Machine Learning.

Introduction to convolutional neural networks.
Example. LeNet Mathematica.

Definition. We define an order K tensor to be an ~~element~~ ~~vector~~ K-dimensional array of numbers $X \in \mathbb{R}^{d_1 \times \cdots \times d_K}$.

Examples.

Vectors are order 1 tensors $X \in \mathbb{R}^n$
Matrices are order 2 tensors $X \in \mathbb{R}^{m \times n}$

An RGBA image which is H pixels high and W pixels wide is represented by an order 3 tensor

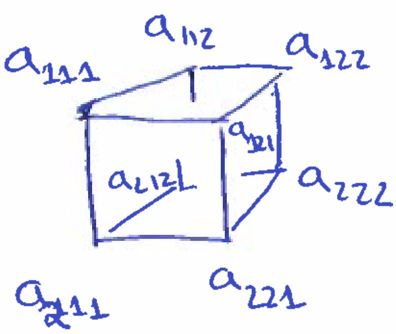
$$X \in \mathbb{R}^{H \times W \times 4}$$

The elements of an order K tensor are denoted with K indices: $X_{i_1 \cdots i_K} \in \mathbb{R}$

Definition. The operator vec reduces every tensor to a vector by ordering the entries in reverse lexicographic order.

Example

$$\mathrm{vec}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$

vec 

$$= \begin{bmatrix} a_{111} \\ a_{211} \\ a_{121} \\ a_{221} \\ a_{112} \\ a_{212} \\ a_{122} \\ a_{222} \end{bmatrix}$$

Recall.

Suppose $z : \mathbb{R}^m \to \mathbb{R}$, and we write $z = z(\vec{y})$.

Then

$$\frac{\partial z}{\partial \vec{y}^T} = Dz, \text{ the } \cancel{\text{matrix}} \overset{\text{1-tensor}}{} \text{ so that } \left[\frac{\partial z}{\partial \vec{y}}\right]_i = \frac{\partial z}{\partial y_i}$$

is a $1 \times m$ matrix of partial derivatives.

If $\vec{y} : \mathbb{R}^n \to \mathbb{R}^m$, and we write $\vec{y} = \vec{y}(\vec{x})$

and

$$\frac{\partial \vec{y}}{\partial \vec{x}^T} = D\vec{y}, \text{ the } \cancel{\text{matrix}} \overset{\text{2-tensor}}{} \text{ so } \left[\frac{\partial \vec{y}}{\partial \vec{x}^T}\right]_{ij} = \frac{\partial y_i}{\partial x_j}$$

Then we can write the chain rule as

$$\frac{\partial z}{\partial \vec{x}^T} = \frac{\partial z}{\partial \vec{y}^T} \frac{\partial \vec{y}}{\partial \vec{x}^T} \quad \text{or} \quad Dz\, Dy$$

$1 \times n$ row vector

row vector $1 \times m$

$m \times n$ matrix

Definition. A CNN is a composition of maps (called <u>layers</u>) defined by tensors $\omega^1_*, \dots, \omega^L_*$ which we denote $\boxed{\omega_i}$. Each map takes tensors to tensors so that the composition

$$X^1 \to \boxed{\omega^1} \to X^2 \to \boxed{\omega^2} \to \cdots \to X^L \to \boxed{\omega^L} \to z$$

is defined.

To classify images, ~~we~~ into C categories,

X^1 = input image and X^L = output prediction

order 3, $H \times W \times 3$ — order 1, C

X^1_{ijk} = color k of pixel i, j

X^L_i = probability that image is category i

The last layer is special. If we know the classification of image X^1 is category i, then (for instance)

$$z = \frac{1}{2} \| \vec{e}_i - X_L \|^2$$

The choice of function here is part of the model (called a loss function).

Given the parameters $\omega^1, \ldots, \omega^{L-1}$, the CNN prediction ~~is given by~~ for image X^1 is given by ~~~~ $\underset{i \in 1,\ldots,C}{\operatorname{argmax}} X_i^L = i \mid X_i^L \geq X_j^L$ for all $j \in 1, \ldots, C$

Idea. Given a "training set" of N images, the total loss is a function z of $\omega^1, \ldots, \omega^{L-1}$.

~~~~ We want to minimize

$$z(\omega^1, \ldots, \omega^{L-1})$$
~~~~

Note: To define the loss layer, we need to know the truth for each image in the training set.

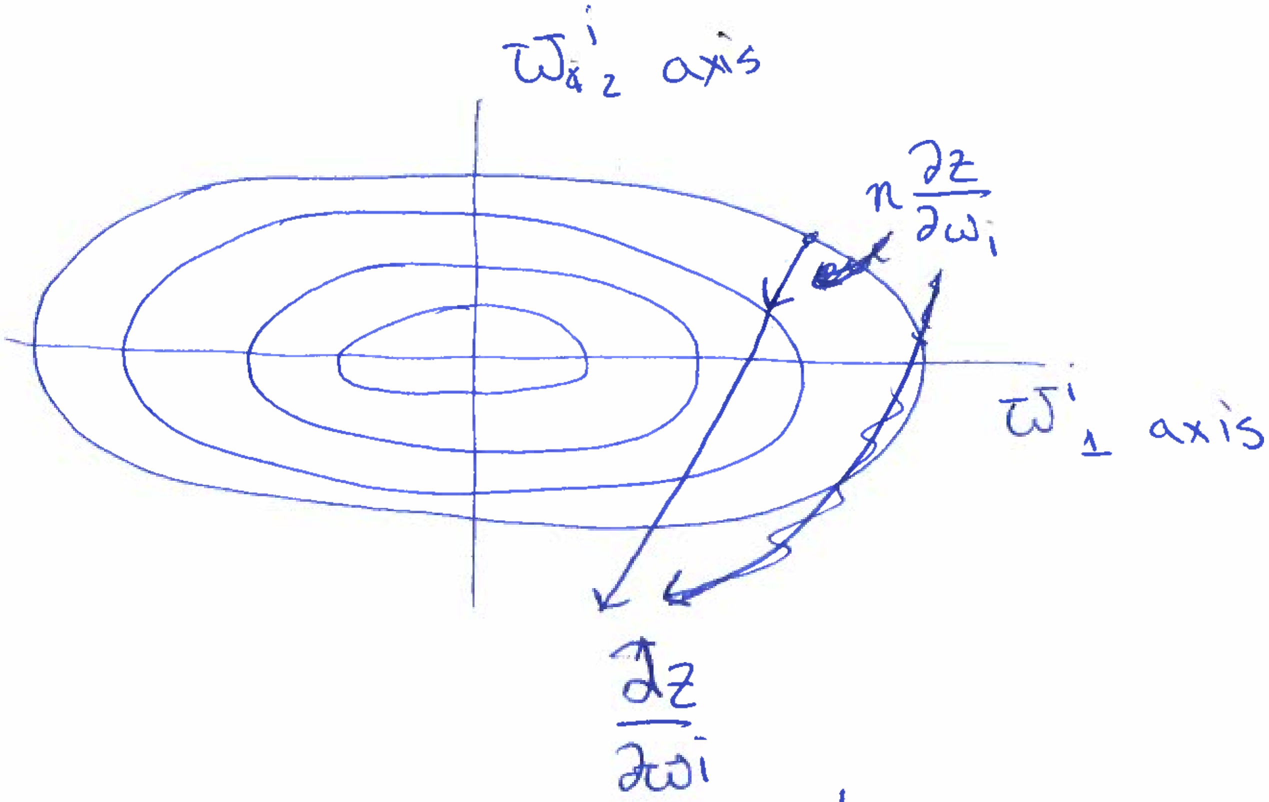
Note 2: The $\varpi^1, \ldots, \varpi^{L-1}$ are a <u>lot</u> of variables, but in theory we really just want to solve

$$\frac{\partial z}{\partial(\varpi^1, \ldots, \varpi^{L-1})} = \vec{0}$$

* We can update ϖ^i by <u>steepest descent</u> by the rule

$$\varpi^i \longleftarrow \varpi^i - \eta \underbrace{\frac{\partial z}{\partial \varpi^i}}_{\text{a tensor of the same dimensions as } \varpi^i}$$

a small number called the learning rate.



Problem. If we minimize loss on a single example, we will get that example right but others wrong. If we minimize on <u>all</u> examples, # computations is large.

Solution. Choose K ~~32 or 64~~ examples randomly, let X^1 be $H \times W \times 3 \times K$ and compute a gradient step for $w^1, \dots, w^{L-1}$, then ~~choose &~~ repeat.

But how do we compute the $\frac{\partial z}{\partial \varpi^i}$?

Back propagation. For every layer, compute

$$\frac{\partial z}{\partial \mathbf{x}^i} \quad \text{and} \quad \frac{\partial z}{\partial \varpi^i}$$

Start with the last layer $\boxed{\varpi^L}$ which has no parameters, so

$$\frac{\partial z}{\partial \varpi^L} = 0, \quad \frac{\partial z}{\partial (\mathbf{x}^L)} = \mathbf{x}^L - t$$

truth.