Tutorial: Using Neural Networks for Physics

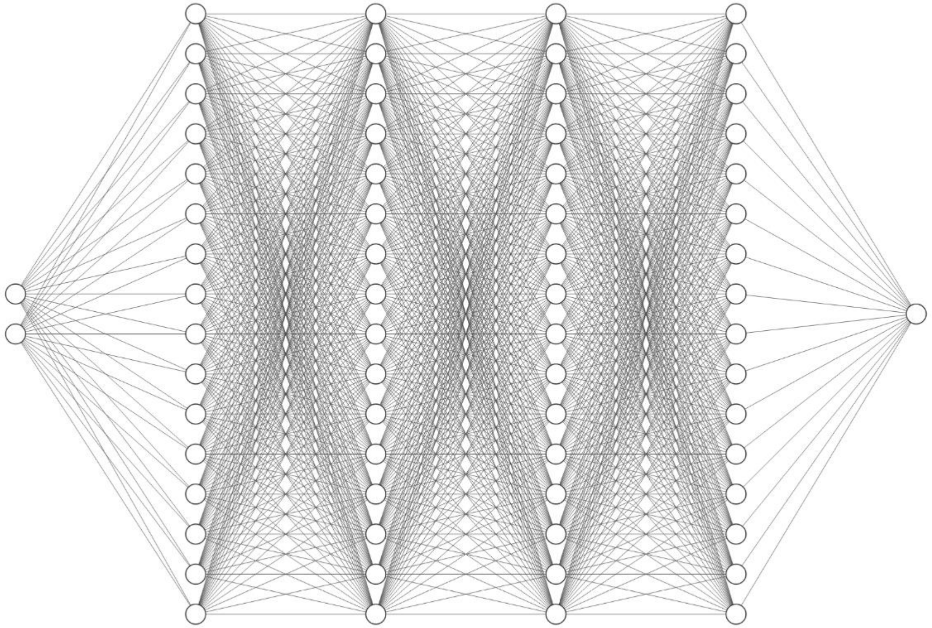
Christopher Leon

Outline

1. Intro to Neural Networks

2. Physics-Informed Neural Networks (PINNs)

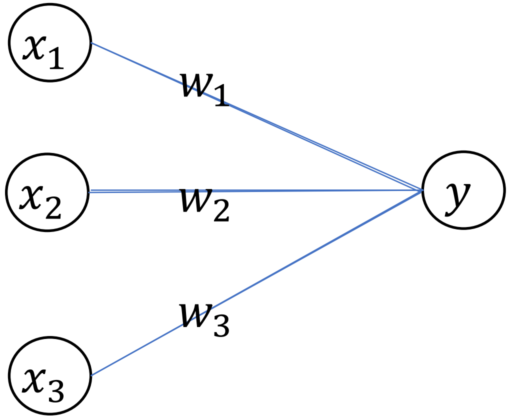
3. Convolutional Neural Networks



Part 1: Intro to Neural Networks

Singe Layer

- Multiply each x_i by w_i and sum:



$$x_1 w_1 + x_2 w_2 + x_3 w_3$$

- Add bias:

$$x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$
$$= \boldsymbol{w} \cdot \boldsymbol{x} + b$$

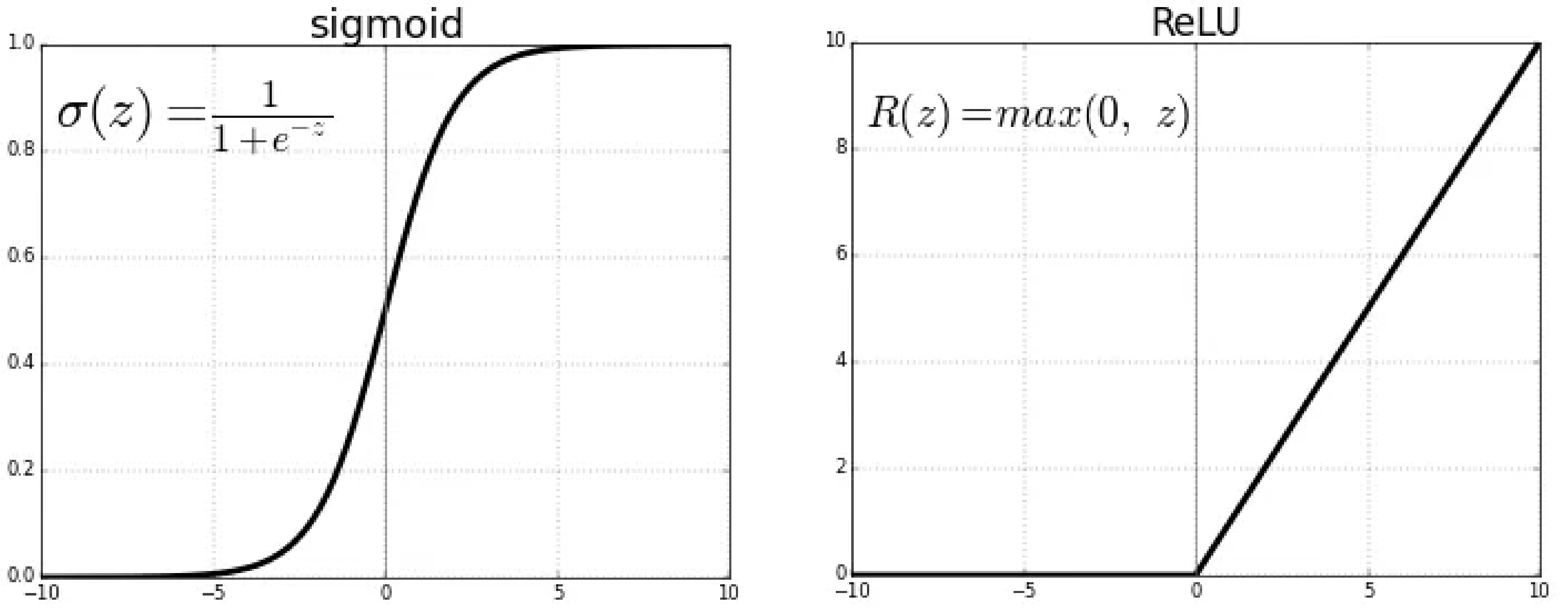
Here w_1, w_2, w_3 and b are parameters

- Put into activation function, σ:

$$y = \sigma(x_1 w_1 + x_2 w_2 + x_3 w_3 + b) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

Activation Function

- Also known as a 'non-linearity'. Inspired by biology.

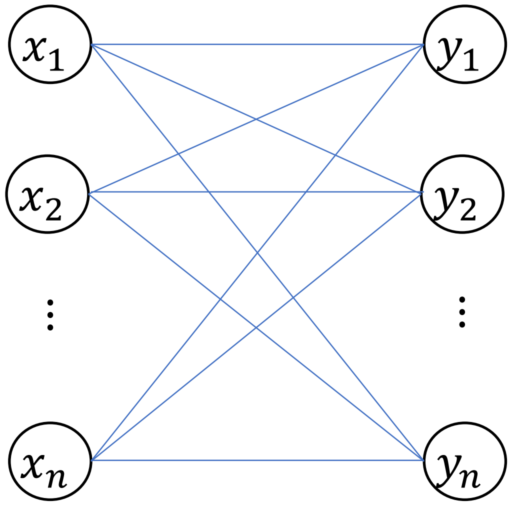


- **ReLU** (or variants) mostly used nowadays. Computationally cheap.
- Sigmoid ⇒ vanishing gradient problem

Singe Layer Dense Network

- Neural Network

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$



$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \sigma(\boldsymbol{w}_1 \cdot \boldsymbol{x} + b_1) \\ \sigma(\boldsymbol{w}_2 \cdot \boldsymbol{x} + b_2) \\ \vdots \\ \sigma(\boldsymbol{w}_n \cdot \boldsymbol{x} + b_n) \end{pmatrix} = \begin{pmatrix} \sigma\big((W\boldsymbol{x})_1 + b_1\big) \\ \sigma\big((W\boldsymbol{x})_2 + b_2\big) \\ \vdots \\ \sigma\big((W\boldsymbol{x})_n + b_n\big) \end{pmatrix}$$

Note: Putting a dummy index (e.g $x_{n+1} = y_{n+1} = 1$) then you can put $W\boldsymbol{x} + \boldsymbol{b}$ into form $\mathrm{B}\tilde{\boldsymbol{x}}$ where B is matrix and $\tilde{\boldsymbol{x}} = (\boldsymbol{x}, 1)$:

$$\begin{pmatrix} \boldsymbol{y} \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} & W & & \boldsymbol{b} \\ 0 & 0 & \ldots 0 & 1 \end{array}\right) \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix}$$

$$\boldsymbol{y} = \sigma \circ A(\boldsymbol{x})$$

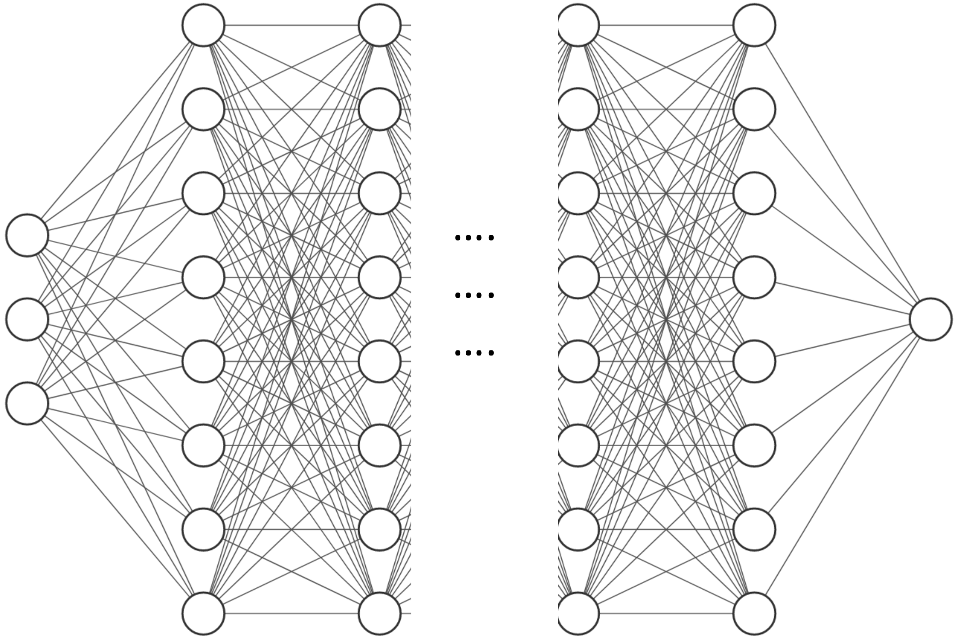
- $\mathrm{A} \rightarrow$ affine transformation: $\mathrm{A}(\mathbf{x}) = \mathrm{W}\mathbf{x} + \mathbf{b}$, where $W \rightarrow$ matrix and $\mathbf{b} \rightarrow$ vector
- In general, $\boldsymbol{y}$ and $\mathbf{x}$ can be different dimensions. W no longer square matrix.

Feed-Forward Neural Networks

- Many Hidden Layers

$$\boxed{\boldsymbol{y} = A_m \circ \ \ldots \circ \sigma \circ A_2 \circ \sigma \circ A_1(\boldsymbol{x})}$$

- Fully connected, dense neural network
- Forward pass → applying above functional compositions to input



Alternatively, for each layer i:

$$\boxed{\boldsymbol{a}_i = \sigma\big(A_i(\boldsymbol{a}_{i-1})\big)}$$

with $\boldsymbol{a}_0 = \boldsymbol{x}$. (Final layer: just use A_i.)

Example

E.g. for a $\mathbb{R} \to \mathbb{R}$ with:

$$y = A_3 \circ \sigma \circ A_2 \circ \sigma \circ A_1(x)$$
$$y = w_3\sigma(w_2\sigma(w_1 x + b_1) + b_2) + b_3$$

Or $\mathbb{R}^2 \to \mathbb{R}^2$ with:

$$\boldsymbol{y} = \sigma \circ A_1(\boldsymbol{x})$$
$$\boldsymbol{y} = \sigma(W_1\boldsymbol{x} + \boldsymbol{b}_1)$$

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \sigma\left(\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right) = \begin{pmatrix} \sigma(w_{11}x_1 + w_{12}x_2 + b_1) \\ \sigma(w_{21}x_1 + w_{22}x_2 + b_2) \end{pmatrix}$$

Universal Function Approximation Theorem

(Informal): *A deep enough NN can approximate any function,* $\mathbb{R}^m \to \mathbb{R}^n$

Neural Network:

$$y = A_m \circ \ \dots \circ \sigma \circ A_2 \circ \sigma \circ A_1(x)$$

- Parameters: Weights & biases
- Doing more functional compositions (“adding” more layers) to improve

Taylor Series:

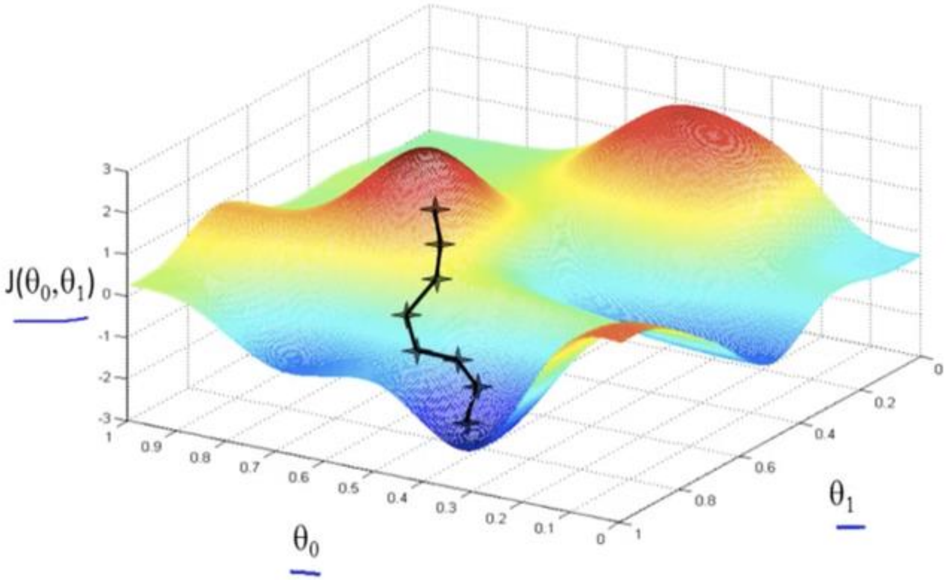
$$y = a_0 + a_1 x + \cdots a_n x^n$$

- Parameters: Coefficients
- Adding higher order terms to improve

Fitting

- Optimization problem. Find parameters $\boldsymbol{\theta}$ (i.e., W’s and b’s) that minimizes your loss function, L.
- E.g.: mean squared error (MSE)

$$L = \sum_i \left(y_{data,i} - f_{\boldsymbol{\theta}}(x_i)\right)^2$$



- Use (variant of) gradient descent
- Need to take derivates

Automatic Differentiation

- Output:

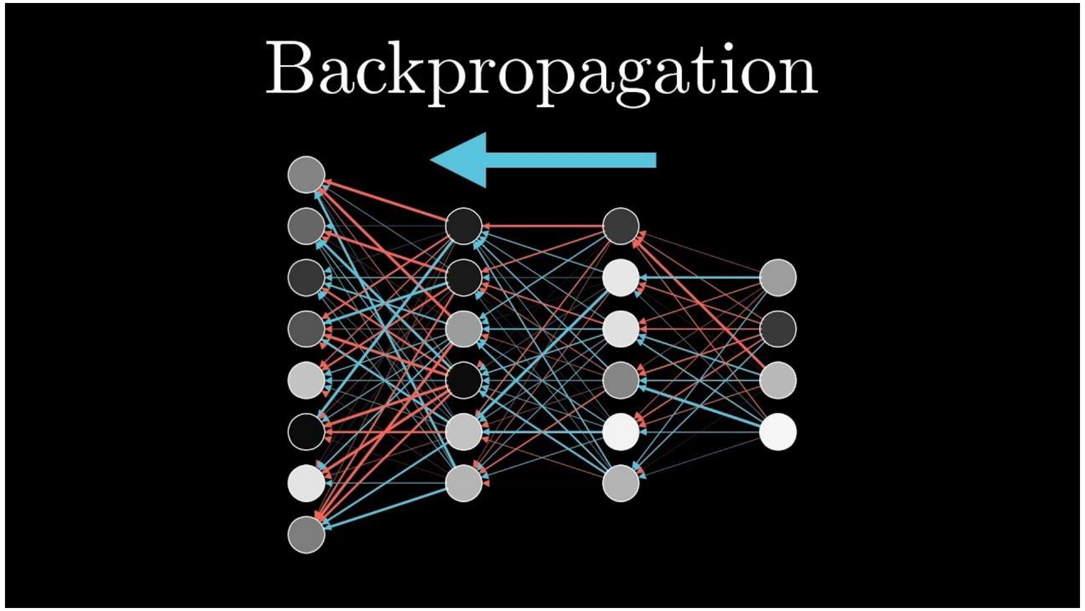
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = A_m \circ \sigma \circ \cdots \ \circ \sigma \circ A_1(\boldsymbol{x})$$

- Using: functional compositional of A's and σ's

- Partial derivatives of $\boldsymbol{\theta}$ involve **chain rule and known derivates of $\boldsymbol{\sigma}$ and $\mathbf{A}$**.

- *Backpropagation*: efficient and accurate algorithm to get derivatives of NN.

⇒Start on final layers, work backwards one at a time to get all



Example: For MSE, need to take

$$\frac{\partial L}{\partial \theta_j} = -\sum_i 2\left(y_{data,i} - f_{\boldsymbol{\theta}}(x_i)\right) \ \frac{\partial f_{\boldsymbol{\theta}}(x_i)}{\partial \theta_j}$$

For, $f_\theta(\boldsymbol{x}) = \sigma \circ A(\boldsymbol{x})$:

$$\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_j} = \frac{\partial \sigma\big((W\boldsymbol{x} + \boldsymbol{b})\big)}{\partial (W\boldsymbol{x} + \boldsymbol{b})} \frac{\partial \big((W\boldsymbol{x} + \boldsymbol{b})\big)}{\partial \theta_j}$$

Exercise: Find expression for above the weights and biases in 1D case (i.e., x is 1D => $wx + b$) with ReLU. Then get gradient of L for each.

In Practice

- Libraries (e.g., TensorFlow and PyTorch) often do autodiff and backprop for you

Simple Derivatives

```
x = tf.Variable(3.0)

with tf.GradientTape() as tape:
  y = x**2
```

```
# dy = 2x * dx
dy_dx = tape.gradient(y, x)
dy_dx.numpy()
```

Derivatives of Model Parameters

```
layer = tf.keras.layers.Dense(2, activation='relu')
x = tf.constant([[1., 2., 3.]])

with tf.GradientTape() as tape:
  # Forward pass
  y = layer(x)
  loss = tf.reduce_mean(y**2)

# Calculate gradients with respect to every trainable variable
grad = tape.gradient(loss, layer.trainable_variables)
```

[Introduction to gradients and automatic differentiation](#) (Tensorflow)

Gradient Descent

$$\theta_j \rightarrow \theta_j - \epsilon \frac{\partial L}{\partial \theta_j}$$

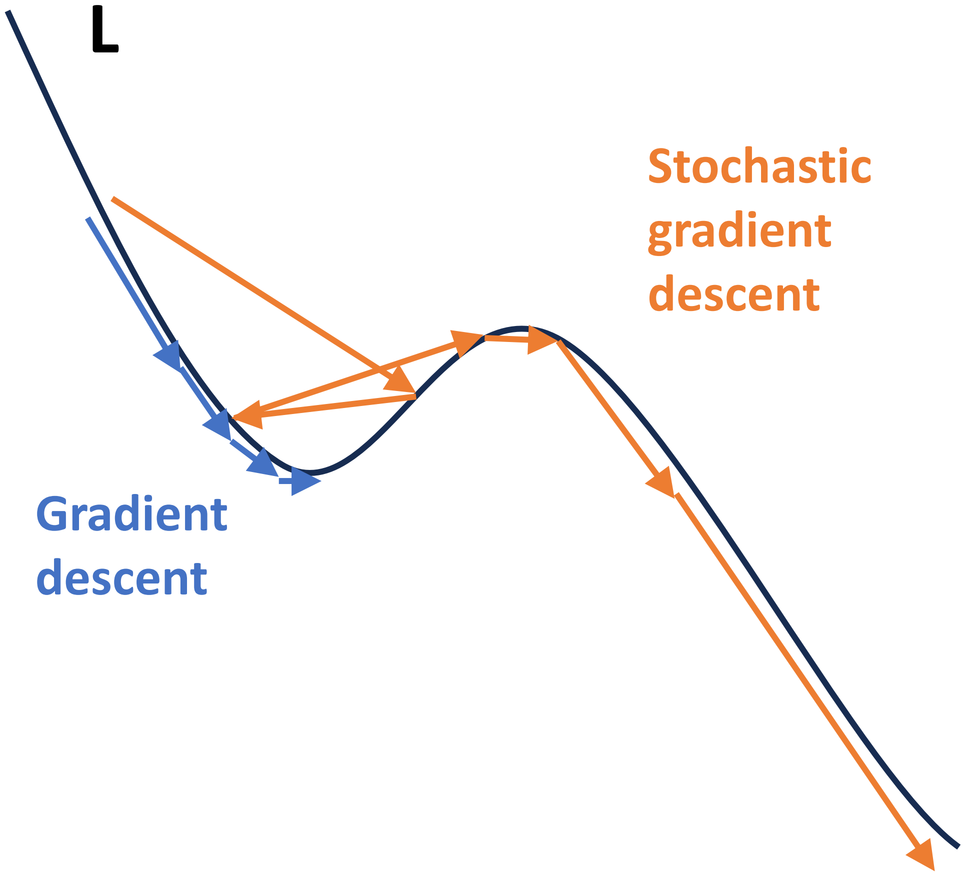
where ϵ is ‘learning rate’; hyperparameter

Stochastic gradient descent: use only part of data (“batch”) for L every parameters update.

- More computationally efficient
- Less likely to end up stuck in local stationary point, explore more

Data

batch	batch	batch	batch	batch	batch



$$\begin{aligned} L_B &= L + (L_B - L) \\ &= L + \eta \end{aligned}$$

Optimization (Optional)

Can improve optimization by:

Beginning with high "noise"

- Escape local stationary points
- Greater exploration of parameter space

As time goes on, lower "noise"

- Narrow in on promising region
- At end, want to find best local point

Can lower "noise" by:

- Increasing batch size
- Decreasing ϵ

Similar to simulated annealing optimization.

- Analogy: crystallization
- Find lowest energy state by starting high temperature, T, gradually lowering T (lessen "thermal noise")
- Too fast: liquid frozen in place, amorphous solid (not lowest)

Randomness, properly harnessed, is a powerful tool

Tensorflow

- Best to store sequence data as arrays ("tensors"):

$$data[\, idx_{data}, idx_X, idx_{color}]$$

- 2D image

$$data[\, idx_{data}, idx_x, idx_y, idx_{color}]$$

Grayscale: $idx_{color} = 1$ (dummy)
Color: $idx_{color} = 1,2,3\ (RBG)$

Scalar field ↔ Grayscale
3D vector field ↔ Color Image
Tensorflow made with images in mind.

Arrays: a[i], b[i], c[i]

<u>Loop</u>

```
for i in range(n):
   a[i] = b[i] + c[i]
```

<u>Vectorization</u>

```
a= b + c
```

Vectorization tends to be much quicker

Notebook

- Creating Neural Networks in TensorFlow

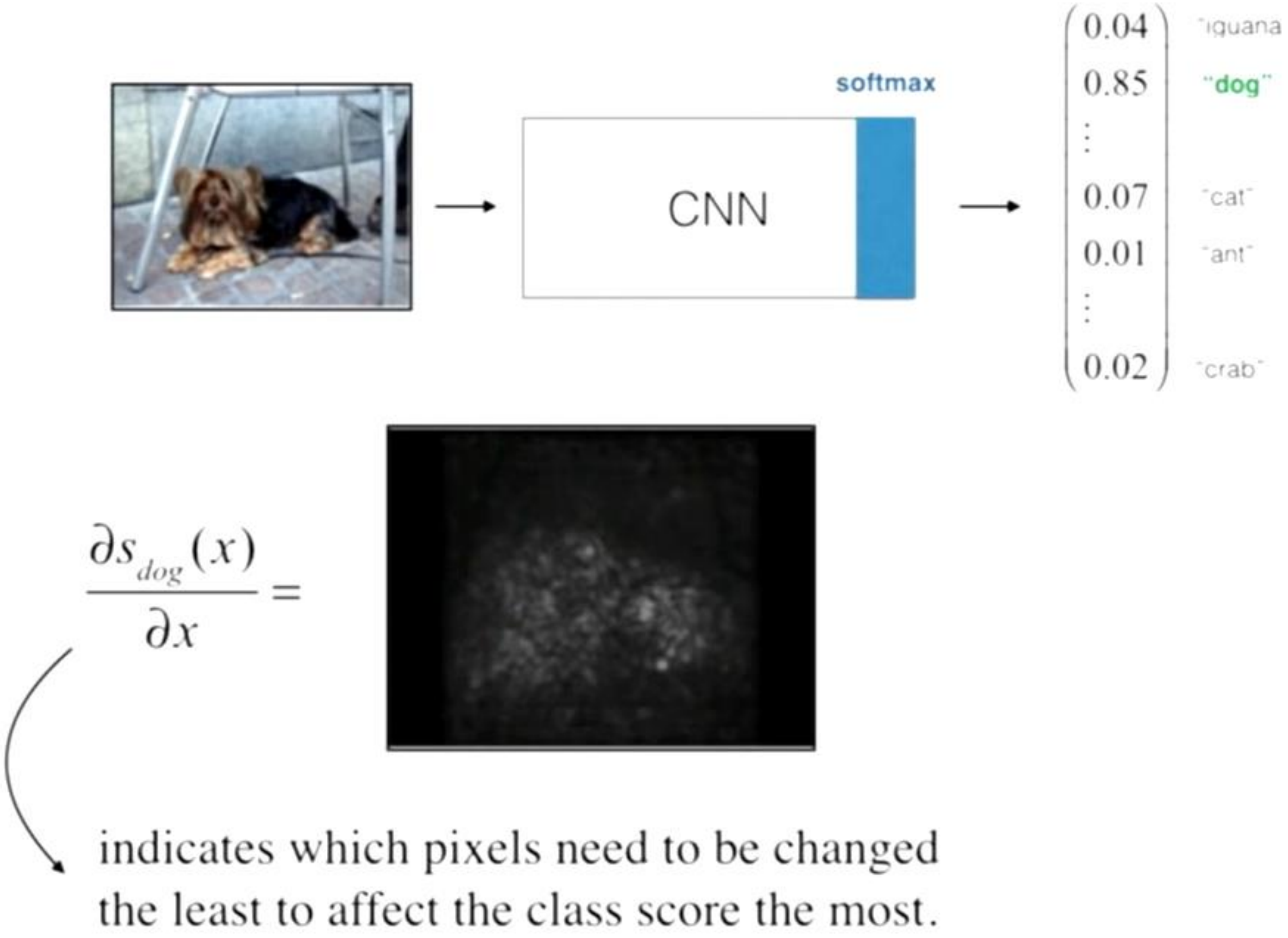
Exercise: Find out what is “momentum” in the context of SGD. Why does it help?

Read about Adam (adaptive moment estimation) optimization method. This is the most commonly used optimization technique.

Part 2: Physics Informed Neural Networks (PINNs)

Autodiff on Inputs

Can use autodiff to calculate derivate of output **w.r.t. inputs** $(\boldsymbol{a_0} = \boldsymbol{x})$



[Karen Simonyan et al. (2014): Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps]

Backpropagate one more layer.

Exercise: Find out what a Softmax layer is. Why does it have that name? (Hint: think about what happens when one score is very large.)

Does Softmax remind you of anything from statistical mechanics?

Physics Informed Neural Networks

- E.g., for 1 dimensional output

$$y = A_m \circ \sigma \circ \cdots \ \circ \sigma \circ A_1(\boldsymbol{x})$$

Can calculate:

$$\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_4}, \frac{\partial^2 y}{\partial x_1^2} \dots .$$

with autodiff.

Again, just use chain rule and known derivates of A and σ.

PDE's

- Much of physics is described by partial differential equation of the form:

$$f = \frac{\partial u}{\partial t} + \mathcal{N} u = 0$$

where $\mathcal{N}$ is an operator involving spatial derivatives. Often want to solve PDE with some initial/boundary conditions.

- Examples: 1D heat equation or 3D Schrodinger equation

$$\frac{\partial u}{\partial t} - k \frac{\partial^2 u}{\partial x^2} = 0 \qquad\qquad i\hbar \frac{\partial u}{\partial t} + \frac{\hbar^2}{2m} \nabla^2 u - V(x) u = 0$$

The PINN Optimization

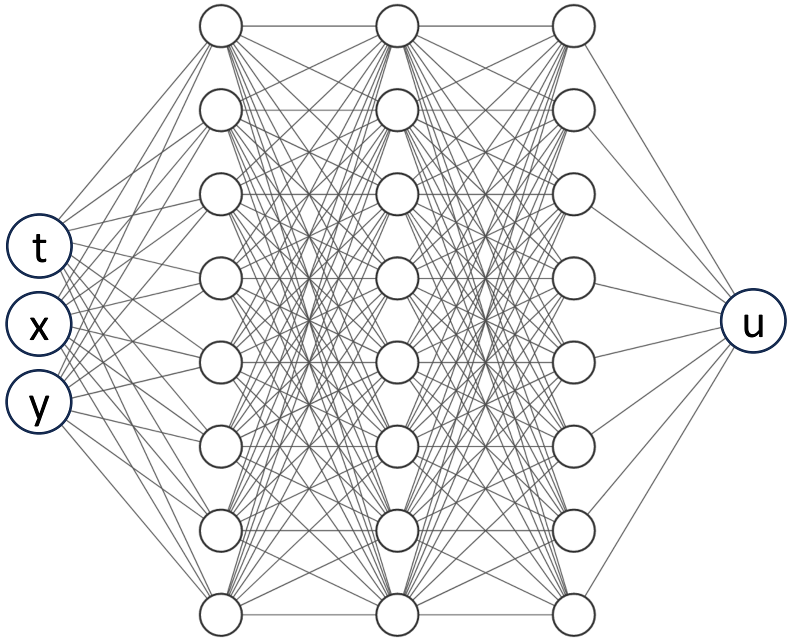
- PINNs: **use a fully connected neural network to solve PDE problem**

- Want NN to satisfy 3 things: PDE, boundary and initial conditions

- Set up loss function:

$$L = L_{IC} + L_{BC} + L_{PDE}$$

- If $L = 0$, NN is a solution of the PDE.



Example: Non-linear Schrodinger Equation

- From Raissi (2019)

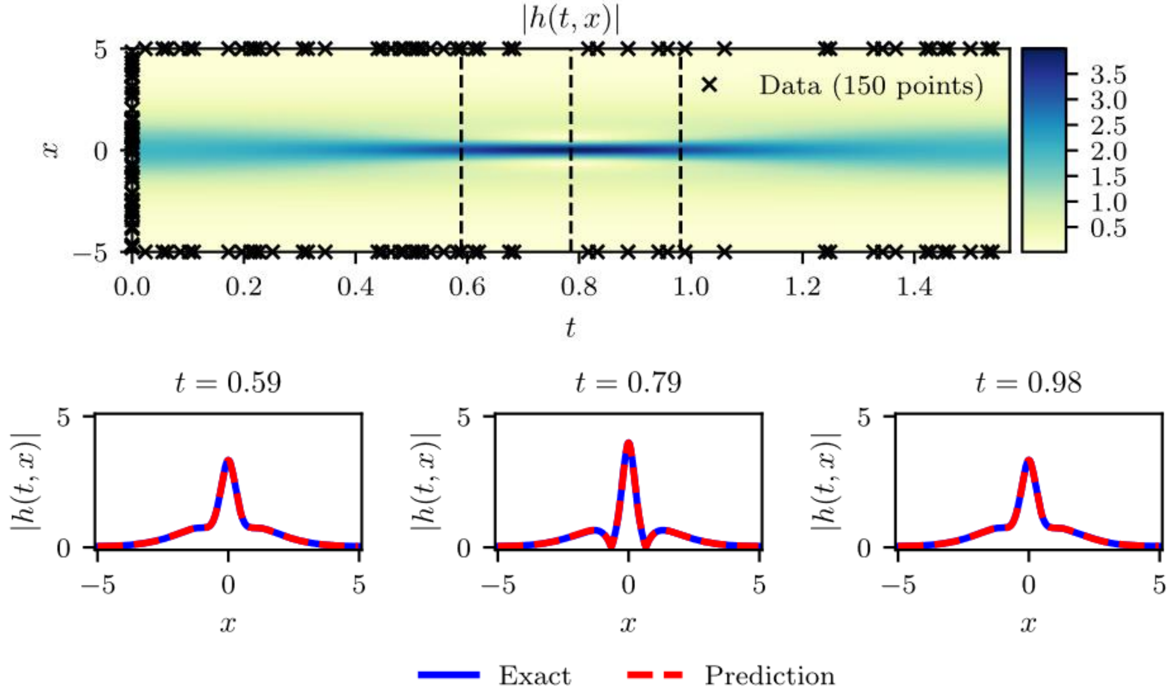
$$
\begin{aligned}
&ih_t + 0.5h_{xx} + |h|^2 h = 0, \quad x \in [-5, 5], \quad t \in [0, \pi/2],\\
&h(0, x) = 2\,\mathrm{sech}(x),\\
&h(t, -5) = h(t, 5),\\
&h_x(t, -5) = h_x(t, 5),
\end{aligned}
$$

Journal of Computational Physics
Volume 378, 1 February 2019, Pages 686-707

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M. Raissi [a], P. Perdikaris [b], G.E. Karniadakis [a]

Citations: 8315



- We have ($L = MSE$):

$$f := ih_t + 0.5h_{xx} + |h|^2 h,$$

$$MSE = MSE_0 + MSE_b + MSE_f,$$

where

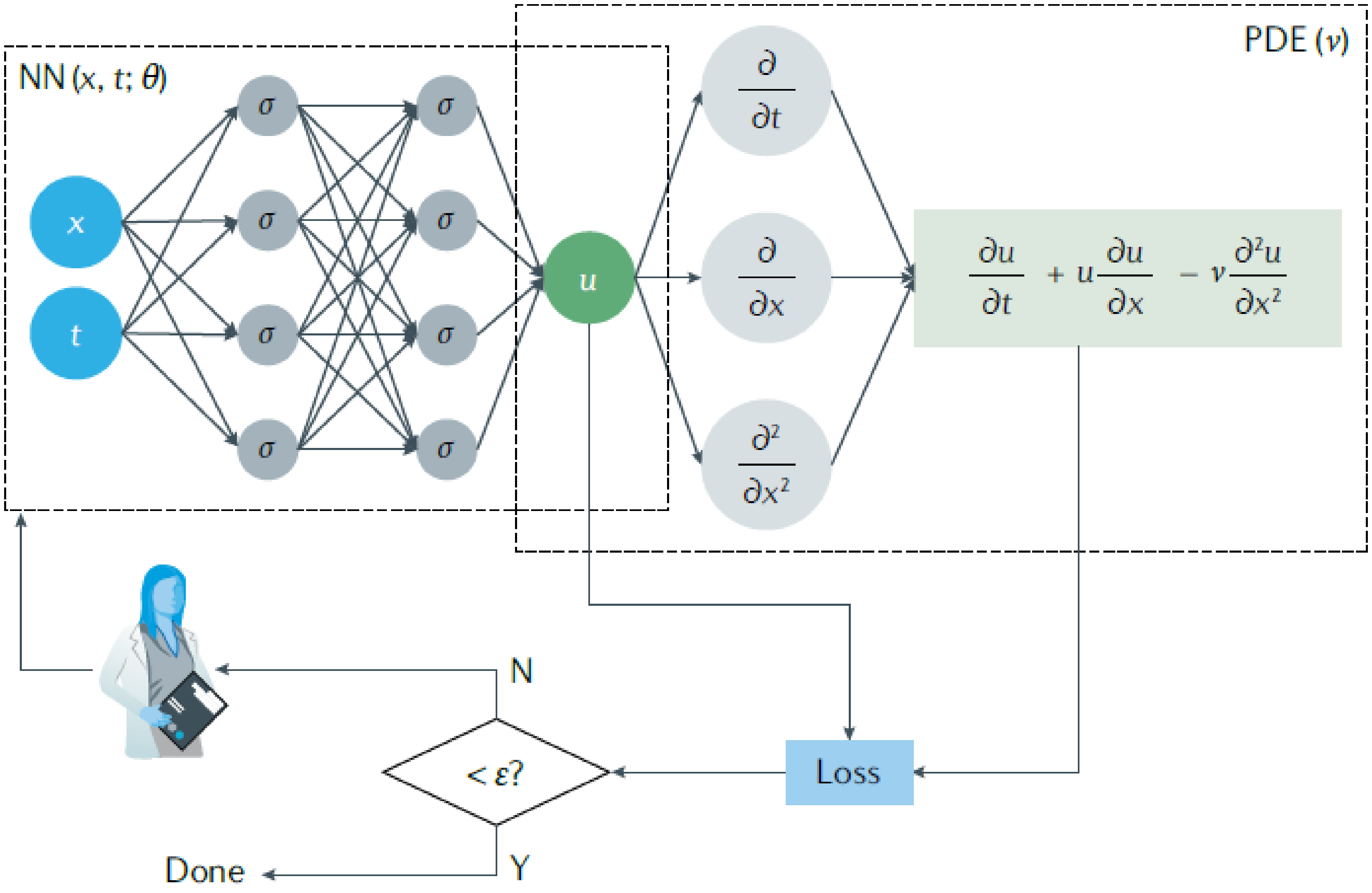
$$MSE_0 = \frac{1}{N_0}\sum_{i=1}^{N_0} |h(0, x_0^i) - h_0^i|^2,$$

$$MSE_b = \frac{1}{N_b}\sum_{i=1}^{N_b} \left(|h^i(t_b^i, -5) - h^i(t_b^i, 5)|^2 + |h_x^i(t_b^i, -5) - h_x^i(t_b^i, 5)|^2\right),$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

- $N_f = 20,000$
- NN: Dense 5 layers, 100 neurons per layer, tanh activation

PINNS - General



Karniadakis, G.E. et al. *Nature Reviews Physics* 3.6 (2021): 422-440.

PINNs

Advantages

- Very general way to solve PDE's
- Compared with conventional PDE methods, easier to master
- Tries to get NN's output to obey physical laws
- Mesh-independent

Disadvantages

- Compared with conventional PDE methods, not well understood theoretically and not as accurate
- Only produces a single instance of a PDE. Need to retrain if BC/IC's are different.
- Different terms in L compete, might not find NN that minimizes all simultaneously.
- Bias against some Fourier modes (Wang, S., Wang, H., & Perdikaris, P. (2021).)

Notebook

- Use PINN approach to find solution to 1D Heat equation:

$$\frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} = 0$$

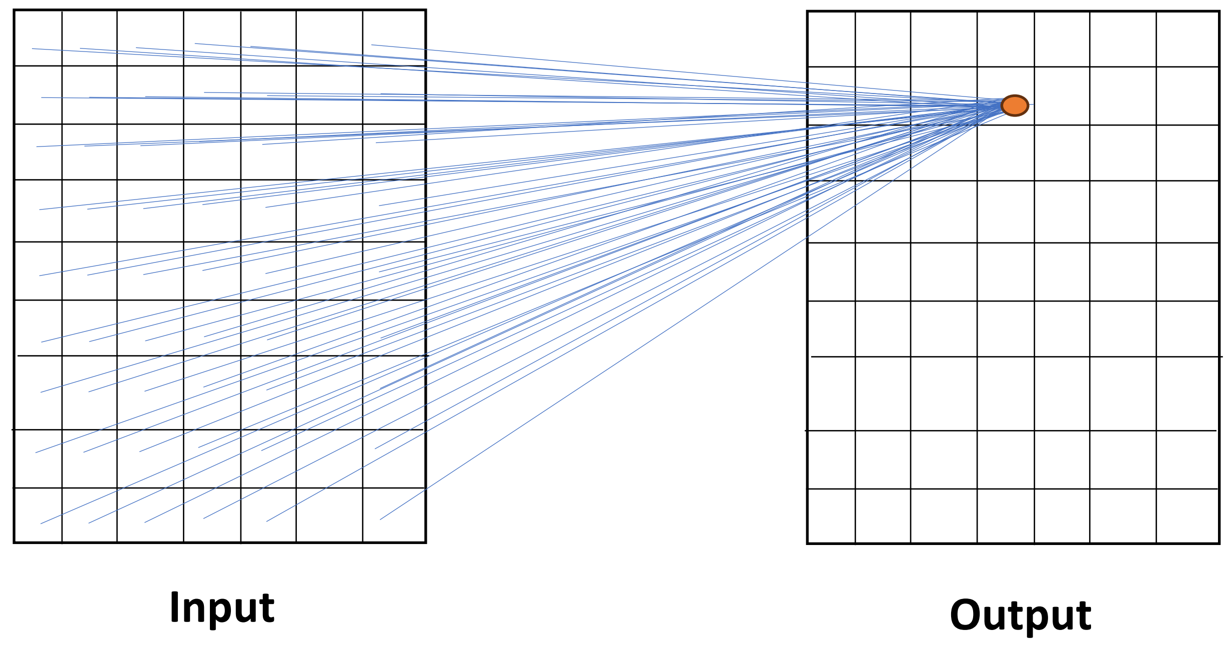
Part 3: Convolutional Neural Networks (CNNs)

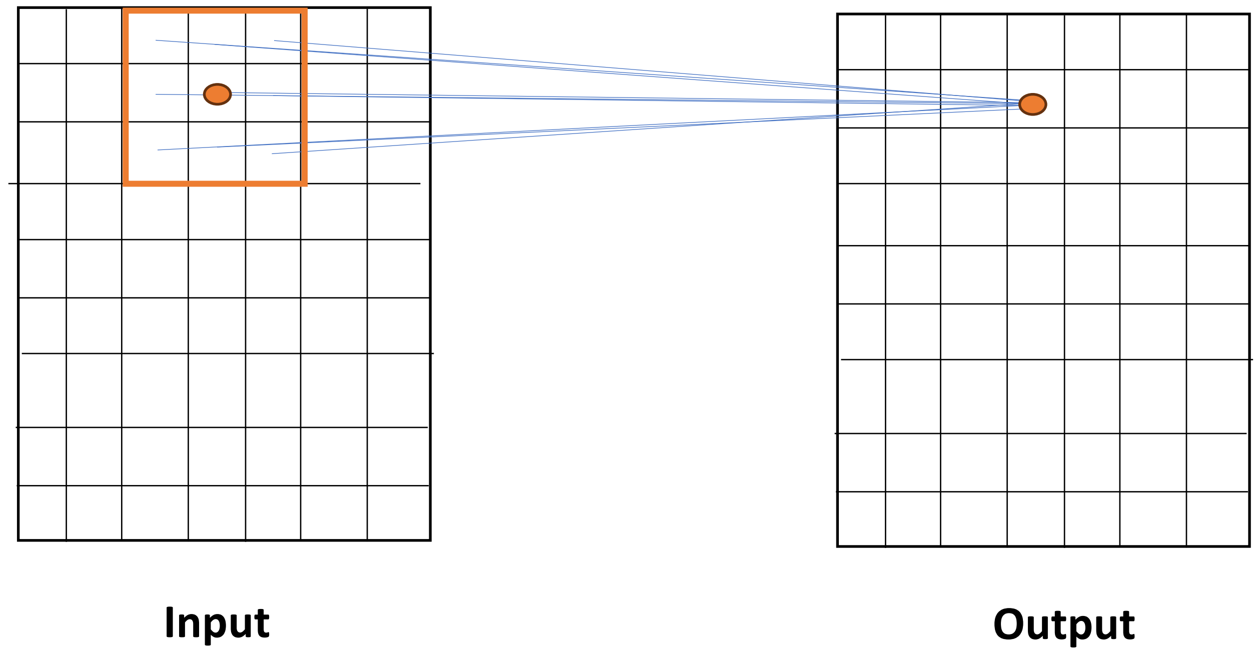
Images

- For 128×128 image $N \sim 10^4$ pixels

- For one dense layer, there are $N^2 \sim 10^8$ weight parameters

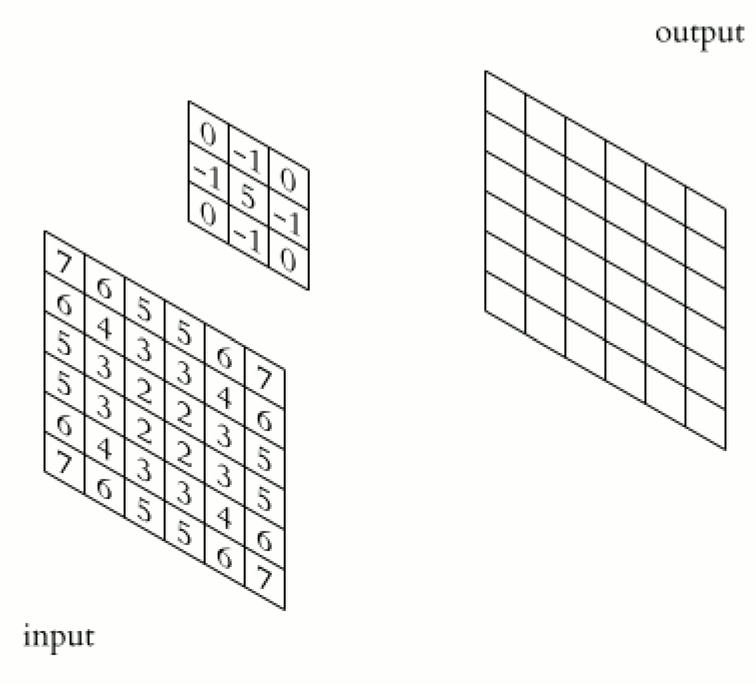
- CNN: only non-zero weights around point
- Weights sharing: same for every point
$$10^8 \rightarrow 9$$



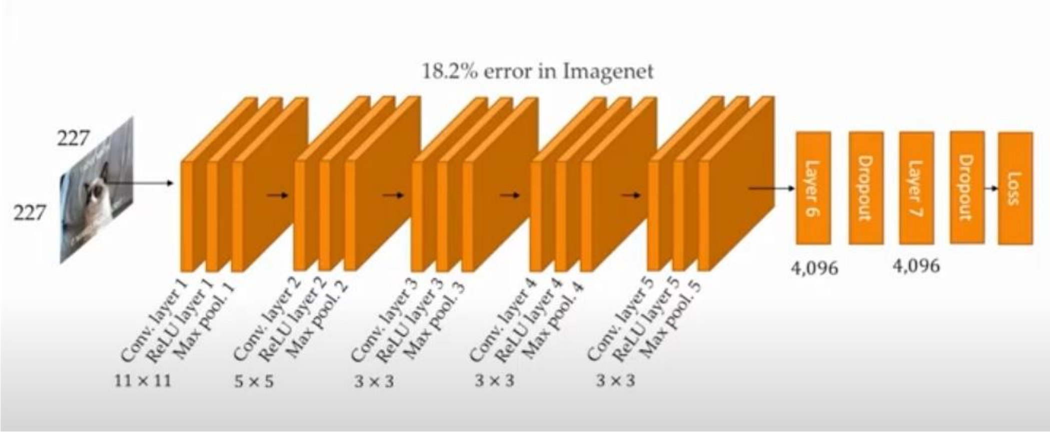


Convolutional Neural Network

- Uses locality
- Inspired by biology of vision



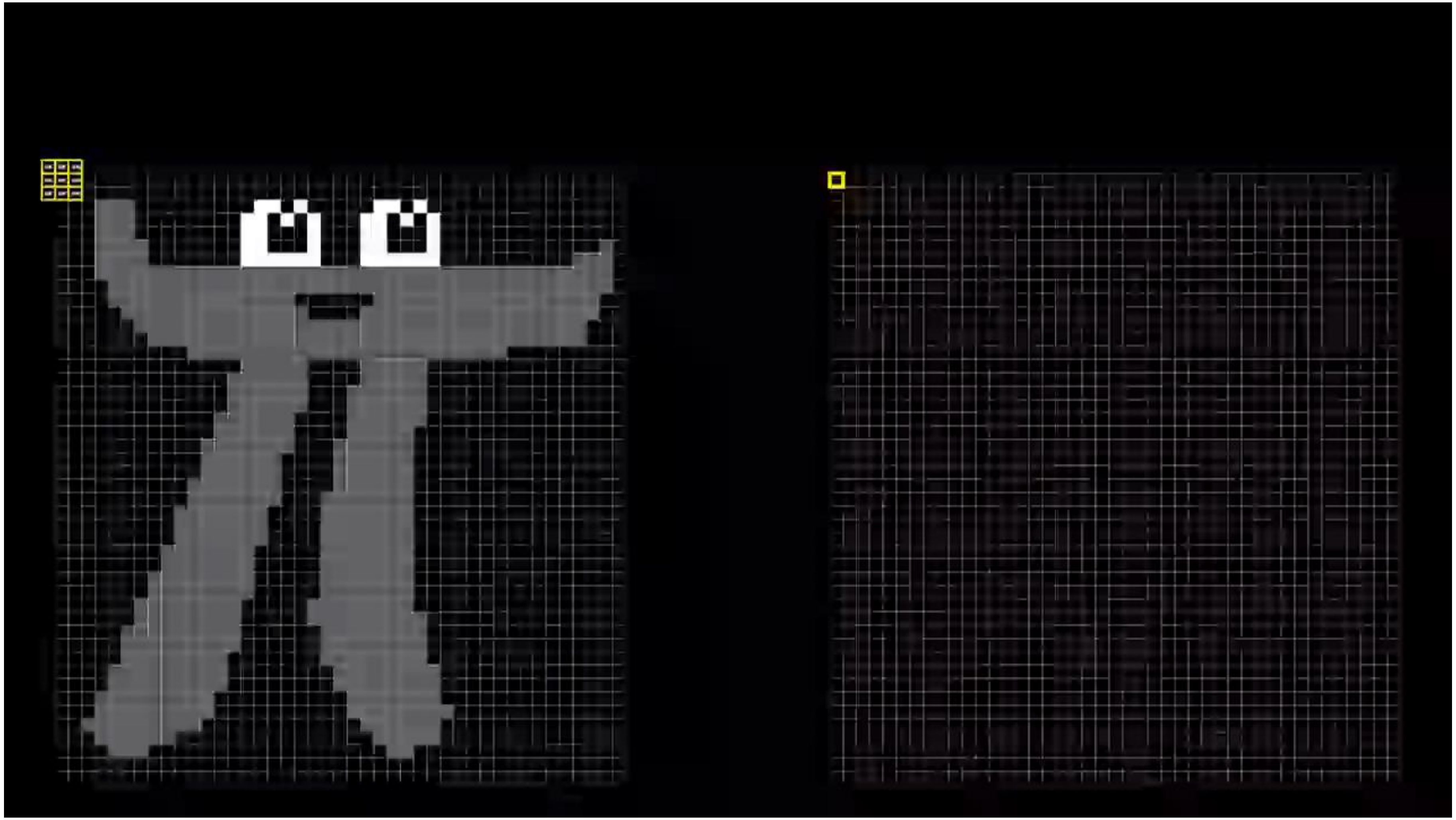
- **Filters** run convolutions on input. NN itself finds useful filters
- Filter output produce **feature maps** (very parallelizable).



- Translation Equivariance

- Output fed to non-linear functions. Process recursive.

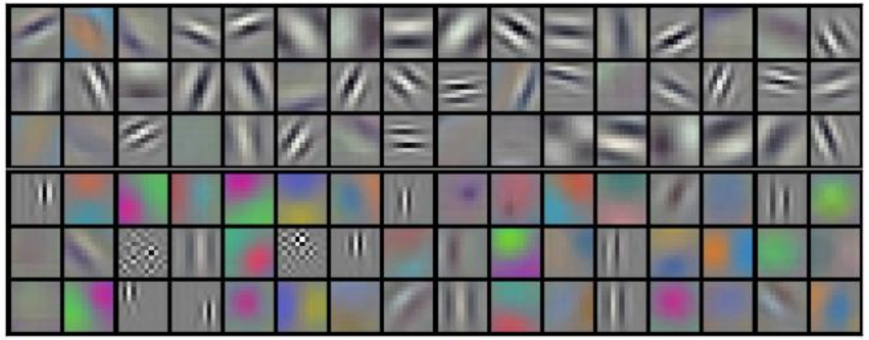
Example: Vertical Edge Filter



[But What Is a Convolution?](#)
3Blue1Brown (YouTube)

Convolutional Neural Network

- CNN **learns its own useful filters**

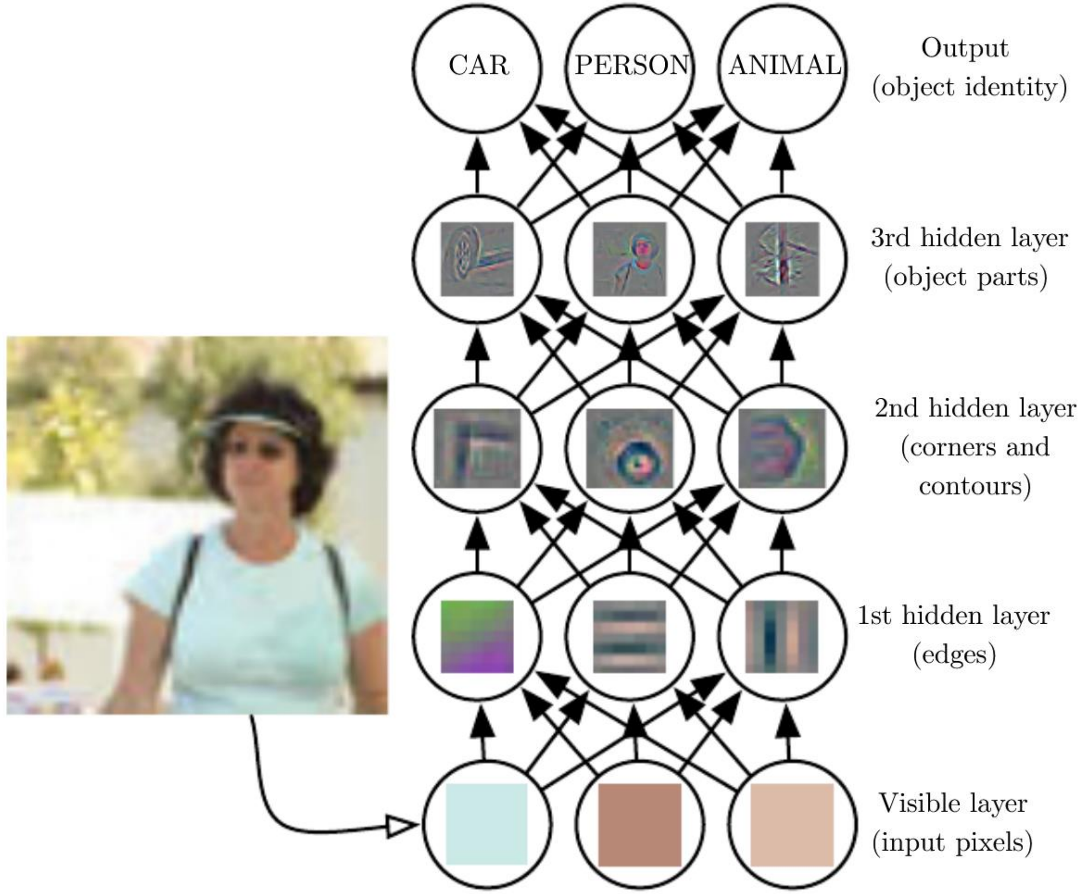


The 96 filters on first conv. layer learned by AlexNet
Krizhevsky, A et al. (2012)

Exercise: How many layers are needed for a 3x3 filter to get the same coverage as a 7x7? How many parameters are needed in each case?

Abstraction Through Depth

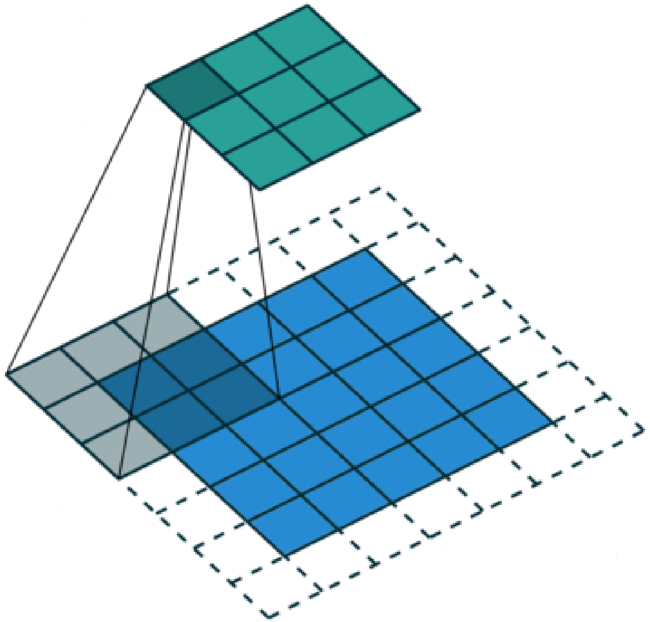
- Build complexity from ground up
- First layers: lower-level features
- Later layers: high level features
- **Machine learns filters by itself**



Zeiler and Fergus (2014).

Strides

- Lowering dimension can be achieved by strides > 1
- Do convolution, but skip pixels
- Example: stride of 2

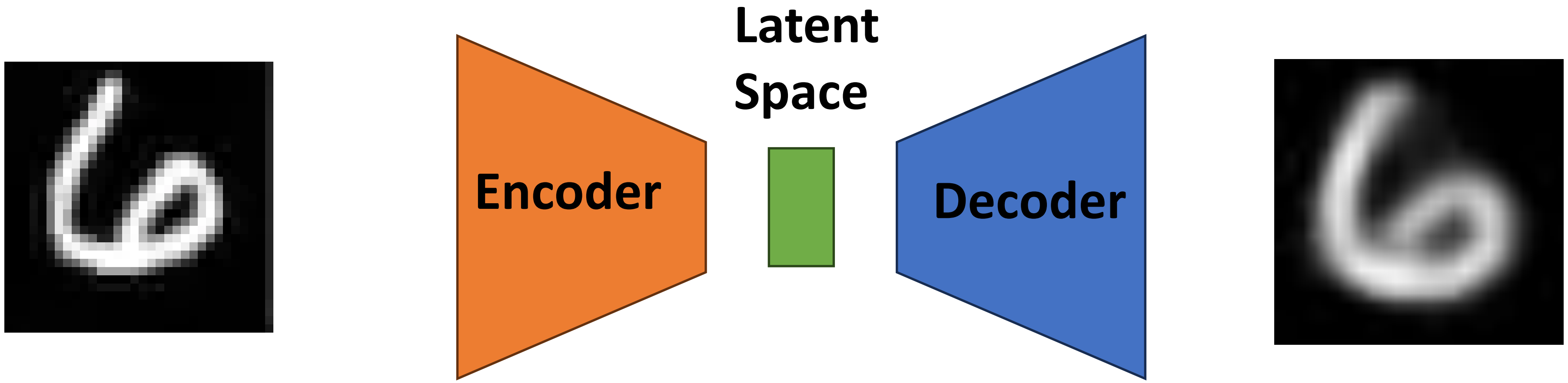


Exercise: Max pooling with stride 2 is another way to reduce the dimensions. Read on how Max Pool works.

Encoder-Decoder

- Example: MNIST database,
- $28 \times 28 = 784$ pixels
- Images in 784-dimensional space.

- Latent space: bottleneck



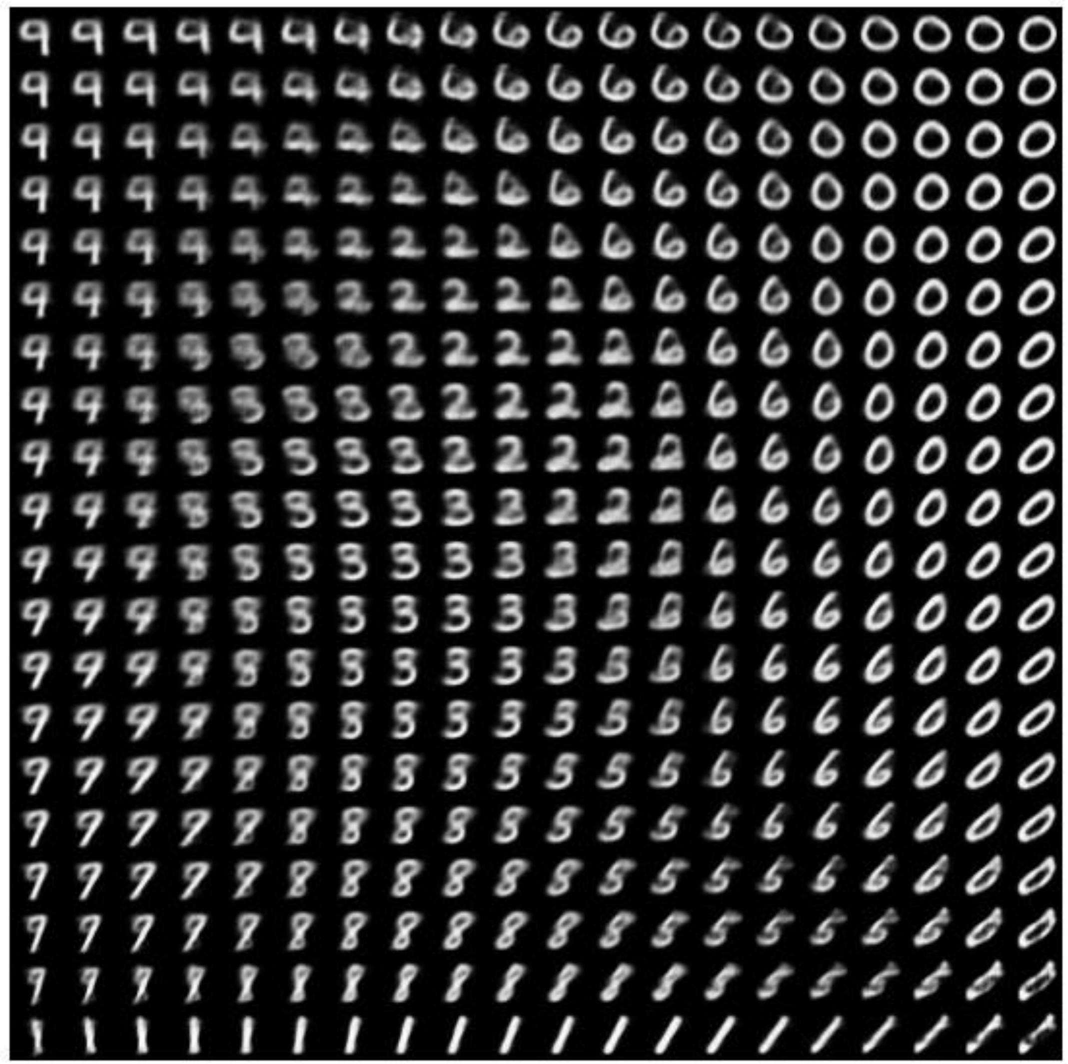
Latent Space

- Dimension reduction:
  $$784 \rightarrow 2$$

- Extract relevant features

- Easier to work in lower dimensional space

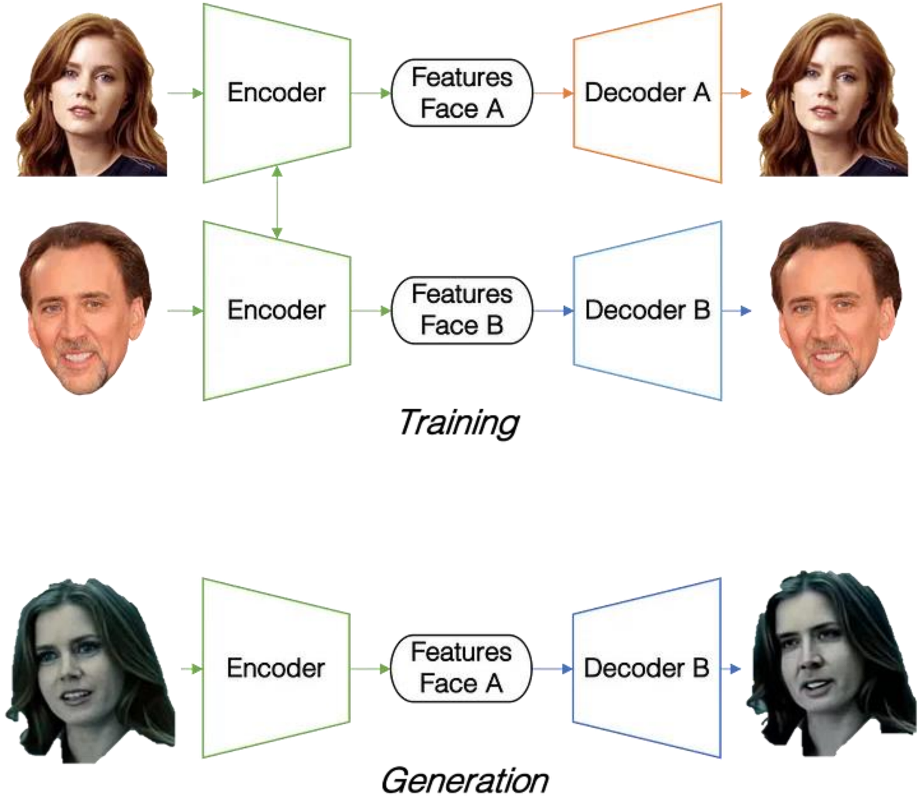
- Dimensions not always interpretable



TensorFlow: Variational Autoencoder Tutorial

CNN Encoder-Decoder

- CNN: Image → Image

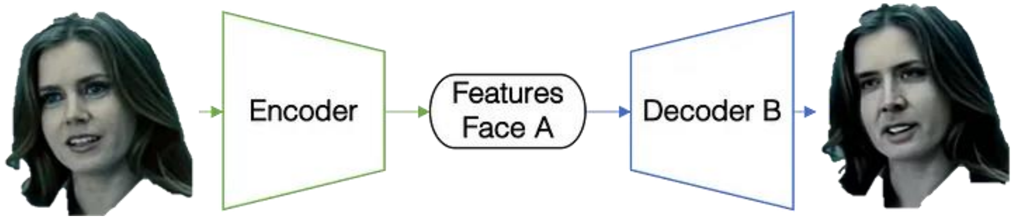


[Overview of CNN-Based Deepfake Detection Methods](#)
Medium – Colin Tan

CNN’s

- CNN: Image → Image

$$a_{ij} \rightarrow b_{ij}$$

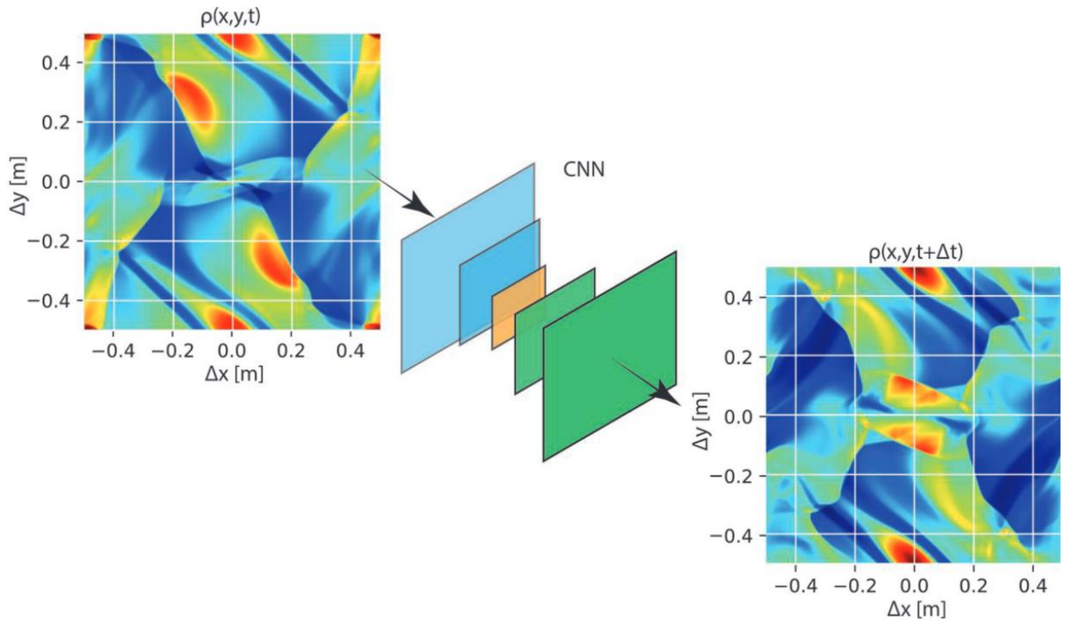


- Operator: Function → Function

$$f(x, y) \rightarrow g(x, y)$$

Example:

$$\rho(x, y, t) \rightarrow \rho(x, y, t + \Delta t)$$



Bormanis, CL, Scheinker
[Physics of Plasmas](#)

Solution Operators

Example: Poisson's equation

$$\nabla^2 \phi(\boldsymbol{r}) = \frac{\rho(\boldsymbol{r})}{\epsilon_0}$$

Solution:

$$\boxed{\phi(\boldsymbol{r}) = \frac{1}{4\pi\epsilon_0} \int d^3\boldsymbol{r}' G(\boldsymbol{r} - \boldsymbol{r}')\rho(\boldsymbol{r}')}$$

$$G(\boldsymbol{r} - \boldsymbol{r}') = \frac{1}{|\boldsymbol{r} - \boldsymbol{r}'|}$$

Solution operator

$$\rho \to \phi$$

Issue: singularities of $G(\boldsymbol{r} - \boldsymbol{r}')$.

Universal Operator Approximation Theorem

(Informal) A neural network can approximate any operator (Chen and Chen, 1995)

- Note: architecture is not necessarily a CNN encoder-decoder

- Operator can offer family of solutions to PDE’s

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 6, NO. 4, JULY 1995 911

Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems

Tianping Chen and Hong Chen

***Abstract*— The purpose of this paper is to investigate neural network capability systematically. The main results are: 1) every Tauber–Wiener function is qualified as an activation function in the hidden layer of a three-layered neural network, 2) for a continuous function in $S'(R^1)$ to be a Tauber–Wiener function, the necessary and sufficient condition is that it is not a polynomial, 3) the capability of approximating nonlinear functionals defined on some compact set of a Banach space and nonlinear operators has been shown, which implies that 4) we show the possibility by neural computation to approximate the output as a whole (not at a fixed point) of a dynamical system, thus identifying the system.**

Mhaskar and Micchelli [11] showed that under some restriction on the amplitude of a continuous function near infinity, any nonpolynomial function is qualified to be an activation function.

It is clear that all the aforementioned works are concerned with approximation to a continuous function defined on a compact set in R^n (a space of finite dimensions). In engineering problems such as computing the output of dynamic systems or designing neural system identifiers, however, we often encounter the problem of approximating nonlinear functionals

CNN Neural Operators

Advantages

- Learns family of solutions to PDE rather than a single instance
- Built-in translation symmetry (formally, translation equivariance)
- Fast at test time (filters + conv. highly parallelizable)

Disadvantages

- Harder to make fit PDE compared to PINN
- Mesh dependent (other NO methods can overcome this)
- Finer details get lost in compression to latent space (can be fixed with skipped connections)

Interesting Directions

Building in symmetries into NNs

- CNNs have translation symmetries
- Physical systems can also have rotational, scale, Galilean, gauge, etc. symmetries
- NN’s with built in symmetries can outperform regular NN’s
- Find symmetries through NNs

Other Neural Operator Approaches

- Deep O-Net
- Fourier Neural Operator

INCORPORATING SYMMETRY INTO DEEP DYNAMICS MODELS FOR IMPROVED GENERALIZATION

Rui Wang *
Computer Science and Engineering
University of California
San Diego, CA 92093
ruw020@ucsd.edu

Robin Walters *
Khoury College of Computer Science
Northeastern University
Boston, MA 02115
r.walters@northeastern.edu

Rose Yu
Computer Science and Engineering
University of California
San Diego, CA 92093
roseyu@ucsd.edu

arXiv: 2002.03061

Automatic Symmetry Discovery with Lie Algebra Convolutional Network

https://proceedings.neurips.cc/paper_files/paper/2021/file/148148d62be67e0916a833931bd32b26-Paper.pdf

Notebook

- We have a lattice (x_i, y_j) at different times t_k
- Dataset: $\rho(t_k, x_i, y_j)$

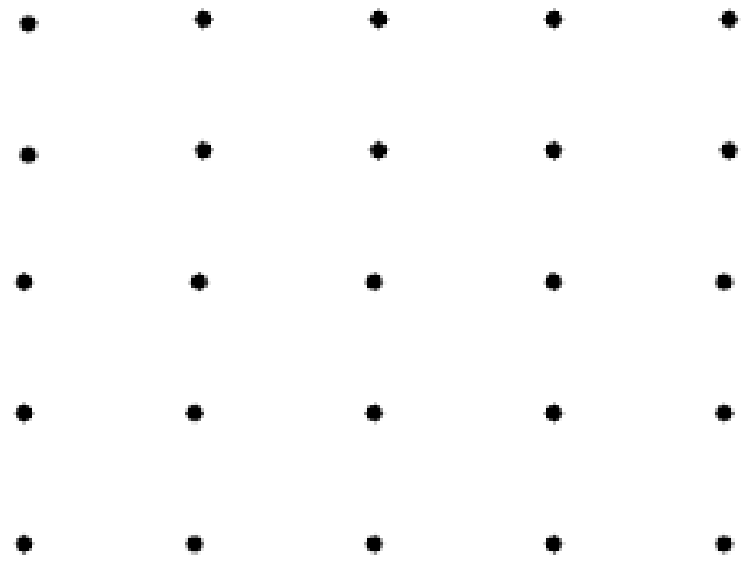
$$\rho[time, xindex, yindex\ , dummy]$$

Where dummy =1.

For 2D magnetic fields $\mathrm{B}^{\mathrm{l}}(t_k, x_i, y_j)$:

$$B[time, xindex, yindex\ , l]$$

where l = 0,1 (x or y).



Resources

Good coding practices

- The Good Research Code Handbook (online)

Neural Networks in Nuclear Physics

- Boehnlein, Amber, et al. "Colloquium: Machine learning in nuclear physics." *Reviews of Modern Physics* 94.3 (2022): 031003.

Tensorflow Tutorials

- TensorFlow 2 quickstart for beginners
- Convolutional Neural Network (CNN)

Resources - Basics

General

- Textbook: Goodfellow, Bengio, and Courville. *Deep learning*. 2016.
  - Available online, with lectures
  - Topics listed below can be found here

Neural Networks

- But what is a neural network? | 3Blue1Brown (YouTube, Chapter 1 of 4 in ML series)

CNNs

- But what is a convolution? | 3Blue1Brown (YouTube)
- Stanford University: Introduction to Convolutional Neural Networks for Visual Recognition (YouTube, whole course)

Resources – Advanced Topics

PINNs

- Raissi, Maziar, Paris Perdikaris, and George E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." *Journal of Computational physics* 378 (2019): 686-707.

NN Optimization

- Smith, Samuel L., et al. "Don't decay the learning rate, increase the batch size." *arXiv preprint arXiv:1711.00489* (2017).

Uncertainty Quantification in NNs

- Gal, Yarin, and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." international conference on machine learning. PMLR, 2016.

Neural Operators

- Kovachki, Nikola, et al. "Neural operator: Learning maps between function spaces." *arXiv preprint arXiv:2108.08481* (2021).

Exercises

Exercise: Batch normalization layers have been found to speed training up greatly. Finding out what 'batch normalization' is.

Note: it's still not theoretically understood why it works.

Exercise: Compare regular CNN architectures to ResNet and U-Net architectures. What are the advantages of skipped connections?

Backup Slide: More on PINNs

- Usually require fewer layers:

- Wide often better than deep

- Activation function often arctan and sigmoid. Disadvantage of ReLU: no differentiable at 0.

Backup: Channels Filters

- Three colors channels: RBG
- For a 128x128 pixels:

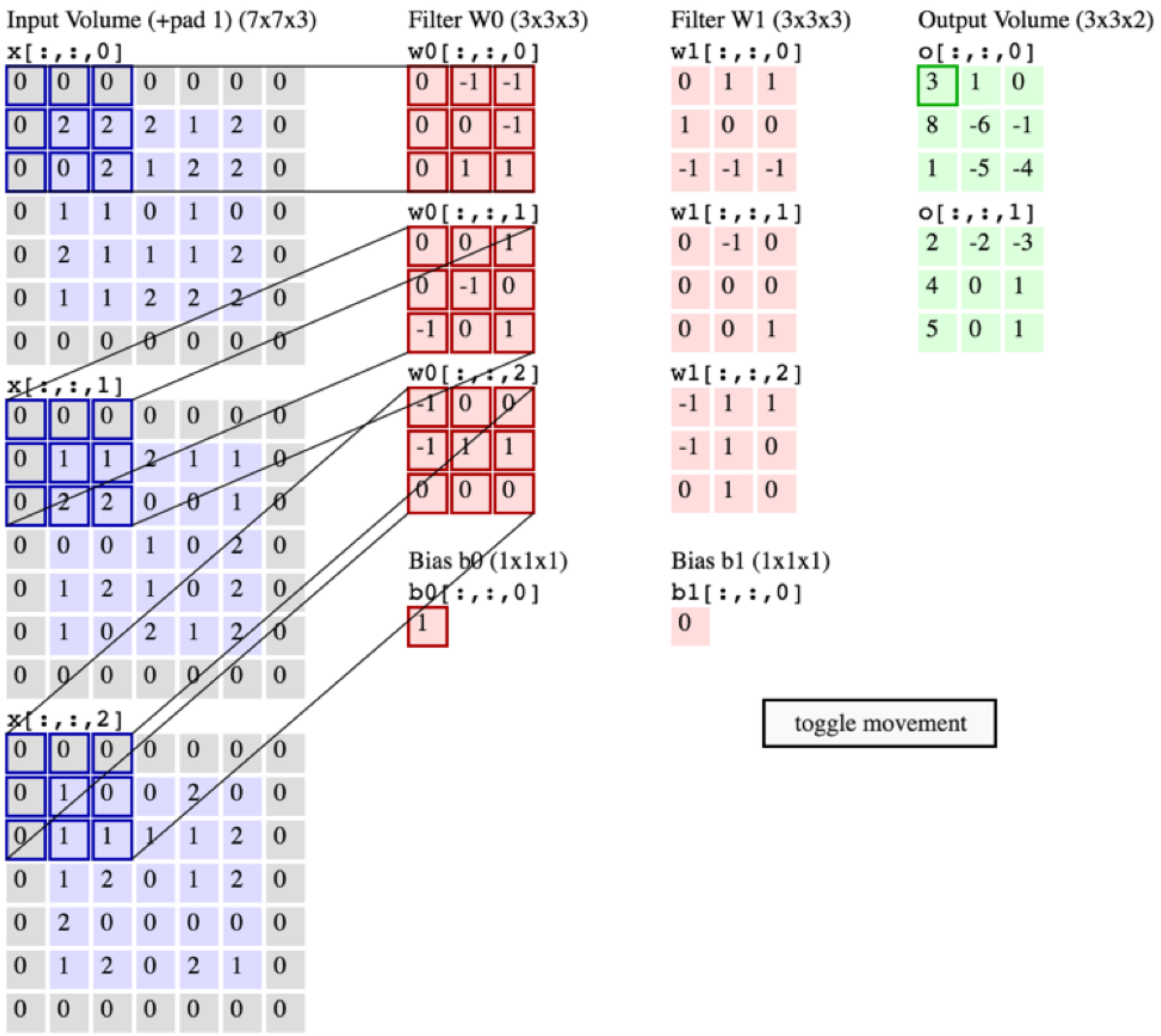
image is an 128x128x3 tensor (a_{ijc})

- 3x3 filter (for example) is needed for EACH color index

  $\Rightarrow$ **Filter is a 3x3x3 tensor** (f_{ijc})

Example: using 2 3x3 filters on color image

- Tensor approach makes parallelization easier



Backup: Differentiation in Tensorflow

- Derivates:

- Creating a full dense network

- Manual gradient descent

- Tensorflow: much goes on “under the hood”. User doesn’t need to worry.

Exercise: Find out what is “momentum” in the context of SGD. Why does it help?

Read about Adam (adaptive moment estimation) optimization method. This is the most commonly used optimization technique.