Introduction to Neural Networks

Fundamental ideas behind artificial neural networks

Haidar Khan
Bulent Yener

Rensselaer

Outline

- Introduction
- Machine Learning framework
- Neural Networks

1. Simple linear models
2. Nonlinear activations
3. Gradient descent

- Demos

What are (artificial) neural networks

- A technique to estimate patterns from data (~1940s)
- Also called “multi-layer perceptrons”
- “neural” – very crude mimicry of how real biological neurons work
- Large network of simple units which produce a complex output

Why do we care about them

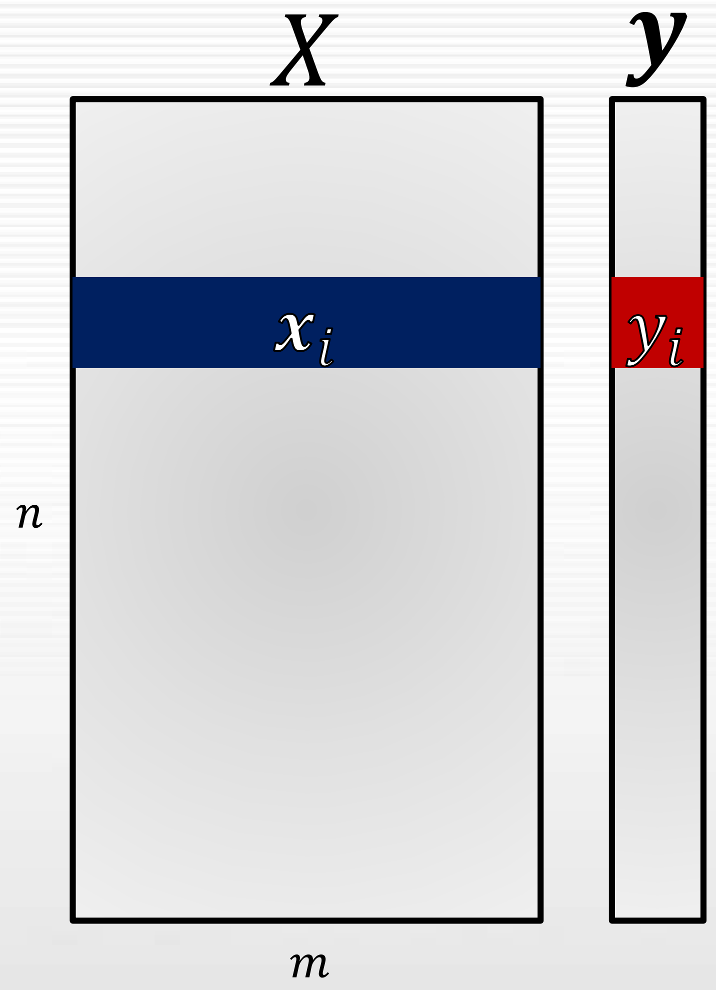
- Key ingredient in real AI
- Useful for industry problems
- Perform best on important tasks
- Yield insights into the biological brain (maybe)

General machine learning framework

- Data – $n \times m$ matrix X
  - rows are observations $\boldsymbol{x}_i$ $(1 \times m)$
- Data labels - $n \times 1$ vector y
- Assume there is some unknown function $f(\cdot)$ that *generates* the label y_i given $\boldsymbol{x}_i$:

$$f(\boldsymbol{x}_i) = y_i$$

- ML problem: estimate $f(\cdot)$
- Use it to generate labels for **new** observations!



Some examples…

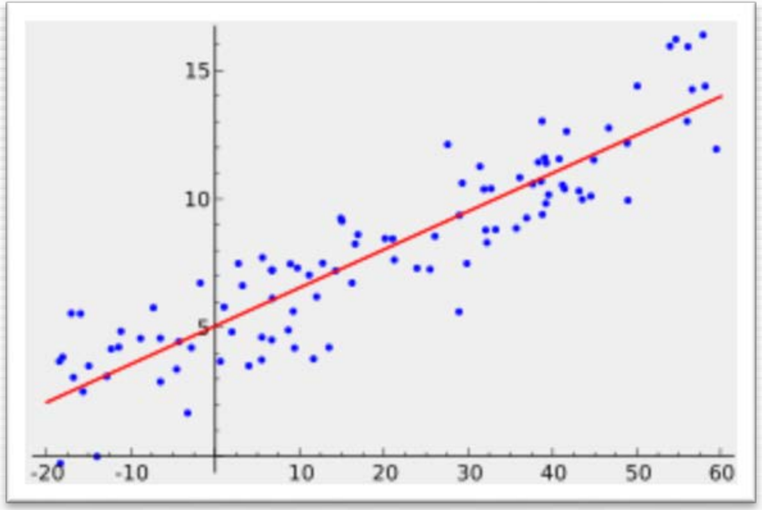
Problem	Data	Data labels
119 images of cats and dogs (20 x 20 pixels)	119×400 matrix of pixel data (we stretch each image into a long vector)	{Cat, Dog}
A 15 question political poll of 139 residents on recent state legislation	139×15 matrix of answers (A-E)	Party affiliation: {Republican, Democrat, Independent}

Recall: Linear regression

- Assume the generating function $f(\cdot)$ is linear
  - Write label y_i as a linear function of X: $y_i = \boldsymbol{x}_i \boldsymbol{w}$
  - Matrix form: $\boldsymbol{y} = X\boldsymbol{w}$
- What should the $m \times 1$ vector $\boldsymbol{w}$ be?
- This is the familiar **least squares regression**:

$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{y}$$

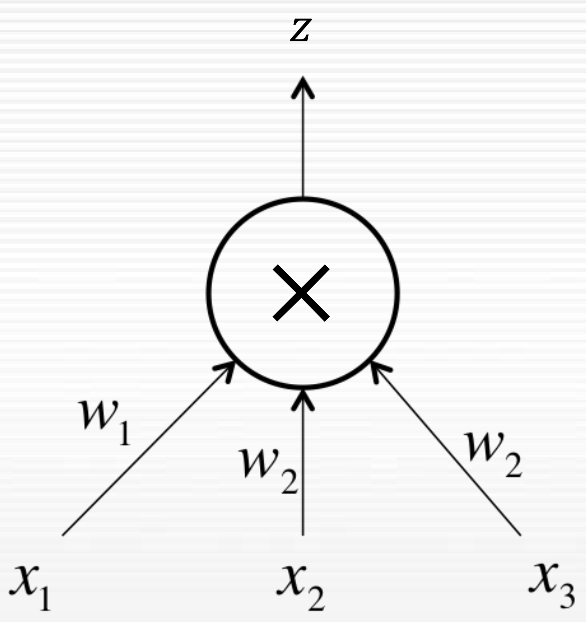
- We will set up the simplest neural network and show we arrive at this same solution!



Declare a simple neural network

- Recall $\boldsymbol{x}$ is $1 \times m$
- One artificial neural unit
- Connects to each input x_i with a weight w_i
- Produces one output z

$$z = \sum_{i}^{m} x_i w_i$$



http://nikhilbuduma.com

Set an objective to learn

- Want network outputs z_i to match labels y_i
  - Choose a loss function E and optimize w.r.t the weights

$$E = \frac{1}{2}\sum_{i}^{N}(z_i - y_i)^2$$

$$E = \frac{1}{2}\sum_{i}^{N}(\boldsymbol{x}_i\boldsymbol{w} - y_i)^2$$

- How to minimize E with respect to $\boldsymbol{w}$?

Equivalence to least squares

- Take the derivative and set it to zero:

$$\frac{dE}{d\boldsymbol{w}} = \sum_{i}^{N} (\boldsymbol{x}_i \boldsymbol{w} - y_i) \boldsymbol{x}_i^T$$

$$\sum_{i}^{N} \boldsymbol{x}_i^T \boldsymbol{x}_i \boldsymbol{w} - \boldsymbol{x}_i^T y_i = \boldsymbol{0}$$

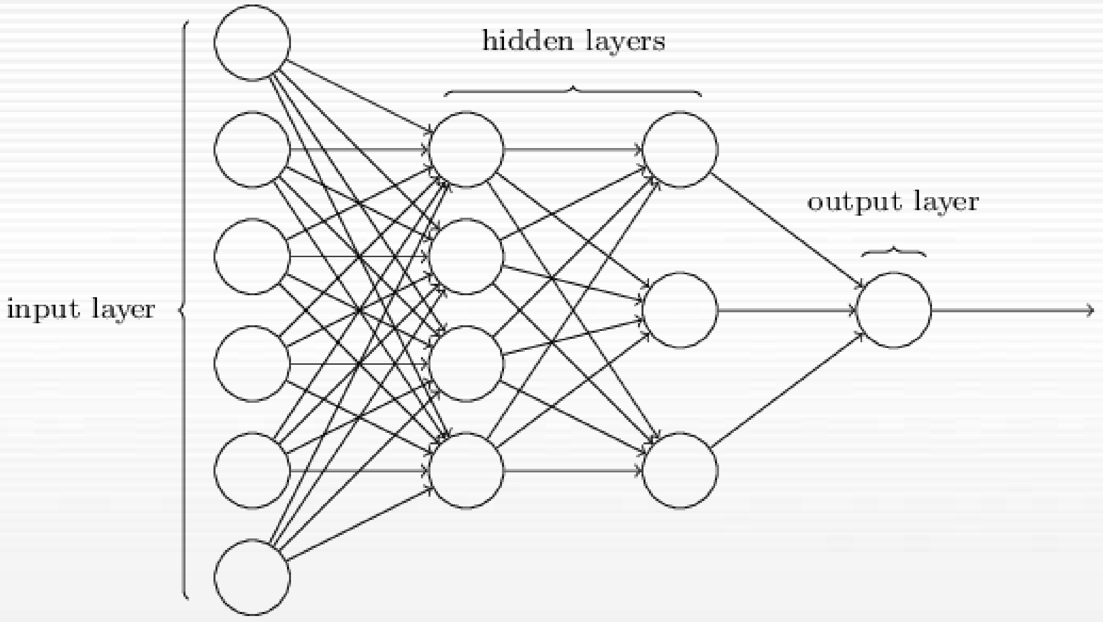
- Written in matrix form this becomes:

$$X^T X \boldsymbol{w} - X^T \boldsymbol{y} = \boldsymbol{0}$$

$$\boxed{\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{y}}$$

Key idea: compose simple units

- Where do we go from here?
- Use many of these simple units and **compose** them in layers:
  - Function composition: $g(h(\cdot))$
- Each layer learns a new representation of the data
  - 3 layer network: $z_i = h_3\left(h_2\left(h_1(\boldsymbol{x}_i)\right)\right)$



http://neuralnetworksanddeeplearning.com

Drawback to only linear units

- Recall our earlier assumption that $f(\cdot)$ is linear
  - This is a very restrictive assumption
- Furthermore, composing strictly linear models is also linear!

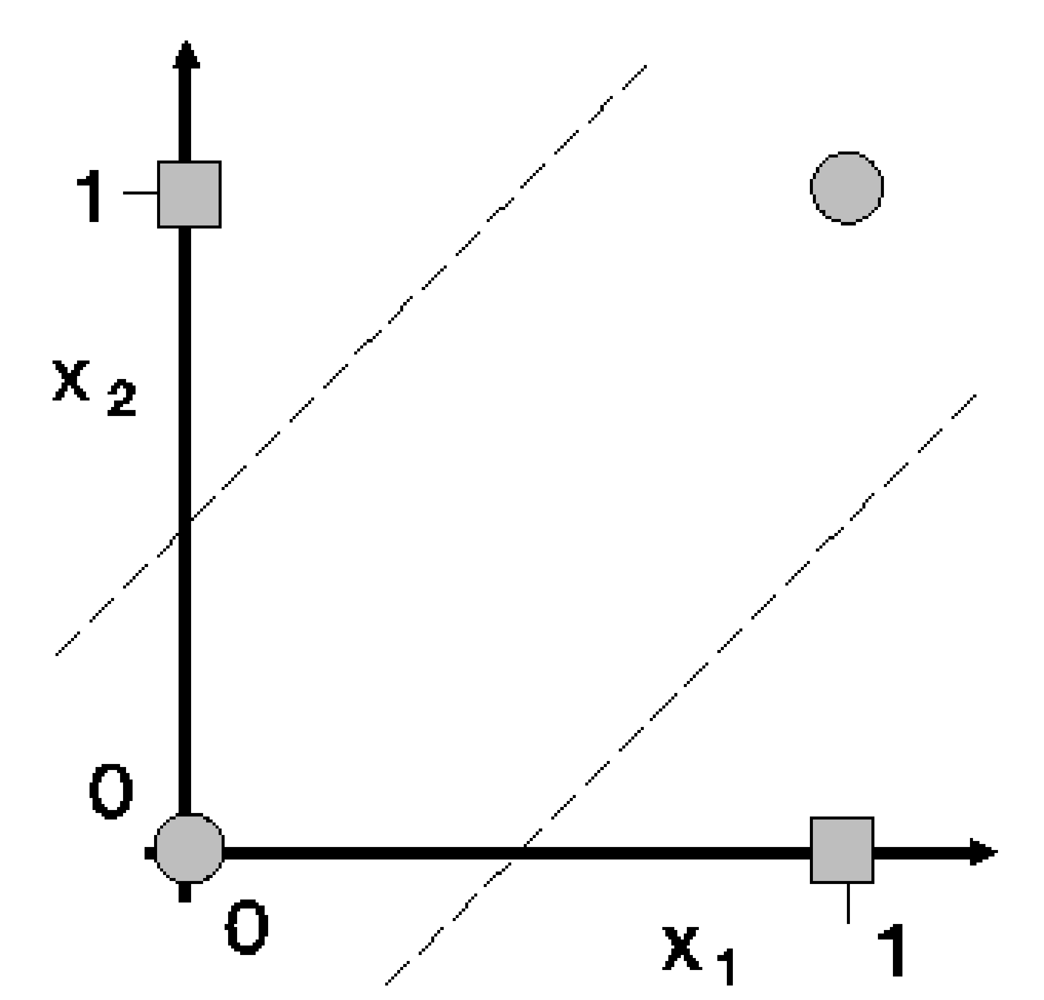
$$z_i = h_3\left(h_2\left(h_1(\boldsymbol{x}_i)\right)\right) = W_3 W_2 W_1 \boldsymbol{x}_i = W_{123} \boldsymbol{x}_i$$

- XOR problem (Minsky, Papert 1969)

XOR problem

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$

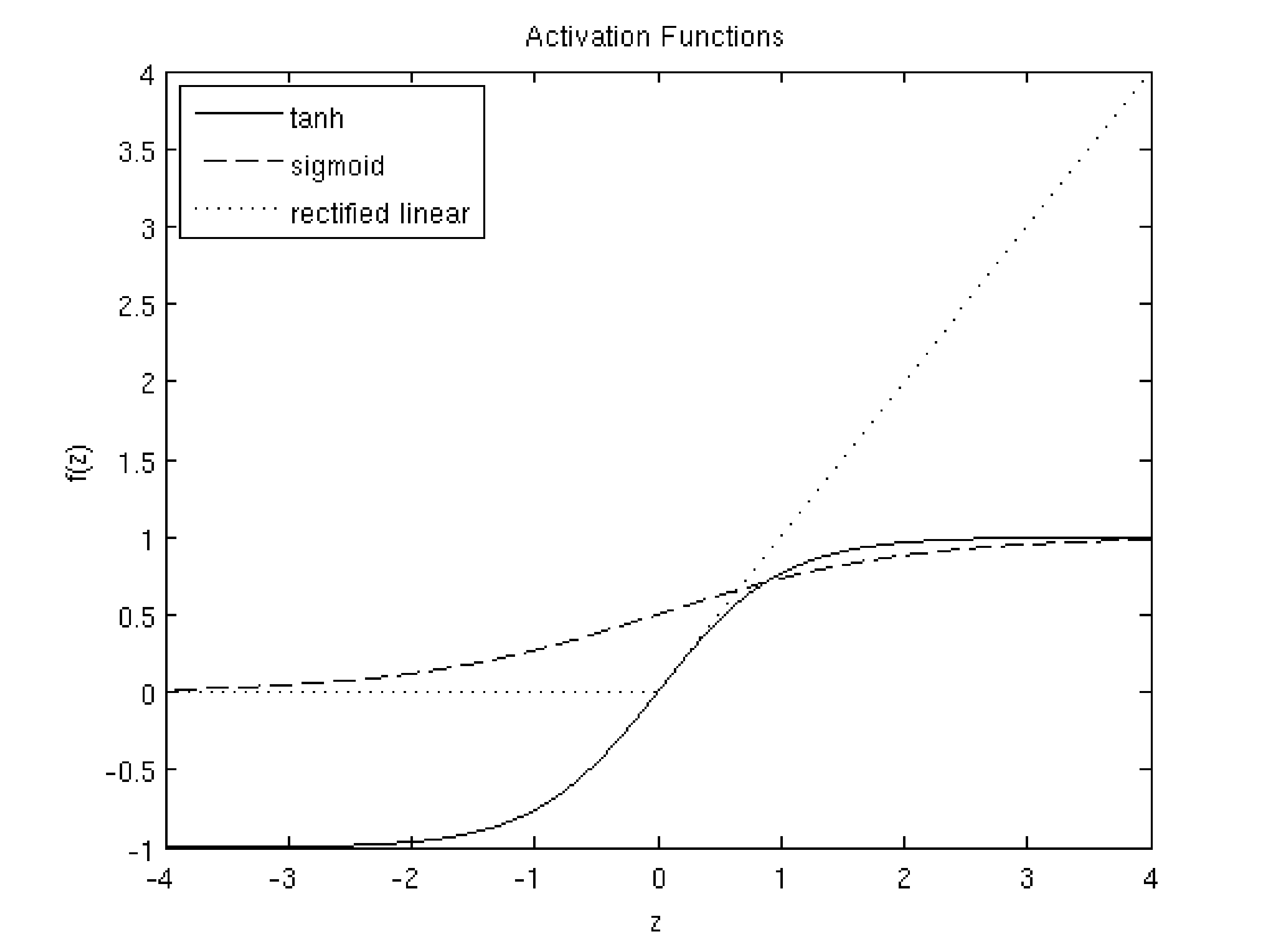


- Can't learn a simple XOR gate using only one straight line

Key idea: non-linear activations

- Solution: add a non-linear function at the output of each layer
- What kind of function?
- Differentiable at least:
  - Hyperbolic tangent: $z = \tanh(\boldsymbol{w}^T \boldsymbol{x}_i)$
  - Sigmoid: $z = \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}_i}}$
  - Rectified Linear: $z = \max(0, \boldsymbol{w}^T \boldsymbol{x}_i)$
- Why? Labels $\boldsymbol{y}$ can be a non-linear function of the inputs (like XOR)

Examples of non-linear activations



http://ufldl.stanford.edu

How do we learn weights now?

- With multiple layers and non-linear activation functions we can't simply take the derivative and set it to 0
- Still can set a loss function and:
  - Randomly try different weights
  - Numerically estimate the derivative

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

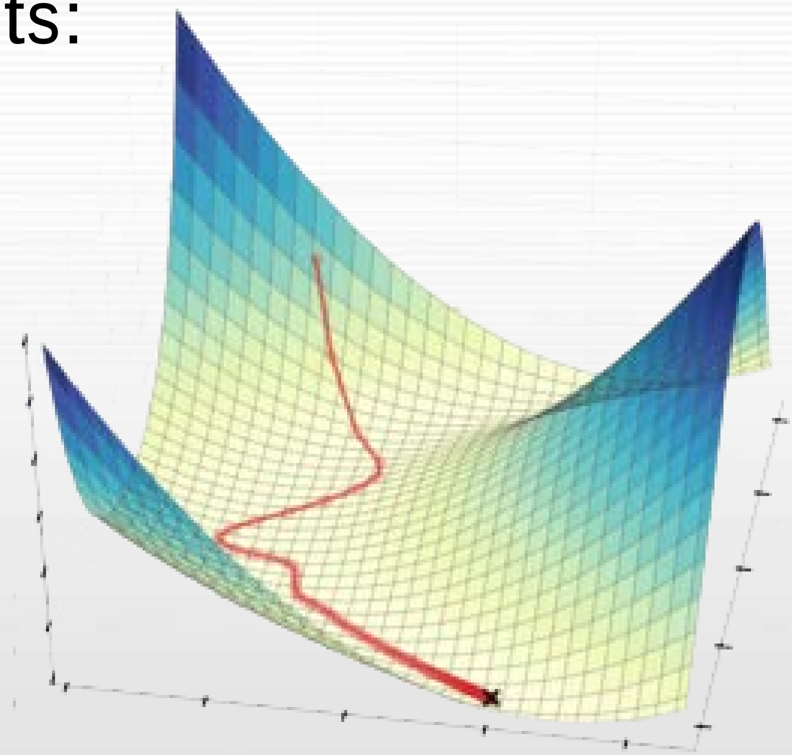
- Terribly inefficient and scale badly with the number of layers...

Key idea: gradient descent on loss function

- Suppose we could calculate the partial derivative of E w.r.t each weight w_i: $\frac{\delta E}{\delta w_i}$ (gradient)
- Decrease the loss function E by updating weights:

$$w_i = w_i + \frac{\delta E}{\delta w_i}$$

- Repeatedly doing this process is called gradient descent
- Leads to a set of weights that correspond to a local minimum of the loss function



Backpropagation to estimate gradients

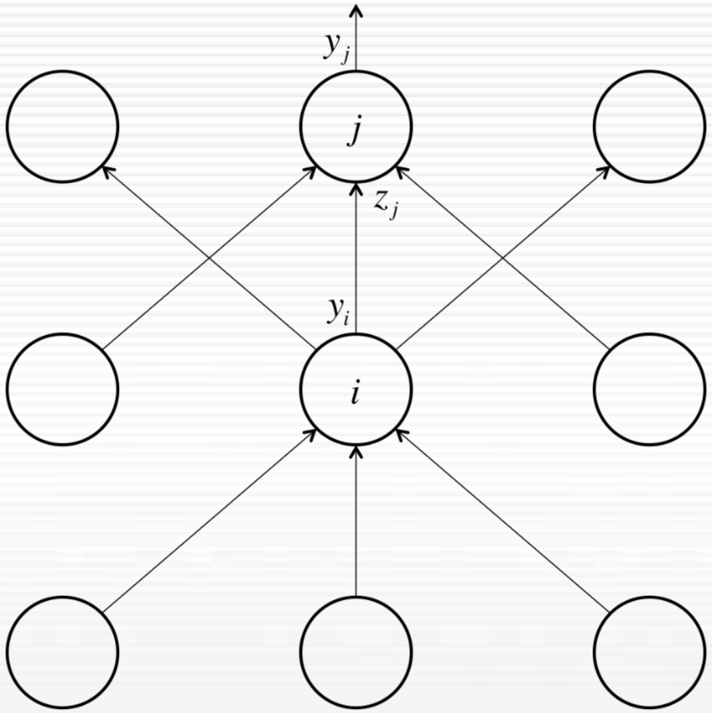
- One of the breakthroughs in neural network research
- Allows to calculate the gradients of the network!
- Core idea behind the algorithm is multiple applications of the chain rule of derivatives:

$$F(x) = f\big(g(x)\big)$$
$$F'(x) = f'\big(g(x)\big)g'(x)$$

- Two passes through the network: forward and backward
  - Forward: calculate $g(x)$ and then $f(g(x))$
  - Backward: calculate $f'(g(x))$ and then $g'(x)$

Multilayer Backpropagation

- Assume we have t_i, t_j, z_j from the forward pass
- Work backward from the output of the network:
- $E = \frac{1}{2}\sum_{j \in output}\left(t_j - y_j\right)^2$, $\frac{\delta E}{\delta t_j} = -\left(t_j - y_j\right)$ (for output neurons)
- $\frac{\delta E}{\delta t_i} = \sum_j \frac{dz_j}{dt_i}\left(\frac{\delta E}{\delta z_j}\right) = \sum_j w_{ij}\left(\frac{\delta E}{\delta z_j}\right)$
- $\frac{\delta E}{\delta z_j} = \frac{\delta t_j}{\delta z_j}\left(\frac{\delta E}{\delta t_j}\right) = t_j\left(\frac{\delta E}{\delta t_j}\right)$
- $\frac{\delta E}{\delta t_i} = \sum_j w_{ij}\, t_j\left(\frac{\delta E}{\delta t_j}\right)$
- $\frac{\delta E}{\delta w_{ij}} = \frac{\delta z_j}{\delta w_{ij}}\left(\frac{\delta E}{\delta z_j}\right) = t_i t_j \frac{\delta E}{\delta t_j}$



http://nikhilbuduma.com

Putting all the pieces together

- 3 key elements to understanding neural networks
  - Composition of units with simple operations (dot-product)
  - Non-linearity activation functions at unit outputs
  - Learn weights using gradient descent
- Using neural networks:
  - Set up data matrix and label vector: X and y
  - Define a network architecture: number of layers, units per layer
  - Choose a loss function to minimize: depends on the task

A couple of demos...

Credits

- Images from:
  - http://nikhilbuduma.com/2015/01/11/a-deep-dive-into-recurrent-neural-networks/
  - http://ufldl.stanford.edu
  - http://neuralnetworksanddeeplearning.com