הסבר לתרגיל 3 בקורס "למידת מכונה"

נכתב על ידי תומר גיל
עבור הקורס 89-511 באוניברסיטת בר-אילן
שמעביר ד"ר יוסי קשת
9/5/2018

תודות מיוחדות: תמיר בוסקילה וטליה בן סימון

Contents

הקדמה 2
תיאור התרגיל 3
טעינת המידע 3
מה יש במידע? 3
מה כל dataset אומר? 3
יצירת פרמטרים ובחירת היפר-פרמטרים 4
פרמטרים 4
היפר-פרמטרים 4
פסאודו-קוד 5
אימון הרשת 5
Forward Pass – חישוב תוצאת הרשת 6
חישוב Loss 6
Back Propagation – חישוב הגרדיאנטים 6
עדכון המשקלים 7
חישוב דיוק ו-loss על ה-dev_set 8
סיימנו? 8
Predicting on the Test Set 8
בעיות, פתרונות וטיפים 9

הקדמה

ראיתי שיש להרבה אנשים בעיה עם התרגיל, אז כתבתי הסבר "קצר" על מה עושים בתרגיל, או לפחות המסגרת של מה אמורים לעשות.

אני לא לוקח שום אחריות על איך התרגיל שלכם יצא אם תעבדו לפי המסמך הזה. השימוש על אחריותכם בלבד.

השתדלתי לחלק את התרגיל לחלקים לוגיים שונים, כדי שיהיה ברור מה צריך לעשות בכל חלק ואיך הם מתחברים. לא כל כך נכנסתי לתיאוריה ול"למה עושים מה שעושים". לא שזה לא חשוב, אבל זה לא קריטי לתרגיל (למבחן לעומת זאת...).

אני ממליץ מאוד לפני שמתחילים לעבור את ה-tutorial הזה שמלמד פייתון ויותר חשוב – שימוש ב-numpy: http://cs231n.github.io/python-numpy-tutorial/ (אין צורך ב-SciPy)

בכל מקרה מקווה שתיהנו מהתרגיל (או לפחות לא תסבלו) והמון בהצלחה!

תיאור התרגיל

אתחיל בלתאר מה בערך צריך לעשות בתרגיל.

טעינת המידע

החלק הראשוני של התוכנית היא קריאת הקבצים והפיכתם למטריצות של numpy. למזלנו numpy יכול לעשות את זה בשבילנו עם np.loadtext(file_name). נשים לב שאחרי שהוצאנו את הקבצים מהrar מהפיאצה נקבל 3 קבצים: train_x (מכיל את מטריצת הדוגמאות עבור סט האימון וה-validation – כל שורה היא דוגמא מגודל 784), train_y (מכיל את התגים של הדוגמאות מהקובץ הנ"ל – כל שורה היא התיוג של השורה המתאימה מtrain_x) ו-test_x (מכיל את הדוגמאות שעליהן נבדוק את המודל שלנו בסוף). נרצה לטעון את שלושתם למטריצות באמצעות loadtext(). (האמת שיש עוד קובץ בשם test.pred, אבל אנחנו נתעלם ממנו)

לאחר שנטען את הדאטה, נרצה ליצור מ-train_x ו-train_y שני סטי אימון: train ו-validation. בשביל זה אנחנו נעשה שני דברים: קודם נעשה shuffle להם[1] (או עם numpy או עם random של פייתון – העיקר לוודא שכשאנחנו משנים את הסדר השורות ב-X לשנות בהתאמה את סדר השורות ב-Y), ואז נחלק אותו לשניים: train עם 80% מהמידע ו-dev עם ה-20% הנותרים.

פסאודו-קוד בסיסי:

```
train x = np.loadtxt("train x")
train y = np.loadtxt("train y")
test x = np.loadtxt("test x")
shuffle(train_x, train_y)
dev_size = train_size * 0.2
dev_x, dev_y = train_x[-dev_size:, :], train_y[-dev_size:]
train_x, train_y = train_x[:-dev_size, :], train_y[:-dev_size]
```

מה יש במידע?

ה-dataset נקרא Fashion-MNIST שמכיל תמונות שחור-אפור-לבן בגודל 28*28 פיקסלים. כל פיקסל הוא float בטווח 0-255 שמתאר כמה כהה הפיקסל. (זה ה-x-ים)
יש 10 מחלקות שהן סוגי לבוש וכד' שונים (יש רשימה בתרגיל) ולכל אחת יש אינדקס ייחודי בטווח בין 0-9 – זה בעצם מתאר לנו את התא בווקטור התוצאות של הרשת שאומר את ההסתברות שהתמונה היא מסוג המחלקה הזו. (זה ה-y-ים)

האם אכפת לכם? לא בטוח. בסופו של דבר כל x הוא וקטור בגודל 784 (28*28) (ראו הערה בהמשך) וכל y הוא מספר בטווח 0-9. אתם תתאמנו עליהם בלי בהכרח לדעת איך נראית כל תמונה.

מה כל dataset אומר?

קיבלנו 3, שלכל אחד יש תפקיד שונה מבחינתנו בתהליך האימון:

- Train set – הסט הגדול ביותר, זה הסט היחידי שהמודל שלנו יתאמן עליו (כלומר ילמד אותו). במהלך האימון נעבור על כל הtrain set כמה פעמים ובכל שלב נכניס אותו דרך הרשת, נבדוק כמה טעינו (loss), נחשב את הגרדיאנטים של כל פרמטר לפי ה-loss ונעדכן את המשקולות לפי הכלל של SGD (פירוט על כל זה – בהמשך)
- Dev (validation) set – זה הסט המשלים של הtrain מקבצי ה_train. עליו נרצה לבדוק loss ממוצע ודיוק עליו לכל אפוק. הוא בעצם דוגמאות שהמודל אף פעם לא "ראה" (לא למד) ואנחנו נבחן כמה טוב הוא בלסווג מידע כזה אתו. יעזור לנו לבחור היפר-פרמטרים.

---

[1] למה לעשות shuffle? הרנדומליות תדאג שלא כל ה-validation set יהיה מאותו סוג (נגיד אם train_x מסודר לפי סוג class) ובעצם תעזור לנו לכוונן את המודל למקרה כללי (דוגמאות שהמודל לא ראה מעולם יסווגו טוב)

- Test set – נועד לתת לצוות הקורס לבחון אותנו (בתרגיל, בעולם האמתי נותן לעשות מעין מבחן סופי). אין לנו את התיוג שלו (מתעלמים מtest.pred) ואנחנו בעצם ניצור test.pred משלנו שמכיל את התיוגים שהמודל שלנו נתן לאחר שאימנו אותו על ה-train set.
שימו לב! הציון יינתן על פי הדיוק של הקובץ הזה, אז אנחנו נרצה לתת את הזה שיצא מהמודל הטוב ביותר שלנו.

יצירת פרמטרים ובחירת היפר-פרמטרים

פרמטרים הם אותם מטריצות/וקטורים משקולות שאנחנו נעדכן במהלך אימון המודל כדי לתת סיווג מדויק יותר – הם החלקים שהמודל ישנה לבד במהלך האימון.
היפר-פרמטרים הם אותם "פרמטרים" לתוכנית שאנחנו מחליטים עליהם, ויש להם השפעה עצומה על הצלחת המודל. המודל לא יכול ללמוד אותם כי הם חלק ממנו, כביכול.

פרמטרים

יש לנו 4 פרמטרים בתוכנית שלנו. בהינתן H גודל השכבה ה-hidden שלנו, הפרמטרים יהיו:

$$W_1 \in \mathbb{R}^{H\times 784}, b_1 \in \mathbb{R}^{H}, W_2 \in \mathbb{R}^{10\times H}, b_2 \in \mathbb{R}^{10}$$

כאשר 784 זה גודל הקלט ו-10 זה מספר המחלקות.

בתחילת התוכנית, נרצה ליצור את הפרמטרים האלו ולאתחל אותם בערכים. בקובץ שהמתרגלים העלו הם השתמשו ב-np.random.rand(size1, size2) שמחזיר מטריצה / וקטור מהגודל המתאים בטווח $[0,1)$. (לי זה עשה בעיות והייתי צריך לאתחל בצורה אחרת – הסתכלו בהערה בהמשך)

שימו לב! ווקטור בnumpy אפשר לייצג בשתי צורות, לפי ה-shape שלו: אם a.shape מחזיר (1, 784) או (784, 1) אז זה בעצם מטריצה עם שורות / עמודות מגודל 1. לעומת זאת אם זה מחזיר (,784) אז זה נקרא 1darray (שהוא עדיין ndarray שזה מטריצה numpy-ית) ואפשר להתייחס אליו כשורה או עמודה כנדרש לרוב הדברים. אם לא תספקו לפונקציות את size2 אז הם יצאו בצורה השנייה.

היפר-פרמטרים

אני חושב על 5 היפר-פרמטרים שונים שאפשר לשחק איתם:

- גודל השכבה הנסתרת – ה-H שתיארנו קודם הוא גודל שאנחנו קובעים ויכולים לשחק אתו. הכלל היחיד בנוגע אליו הוא שהוא יהיה מספר כלשהו בין 784 ל-10 (המטרה היא לאט לאט לצמצם את גודל המידע – המודל ילמד איך לעשות את זה בצורה הטובה ביותר (בתקווה))
- קצב הלמידה – להזכירכם, כלל העדכון של משקל כלשהו לפי SGD הוא $W_{i_{new}} = \mathrm{W}_{\mathrm{i}_{old}} - lr * \frac{\partial Loss(\Theta)}{\partial W_i}$ כאשר $Loss(\Theta)$ זה הלוס עבור הדוגמא הנוכחית בהינתן הפרמטרים של המודל (תטא), כלומר $\frac{\partial Loss(\Theta)}{\partial W_i}$ זה בעצם הגרדיאנט של הלוס לפי W_i. lr (או learning rate) הוא כמה "משקל" אנחנו נותנים לגרדיאנט בתיקון הפרמטר. הכלל העיקרי הוא שבד"כ הוא חיובי קטן מ-1. ערכים מומלצים לבדוק הם 0.1, 0.01 ו-0.001 ומשם שחקו כרצונכם.
- מספר האפוקים (epochs) – אפוק הוא מעבר על כל הדוגמאות בסט האימון, ואימון עליהן (עדכון משקולות לפי גרדיאנטים של הלוס על הדוגמא). כמה פעמים נעבור על האימון קריטי להצלחת המודל: ישנן דוגמאות שמעבר אחד עליהן לא יצליח ללמד את המודל לסווג אותן נכונה, אבל אם נעשה יותר מדי מעברים על הדאטה נוכל להגיע למצב של overfitting בו המודל למד את סט האימון פרפקט אבל לא ממש עושה חיל בלסווג את סט ה-validation (אפשר לחשוב על זה כאילו המודל למד "לשנן" דוגמאות במקום ללמוד את ההיגיון מאחוריהן)
- פונקציית אקטיבציה – כמו שלמדנו בשיעור, בכדי להכניס אלמנט של אי-ליניאריות למודל אנחנו נשתמש בפונקציות לא ליניאריות בתור מעברים בין שכבה לשכבה. ישנן 3 עיקריות שאני מכיר: sigmoid (logistic) שמחזירה תוצאה בין 0 ל-1, tanh שמחזירה תוצאה בין 1-

ל-1 ו-ReLU שמחזירה תוצאות אי-שליליות. כל אחת עובדת קצת שונה וטובה לדברים קצת שונים, אבל בשבילנו כולם עובדות די אותו דבר. תנסו כמה שונות ותחליטו מה הכי טוב לכם. **שימו לב!** יחד עם פונקציית האקטיבציה חשוב להביא את הנגזרת המתאימה לה – זה חלק חשוב מתהליך ה-back propagation שיחשב לנו את הגרדיאנטים.

- גודל באטץ' (batch) – עקרונית אפשר לעשות כמה דוגמאות ביחד, לחשב loss משותף ולעדכן פעם ב[גודל באטץ'] דוגמאות (כביכול מקטין את זמן האימון של הרשת – במחיר הדיוק). אני אישית לא כל כך נגעתי בזה וכשנגעתי זה לא הלך לי, אז אני לא אכנס לזה. מוזמנים לנסות 😉

פסאודו-קוד

אתחול פרמטרים בקטנה...

```
input_size, hidden_size, output_size = 28*28, ..., 10
W1, b1 = initialize_weights(hidden_size, input_size), initialize_weights(hidden_size)
W2, b2 = initialize_weights(output_size, hidden_size), initialize_weights(output_size)
params = [W1,b1,W2,b2]
```

אימון הרשת

נתחיל מסוג של פסאודו-קוד שמתאר את פונקציית האימון ונסביר מה כל חלק אומר.

```
def train(params, epochs, active_func, lr, train_x, train_y, dev_x, dev_y):
    print "epoch","avg. train loss","avg. dev loss","accuracy on dev"
    for i in xrange(epochs):  # for each epoch:
        sum_loss = 0.0
        shuffle(train_x, train_y)  # shuffle train examples - helps the
model to learn (won't just remember order)
        for x, y in zip(train_x, train_y):
            out = forward(params, active_func, x)  # get probabilities
vector as result, where index y is the probability that x is classified as
tag y
            loss = Loss(out, y)  # compute loss to see train loss (for
hyper parameters tuning)
            sum_loss += loss
            gradients = backprop(params, x, y, active_func)  # returns the
gradients for each parameter
            params = update_weights_sgd(params, gradients, lr)  # updates
the weights
        dev_loss, acc = predict_on_dev(params, active_func, dev_x, dev_y)
# after each epoch, check accuracy and loss on dev set for hyper parameter
tuning
        print i, sum_loss / train_x.shape[0], dev_loss, "{}%".format(acc *
100)
```

עבור כל אפוק נעבור על כל הדאטה בסט האימון. עבור כל דוגמא נחשב את תוצאת המודל (forward pass), נחשב את הloss ואז את הגרדיאנטים של כל משקל (back propagation). לאחר מכן נעדכן את המשקלים לפי כלל העדכון של SGD ($W_{i_{new}} = \mathrm{W}_{i_{old}} - lr * \frac{\partial Loss(\Theta)}{\partial W_i}$).

כמו כן נחשב עבור כל אפוק את הלוס והדיוק על ה-dev_set כדי שנדע מה מצב המודל וכמה ההיפר-פרמטרים שלנו עובדים טוב. נעבור עכשיו על כל חלק ונסביר מה אמור לקרות בו:

Forward Pass – חישוב תוצאת הרשת

(אני אשתדל להיצמד לשמות שנתנו בתרגול אבל לא מבטיח כלום)

להזכירכם, יש לנו את הפרמטרים הבאים: $W_1 \in \mathbb{R}^{H\times 784}, b_1 \in \mathbb{R}^H, W_2 \in \mathbb{R}^{10\times H}, b_2 \in \mathbb{R}^{10}$ ואת פונקציית האקטיבציה הלא ליניארית $g \in \{sigmoid, tanh, ReLU\}$.
המעבר של קלט ברשת הוא בעצם המשוואות הבאות שיספקו לנו את וקטור ההסתברויות $\hat{y}$ (בהינתן x של דוגמא מסוימת):

$$\hat{y} = f(x; W_1, b_1, W_2, b_2) = softmax(z_2)\ (\in \mathbb{R}^{10})$$
$$z_2 = W_2 h + b_2\ (\in \mathbb{R}^{10})$$
$$h = g(z_1)\ (\in \mathbb{R}^H)$$
$$z_1 = W_1 x + b_1\ (\in \mathbb{R}^H)$$

לא כזה נורא, נכון? אפשר להסתכל על forward כמעבר על המשוואות האלו מלמעלה למטה, כאשר כל פעם מציבים את מה שיש ומחשבים.

מה זה אומר להפעיל את g על ווקטור? g היא פונקציה שמקבלת x סקלר ומחזירה סקלר אחר, אבל אנחנו נרצה לקבל וקטור חזרה. פשוט מאוד – הפעלה של g על וקטור תחזיר וקטור שבו על כל איבר הופעל g. למזלנו, בnumpy אפשר לכתוב פונקציות שיעבדו על ווקטורים וגם על ערכים סקלרים – שווה לבדוק את העניין.

שימו לב – ה-back propagation ישתמש בכמה מהערכים שחישבנו ב-forward pass, ולכן אולי אין טעם לחשב במהלך האימון את ה-forward אם נחשב אותו בכל מקרה ב-backprop – או לחילופין לתת ל-backprop גם את ערכי הביניים הרלוונטיים כדי שלא יחשב שוב. (הערכים הרלוונטיים הם $\hat{y}$, h ו-z1).

חישוב Loss

ה-loss אומר לנו בעצם כמה מידע "איבדנו" או יותר נכון כמה "רחוקים" אנחנו מלצדוק. המטרה שלנו בסופו של דבר היא למזער את ה-loss (ולהעלות את הדיוק, שהם בערך ביחס הפוך אבל לא תמיד).
אם נבדוק מה הloss הממוצע ב-train וב-dev כל אפוק נוכל לדעת אם המודל לומד מאפוק לאפוק (וכמה), אם יש overfitting (לוס train יורד אבל dev עולה) ועוד כמה מקרים מעניינים – כלומר סה"כ שווה להשקיע בו.

חישוב הloss שלנו הוא ה-Negative Log Loss שבהינתן ווקטור הסתברויות $\hat{y}$ ותיוג y, בצורת וקטור של אפסים שבמקום הנכון יש 1 (למשל עבור תיוג 3 נקבל $(y = (0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0)$) תחזיר לנו את הלוס לפי הנוסחה הזו: $loss(\hat{y}, y) = -\sum_i y_{[i]} \log(\hat{y}_{[i]})$.

Back Propagation – חישוב הגרדיאנטים

זה החלק הכי קשוח של התרגיל. אז אם שרדתם עד עכשיו – נשימה עמוקה ונעבור את זה ביחד.

יש לנו 4 פרמטרים, אז אנחנו צריכים 4 גרדיאנטים. אם נסמן את הloss כ-L, נרצה לחשב את

$$\frac{\partial \mathrm{L}}{\partial W_1}, \frac{\partial \mathrm{L}}{\partial b_1}, \frac{\partial \mathrm{L}}{\partial W_2}, \frac{\partial \mathrm{L}}{\partial b_2}$$

אבל מה זה בעצם אומר גרדיאנט של מטריצה או וקטור? פשוט מאוד, זה מטריצה מגודל זהה שבה כל תא הוא הגרדיאנט של הלוס לפי התא הזה. לדוגמא:

$$gW_1 := \frac{\partial L}{\partial W_1} \in \mathbb{R}^{H\times 784}, gW_{1_{[i,j]}} := \frac{\partial L}{\partial W_{1_{[i,j]}}}$$

בפועל זה די אומר לגזור כרגיל, ולפעמים צריך לפצל את המטריצות לשורות או ווקטורים לתאים.

עכשיו, זוכרים את המשוואות היפות מ-forward? ושאמרנו שחישוב forward זה כמו לעבור על המשוואות מלמטה למעלה? בשביל לחשב את הגרדיאנטים של כל משקל, אנחנו נעבור על המשוואות מלמעלה למטה ונחשב את הגרדיאנטים של הלוס על פי כל ערך ביניים ונשתמש בכלל השרשרת בכדי לחשב אותם בקלות.

$$loss(\hat{y}, y) = -\sum_i y_{[i]} \log(\hat{y}_{[i]})$$
$$\hat{y} = f(x; W_1, b_1, W_2, b_2) = softmax(z_2)\ (\in \mathbb{R}^{10})$$
$$z_2 = W_2 h + b_2\ (\in \mathbb{R}^{10})$$
$$h = g(z_1)\ (\in \mathbb{R}^{H})$$
$$z_1 = W_1 x + b_1\ (\in \mathbb{R}^{H})$$

(אני לא הולך לחשב את הגרדיאנטים פה – עזרתי לכמה אנשים לחשב אז זה מסתובב, ויש באינטרנט אם אתם לא בטוחים בחישוב. בטוח שתצליחו 😊)

נתחיל מ-W2 ומ-b2 – הם יחסית קלים לגזירה, כי הם כמו לגזור שכבה אחת עם סופטמקס (כמו בתרגיל הקודם, אבל שהפעם h הוא הקלט לשכבה). בנוסף בשביל השכבה הבאה, נרצה לחשב את הנגזרת של הלוס לפי הקלט לשכבה הזו (h):

$$\frac{\partial \mathrm{L}}{\partial W_2} = \frac{\partial \mathrm{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial W_2}, \qquad \frac{\partial \mathrm{L}}{\partial b_2} = \frac{\partial \mathrm{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial b_2}, \qquad \frac{\partial \mathrm{L}}{\partial h} = \frac{\partial \mathrm{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial h}$$

ואלו הגרדיאנטים של כל השכבה השנייה. נשמור הצד את הגרדיאנטים של W2 ו-b2, ונמשיך לחשב את של W1 ו-b1:

$$\frac{\partial \mathrm{L}}{\partial W_1} = \frac{\partial \mathrm{L}}{\partial h} \frac{\partial h}{\partial z_1} \frac{\partial z_1}{\partial W_1}, \qquad \frac{\partial \mathrm{L}}{\partial b_1} = \frac{\partial \mathrm{L}}{\partial h} \frac{\partial h}{\partial z_1} \frac{\partial z_1}{\partial b_1}$$

נשים לב ש- $\frac{\partial h}{\partial z_1} = g'(z_1)$ כלומר לנגזרת של הפונקציה g שבחרנו (משתנה לפי בחירת g). נשים לב שהפונקציה `backprop(params, x, y, active_func)` צריכה לדעת מה הנגזרת של active_func בכדי לחשב את הגרדיאנטים.

(שוב, לחשב את הנוסחאות לנגזרות זה אחד החלקים הקשים בתרגיל. אל תתייאשו, וקחו הרבה השראה מהמצגות – הן עושות אותו תהליך פשוט על רשת טיפה שונה)

למה בעצם מחשבים גרדיאנטים?

אז למי שלא הבין למה הוא גוזר פונקציות אחרי אינפי 2 (ברוך שפטרנו) - זה בשביל לצמצם את הLoss.

בהרצאות הראשונות הגדרנו את פונקציית הלוס שבהינתן תוצאת הרשת עבור דוגמה x כלשהי ואת התיוג האמתי שלה y ומחזירה כמה מידע "איבדנו" או כמה "רחוקים" אנחנו מלצדוק.

עכשיו המטרה בלמידה היא לצמצם את הלוס על כל דוגמא, ע"י שנמצא את הפרמטרים (W-ים ו-b-ים) שיתנו את הלוס הכי נמוך על כל הדוגמאות - בעצם יש לנו פונקציה (loss) שאנחנו רוצים למצוא את נקודת המינימום עבורה (כאשר ה"נקודה" היא ערכי ה-W-ים ו-b-ים).

עכשיו אם היית נותן לכם פונקציה לא ידועה ומבקש למצוא לה את המינימום, הייתם יכולים לנסות לחשב ערך בנקודה ואז לחשב את הנגזרת בנקודה - אם נקבל נגזרת חיובית נדע שאנחנו בעליה ונרצה ללכת בכיוון השני, ואם שלילית נדע שכדאי ללכת בכיוון בשביל למצוא מינימום.

זה פישוט עצום למה שקורה בלמידה - האלגו' SGD דואג לצמצם לאט לאט את המשקלים לכיוון המינימום, באמצעות הloss שיצא ("כמה טעינו") והגרדיאנטים של הloss לפי כל פרמטר ("באיזה כיוון צריך לתקן"). (מי שלמד בינה אולי זוכר שגם אז דיברנו על "כמה טעינו" ו"באיזה כיוון") עד שנתכנס לערכים טובים יחסית.

עדכון המשקלים

פשוט מאוד לעדכן את המשקלים לפי SGD: $W_{i_{new}} = \mathrm{W}_{\mathrm{i}_{old}} - lr * \frac{\partial Loss(\Theta)}{\partial W_i}$ (זהו נשבע פעם אחרונה שאתם רואים בקובץ את הנוסחה הזו)

חישוב דיוק ו-loss על ה-dev_set

סיימנו אפוק אחד של למידה על הדוגמאות! כל הכבוד לנו!
אבל לפני שמתחילים לפתוח שמפניות בואו נוודא שזה באמת עובד לנו – ובדיוק בשביל זה הכינינו מראש Validation Set.

הרעיון פשוט מאוד: נעבור על כל הדוגמאות, ועבור כל אחת נחשב את התוצאה מה-forward ואת ה-loss, נספור כמה פעמים צדקנו ונחזיר את ממוצע הלוס ואת הדיוק.
נשים לב שאין צורך ב-backprop ולא בעדכון משקולות – למעשה זה אפילו **אסור!** המטרה שלנו היא בעצם לא ללמוד את הדוגמאות ב-dev, אלא לבדוק מה מצבנו. זה הכל.

פסאודו-קוד קטנטנן:

```
def predict_on_dev(params, active_func, dev_x, dev_y):
    sum_loss = good = 0.0  # good counts how many times we were correct
    for x, y in zip(dev_x, dev_y):
        out = forward(params, active_func, x)  # get probabilities
vector as result, where index y is the probability that x is classified
as tag y
        loss = Loss(out, y)  # compute loss to see train loss (for
hyper parameters tuning)
        sum_loss += loss
        if out.argmax() == y:  # model was correct
            good += 1
    acc = good / dev_x.shape[0]  # how many times we were correct / #
of examples
    avg_loss = sum_loss / dev_x.shape[0]  # avg. loss
    return avg_loss, acc
```

סיימנו?

ובכן, כמעט. כלומר, סיימנו לכתוב את רוב הקוד, אבל עדיין יש דברים לעשות – למצוא את ההיפר פרמטרים שיהפכו את הרשת שלנו להיות הרשת הכי טובה שיש.

איך עושים את זה? פשוט מאוד, מנסים שינויים ובודקים מה עובד יותר טוב ומה פחות. הדיוק הוא הפרמטר החשוב כי בסופו של דבר מעניין אותנו להיות כמה שיותר צודקים – אבל הלוס חשוב כמעט באותה מידה כי כשהלוס יורד זה אומר שהרשת לומדת.

הרעיון הוא שתמשיכו לנסות ולהריץ עד שמגיעים לתוצאה שאוהבים – כי השלב הבא הוא המבחן האמתי.

Predicting on the Test Set

כעת הגיע הרגע הגורלי, בו אנחנו נוציא מהרשת פלטים שעל פיהם יקבע הציון שלנו.
זה מאוד פשוט בפועל – נעבור על כל דוגמא ב-test_x, נעשה עבורה forward וניקח את ה-argmax() (אגב למה לוקחים argmax? זה בעצם הקלאס שהמודל חושבת שיש את ההסתברות הגבוהה ביותר שהוא הנכון) ונכתוב אותו לשורה המתאימה בקובץ.

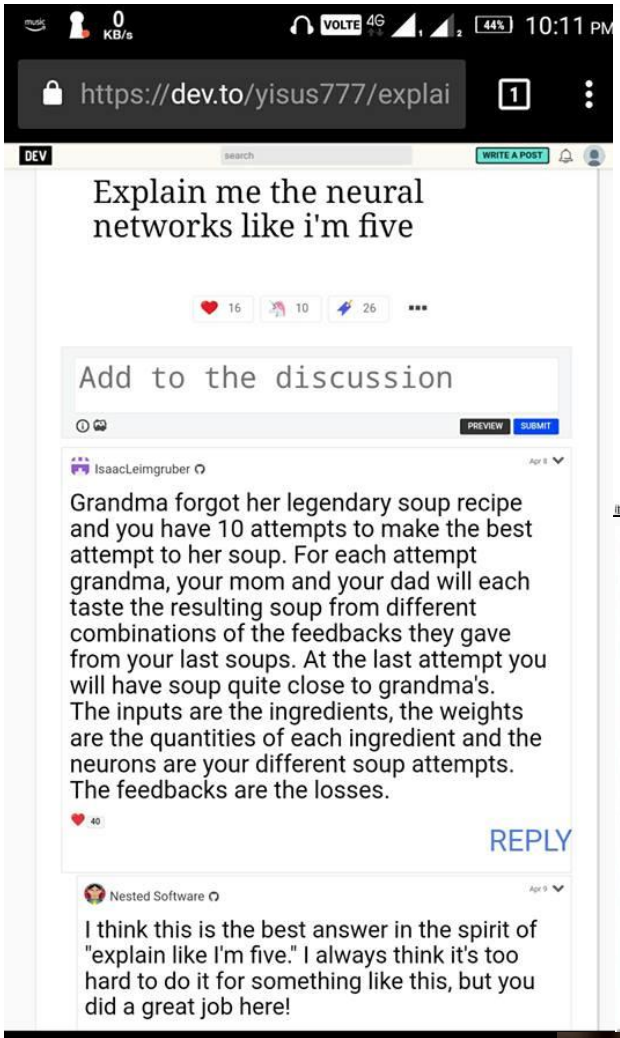
יאללה פסאודו-קוד אחרון לכיף:

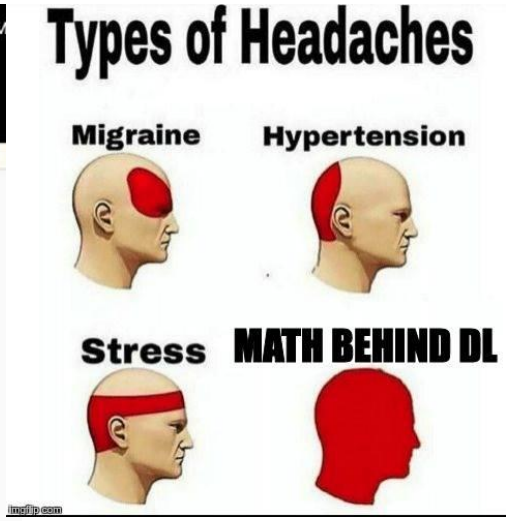
```
f = open("test.pred", "w")
for x in test_x:
    out = forward(params, active_func, x)
    f.write(str(out.argmax()))
f.close()
```

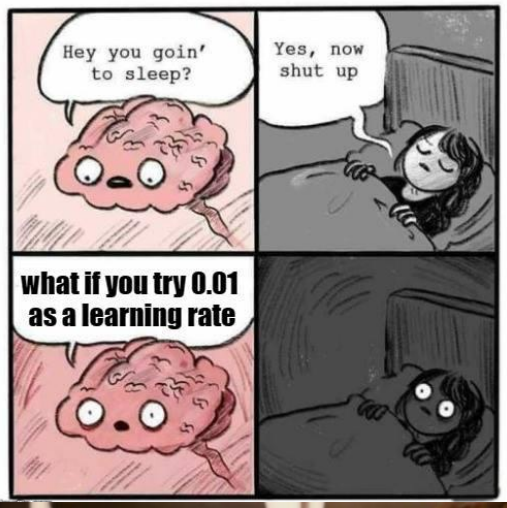
בעיות, פתרונות וטיפים

אני אנסה לתאר פה כל מיני בעיות שאני נתקלתי בהן ופתרונות נפוצים שאולי הם לא טריוויאליים. אם למישהו קרה משהו מעניין הוא מוזמן להגיד לי ואני אוסיף.

- **"פיצוץ" של תוצאות השכבה הראשונה** – לי קרה הרבה בהתחלה ש-z1 היו מספרים ממש גדולים, ואז sigmoid או tanh החזירו תמיד 1 על כולם, וזה דפק את הפרדיקציה ואת הגרדיאנטים.
  הפתרון שמצאתי הוא דווקא שלושה:
  1) נרמול הקלטים: (בהצעתו של תמיר המלך) נחלק את כל הקלטים ב-255.0, ככה שבמקום קלטים בטווח 0-255 נעבור לקלטים בטווח [0,1].
  2) אתחול המשקלים בצורה יוניפורמיות בטווח בין 0.08- ל-0.08 ($W \sim U(-0.08, 0.08)$)
  3) אתחול על פי השיטה של Glorot: בקצרה, בהינתן שאנחנו רוצה לאתחל משקל בגדלים n x m, נגדיר את אפסילון כ- $\varepsilon = \sqrt{\frac{6}{n+m}}$ ואז המשקל W ידגם יוניפורמית: $W \sim U(-\varepsilon, \varepsilon)$
  (יש פונקציית numpy בשם random.uniform)
- **Loss אינסופי** – זה קורה בד"כ בגלל שהמודל "ממש בטוח" בעצמו אז התוצאה מהסופטמקס היא ווקטור של אפסים עם אחד במקום שהוא חושב. אז אם הוא טועה, הלוס שהוא מקבל על להגיד שיש 0% סיכוי שהתיוג הוא התשובה הנכונה הוא גדול מאוד – ולמעשה אינסופי (כי log(0)...)
  פתרון: כרגע אין. אשמח לשמוע הצעות. בעיקר כרגע הפסקתי להשתמש ב-ReLU כי הוא עשה לי את הבעיות האלו.
- **רשת איטית** – הרשת לא פשוטה למימוש לכן סביר להניח שלא נממש אותה בצורה האופטימלית וחבל. יש הרבה פונקציות של numpy שיכולות לעזור לנו בביצועים (למשל outer, dot ושימוש בפעולות שעובדות על כל המטריצה).
  באופן כללי קחו לכם אתגר: בלי להשתמש בלולאות בכלל, חוץ מה-4 שצריך בשביל לעבור על הדאטה (מעבר על כל אפוק, מעבר על כל דוגמא בסט האימון, מעבר על כל סט ה-validation ומעבר על כל סט ה-test). ממליץ להיעזר ב tutorial שיש בהקדמה.

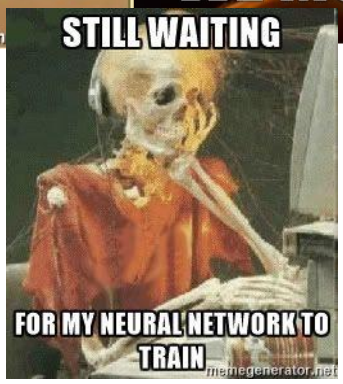

https://dev.to/yisus777/explai
DEV
WRITE A POST
Explain me the neural networks like i'm five
16
10
26
Add to the discussion
PREVIEW
SUBMIT
IsaacLeimgruber
Grandma forgot her legendary soup recipe and you have 10 attempts to make the best attempt to her soup. For each attempt grandma, your mom and your dad will each taste the resulting soup from different combinations of the feedbacks they gave from your last soups. At the last attempt you will have soup quite close to grandma's.
The inputs are the ingredients, the weights are the quantities of each ingredient and the neurons are your different soup attempts.
The feedbacks are the losses.
40
REPLY
Nested Software
I think this is the best answer in the spirit of "explain like I'm five." I always think it's too hard to do it for something like this, but you did a great job here!


Types of Headaches
Migraine
Hypertension
Stress
MATH BEHIND DL
imgflip.com


Hey you goin' to sleep?
Yes, now shut up
what if you try 0.01 as a learning rate

IT IS FRIDAY, ALL MY FRIENDS ARE AT PARTY
BUT, I'M HERE LEARNING BACK PROPAGATION

PERCEPTRONS ARE TOO LINEAR
WE MUST GO DEEPER


STILL WAITING
FOR MY NEURAL NETWORK TO TRAIN
memegenerator.net