta Z1$, $\delta W3$, $\delta W2$, $\delta W1$, $\delta b3$, $\delta b2$, $\delta b1$

(Gradient descent to update parameters): $W3 \leftarrow W3 - lr * \delta W3, \quad b3 \leftarrow b3 - lr * \delta b3, \ldots,$

(Diagnostics): Compute "Loss" from time to time to check if it is decreasing