On Deep Learning

Alexander G. Ororbia II

Introduction to Machine Learning

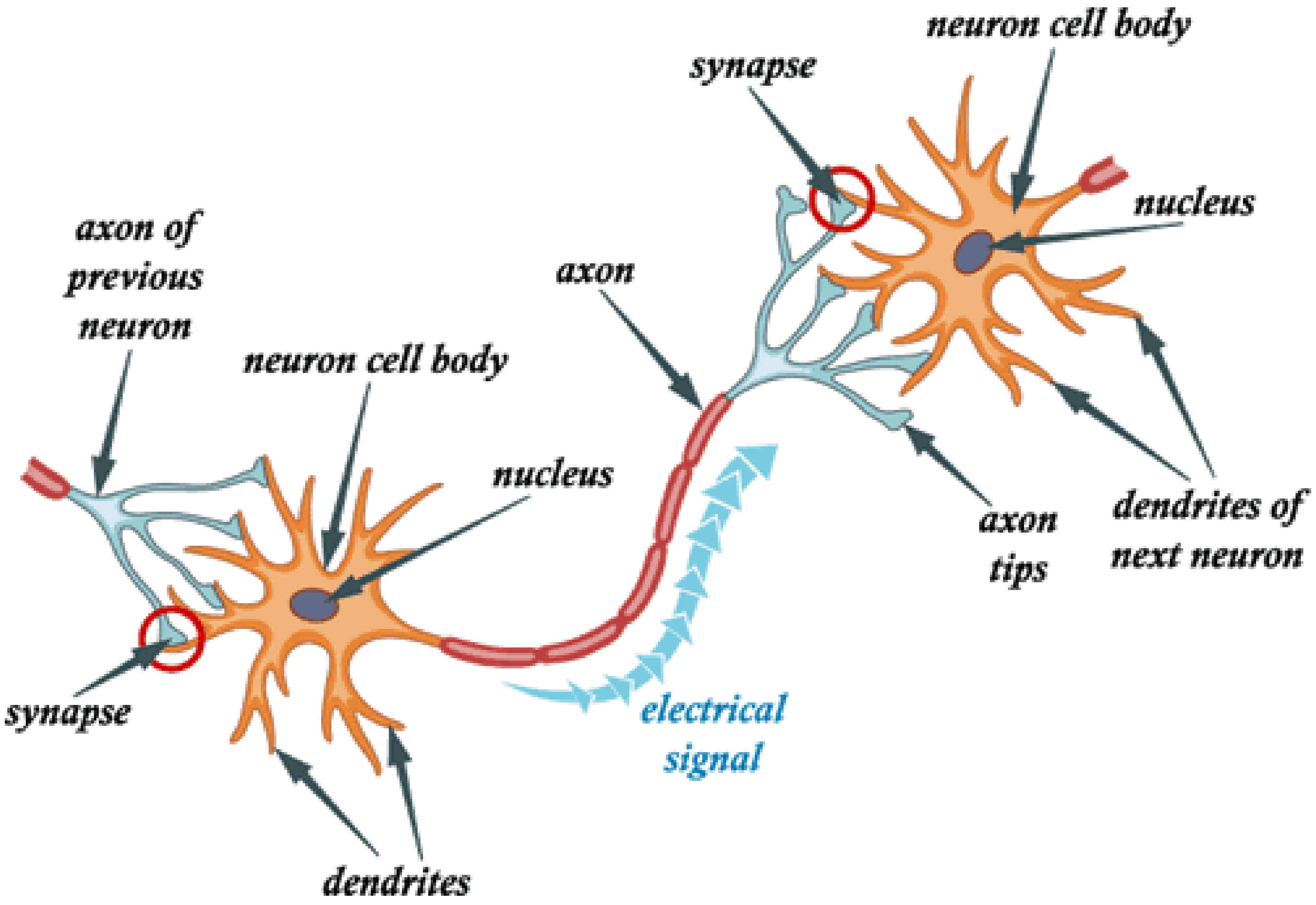
CSCI-736

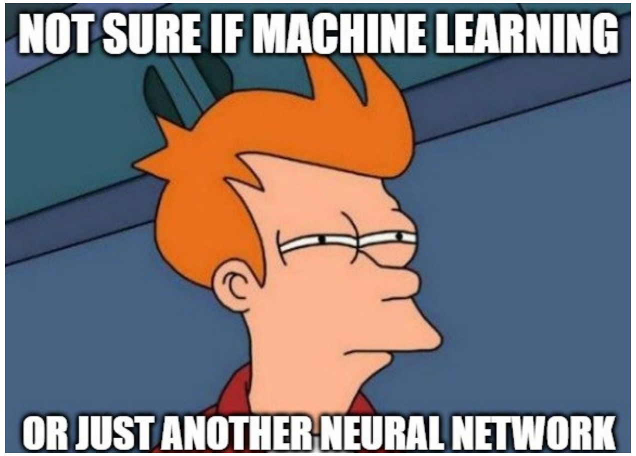
1/23/2025

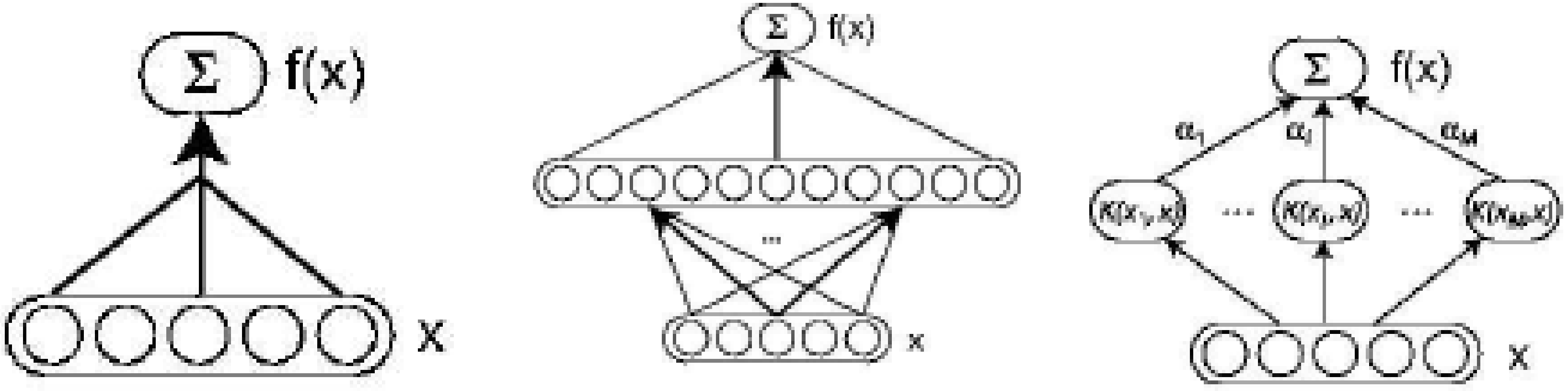
Companion reading:
Chapter 6-8 of Deep Learning textbook

Artificial Neural Networks (ANNs): Neurobiological Motivations

- **Human brain** = a good candidate learning algorithm
  - Evidence of layered architectures in neuro-scientific research (i.e., cortical structures)
- Early success of specialized yet deep architectures
  - Convolutional Networks, NeoCognitron





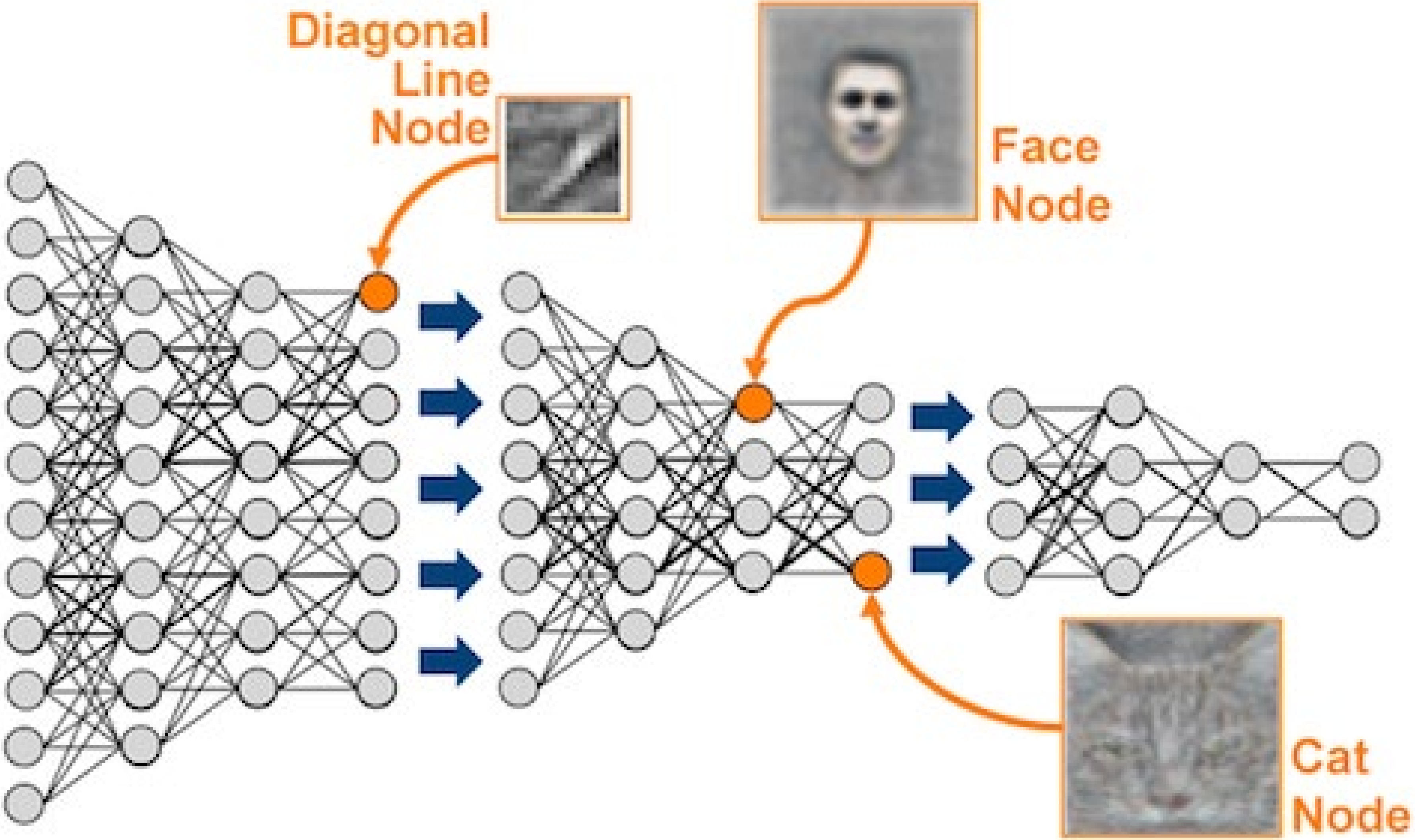


(a) Linear model architecture

(b) Single layer neural network architecture

(c) Kernel SVM architecture

Most of machine learning models can be viewed as a type of ANN…if you squint hard enough…



Diagonal Line Node
Face Node
Cat Node

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Background

A Recipe for

Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of th

– Decision functio

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

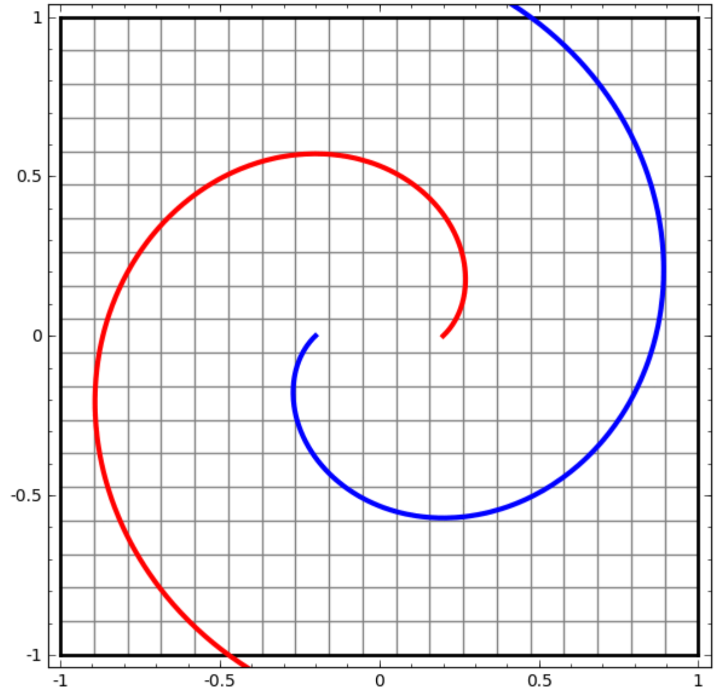
$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

opp site the gradient)

$$- \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Reverse Mode Differentiation

- Application of the chain-rule from (vector) calculus
- Can view ANNs at level of processing elements (PEs)—neuronal graph
  - Follow dot-arrow diagram to get partial derivative scalars
  - Limited flexibility, but simple to understand
- Can view this at lowest level—computation graph
  - Follow graph of operators & get partial derivatives using sub-rules (sum rule, product rule, etc.)
  - Highly flexible
  - Tools that do this:
    - Theano: http://deeplearning.net/software/theano/
    - TensorFlow: https://www.tensorflow.org/



Deep calculus!

Approaches to Differentiation

Given $f : \mathbb{R}^A \rightarrow \mathbb{R}^B, f(\mathbf{x})$

Compute $\dfrac{\partial f(\mathbf{x})_i}{\partial x_j} \forall i, j$

1. Finite Difference Method
   - Pro: Great for testing implementations of backpropagation
   - Con: Slow for high dimensional inputs / outputs
   - Required: Ability to call the function f($\mathbf{x}$) on any input $\mathbf{x}$
2. Symbolic Differentiation
   - Note: The method you learned in high-school
   - Note: Used by Mathematica / Wolfram Alpha / Maple
   - Pro: Yields easily interpretable derivatives
   - Con: Leads to exponential computation time if not carefully implemented
   - Required: Mathematical expression that defines f($\mathbf{x}$)
3. Automatic Differentiation - Reverse Mode
   - Note: Called *Backpropagation* when applied to Neural Nets
   - Pro: Computes partial derivatives of one output $f(\mathbf{x})_i$ with respect to all inputs x_j in time proportional to computation of f($\mathbf{x}$)
   - Con: Slow for high dimensional outputs (e.g. vector-valued functions)
   - Required: Algorithm for computing f($\mathbf{x}$)
4. Automatic Differentiation - Forward Mode
   - Note: Easy to implement. Uses dual numbers.
   - Pro: Computes partial derivatives of all outputs $f(\mathbf{x})_i$ with respect to one input x_j in time proportional to computation of f($\mathbf{x}$)
   - Con: Slow for high dimensional inputs (e.g. vector-valued $\mathbf{x}$)
   - Required: Algorithm for computing f($\mathbf{x}$)

The Finite Difference Method

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

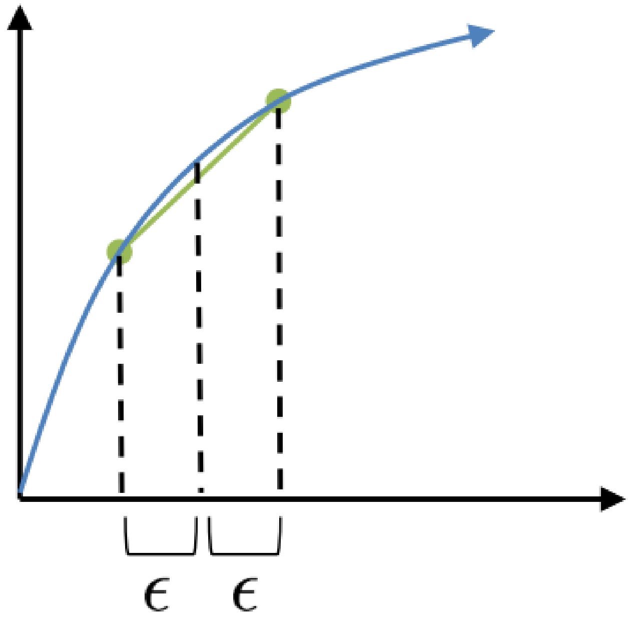
The *centered* finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{(J(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{d}_i) - J(\boldsymbol{\theta} - \epsilon \cdot \boldsymbol{d}_i))}{2\epsilon}$$

where $\boldsymbol{d}_i$ is a 1-hot vector consisting of all zeros except for the ith entry of $\boldsymbol{d}_i$, which has value 1.

Notes:

- Suffers from issues of floating point precision, in practice
- Typically only appropriate to use on small examples with an appropriately chosen epsilon



Backpropagation of Errors

Just a lil bit of white board time!

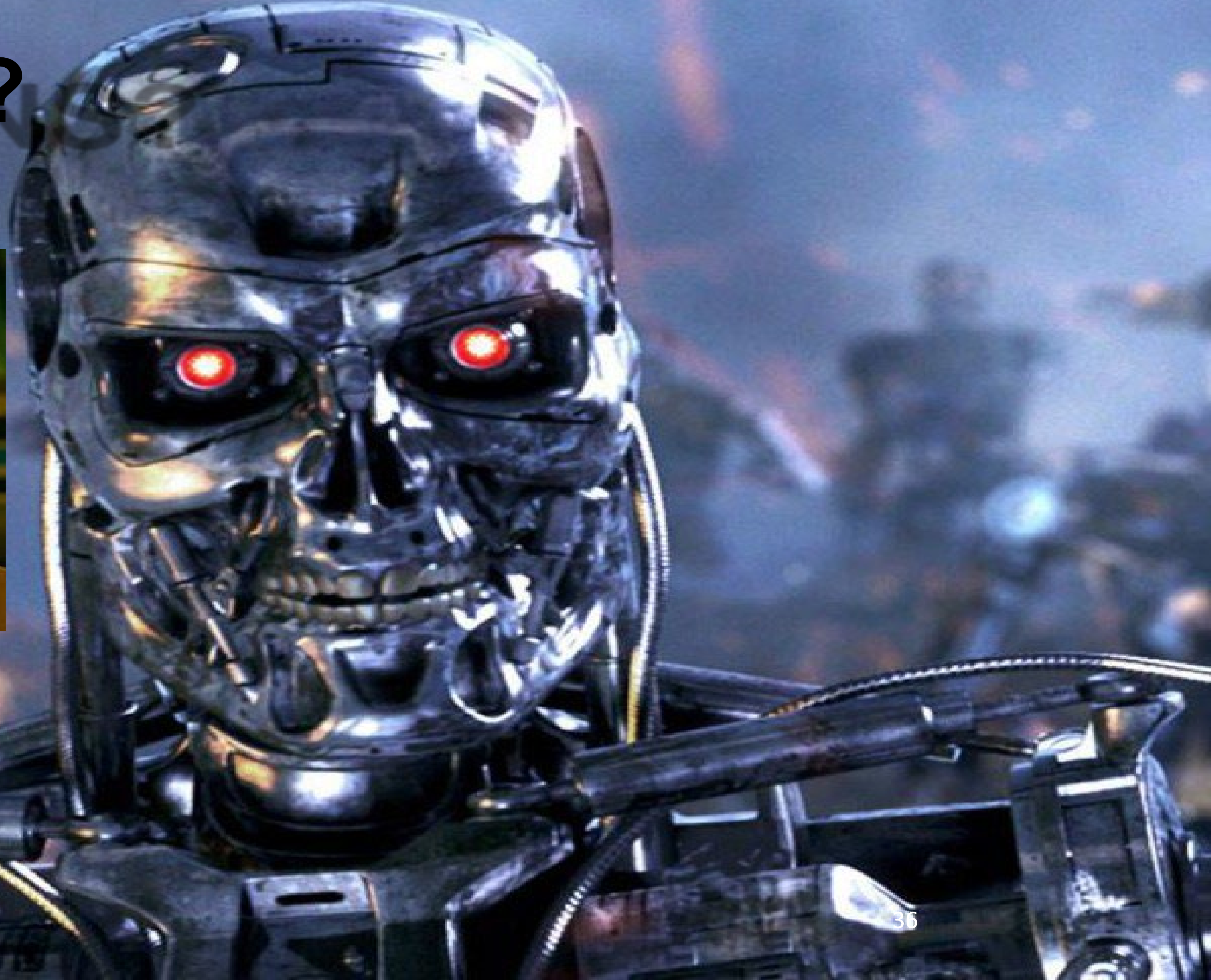
The Vanishing Gradient Problem

- Solving credit assignment problem with back-propagation too difficult
  - Difficult to know how much importance to accord to remote inputs (Bengio et al., 1994)
  - Information passed through a chain of multiplications back through network
    - Any value slightly less than 1 in hadamard product, and derivative signal quickly shrinks to useless values (near zero)
  - Learning long-term dependencies in temporal sequences becomes near impossible
- Complementary problem: Exploding gradients
  - Any value greater than 1 in hadamard, derivative signal increases dramatically (numerical overflow)

Random Parameter Initializations

- Classical approaches
  - Sample from *~U(-a, a)*, where is *a* small scalar
  - Sample from *~N(0, a)*, where is *a* small standard deviation
- Fan-in-Fan-out (number inputs, number output)
  - Calibrate by variances of neuronal activities
- Simple distributional schemes
  - Fan-in/Fan-out Uniform
  - Fan-in/Fan-out Gaussian (good for ReLU activations)
- Orthogonal Initialization
  - Use Singular Value Decomposition (SVD) to find initial weights
- Identity Initialization / Constraint (for RNNs)
  - Does not always work unless constraint is enforced
- Or other intelligent methods?
  - Greedy layer-wise pre-training (we will go over this later in the course!)

QUESTIONS?



Extra Content