Assignment 1 Writeup

- Name: Bojun Yang
- GT Email: byang301@gatech.edu
- GT ID: 903254309

Two-Layer Neural Network

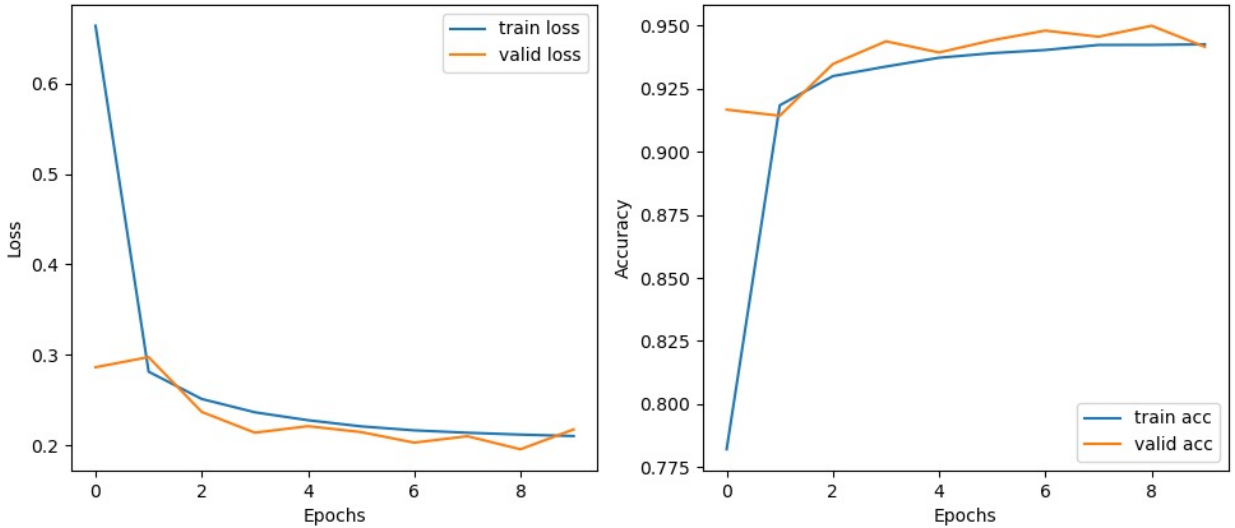
1. Learning Rates

Tune the learning rate of the model with all other default hyper-parameters fixed. Fill in the table below:

	lr=1	lr=1e-1	lr=1e-2	lr=5e-2
Training Accuracy	0.9424	0.9223	0.7295	0.9087
Test Accuracy	0.9495	0.9248	0.7623	0.9127

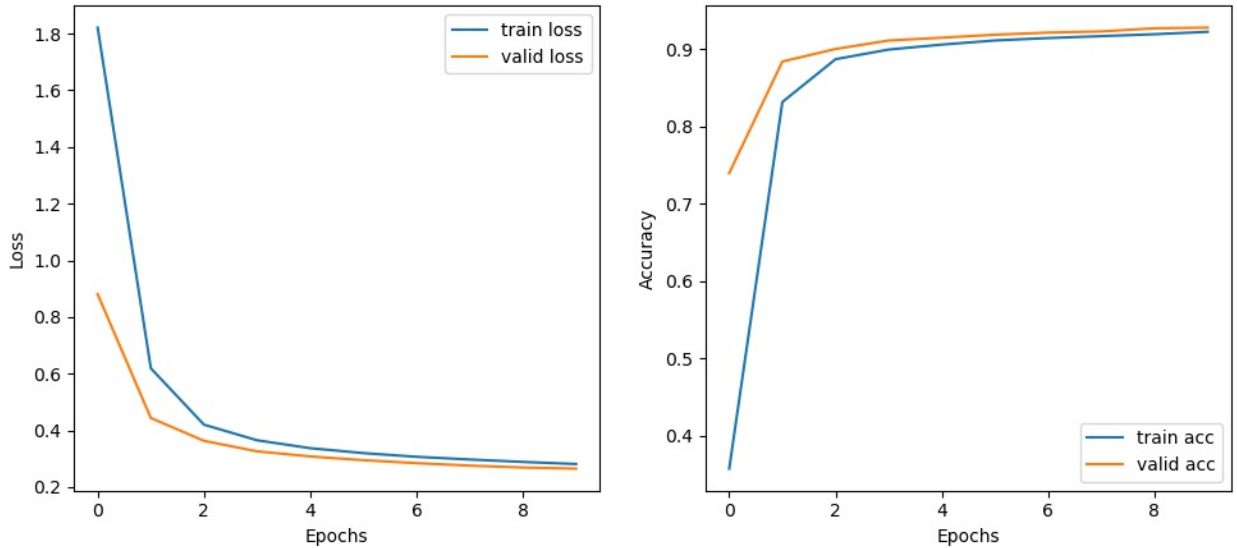
1. Learning Curve

Lr = 1



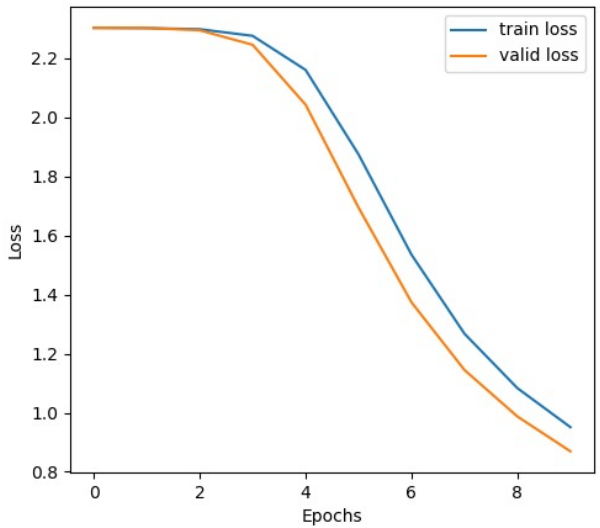
1. Learning Curve

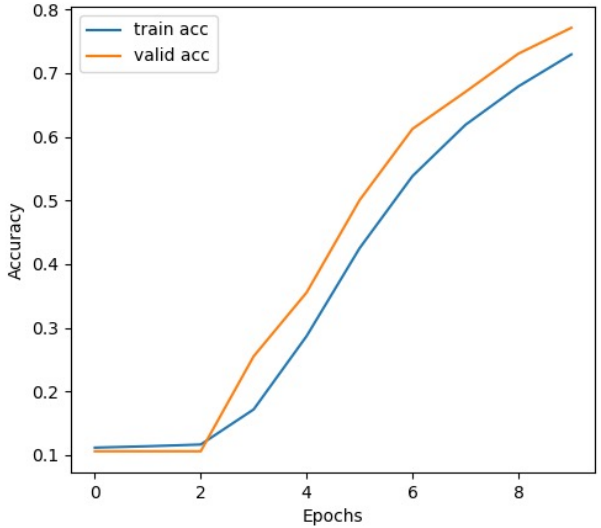
Lr = 1e-1



1. Learning Curve

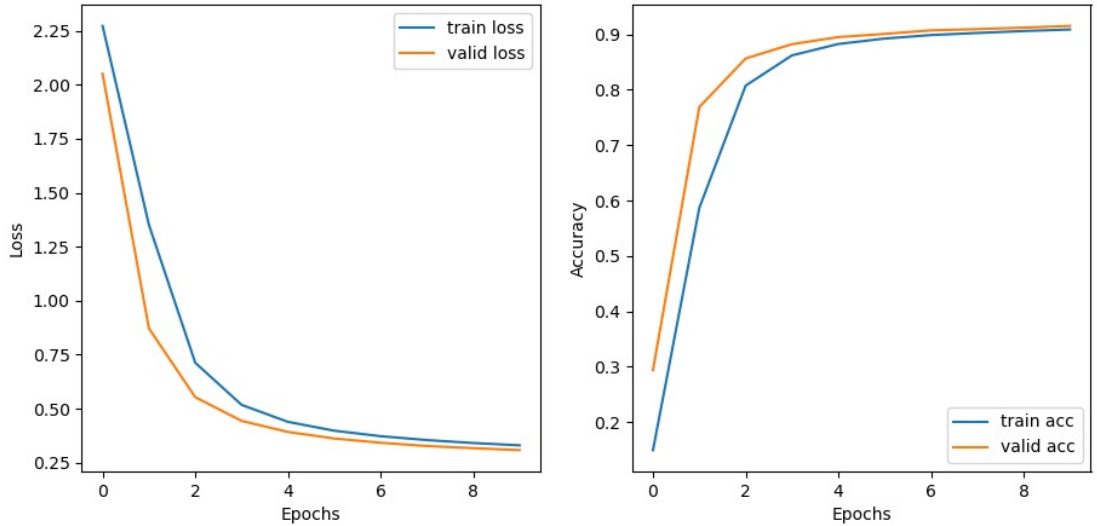
Lr = 1e-2





1. Learning Curve

Lr = 5e-2



1. Learning Rates

The accuracy for lr=1 was the highest out of all the models. However, its loss and accuracy curve are unstable. The validation accuracy rises very fast in the beginning and then goes up and down for the rest. For large learning rates, overfitting is more likely and the model can overshoot the optima if the steps are too large. This is what can cause the up and down learning curves. Since our models only had 10 epochs, a large learning rate meant that it could get its accuracy up fast, but it potentially overshot the optima.

The smaller the learning rate, the slower the accuracy will increase. With smaller learning rates, the model is less likely to overfit but it will take the model longer to reach the optima. For lr=1e-2, the accuracy was the lowest because 10 epochs was not enough for it to reach the optima. However, this model did not overfit as the testing accuracy was generally higher than the training accuracy.

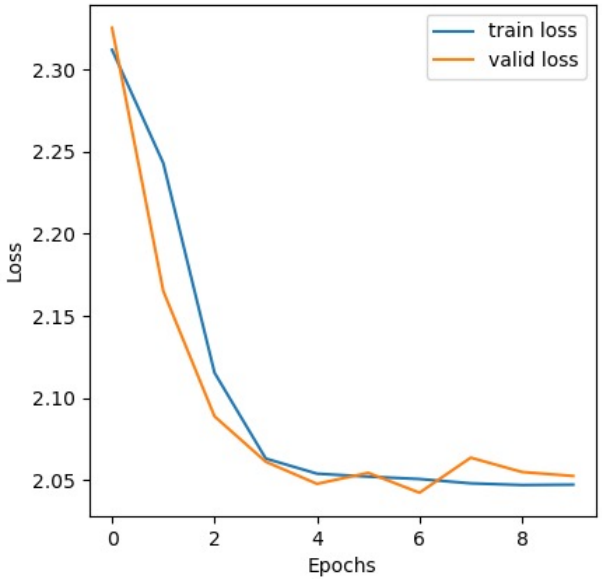
2. Regularization

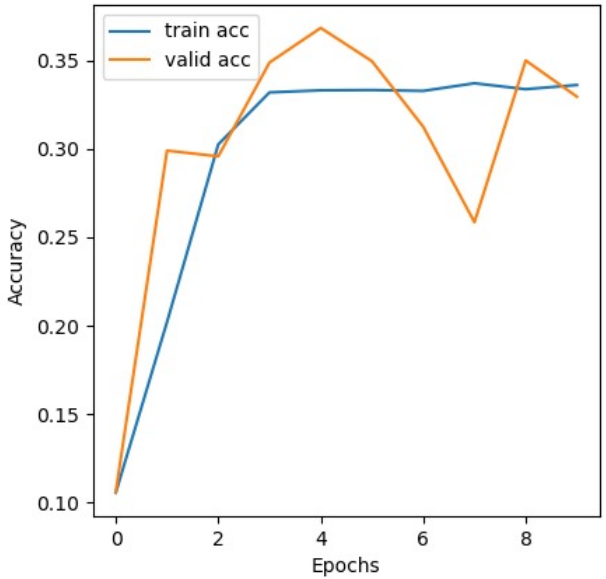
Tune the regularization coefficient of the model with all other default hyper-parameters fixed. Fill in the table below:

	alpha=1e-1	alpha=1e-2	alpha=1e-3	alpha=1e-4	alpha=1e-0
Training Accuracy	0.3319	0.8844	0.9212	0.9299	0.0989
Validation Accuracy	0.3685	0.8945	0.9282	0.9358	0.106
Test Accuracy	0.3776	0.8932	0.9256	0.9316	0.1135

2. Regularization

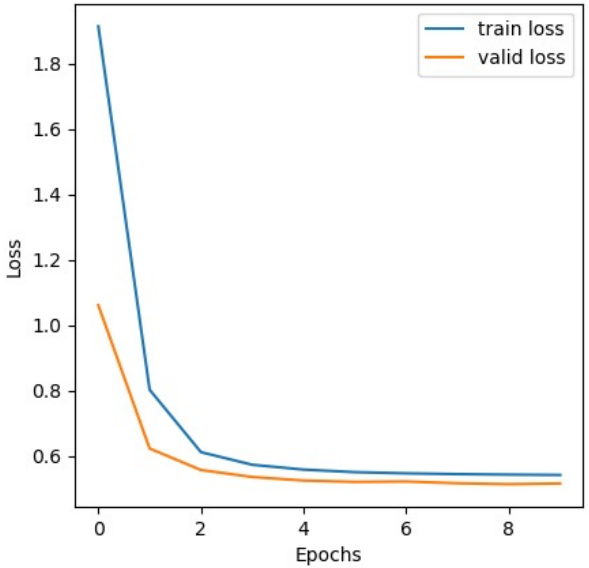
alpha=e-1

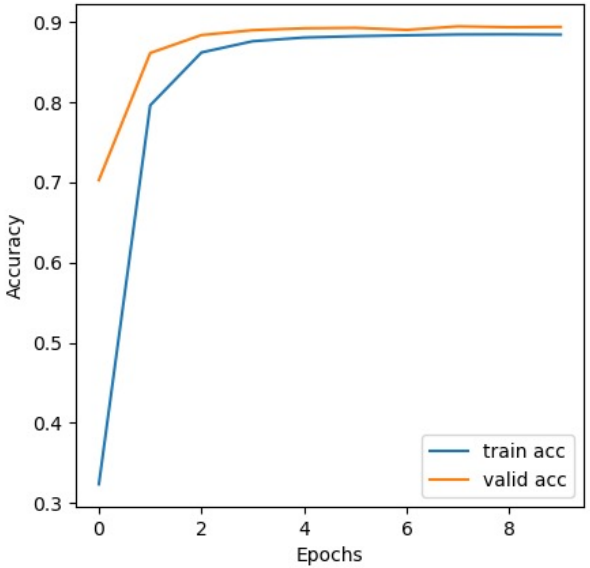




2. Regularization

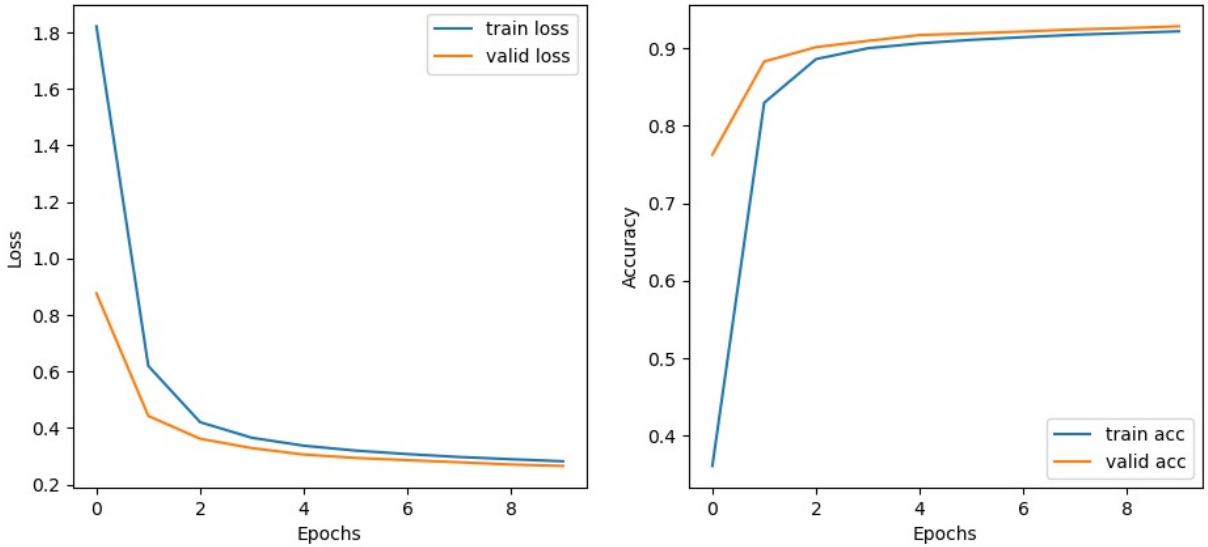
alpha=e-2





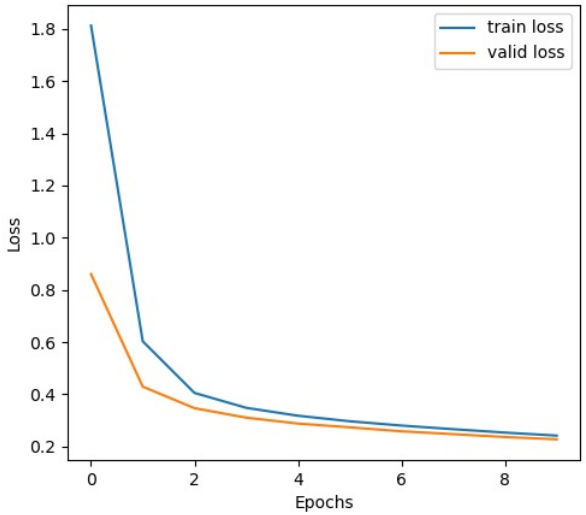
2. Regularization

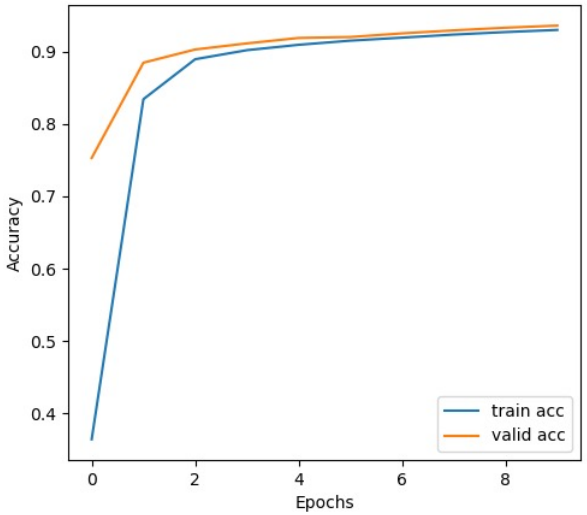
alpha=e-3



2. Regularization

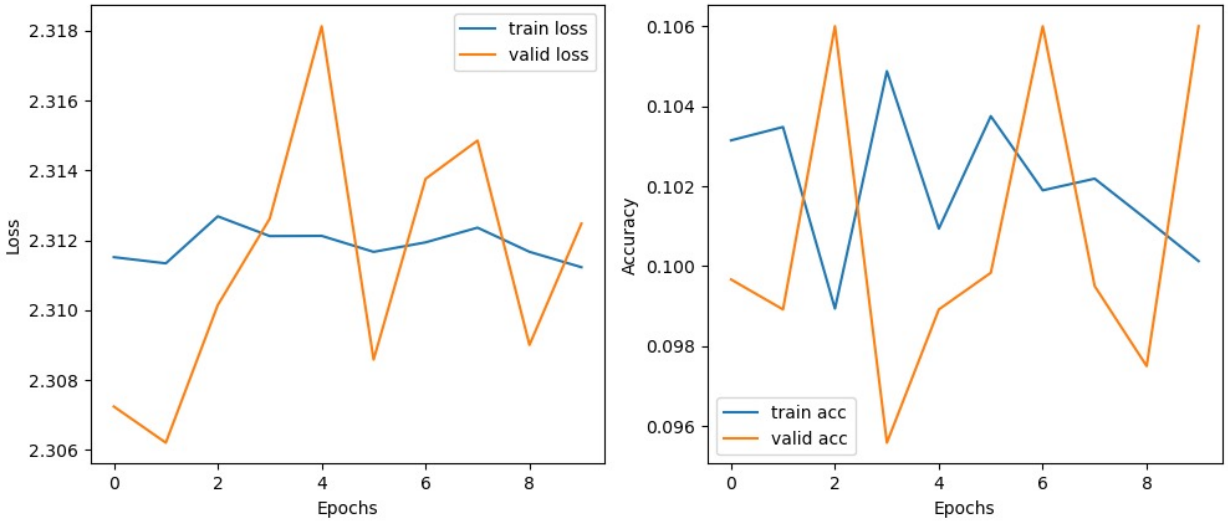
alpha=e-4





2. Regularization

alpha=e-0



2. Regularization

Regularization penalizes higher terms in the model to avoid overfitting. The L2 regularization we implemented basically tries to push the weights towards 0. The higher the regularization coeff, the more likely the model is to underfit, so if the coeff is too small, the model will tend to overfit. We can see this in our graphs where a coeff of 1 is way too big and the model doesn't learn anything, with learning curves of no trend. Based on our graphs, it seems an optimal regularization coeff is around 1e-3 to 1e-4. We still saw some random up and downs with a coeff of 1e-1.

To pair with learning rate, if the learning rate is higher, there is a higher chance of overfitting. To counteract this, we should up the regularization coeff to counteract overfitting.

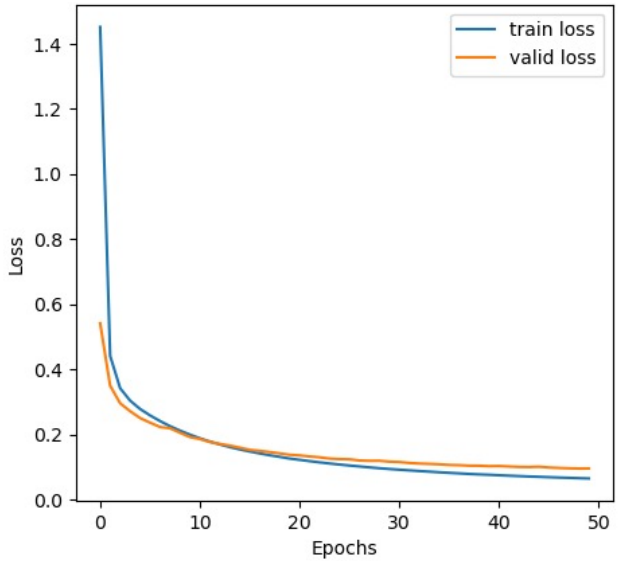
3. Hyper-parameter Tuning

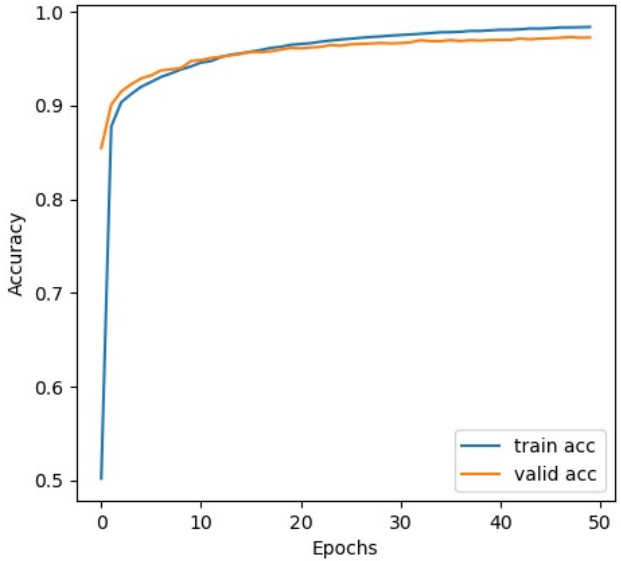
Bach-size	Lr	Reg	Epochs	Momentum
64	0.15	0.0001	50	0.9

Training acc	Validation acc	Test acc
0.9833	0.9729	0.9735

Briefly explain why your choice works:

I experimented with many values at 10 and 20 epochs. I couldn't get the accuracy to go above 95-95% so I increased the number of batches with the best combination of learning rate and regularization coefficient. It didn't seem my model was overfitting too much so I set the regularization coefficient to 1e-4 and bumped the lr up a bit so I didn't have to run too many epochs.


train loss
valid loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
10
20
30
40
50


train acc
valid acc
Accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
10
20
30
40
50

1. let $r(t) = \langle x_1(t), \dots, x_d(t) \rangle$

$r(t_0) = x_0$ for some $t_0 \in \mathbb{R}$

$r(t) \in L_{f(x_0)} \quad \forall t$

$f(x_1, \dots, x_d) = c$ ← level curve

let $g(t) = f(x_1(t), \dots, x_d(t))$

Since curve r lies in the surface level

$r(t_0) = \langle x_{10}, \dots, x_{d0} \rangle \Rightarrow g(t) = f(x_1(t), \dots, x_d(t)) = c$

Differentiate g

$$\frac{dg}{dt} = \frac{df}{dx_1}\Big|_{r(t_0)} \frac{dx_1}{dt}\Big|_{t_0} + \dots + \frac{df}{dx_d}\Big|_{r(t_0)} \frac{dx_d}{dt}\Big|_{t_0} = 0$$

In vector form

$$\underbrace{\left\langle \frac{df}{dx_1}\Big|_{r(t_0)}, \dots, \frac{df}{dx_d}\Big|_{r(t_0)} \right\rangle}_{\nabla f_0} \cdot \underbrace{\left\langle \frac{dx_1}{dt}\Big|_{t_0}, \dots, \frac{dx_d}{dt}\Big|_{t_0} \right\rangle}_{r'(t_0)} = 0$$

$r'(t_0)$ ← tangent of r at t_0

$\Rightarrow \nabla f_0 \cdot r'(t_0) = 0 \Rightarrow \nabla f_0$ is orthogonal to tangent of $r(t)$ at t_0

Because of the orthogonality, we know that the gradient points to the next surface level. In deep learning we want to find t_0 that minimizes the loss, represented by the surface curve. Gradient descent moves the opposite direction of the magnitude of the orthogonal vector, minimizing the function f.

2. $g : \mathbb{R}^n \to \mathbb{R}$

local min at some w^t if $\exists$ some $\gamma > 0$ s.t. $\forall w \in \mathbb{R}^n$, $\|w^t - w\|_2 < \gamma \Rightarrow g(w^t) \leq g(w)$

g has local min at $w^t \Rightarrow \nabla g(w^t) = 0$:

$w^t = \langle w_1, \ldots, w_n \rangle$

let $g_i(w^t)$ be the partial derivative of g w.r.t. w_i, $1 \leq i \leq n$

$\Rightarrow$ for all i s.t. $1 \leq i \leq n$ $g_i(w^t) = 0$

∴ $\nabla g(w^t) = [\ 0\ ]$ ▨

$\nabla g(w^t) = 0 \not\Rightarrow$ g has local min at w^t :

If ∇g is 0, w^t could still be a saddle point

Consider $g = (x^2 - y^2)$ and $w^t = \langle 0, 0 \rangle$

$$\frac{dg}{dx}(w^t) = 2x = 2(0) = 0$$

$$\frac{dg}{dy}(w^t) = -2y = -2(0) = 0$$

$\langle 0,0 \rangle$ is not a local min but $\nabla g(w^t) = 0$

It is not a local min. We can look at the graph of $x^2 - y^2$ or check second derivatives. $g'_x(w^t) = 2$ $g'_y(w^t) = -2$

▨

3. Prove $g: \mathbb{R}^n \to \mathbb{R}$ is convex and $\nabla g(w^*) = 0 \Rightarrow w^*$ is global min

Definition of convex:

$g(\lambda \vec{x}_1 + (1-\lambda)\vec{x}_2) \leq \lambda g(\vec{x}_1) + (1-\lambda) g(\vec{x}_2)$ for arbitrary $\vec{x}_1, \vec{x}_2$

$g(\vec{x}_2 + \lambda(\vec{x}_1 - \vec{x}_2)) \leq g(\vec{x}_2) + \lambda(g(\vec{x}_1) - g(\vec{x}_2))$

$\lambda(g(\vec{x}_1) - g(\vec{x}_2)) \geq g(\vec{x}_2 + \lambda(\vec{x}_1 - \vec{x}_2)) - g(\vec{x}_2)$

$g(\vec{x}_1) - g(\vec{x}_2) \geq \frac{1}{\lambda}\left[g(\vec{x}_2 + \lambda(\vec{x}_1 - \vec{x}_2)) - g(\vec{x}_2) \right]$

As $\lambda \to 0$, the point on the line segment parameterized by λ becomes $\vec{x}_2$

$\therefore$ right hand side becomes $\nabla g(\vec{x}_2)(\vec{x}_1 - \vec{x}_2)$

Sub in w^* for $\vec{x}_2$

$\therefore$ $g(\vec{x}_1) - g(w^*) \geq 0$

$\Rightarrow g(\vec{x}_1) \geq g(w^*)$

$\vec{x}_1$ is all possible points on g so it is w

$\Rightarrow g(w^*) \leq g(w)$ $\forall$ w on g

$\Rightarrow w^*$ is the global minima ▨

4. $S_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$

Gradient of S_i w.r.t. z_j (j^{th} input) $= \frac{dS_i}{dz_j} = \frac{d\,\frac{e^{z_i}}{\sum_k e^{z_k}}}{dz_j}$

Quotient Rule: $f(x) = \frac{g(x)}{h(x)}$ $\quad g(x) = e^{z_i}$ $\quad h(x) = \sum_k e^{z_k}$

$$f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$$

Case $i = j$:

$$\frac{dS_i}{dz_j} = \frac{(e^{z_i}\sum_k e^{z_k}) - (e^{z_i}e^{z_j})}{(\sum_k e^{z_k})^2}$$

$$= \underbrace{\frac{e^{z_i}}{\sum_k e^{z_k}}}_{S_i} - \frac{e^{z_i}e^{z_j}}{(\sum_k e^{z_k})^2}$$

$$= S_i - S_i \cdot S_j = S_i(1 - S_j)$$

Case $i \neq j$:

$$\frac{dS_i}{dS_j} = \frac{0 - e^{z_i}e^{z_j}}{(\sum_k e^{z_k})^2} = -S_i S_j$$

$$\therefore \frac{dS_i}{dS_j} = \begin{cases} S_i(1 - S_j) & \text{if } i = j \\ -S_i S_j & \text{if } i \neq j \end{cases}$$

$$\text{Jacobian} = \begin{bmatrix} S_1(1-S_1) & -S_1S_2 & \cdots & -S_1S_n \\ -S_2S_1 & \ddots & & \\ \vdots & & \ddots & \\ -S_nS_1 & & & S_n(1-S_n) \end{bmatrix}$$

5. $S(x) = \underset{y \in \mathbb{R}^d}{\operatorname{argmin}} \; -x^T y - H(y)$ s.t. $1^T y = 1, \; 0 \le y_i \le 1 \; \forall i$

$$H(y) = -\sum_i y_i \log(y_i)$$

Lagrangian: $L(y, v) = -x^T y - H(y) + v(1^T y - 1)$

KKT conditions of L gives us:

$$\nabla_y L(y, v) = 0 \quad , \quad v(1^T y - 1) = 0$$

$$\nabla_y L(y, v) = \log(y_i) + 1 - x_i + v = 0$$

$$\Rightarrow y_i = e^{-1 + x_i - v} = e^{x_i - 1 - v}$$

$\log(y_i)$ guarantees $y_i > 0 \Rightarrow 0 < y_i \le 1$

constraint $1^T y = 1$:

$$1 = \sum y_i = \sum (e^{x_i - 1 - v}) = \sum (e^{x_i} e^{-1} e^{-v})$$

$$\Rightarrow v = \log \sum e^{x_i - 1}$$

sub in v:

$$y_i = e^{x_i - 1 - v} = e^{x_i} e^{-1} e^{-\log \sum e^{x_i - 1}}$$

$$= \frac{e^{x_i}}{e \sum e^{x_i - 1}} = \frac{e^{x_i}}{e e^{-1} \sum e^{x_i}} = \frac{e^{x_i}}{\sum e^{x_i}} = 1 = 1^T y$$

∎

6. If G DAG $\Rightarrow$ G has topological ordering

$G = (V, E)$ $\quad V = \{v_1, \ldots, v_n\}$ $\quad E = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in V\}$

Topological ordering of $G = \{v_1, \ldots, v_i, \ldots, v_n\}$ s.t. for every edge $e_{ij} = (v_i, v_j)$ $\quad i < j$

Proof by induction:

Base case: $n=1$ 1 node then that single node is the topological order

Inductive Step: Given G_n of size n is a DAG, G_n has topological order

Show if G_{n+1} is a DAG $\Rightarrow$ G_{n+1} has topological order

G_{n+1} is DAG $\Rightarrow$ exists a node in G with no incoming edges

subproof: assume G_{n+1} is DAG and all nodes have incoming edges then there has to be a cycle some where contradicting the DAG property

Find the node, m, in G_{n+1} with no incoming edges

$(G_{n+1} - m)$ is a DAG of size n because removing a node with no incoming preserves the DAG property. $(G_{n+1} - m)$ is a DAG of size n and has a topological ordering t_n. Then the topological ordering of G_{n+1} is $\{m, t_n\}$ since no incoming edges to m, there will be no node with a higher number pointing to m ■

7. If G has topological order $\Rightarrow$ G is DAG

Proof by contradiction:

suppose not: G has topological order but is not DAG, aka a directed cyclic graph

let topological ordering of G be $t = \{v_1, \ldots, v_n\}$

since G is cyclic, there exists at least 1 pair of nodes v_i, v_j s.t. there is a path $v_i \to v_j$ and a path $v_j \to v_i$

In t, either $i < j$ or $j < i$

case $i < j$:

let the path $v_j \to v_i$ be $v_j \to v_{j1} \to \cdots \to v_{jm} \to v_i$

$j < jx \quad \forall\ x \in (1, m)$ in t

we have $i < j$ so there will be a contradiction at v_{jm} that points to v_i where $i < j$, $j < jm$, and $jm < i$

e.g. $i = 1$, $j = 2$, $jm = 3$ then $jm < i \to 3 < 1$

contradiction

case $j < i$:

let path $v_i \to v_j$ be $v_i \to v_{i1} \to \cdots \to v_{im} \to v_j$

$i < ix \quad \forall\ x \in (1, m)$ in t

we have $j < i$ and like the other case $im < j$

$\therefore$ we have $j < i$, $i < im$, and $im < j$

contradiction

All possible cases have been shown to contradict,

$\therefore$ if G has topological order $\Rightarrow$ G is a DAG

$\blacksquare$

Key contributions
The paper proposed a search method for neural network architectures that can perform tasks without weight training. The idea was based on observations of animals that have certain abilities at birth. The search was done by using a single shared weight param for every network connection and evaluating the network on a wide range. This searches for architecture with strong inductive biases and the paper shows neural networks found that can perform tasks effectively although not all efficiently.

Strengths
They showed weight agnostic neural networks achieve much higher than random accuracy on MNIST using random weights and that it can perform reinforcement tasks without weight training. In neural architecture search, the goal was to produce architectures that outperform those by humans once trained. This paper calls attention to the innate structure of networks while before it was all focused on training weights.

Weaknesses
The networks are optimized to perform well over a single weight so it is highly sensitive to the initial weight. The WANNs fail when individual weight values are assigned randomly, so the weight still has an effect. The search method proposed in the paper is also constricted to the space of neural architectures that gradient descent is able to train. Models that rely on discrete components or utilize adaptive computation mechanisms are challenging.

My Takeaways
I had participated in research of evolutionary based automated neural network search where genes were network components/layers. Our research used survival of the fittest and genetics to search space of neural networks which reminded me of the at birth concept in the paper. If my evolutionary NAS was initiated with weight agnostic neural networks it would be interesting to see how networks with strong inductive biases evolve as the strong genes (components) are passed on.