Lecture 8 - Neural Networks II

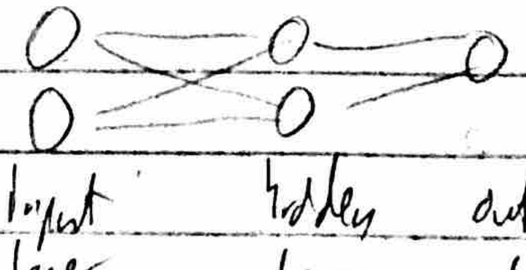
Today
- Extended review
  - Feed-forward neural networks
  - Vector/matrix representation
  - Practice
- Training neural networks
  - Loss function
  - Gradient descent
  - Back propagation

Extended Review

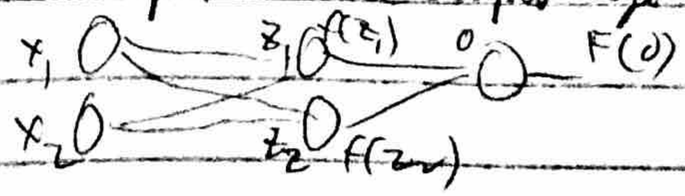
Feed-forward neural networks
Vector/matrix representation
] See lecture?

Practice

(1) Draw a neural network with 2 input units, 2 hidden units, and 1 output unit (no bias). Label the input layer, hidden layer, and output layer.

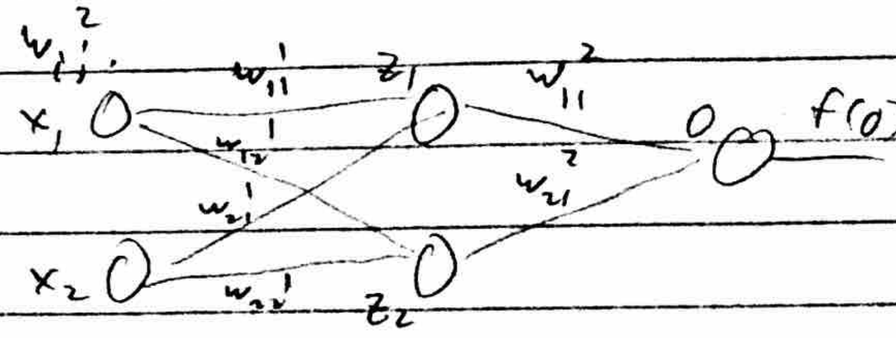


(2) Label all of the units, using x_1, x_2 for input, z_1, z_2 for hidden and o for the output. Indicate the output of the hidden layer $f(z_1)$, $f(z_2)$ and the output of the output layer $F(o)$.

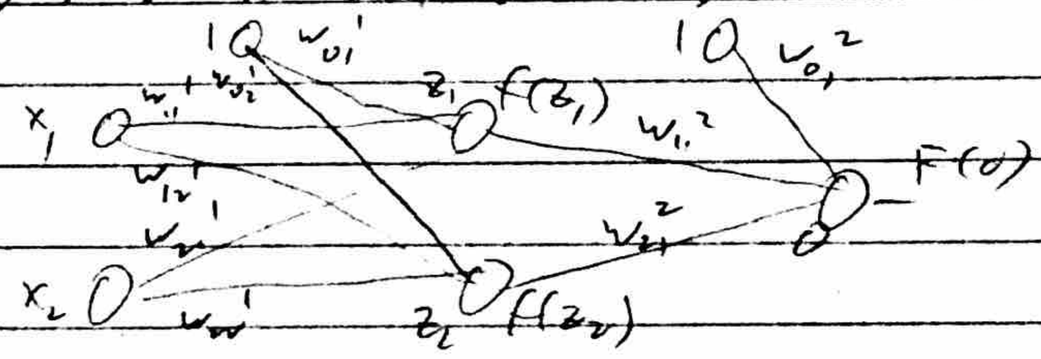


q-2

③ Label the weights with the appropriate subscripts. Weights in the first layer should be labelled w_{ij}^1 and weights in the second layer should be labelled w_{ij}^2



④ Add and label the biases for the hidden layer and the output layer.



⑤ Write the equations for z_1 and z_2.

$z_1 = x_1 w_{11}^1 + x_2 w_{21}^1 + 1 \cdot w_{01}^1$

$z_2 = x_1 w_{12}^1 + x_2 w_{22}^1 + 1 \cdot w_{02}^1$

⑥ Express the above equations in matrix/vector form.

$z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ $w^1 = \begin{bmatrix} w_{11}^1 & w_{21}^1 \\ w_{12}^1 & w_{22}^1 \end{bmatrix}$ $b^1 = \begin{bmatrix} w_{01}^1 \\ w_{02}^1 \end{bmatrix}$

$z = x \cdot w^1 + b^1$

⑦ Write the equation for o.

$o = f(z_1) w_{11}^2 + f(z_2) w_{21}^2 + w_{01}^2$

⑧ Express the above equation in matrix form.

$a = \begin{pmatrix} f(z_1) \\ f(z_2) \end{pmatrix}$ $w^2 = [w_{11}^2 \; w_{21}^2]$ $b^2 = [w_{01}^2]$

$o = a \cdot w^2 + b^2$

F(0)

⑨ Express the final output of the network as a matrix equation in terms of x and the weights

Output $= F(f(x \cdot w^1 + b^1) \cdot w^2 + b^2)$

⑩ Let $x = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$, $w^1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, $b^1 = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$. What is $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$?

$z_1 = x_1 w_{11}^1 + x_2 w_{12}^1 + w_{01}^1 = 2 \cdot 1 + (-1) \cdot 1 - 3 = -2$

$z_2 = x_1 w_{21}^1 + x_2 w_{22}^1 + w_{02}^1 = 2 \cdot 1 + (-1) \cdot (-1) - 1 = 2$

$z = \begin{pmatrix} -2 \\ 2 \end{pmatrix}$

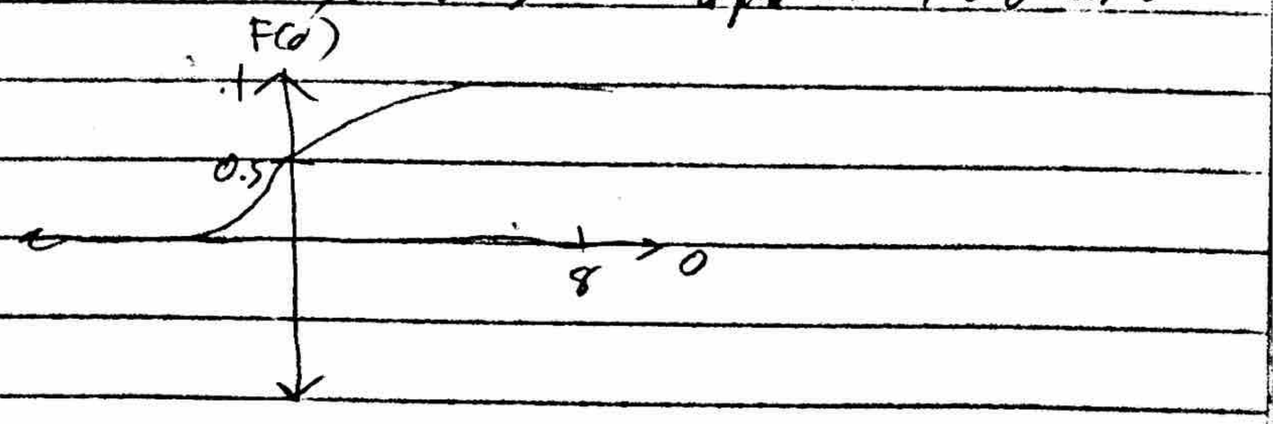
⑪ $w^2 = [1 \quad 3]$, $b^2 = [2]$, $f = ReLU$ $(f(z) = \max(0, z))$.
What is o?

$o = f(z_1) w_{11}^2 + f(z_2) w_{21}^2 + w_{01}^2 = \max(0, -2) \cdot 1 + \max(0, 2) \cdot 3 + 2$

$= 0 + 6 + 2 = 8$

⑫ Let $F = sigmoid$ $\left(F(0) = \sigma(0) = \frac{1}{1+e^{-0}}\right)$. What is the output of the network?

$F(0) = \frac{1}{1+e^{-8}} = 0.99966$



Training neural networks

Now we're pretty good at working with feed forward neural networks and using them to make predictions.

But the one question we haven't addressed yet is how to select the weights. Next we're going to discuss how to learn the weights.

We're going to learn the weights the way we've learned parameters in some of the other algorithms we've discussed (ex. Pegasos for online SVM). We're going to define a loss function and we're going to try to optimize it.

Loss function

Our loss function is going to be a function of the parameters of the network.

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(y^{(i)}, F(x^{(i)}; \theta)) + \frac{\lambda}{2} \|\theta\|^2$$

This should look familiar to loss functions we've worked with before.

θ → parameters of the network (all weights and biases)

note: θ is not a vector; it's a collection of all parameters in the network

$\frac{1}{n} \sum_{i=1}^{n}$ → averaging loss across all training examples

$F(x^{(i)}; \theta)$ → prediction of network for training example $x^{(i)}$ using parameters (weights + biases) θ

$y^{(i)}$ → correct label for training example $x^{(i)}$

$y^{(i)}, F(x^{(i)}; \theta)$ → agreement between network and correct label

$\text{Loss}(y^{(i)}, F(x^{(i)}; \theta))$ → the loss for training example $x^{(i)}$

Loss can be any function - we choose it depending on what we're trying to optimize for and what task (ex, classification, regression) we're performing

$\frac{\lambda}{2} \|\theta\|^2$ → regularization - keeps the parameters relatively small to prevent overfitting

Goal: choose θ to minimize the loss function.

How: Gradient descent

Gradient Descent

We've talked about this before. The idea is that the gradient points in the direction of increasing loss, so we alter the weights to go in the opposite direction.

$$w_{ij} = w_{ij} - \eta \frac{\partial}{\partial w_{ij}} \text{Loss}(y, F(x; \theta))$$

η = learning rate - controls how far we travel on each gradient descent step

Problem: How do we compute $\frac{\partial Loss}{\partial w_{ij}}$ if w_{ij} is not near the end of the network?

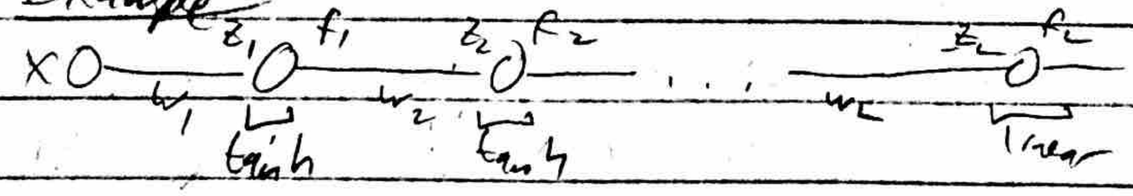
The issue is that writing L as a function of the last weights in the network is easy, but writing L in terms of weights earlier in the network is hard since those weights pass through many other computations before reaching L at the end of the network. This makes computing the derivative directly too difficult.

Backpropagation

Idea: Determine derivatives one layer at a time starting at the end of the network and working toward the beginning. We can compute the derivatives of the current layer by using the derivatives of the next layer.

How? chain rule from calculus.

Example



$loss = Loss(y, f_L)$ since f_L is output of network

Since last unit is linear, we have $f_L = z_L = z_{L-1} \cdot w_L$

$loss = Loss(y, z_{L-1} \cdot w_L)$

Computing the derivative with respect to w_L is easy because the loss is expressed as a function of w_L.

$\frac{\partial Loss(y, z_{L-1} \cdot w_L)}{\partial w_1} = y \cdot z_{L-1} \cdot Loss'(y, z_{L-1} \cdot w_L)$ ← we know how to compute this

But what about the derivative with respect to w_{L-1}?

$\frac{\partial Loss(y, f_L)}{\partial w_{L-1}}$?

What derivative can we easily compute?

$\frac{\partial \text{loss}(y, z_L, w_L)}{\partial z_L} = y w_L \text{loss}'(y, z_L, w_L)$ ← know how to compute

How can we express $\frac{\partial \text{loss}(y, z_L, w_L)}{\partial w_{L-1}}$ in terms of $\frac{\partial \text{loss}(y, z_L, w_L)}{\partial z_L}$?

Use the chain rule.

How does information flow from w_{L-1} to z_L.

$w_{L-1} \to z_{L-1} \to f_{L-1} \to z_L$

$$\frac{\partial \text{loss}(y, z_L, w_L)}{\partial w_{L-1}} = \underbrace{\frac{\partial z_{L-1}}{\partial w_{L-1}} \frac{\partial f_{L-1}}{\partial z_{L-1}} \frac{\partial z_L}{\partial f_{L-1}}}_{\text{can easily compute}} \underbrace{\frac{\partial \text{loss}(y, z_L, w_L)}{\partial z_L}}_{\text{know}},$$

$z_L = f_{L-1} \cdot w_L \to \frac{\partial z_L}{\partial f_{L-1}} = w_L$

$f_{L-1} = \tanh(z_{L-1}) \to \frac{\partial f_{L-1}}{\partial z_{L-1}} = \tanh'(z_{L-1})$

$z_{L-1} = f_{L-2} \cdot w_{L-1} \to \frac{\partial z_{L-1}}{\partial w_{L-1}} = f_{L-2}$

Now we just multiply to get $\frac{\partial \text{loss}(y, z_L, w_L)}{\partial w_{L-1}}$.

We can use a similar process to compute $\frac{\partial \text{loss}(y, z_L, w_L)}{\partial z_{L-1}}$.

Now to compute $\frac{\partial \text{loss}(y, z_L, w_L)}{\partial w_{L-2}}$, we use $\frac{\partial \text{loss}(y, z_L, w_L)}{\partial z_{L-1}}$ which we just computed plus the chain rule.

We repeatedly apply the chain rule to compute derivatives all the way back to the beginning of the network.

Once we've computed all the derivatives we use the gradient descent update step to update the weights.