Recap: A critical ingredient for our new approach: **Loss functions**

A function $L$ which measures how “wrong” a network is
Mean Squared Error (MSE)

Average squared residual (residual: difference between predicted and true value)

Decreasing the MSE = the model has less error = data points fall closer to the regression line

\[
MSE = \frac{\sum_{k=1}^{n} (y^k - \hat{y}^k)^2}{n}
\]

\(y^k\): true output value
\(\hat{y}^k\): predicted output value
\(n\): number of samples

Courtesy: https://statisticsbyjim.com/regression/mean-squared-error-mse/
Cross Entropy Loss (for Binary classification)

\[ y = \text{true label of class } (0 \text{ or } 1) \]
\[ p = \text{predicted probability of class } 1 \]

\[-(y \log(p) + (1 - y) \log(1 - p))\]

Some examples:

- \[ y = 1, p = 0.9 \]
- \[ y = 0, p = 0.9 \]
- \[ y = 1, p = 0.001 \]
- \[ y = 0, p = 0.001 \]

We get this probability by using a Sigmoid function

\[
\begin{align*}
\log(0.9) &= -0.04 \\
\log(0.5) &= -0.3 \\
\log(0.001) &= -3
\end{align*}
\]
Cross Entropy Loss (for Multi-class classification)

\[- \sum_{j=1}^{m} y_j \log(p_j)\]

<table>
<thead>
<tr>
<th>p</th>
<th>Classes (m)</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>“0”</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>“1”</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>“2”</td>
<td>1</td>
</tr>
</tbody>
</table>

Some examples:
\[
\log(0.9) = -0.04 \\
\log(0.5) = -0.3 \\
\log(0.001) = -3
\]

We can get these probabilities by using a Softmax function.
Next critical ingredient for our new approach: Optimizer
Today’s goal – learn about the optimizer

(1) What does it mean to optimize?
(2) Gradient descent for linear regression
(3) Start building a neural network
(4) Calculating gradients for composite functions (Chain rule)
What does it mean to optimize?

“Optimization” comes from the same root as “optimal”, which means best. When you optimize something, you are “making it best”.

For our case, we want to minimize the loss function to get the “best” model!
What does it mean to optimize?

1. Calculate the parameter update values
2. Update the parameters

Optimal solution

![Graph showing optimal solution](image)
Gradient (measuring the change)

Calculating partial derivative of the Loss
with respect to the weights/parameters
Vector Calculus Recap

• Partial derivative: the derivative of a multivariable function with respect to one of its variables

Example: $f(x, w, b) = wx + b$.

The partial derivative of $f$ with respect to $w$ is

$$\frac{\partial f}{\partial w} = \frac{\partial}{\partial w} (wx + b) = x + 0 = x.$$
Vector Calculus Recap

• Partial derivative: the derivative of a multivariable function with respect to one of its variables

• Example: \( f(x, w, b) = wx + b \)

• The partial derivative of \( f \) with respect to \( w \) is \( \frac{\partial f}{\partial w} \)
Vector Calculus Recap

• Partial derivative: the derivative of a multivariable function with respect to one of its variables
• Example: \( f(x, w, b) = wx + b \)
• The partial derivative of \( f \) with respect to \( w \) is \( \frac{\partial f}{\partial w} \)
• How to compute? -- treat all other variables as constants and differentiate

\[
\frac{\partial f}{\partial w} =
\]
Vector Calculus Recap

• Partial derivative: the derivative of a multivariable function with respect to one of its variables

• Example: $f(x, w, b) = wx + b$

• The partial derivative of $f$ with respect to $w$ is $\frac{\partial f}{\partial w}$

• How to compute? -- treat all other variables as constants and differentiate

\[
\frac{\partial f}{\partial w} = \frac{\partial}{\partial w} (wx + b) = \frac{\partial}{\partial w} (wx) + \frac{\partial}{\partial w} (b) = x + 0 = x
\]
Gradient Descent

$$\Delta w = -\alpha \cdot \frac{\partial L}{\partial w}$$

Learning rate

Slope

Optimal solution

Courtesy: https://statisticsbyjim.com/regression/mean-squared-error-mse/
Impact of Learning Rate

Learning rate too small?  
**Slow Convergence**  
\( \alpha = 10^{-8} \)

Learning rate too big?  
**Instability**  
(“overshooting”)  
\( \alpha = 10^{-1} \)

\[ \Delta w = -\alpha \cdot \frac{\partial L}{\partial w} \]
Gradient Descent (updating parameters)

\[ \Delta w = -\alpha \cdot \frac{\partial L}{\partial w} \]

Learning rate

Slope

Optimal solution

\[ w_{new} = w_{old} - \alpha \cdot \frac{\partial L}{\partial w} \]

Courtesy: https://statisticsbyjim.com/regression/mean-squared-error-mse/
Recap: Mean Squared Error (MSE)

Average squared residual (residual: difference between predicted and true value)

Decreasing the MSE = the model has less error = data points fall closer to the regression line

\[ MSE = \frac{\sum_{k=1}^{n} (y^k - \hat{y}^k)^2}{n} \]

- \( y^k \): true output value
- \( \hat{y}^k \): predicted output value
- \( n \): number of samples

What could be the purpose of squaring the distance?

Courtesy: https://statisticsbyjim.com/regression/mean-squared-error-mse/
Gradient Descent of MSE (1 sample)

\[ L = (y - \hat{y})^2 \]

\[ = (y - f(x))^2 \]

\[ = y^2 + f(x)^2 - 2yf(x) \]

\[ = y^2 + (wx + b)^2 - 2y(wx + b) \]

\[ = y^2 + w^2x^2 + b^2 + 2wxb - 2ywx - 2yb \]

\[ \frac{\partial L}{\partial w} = ? \]

\[ \frac{\partial L}{\partial w} = 2wx^2 + 2xb - 2yx \]

\[ \frac{\partial L}{\partial w} = 2x(wx + b - y) \]

\[ \Delta w = -\alpha \cdot \frac{\partial L}{\partial w} \]

\[ w_{new} = w_{old} - \alpha \cdot \frac{\partial L}{\partial w} \]

Any questions?
Convex functions

Figure: https://fmin.xyz/docs/theory/Convex_function/
Convex and Non convex functions

Why do we care about non-convex functions?

Figure: https://fmin.xyz/docs/theory/Convex_function/
Why we care about non-convex functions?

A Multi-Layered Neural Net

Gradient descent can help the neural net learn!
Let’s start building our neural network model

• This is a simplified view of our model with an input and a linear layer

\[ l_j = \sum_k W_{j,k} x_k + b_j \]

Product of weight matrix with input vector
Our Weight Matrix

- We have an input vector of size $n$ and an output vector of size $m$, so our weights matrix $W$ is of dimensionality $m \times n$.
- $w_{j,i}$ is the $j^{th}$ row and the $i^{th}$ column of our matrix, or the weight multiplied by the $i^{th}$ index of the input which is used to create the $j^{th}$ index in the output.
Our Weight Matrix [Example]

\[ x = [x_1 \ x_2] \]
Adding MSE Loss to Our Network

\[ l_j = \sum_k w_{j,k} x_k + b_j \]

\[ l = w \cdot x + b \]

\[ L = (y - l)^2 \]
Looking at composite function!
Using gradient descent to update parameters

• Recall the parameter update for Gradient Descent: $\Delta w = -\alpha \cdot \frac{\partial L}{\partial w}$

• $L$ is a composition of a series of functions (linear layers, loss layer, maybe more...)

• How do we compute the derivative of a composition of functions?
  • Hint: think back to your calculus classes...
Chain rule

If \( f \) and \( g \) are both differentiable and \( F(x) \) is the composite function defined by \( F(x) = f(g(x)) \) then \( F \) is differentiable and \( F' \) is given by the product

\[
F'(x) = f'(g(x)) \cdot g'(x)
\]
Applying Chain rule [Example]

\[ f(x) = x^2 \quad \text{g}(x) = (2x^2 + 1) \]

\[ F(x) = f(g(x)) \]

\[ F(x) = (2x^2 + 1)^2 \]
The Chain Rule (for Differentiation)

• Given arbitrary function: \( f(g(x)) \Rightarrow \frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx} \)

Backpropagation
(or backward pass)

Each layer computes the gradients with respect to its variables and passes the result backwards.
The Chain Rule in Our Network

- Here’s our function: \( L(l(w)) \Rightarrow \frac{dL}{dw} = \frac{dL}{dl} \cdot \frac{dl}{dw} \)

Remember – We calculate the gradient of L with respect to the parameters for learning them using gradient descent!
Derivative of loss layer

\[
\frac{dL}{dl} = \frac{d(y-l)^2}{dl}
\]

\( L = (y - l)^2 \)
Derivative of linear layer

\[
\frac{dl}{dw} = \frac{d(wx+b)}{dw}
\]

\[l = wx + b\]

\[L = (y - l)^2\]

\[\Sigma\]
Putting it all together

\[ \frac{dL}{dw} = \frac{dL}{dl} \cdot \frac{dl}{dw} = \]

\[ l = wx + b \]

\[ \frac{dh}{dw} \]

\[ \frac{dL}{dl} \]
Putting it all together

\[ \frac{dL}{dw} = \frac{dL}{dl} \cdot \frac{dl}{dw} = -2(y - 1).x = -2x(y - wx - b) = 2x(wx + b - y) \]

Have we seen this before?
Gradient Descent of MSE (1 sample)

\[ L = (y - \hat{y})^2 \]

\[ = (y - f(x))^2 \]

\[ = y^2 + f(x)^2 - 2yf(x) \]

\[ = y^2 + (wx + b)^2 - 2y(wx + b) \]

\[ = y^2 + (wx + b)^2 - 2y(wx + b) \]

\[ = y^2 + (wx + b)^2 - 2y(wx + b) \]

\[ = y^2 + w^2x^2 + b^2 + 2wxb - 2yw - 2yb \]

\[ \frac{\partial L}{\partial w} = 2wx^2 + 2xb - 2yx \]

\[ \frac{\partial L}{\partial w} = 2x(wx + b - y) \]

\[ \Delta w = -\alpha \cdot \frac{\partial L}{\partial w} \]
Adding more layers!

\[ f(h(g(x))) \Rightarrow \frac{df}{dx} = \frac{df}{dh} \cdot \frac{dh}{dg} \cdot \frac{dg}{dx} \]

Can we add any function?

Any questions?
Few more important points: Backpropagation

- The process of calculating gradients of functions via chain rule in a neural network.
- Is a part of and **NOT** the whole learning algorithm.
- Can be calculated with respect to any variable of choice.
- For **learning in neural networks** we calculate gradients with respect to the weights.
Recap

Optimization

- Calculating gradients
- Gradient Descent for MSE
- Convex and Non-convex functions

Building a neural network

- Simple model with linear layer
- Adding loss layer (regression)
- Chain rule to calculate gradients (Backpropagation)