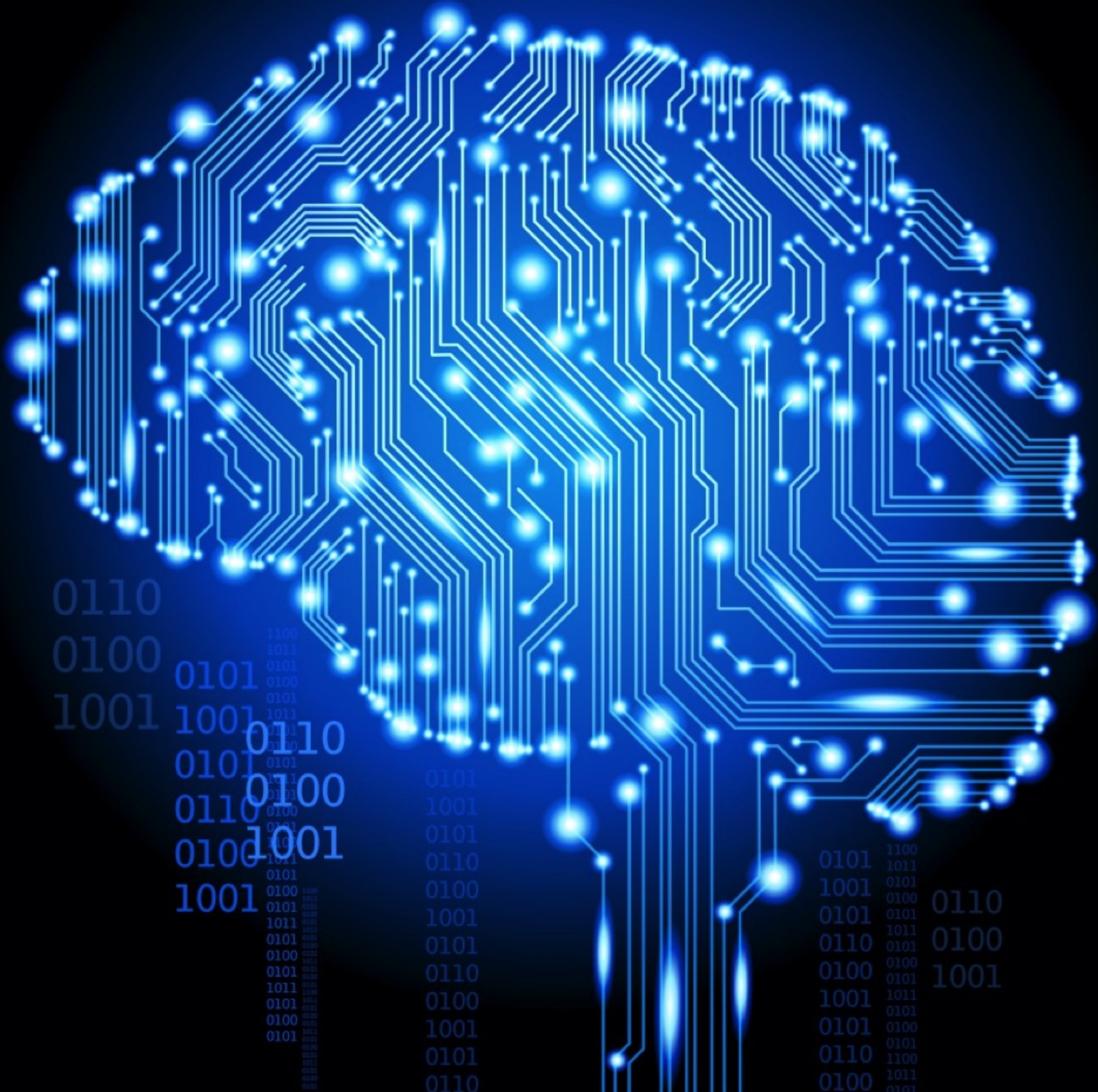
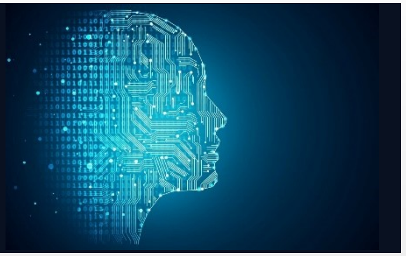
DEEP LEARNING
Part II

Leeor Langer



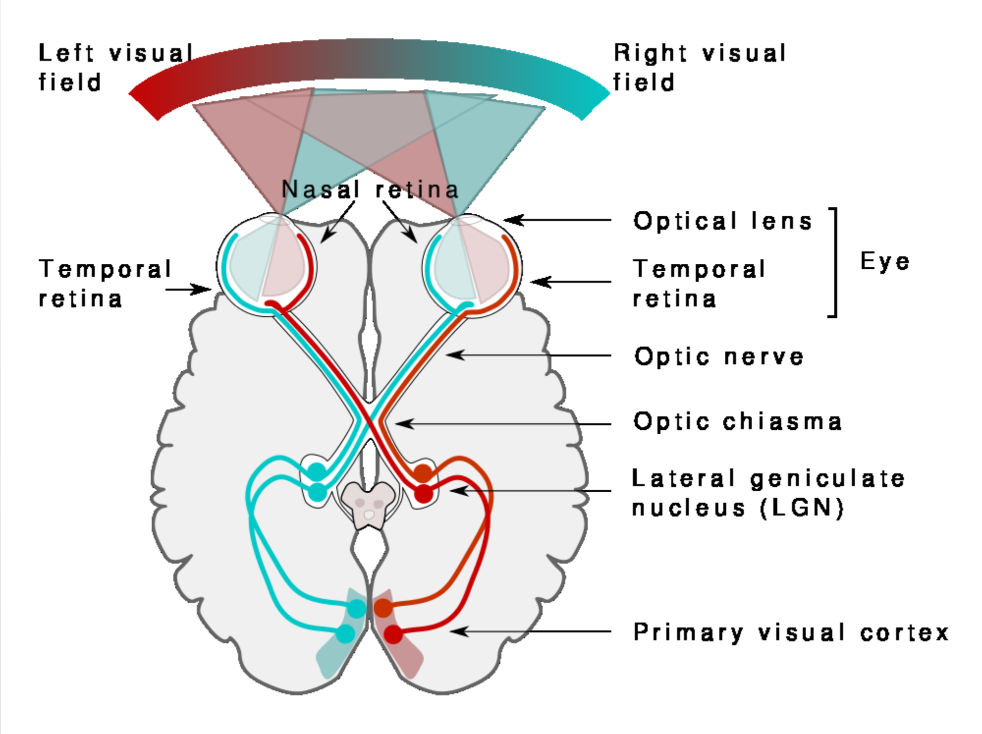
AGENDA

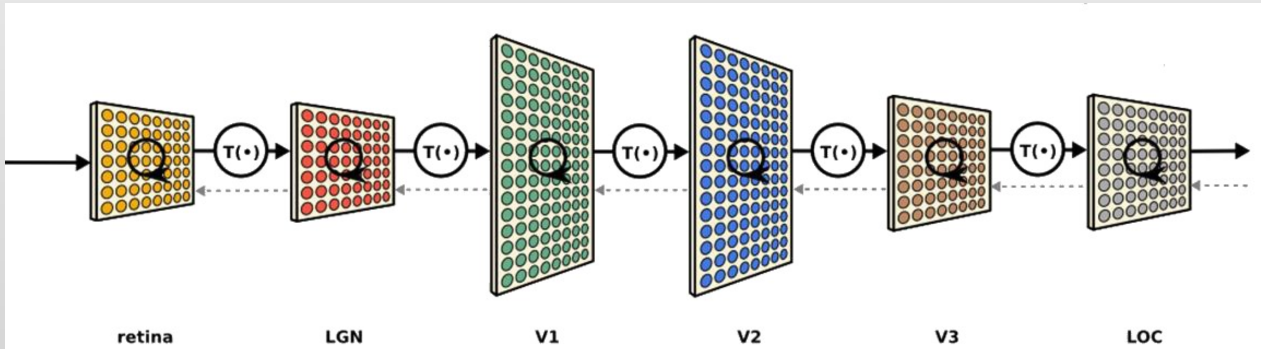


1. Intro: Convolutional Neural Networks
2. Cutting edge Neural Networks in Vision
3. End-to-end training
4. Cutting edge Neural Networks in Audio
5. How to structure a deep learning project for production

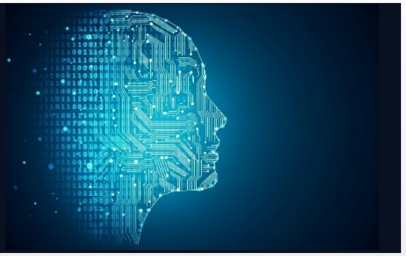
TECHNOLOGY TRENDS – COMPUTER VISION

Convolutional Neural Networks, inspired by the human mind:





AI TIMELINE – CONVOLUTIONAL NEURAL NETS



1960s
- Receptive Fields(Huber, Wiesel)

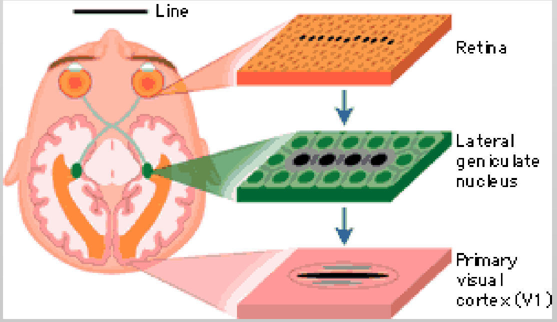
1970s
- Sobel Operator (Sobel, Feldman)

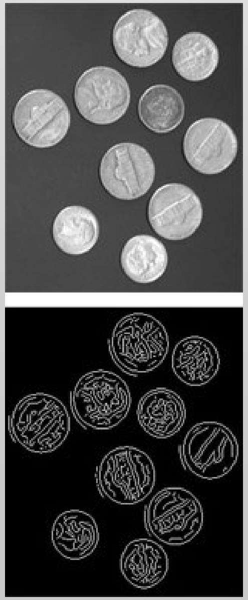
1980s
- Neocognitron (Fukushima)

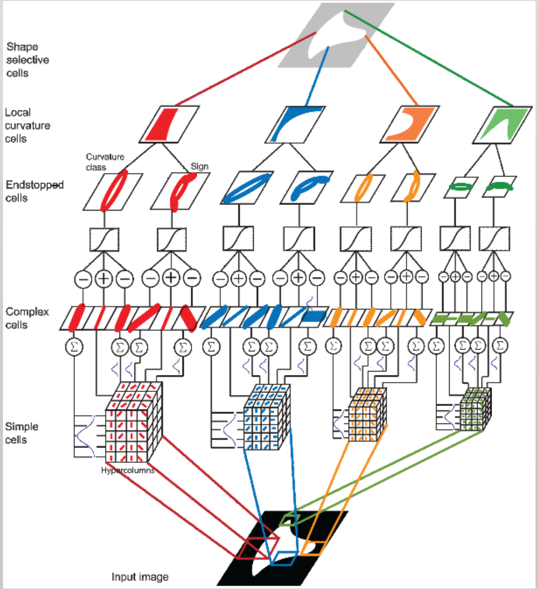
1990s
- LeNet-5 (Lacun)

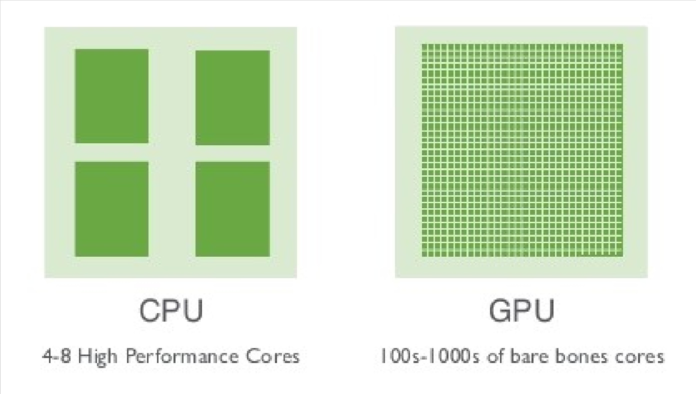
2000s
- GPGPU

2010s
- ImageNet Challenge
- (Krizhevsky, Hinton)





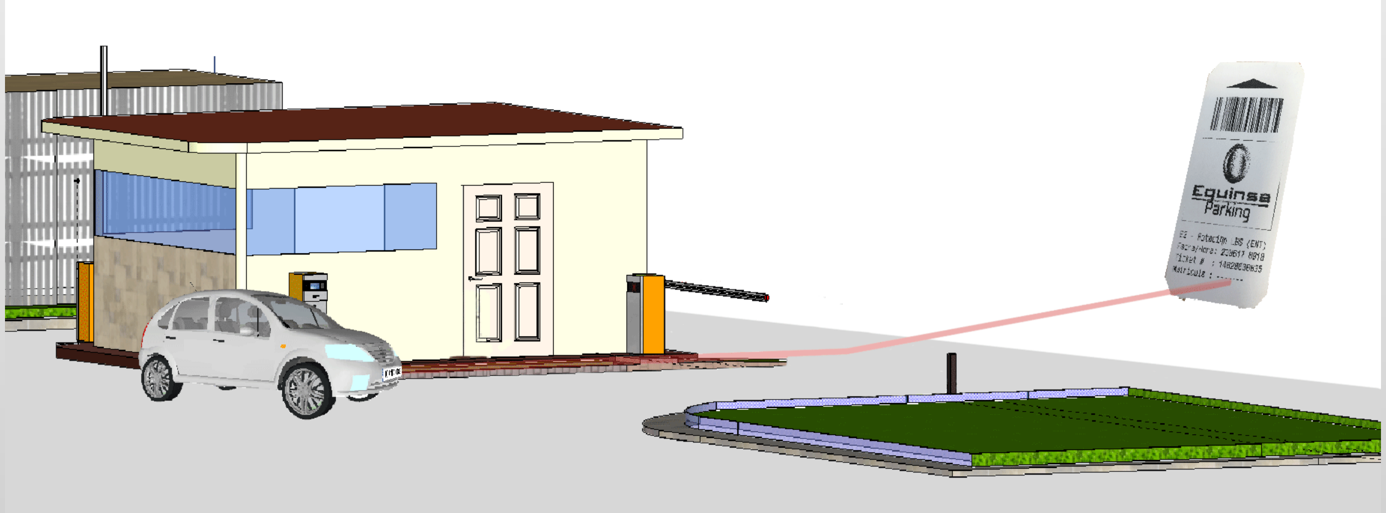




THE SEARCH FOR INVARIANCE IN CV

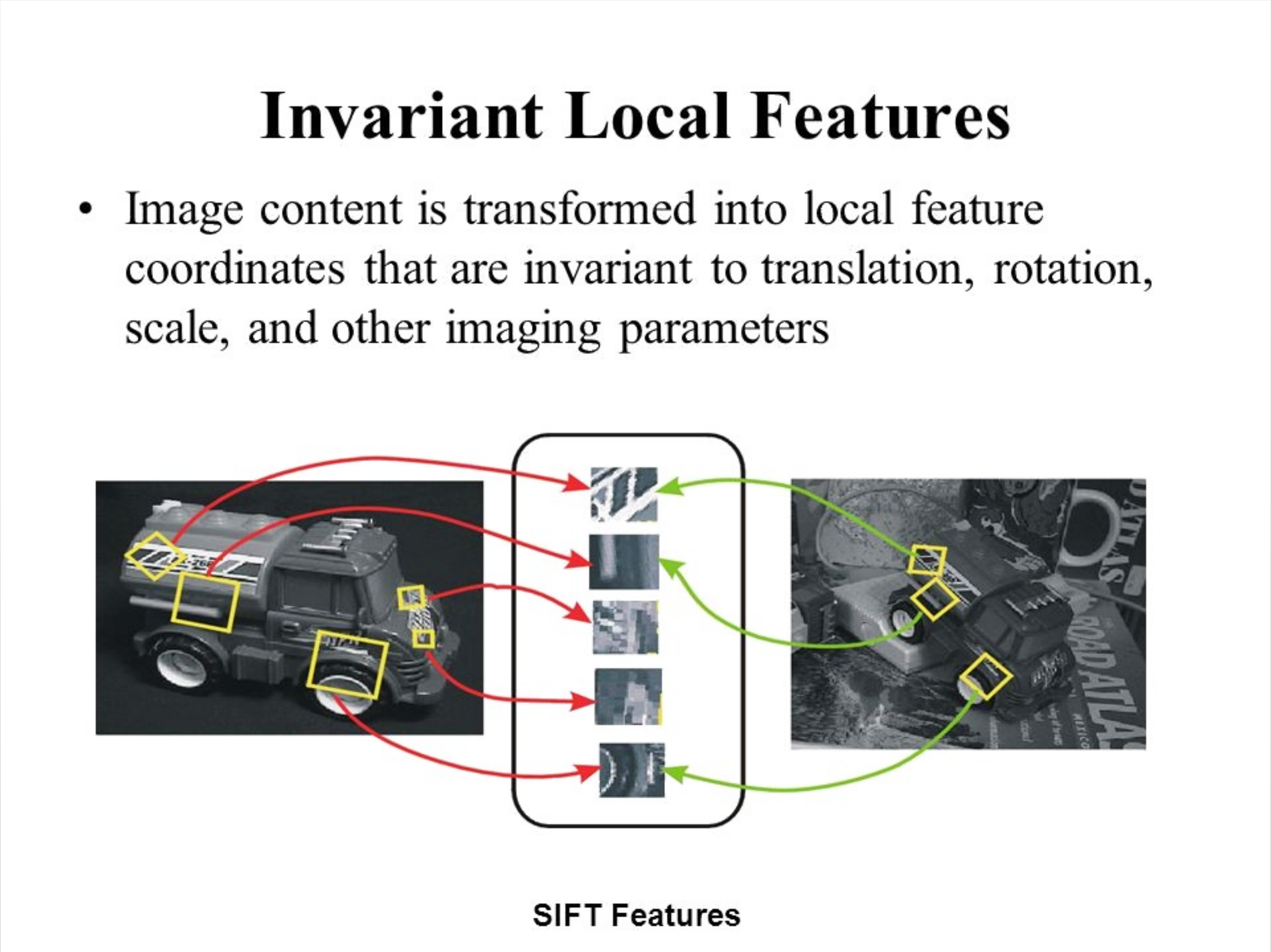
- Images are analyzed by looking for "interesting points" or "features"
- We want such features to have various properties:
- Scale Invariance
- Translation Invariance
- Rotation Invariance
- …

- SIFT (*Lowe et al*)
- HoG (*Dalal*)
- Multi-resolution

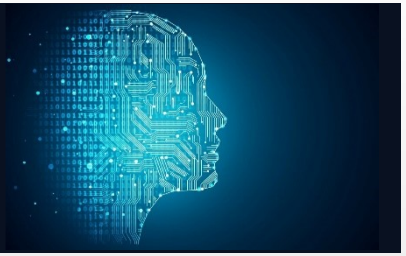


THE SEARCH FOR INVARIANCE IN CV

- Images are analyzed by looking for “interesting points” or “features”
- We want such features to have various properties:
- Scale Invariance
- Translation Invariance
- Rotation Invariance
- …
- SIFT (*Lowe et al*)
- HoG (*Dalal*)
- Multi-resolution



CONVOLUTION \ FILTERING

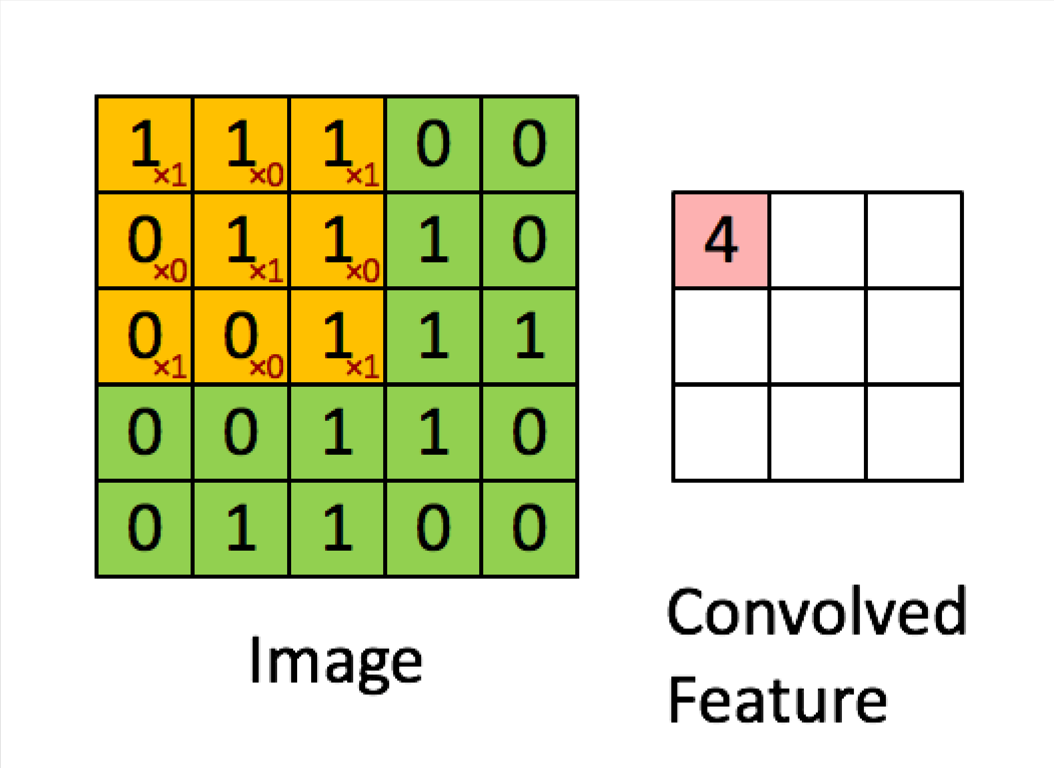


- Convolution is the basis of every feature descriptor
- Most modern and historic computer vision algorithms use convolution *somewhere*
- Convolution 1d continuous definition:

$$(f * h)(t) \equiv \int_{-\infty}^{\infty} f(t-\tau) \cdot h(\tau) d\tau$$

- Convolution 2d discrete definition:

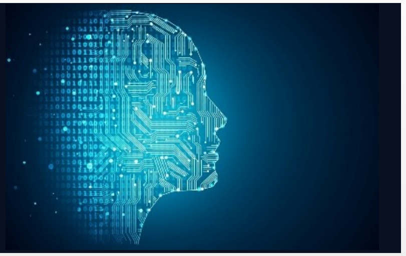
$$(f * h)[n_1, n_2] \equiv \sum_{i \in N_1} \sum_{j \in N_2} h[i, j] \cdot f(n_1 - i, n_2 - j)$$



CONVOLUTION \ FILTERING

- Sobel Filter
- $Sobel = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$

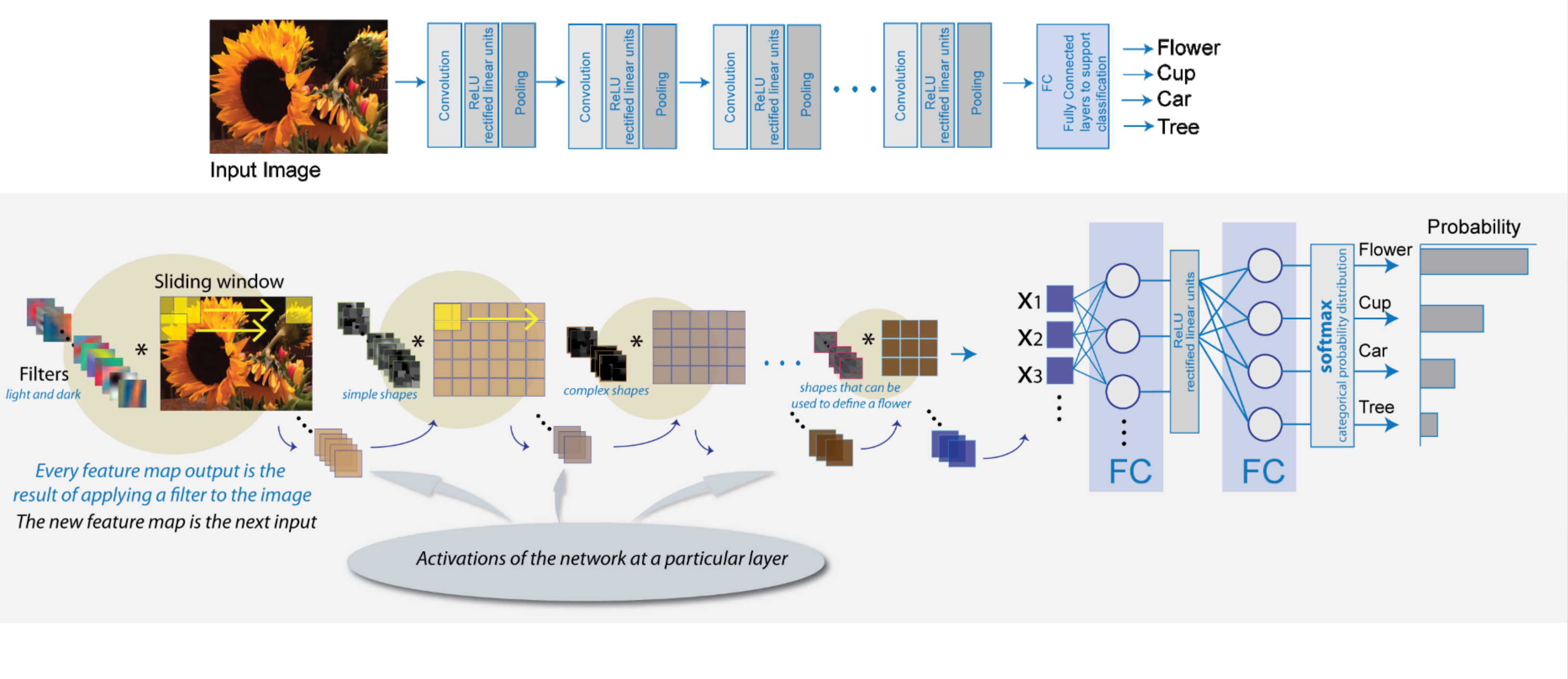
TENSORFLOW IN PYTHON



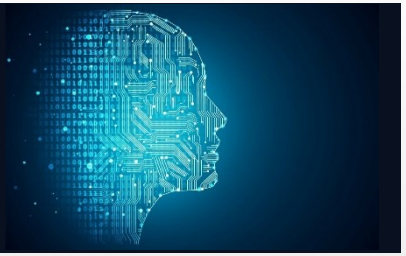
- Key concepts in deep learning simulations
  - MNIST Database
  - Implement Sobel Filter
  - Draw Convolved image with Sobel Filter and Sobel Filter Transposed

THE SEARCH FOR INVARIANCE IN CV

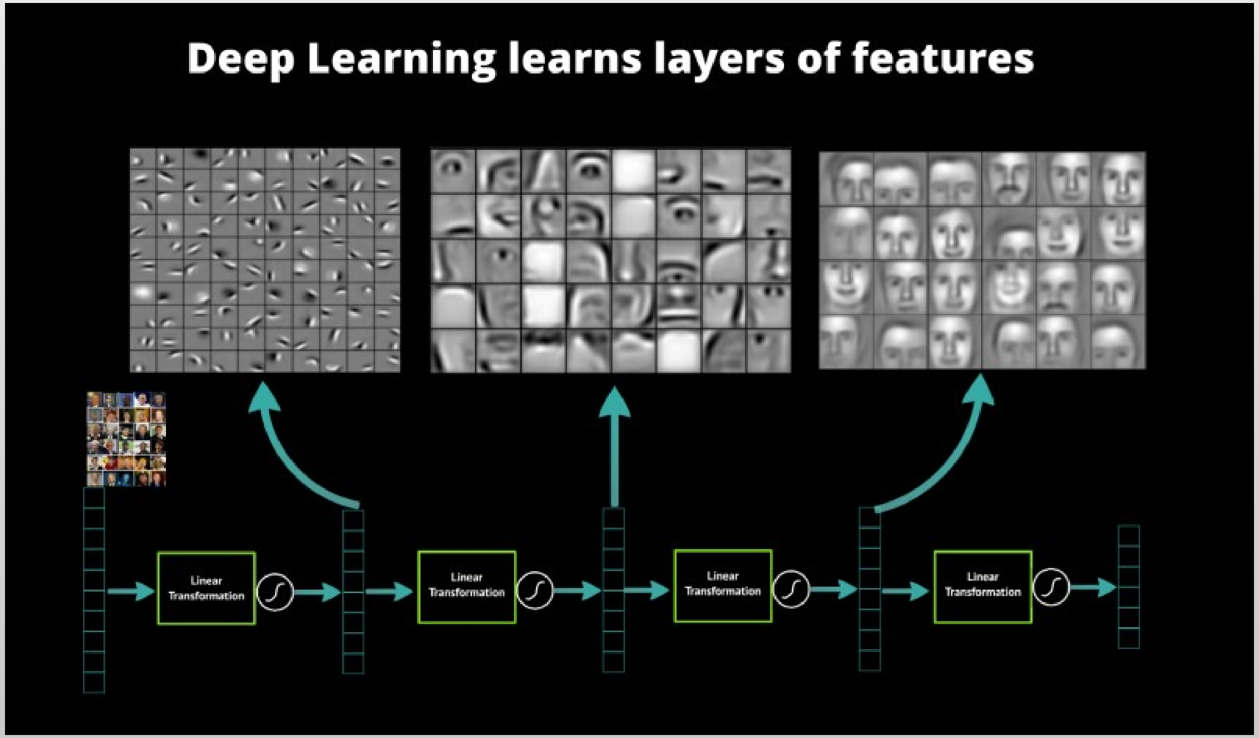
- Instead of "hand-engineered" filters, lets learn the filters from data
- Local activations (image following filter) are pooled together, usually by max pooling
- Activations are usually transformed by a non-linear function (Universal Approximation Theorem).



THE SEARCH FOR INVARIANCE IN CV



- Instead of “hand-engineered” filters, lets learn the filters from data
- Local activations (image following filter) are pooled together, usually by max pooling
- Activations are usually transformed by a non-linear function (Universal Approximation Theorem).



THE SEARCH FOR INVARIANCE IN CV

- Same idea happens in audio:

Low level features (pitch, volume)

↓

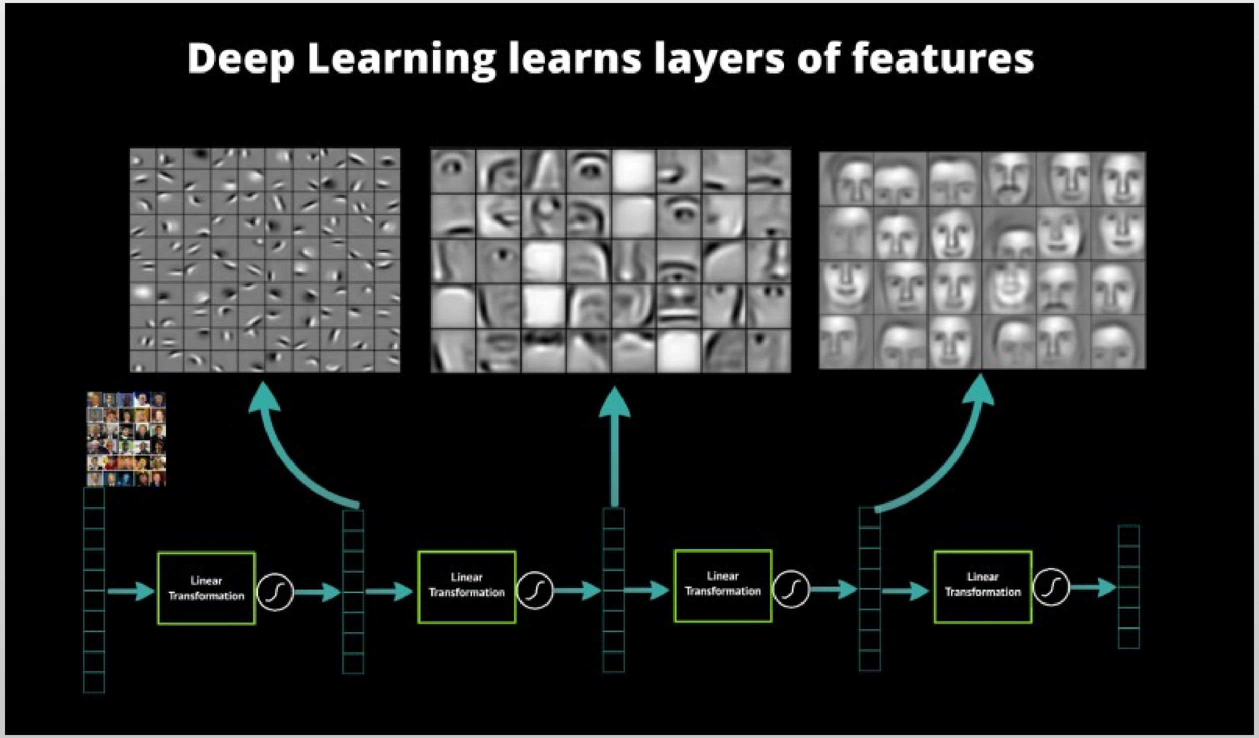
Phenomes

↓

Words

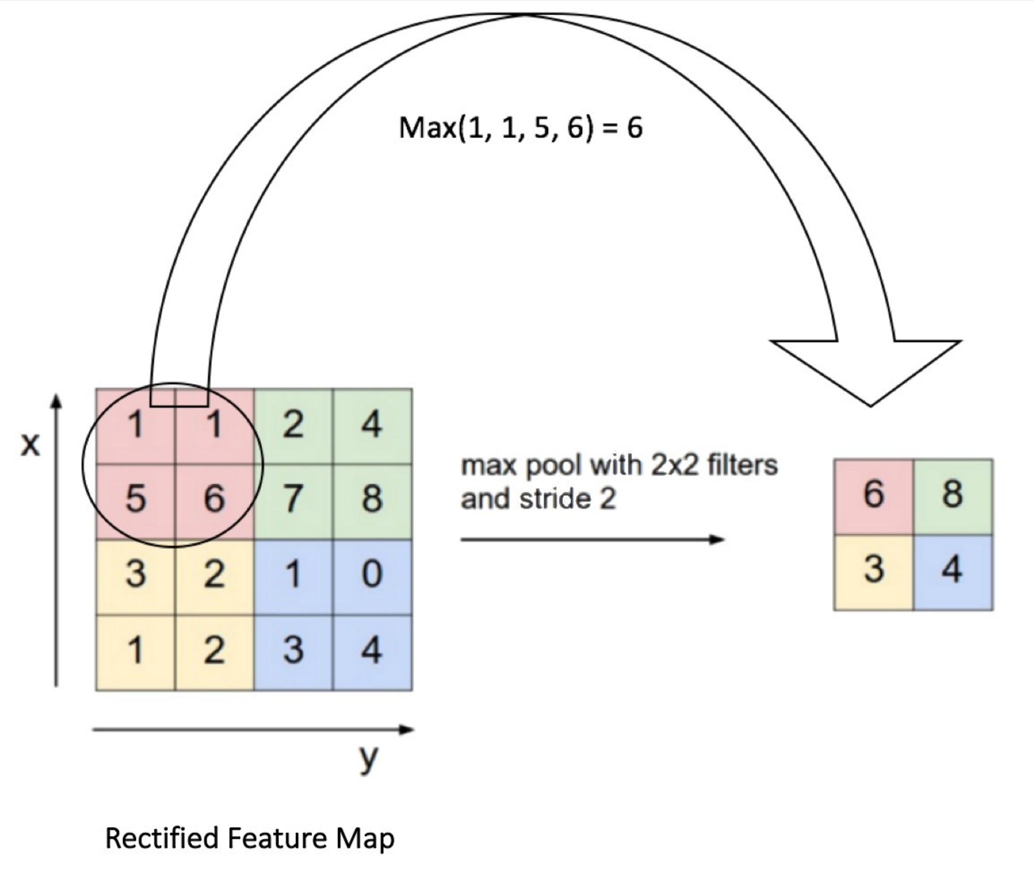
↓

Sentence

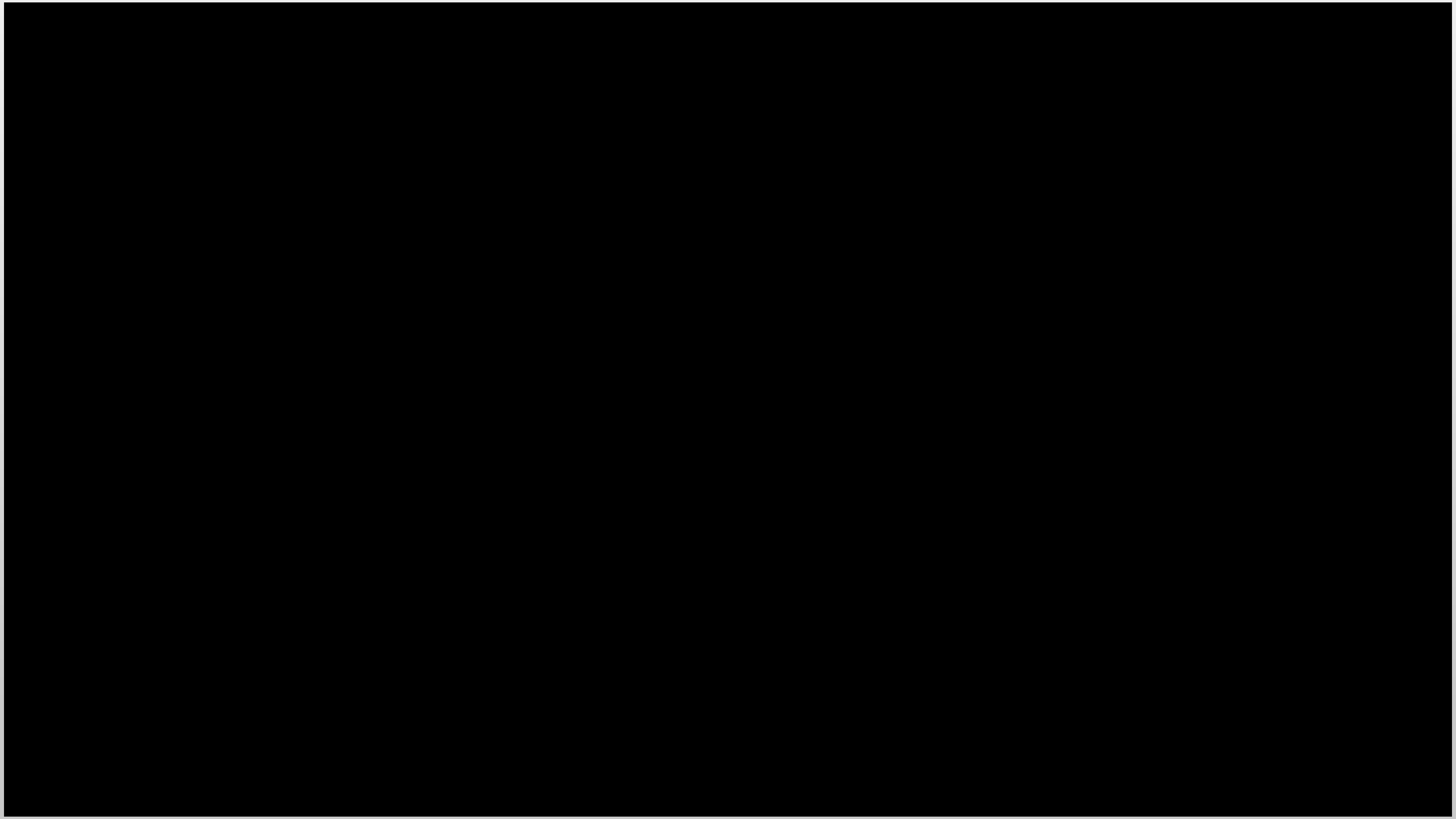


FROM CONVOLUTION TO CLASSIFICATION

- Instead of “hand-engineered” filters, lets learn the filters from data
- Filtered images are usually transformed by a non-linear function (Universal Approximation Theorem). These images are termed “activations”
- Local activations are pooled together, usually by max pooling
- max pooling:



FROM CONVOLUTION TO CLASSIFICATION



FROM CONVOLUTION TO CLASSIFICATION

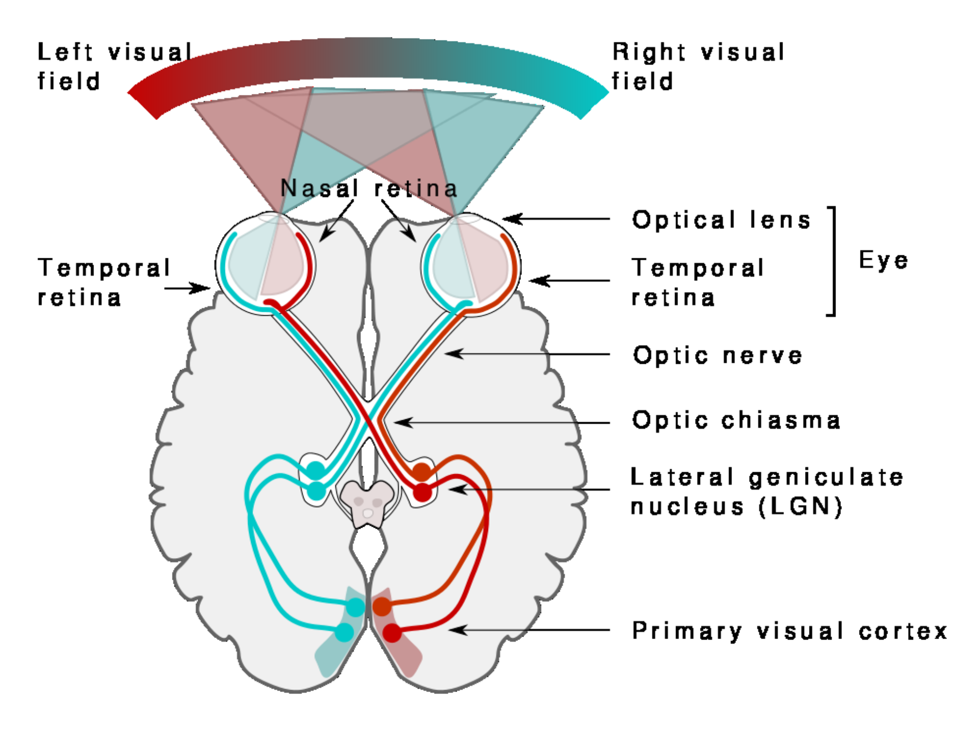
- We repeat the operations: filter, activation, pooling multiple times
- Remember backpropagation?

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

- Layers connected in a *feedforward* manner can be derived using the chain rule:

$$(f^{\circ} g^{\circ} h^{\circ} \dots)'$$

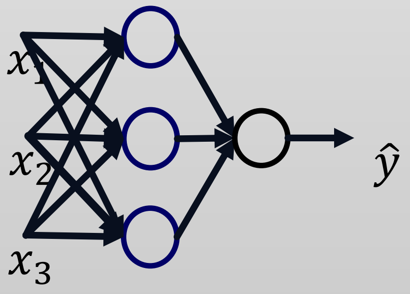
- Convolutional layers stacked together can be optimized in an *end-to-end* manner, without feature engineering or data assumptions!



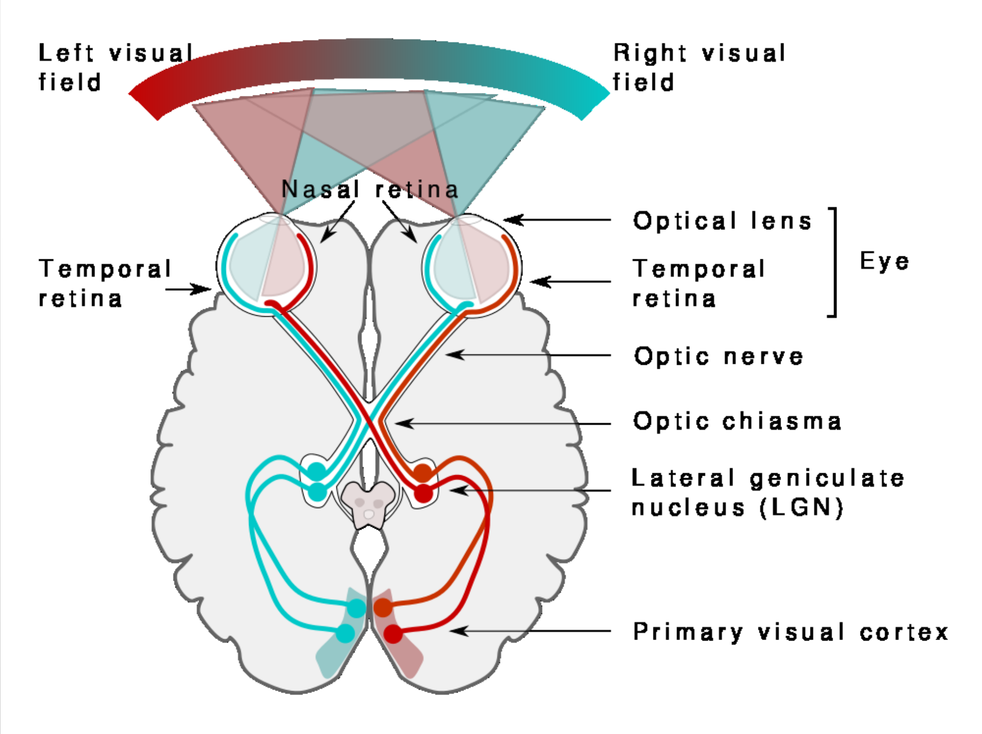
FROM CONVOLUTION TO CLASSIFICATION

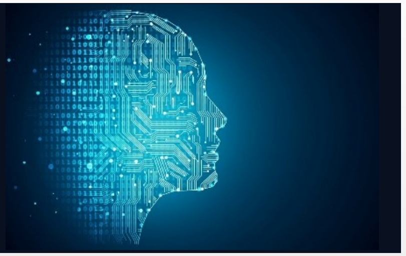
- Two types of layers are used in conventional convnets: Convolutional layers and fully connected layers
- Fully connected layers were studied in the previous part (part I)

$$h_1 = \sigma(\theta_{11} \cdot x_1 + \theta_{21} \cdot x_2 + \theta_{31} \cdot x_3)$$

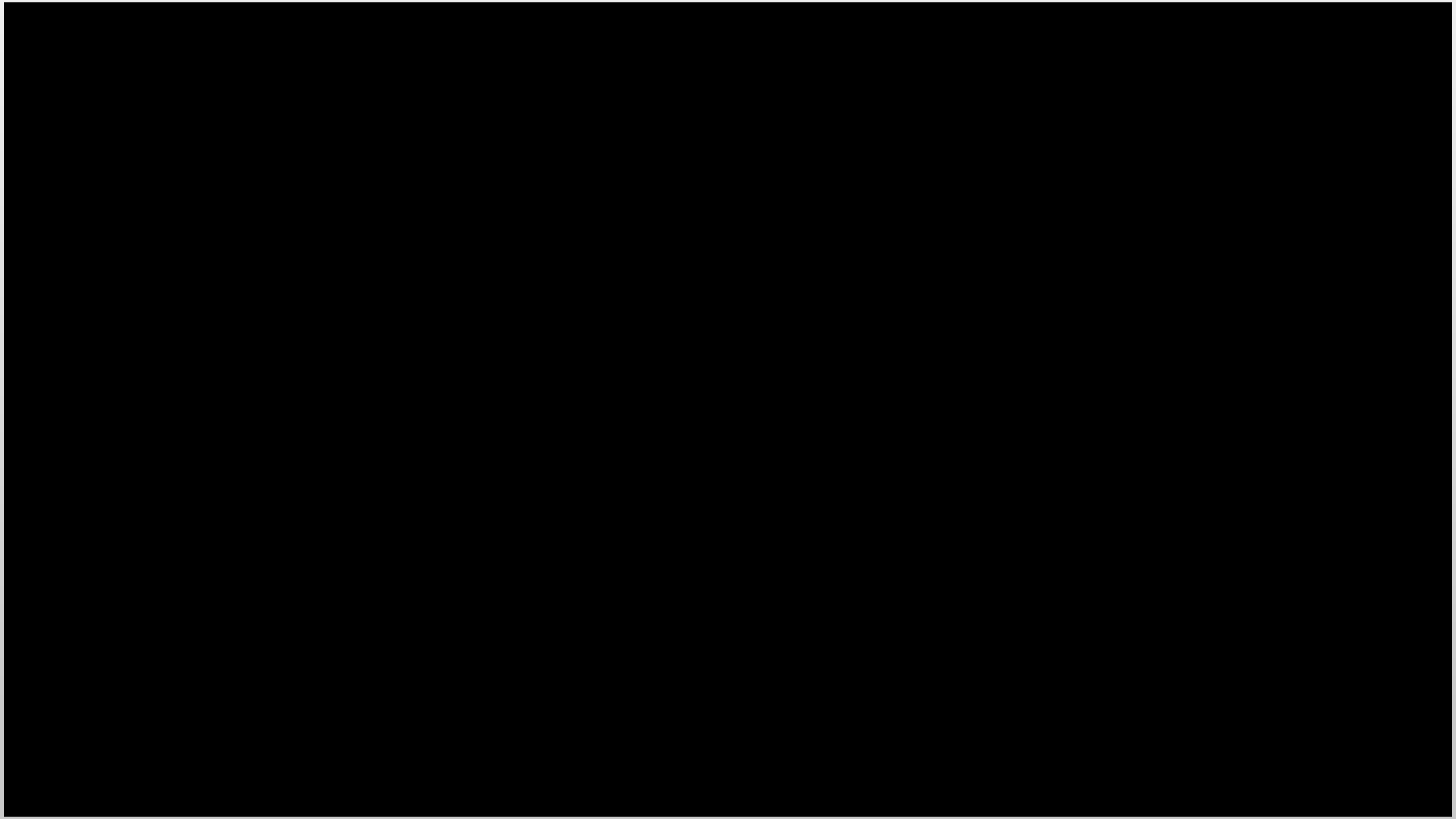


- Remember these layers are equivalent to matrix multiplications

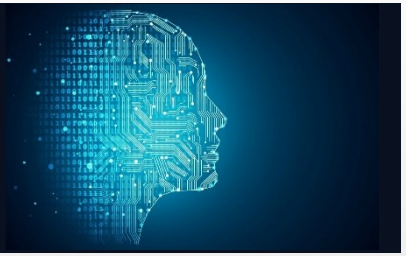




CONVOLUTIONAL NEURAL NETWORK

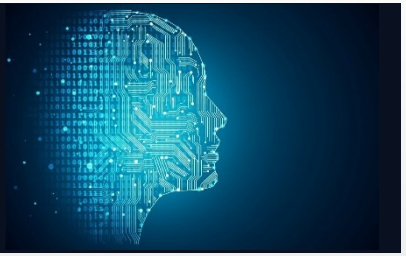


TENSORFLOW IN PYTHON



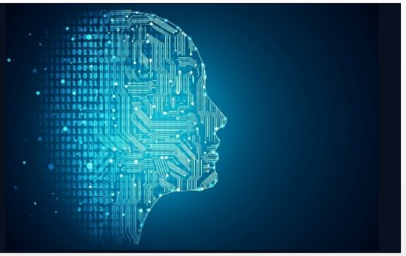
- Key concepts in deep learning simulations
  - MNIST Database
  - CNN Example
  - See conv2d doc and maxpool doc

TENSORFLOW IN PYTHON



- Key concepts in deep learning simulations
  - MNIST Database
  - CNN Example – Add Tensorboard Visualization
  - Visualize filters at first layer, see image summary doc
  - Visualize histogram of activations in fully connected layer
  - Record accuracy and loss
  - Make sure summaries makes sense!

TENSORFLOW IN PYTHON



- Key concepts in deep learning simulations
  - CNN – Fashion MNIST Challenge
  - See repository for more info

INITIALIZATION

What initialization?

- Symmetry breaking problem

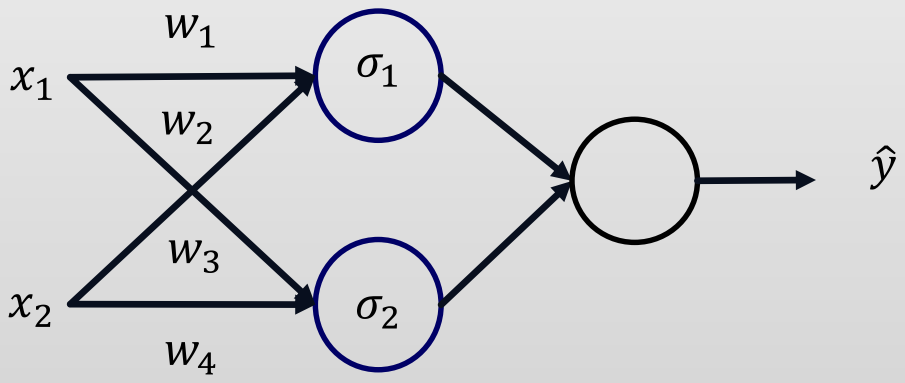
If we initialize the weights matrix with zeros (or constant)

$$w_0 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

the weight will yield symmetric values

$$w_t = \begin{bmatrix} a & b \\ a & b \end{bmatrix}$$

→ We initialize with random "small" matrices (in order to avoid saturating "squash" functions)



INITIALIZATION

What initialization?

- Symmetry breaking problem

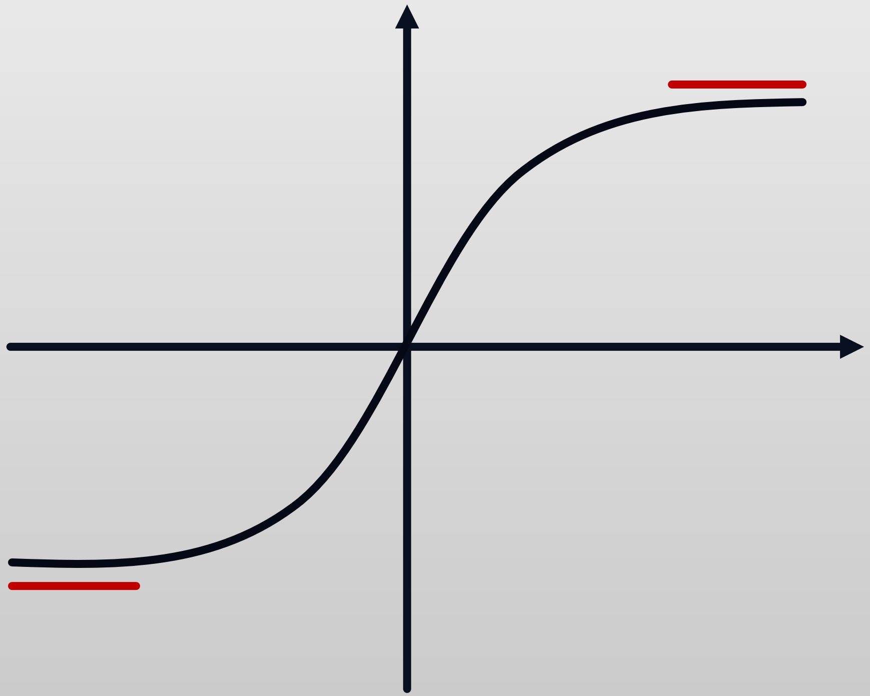
If we initialize the weights matrix with zeros (or constant)

$$w_0 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

the weight will yield symmetric values

$$w_t = \begin{bmatrix} a & b \\ a & b \end{bmatrix}$$

→ We initialize with random "small" matrices (in order to avoid saturating "squash" functions)

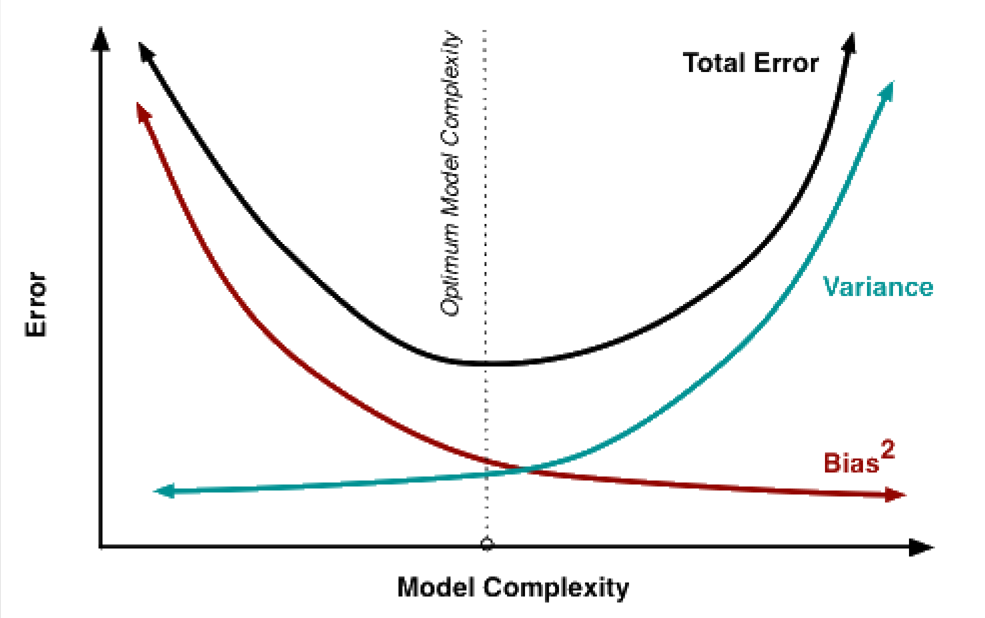


END-TO-END TRAINING

Deep Learning introduces new tools to deal with the bias-variance tradeoff.

*"Make the net big enough until it overfits, then regularize the hell out of it." **–Yann LeCun***

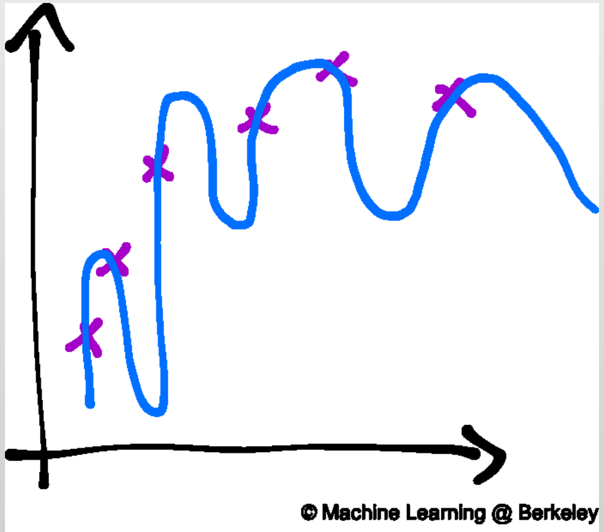
- Adding parameters to a net means it is bigger and more complex → Bias decreases, variance increases (overfitting)
- Regularization techniques are applied to mitigate the increased variance

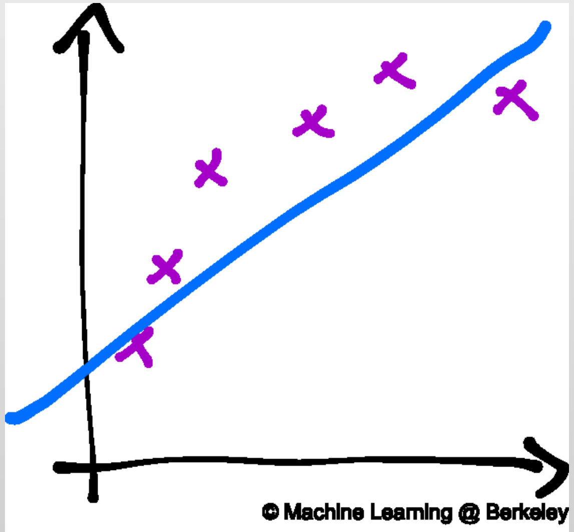


REGULARIZATION

What is regularization?

- Regularization is a technique to mitigate overfitting, usually by introducing some sort of **prior knowledge**.
- This is why ML\DL takes time to develop, since **domain knowledge is hard to encode in end-to-end classifiers**.
- Regularization punishes various complex and unlikely model parameters.





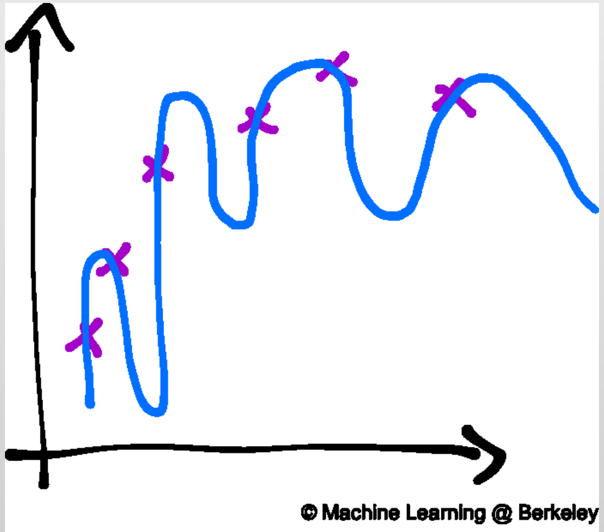
REGULARIZATION

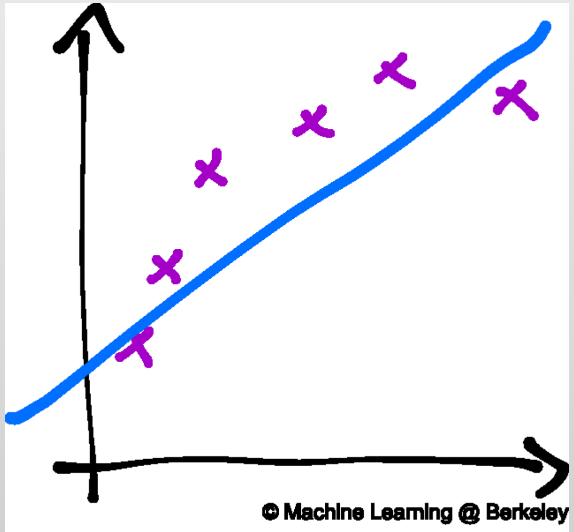
L2 regularization:

- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i w_i^2$

L1 regularization:

- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i |w|$





REGULARIZATION

L2 regularization:

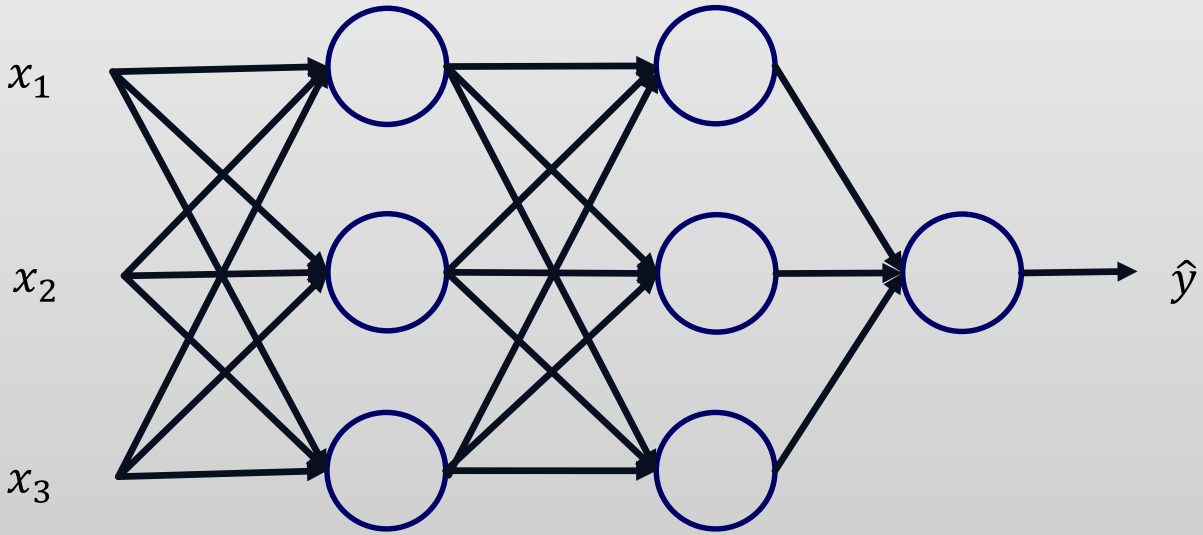
- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i w_i^2$

L1 regularization:

- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i |w|$

Model becomes *sparser*:

- $w \approx 0$



REGULARIZATION

L2 regularization:

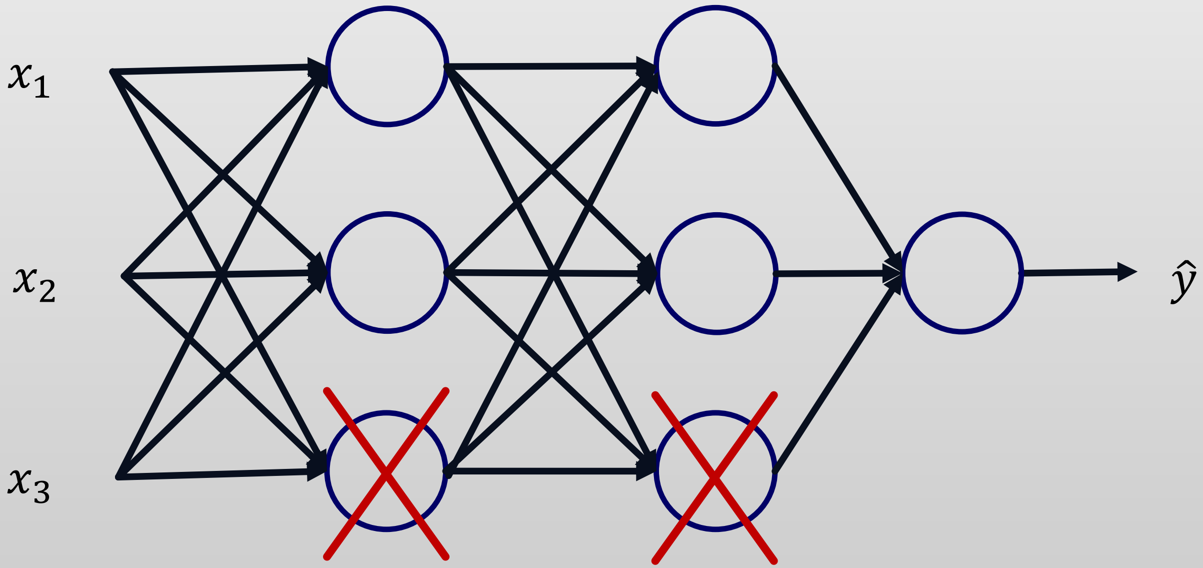
- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i w_i^2$

L1 regularization:

- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i |w|$

Model becomes "*sparser*":

- $w \approx 0$



REGULARIZATION

L2 regularization:

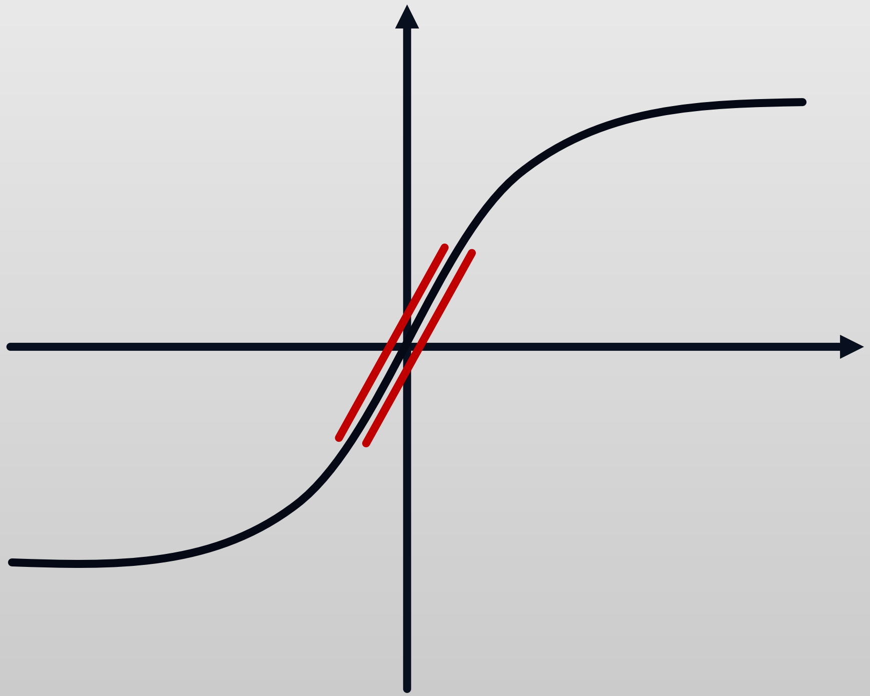
- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i w_i^2$

L1 regularization:

- $L(\theta) = \frac{1}{2 \cdot M} \sum_{i=0}^{M} (\hat{y}(x_i, \theta) - y_i(x_i))^2 + \alpha \cdot \sum_i |w|$

Model becomes "*linear*":

- $w \approx 0$



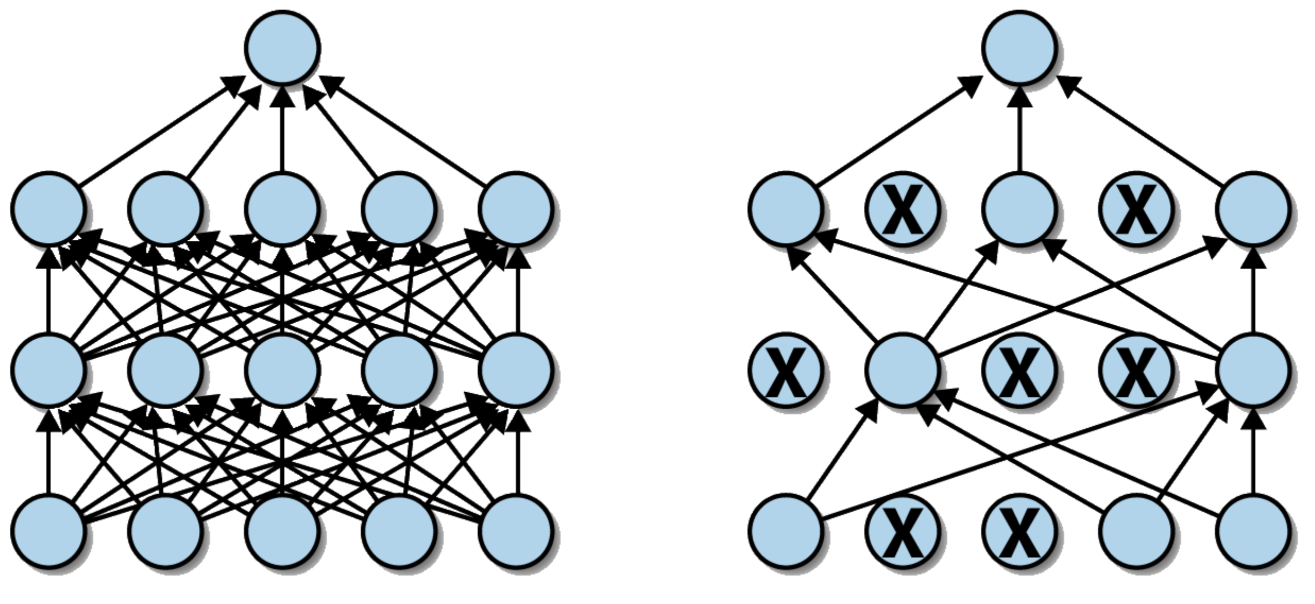
TENSORFLOW IN PYTHON

- Key concepts in deep learning simulations
  - CNN – Fashion MNIST Challenge
  - Add L2 regularization
  - Compare feature space

REGULARIZATION

Dropout regularization:

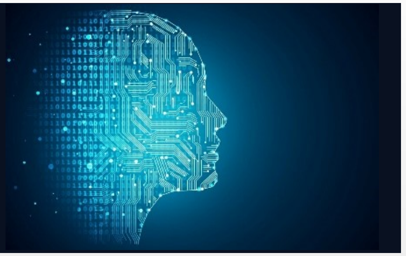
- Randomly “disconnect” weights in each layer
- Avoid strong gradient update for a small subset of neurons (which usually leads to overfitting)
- Tends to make the neural net more “balanced”
- Proved to be equivalent to L2 normalization with “adaptive” weight penalty → It has a somewhat similar effect



(a) Standard Neural Net

(b) After applying dropout

TENSORFLOW IN PYTHON

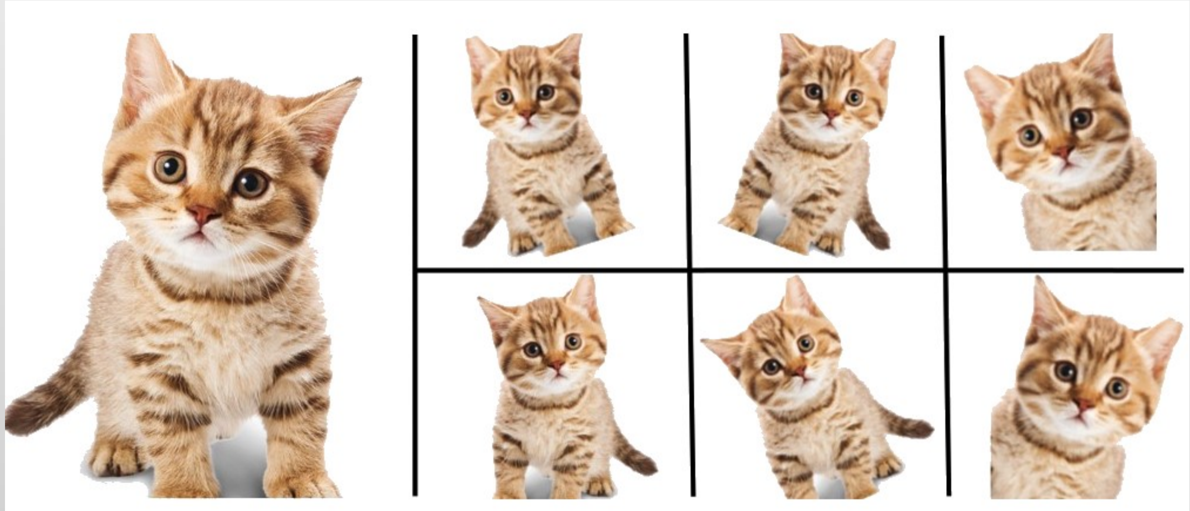


- Key concepts in deep learning simulations
  - CNN – Fashion MNIST Challenge
  - Add dropout regularization
  - Compare feature space

REGULARIZATION

Data augmentation:

- Assume we know how “similar” datasets look.
- Add transformations as additional data points (flip, rotate, zoom, color jitter…)
- In essence we are interpolating the input space, i.e. adding data points heuristically.



REGULARIZATION

Early stopping

- The name is confusing, stop iterating backprop gradient descent when we start overfitting.