ANNs
○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Deep Learning

Prepared for the IMF Workshop on Computational Economics (Dec 2025)

John Stachurski

2025

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Topics

- What are artificial neural networks?

- How are they trained on data?

- Why is DL so successful?

Artificial neural networks (ANNs) are the core of deep learning

Deep learning $=$ training ANNs with multiple hidden layers

Major successes:

- Natural language processing

- Image recognition

- Speech recognition

- Games

- LLMs

History

- 1940s: McCulloch & Pitts create mathematical model of NN

- 1950s: Rosenblatt develops the perceptron (trainable NN)

- 1980s: Backpropagation algorithm enables training of MLPs

- 1990s: SVMs temporarily overshadow ANNs in popularity

- 2000s: Deep learning finds successes in large problems

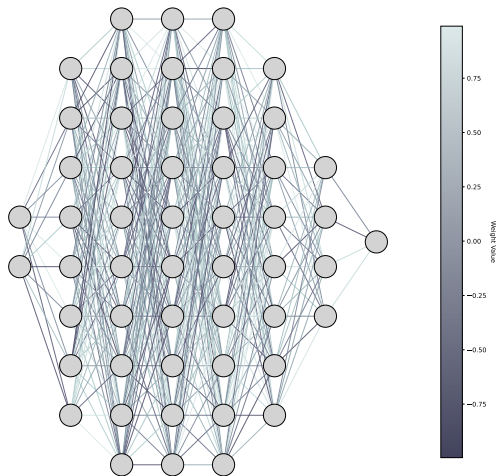**Last 10 years:** Explosion of progress in DL

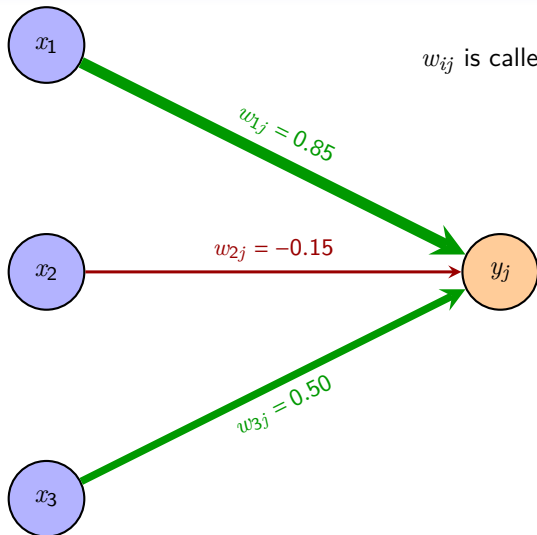- CNNs, RNNs, LSTMs, transformers, LLMs, etc.
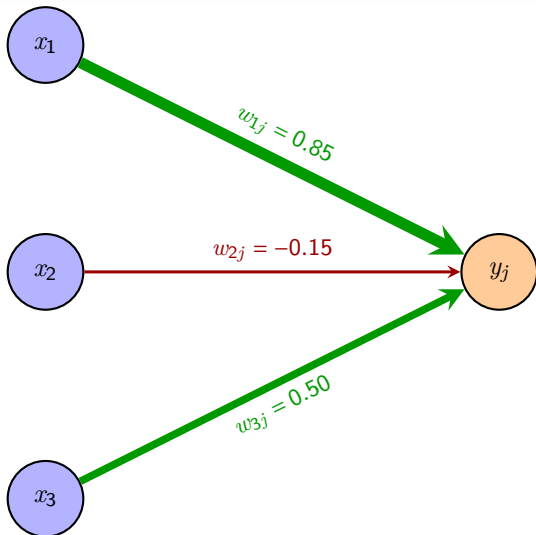
A model of the human brain

– source: Dartmouth undergraduate journal of science

ANNs
○○○○●○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

A mathematical representation: directed acyclic graph

ANNs
○○○○●○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

ANNs
○○○○○●○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

$$y_j = \sum_i x_i w_{ij}$$

ANNs
○○○○○○○●○○○○○○○
Training ANNs
○○○○○○○○○○○○○○○○○○○
DL with JAX
○○○○○○○○○

$$y_1 = \sum_i x_i w_{i1}$$

$$y_2 = \sum_i x_i w_{i2}$$

$$\implies y = xW$$

ANNs
○○○○○○○●○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Note that we are using row vectors: $y = xW$

This is natural given our notation

- w_{ij} points from i to j

- hence $y_j = \sum_i x_i w_{ij}$ (total flow of activation to node j)
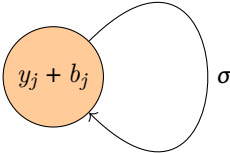
- hence $y = xW$

Also fine to use column vectors — just transpose W

ANNs
○○○○○○○○○●○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Next steps

After computing $y_j = \sum_i x_i w_{ij}$ we

1. add a bias term b_j and

2. apply a nonlinear "activation function" $\sigma \colon \mathbb{R} \to \mathbb{R}$

applying activation function $\left(y_j + b_j \right)$ σ

ANNs
○○○○○○○○○○●○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

First add bias:

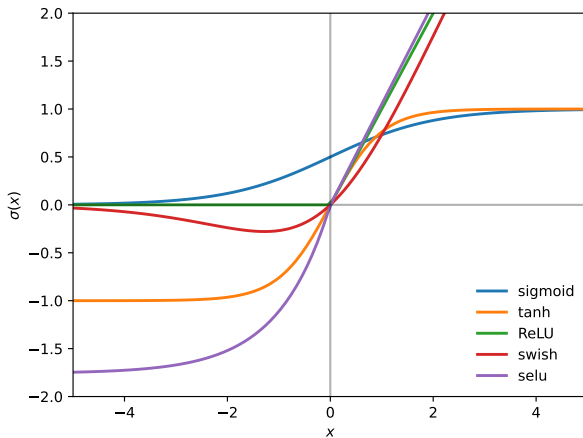$$y_j = \sum_i x_i w_{ij} \qquad \rightarrow \qquad y_j = \sum_i x_i w_{ij} + b_j$$

Then apply activation:

$$y_j = \sum_i x_i w_{ij} + b_j \qquad \rightarrow \qquad y_j = \sigma \left(\sum_i x_i w_{ij} + b_j \right)$$

Applying σ pointwise, we can write this in vector form as

$$y = \sigma(xW + b)$$

ANNs
○○○○○○○○○○○○●○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Common activation functions

ANNs
○○○○○○○○○○○○●○○

Training ANNs
○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Universal function approximation I

Theorem. (Cybenko–Hornik 1989, 1991) Let

- $K \subset \mathbb{R}^n$ be compact,
- f be a continuous functions from K to $\mathbb{R}^m$, and
- σ be a continuous map from $\mathbb{R}$ to itself

If σ is not polynomial, then, for every $\varepsilon > 0$, there exist matrices A, C and vector b such that

$$\sup_{x \in K} \|f(x) - C\sigma(Ax + b)\| < \varepsilon$$

(The function σ is applied component-wise)

ANNs
○○○○○○○○○○○○○○●○

Training ANNs
○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Universal function approximation II

Recall that the ReLU activation has the form $\sigma(x) = \max\{x, 0\}$

Theorem. Let

- $f \colon \mathbb{R}^n \to \mathbb{R}^m$ be Borel measurable and
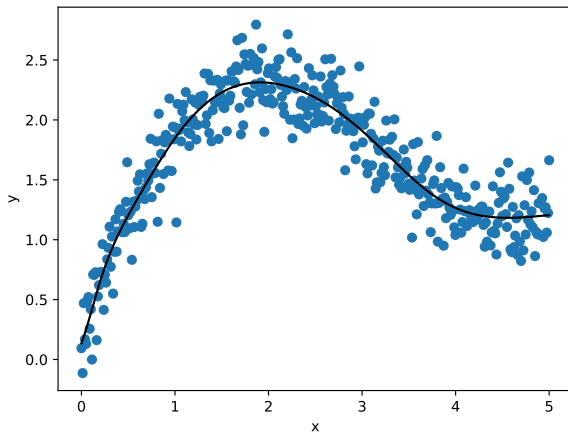- $d = \max\{n + 1, m\}$

If f is Bochner–Lebesgue integrable, then, for any $\varepsilon > 0$, there exists a fully connected ReLU network g of width d satisfying

$$\int \|f(x) - g(x)\| \, \mathrm{d}x < \varepsilon$$

ANNs
○○○○○○○○○○○○○○○●

Training ANNs
○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Before we get too excited: many function classes have the universal
function approximation property

- polynomials in $C_0(K) :=$ continuous functions on compacts

- finite linear combinations of an ONS in a Hilbert space

- SVMs with RBF kernels in reproducing kernel Hilbert space

- Gaussian processes with RBF kernels in $C_0(K)$

- random forests

- etc.

ANNs
○○○○○○○○○○○○○○○○

Training ANNs
●○○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Training

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○●○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Aim: Learn to predict output y from input x

- $x \in \mathbb{R}^k$

- $y \in \mathbb{R}$ (regression problem)

Egs.

- x = cross section of returns, y = return on oil futures tomorrow

- x = weather sensor data, y = max temp tomorrow

Problem:

- observe $(x_i, y_i)_{i=1}^n$ and seek f such that $y_{n+1} \approx f(x_{n+1})$

Nonlinear regression: choose model $\{f_\theta\}_{\theta\in\Theta}$ and minimize the empirical loss

$$\ell(\theta) := \frac{1}{n}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

In the case of ANNs, we consider all f_θ having the form

$$f_\theta = G_m \circ G_{m-1} \circ \cdots \circ G_2 \circ G_1$$

where

- $G_\ell(x) = \sigma_\ell(xW_\ell + b_\ell)$

- $\theta := \{W_1, b_1, W_2, b_2, \ldots, W_m, b_m\}$

- σ_ℓ is an activation function

Nonlinear regression: choose model $\{f_\theta\}_{\theta \in \Theta}$ and minimize the empirical loss

$$\ell(\theta) := \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$
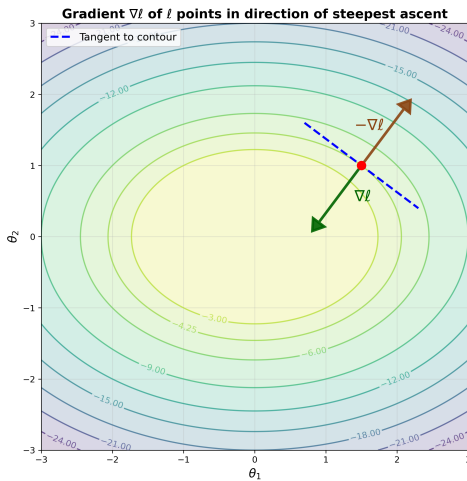
In the case of ANNs, we consider all f_θ having the form

$$f_\theta = G_m \circ G_{m-1} \circ \cdots \circ G_2 \circ G_1$$

where

- $G_\ell(x) = \sigma_\ell(x W_\ell + b_\ell)$

- $\theta := \{ W_1, b_1, W_2, b_2, \ldots, W_m, b_m \}$

- σ_ℓ is an activation function

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○●○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Minimizing the loss functions

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○○●○○○○○○○○○○○○

DL with JAX
○○○○○○○○○

Gradient descent

Algorithm: Implement $\nabla\ell$ and then update guess θ_k via

$$\theta_{k+1} = \theta_k - \lambda_k \cdot \nabla\ell(\theta_k)$$

- take a step in the opposite direction to the grad vector

- λ_k is the **learning rate**

- iterate until hit a stopping condition
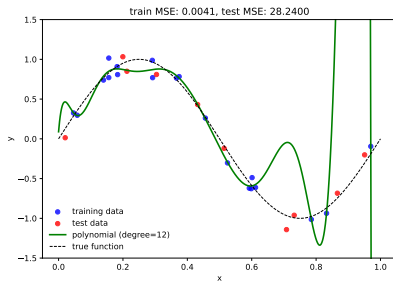
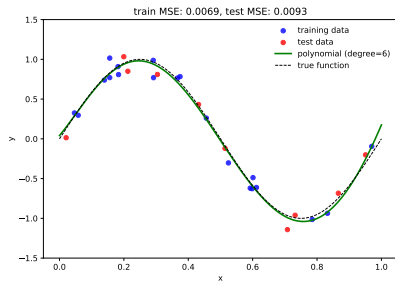- in practice replace $\ell(\theta)$ with batched loss $\rightarrow$ **SGD**
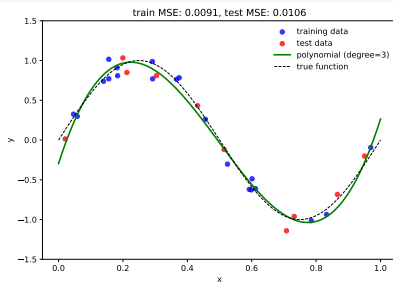
$$\frac{1}{|B|} \sum_{i \in B} (y_i - f_\theta(x_i))^2$$

Extensions

- Many variations on SGD

- Loss functions with regularization

- Cross-entropy loss (classification)

- Convolutional neural networks (image processing)

- Recurrent neural networks (sequential data)

- Transformers (LLMs)

- etc.

ANNs
ooooooooooooooo

Training ANNs
oooooooooooooooooo

DL with JAX
oooooooooo

A mystery

What about overfitting?

ANNs
○○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○●○○○○○○○○○

DL with JAX
○○○○○○○○○

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○●○○○○○○○

DL with JAX
○○○○○○○○○

model complexity

ANNs
0000000000000

Training ANNs
00000000000●0000000

DL with JAX
000000000

If production-level DL models are so large, why don't they overfit?

Answer 1 Underlying relationships are complex – need complex model

Answer 2 Engineers avoid using full complexity of the model

- Early stopping halts training when test loss starts to rise

- Drop out – shut down some neurons during training

ANNs
○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○●○○○○○○

DL with JAX
○○○○○○○○○

Answer 3 Adding randomization to training prevents overfitting

- Related to benefits of random dropout (randomization)

- Related techniques in DL such as DropConnect and Stochastic Depth

- Stochastic gradient descent injects randomness

Randomness "adds some smoothing" to a given data set

"The model must learn to be robust to these perturbations, which encourages it to find smoother, more generalizable decision boundaries rather than fitting to noise in the training data."

ANNs
○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○●○○○○○○

DL with JAX
○○○○○○○○○

Answer 4 Modern architectures have inductive biases

Egs.

- Translation invariance in CNNs

- Localization in CNNs – pixels influenced more by neighbor pixels

- Parameter sharing in RNNs – similarity of transformations across time

Injection of good priors guides learning

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○●○○○○

DL with JAX
○○○○○○○○○

Summary

Why can DL successfully generalize?

CS story

Because

- based on a model of the human brain!

- A universal function approximator!

- Can break the curse of dimensionality!

Really?

ANNs
0000000000000000

Training ANNs
00000000000000●000

DL with JAX
000000000

CS story

Because

- based on a model of the human brain!

- A universal function approximator!

- Can break the curse of dimensionality!

Really?

Claude AI

"Traditional methods typically require you to

- specify the intrinsic dimension,

- choose kernel parameters, or

- make structural assumptions about the manifold

Neural networks automatically adapt to intrinsic structure without requiring this prior knowledge"

Really?

Claude AI

"Traditional methods typically require you to

- specify the intrinsic dimension,

- choose kernel parameters, or

- make structural assumptions about the manifold

Neural networks automatically adapt to intrinsic structure without requiring this prior knowledge"

Really?

Alternative story

- Easy to understand with limited maths background

- Scales well to high dimensions

- Function evaluations are highly parallelizable

- Flexible – can inject inductive biases

- Smooth recursive structure suited to calculating gradients

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○●

DL with JAX
○○○○○○○○○

On top of — because of — this:

- Has received massive investment from the CS community

  - algos

  - software

  - hardware

- Many incremental improvements to improve regularization

- Steady increase in injection of domain-specific knowledge

  - CNNs

  - RNNs

  - transformers, etc.

ANNs
⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾

Training ANNs
⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾⦾

DL with JAX
●⦾⦾⦾⦾⦾⦾⦾⦾⦾

DL with JAX

"JAX is a Python library designed for high-performance numerical computing, especially machine learning research."

"Its API for numerical functions is based on NumPy."

"Both Python and NumPy are widely used and familiar, making JAX simple, flexible, and easy to adopt."

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○

DL with JAX
○●○○○○○○○

JAX is more general-purpose than PyTorch or TensorFlow

- PT and TF both explicitly designed for deep learning

- JAX is a platform for scientific computing designed with machine learning / AI applications in mind

DL in JAX on one slide:

ANNs
○○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○○

DL with JAX
○○●○○○○○○○

```python
def f(θ, x, σ=jnp.tanh):
    for W, b in θ:
        x = σ(x @ W + b)
    return x


def loss(θ, x, y):
  return jnp.sum((y - f(θ, x))**2)


loss_gradient = jit(grad(loss))


def train(θ, x_data, y_data):
    for i in range(m):
        θ = θ - λ * loss_gradient(θ, x_data, y_data)
    return θ
```

ANNs
0000000000000000

Training ANNs
0000000000000000

DL with JAX
000●00000

JAX PyTrees

Consider a function of the form

$$f_\theta = G_m \circ G_{m-1} \circ \cdots \circ G_2 \circ G_1$$

where

- $G_\ell x = \sigma_\ell(x W_\ell + b_\ell)$ for $\ell = 1, \ldots, m$

- θ represents the "vector" of all parameters

- σ_ℓ is a given function

The idea that the vector θ contains all parameters is conceptually useful but awkward within code...

ANNs
○○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○●○○○○

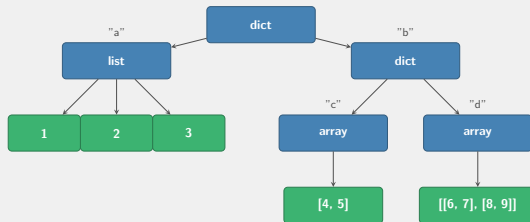To handle these kinds of situations we can use PyTrees

- A tree-like data structure built from Python containers

- A concept, not a data type

- Used to store parameters

Egs.

- A list of dictionaries, each dictionary contains parameters
- A dictionary of lists
- A dictionary of lists of dictionaries
- etc.

JAX PyTree Structure

```
pytree = {
    "a": [1, 2, 3],
    "b": {"c": jnp.array([4, 5]), "d": jnp.array([[6, 7], [8, 9]])}
}
```

ANNs
○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○●○○

JAX can

- apply functions to all leaves in a PyTree structure

- differentiate functions with respect to the leaves of PyTrees

- etc.

ANNs
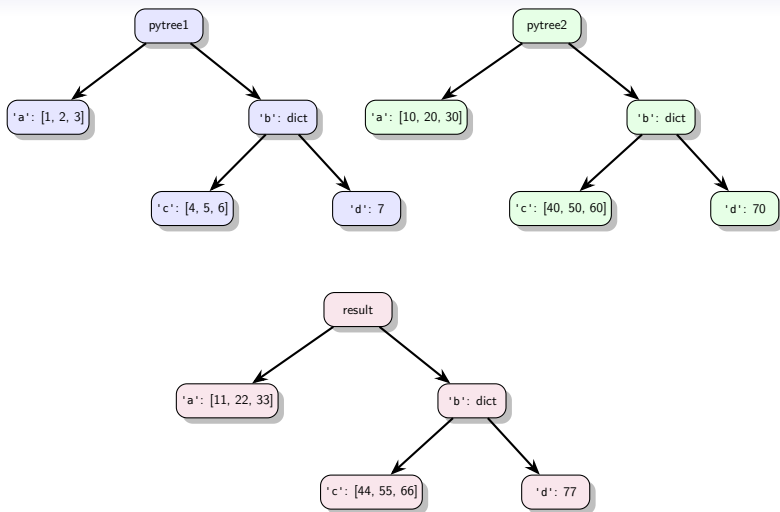○○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○●○

Figure: jax.tree.map(lambda x, y: x + y, pytree1, pytree2)

ANNs
○○○○○○○○○○○○○○○

Training ANNs
○○○○○○○○○○○○○○○○○○○

DL with JAX
○○○○○○○○○●

```python
# Apply gradient updates to all parameters
def sgd_update(params, grads, learning_rate):
    return jax.tree.map(
        lambda p, g: p - learning_rate * g,
        params,
        grads
    )

# Calculate gradients (PyTree with same structure as params)
loss_grad = jax.grad(loss_fn)
grads = loss_grad(params, x, y)

# Update all parameters at once
updated_params = sgd_update(params, grads, 0.01)
```