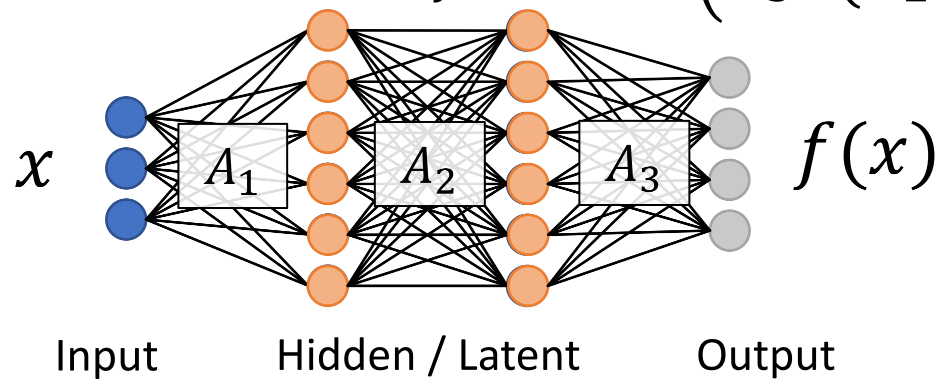
Basics of Deep Learning

David I. Inouye

What is deep learning? Sequential transformations learned from data

- Classical deep neural networks $f(x) = \sigma\left(A_3\sigma\left(A_2\sigma(A_1 x)\right)\right)$



- More generally, deep models are sequential transformations:

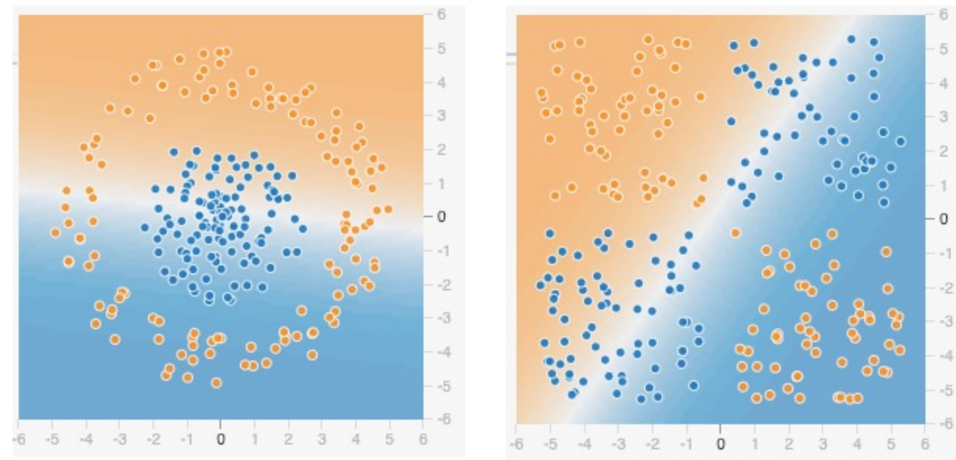
$$f(x) = f_3\left(f_2\left(f_1(x)\right)\right)$$

  - $z^{(1)} = f_1(x)$ (Layer 1)
  - $z^{(2)} = f_2\left(z^{(1)}\right)$ (Layer 2)
  - $z^{(3)} = f_3\left(z^{(2)}\right)$ (Layer 3)

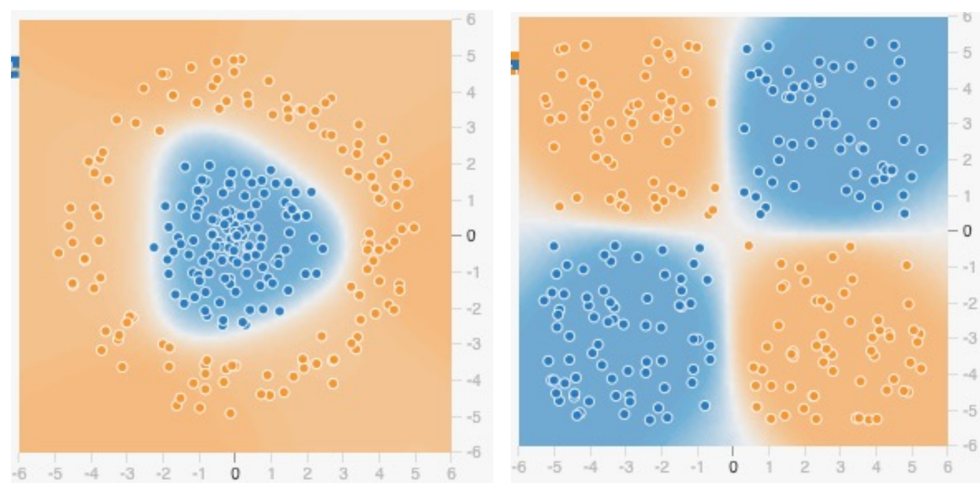
- Deep learning estimates these transformations from data

Motivation 1: Linear models cannot model complex classification boundaries

- Linear models cannot capture complex patterns



- With deep neural network, we can capture non-linear patterns

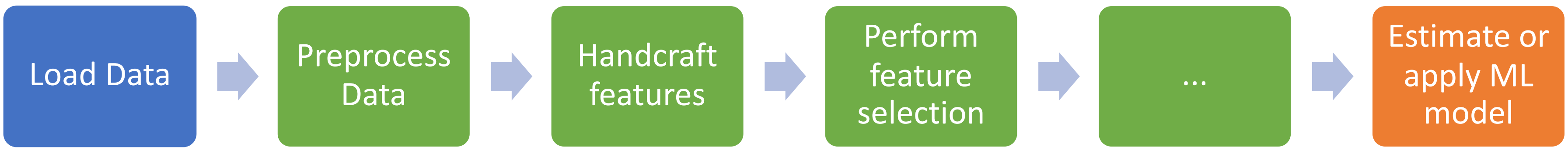


https://playground.tensorflow.org/

Motivation 2: Hand crafting features can increase performance but is expensive

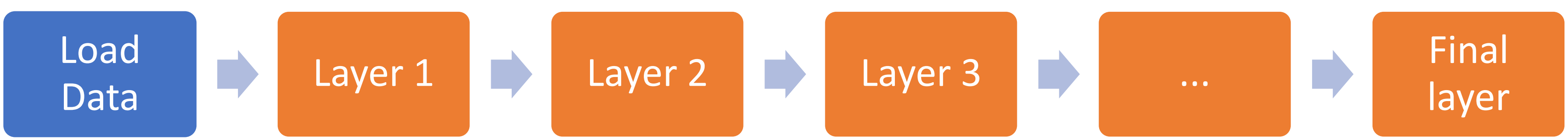
Classical Machine Learning

Feature engineering



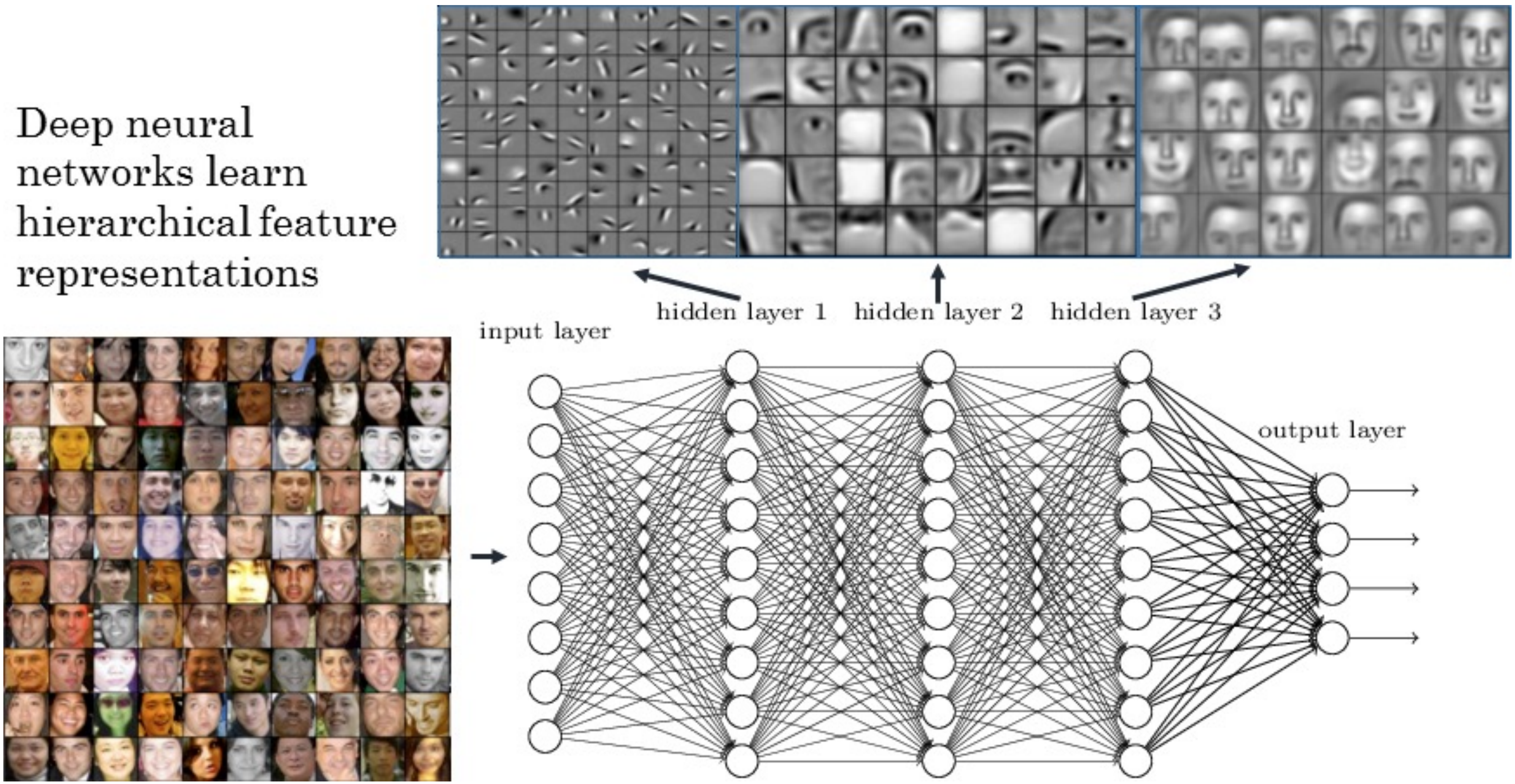
Deep Learning

Let the deep model do all the feature engineering automatically! :-)



Caveat: But now you have to select the model architecture (a little like feature engineering).

Motivation 3: Deep learning can automatically learn a hierarchy of representations



https://towardsdatascience.com/a-road-map-for-deep-learning-b9aee0b2919f

The key design choices of deep learning are architecture, algorithm, and objective function

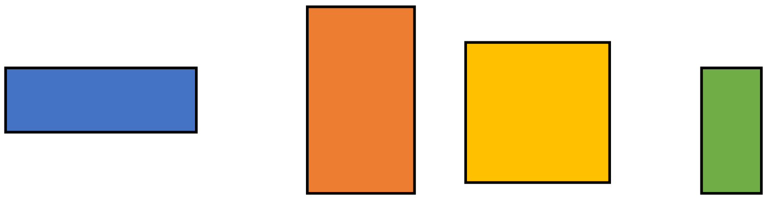
1. Deep model **architecture**

2. Deep learning optimization **algorithm**

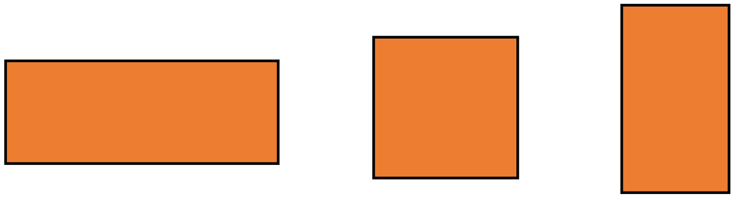
3. Deep learning **objective function** design
   - (Application specific so we will discuss later with GANs, VAEs, etc.)

The **model architecture** defines the structure of the model (though not parameter values)

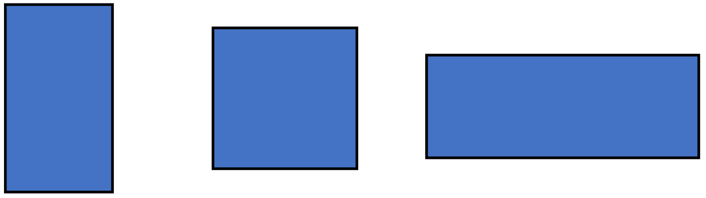
- Which layers or modules?
  - Fully connected
  - Convolutional
  - Residual blocks
  - …



- How big?
  - What is the dimensions of the input and output?



- How many and in what order?



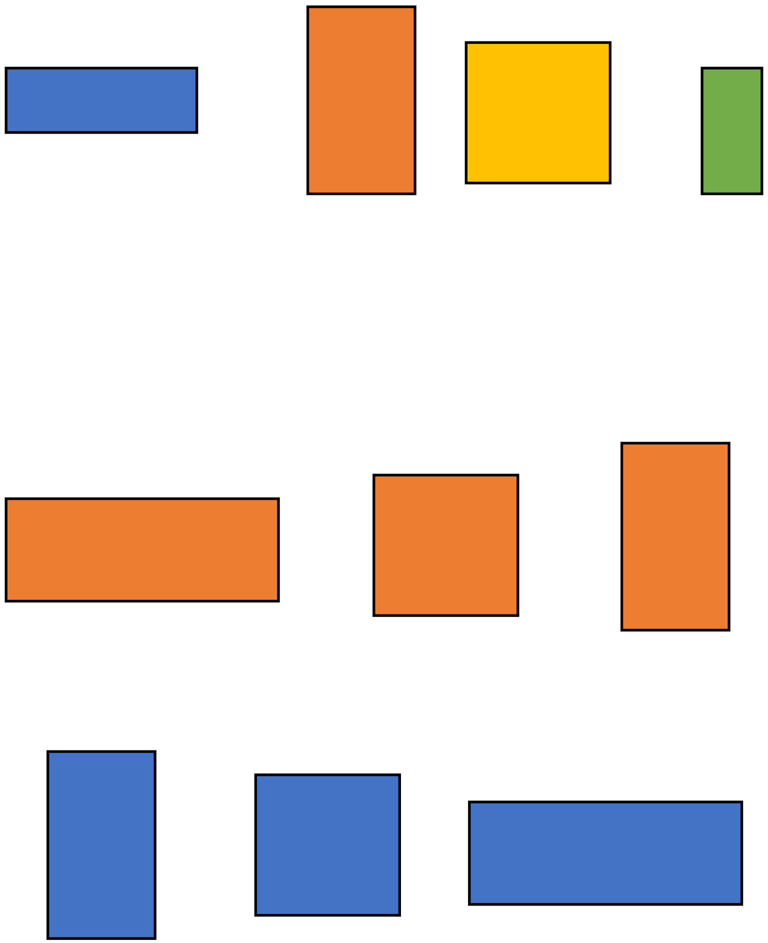
The architecture defines the <u>inductive bias</u> of the model

- **<u>Inductive bias</u>** is the bias of the model to perform better on certain problems

- A modern view of the "No Free Lunch Theorem"

- Example: Convolutional networks perform very well on image data

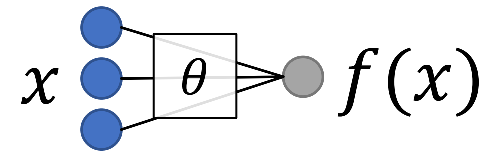
- Example: Attention-based "Transformer" networks have proven particularly successful for sequence data



<u>Fully connected layers</u> are linear functions followed by elementwise **non-linear** activation functions

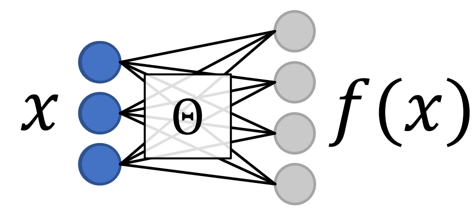
- Remember logistic regression:

  $$f(x) = \sigma(\theta^T x)$$



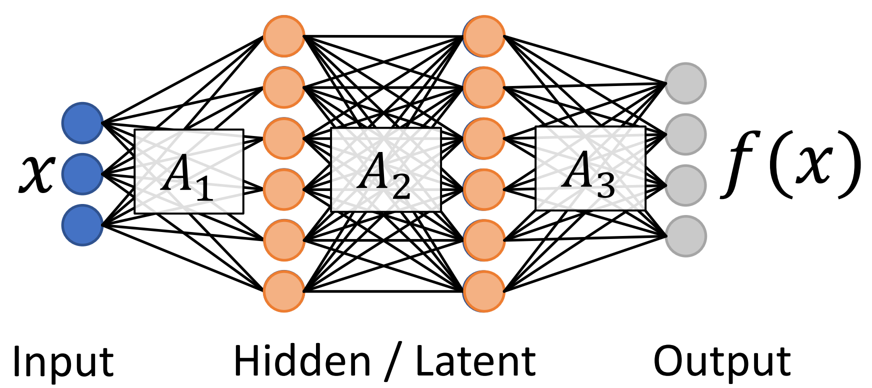
- A fully connected layer can be seen as multiple logistic regressions:

  $$f_{FC}(x) = \left[\sigma(\theta_1^T x), \cdots, \sigma\left(\theta_k^T x\right)\right]$$



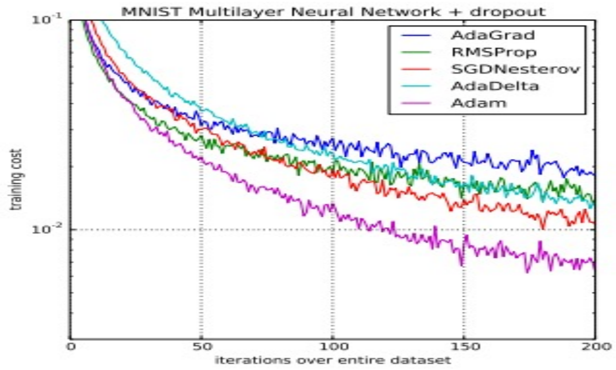
- A deep fully connected network is multiple fully connected layers:

  $$f(x) = \sigma\left(A_3 \sigma\left(A_2 \sigma(A_1 x)\right)\right)$$

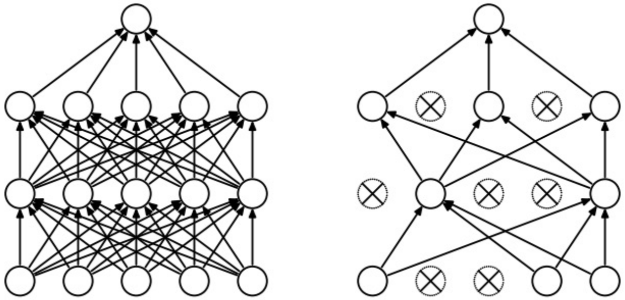


The <u>optimization algorithm</u> defines how the parameters will be updated

- Optimizer
  - SGD, ADAM, etc.
  - Step size



- Special "optimization" layers
  - BatchNorm
  - Dropout



(a) Standard Neural Net
(b) After applying dropout.

- Order of optimization updates
  - Example: Multiple inner optimization problems (e.g., adversarial optimization, GAN)

Automatic differentiation enables decoupling between architecture design and algorithm

- All computation can be broken into simple components
  - Examples: sum, multiply, exponential, convolution
- Derivatives can be derived mathematically
- Derivatives for **any composition** can be derived via chain rule! ☺
- (Prof. Jeffrey Siskind was a pioneer in automatic differentiation, see https://www.jmlr.org/papers/volume18/17-468/17-468.pdf)

Reverse-mode automatic differentiation can be computed in almost the same time as the original computation itself!

- **Forward pass**: Original objective computation
$$\mathcal{L}(X, y; \theta) = \frac{1}{n} \sum_i \ell\left(y_i, f_k\left(\cdots f_2\left(f_1(x_i)\right)\right)\right)$$
- **Backward pass**: Compute gradient by stepping backwards through computation
$$\nabla_\theta \mathcal{L}(X, y; \theta)$$
  - Also called "backpropagation" algorithm since it backpropagates the derivative
- Amazingly, the cost of the forward and backward passes are **equal up to a constant**
- How many forward passes to approximate derivative via small finite differences?
- $O(M)$ where M is the number of parameters!

PyTorch and TensorFlow implement automatic differentiation directly

- Demo doing automatic differentiation