CS 182 Lecture 3: Initialization and Regularization

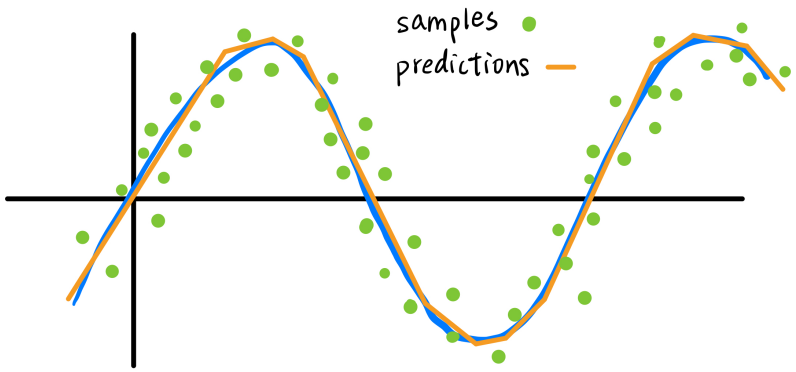
September 1, 2022

Lecturer: Anant Sahai — Scribes: Shreyas Krishnaswamy

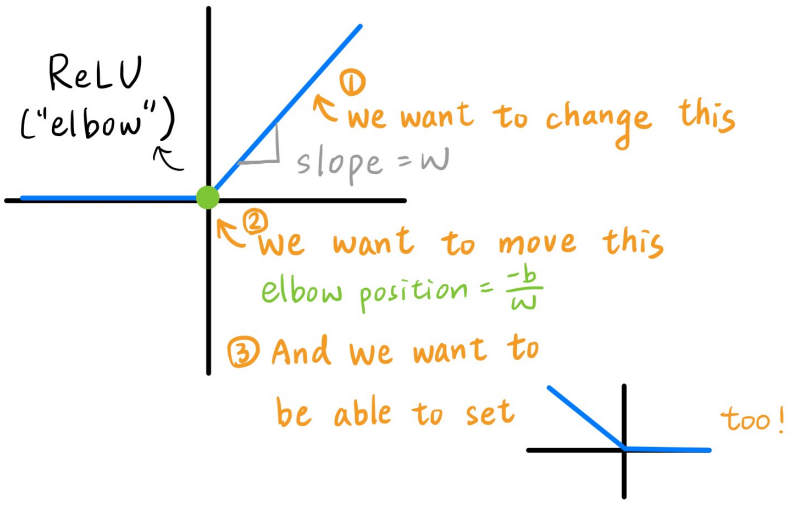
1 Finish up Basic ReLU Net

Neural Nets are easily (for a computer) differentiable function approximators.

1.1 Approximation (e.g., piecewise linear)

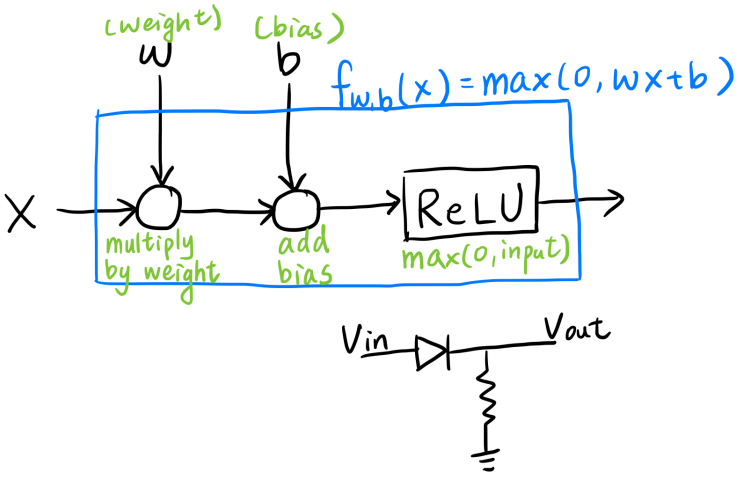


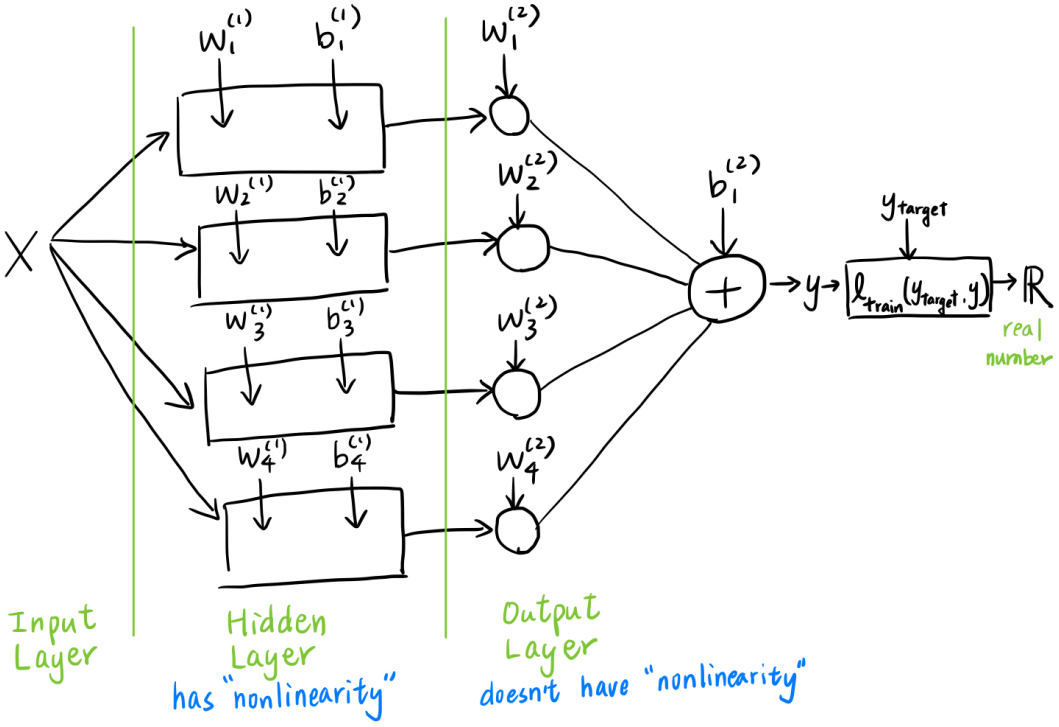
- How do you represent piecewise linear functions in a way that is differentiable for a computer?
  → Build out of "elbows"!



- How does the "elbow" depend on w (weight) and b (bias)?
  → Elbow located at $-\frac{b}{w}$ and slope $= w$

- How do you move the "elbow"?
  → Change w and/or b.





- The optimizer is trying to make the final real number (loss) as small as possible across all the training points. To push the loss down, how much do I have to move y, b's, and w's? This process goes backward and the elbow changes.

1.2 Initialization

Current Basic Folk Wisdom:

- Use whatever worked on a related problem. (e.g., pre-trained network, literature review, etc)
- Random initialization using Gaussian $N(0, \sigma^2)$.
    * Xavier Initialization: $\sigma^2 = \frac{1}{d}$, where d is the fan-in of this unit.
    * $Var(\sum_{i=1}^{d} X_i) = dk$ (X_i are independent, has $mean = 0$ and $var = k$). Note that all variances add up to 1.

Xavier initialization is appropriate when the nonlinearity is not a ReLU (e.g. hyperbolic tangents, sigmoids, etc.). For ReLUs, **He initialization** (a.k.a. Kaiming initialization) is appropriate.

He Initialization

$$\text{Gaussian } N\left(0, \frac{2}{d}\right)$$

Why $\frac{2}{d}$ instead of $\frac{1}{d}$? When initializing ReLU weights, we want our ReLU elbows to be close to where the action is. In other words, we want the ReLUs to actually produce non-linearities, so our model can emulate nonlinear phenomena.

If this is the case, about half of our ReLUs should be in the off-state, so the actual fan-in for ReLUs is halved, which is why there's a two in the numerator of $\frac{2}{d}$.

There are many possibilities for initializing the biases. Some include:

- Xavier initialization on the biases
    * Motivation: the bias can be considered another weight. Instead of using d, use $d + 1$ (i.e. add the bias's fan-in) when initializing the corresponding unit.
- Make them all 0
- Make them all small random numbers
- Make them all the same small number (e.g. 0.01)

For different situations, some of these work better than others.

1.3 Aside: Dead ReLUs

If both weights and biases are distributed with normal distributions, the ratio $\frac{b}{w}$ will be a Cauchy distribution. This causes some ReLUs to be located far apart from the others. These ReLUs are considered "dead" since they output 0, and changing the weights or biases slightly doesn't affect their output.

2 Revisiting Regularization

2.1 Regularization in the Loss Function

This section explores **regularization**. Let's begin by looking at general loss functions with and without regularization.

Loss function without regularization

$$L_\theta = \frac{1}{n}\sum_{i=1}^{n} \ell_{train}\left(y_i, f_\theta\left(x_i\right)\right)$$

$x_i = i$th data point (model input)
$y_i =$ Ground truth (i.e. expected) output for data point x_i
$f_\theta =$ Model with parameters θ
$n =$ Number of training data points
$\ell_{train} =$ Loss on training data point i
$L_\theta =$ Total loss

This function tracks the loss between our parameterized model's outputs and the expected outputs. We can add a regularization function to it to create a loss function with regularization.

Loss function with regularization

$$L_\theta = \frac{1}{n}\sum_{i=1}^{n} \ell_{train}\left(y_i, f_\theta\left(x_i\right)\right) + R(\theta)$$

$R(\theta) =$ Regularization term on model parameters θ

2.2 Regularization in Least Squares

For a more intuitive understanding of regularization, we can look at least squares. Let's start by reviewing the problem: we have a matrix X and vector $\vec{y}$, and we want to calculate the weights $\vec{w}$ that best approximates $X\vec{w} = \vec{y}$.

Ordinary Least Squares (OLS)

Problem: $\text{argmin}_w \; ||\vec{y} - X\vec{w}||^2$

Solution: $\hat{\vec{w}} = \left(X^T X\right)^{-1} X^T \vec{y}$

We can add regularization to OLS. One version of regularized OLS is called ridge regression:

Ridge Regression

$$\text{Problem: } \text{argmin}_w \ ||\vec{y} - X\vec{w}||^2 + \lambda||\vec{w}||^2$$

$$\text{Solution: } \hat{\vec{w}} = \left(X^T X + \lambda I\right)^{-1} X^T \vec{y}$$

λ is the **regularization parameter**. It penalizes high-magnitude weight vectors (can you see why?). We can select different λ values to control the behavior of ridge regression. When λ is higher, the weight magnitude penalty is more severe; when it's lower, the penalty is lighter, which allows ridge regression to settle on weight vectors with higher magnitudes.

Note that when $\lambda = 0$, ridge regression becomes unregularized. This unregularized formulation simplifies to the OLS problem and solution. Intuitively, setting $\lambda = 0$ tells ridge regression to disregard the weight vector's magnitude altogether, allowing it to solve the problem as though it were simply OLS.

2.2.1 Gradient Descent

We can also solve least squares with **gradient descent**. Gradient descent is a process where we aim to improve our model by repeatedly moving its parameters in the direction that minimizes the cost function. For least squares, the parameters are stored in $\hat{\vec{w}}$.

The gradient for least squares without regularization is

OLS Gradient Descent

$$\text{Gradient (with step size): } \eta 2X^T \left(\vec{y} - X\hat{\vec{w}}\right)$$

$$\text{Update step: } \hat{\vec{w}}_{(t+1)} = \hat{\vec{w}}_{(t)} + \eta 2X^T \left(\vec{y} - X\hat{\vec{w}}_{(t)}\right)$$

$$\begin{aligned}
\hat{\vec{w}}_{(t+1)} &= \text{Weights } \hat{\vec{w}} \text{ at timestep } t+1 \\
\hat{\vec{w}}_{(t)} &= \text{Weights } \hat{\vec{w}} \text{ at timestep } t \\
\eta &= \text{Step size} \\
\vec{y} - X\hat{\vec{w}}_{(t)} &= \text{Residual at timestep } t
\end{aligned}$$

We can create a similar update step for ridge regression:

Ridge Regression Gradient Descent

$$\text{Gradient (with step size): } = \eta \left(2X^T \left(\vec{y} - X\hat{\vec{w}}\right) - 2\lambda\hat{\vec{w}}\right)$$

$$\text{Update step: } \hat{\vec{w}}_{(t+1)} = (1 - 2\eta\lambda)\, \hat{\vec{w}}_{(t)} + \eta 2X^T \left(\vec{y} - X\hat{\vec{w}}_{(t)}\right)$$

Note that the ridge regression update step can be derived by rearranging the gradient and adding it to the weights at time t.

The coefficient $(1 - 2\eta\lambda)$ causes an effect called **weight decay**. On top of adding an update, the regularized update also decays the weights, making the system of gradient descent more stable.

2.3 Data Augmentation View

We can also look at regularization through the **data augmentation** lens. Suppose we want to solve an OLS problem where we have appended fake data points to our X matrix. For example:

Data-Augmented System

$$\begin{bmatrix} X \\ \sqrt{\lambda}I \end{bmatrix} \vec{w} \approx \begin{bmatrix} \vec{y} \\ \vec{0} \end{bmatrix}$$

Dimensions

$$\begin{aligned} X &: n \times d \\ \sqrt{\lambda} &: \text{constant} \\ I &: d \times d \ \text{(identity)} \\ \vec{w} &: d \times 1 \\ \vec{y} &: n \times 1 \\ \vec{0} &: d \times 1 \end{aligned}$$

$\sqrt{\lambda}I$ represents some fake data. We append zeros $\left(\vec{0}\right)$ to $\vec{y}$ in order to match our output's dimensions to the augmented X matrix. Once again, λ is a regularization parameter.

We can apply the OLS solution to this system:

$$\left(\begin{bmatrix} X \\ \sqrt{\lambda}I \end{bmatrix}^T \begin{bmatrix} X \\ \sqrt{\lambda}I \end{bmatrix} \right)^{-1} \begin{bmatrix} X \\ \sqrt{\lambda}I \end{bmatrix}^T \begin{bmatrix} \vec{y} \\ \vec{0} \end{bmatrix} = \left(X^T X + \lambda I\right)^{-1} X^T \vec{y}$$

The output matches the ridge regression solution!

What's happening here? Intuitively, each of the data points added to the X matrix is trying to make the corresponding feature go to 0 in order to fit to the zeros we appended to $\vec{y}$ – just like a regularizer! Recall that ridge regression similarly penalized high-magnitude weight vectors, which also added pressure to reduce the weight vector's magnitude.

The data augmentation view is useful because it's intuitive to generalize. We can add modified or fake data like we did in this example to other machine learning problems or architectures in order to regularize them.

2.4 Feature Augmentation

Another approach to regularization is **feature augmentation**, where we add irrelevant features to our system.

Feature-Augmented System

$$\begin{bmatrix} X & \sqrt{\lambda}I \end{bmatrix} \begin{bmatrix} \vec{w} \\ \vec{f} \end{bmatrix} = \vec{y}$$

Dimensions

$$\begin{aligned} X &: n \times d \\ \sqrt{\lambda} &: \text{constant} \\ I &: n \times n \ \text{(identity)} \\ \vec{w} &: d \times 1 \\ \vec{f} &: n \times 1 \\ \vec{y} &: n \times 1 \end{aligned}$$

$\sqrt{\lambda}I$ represents some fake features that we have added to the X matrix. To make the dimensions match, we have appended fake weights $\vec{f}$ (that we don't care about) to $\vec{w}$ (which we do care about).

Note that since the augmented X matrix is wider than it is tall, the problem uses strict equality. This also means the system has infinitely many solutions, and we need to pick one of them.

For now, let's use the Moore-Penrose Pseudoinverse to find a minimum-norm solution to this problem.

$$\begin{aligned} \begin{bmatrix} \hat{\vec{w}} \\ \hat{\vec{f}} \end{bmatrix} &= \begin{bmatrix} X^T \\ \sqrt{\lambda}I \end{bmatrix} \left(\begin{bmatrix} X & \sqrt{\lambda}I \end{bmatrix} \begin{bmatrix} X^T \\ \sqrt{\lambda}I \end{bmatrix} \right)^{-1} \vec{y} \\ \hat{\vec{w}} &= X^T \left(XX^T + \lambda I \right)^{-1} \vec{y} \end{aligned}$$

We arrive at the second expression above by discarding the fake weights $\hat{\vec{f}}$ since we're only looking for the weights stored in $\hat{\vec{w}}$.

Note that the final expression for the $\hat{\vec{w}}$ vector looks a bit different than the OLS solutions from the other setups. This expression is actually the dual perspective of ridge regression, called **kernel ridge regression**.

This means that augmenting with features actually has a regularizing effect. This phenomenon is one of the reasons people say deep neural networks are effective. Not only are they more expressive, but by adding many extra features, they may also be more effective regularizers.