LECTURE 8: NEURAL NETWORKS, BACKPROPAGATION
A NEURON

A single neuron
A computational unit with \( n \) (3) inputs and 1 output and parameters \( W, b \)

Inputs  Activation function  Output
A NEURON IS A BINARY LOGISTIC REGRESSION UNIT

$f = \text{nonlinear activation fct. (e.g. sigmoid)}, \ w = \text{weights, } b = \text{bias, } h = \text{hidden, } x = \text{inputs}$

$h_{w,b}(x) = f(w^T x + b)$

$b: \text{We can have an “always on” feature, which gives a class prior, or separate it out, as a bias term}$

$f(z) = \frac{1}{1 + e^{-z}}$

$w, b$ are the parameters of this neuron i.e., this logistic regression model
A NEURAL NETWORK = RUNNING SEVERAL LOGISTIC REGRESSIONS AT THE SAME TIME

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs ...

But we don’t have to decide ahead of time what variables these logistic regressions are trying to predict!
A neural network = running several logistic regressions at the same time

... which we can feed into another logistic regression function

It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.
A NEURAL NETWORK = RUNNING SEVERAL LOGISTIC REgressions AT THE SAME TIME

Before we know it, we have a multilayer neural network....
Matrix Notation for a Layer

We have

\[ a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1) \]
\[ a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2) \]

etc.

In matrix notation

\[ z = Wx + b \]

\[ a = f(z) \]

where \( f \) is applied element-wise:

\[ f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)] \]
NON-LINEARITIES

- Example: function approximation, e.g., regression or classification
  - Without non-linearities, deep neural networks can’t do anything more than a linear transform
  - Extra layers could just be compiled down into a single linear transform: $W_1 W_2 x = Wx$
  - With more layers, they can approximate more complex functions!
BACKPROPAGATION
BACKPROPAGATION

- Backward propagation of errors
- Compute gradients algorithmically.
- Used by deep learning frameworks (Tensorflow, PyTorch, Keras, etc.)
LET'S GO BACK TO THE SAME EXAMPLE

- Output unit computation

\[
sigmoid(0.90 \times 4.5 + 0.17 \times -5.2 + 1 \times -2.0) = sigmoid(1.17) = \frac{1}{1 + e^{-1.17}} = 0.76
\]
HOW TO DECREASE THE ERROR?
TO LEARN...
ERROR

- Computed output: $y = .76$
- Correct output: $t = 1.0$

⇒ How do we adjust the weights?
KEY CONCEPTS

- Gradient descent
  - error is a function of the weights
  - we want to reduce the error
  - gradient descent: move towards the error minimum
  - compute gradient $\rightarrow$ get direction to the error minimum
  - adjust weights towards direction of lower error

- Back-propagation
  - first adjust last set of weights
  - propagate error back to each previous layer
  - adjust their weights
GRADIENT DESCENT
DERIVATIVE OF SIGMOID

• Sigmoid
  \[ \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \]

• Reminder: quotient rule
  \[ \left( \frac{f(x)}{g(x)} \right)' = \frac{g(x)f'(x) - f(x)g'(x)}{g(x)^2} \]

• Derivative
  \[
  \frac{d \text{sigmoid}(x)}{dx} = \frac{d}{dx} \frac{1}{1 + e^{-x}} \\
  = 0 \times (1 - e^{-x}) - (-e^{-x}) \\
  = \frac{e^{-x}}{(1 + e^{-x})^2} \\
  = \frac{1}{1 + e^{-x}}(\frac{e^{-x}}{1 + e^{-x}}) \\
  = \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}}\right) \\
  = \text{sigmoid}(x)(1 - \text{sigmoid}(x))
  \]
FINAL LAYER UPDATE

- Linear combination of weights $s = \sum_k w_k h_k$
- Activation function $y = \text{sigmoid}(s)$
- Error (L2 norm) $E = \frac{1}{2}(t - y)^2$
- Derivative of error with regard to one weight $w_k$

$$\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}$$
**FINAL LAYER UPDATE (1)**

- Linear combination of weights \( s = \sum_k w_k h_k \)
- Activation function \( y = \text{sigmoid}(s) \)
- Error (L2 norm) \( E = \frac{1}{2}(t - y)^2 \)
- Derivative of error with regard to one weight \( w_k \)
  \[
  \frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}
  \]
- Error \( E \) is defined with respect to \( y \)
  \[
  \frac{dE}{dy} = \frac{d}{dy} \frac{1}{2}(t - y)^2 = -(t - y)
  \]
FINAL LAYER UPDATE (2)

- Linear combination of weights $s = \sum_k w_k h_k$
- Activation function $y = \text{sigmoid}(s)$
- Error (L2 norm) $E = \frac{1}{2}(t - y)^2$
- Derivative of error with regard to one weight $w_k$
  $$\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}$$
- $y$ with respect to $x$ is $\text{sigmoid}(s)$
  $$\frac{dy}{ds} = \frac{d}{ds} \text{sigmoid}(s) = \text{sigmoid}(s)(1 - \text{sigmoid}(s)) = y(1 - y)$$
**FINAL LAYER UPDATE** (3)

- Linear combination of weights \( s = \sum_k w_k h_k \)
- Activation function \( y = \text{sigmoid}(s) \)
- Error (L2 norm) \( E = \frac{1}{2}(t - y)^2 \)
- Derivative of error with regard to one weight \( w_k \)
  \[
  \frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}
  \]
- \( x \) is weighted linear combination of hidden node values \( h_k \)
  \[
  \frac{ds}{dw_k} = \frac{d}{dw_k} \sum_k w_k h_k = h_k
  \]
PUTTING IT ALL TOGETHER

- Derivative of error with regard to one weight $w_k$

$$\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}$$

$$= -(t - y) \ y(1 - y) \ h_k$$

- error
- derivative of sigmoid: $y'$

- Weight adjustment will be scaled by a fixed learning rate $\mu$

$$\Delta w_k = \mu \ (t - y) \ y' \ h_k$$
MULTIPLE OUTPUT NODES

- Our example only had one output node
- Typically neural networks have multiple output nodes
- Error is computed over all $j$ output nodes

$$E = \sum_j \frac{1}{2} (t_j - y_j)^2$$

- Weights $k \rightarrow j$ are adjusted according to the node they point to

$$\Delta w_{j \leftarrow k} = \mu (t_j - y_j) y'_j h_k$$
HIDDEN LAYER UPDATE

• In a hidden layer, we do not have a target output value

• But we can compute how much each node contributed to downstream error

• Definition of error term of each node

\[ \delta_j = (t_j - y_j) y'_j \]

• Back-propagate the error term

(why this way? there is math to back it up...)

\[ \delta_i = \left( \sum_j w_{j \leftarrow i} \delta_j \right) y'_i \]

• Universal update formula

\[ \Delta w_{j \leftarrow k} = \mu \delta_j h_k \]
OUR EXAMPLE

- Computed output: \( y = .76 \)
- Correct output: \( t = 1.0 \)

- Final layer weight updates (learning rate \( \mu = 10 \))
  - \( \delta_G = (t - y) \ y' = (1 - .76) \ 0.181 = .0434 \)
  - \( \Delta w_{GD} = \mu \ \delta_G \ h_D = 10 \times .0434 \times .90 = .391 \)
  - \( \Delta w_{GE} = \mu \ \delta_G \ h_E = 10 \times .0434 \times .17 = .074 \)
  - \( \Delta w_{GF} = \mu \ \delta_G \ h_F = 10 \times .0434 \times 1 = .434 \)
OUR EXAMPLE

- Computed output: $y = .76$
- Correct output: $t = 1.0$
- Final layer weight updates (learning rate $\mu = 10$)
  - $\delta_G = (t - y) y' = (1 - .76) 0.181 = .0434$
  - $\Delta w_{GD} = \mu \delta_G h_D = 10 \times .0434 \times .90 = .391$
  - $\Delta w_{GE} = \mu \delta_G h_E = 10 \times .0434 \times .17 = .074$
  - $\Delta w_{GF} = \mu \delta_G h_F = 10 \times .0434 \times 1 = .434$
HIDDEN LAYER UPDATES

- Hidden node D
  - $\delta_D = \left( \sum_j w_{j \rightarrow_i} \delta_j \right) y'_D = w_{GD} \delta_G y'_D = 4.5 \times 0.0434 \times 0.0898 = 0.0175$
  - $\Delta w_{DA} = \mu \delta_D h_A = 10 \times 0.0175 \times 1.0 = 0.175$
  - $\Delta w_{DB} = \mu \delta_D h_B = 10 \times 0.0175 \times 0.0 = 0$
  - $\Delta w_{DC} = \mu \delta_D h_C = 10 \times 0.0175 \times 1 = 0.175$

- Hidden node E
  - $\delta_E = \left( \sum_j w_{j \rightarrow_i} \delta_j \right) y'_E = w_{GE} \delta_G y'_E = -5.2 \times 0.0434 \times 0.2055 = -0.0464$
  - $\Delta w_{EA} = \mu \delta_E h_A = 10 \times -0.0464 \times 1.0 = -0.464$
  - etc.
REFERENCES

- Introduction to Neural Networks, Philipp Koehn, 3 October 2017
- Artificial Neural Networks and Deep Learning, Christian Borgelt, University of Konstanz, Germany
- Natural Language Processing with Deep Learning, Richard Socher, Kevin Clark, Stanford University, 2017
- Richard Socher, Neural Networks, 2018