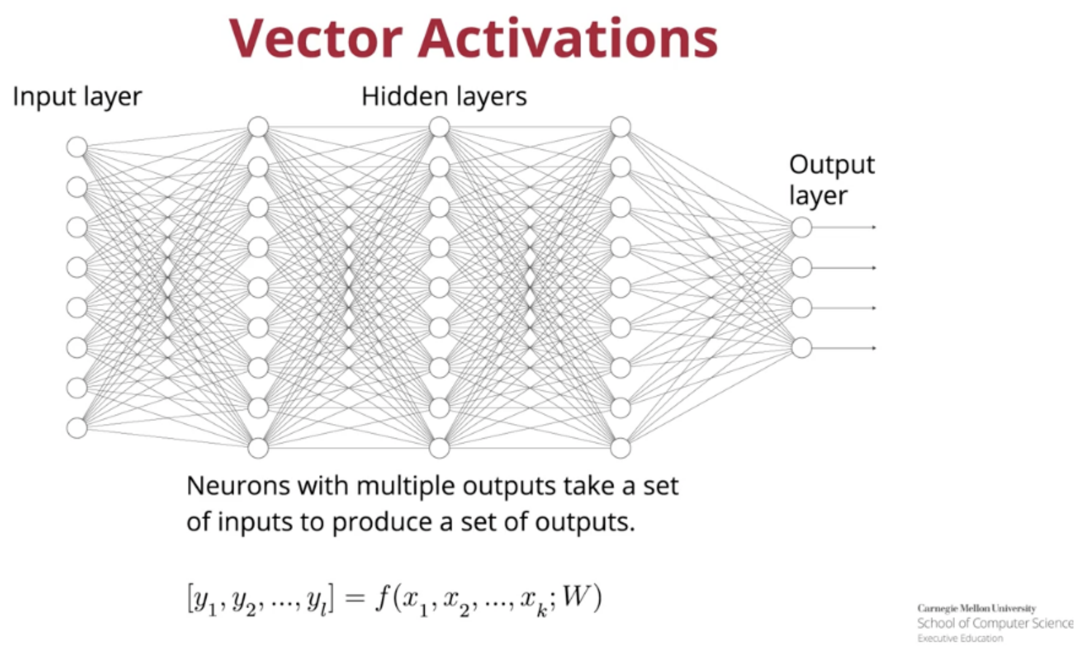
Module 2: Training Multilayer Perceptrons

Activation Functions

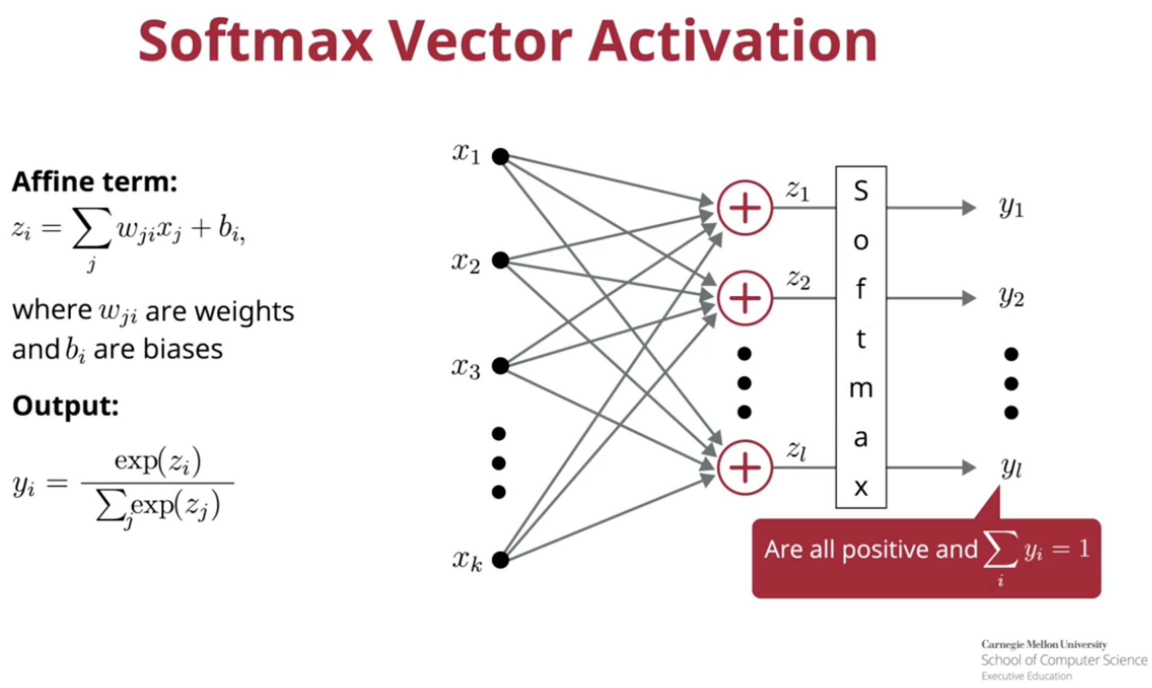
Tanh:

Vector Activation



In contrary to scalar neuron which takes single input and produces single output, vector neurons take series of inputs and return series of outputs.

Softmax Vector Activation



In softmax vector activations, changing one parameter causes change of all parameters. That's because full Output layer sums up to 1, each output value is a probability, thus changing one value up we need to decrease all other values.

Representing the desired output

Scalar output - Will output scala output neuron, notation is d = scalar - real value

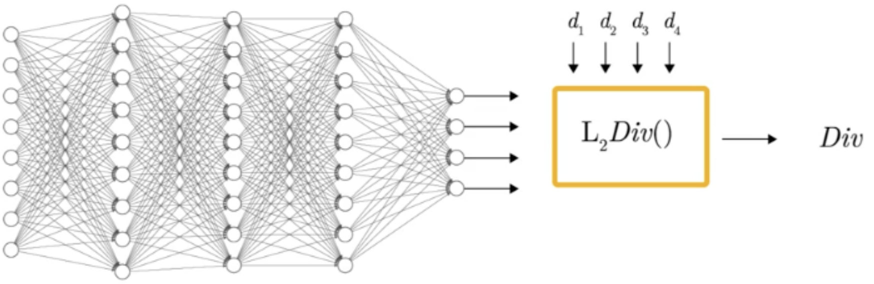
Vector output - will output vector output neuron, notation is d = [d1, d2, …, dl]

Binary output - will output binary neuron, notation is d =1 for yes and d = 0 for no used for binary classification issues.

Function Minimization

Scaled L2 Divergence

Scaled L_2 Divergence



$$Div(Y,d) = \frac{1}{2}\|Y-d\|^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$$

$$\frac{d\,Div(Y,d)}{dy_i} = (y_i - d_i)$$

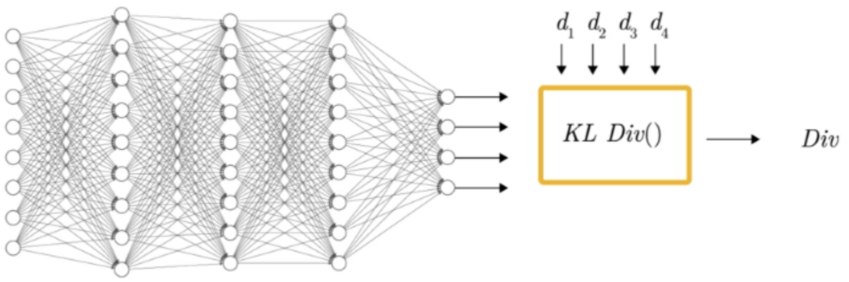
$$\nabla_Y Div(Y,d) = [y_1 - d_1, y_2 - d_2, \ldots]$$

Carnegie Mellon University School of Computer Science Executive Education

---

KL-Divergence for Classification

KL-Divergence for Classification



The KL divergence between y and d is given by:

$$Div(Y,d) = \sum_i d_i \log\frac{d_i}{y_i} = \sum_i d_i \log d_i - \sum_i d_i \log y_i = -\log y_c$$

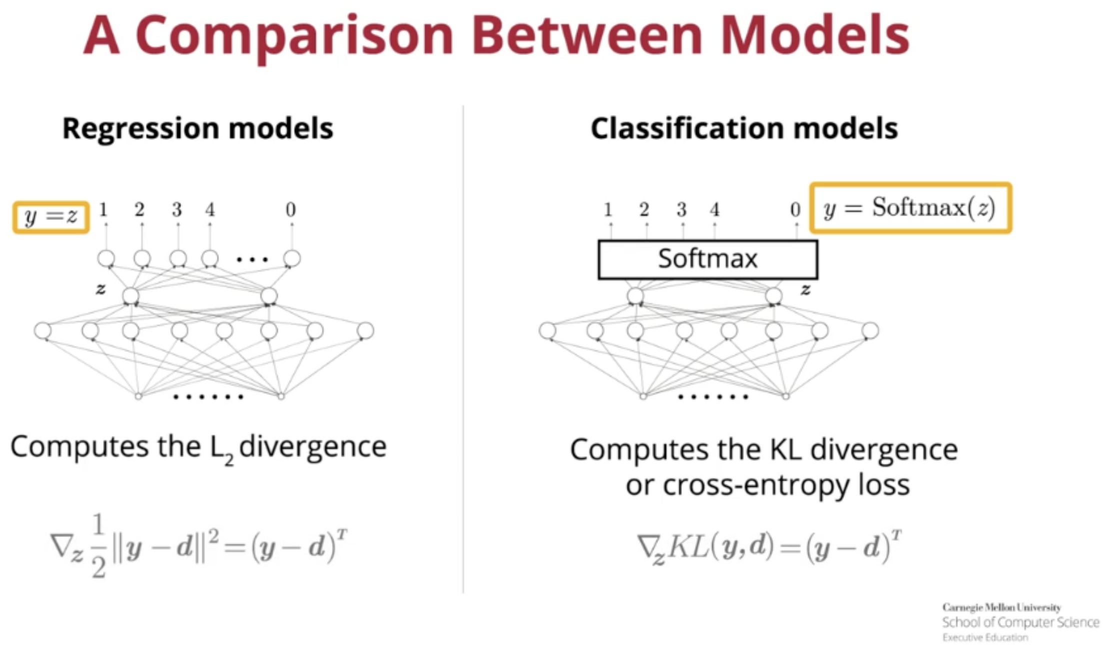
$$\frac{dDiv(Y,d)}{dY_i} = \begin{cases} -\frac{1}{y_c} \text{ for the } c\text{-th component} \\ 0 \text{ for remaining component} \end{cases}$$

$$\nabla_Y Div(Y,d) = \begin{bmatrix} 0 & 0 & \ldots & \frac{-1}{y_c} & \ldots & 0 & 0 \end{bmatrix}$$

Carnegie Mellon University School of Computer Science Executive Education

Divergence decreases when y_c increases where y_x is the probability of target class. That means increasing of probability of correct class will cause decreasing of divergence, which eventually causes better fit to desired output.

When we use which divergence?



The Chain Rule Intuition

Example 1.

Let's say we have 2 separate functions 1. with height and weight, 2. with shoe size and height. In 1. function we can predict height for given weight, then using 2. plot we can predict shoe size using height. That means with weight we can predict shoe size despite the fact both variables are not indirectly connected. That's what is a chain rule. To be more specific if

For func 1.:

dHeight/dWeight = 2

Since the function is constant:

Height = dHeight/dWeight * Weight = 2 * Weight

For func 2.:

dSize/dHeight = ¼

since the function is constant:

Size = dSize/dHeight * Height = ¼ * Height

For Chain Rule:

Since all 3 variables are directly connected we can plug one equation into another:

Size = dSize/dHeight * dHeight/dWeight * Weight

Now if we want to determine exactly how Shoe Size changes with respect to changes in Weight we can do this:

dSize/dWeight = dSize/dHeight * dHeight/dWeight - taken from previous equation (I've just removed Weight from right side because we're using it in the derivative)

dSize/dWeight = ¼ * 2 = ½

what means with every 1 unit increase of weight we note ½ unit increase of shoe size. That's how we've connected 2 variables size and weight with one common variable, which was height.

Example 2.

If we have function 1. Hunger = Time^2 + ½ and function 2. IceCream = sqrt(Hunger) we can try to connect time with craving of ice creams.

For func 1.:

dHunger/dTime = 2Time

For func 2.:

dIceCream/dHunger = ½*(1/sqrt(Hunger))

For Chain Rule:

So dIceCream/dTime = dIceCream/dHunger * dHunger/dTime

Now let's plug Hunger equation into IceCream equation:

IceCream = sqrt(Time^2 + ½)

dIceCream/dTime = ½*(1/sqrt(Time^2 + 1/2)) * 2Time = 1/sqrt(Time^2 + ½) * Time

That's how we can connect ice creams craving with time.

Similarly we operate with residual and intercept of desired model within loss function.

Gradient Descent Intuition

Let's say we have function Predicted_Height = intercept + slope * Weight that tries to fit line to data. We don't know intercept and the slope just yet. Gradient descent is for setting up intercept and the slope to fit line to the data as best as it's possible.

1. We pick random line, e.g. with intercept 0 (starting from center)
2. for given x we quantify residual between our prediction of y and actual y
3. We quantify Sum of squared residuals for all of the points from training set:

   Sum of squared residuals =
   Sum((observed - predicted)^2) =
   Sum((observed - (intercept + slope * x))

4. We place sum of square residuals on plot where y is Sum of squared residuals and x is Intercept
5. By tweaking intercept and the slope can see change of Sum of squared residuals value. Since we're having loss function with 3 variables (sum of squared residuals, intercept and slope) we're working on 3d function.
6. The goal is to get to the lowest point of the plot from point 4
7. To achieve lowest point we use Gradient Descent, which takes small steps towards the lowest point. Using Least Square method we could simply use slope of the function which would have to equal 0. In our example we're using Chain Rule to define slope and intercept.
8. Gradient Descent uses steps, the question is how big steps. Size of step is related to the derivative of the function (slope), if the slope is big the steps would be also big. When the slope is decreasing the steps are also decreasing, because that means we're getting to the lowest point. At the end we're taking tiny steps to get to the optimal minimum of the loss function. This step is equal slope * learning rate. Thus we should start with bigger learning rate and finish with small learning rate.
9. Gradient Descent stops when step size is very close to 0 or maximum amount of steps was achieved (usually 1000 steps, but can be changed)

Stochastic Gradient Descent is similar, the difference is that it uses random data points that will speed up training process.

Backpropagation Intuition

Backpropagation is about optimizing all parameters, weights and biases. It tries to optimize all parameters starting from the last one (from the back). If we have last parameter which is bias20, we quantify the prediction without this bias term. Then we quantify the residuals for this unbiased function (assuming our bias20 has some initial value, usually it's 0, so unchanged). We repeat the process for all data points and quantify SSR (Sum of Squared Residuals). If b20 = 0 we get some SSR, let's say 21. Then we put this value on the plot where y axis is SSR and the x axis is b20. The goal is to find this parameter value (in this case b20), where SSR is the lowest. For this we're using Gradient Descent, what means we need to find the derivative of the SSR with respect to b20.

we need to find: dSSR / db20

we need to remember that: SSR = Sigma(Observedi-Predictedi)^2

we also need to remember that: Predictedi = prediction + b20

then we can apply Chain Rule: dSSR / db20 = dSSR / dPredicted * dPredicted / db20 - that returns the slope where b20 = 0

we need to input this slope to this equation: Step Size = slope * learning rate (Gradient Descent).

After we receive step size we apply it to b20 and receive new b20 value: New b20 = 0 - step size

We repeat this process with new b20 and do it as long our SSR will be near 0 (as in Gradient Descent)

This process is done for each parameter, we just need to remember that we're going from the back, not from the beginning.