שכבות Dropout

בעוד שרגולריזציה דרך שינוי פונקציית המחיר כפי שהצגנו בתחילת יחידת לימוד זו היא שיטה הנמצאת בשימוש במגוון מודלים של למידת מכונה, קיימת שיטת רגולריזציה אפקטיבית במיוחד הלוקחת בחשבון את מבנה המודלים של למידה עמוקה ולכן היא שימושית רק עבורם, בה נעסוק בפרק הנוכחי. בשיטה זו, הנקראת Dropout, אנו מנסים למנוע מהרשת לשנן את סט האימון על ידי הוספת רעש לתהליך האימון: בכל איטרציה של אלגוריתם האופטימיזציה רק חלק אקראי מהנוירונים ישתתף בתהליך הלימוד. מימוש רעיון זה מבוצע בשני שלבים.

ראשית, נגדיר שכבת Dropout המקבלת קלט X , ומאפסת כל אחד מרכיביו באקראי, בהסתברות קבועה מראש ובאופן בלתי תלוי.

```
class Dropout(nn.Module):
  def __init__(self,drop_rate=0.5):
    super().__init__()
    assert(0 < drop_rate < 1)
    self.drop_rate = drop_rate
  def forward(self,X):
    mask = torch.rand(X.shape) > self.drop_rate
    return X*mask
```

בכל מעבר קדימה ברשת מוגרל מחדש טנזור של משתנים מקריים אחידים סטנדרטיים בעל אותו מימד כמו X , בו אנו משתמשים לחשב את המשתנה הבוליאני `mask`. זכרו שפונקציית הצפיפות של מ"מ אחיד $u \sim U[0,1]$ היא

$$f(x)=\begin{cases}1 & x\in[0,1]\\ 0 & x\notin[0,1]\end{cases}$$

ולכן מתקיים

$$P(u>p)=\int_p^1 1dx=1-p.$$

מכאן נסיק שבמכפלה `X*mask` כל קואורדינטה מאופסת בהסתברות `drop_rate`, וזאת באופן בלתי תלוי בשאר הערכים. שאר הקואורדינטות מועברות הלאה ללא שינוי.

השלב השני במימוש השיטה הוא שילוב שכבות Dropout בהגדרת הרשת, לאחר כל שכבת נוירונים. למשל, נוסיף שכבות לרשת הבאה,

```
model = nn.Sequential(nn.Flatten(),
          nn.Linear(784,100),nn.ReLU(),
          nn.Linear(100,10),nn.ReLU(),
          nn.Linear(10,10),
          nn.LogSoftmax(dim=1))
```

ונקבל את הרשת

```
model_dropout = nn.Sequential(nn.Flatten(),
          nn.Linear(784,100),nn.ReLU(),
          Dropout(),
          nn.Linear(100,10),nn.ReLU(),
          Dropout(),
          nn.Linear(10,10),
          nn.LogSoftmax(dim=1))
```

שימו לב שלא הוספנו שכבת Dropout מיד לאחר הקלט, שכן אנו מעוניינים לתת לרשת את מלוא המידע הקיים בנתונים, וכן לא לאחר השכבה האחרונה, שכן אם ברצוננו להשתמש ברשת לחיזוי שייכות לאחת מ10 מחלקות, איפוס נוירונים בשכבה זו יפגע באופן קריטי ביכולת החיזוי של הרשת.

לאחר הגדרת הרשת מחדש, יש לבצע את תהליך האימון כבעבר. קיומן של שכבות ה-Dropout יביא לרגולריזציה של המודל ללא שינוי נוסף, אך לאחר האימון ניתקל בבעיה נוספת: בדומה לשימוש בשכבות Batch Normalization, הרנדומליות המובנית בתוך שכבת Dropout אינה רצויה בעת החיזוי. במימוש הנוכחי נקבל תוצאות שונות בהתאם להגרלת המ"מ בשכבות אלו. לכן נשנה את הגדרת המעבר קדימה כך שבמצב חיזוי לא יתבצע כל חישוב:

```
  def forward(self,X):
    if self.training:
      mask = torch.rand(X.shape) > self.drop_rate
      return X*mask
    else:
      return X
```

זכרו שניתן להעביר את הרשת ממצב אימון למצב חיזוי וחזרה בעזרת המתודות `model.eval()` ו-`model.train()`.

כפי שהמעבר קדימה מוגדר כעת, מחוץ למצב אימון שכבות ה-Dropout אינן פעילות. שינוי זה אינו מספק, ואף יפגום ביכולת החיזוי של הרשת בהשוואה למצב הקודם: לאורך כל תהליך האימון שכבות ה-Dropout החלישו את הפלט מהשכבה הקודמת, ובעת החיזוי הפלט עובר הלאה באופן מלא, אך לא באופן המתאים לעוצמה שלפיה פרמטרי הרשת אומנו. שימו לב שאם כל איבר ב- X נפל בהסתברות p אז השכבה הבאה אומנה לקבל אות בעוצמה ממוצעת $(1-p)X$. לכן, התיקון האחרון שנבצע לשכבת Dropout, הוא להפחית את עוצמת האות העובר הלאה בעת החיזוי, כפי שניתן לראות בשורה האחרונה בקטע הקוד הבא.

```
class DropoutFinal(nn.Module):
  def __init__(self,drop_rate=0.5):
    super().__init__()
    assert(0 < drop_rate < 1)
    self.drop_rate = drop_rate
  def forward(self,X):
    if self.training:
      mask = torch.rand(X.shape) > self.drop_rate
      return X*mask
    else:
      return (1-self.drop_rate)*X
```

השימוש ב-Dropout הפך לנפוץ מאוד מאז הצגתו, והערך שלו רב. קיימים מספר הסברים אפשריים להצלחת השיטה, שניים פופולריים הם:

1. שיטה זו דומה לשיטות ensemble קלאסיות של למידת מכונה: בשיטות אלו מאמנים מספר מודלים (במקביל או זה אחר זה) על אוסף הנתונים, ולבסוף החיזוי של ה-ensemble מתקבל בהחלטה משותפת של כל המודלים. כמו ensemble, אשר מטבעו פחות רגיש להתאמת יתר (שכן היא צריכה לבוא לידי ביטוי בהתאמת יתר זהה לכל תתי המודלים), כך ניתן לחשוב על תהליך האימון של רשת עם Dropout כאימון של מספר רב של מודלים שונים: בכל איטרציה מוגרלת קישוריות שונה בין נוירוני הרשת, והרשת הייחודית לאיטרציה זו משאירה אותותיה במשקלי הרשת. לבסוף התחזית המתקבלת לאחר האימון היא מעין ממוצע של תחזיות הרשתות השונות.
2. שיטה זו מונעת יצירת תלות קיצונית בין נוירונים, שכן בכדי להצליח בתחזית בנוכחות Dropout, המודל צריך לפתח **יתירות** (Redundancy): האקטיבציה של נוירון מסויים חייבת להיות מושפעת ממספר רב של נוירונים אחרים, כי שכבת Dropout יכולה לאפס כל אחד בודד מהם. בהתחשב בכך, פחות סביר שלאחר האימון יתקבלו משקלים גדולים במיוחד, תוצאה הדומה לרגולריזציית L_2.

כמו רכיבים נוספים של רשתות נוירונים, גם ללא הסבר מדויק או הוכחות תיאורטיות לאפקטיביות השיטה לא נהסס להשתמש ב-Dropout , שכן הוכחות מציאותיות לביצועיה במשימות פרקטיות קיימות בשפע.

שאלות לתרגול

1. ב-PyTorch שכבות Dropout ממומשות באופן שונה במעט מהנ"ל: פלט המעבר קדימה בעת האימון הוא `X*mask/(1-drop_rate)` בעוד שבעת החיזוי השכבה אינה מבצעת כל שינוי בקלט. הסבירו למה שינוי זה גם הוא מהווה פתרון עבור הבעיה של עוצמת אות שונה בעת אימון ובעת חיזוי. איזה יתרון חישובי יש למימוש זה על פני המימוש הנ"ל, בהנחה שהרשת המאומנת תשמש לחיזוי אוסף גדול מאוד של נתונים?
2.

א. אמנו רשת עמוקה לחיזוי המחלקות של אוסף הנתונים Fashion-MNIST. בחרו רשת מספיק עמוקה כך שבבירור תהיה התאמת יתר.

ב. הוסיפו לרשת שכבות Dropout ואמנו אותה שוב. בחרו את הפרמטר `drop_rate` של כל שכבה כך שמצד אחד לא תהיה התאמת יתר, ומצד שני דיוק החיזוי לא ייפגע.

ג. אמנו את אותה הרשת עם שכבות Dropout אשר אינן מתקנות את ההבדל בעוצמת האות של הפלט במצב אימון ובמצב חיזוי.

ד. השוו את ביצועי שלושת הגרסאות של הרשת על גבי סט הבדיקה.