

סיכון למידה יישומית

Orel Zamler

Cross validation:

המטרה - גריצה לחפש היפר פרמטרים רובוטיים.
נחלק את המידע, לא של הטסט, לכמה חלקים. בהתאם את ההיפר פרמטרים לפי כל החלקים חוץ מאחד ונבדוק עליו אם הפרמטרים מתאימים למידע שהוא לא ראה. כך נעשה עבור כולם, כל פעם ניקח חלק אחד אחר לשימוש הבדיקה. לבסוף, נבצע ממוצע על התוצאות.

מה ההבדל בין אימון לוידציה?

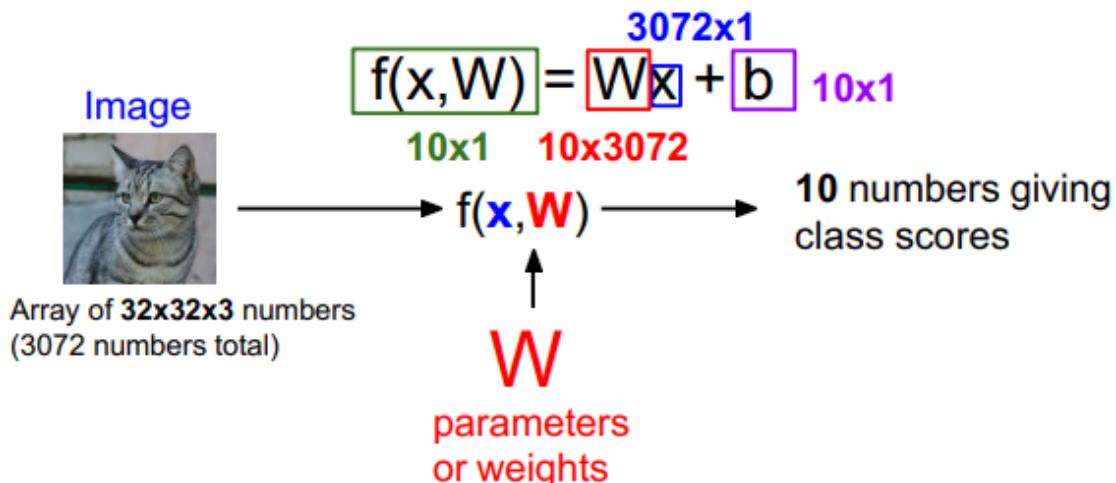
אימון – שימוש בליילום בשביל האימון כדי לגנות דיווק ולשפר וכו'...
וילדציה – שימוש בליילום לטובת בדיקת תאיות פרמטרים/פתרונות לאחר אימון ראשון.

למה המרחק האוקלידי ודמיון לא השוואת טובה בלמידה מכונה?

הראו דוגמה בשיפור של תמונה של בחורה שיצרו לה דפקטים והראו שהמרחב האוקלידי בתמונה נשאר כמעט זהה למורות העייניות. לעומת זאת, חישוב מרחק ע"י פונקציית אל שתיים לא מזהה בaczera טובה את התפיסה של המרחק בין תמונות.

Linear Classifier:

Parametric Approach: Linear Classifier



Last time: Batch Normalization

Input: $x : N \times D$

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Learnable params:

$$\gamma, \beta : D$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Intermediates: $\mu, \sigma : D$
 $\hat{x} : N \times D$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Output: $y : N \times D$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

שיטת שונועה לשפר את האימון והביצועים של רשת נירונית. הרעיון הכללי הוא לנורמל את הקלט של כל שכבה ברשת בזמן אימון. זה עוזר למנוע כל מיני בעיות ויכול להוביל להתקנות מהירה ויציבה יותר באימון. כך היא עבדת:

- א. עברו כל מיני באטץ' של המידע שעובר בראשת, נורמל את האקטיבציה של השכבה ביחס לממוצע והשונות של אותו מיני באטץ' ספציפי.
- ב. נורמליזציה – עברו על תכמה או אקטיבציה במני באטץ' הממוצע והשונות מחושבים, ומהידע מnoraml ע"י הפקחת הממוצע וחולקה בסטייה הסטנדרטית.
- ג. סקיליניג ושיפטיניג – מתבצע אחר הnormal ע"י פרמטרים נלמדים. פרמטרים אלו מעודכנים בזמן אימון בעזרתו הבאק פרופוגישן.

מתמטית- בהינתן מין באטץ' בגודל אמ' וקלט איקס:

- א. נחשב את הממוצע והשונות:

$$\text{mean} = (1/m) * \text{sum}(x)$$

$$\text{variance} = (1/m) * \text{sum}((x - \text{mean})^2)$$

- ב. נורמל את הקלט:

$$x_{\text{normalized}} = (x - \text{mean}) / \text{sqrt}(\text{variance} + \epsilon)$$

- ג. סקיליניג ושיפטיניג:

$$y = \gamma * x_{\text{normalized}} + \beta$$

יתרונות השיטה:

- א. מהירות - אפשר קצב למידה גבוה יותר ולכן התוכנות תהיה מהירה יותר.
- ב. סקיליניג - מפחית את השינוי של משתנים משותפים פנימיים ולכן הופך את הרשת לייצה יותר.
- ג. גנריות – משפר את הגנריות של הביצועים של המודל על מידע שעוזר לא נראה.

במהלך החיזוי על מידע שלא נראה עדין, הממוצע והשונות לא מחושבים מהמיini באטץ', אלא בAMDIM מהתוצאות הרצים של נתוני האימון. מה שמבטיח עקביות ויציבות במהלך החיזויים.

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x += learning_rate * dx
```

SGD+Momentum

$$\begin{aligned} v_{t+1} &= \rho v_t + \nabla f(x_t) \\ x_{t+1} &= x_t - \alpha v_{t+1} \end{aligned}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x += learning_rate * vx
```

:SGD

היא שיטת אופטימיזציה שמטרתה למנם את פונקציית הלס של המודל ע"י ביצוע התאמות לפרמטרים. הרעיון הכללי הוא, לנوع כיון הירידה בתוליה בפונקציית הלס ביחס ופרמטרים של המודל. במוקם לחשב את הנגזרת של כל הדאטה סט, נחשב את הנגזרת של כל דוגמה נפרדת או מייניבאטץ' באימון ונעדכן את הפרמטרים של המודל בכל איטרציה בהתבסס על החישוב שעשינו (יכול לזרז מאד את התכנסות במילוי אם הדאטה סט גדול). ואלו השלבים:

- נתחל את הפרמטרים של המודל בזורה רנדומלית.
- נעררב את הדאטה סט של האימון רנדומלית.
- עבור כל דוגמה או מיין באטץ' באימון:
 - נחשב את הנגזרת של פונקציית הלס ביחס לפרמטרים של המודל.
 - נעדכן את הפרמטרים בכיוון ההפוך של הנגזרת שמצאו בכי' למינם את הלס.
 - נחזיר על השלבים עד שנשים עם כל הבאטצים'.
- נחזיר על שלבים ב, ג' עבור כמות מסוימת של פעמים(איפוקים) או עד להתכנסות.

:חסרונות:

- יכול לסייע מעדכנים רועשים מה שהופך את האופטימיזציה לפחות יציבה.

:Momentum

היא טכניקה לשיפור התכנסות והיציבות של אלגוריתמי אופטימיזציה. היא מציגה ממוצע נוע של השיפועים המוחسبים במהלך האימון. במוקם לעדכן את הפרמטרים של המודל על סמך השיפוע של האיטרציה הנוכחית בלבד, מומנטום מתחשב בממוצע המשוקל של הגדי'אנטיים מאיטרציות קודמות. ממוצע נוע זה עוזר לחלקיק את העדכנים הרועשים ומאפשר לאלגוריתם להמשיך לנוע באותו כיוון גם כאשר שיפועים משתנים. ואלו השלבים:

- אתחל הפרמטרים של המודל והמהירות לאפס.
- לכל דוגמה או מיין באטץ':
 - נחשב את הנגזרת של הלס ביחס לפרמטרים של המודל.
 - עדכן המהירות ע"י שילובו עם השיפוע:
- מהירות = (מומנטום(היפר-פרמטר) * מהירות) + קצב למידה * שיפוע**
- עדכן הפרמטרים באמצעות המהירות:

פרמטרים = פרמטרים פחות מהירות.
- נחזיר על השלבים עד שהשתמשנו בכל הדוגמאות
- נחזיר על שלב ב כמה פעמים(איפוקים) או עד להתכנסות.

פרמטר המומנטום הוא היפר-פרמטר השולט בתורמת המהירות הקודמת לעדכן הנוכחי. ערכים אופייניים לתנופה הם בטוויה של 0.9 עד 0.99.

ניסי לב - ערכים גבוהים יותר של מומנטום הופכים את האלגוריתם לעמיד יותר בפני תנודות

בשיפוע אך עלולים לחרוג מהנקודה האופטימלית. מצד שני, ערכים נמכרים יותר של מומנטום עשויים להוביל לתוכנות איטית יותר.

על ידי שימוש במומנטום, אלגוריתם האופטימייזציה צובר אינרציה ויכול להאיץ לאורך היכוונים הרדודים של פונקציית האובדן תוך שיכוך תנודות לאורך כיוונים תלולים. זה עוזר לאלגוריתם להתכנס מהר יותר ולהימנע מלhalt תהליך האופטימייזציה.

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1}$$

$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

Stanford

Nesterov Momentum

מבצע שינוי קטן בעדכון המומנטום הסטנדרטי, שיעור לצפות את ההשפעה של מונח המומנטום על האיטרציה הבאה. זה מאפשר אלגוריתם "להסתכל קדימה" ולהתאים את כיוון העדכון שלו בצורה מדויקת יותר. ואלו השלבים:

א. אתחול פרמטרים ומהירות לאפס.

ב. לכל דוגמה או מני באטץ:

1. עדכון הפרמטרים של המודל בעזרת המהירות הנוכחיות:

parameters_temp = parameters - momentum * velocity

2.-Calculating the gradient of the loss with respect to the parameters of the model at step 1.

3. עדכון את המהירות:

velocity = (momentum * velocity) + learning_rate * gradient

4. עדכון הפרמטרים בעזרת המהירות:

parameters = parameters - velocity

5. נחזיר על השלבים של עד שיגמרו הדוגמאות.

ג. נחזיר על השלבים כמה פעמים (איפוקים) או עד להתכנסות.

בשלב ב1 במקום להשתמש בפרמטרים המקוריים לחישוב השיפוע הוא מעדכן תחיליה את הפרמטרים בכיוון המהירות הנוכחיות (מומנטום * מהירות) לפני חישוב השיפוע(ההסתכלות קדימה).

יתרונות:

יכול להאיץ התוכנות של תהליכי האופטימייזציה ואף להגיע לביצועים טובים יותר מהמומנטום הסטנדרטי.

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

הוא אלגוריתם אופטימיזציה של קצב למידה אדפטיבי המשמש לאימון מודלים של למידת מכונה. הרעיון המרכזי מאחורי Adagrad הוא לתת לכל פרמטר של המודל את קצב הלמידה שלו, המאפשר לו לטפל ביעילותBNODES דليلים ולהתמקד בתכונות או פרמטרים המתרחשים בתקירות נמוכה יותר הדורשים עדכנים קיטנים יותר. זה מועיל במיוחד עבור מודלים של למידה عمוקה שבהם פרמטרים שונים עשויים להיות בעלי קנה מידה וחישבות שונות בתכנית. אלו השלבים:

1. נתחל את הפרמטרים של המודול ופרמטר מעקב של הסכום המרובע של הנגזרות.
2. לכל דוגמה או מINI באטץ:
 - א. חישוב נגזרת של הלווי ביחס לפרמטרים של המודול בעזרת הדוגמה הנוכחית.
 - ב. עדכן פרמטר המעקב:
$$\text{sum_of_gradients} += \text{gradient}^2$$
 - ג. חישוב קצב הלמידה ההסתגלותי עבור כל פרמטר:
$$\text{adaptive_learning_rate} = \text{learning_rate} / \sqrt{\text{sum_of_gradients} + \text{epsilon}}$$
 - ד. עדכן הפרמטרים של המודול:
$$\text{parameters} = \text{parameters} - \text{adaptive_learning_rate} * \text{gradient}$$
- ה. נחזיר על שלבים של עד לסיום הדוגמאות.
3. נחזיר על שלב 2 כמהות מסוימת של פעמים (איפוקים) או עד להתקנסות.

המאפיין המרכזי של Adagrad הוא שהוא כובר את השיפורים בריבוע עבור כל פרמטר לארוך זמן, ולמעשה נותן קצב למידה גבוה יותר לפתרומים בעלי שיפורים קטנים יותר וקצב למידה נמוך יותר לפתרומים בעלי שיפורים גדולים יותר. פעולה זו מאפשרת לאדרגראד להתאים אוטומטית את קצב הלמידה ללא צורך בכונון ידני, שימושו במירבי בתרחישים עם נתונים דילים או מודלים מורכבים. **עם זאת**, הצברות של שיפורים בריבוע של Adagrad יכולה להוביל לשינוי למידה קטנים מאוד בשלבים המאוחרים של האימון, מה שעשו להאט את תהליכי התוכנסות. כדי לטפל בעיה זו, הוצגו אלגוריתמים אחרים של אופטימיזציה אדפטיבית כמו RMSprop ו-Adam.

RMSProp

RMSProp נועד לתת מענה לכמה מהמגבילות של Adagrad, במיוחד הבעיה של ירידה בשיעורי הלמידה לארוך זמן. בעוד Sh-Adagrad כוברת את ההדרגות בריבוע עבור כל פרמטר, מה שمبיא לשיעורי למידה קטנים מאוד באיטרציות מאוחרות יותר, RMSProp משתמש בממוצע נוע של מעבר ריבוע כדי לשנות בקצב הלמידה, מה שמחזק את סוגיות העדכנים האגרסיביים והפוחתים.

היתרון העיקרי של RMSProp על פני Adagrad הוא יכולת שלו להתאים את קצב הלמידה בצורה עיליה יותר, מניעת עדכנים אגרסיביים מדי והימנע מירידה בשיעורי הלמידה באיטרציות מאוחרות יותר. זה מאפשר ל-RMSProp להשיג התוכנסות מהירה יותר וביצועים טובים יותר במગן רחב של שימושות למידת מכונה. **עם זאת**, RMSProp עשוי עדין לסייע מכמה אגרים כמו הרגישות לבחירת

קצב הדעיכה, מה שהוא יכול לסייע של אלגוריתמים מתקדמים יותר לאופטימיזציה אדפטיבית כמו ADAM.

Adam (full form)

```

first_moment = 0
second_moment = 0
for t in range(num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))

```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\text{learning_rate} = 1e-3$ or $5e-4$ is a great starting point for many models!

:Adam

אלגוריתם אופטימיזציה קצב למידה אדפטיבית המשמש לאימון מודלים של למידת מכונה.

אדם משלב את הרעיון של אופטימיזציה של מומנטום -RMSProp כדי ליצור אלגוריתם קצב למידה אדפטיבי המשלב את היתרונות של שתי השיטות. הוא שומר על שני משתנים ממוצע נוע: הרוגן הראשון (הממוצע) של השיפועים והרגע השני (השונות הלא ממוקדת) של השיפועים. רגעים אלו משמשים לאחר מכן לעדכן הפרמטרים של המודל.

קצב הלמידה האדפטיביים של אדים מאפשרים לו להתמודד עם שיפורים שונים ביעילות ובייעילות. הוא משלב את היתרונות של קנה המידה האדפטיבי של RMSProp של קצב למידה עם היכולת של המומנטום להתמודד עם כיוונים שונים של שיפורים. כתוצאה לכך, אדים לעיתים קרובות מת恭ס מהר יותר ומשיג ביצועים טובים יותר מאלגוריתמי אופטימיזציה אחרים, מה שהופך אותו לבחירה פופולרית לאימון מודלים של למידה عمוקה ומשימות אחרות של למידת מכונה.

Lecture 7 | Training Neural Networks II

Vanilla Dropout: Not recommended implementation (see notes below) ***

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
 """ X contains the data """
 # forward pass for example 3-layer neural network
 H1 = np.maximum(0, np.dot(W1, X) + b1)
 U1 = np.random.rand(*H1.shape) < p # first dropout mask
 H1 *= U1 # drop!
 H2 = np.maximum(0, np.dot(W2, H1) + b2)
 U2 = np.random.rand(*H2.shape) < p # second dropout mask
 H2 *= U2 # drop!
 out = np.dot(W3, H2) + b3
 # backward pass: compute gradients... (not shown)
 # perform parameter update... (not shown)

 def predict(X):
 # ensembled forward pass
 H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
 H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
 out = np.dot(W3, H2) + b3

Dropout Summary

drop in forward pass

scale at test time

:Dropout

נשירה היא טכניקת רגולציה המשמשת למניעת התאמת יתר במודלים של למידה عمוקה, במיוחד בשרותות עצביות.

הרעין המרכזי אחורי נשירה הוא "נשירה" אקראית של חלק של נירונים (יחידות) בשכבה במהלך האימון, ולמעשה להפוך אותם ללא פעילים עבור איטרציה זו. המשמעות היא שהרשota אינה מסתמכת במידה רבה על שום נירון מסוים ובמקרה זאת לא מגדת תכונות חזקיות יותר וניננות להכללה. במהלך היסק (בדיקה או חיזוי), נעשה שימוש בכל הנירונים, אך משקלם מופחת לפי שיעור הנשירה המשמש במהלך האימון כדי לפצות על הנירונים הלא פעילים.

נשירה פועלת כזורה של במידה אנסמבלית, שבה תת-רשאות מרובות מאומנות במקביל לפרמטרים משותפים, אך בכל רשות יש כמה נירונים מושבתים באופן אקראי. אפקט האנסמבל הזה עוזר להפחית התאמת יתר על ידי ייצירת דגם חזק יותר שלא יוכל להסתמך יותר מדי על תוכנה מסוימת או שילוב של תוכנות מה שהופך את המודל שלנו לרובוטי יותר.

נשירה הוכחה כטכניקת רגולציה יעילה, במיוחד בעת אימון רשתות עצביות גדולות וモרכבות. זה יכול להוביל להכללה משופרת, למנוע מהרשota לשன את נתוני האימון ולהפוך אותה למסוגלת יותר לטפל בנתונים חדשים שלא נראים. נשירה נמצאה בשימוש נרחב בארכיטקטורות למידה عمוקה שונות והייתה מרכיב מרכזי בהשגת תוצאות מתקדמות במשימות למידה מוגבהת.

Regularization: A common pattern

Training: Add random noise

Testing: Marginalize over the noise

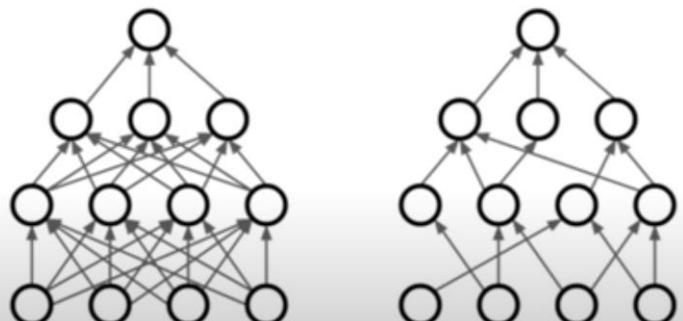
Examples:

Dropout

Batch Normalization

Data Augmentation

DropConnect

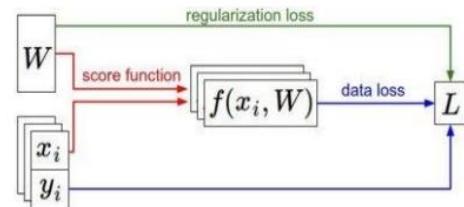


- We have some dataset of (x, y)
- We have a **score function**: $s = f(x; W) = Wx$ e.g.
- We have a **loss function**:

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right)$$

Softmax

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \quad \text{Full loss}$$



:Softmax

הוא לוקח וקטור קלט של מספרים ממשיים וממיר אותם להטפלות הסתברות, כאשר כל אלמנט בפלט מייצג את ההסתברות שהמחלקה המתאימה תהיה המחלקה הנכונה.

בහינתן וקטור קלט $[z_1, z_2, \dots, z_n] = z$, הפונקציה softmax מייצרת וקטור פלט $[s_1, s_2, \dots, s_n] = s$ אשר:

$$(s_1 = \exp(z_1) / \sum \exp(z_i), s_2 = \exp(z_2) / \sum \exp(z_i), \dots, s_n = \exp(z_n) / \sum \exp(z_i))$$

פונקציית softmax "מעמכת" את ערכי הפלט (z) להסתברויות (s) על ידי ערכית כל אלמנט ולآخر מכון מרormal אוטם על ידי חילוקה בסכום כל הערכים המעורכים. הפלט של פונקציית softmax הוא התפלגות הסתברות, כלומר כל האלמנטים בווקטור הפלט s אינם שליליים ומסכימים ל-1.

softmax משמש בדרך כלל כפונקציית הפעלה של שכבה הפלט בעיות סיווג מרובות מחלקות, כאשר המטרה היא להקצות דגימת קלט לאחת מספר מחלקות. המחלקה עם ההסתברות הגבוהה ביותר בפלט softmax נחשבת למחלקה החזiosa עבור הקלט המסוים זהה.

בנוסף לישום שלו בסיווג, softmax משמש גם בתחוםים אחרים של למידת מכונה, כגון באימון של רשתות עצביות עבור שימושים כמו מודל שפה, שם הוא משמש כדי להפוך את ציוני הפלט להסתברויות לייצור המילה הבאה ברכף.

:SVM

אלגוריתם למידת מכונה מפוקחת חזק וופולרי המשמש למשימות סיווג ורגסיה. SVMים ייעילים במיוחד לפתרון בעיות סיווג ביןארי, כאשר המטרה היא לסוג נתונים נתוני קלט לאחת משתי מחלקות. עם זאת, ניתן להרחיב את ה-SVM כדי לטפל גם בסיווג רב-מעמד.

המטרה העיקרית של SVM היא למצוא את המישור האופטימלי המפריד בצורה הטובה ביותר בין נקודות הנתונים של מחלקות שונות במרחב התכוונות. ההיפר-מישור הוא גבול החלטה שמקסם את המרווח (המרחק) בין שתי נקודות הנתונים הקרובות ביותר ממחקות שונות, המכונים וקטורי תמייכה. מושג זה ידוע בתוור סיווג מרוחך מksamימי.

יתרונות:

- יכולתם לטפל בנתונים בממד גבוה

- גבולות החלטה לא ליניארים באמצעות שימוש בקרנליים.
- עיל גם במקרים שבהם הנתונים אינם מופרדים היטב ועלולים לחפו.

חסרונות:

- עלול להיות יקר מבחינה חישובית עבור מערכיו נתונים גדולים
- הביצועים עשויים להיות רגשים לבחירת הפרמטרים של היפר, כגון הפרמטר הליבה והרגולציה.

מצגת 3 (סוף):

גרדיאנט אנלטי:

SHIPOU ANALTI, המכונה גם SHIPOU בצורה סגורה, מתייחס לנזרת של פונקציה המחשבת בצורה אנלטית באמצעות נוסחאות מתמטיות. בהקשר של למידת מכונה ואופטימיזציה, השיפוע של פונקציה מייצג את הכוון והגודל של העלייה או הירידה התולוה ביותר בנקודת מסיממת. הוא מספק מידע חיוני עבור אלגוריתמי אופטימיזציה לעדכון איטרטיבי של הפרמטרים של מודל על מנת לחדור או למקסם את פונקציית המטריה.

הגרדיאנט האNALTI מחושב על ידי לקיחת הנגזרות החלקיים ביחס לכל פרמטר או משתנה. הוא מספק ביטוי מתמטי מדויק לשיפוע מוביל להזדקק לקירוב מספרי או הבדלים סופיים.

לדוגמה, שקול פונקציה רב-משתנית (y, x, f) התלויה במשתנים x ו- y . הגרדיאנט האNALTI של (y, x, f) יהיה הווקטור [$\hat{f}/\hat{x}, \hat{f}/\hat{y}$], המייצג את הנגזרות החלקיים של f ביחס ל- x ו- y , בהתאם.

היתרון של מחושב גרדיאנטים אנלטיים הוא שהם מספקים נגזרות מדויקות ויעילות, במיוחד עבור פונקציות עם נוסחאות מתמטיות ידועות. שיפורים אנלטיים מאפשרים התכנסות מהירה יותר במהלך אופטימיזציה, מכיוון שהם מספקים מידע מדויק על כיוון וקצב השינוי.

עם זאת, מחושב שיפורים אנלטיים לא תמיד אפשרי או מעשי, במיוחד עבור פונקציות או מודלים מורכבים הכוללים פועלות מתמטיות מורכבות. במקרים כאלה, ניתן להשתמש בשיטות מספוריות, כגון הבדלים סופיים או בידול אוטומטי, כדי להעריך את הגרדיאנט. שיטות אלו מעירכות את השיפוע על ידי הרכבת הפונקציה במספר نقاط וחישוב הבדלים סופיים או שימוש בטכניקות אלגוריתמיות להבדלה אוטומטית של הפונקציה.

In summary:

- Numerical gradient: approximate, slow, easy to write
- Analytic gradient: exact, fast, error-prone

=>

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check**.

:Gradient Descent

- We can write the algorithm in terms of the examples
- Let's compute the derivative

$$\begin{aligned}
 \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2 \\
 &= 2 \cdot \frac{1}{2} (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_{\theta}(x) - y) \\
 &= (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right) \\
 &= (h_{\theta}(x) - y) x_j
 \end{aligned}$$

- So we have

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \quad (\text{for every } j)$$

}

:Stochastic Gradient Descent (SGD)

- Do gradient descent
 - But instead of computing the gradient based on all the m examples
 - At each iteration compute the gradient based on a sample of n<<m examples
-

```

# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update

```

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

Full sum expensive
when N is large!

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Approximate sum
using a **minibatch** of
examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

מצגת 4

:Neural Networks and Backpropagation

Where we are...

$$s = f(x; W) = Wx \quad \text{Linear score function}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \text{SVM loss (or softmax)}$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda \sum_k W_k^2 \quad \text{data loss + regularization}$$

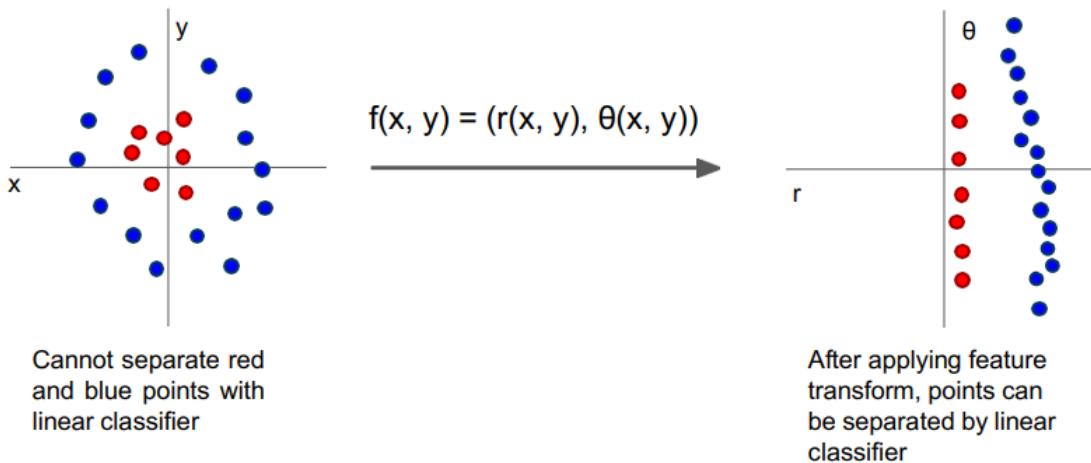
How to find the best W?

$\nabla_W L$

הבעיה: קלואסיפירים לינארים הם לא מספיק עצמאתיים.

- הם לומדים טמפלט אחד פרט לאלס
- קשה להם לזהות צורות ודפוסים שהם לא לינארים...

Image Features: Motivation



תכונות תמונה:

תכונות תמונה מתיחסות לייצוגים או מתאים שתוכננו באופן ידני אשר לצדדים מאפיינים או דפויים **ספרטיפיים** בתמונה. תכונות אלו נשלפות בדרך כלל באמצעות טכניקות כמו SIFT (שינוי מאפיינים בקנה מידה בלתי משתנה), SURF (תכונת חזקות מואצת) או HOG (היסטוגרמה של הדרגות מכוונות). טכניקות אלו שואפות לזרה וליצג דפויים או מבנים מקומיים בתמונה, כגון קצנות, פינות או טקסטורות. לאחר החילוץ, ניתן להשתמש בתכונות אלו למשימות כמו זיהוי אובייקטים, סיוג תמונה או אחזור תמונה.

מאפיינים של תכונות תמונה:

1. עבודה יד: תכונות תמונה מתוכננות ועוצבות על ידי חוקרים או מומחי תחום, שגדירים את הדפויים או המאפיינים הספרטיפיים שהם רצים לפכו.
2. מקומי: תכונות התמונה מחושבות באופן מקומי בתוך התמונה, תוך התמקדות באזוריים קטנים יותר או טליים במקומות להתחשב בתמונה כולה.
3. אי-ויראיות: תכונות תמונה משלבות לעיתים קרובות מאפייני אי-ויראייה, כגון אי-ויראיות בקנה מידה או אי-ויראיות של סיבוב, כדי להפוך אותן לרובוטיות יותר.

רשתות עצביות קונולוציונליות (ConvNets):

יעדוע גם בשם Convolutional Neural Networks או CNNs, הם סוג של מודל למידה עוקבה **שתוכנן במיוחד** לניתוח נתונים חזותיים כמו **תמונות**. ConvNets מורכבים משכבות מרובות של נירונים מחוברים זה לזה, כולל שכבות קונולוציה, שכבות מאגר ושכבות מחוברות לחלוין. **רשתות אלו** לומדות **לחילוץ** **תכונות** **רלוונטיות** **ישירות** **מן-****תמונה** **הגולמיים** **באמצעות** **תהליכי** **הדרכה** **על** **מערכות** **נתונים מסומנים**.

מאפיינים של ConvNets

1. למידה מוקצת לקצה: ConvNets לומדים לחץ אוטומטית תכונות שימושיות מנתוני תמונה גולמיים, מבליהם להסתמך על אלגוריתמים לחילוץ תכונות בעבודת יד. הרשות למדתهن תכונות בrama נמוכה (למשל, קטנות, טקסטורות) והן תכונות ברמה גבוהה (למשל, צורות אובייקט, ייצוגים סמנטיים) בזרה היררכית.

2. ייצוג היררכי: ConvNets מרכיבים בדרך כלל שכבות רבות, המאפשרות להם ללמידה ייצוגים היררכיים של נתוני הקלט. תכונות ברמה נמוכה נלמדות בשכבות הראשונות, ותכונות ברמה גבוהה יותר נלמדות בשכבות הבאות.

3. הקשר גלובלי: ConvNets מחשבים את התמונה כולה במהלך תהליכי חילוץ התכונות. שכבות קונולוציה מבצעות פעולות שדה קליטה מקומות, המאפשרות לרשת למכוד יחסים ומבנה מרחבית על פני התמונה.

יתרונות:

ConvNets הראו הצלחה יוצאת דופן במקריםות שונות של ראיית מחשב, כולל סיוג תמונה, זיהוי אובייקטים, פילוח סמנטי ויצירת תמונות. יש להם יתרון של למידה אוטומטית של תכונות רלוונטיות לשירות מהנתונים, מה שפחית את הצורך בהנדסת תכונות ידנית.

חסרונות:

ConvNets דורשים לעיתים קרובות כמויות גדולות של נתונים אימון מותגים ומשabi חישוב שימושיים לאימון בשל הארכיטקטורות המורכבות שלהם.

:Neural Networks

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

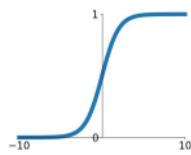
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

“Neural Network” is a very broad term; these are more accurately called “fully-connected networks” or sometimes “multi-layer perceptrons” (MLP)

Activation functions

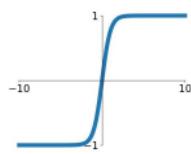
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



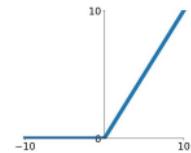
tanh

$$\tanh(x)$$



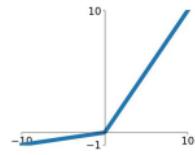
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

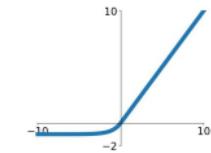


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmoid:

- גורם למספרים להיות בין אפס לאחד.
- פופולרי כי הוא נותן "firing rate" מרווחה של הנירון.

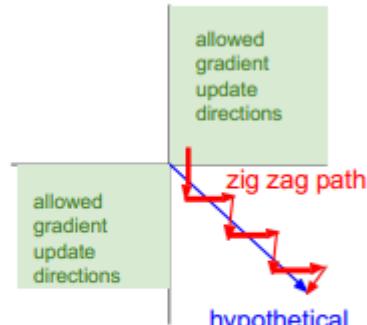
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

בעיות:

- נירונים מרווחים הורגים את הגרדיינט, לדוגמה: $x = -10$.
- החישוב של איקצת יקר.
- התוצאות הם לא סביב האפס.

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on w ?
 Always all positive or all negative :(
 (For a single element! Minibatches help)

Tanh:

- גורם למספרים להיות בין מינוס אחד לאחד.
- תוצאות סביב האפס.

בעיות:

- עדין נירונים מרווחים הורגים את הגרדיינט.

:Relu

- יעיל חישובית.
- מתכנס מהר יותר מקודמי בסיכום.
- הוא איננו מורווה.

בעיות:

- התוצאות הם לא סבירי האפס.
- גרדיאנט נעלם כאשר הנגזרת שלילית – הורג את הגרדיאנט.
- רלי מות – אם המשטנה ביואס שלילי מאוד אז הוא יכול להרוג את המירון בכך שתמיד יהרוג את הגרדיאנט.
- בארכיטקטורה עם הרבה שכבות בעיית הרלי מות יכולה להוביל לירידה משמעותית ביציאות. פתרונות אפשרי – ליק' רלי...

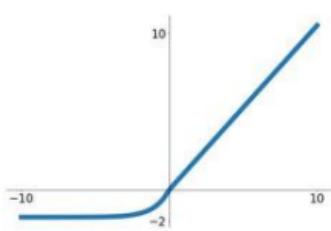
:Leaky ReLU

- לא מורווה.
- יעיל חישובית.
- מתכנס מהר יותר מסיגמוד וטאן אייז'ט.
- לא ימות.

:Exponential Linear Units (ELU)

- כל היתרונות של Relu.
- קרוב יותר למוצע אפס בתוצאות.
- מוסיף קצת רובוטיות לרעש.

Scaled Exponential Linear Units (SELU)



- Scaled version of ELU that works better for deep networks
- “Self-normalizing” property;
- Can train deep SELU networks without BatchNorm
 - (will discuss more later)

$$f(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda \alpha(e^x - 1) & \text{otherwise} \end{cases}$$

$\alpha = 1.6733, \lambda = 1.0507$

TLDR: In practice:

- Use **ReLU**. Be careful with your learning rates
- Try out **Leaky ReLU / Maxout / ELU / SELU**
 - To squeeze out some marginal gains
- Don't use **sigmoid** or **tanh**

:fully connected layer

שכבה מחוברת לחולוטין, הידועה גם בשם שכבה צפופה, היא סוג של שכבה הנפוצה בשימוש ברשתות עצביות, כולל רשתות עצביות (ConvNets) ו- Multilayer Perceptrons (MLPs). הוא אחראי על לימוד יחסים מורכבים בין תכונות הקלט לבין תוויות הפלט.

בשכבה מחוברת לחולוטין, כל נוירון מחובר לכל נוירון בשכבה הקודמת, ויצרת רשת מחוברת לחולוטין. כל חיבור בין נוירונים קשור לממשק שקובע את עצמתו וחישובתו של אותו חיבור.

במהלך שלב ההפצה קדימה, הקלט מהשכבה הקודמת מוכפלים במשקלים התואמים שלהם ומוסברים דרך פונקציית הפעלה כדי לייצר את ערכיו הפלט של השכבה. פונקציית הפעלה מציגה אי-LINEARITY לרשות, ומאפשרת לה למדוד תכונות מורכבות ולבצע טרנספורמציות לא-LINEARITIES.

מבחינה מתמטית, ניתן לחשב את ערך הפלט y של שכבה מחוברת במלואה באופן הבא:

$$y = \text{הפעלה}(b + x * W)$$

- W מייצג את מטריצת המשקל של השכבה, כאשר כל שורה מתאימה למשקלים של נוירון ספציפי.
- x מייצג את וקטור הפלט מהשכבה הקודמת.

- b מייצג את וקטור ההטייה, שמתווסף מבחינה אלמנטית לכינוסות המשקללוות.
- הפעלה היא פונקציית הפעלה המימושה מבחינה אלמנטית לכינוסות המשקללוות.

הפלט של שכבה מחוברת במלואה משמש כקלט לשכבות הבאות בראשת. המשקלים וההטיות של השכבה המחוברת במלואה נלמדים במהלך תהליכי האימון באמצעות אלגוריתמי אופטימיזציה כמו ירידה בשיפוע והפצה לאחרור, כאשר הרשות מתאימה את המשקלות כדי למנוע את ההפסד בין הפלטים החזויים לבין התוויות האמיתיות.

עתים קרובות נעשה שימוש בשכבות מחוברות לחולוטין בשכבות האחרונות של רשת עצבית כדי לייצר את הפלט הרצוי. למשל, שכבת הפלט של רשת היא בדרך כלל שכבה מחוברת לחולוטין עם הפעלת SoftMax, אשר מפיקה את הסתברויות החזויות עבור כל מחלקה.

```
# forward-pass of a 3-layer neural network:  
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)  
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)  
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)  
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)  
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Full implementation of training a 2-layer Neural Network needs ~20 lines:

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```

Define the network

Forward pass

Calculate the analytical gradients

Gradient descent

Problem: How to compute gradients?

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x) \quad \text{Nonlinear score function}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \text{SVM Loss on predictions}$$

$$R(W) = \sum_k W_k^2 \quad \text{Regularization}$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(W_1) + \lambda R(W_2) \quad \text{Total loss: data loss + regularization}$$

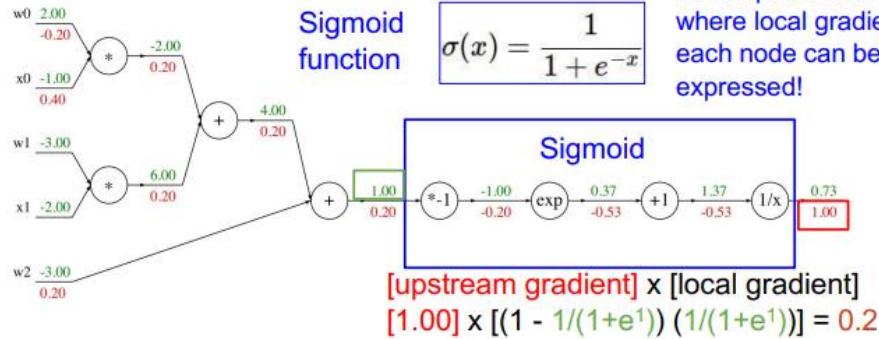
If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn W_1 and W_2

לحساب את הנגזרת על דף לא בא בחשבון. זה מיגע, אם נרצה להחליף פונקציית לוס נהיה ח"ב אם לחשב מחדש כמעט הכל ובלי קשר זה לא רלוונטי לחישובים מסווגים.

פתרונות: Backpropagation

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



Sigmoid local gradient: $\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x)$

:Convolution layer

- שומר על המבנה המרחבי.

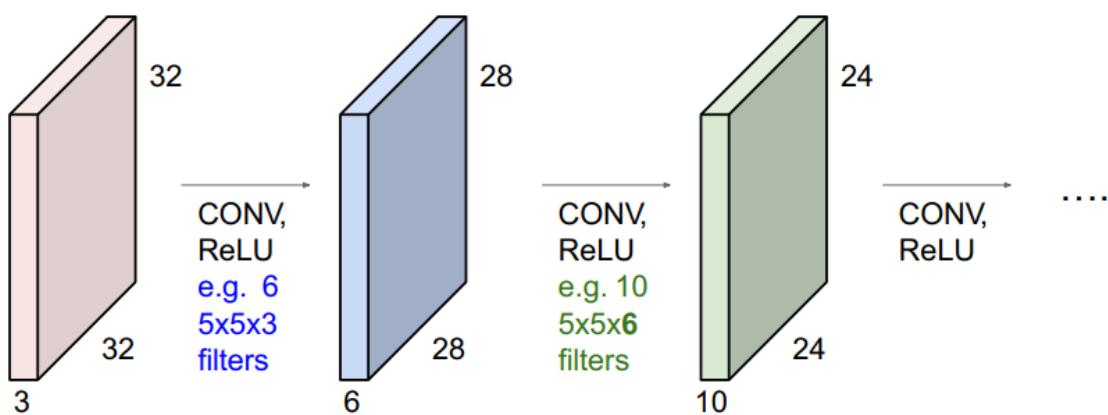
יצירת שכבה הקונולוציה היא בעזרת הפילטרים הנלמדים (המשקלים שלו).
הפלט: $(1 + (\text{צד}) / (\text{רוחב - אורך}))$.

הפלט עם פדים: $(1 + (\text{צד}) / (\text{פדים}^2 + \text{רוחב - אורך}))$.

הערה: נפוץ לרפד עם $1/2$ – גודל פילטר.

בעיה: שימושתמים בפילטרים מסוימים באופן חוזר עוזמת המרחב מתכווץ:

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



דוגמא לחישוב כמה פרמטרים יש לנו בשכבה:

Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2



Number of **parameters** in this layer?

each filter has **5*5*3 + 1 = 76** params (+1 for bias)
=> **76*10 = 760**

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$

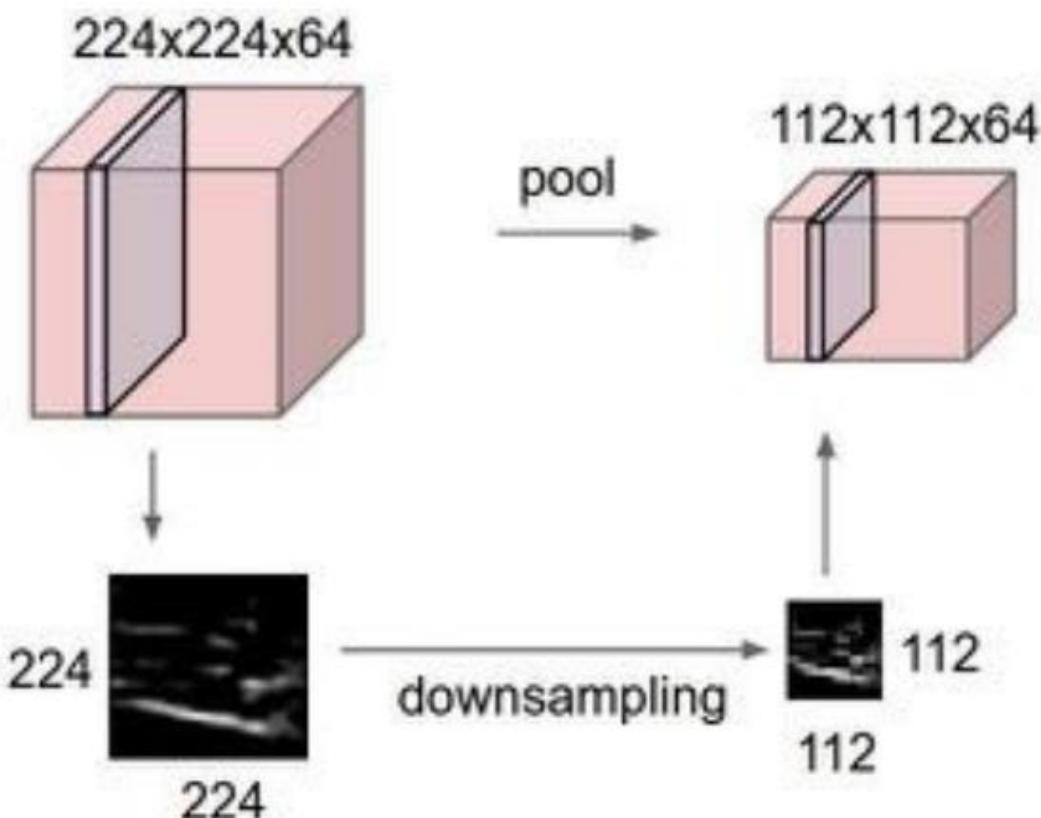
where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

:Pooling layer

הופך את הייצוג של השכבה לפחות יותר וניתן יותר לניהול.



לפי פילטר בגודל מסוים לוקח את הערך המקסימלי באותו שטח ובכך מקטין את המינימום.

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

This will produce an output of $W_2 \times H_2 \times C$ where:

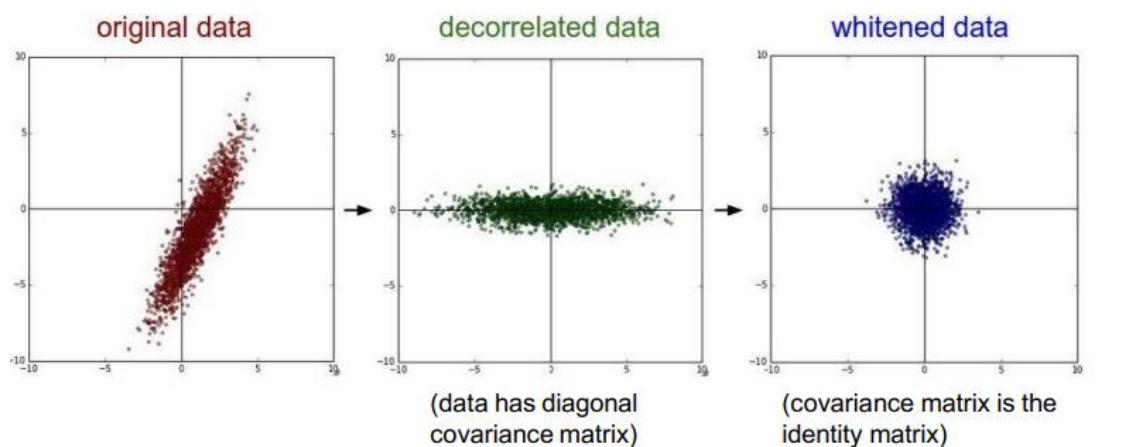
- $W_2 = (W_1 - F) / S + 1$
- $H_2 = (H_1 - F) / S + 1$

Number of parameters: 0

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like
 $[(\text{CONV-RELU})^N \cdot \text{POOL?}]^M \cdot (\text{FC-RELU})^K \cdot \text{SOFTMAX}$
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
 - but recent advances such as ResNet/GoogLeNet have challenged this paradigm

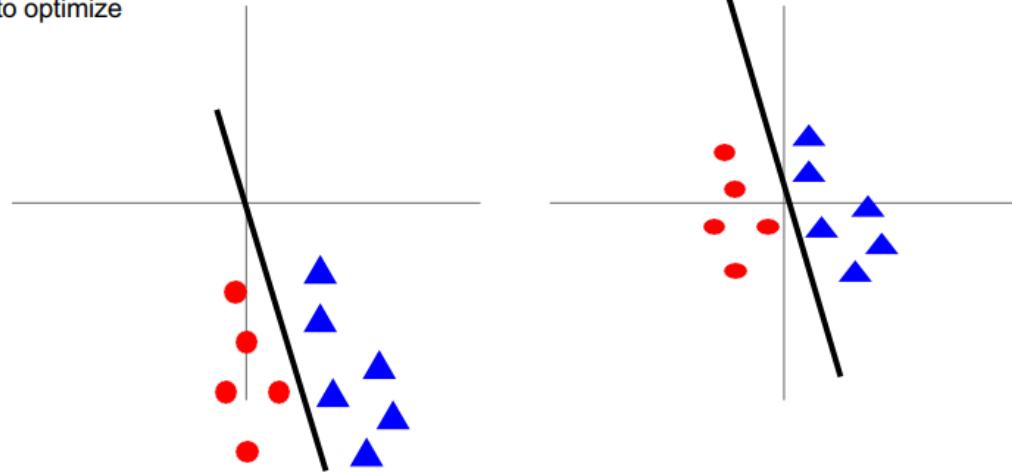
:Data Preprocessing



. בפועל - PCA

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize

After normalization: less sensitive to small changes in weights; easier to optimize



:Weight Initialization

רעיון ראשון: נעדכן משקלים רנדומליים וטיה סטנדרטית קבועה. **בעיה:** עובד טוב ברשומות קצרנות בלבד.

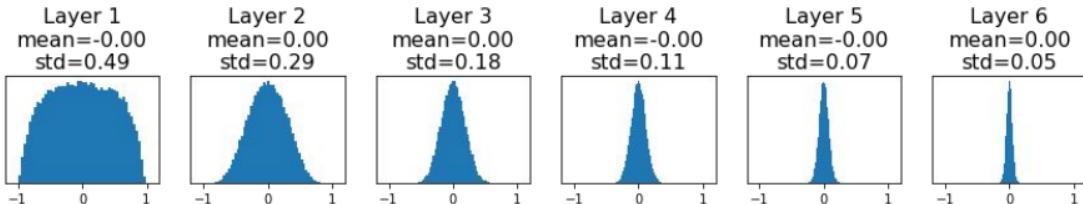
Weight Initialization: Activation statistics

```
dims = [4096] * 7      Forward pass for a 6-layer
hs = []                  net with hidden size 4096
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?

A: All zero, no learning =(



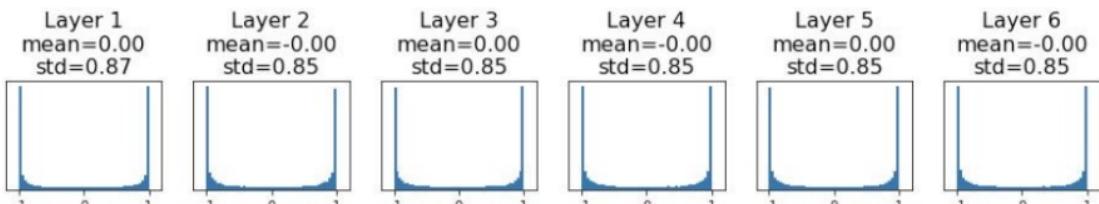
Weight Initialization: Activation statistics

```
dims = [4096] * 7      Increase std of initial
hs = []                  weights from 0.01 to 0.05
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

All activations saturate

Q: What do the gradients look like?

A: Local gradients all zero, no learning =(

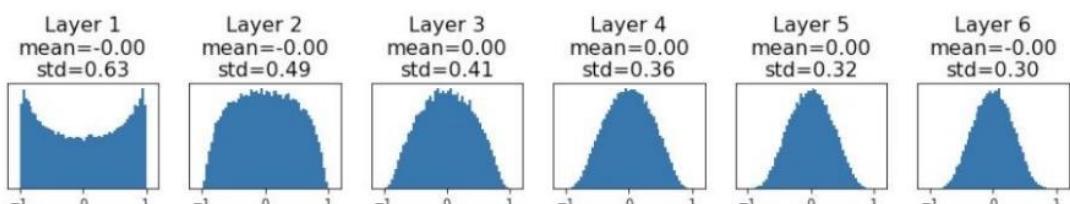


Weight Initialization: “Xavier” Initialization

```
dims = [4096] * 7      "Xavier" initialization:
hs = []                  std = 1/sqrt(Din)
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Just right”: Activations are nicely scaled for all layers!

For conv layers, Din is $\text{filter_size}^2 * \text{input_channels}$



Weight Initialization: “Xavier” Initialization

```

dims = [4096] * 7      "Xavier" initialization:
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)

```

“Just right”: Activations are nicely scaled for all layers!

For conv layers, Din is kernel_size² * input_channels

Derivation:

$$y = Wx$$

$$\text{Var}(y_i) = \text{Din} * \text{Var}(x_i w_i)$$

$$h = f(y)$$

$$= \text{Din} * (\mathbb{E}[x_i^2]\mathbb{E}[w_i^2] - \mathbb{E}[x_i]^2 \mathbb{E}[w_i]^2)$$

$$= \text{Din} * \text{Var}(x_i) * \text{Var}(w_i)$$

[Assume x, w are iid]

[Assume x, w independant]

[Assume x, w are zero-mean]

If $\text{Var}(w_i) = 1/\text{Din}$ then $\text{Var}(y_i) = \text{Var}(x_i)$

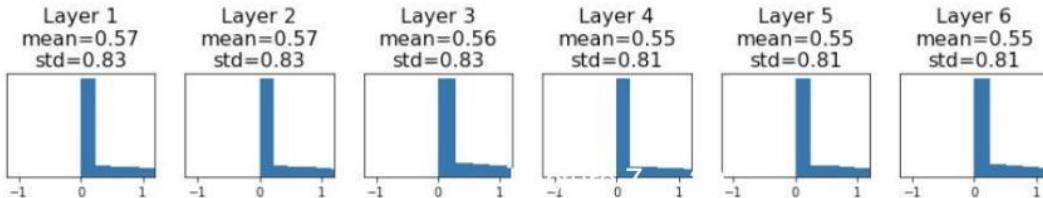
Weight Initialization: Kaiming / MSRA Initialization

```

dims = [4096] * 7  ReLU correction: std = sqrt(2 / Din)
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)

```

“Just right”: Activations are nicely scaled for all layers!



Batch Normalization

Batch Normalization (BatchNorm) is a technique used in deep learning models, particularly in convolutional neural networks (ConvNets), to improve the training speed and stability of the network. It aims to address the internal covariate shift problem, which refers to the change in the distribution of layer inputs as the parameters of the preceding layers change during training.

The main idea behind BatchNorm is to normalize the input to a layer by subtracting the mean and dividing by the standard deviation of the batch of inputs. This normalization step is performed independently for each feature dimension in the input. The normalized inputs are then scaled and shifted using learnable parameters, known as the scale and shift parameters, which allow the model to adapt the representation.

By normalizing the inputs within each mini-batch during training, BatchNorm reduces the dependence of the network on the scale and distribution of the inputs. This helps to stabilize and speed up the training process. Additionally, it has a regularizing effect by introducing some noise to the model, which can reduce overfitting.

During inference or testing, BatchNorm uses the population statistics (mean and standard deviation) estimated during training to normalize the inputs, rather than computing the

statistics from the current batch. This ensures consistent behavior between training and inference.

Benefits of using BatchNorm include:

1. Improved convergence: By reducing the internal covariate shift, BatchNorm enables faster convergence during training, allowing the network to reach higher accuracies in fewer training iterations.
2. Regularization: The normalization and noise introduced by BatchNorm act as a form of regularization, reducing overfitting and improving generalization.
3. Reduced sensitivity to initialization: BatchNorm reduces the sensitivity of the network to the choice of initial weights. It allows the network to converge even with suboptimal weight initialization.
4. Gradient flow: BatchNorm helps to mitigate the vanishing or exploding gradients problem by ensuring that the inputs to each layer have a similar scale, which improves the flow of gradients during backpropagation.

BatchNorm has become a standard component in many deep learning architectures, and it has shown significant benefits in improving the training stability, convergence speed, and generalization performance of ConvNets.

"אם אתה רוצה פונקציית אקטיבציה עם ממוצע אף פשוט תיצור אותה צו"

תרומות:

- * עוזר להטמודד עם גרדיאנטים לא יציבים.
- * להתאמן מהר יותר.
- * מתמודד עם אוברפיטינג.

נורמליזציה – להפוך את הקלט להיות בין 0 ל 1.

סטנדריזציה – להפוך את הממוצע להיות 0 ואת השונות להיות 1.

למה שנדורך את זה? בין כל שכבה אנחנו נכנסים באטץ' נורמליזציה, כלומר יבוצע נורמליזציה וסטנדריזציה. כביכול כל איפוק לוקח יותר זמן אבל ניתן הגיעו לתוצאות דומות עם פחות איפוקים מאשר באטץ' נורמליזציה.

consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

this is a vanilla
differentiable function...

Batch Normalization: Test-Time

Estimates depend on minibatch;
can't do this at test-time!

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

Learning $\gamma = \sigma, \beta = \mu$
will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j} \quad \text{Per-channel mean, shape is } D$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2 \quad \text{Per-channel var, shape is } D$$

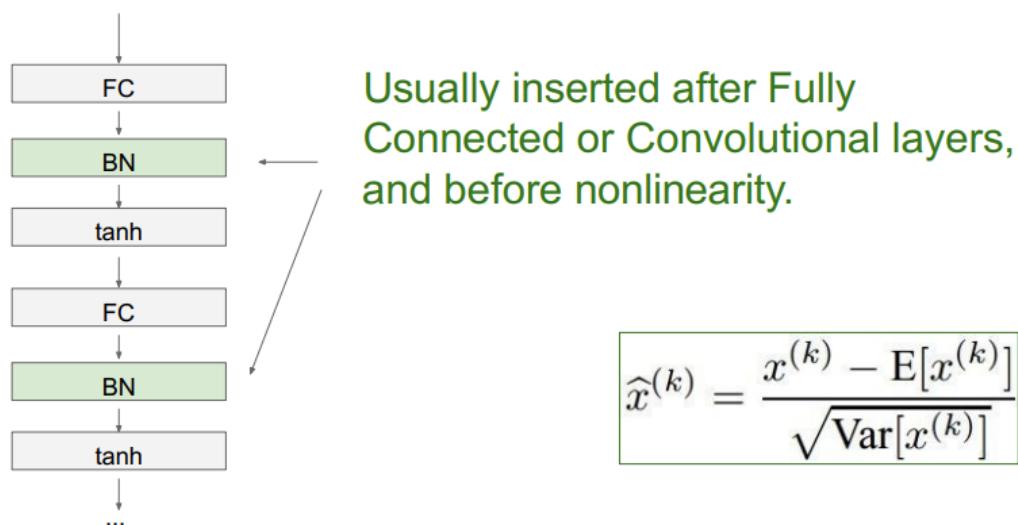
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \quad \text{Normalized x, Shape is } N \times D$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j \quad \text{Output, Shape is } N \times D$$

הערה: בזמן בדיקה הוא הופך לפונקציה לאנארית! אפשר לשים אותו לאחר פוליאנאריטי קונקטיד או שכבת קונולוציה קודמת.

Batch Normalization

[Ioffe and Szegedy, 2014]



תירונות:

הופך את הרשות לכליה יותר לאימוץ.

אפשר קפיצות גדולות יותר והתקנסות מהירה יותר.

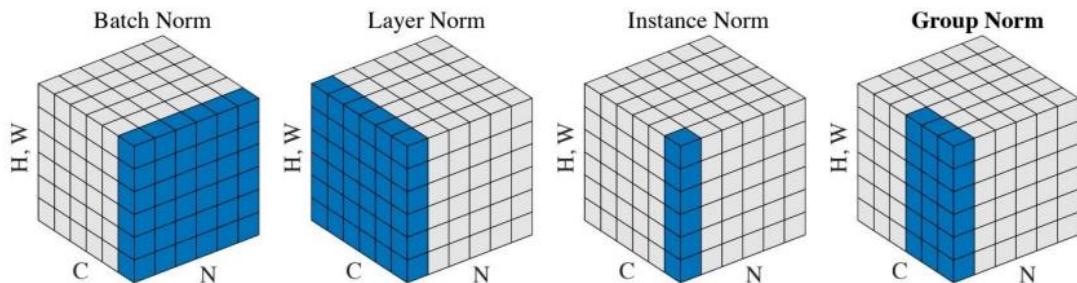
פועל כרגולרייזציה בזמן אימון.

אפס תקורה בזמן הבדיקה.

בעיה:

מתנהג שונה בזמן אימון וטסטים – מקור לבאגים.

צריך הרבה DATA כדי לאמן רשות ס' אן אן (קונולוציה).

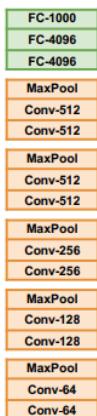


: Transfer learning

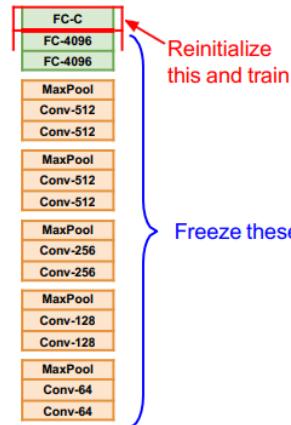
Transfer Learning with CNNs

Donahue et al., "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al., "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

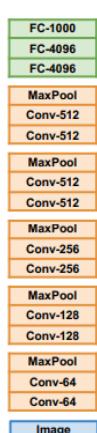
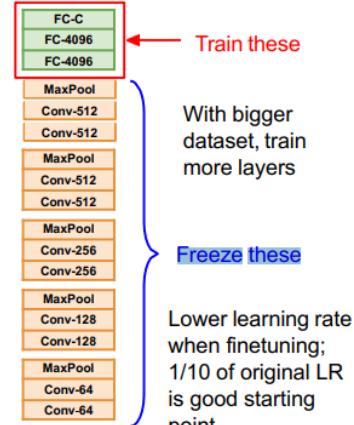
1. Train on Imagenet



2. Small Dataset (C classes)

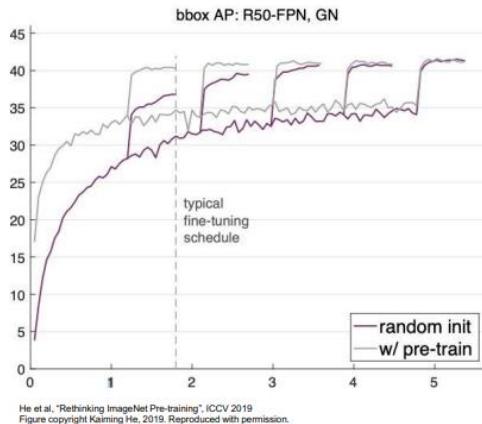


3. Bigger dataset



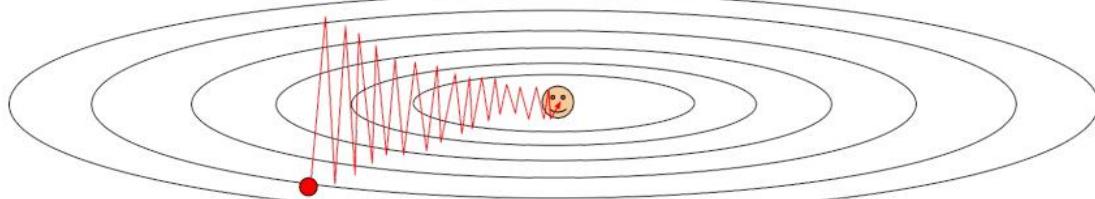
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

בעריה: להתאמן מפעם אחרה יכול לטעות אותו דבר כמו לקחת מודל מאומן ולהמשיך לאמן אותו אבל זה לוקח הרבה יותר זמן. **בנוסף**, מצאו שלבסוף עוד מידע זה יותר טוב מאשר לעשות פין טינויניג על מטלה דומה.



:Optimization

בעיות עם גי די: התקדמות איטית מאוד לאורך המימד הרדום, ריצוף לאורך כיוון תלול.



אם לפונקציית הלוס יש מינימום לוקאלי אז מקבל גרדיאנט אףו ולכז נתקע – נפוץ במימדים גבוהים. הגרדיאנט שלנו מגיע מימי באטץ' ולכז יכול להיות עם רעש.

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

$$\begin{aligned} v_{t+1} &= \rho v_t + \nabla f(x_t) \\ x_{t+1} &= x_t - \alpha v_{t+1} \end{aligned}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

עם מומנטום SGD

המשמעות מתעדכנות בכל SGD-בסיסי. SGD עם מומנטום הוא הרחבה של אלגוריתם SGD איטרציה על ידי חישוב הגרדיאנט של פונקציית ההפסד ביחס למשקלים והתאמת המשקלות לכיוון יכול להיות איטי להתכנס, במיוחד כאשר לפונקציית ההפסד יש SGD, הגרדיאנט השילוי. עם זאת משטחים לא סדריים.

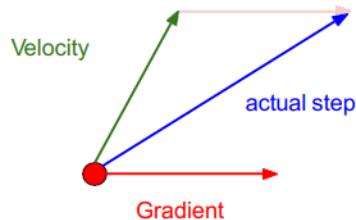
מומנטום מטפל בסוגיה זו על ידי הצגת מונח "מומנטום", המציין את ירידת השיפוע בכיוון הרלוונטי ומשכך תנודות בכיוונים לא רלוונטיים. הוא עושה זאת על ידי שימוש שמיירה על מומצע דעיכה אקספוננציאלית של שיפועים עבר ושימוש במידע שיפוע מצטבר זה כדי לעדכן את המשקלות.

```
v = beta * v - learning_rate * gradient
weights = weights + v
```

דוגמה:

כאן, כי הוא המהירות או השיפוע המצטבר, בטא הוא מקדם התנופה (בדרכו כל ערך בין 0 ל1), קצב הלמידה הוא גודל הצעד, ושיפוע הוא השיפוע של פונקציית ההפסד.

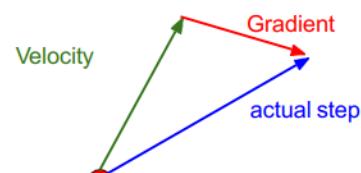
Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Ierov, "A method of solving a convex programming problem with convergence rate O(1/k^2)", 1983

Nesterov Momentum



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

שיפוע מואץ של נסטורוב(אן אי גי):

עם מומנטום שմשפר עוד יותר את התנוגות SGD הוא גרסה של SGD שיפוע מואץ של התכנסות. במומנטום סטנדרטי, השיפוע מוערך בנקודה הנוכחית ומשמש לחישוב מונח המומנטום. עם זאת, זה יכול לגרום למומנטום לחרוג ולנקוט בצעדים מיותרים כאשר המיקום הנוכחי קרוב למיינימום מקומי.

נתפל בבעיה זו על ידי שילוב של צעד מבט קדימה. במקום להעריך את SGD שיפוע מואץ של השיפוע בנקודה הנוכחית, הוא מעריך את השיפוע בנקודה שאליו מונח המומנטום ייקח אותו. זה מאפשר לאלגוריתם לקבל הערכה מדויקת יותר של השיפוע ולהתאים את המשקלות בהתאם.

דוגמה:

```
v = beta * v - learning_rate * gradient(weights + beta * v)
weights = weights + v
```

הן אס גי די עם מומנטום והן שיפוע מואץ של נסטורוב הוכחו כמשפרות את מהירות התכנסות והביצועים של מודלים של במידה עמוקה. עם זאת, הבחירה ביניהם תלויה לעיתים קרובות בבעיה ובמבנה הנתונים הספציפיים, ומומלץ להתנסות בשתי הטכניקות כדי לקבוע איזו מהן העבודה הכל טוב עבור תרחיש מסוים.

:AdaGrad

באלגוריתמים מסורתיים של ירידה בשיפוע, נעשה שימוש בקצב למידה יחיד עבור כל פרמטרי המודל במהלך כל איטרציה. עם זאת, ניתן שגישה זו לא תהיה אידיאלית מכיוון שלפרמטרים מסוימים יכול להיות גודל גרדיאנט גדול בהרבה מאשרים, מה שוביל להתכנסות איטית או תנודות.

נטפל בבעיה זו על ידי התאמת שיעורי הלמידה עבור כל פרמטר על סמך השיפורים ההיסטוריהים שלו. הרעיון המרכזי מאחורי הוא לתת שיעורי למידה קטנים יותר לפרמטרים המתעדכנים לעיתים קרובות ושיעורי למידה גדולים יותר לפרמטרים שיש להם עדכונים נדירים.

האינטואיציה מאחורי היא שלפרמטרים המתעדכנים לעיתים קרובות יהיו מctraversים גדולים, מה שוביל ל渴求 למידה קטן יותר, מה שעזר לייצר את תהליך האימון. מצד שני, לפרמטרים שמקבלים עדכונים נדירים יהיו מctraversים קטנים יותר, מה שוביל ל渴求 למידה גדולים יותר, מה שמאפשר הסתגלות מהירה יותר בעת עדכונים.

אדריאן הוא אלגוריתם אופטימייזציה רב עצמה, במיוחד בתרחישים שבהם יש שיפורים דלילים או כאשר מתמודדים עם בעיות אופטימייזציה לא קמורות וѓבוחות ממדים. עם זאת, **מגבלת אחת שלו היא ששיעור הימידה עשויים להיות קטנים מדי עם הזמן, ולמעשה להאט את תהליך הלמידה באופן משמעותי.** כדי לטפל בבעיה זו, הוצעו אלגוריתמים אחרים של אופטימייזציה אדריאנית כמו RMSprop, Adam

RMSProp: “Leaky AdaGrad”

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

ב RMSProp מתקנים את קצב השינוי של ה learning rate.

- Learning rate decay מתקנים את מספר הצעדים ככל שהקצב למידה מתקדם לנקודת האוקף. וכן בהסתברות גבוהה נמצא את המינימום.

:Adam

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

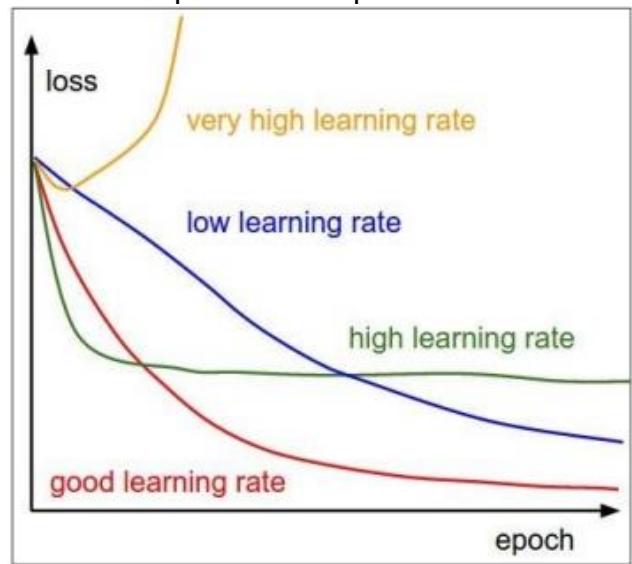
AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

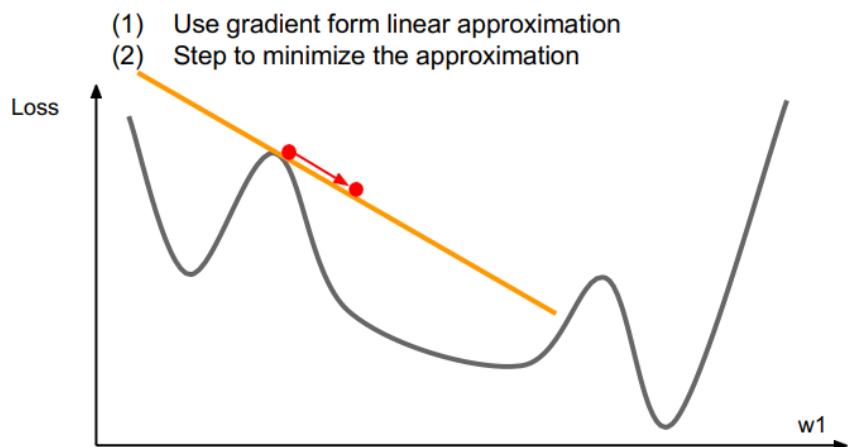
:Learning Rate Decay

לכל האופטימייזרים שראינו יש היפר פרמטר שנקרא "קצב למידה". איך נדע מי הכי טוב? כמובן טוביים כדי להתחיל בערך גבוה ועם הזמן לרדת.

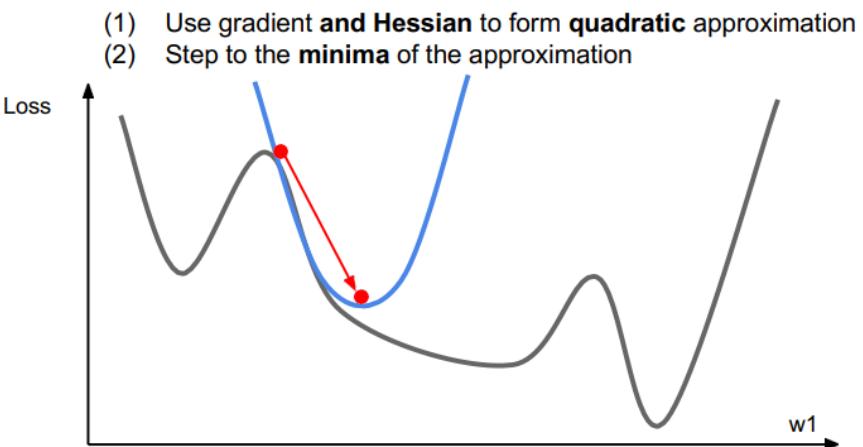


חוק אצבע – אם מגדילים את הבאטץ' באן כדי להגדיל גם את הקצב למידה באן.

First-Order Optimization



Second-Order Optimization



:L-BFGS

בדרך כלל עובד טוב בבאצ' מלא במצב דטרמיניסטי. כמובן, אם יש לנו פונקציה דטרמיניסטית אף איקס אז הוא נראה יעבוד ממש טוב.

לא עובר בצורה טובה להגדרות מיני-באצ'. נותן תוצאות לא טובות. מתאים שיטות ברמה השנייה למינדים גדולים .

בפועל:

אדם – בחירה דיפולרית בהרבה מקרים. הרבה פעמים עובד טוב אפילו עם קצב למידה קבוע.

אס גי די + מומנטום – יכולה לעבוד יותר טוב מאשר אדם אבל יצטרך נראה הרבה שינוי בקצב הלמידה ...

הערה: אם ניתן להרשות לעצמו באצ' מלא אז ננסה את האחרון (אל-בי אף גי אס).

לסיכום:

אלגוריתמים לאופטימיזציה טובים עוזרים להפחית את הלווט באיימון. אבל מה שבאמת חשוב לנו להפחית זה להפחית את הפער בין הדיקון באימון שבדרך כלל יותר טוב לדיקון לדאטה חישוב (ולידציה).

עצור את הלימוד כאשר הדיקון של הולידציה יורדת או מואמן לאורך הרבה זמן אבל תמיד נעקוב אחרי תמונהת המצב של המודל שעוזב היכי טוב על הולידציה.

:Model Ensembles

נאמן כמה מודלים ובזמן בדיקה נעשה ממוצע ביניהם.

:Regularizations

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout

Batch Normalization

Data Augmentation

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

:Dropout

בכל מעבר קדימה נבטל באופן רנדומלי אחוז מסוים של נירונים ע"י כך שנאפסו אותם. כמה אחוזים זה היפר פרמטר. 0.5 זה אחוז נפוץ. מה שמנוע התאמת משותפת של תכונות.

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

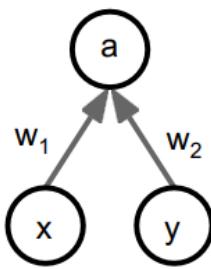
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

Want to approximate
the integral

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

Consider a single neuron.



At test time we have: $E[a] = w_1x + w_2y$

During training we have:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

At test time, multiply by dropout probability

```
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

At test time all neurons are active always

=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

Inverted dropout

```

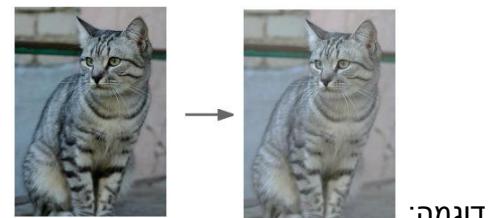
def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

```

הקוד של הטסט נשאר זהה.

Data Augmentation

הזזה אופקית, חיתוך רנדומלי והגדלה, שינוי צבע\תאורה, רוטציה, מתייהה, וכו'...



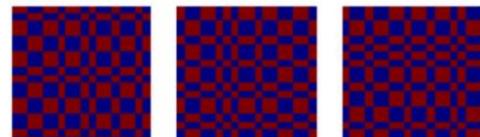
Fractional Pooling

Training: Use randomized pooling regions

Testing: Average predictions from several regions

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling



Stochastic Depth

Training: Skip some layers in the network

Testing: Use all the layer

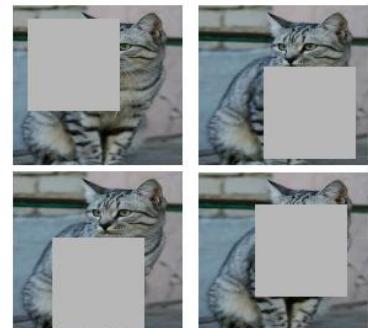
Cutout

Training: Set random image regions to zero

Testing: Use full image

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop



Mixup

Training: Train on random blends of images

Testing: Use original images

Examples:

Dropout
 Batch Normalization
 Data Augmentation
 DropConnect
 Fractional Max Pooling
 Stochastic Depth
 Cutout / Random Crop
 Mixup



בפועל:

באיימון – מוסיף רעש רנדומלי.

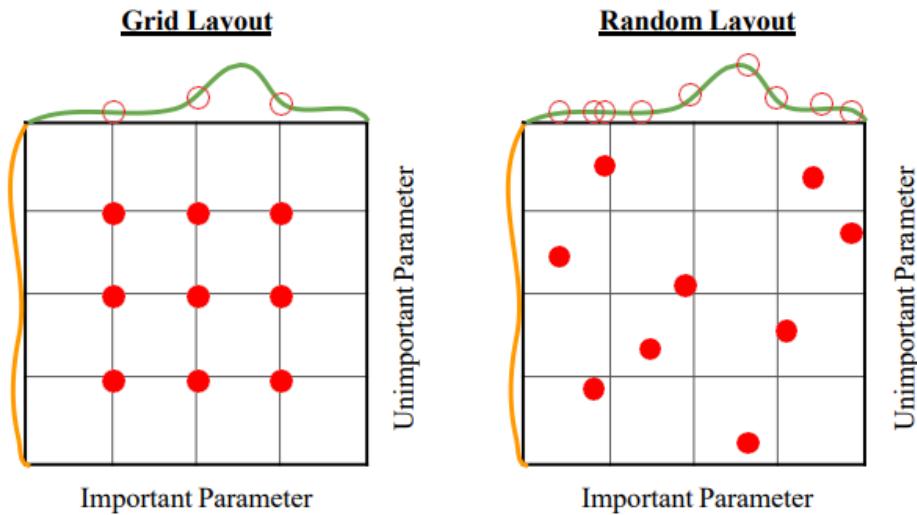
בטעט – חישוב ההסתגלות השולית ע"י סכמתה של כל הערכים האפשריים של משתני הרעש, ובכך נקבל התפלגות על המשתנים המעניינים.
 בשקלן דרופאוט עבר שכבות פולִי קונקטד גדולות.
 באטץ' נורמליזציה ואגמננטציה הם כמעט תמיד רעיונות טובים.

שלבים בבחירה היפר פרמטרים:

1. בדיקת הלסוס הראשון.
2. אובייקטיב כמות קטנה – לנסות להשיג 100% דיוק באימון.
 אם הלסוס לא יודע – קצב אימון נמוך מדי', אתחול לא טוב.
 אם הלסוס מתפוצץ או נאנז – קצב לימוד מהיר מדי', אתחול לא טוב.
 3. למצוא קצב לימוד שייגרום ללסוס לרדת.
4. להכירח גריד מסוים, לאמן 5-1 איפוקים – לבחור כמה קצבים ולאמן עליהם את הדוגמאות.
5. לבחור את המודל הכי טוב מ4 ולאמן אותו יותר בלי יתרה בקצב לימוד.
 6. לבחון את העקומה של הלסוס.
 אם הדיוק של הולידציה ממשיך לעלות – לאמן יותר.
 אם נפתח פער בין הולידציה לאיימון מבחינת דיוק – אובייקטיבינג.
 אם אין פער – אנדרפיטינגן, לאמן יותר ומודל גדול יותר.
היפר פרמטרים לדוגמה – ארכיטקטורת הרשת, קצב למידה, רגולציה, דרופאוט....

Random Search vs. Grid Search

*Random Sea
Hyper-Parar.
Bergstra and*



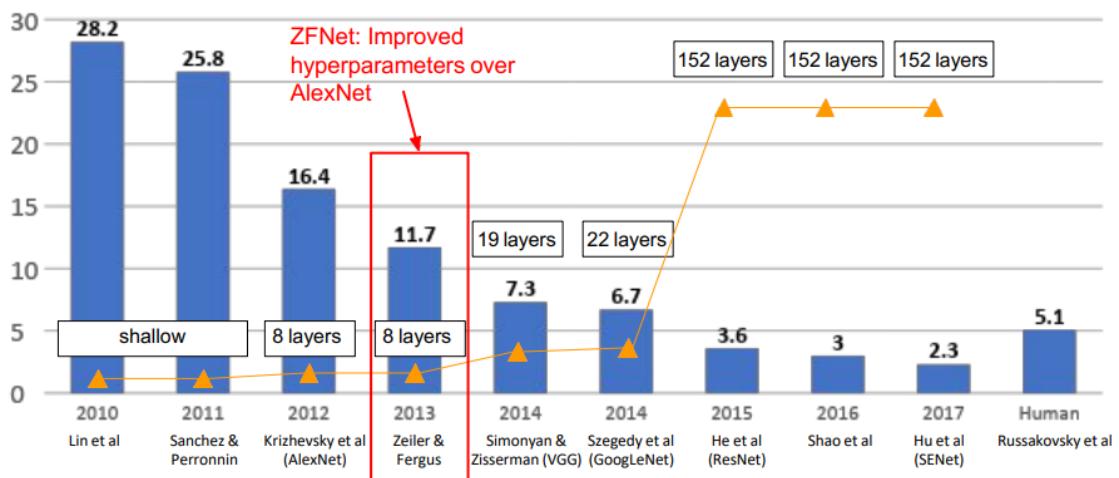
לטיכופ:

כדי לשפר את הטעות באימון- אופטימיזציה, התאמת קצב הלמידה.

כדי לשפר את הטעות בטפס – רגולציה, בחירת היפר פרמטרים.

:CNN Architectures

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



:AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

- [227x227x3] INPUT
- [55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
- [27x27x96] MAX POOL1: 3x3 filters at stride 2
- [27x27x96] NORM1: Normalization layer
- [27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
- [13x13x256] MAX POOL2: 3x3 filters at stride 2
- [13x13x256] NORM2: Normalization layer
- [13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
- [13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
- [13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
- [6x6x256] MAX POOL3: 3x3 filters at stride 2
- [4096] FC6: 4096 neurons
- [4096] FC7: 4096 neurons
- 1000 FC8: 1000 neurons (class scores)

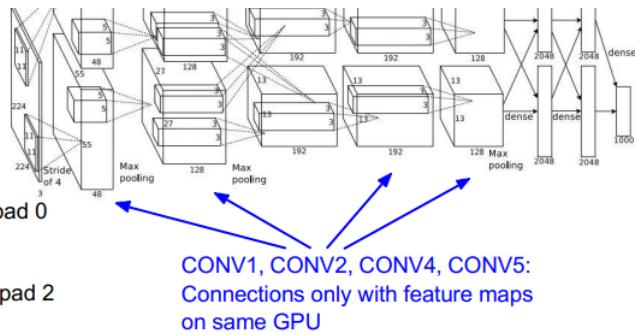


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

8 שכבות.

שכבה ראשונה:

קלט: 227*227*3

פילטרים: 96 פילטרים של 11 על 11 עם סטריד של 4.

פלט: [55*55*96] כי $55 \times 55 \times 96 = (227 - 11)/4 + 1 = 55$.

מספר פרמטרים: 35 אלף = $(11 \times 11 \times 3) \times 96 = 35$.

...

:VGGNet

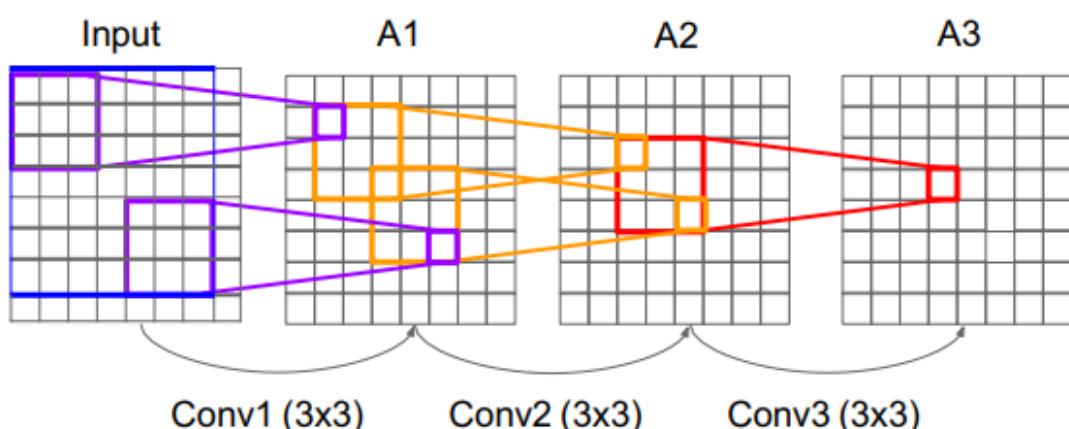
16-19 שכבות.

שימוש בפילטרים קטנים, ורשת עמוקה יותר.

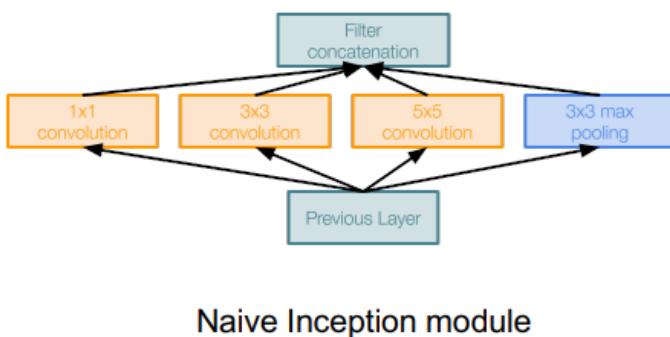
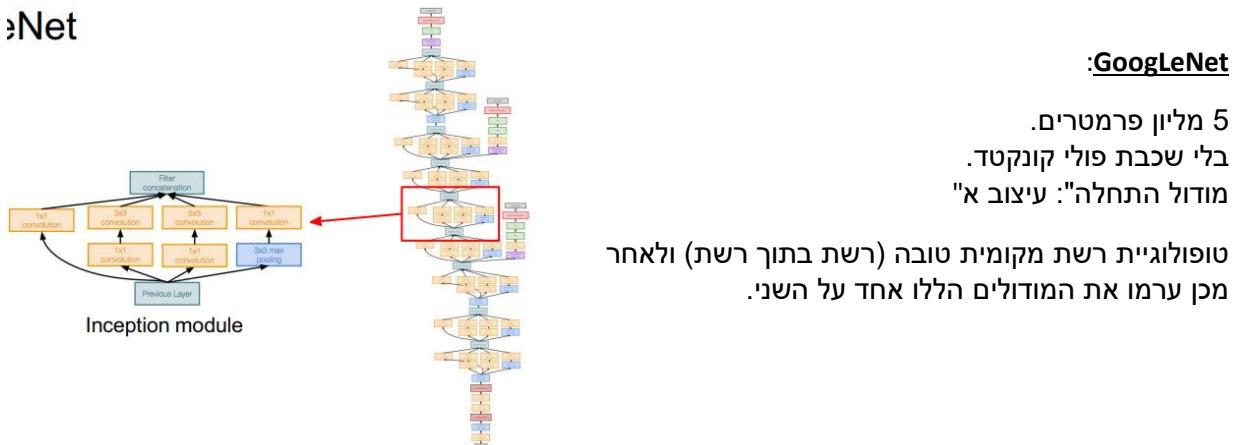
רק קונולוציות של 3 על 3 עם סטריד 1 ופָד 1 ומוקס פולינג 2 על 2 עם סטריד 2.

הסיבה לשימוש בפילטרים הקטנים – 3 פילטרים קטנים של 3 על 3 יש את אותה כמות שדה קליטה כמו 7 על 7 אבלعمוק יותר ויוצר שכבות לא לינאריות ופחות פרמטרים: $(2^7)^2$ (כמויות צ'אנלים) * 7^2 לעומת $(2^3)^2$ (כמויות צ'אנלים) * 3^2 .

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Inception

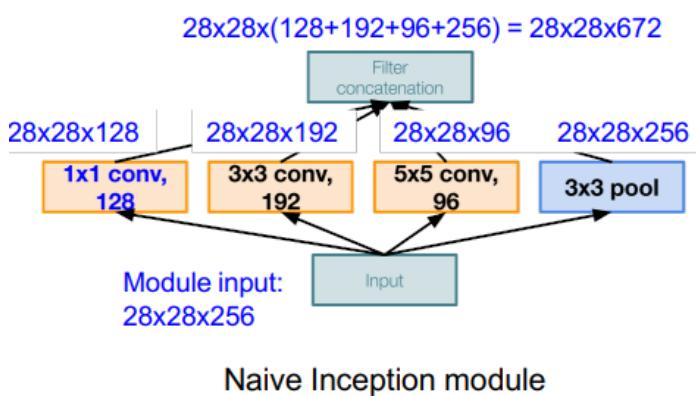


Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What is output size after filter concatenation?



Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:

[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$
[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total: 854M ops

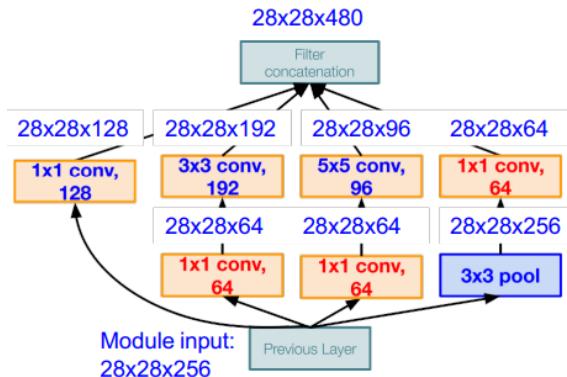
Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

פתרון: שכבות "צואר בקבוק" שמשתמשות בקונולציות של 1 על 1 כדי להוריד את גודל הצי'אנלים של הפעיצרים.

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

הערה: אחרי השכבה קונולוצית האחרונה משתמשים בשכבה פוליג שמצועת מרחיבת כל פיצר אף לפני השכבה פול' קונקטד האחרונה – לא משתמשים יותר בכמה שכבות פול' קונקטד – יקר.

:ResNet

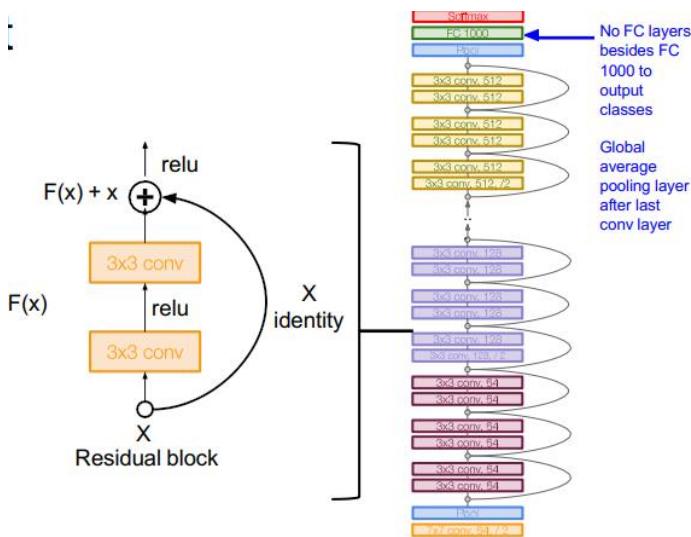
רשתות עמוקות מאוד משתמשות בחיבורים רזידואליים.

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both training and test error

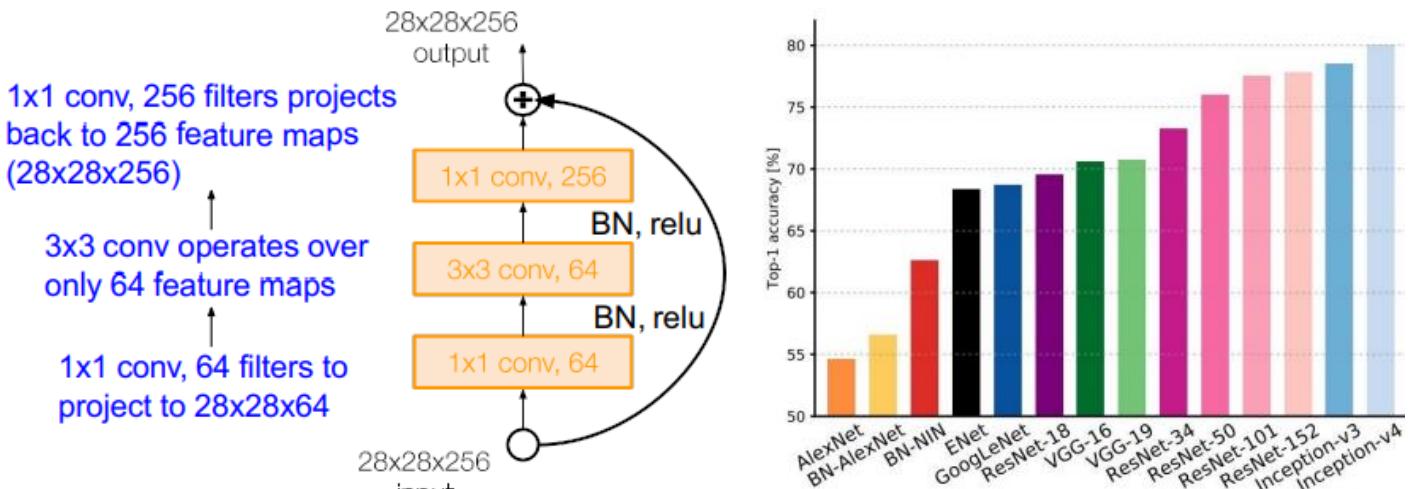
-> The deeper model performs worse, but it's **not caused by overfitting!**



עובדה: למודלים עמוקים יש יותר כוח הציגה (ויתר פרמטרים) מאשר מודלים רדודים יותר.

השערה: הבעה באופטימיזציה – קשה יותר לעשות אופטימיזציה למודלים עמוקים.

מה המודל העמוק יכול לעשות כדי להיות לפחות טוב כמו הרdock? עתיק אותו וויסוף שכבות למפת הזרחות.



הערה: ברשתות עמוקות יותר הם משתמשים ב"צואר בקבוק".

Identity Mappings in Deep Residual Networks

يُذكر أن هناك طرق أخرى لـ"الصياغة" (Residual Connection) في شبكات عميقه، مثل إضافة خروج المدخل إلى خروج الشبكة قبل الهدف.

EIFITI :

מייפוי זהה ברשתות שיירויות עמוקות מתייחס לרעיון של חיבור ישיר של הקלט של שכבה לפلت שהן סוג פופולרי של ארכיטקטורת (ResNets) של ה-ResNets. רעיון זה הוא מרכיב בסיסי של רשתות שיירויות רשת עציבית عمוקה.

ברשת נירונית סטנדרטית, כל שכבה לומדת מייפוי מהקלט שליה לפلت שלה. עם זאת, ככל שהרשת הופכת עמוקה יותר, זה יכול להיות מתאגר עבור הרשת ללמידה "יצוגים שימושיים והשיפועים יכולים מתפלים בבעיה זו על ידי הצגת חיבור ResNets. להיעלם או להפוץ, מה שמקשה על האימון דילוג, או קיצורי דרך, המאפשרים לרשת ללמידה מייפויים שיוריים.

היא שקול יותר לרשת ללמידה את מייפוי השינוי (ההבדל בין הפלט ResNets התובנה המרכזית של הרצוי לקלט) במקום לנסוטה ללמידה את כל המייפוי מאפס. זה מושג על ידי הצגת מייפוי זהות, המאפשרים לרשת להפיץ את הקלט שיירות לשכבות הבאות.

בלוק שיורי מורכב מספר רב של שכבות מוערמות, כאשר הקלט של הבלוק מועבר, ב-ResNets, דרך סדרה של טרנספורמציות לא-لينיאריות. מייפוי זהות מתווסף לאחר מכן לפلت של טרנספורמציות אלה. חיבור הדילוג מאפשר לשיפור לזרום שיירות מהפלט לקלט, תוך עקיפת שכבות הביניים. מבחינה מתמטית, ניתן להגדיר את הפלט של בלוק שיורי כ:

$$(\text{קלט}) F + \text{פלט} = \text{קלט}$$

מייצג את הטרנספורמציות שהופעלו על הקלט, ופלט מייצג 'F', כאן, 'קלט' מייצג את הקלט לבlok את הפלט הסופי של הבלוק.

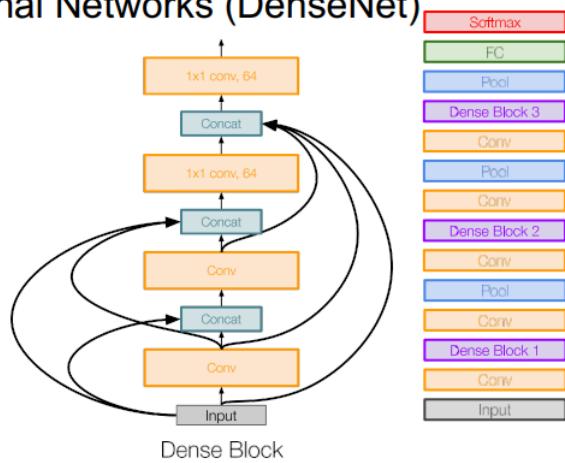
מאפשרים לרשת ללמידה פונקציות שיירויות, מה שמאפשר אימון ResNets, על ידי שילוב מייפוי זהות כל יותר של ארכיטקטורות עמוקות. חיבור הדילוג מספקים גם דרך להפיץ גרדיאנטים בזרה עיליה. יותר, להקל על בעיית הגרדיאנט הנעלם/המתפוץץ ולאחר מכן זרימת מידע טוביה יותר דרך רשת.

נעשה שימוש נרחב והשיגו שיורי ביצועים שימושיים במשימות שונות של ראיית ResNets-ב מחשב, כגון סיווג תמונה, זיהוי אובייקטים ופילוח סמנטי.

Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



: Neural Architecture Search with Reinforcement Learning (NAS)

Iterate:

- 1) Sample an architecture from search space
- 2) Train the architecture to get a “reward” R corresponding to accuracy
- 3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)

:Smart Compound Scaling

Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.

Search for optimal set of compound scaling factors given a compute budget (target memory & flops).

Scale up using smart heuristic rules

Main takeaways

AlexNet showed that you can use CNNs to train Computer Vision models.

ZFNet, VGG shows that bigger networks work better

GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers

ResNet showed us how to train extremely deep networks

- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to Efficient networks:

- Lots of tiny networks aimed at mobile devices: **MobileNet, ShuffleNet**

Neural Architecture Search can now automate architecture design

:10

Practical Learning in Computer Vision and Image Processing

Supervised Learning

מקבלים קלט מידע ולייבל והמטרה ללמידה פונקציה למפות את המידע ללייבל.

לדוגמה: קלאסיפיקציה, זיהוי אובייקטים....

Unsupervised Learning

מקבלים מידע בלי לייבלים. המטרה – ללמידה איזשהו מבנה של הדטה.

לדוגמה: קולסטרולינג, הורדת ממדים, למידת פיצרים, הערכת דחיסות...

המטרה: ללמידה מבנה כלשהו שחייבי בדטה.

Generative Modeling

בاهינתן מידע לאימון, ניצור מידע חדש מאותה התפלגות.

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

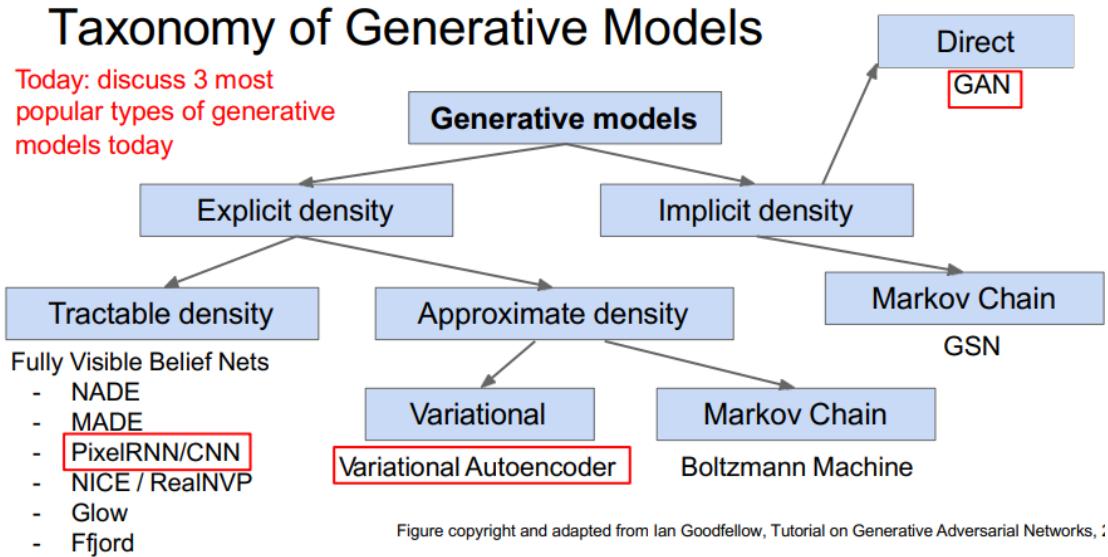


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, :

Fully visible belief network (FVBN)

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

↑ ↑

Likelihood of image x Probability of i 'th pixel value given all previous pixels

Complex distribution over pixel values => Express using a neural network!

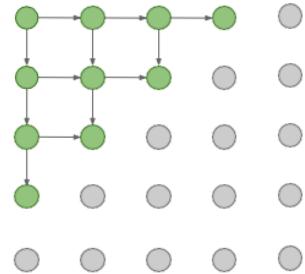
Then maximize likelihood of training data

PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow in both training and inference!

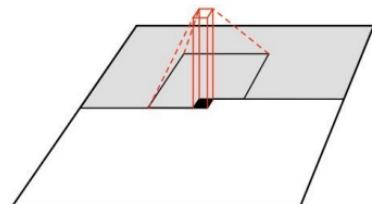


PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)



Generation is still slow:
For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

Figure copyright van der Oord et al., 2016. Reproduced with permission

Pros:

- Can explicitly compute likelihood $p(x)$
- Easy to optimize
- Good samples

Con:

- Sequential generation => slow

So far...

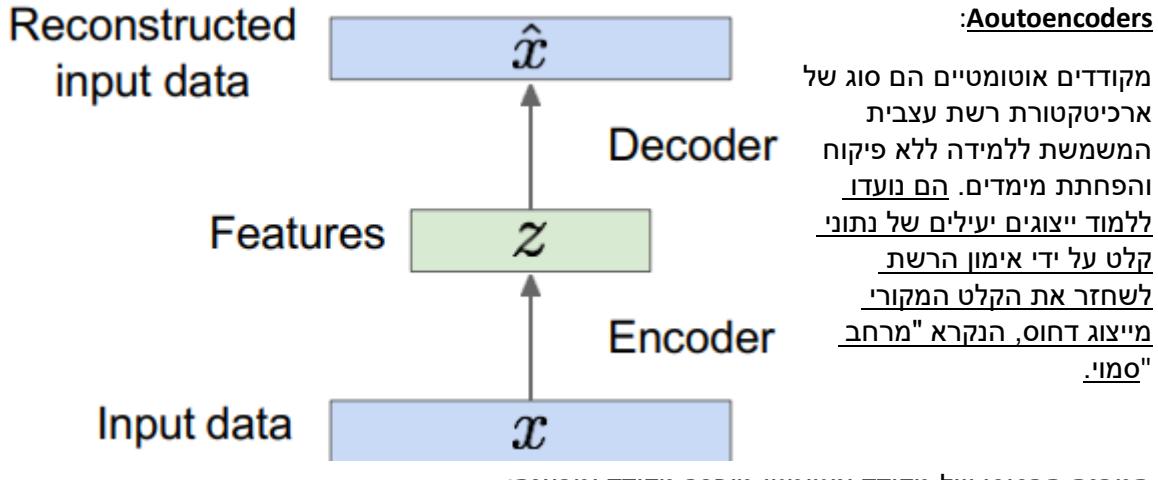
PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent z :

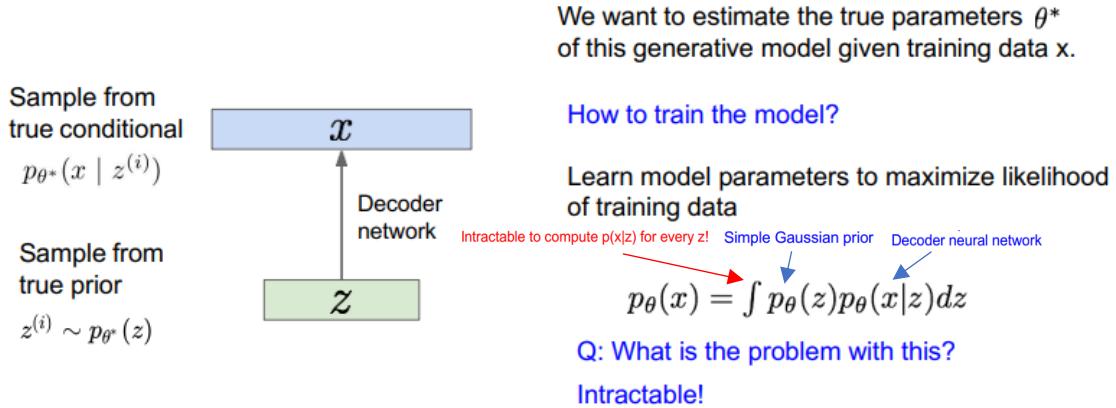
$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Cannot optimize directly, derive and optimize **lower bound** on likelihood instead



1. **מתקודד:** המתקודד לוקח דגימת נתונים קלט ומפענה אותה לייצוג ממד' נמוך יותר במרחב הסמי. בדרך כלל, המתקודד מורכב מכמה שכבות נסתרות המפחיתות בהדרגה את הממדיות של נתונים הקלט, ולוכדות את התכונות החיוניות שלו.
2. **מרחב סמי:** המרחב הסמי הוא ייצוג דחוס של נתונים הקלט המתקבלים מהמתקודד. בדרך כלל יש לו ממדיות נמוכה יותר בהשוואה לנתוני הקלט והוא משמש כగרסה מתקודדת של הקלט.
3. **מפענה:** המפענה לוקח את ייצוג המרחב הסמי ומשחזר ממנו את נתונים הקלט המקוריים. מבנה המפענה סימטרי למתקודד, עם שכבות נסתרות המרחיכות בהדרגה את הממדיות עד שהפלט תואם את מידות הקלט.

: Variational Autoencoders



Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$

Solution: In addition to modeling $p_\theta(x|z)$, learn $q_\phi(z|x)$ that approximates the true posterior $p_\theta(z|x)$.

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational inference is to approximate the unknown posterior distribution from only the observed data x

Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^n p_\theta(x_i|x_1, \dots, x_{i-1})$$

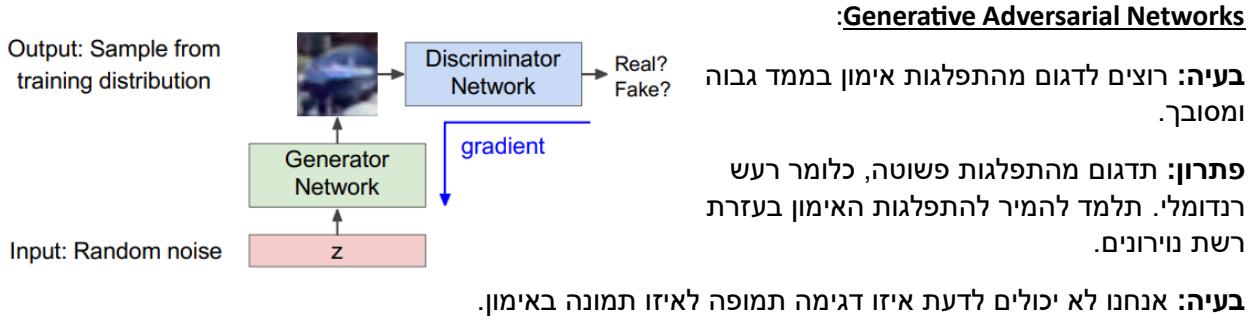
VAEs define intractable density function with latent \mathbf{z} :

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: not modeling any explicit density function!



בעיה: רוצים לדגום מהתפלגות אימון בממד גובה ומסובך.

פתרונות: תדגום מהתפלגות פשוטה, ככלומר רعش רנדומלי. תלמד להמיר להתפלגות האימון בעזרת רשות ניירונים.

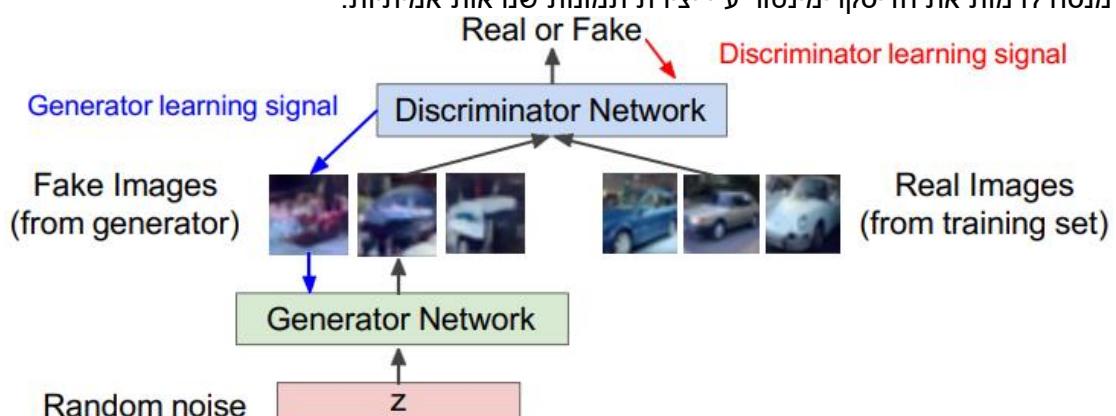
בעיה: אנחנו לא יכולים לדעת איזו דגימה תמומפה לאיזו תמונה באימון.

• Discriminator network

חוות דעתם על החלטה זו

• Generator network

מגנה לרמות את הדיסקריימינטור ע' יצירת תמונות שנראות אמיתיות.



Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \underbrace{\log D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \underbrace{\log(1 - D_{\theta_d}(G_{\theta_g}(z)))}_{\text{Discriminator output for generated fake data } G(z)} \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
 - Generator (θ_g) wants to **minimize objective** such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)

לקיחת הלוגריתם של $(x)D$ ו- $(z)D$ יש השפעה של הפיכת בעיית המינימום המקורית לבעית מקסום עבור המבידיל. על ידי מיקסום $(x)D$ (log, מעודדים את המבחן להקצות הסתברויות גבוהות לנתחים אמיתיים $(x)D$ קרוב ל-1) ונענש כאשר הוא מקצת הסתברויות גבוהות לנתחים שנוצרו $((z)D$ קרוב ל-1)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. Gradient ascent on discriminator

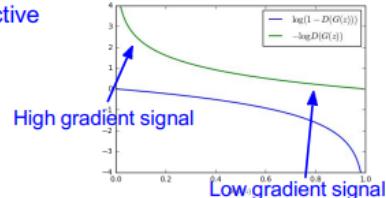
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. Instead: Gradient ascent on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Training GANs: Two-player game

Putting it together: GAN training algorithm

```

for number of training iterations do
    for k steps do
        • Sample minibatch of m noise samples {z(1), ..., z(m)} from noise prior pg(z).
        • Sample minibatch of m examples {x(1), ..., x(m)} from data generating distribution pdata(x).
        • Update the discriminator by ascending its stochastic gradient:
            
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

    end for
    Some find k=1 more stable, others use k > 1, no best rule.
    Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!
    end for
    • Sample minibatch of m noise samples {z(1), ..., z(m)} from noise prior pg(z).
    • Update the generator by ascending its stochastic gradient (improved objective):
        
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

```

Generative Adversarial Nets: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

GANs

Don't work with an explicit density function

Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- Beautiful, state-of-the-art samples!

Cons:

- Trickier / more unstable to train
- Can't solve inference queries such as p(x), p(z|x)

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

לקרא אט מטלה שלוש + ארבע

:Generative vs. Self-supervised Learning

שניהם מכונים למדוד מהמידע בלבד לייבלים.

$$p_{\text{data}}(x)$$

הגנרטיבי מתכוון לממד את התפלגות הדאטה, כלומר ליצור תמונות מציאותיות שיטות ללמידה השגחה עצמית: קלסיפיקציה, ריגרשן... הליבלים של אותן מטלות נוצרים אוטומטית.

:Self-supervised pretext tasks

לדוגמה: ללמידה להשלים תמונה חסירה, לשנות תמונה...

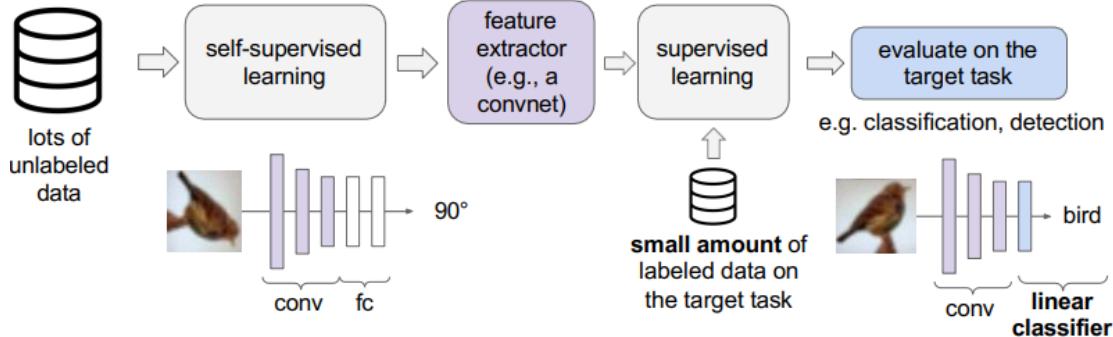
פתרונות אוטם מטלות אפשרות למודל ללמידה פיצרים טובים. אנחנו יכולים ליצור אוטומטית ליבלים עבור מטלות.

הערה: ללמידה דברים ברמת הפיקסלים זה בדרך כלל לא נכון, במקום למד פיצרים סמנטיים ברמה הגבוהה.

איך לחשב שיטת ללמידה השגחה עצמית?

בדרך כלל לא מספיק לנו מהביצועים, כלומר אם המודל לומד לחזות סיבוב תמונה בצורה מושלמת.

נחשב את מקודדי התוכנות הנלמדים במשימות יעד במוריד הזרם.



1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

2. Attach a shallow network on the feature extractor; train the shallow network on the target task with small amount of labeled data

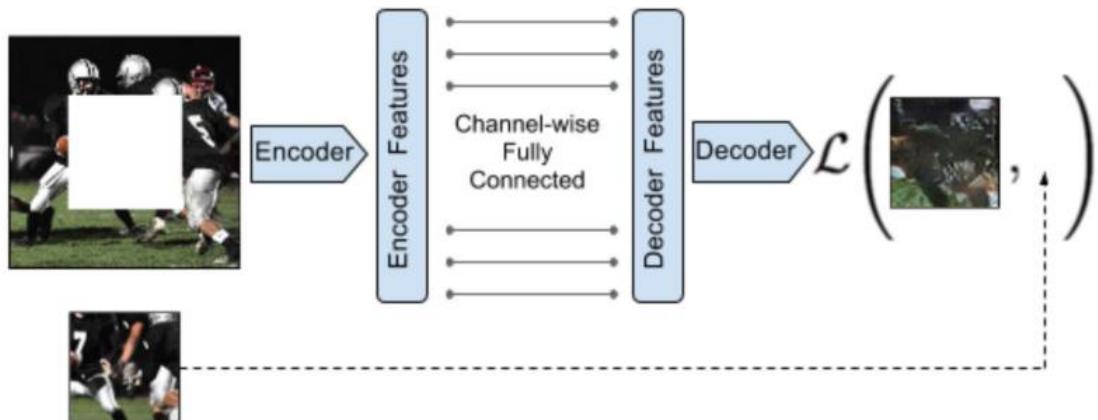
שימושים בתמונה רחבה יותר: מידול שפה, למידה מונחה חיזוקים...

Pretext tasks from image transformations

:predict rotations

השערה: מודל יכול ליזהות את הרוטציה הנקונה של אובייקט רק אם יש לו מושג איך האובייקט אמור להראות מראש.

:Learning to inpaint by reconstruction



Learning to reconstruct the missing pixels

Loss = reconstruction + adversarial learning

$$L(x) = L_{recon}(x) + L_{adv}(x)$$

$$L_{recon}(x) = \|M * (x - F_\theta((1 - M) * x))\|_2^2$$

$$L_{adv} = \max_D \mathbb{E}[\log(D(x))] + \log(1 - D(F((1 - M) * x)))$$

Adversarial loss between “real” images and *inpainted images*

:video coloring

Idea: model the *temporal coherence* of colors in videos

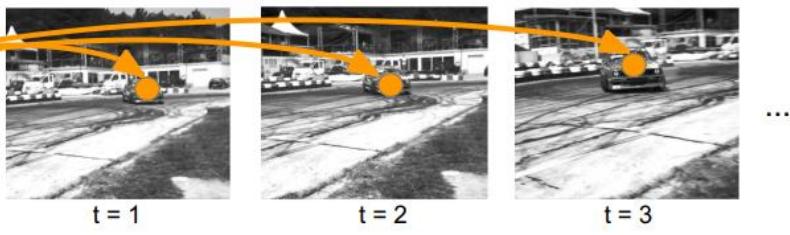
reference frame



t = 0

how should I color these frames?

Should be the same color!



...

Hypothesis: learning to color video frames should allow model to learn to track regions or objects without labels!

מטרת הלמידה: לבסס מיפוי בין רפנסים ופרימיטים חשובים במרחב הפיזרים המלומדים. נשתמש במפה כדי להעתיק את הצבע הנקון.

attention map on the reference frame	predicted color = weighted sum of the reference color	loss between predicted color and ground truth color
$A_{ij} = \frac{\exp(f_i^T f_j)}{\sum_k \exp(f_k^T f_j)}$	$y_j = \sum_i A_{ij} c_i$	$\min_{\theta} \sum_j \mathcal{L}(y_j, c_j)$

לעומת:

התמקדות ב"הגין בריא ויזואלי", כלומר לחזות רוטציות, סידור מחדש, צביעה... המודלים מוקדים בלימוד פיצ'רים טובים על תכונות טבעיות, כלומר תצוגה סמנטית של קטגוריות אובייקט ב כדי לפתור מטלה.

לא מספיק לנו מהביצועים של המטלה, אלא כמה שימושיים הפיצ'רים למטלות המשך(קלאסיפיקציה, דיזה, סגמנטציה).

בעיה: לחושב על מטלות ייחידיות זה מיגע וגם היצוגים יכולים לא להיות כליליים.
למקרה של ייצוגים יכולות להיות קשורה למטלה ספציפית. איך נוכל לעלות מטלה יותר כללית?

:Contrastive Representation Learning

אנחנו רוצים:

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

x: reference sample; x^+ positive sample; x^- negative sample

בהינתן פונקציית ניקוד נבחרת, אנו שואפים ללמידה מוקוד פונקציה אף שמניבת ניקוד גבוהה עבור זוגות חיוביים(איקס, איקס פלוס) וצינויים נמוכים עבור זוגות שליליים(איקס, איקס מינוס).

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[\log \frac{\overline{\exp(s(f(x), f(x^+))}}}{\underbrace{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))}_{\begin{array}{l} \text{score for the} \\ \text{positive pair} \end{array}} \right] \frac{\overline{\exp(s(f(x), f(x_j^-)))}}{\begin{array}{l} \text{score for the N-1} \\ \text{negative pairs} \end{array}}$$

This seems familiar ...

Cross entropy loss for a N-way softmax classifier!

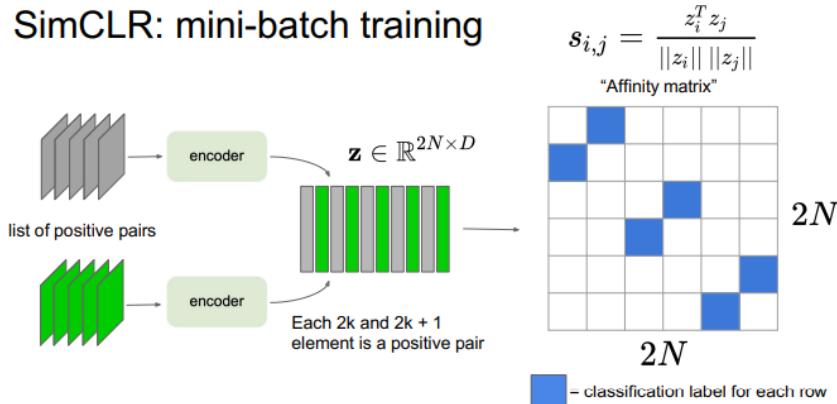
I.e., learn to find the positive sample from the N samples

Non-linear projection head and strong data augmentation are crucial for contrastive learning.

:SimCLR

היא מסגרת למידה בפיקוח עצמו (Simple Contrastive Learning of Representations) (SimCLR) מרטון חנוך צ'ן וויליאם צ'ן Chen et al. בשנת 2020. מטרתו היא ללמד ייצוגים חזותיים רבים עצמאית מנתונים בלבד ששהציגו תווית על ידי מיקסום ההסתכמה בין תוצאות מוגדרות שונות של אותה תמונה.

SimCLR: mini-batch training



SimCLR: a simple framework for contrastive representation learning

- **Key ideas:** non-linear projection head to allow flexible representation learning
- Simple to implement, effective in learning visual representation
- Requires large training batch size to be effective; large memory footprint

הרענון המרכזי מ אחורי הוא ליצור גרסאות מוגברות של תמונה קלט ולעודד את המודול ללמידה ייצוגים שהאינם משתנים להגדלות אלו אף נבדלים עבורה תמונות שונות. זה מושג באמצעות מטרת למידה ניגודית.

מורכבות מהשלבים הבאים:

1. Data Augmentation: Each input image is randomly augmented multiple times to create different views of the same image. Common augmentations include random cropping, color distortion, and Gaussian blur.

2. Encoder Network: An encoder network, typically based on convolutional neural networks (CNNs), maps the augmented images to a latent representation.

3. Projection Head: The latent representations are further transformed by a projection head, which consists of a fully connected layer followed by a normalization operation. The projection head maps the representations to a lower-dimensional feature space.

4. Contrastive Loss: The contrastive loss encourages representations of the same image to be closer in the feature space while pushing away representations of different images. This is done using a contrastive loss function, such as the InfoNCE (Normalized Cross-Entropy) loss, which maximizes agreement between positive pairs (augmented views of the same image) and minimizes agreement between negative pairs (augmented views of different images).

5. Training: The model is trained to optimize the contrastive loss using stochastic gradient descent or other optimization algorithms. The learning process encourages the model to learn semantically meaningful representations that capture relevant visual information.

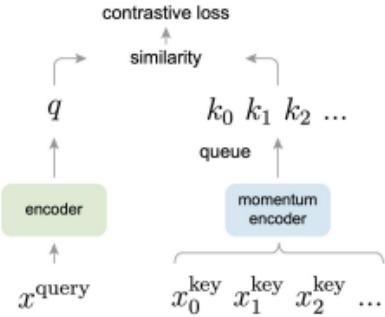
SimCLR has been shown to achieve state-of-the-art performance in self-supervised learning on various vision tasks, even surpassing supervised pretraining in some cases. The learned representations from SimCLR can be transferred to downstream tasks such as image classification, object detection, and segmentation, often with improved performance compared to models trained from scratch.

SimCLR has had a significant impact on the field of self-supervised learning, demonstrating the potential of learning powerful representations from unlabeled data using contrastive learning principles.

:MoCo

MoCo (v1, v2): contrastive learning using momentum sample encoder

- Decouples negative sample size from minibatch size; allows large batch training without TPU
- MoCo-v2 combines the key ideas from SimCLR, i.e., nonlinear projection head, strong data augmentation, with momentum contrastive learning

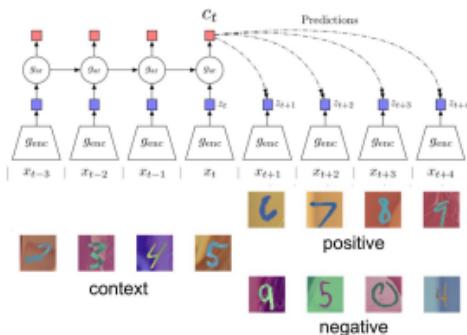


Contrastive Predictive Coding (CPC)

Is a self-supervised learning framework that aims to learn meaningful representations from sequential data. It involves an encoder network to map the input sequence to a latent representation and an autoregressive model to predict future segments. The framework uses a contrastive loss function to encourage the model to learn representations that capture context and exhibit invariance. By optimizing the contrastive loss, CPC enables the learning of representations without the need for labeled data. The learned representations can be transferred to various downstream tasks, improving their performance. CPC has been successful in areas such as speech, text, and video processing, advancing the field of self-supervised learning for sequential data.

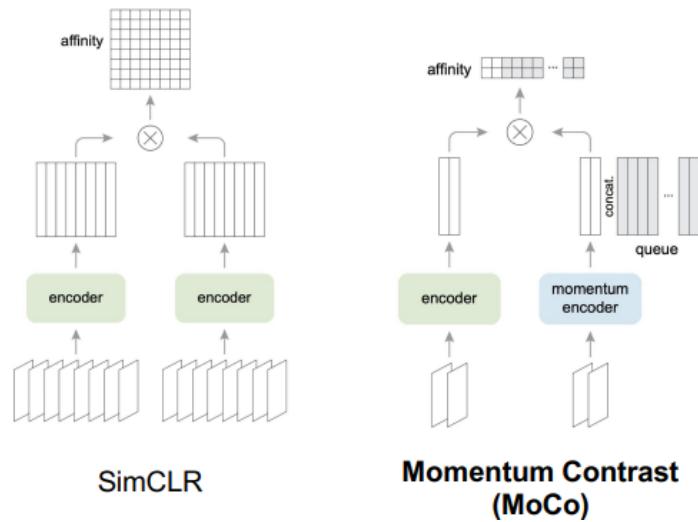
CPC: sequence-level contrastive learning

- Contrast “right” sequence with “wrong” sequence.
- InfoNCE loss with a time-dependent score function.
- Can be applied to a variety of learning problems, but not as effective in learning image representations compared to instance-level methods.



:Self-Supervised Learning

רעיון כללי: נעמיד פנים שיש חלק מהמידע שאנו לא יודעים ונלמד את הרשת מירוניים לחזות אותו.



מצגת 12:

:Generative Pre-trained Transformer(GPT)

מה זה טרנספורמර?

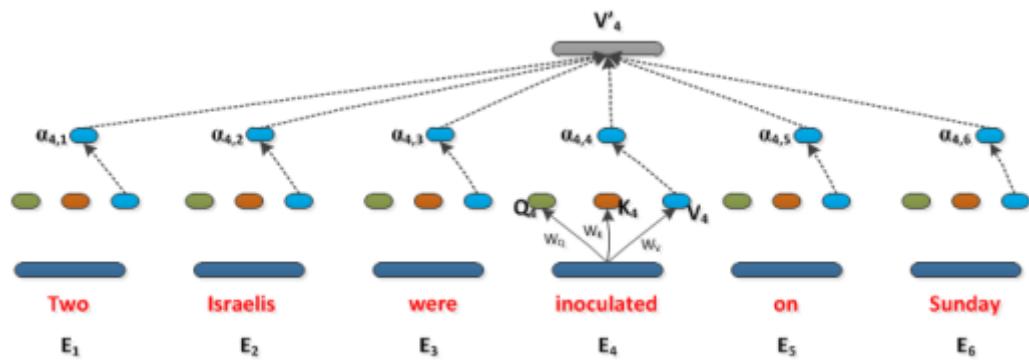
צופה את המילה הבאה בהתבסס לרוב על כל המילים הקודמות.

:Attention

בהתנן אוסף מילים, אנחנו נרצה לחזות את המילה/התרגום הבא.

שלב א: נקודד כל מילה לייצוג שונה כדי שימושיו נverb עם מילים מקודדות.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



$$\alpha_{4,i} = \frac{e^{Q_4 K_i}}{\sum_1^6 e^{Q_4 K_i}} \quad V'_4 = \sum_{i=1}^6 \alpha_{4,i} V_i \quad \alpha_{4,i} = \text{softmax}(Q_4 \cdot K_i)$$

ኒקود גבוי מהטסומטיקס ישמור על החישובות של המילים שהמודל לומד. ניקוד נמור "ויציא" מילים לא רלוונטיות.

Attention is a mechanism in deep learning that allows a model to focus on specific parts of the input or context while processing information. It has gained significant popularity, especially in natural language processing and computer vision tasks, for its ability to capture relevant information and improve model performance.

In the context of neural networks, attention can be seen as a soft alignment or weighting mechanism that assigns importance to different parts of the input. Rather than treating all input elements equally, attention mechanisms enable the model to allocate more resources to the most relevant parts of the input.

The core idea behind attention is to compute attention weights, often through a scoring mechanism, that reflects the importance or relevance of each input element. These weights are then used to compute a weighted sum of the input elements, which becomes an attention-based representation or context vector.

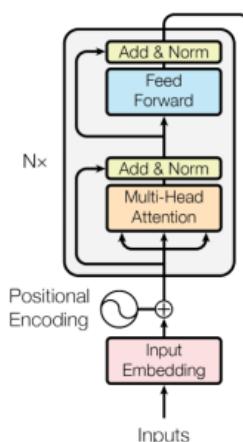
One popular attention mechanism is called the "Transformer" attention, which was introduced in the context of the Transformer model for machine translation. It consists of three main components:

1. Query, Key, and Value: Each input element is associated with query, key, and value vectors. **The query represents the element for which attention weights will be computed, while the keys and values represent the elements that will be attended to.**
2. Attention Scores: **Attention scores are calculated by measuring the similarity or compatibility between the query and the keys.** Common methods for scoring include dot product, scaled dot product, or cosine similarity. The scores are often normalized using a softmax function to obtain attention weights.
3. Weighted Sum: **The attention weights are applied to the values, resulting in a weighted sum that represents the attended context.** This context vector captures the relevant information from the input, emphasizing elements that are more important for the current task or context.

Attention mechanisms have proven to be effective in various applications. In natural language processing, they have greatly improved machine translation, text summarization, question answering, and language modeling tasks. In computer vision, attention has been used for tasks such as image captioning, object detection, and image generation.

Overall, attention mechanisms have revolutionized the field of deep learning by allowing models to dynamically focus on relevant

information. They provide a flexible and powerful way to model dependencies and relationships in data, leading to significant performance improvements in various domains.



איך נוכל להשתמש בזיה לטובת תרגום?

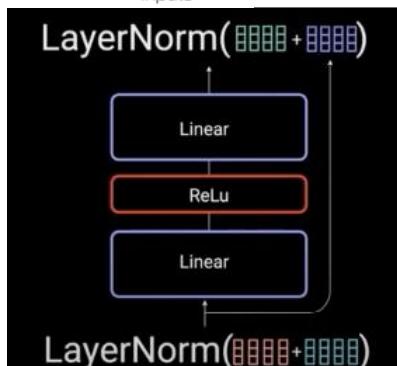
מקרה:

מייצג כל מילה ברצף הקלט כך שהמשמעות מזוקקת וلتוך הייצוג.

מייצג כל מילה במרחב וקטורי בו למרחוק בין מילים יש משמעות.

יהיה לנו וקטור מייצג לכל מילה שאוטו נלמד.

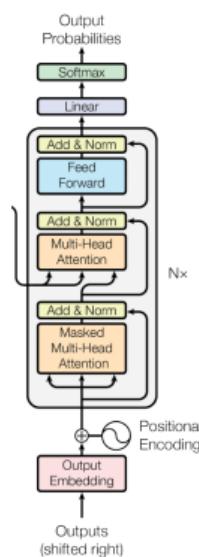
אין חשיבות לסדר ברצף מילים כי אנחנו מתייחסים למילים עצמן אבל סדר יכול לשנות את המשמעות.



Feed Forward

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- FC : $512 \rightarrow 2048 \rightarrow 512$
- Input – $M \times d_{\text{model}}$ (samples X Length)
- Output – $M \times d_{\text{model}}$ (samples X Length)

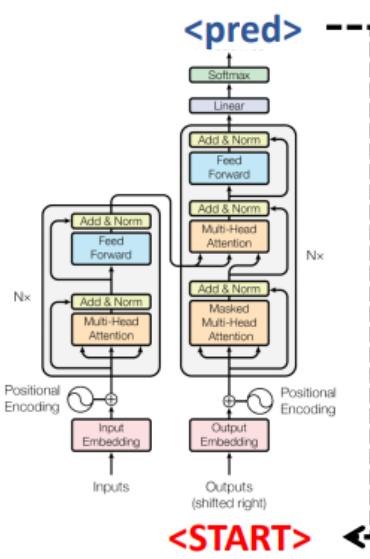


מפענה: מוציא את הייצוג של המילה הבאה בהתבסס על הקודמות.

יש לו את אותה תת שכבה כמו המקרה.

המפענה מנוסה בשכבה ליניארית הפעולה כמסוג, ווופטמאקס כדי לקבל את המילה הסටבריות.

הוא מתחילה בטוקן ולקוח את רשימת הפלטים הקודמים כקלט נוספת לפולט של המקרה שכולל את המידע מהקלט. בכל שלב הוא מוסיף מילה חדשה.



:Multi-Head Attention

דומה להרבה פילטרים.

שרשור והקינה באמצעות שכבה ליניארית.

$$\text{Softmax} \left(\begin{bmatrix} 0.7 & -\inf & -\inf & -\inf \\ 0.1 & 0.6 & -\inf & -\inf \\ 0.1 & 0.3 & 0.6 & -\inf \\ 0.1 & 0.3 & 0.3 & 0.3 \end{bmatrix} \right) = \begin{bmatrix} \text{<start>} & \text{I} & \text{am} & \text{fine} \\ \text{I} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.37 & 0.62 & 0 & 0 \\ 0.26 & 0.31 & 0.43 & 0 \\ 0.21 & 0.26 & 0.26 & 0.26 \end{bmatrix} \\ \text{am} & & & \\ \text{fine} & & & \end{bmatrix}$$

הערה: בגלל שאנחנו רוצים להסתמך על המילים הקודמות והמילה הנוכחית נזכיר את המפענה להתחשב רק בהם ולא במילים הקשורות עתידיות בחישוב.

מצגת 13:

Detection and Segmentation

Semantic Segmentation

רעיון 1: חלוןZZ. בעיה – לא אפשרי לזהות בלי הקשר. איז איז נכון הקשר?

פתרון לא יעיל – זיהוי פיקסל אמצעי עם רשות סיאן אין. בעיה – לא יעיל.

פתרון 2 – קונולוציה. בעיה – הפלט חייב להיות בגודל של הקלט. בנוסף, זה לא יעיל, מאחר ואין שימוש חוזר (רקורסיבי) בפיצרים בין הפיצים.

פתרון 3 – פולי קונולושן. נתכנן את הרשת עם שכבות קונולוציה בלי אופרטורים של הורדה בגודל ב כדי לחזות בשכלי פיקסלים כולם ביחד. **בעיה** – קונולוציה בגודל התמונה המקורית יהיה דבר יקר.

פתרון 4 – פולי קונולושן עם הורדה והחזה לגודל המקורי.

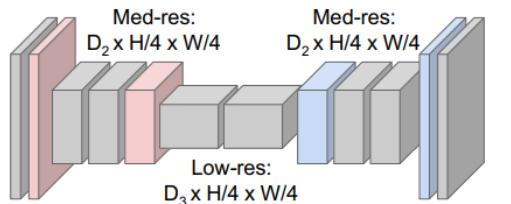
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution



Input:
 $3 \times H \times W$

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



High-res:
 $D_1 \times H/2 \times W/2$

High-res:
 $D_1 \times H/2 \times W/2$

Upsampling:
Unpooling or strided transpose convolution



Predictions:
 $H \times W$

:Unpooling

In-Network upsampling: “Max Unpooling”

Max Pooling

Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4×4



Output: 2×2

Max Unpooling

Use positions from pooling layer

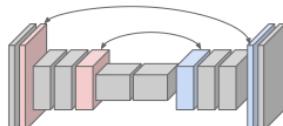
1	2
3	4

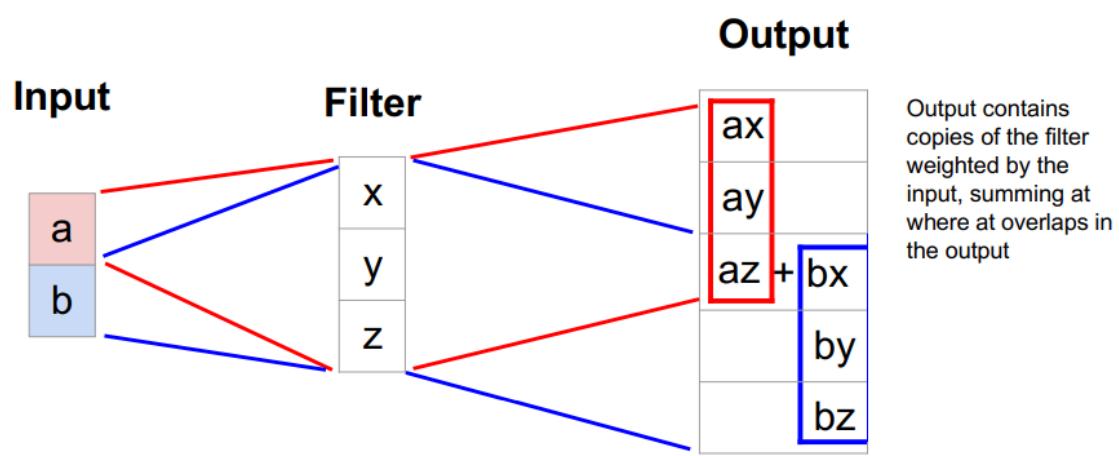
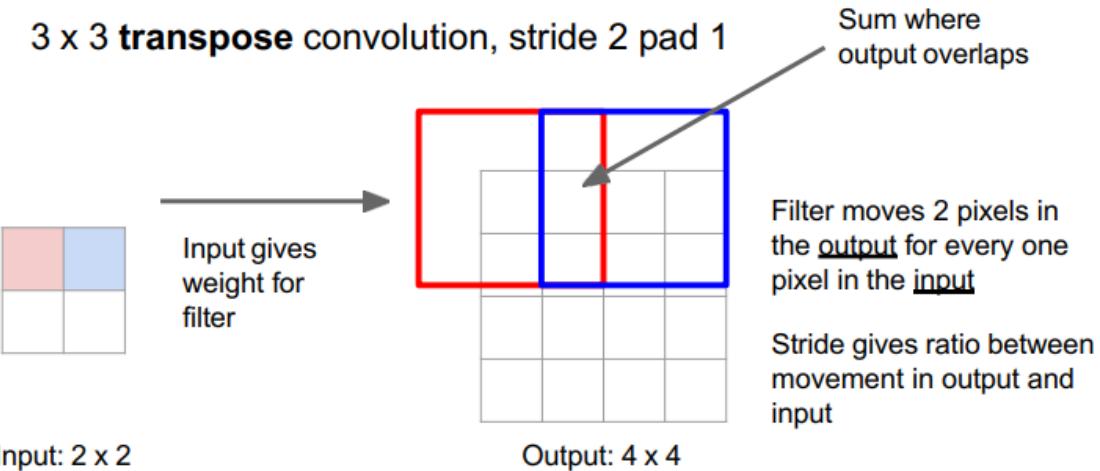
Input: 2×2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4×4

Corresponding pairs of
downsampling and
upsampling layers





:downsampling

ניתן להגדיר את גודל הסטריד בשם זה.

:U-NET

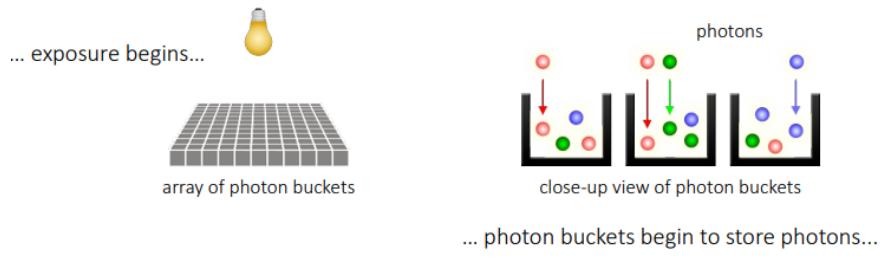
U-Net is a convolutional neural network architecture used for **image segmentation**, particularly in the field of medical imaging. It **consists of an encoder path that extracts features from the input image and a decoder path that upsamples the features to produce the segmentation output. The architecture incorporates skip connections to preserve spatial information and improve segmentation accuracy**. U-Net has been successful in various image segmentation tasks and has become widely **used in medical imaging applications**.

Computer Vision and Image Processing

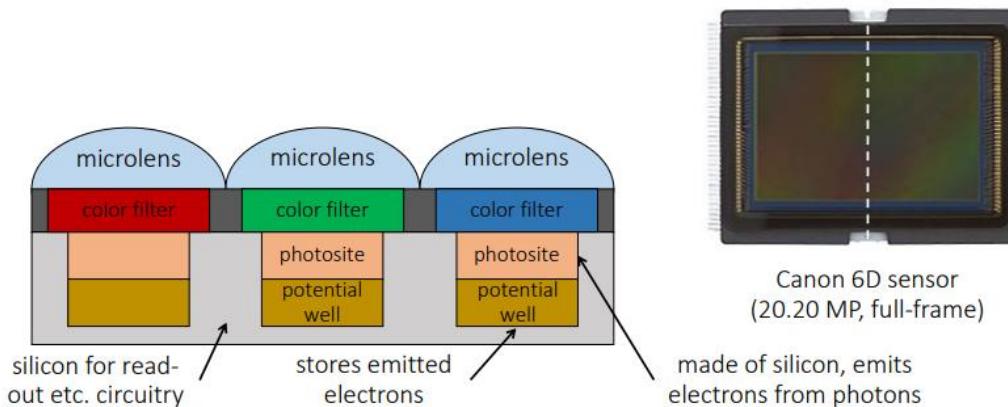
הערה: הערכים של פיקסלים בתמונה והערכים של הפלט מהסנור של המצלמה הם שני דברים נפרדים.

Imaging sensors

When the camera shutter opens...



Basic imaging sensor design



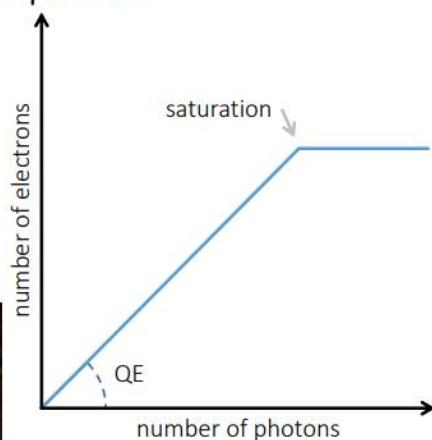
The term "photosite" can be used to refer to both the entire pixel and only the photo-sensitive area.

Photosite response

The photosite response is mostly linear, but:

- non-linear when potential well is saturated (over-exposure)
- non-linear near zero (due to noise)

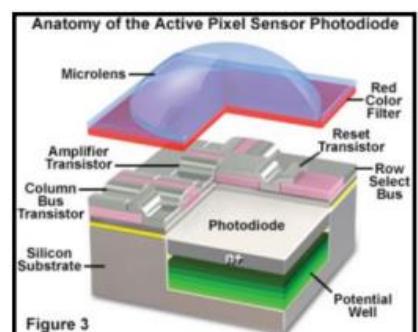
high-dynamic-range imaging deals with these issues.



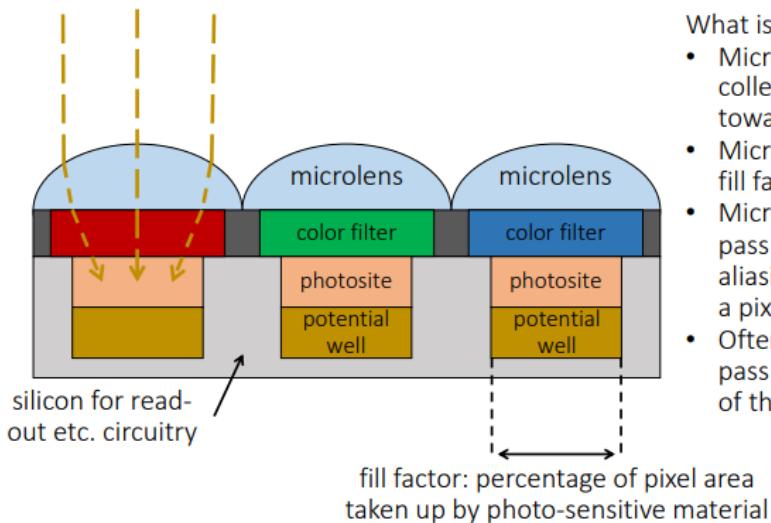
Saturation means that the potential well is full before exposure ends.

How many of the incident photons will the photosite convert into electrons?

$$QE = \frac{\# \text{ electrons}}{\# \text{ photons}}$$



Microlenses (also called lenslets)



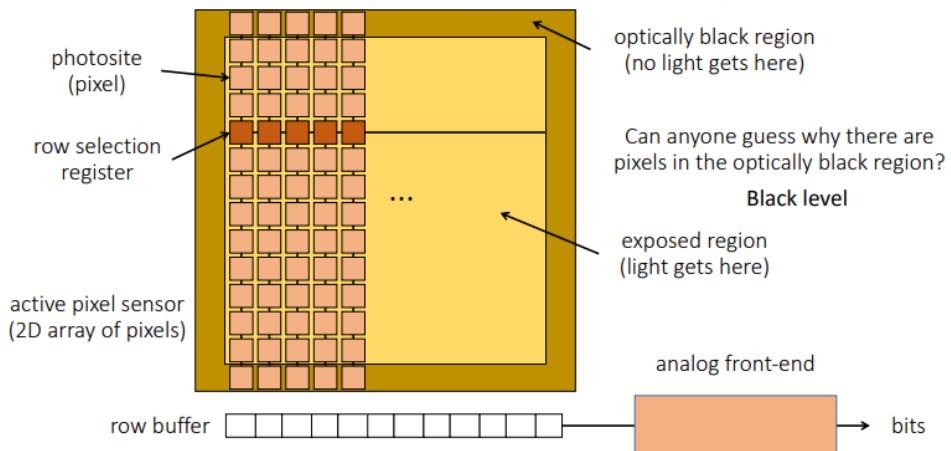
What is the role of the microlenses?

- Microlenses help photosite collect more light by bending rays towards photosensitive pixel area.
- Microlenses increase the *effective* fill factor.
- Microlenses also spatially low-pass filter the image to prevent aliasing artifacts by implementing a pixel-sized 2D rect (box) filter.
- Often an additional optical low-pass filter (OPLF) is placed in front of the sensor to improve prefilter.

עדשות מיקרו עוזרות לפוטוסיט לאסוף יותר אור על ידי כיפוף קרניים לכיוון אזור פיקסלים וрегиש לאור עדשות מיקרו מגדילות את האפקטיביות גורם מילוי.

לנסטורים בדרך כלל יש שכבת זכוכית לפניים שמתפקיד כפילטר נמוך שמנוע אלIASים שנקרה או אף, אבל בגלל אותו פילטר אנחנו מסודדים רזולוציה. ביום בגין הכמות פיקסלים המטורפת הפילטר הזה הפר לא נחוץ.

CMOS sensor (very) simplified layout



Vignetting

מילה מפוצצת לכך שפיקסלים שרחוקים מהמרכז מקבלים פחות אור.

4 סיבות לכך:

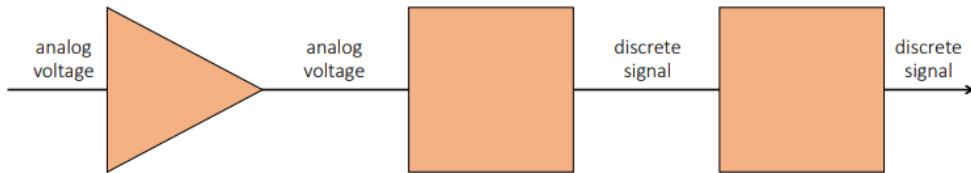
מכני – קרני או רוחמים ע"י פילטר, מכסה או אובייקט כלשהו.

עדשות – קרני או רוחמות ע"י העדשות

טבעי – בגין חוקי הטבע.

פיקסל – רגישות תלויות צוואר.

Analog front-end



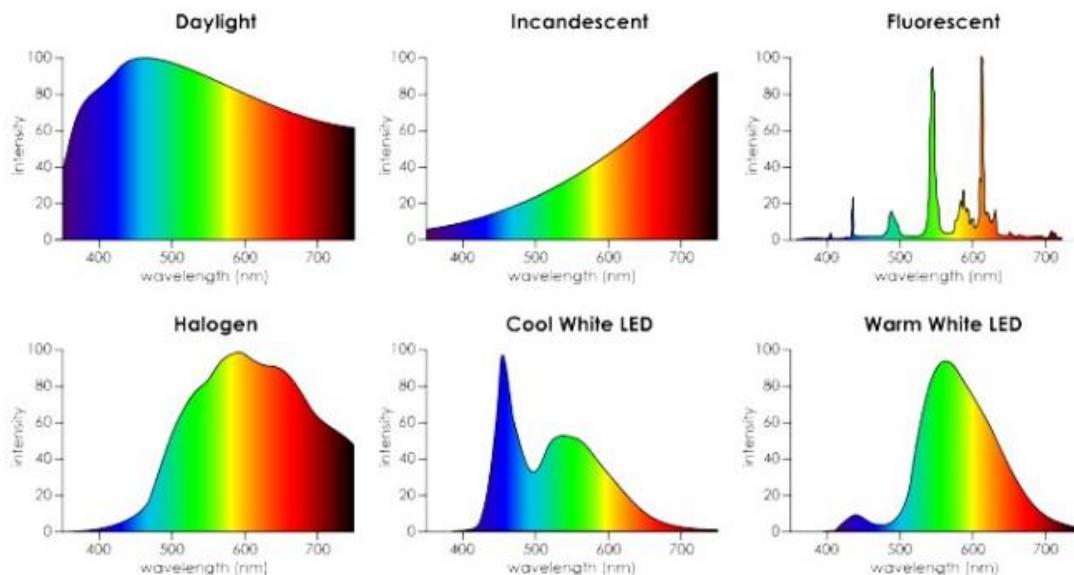
כאשר תריס המצלמה נפתח הסנסור/חישון:

שומר אוטם בברא לאלקטרונים שהציגו אם התאים לא → ממיר פוטונים מקרים לאלקטרונים קורא את הבאר שורה → מבצע זאת עד שהתריס נסגר ואז מבצע אנלוג פרונט אנד → מלאים ממיר אוטם לאותות דיגיטליים → מוסיף איזשהו רוח לאותות אלה → וממיר לאותות אנלוגיים מחזיר את התמונה→ מתקן אי לינאריות→

Spectral Power Distribution (SPD)

רוב סוגי האור מכילים יותר מאורך גל אחד.

אנחנו יכולים לתאר אור בהתאם על ההתפלגות של הכוח על אורך גלים שונים.



Spectral Sensitivity Function (SSF)

לכל חישון יש רגישויות שונות לאור גלים.

$$\text{sensor response} \longrightarrow R = \int_{\lambda} \Phi(\lambda) f(\lambda) d\lambda$$

light SPD sensor SSF

↓ ↓

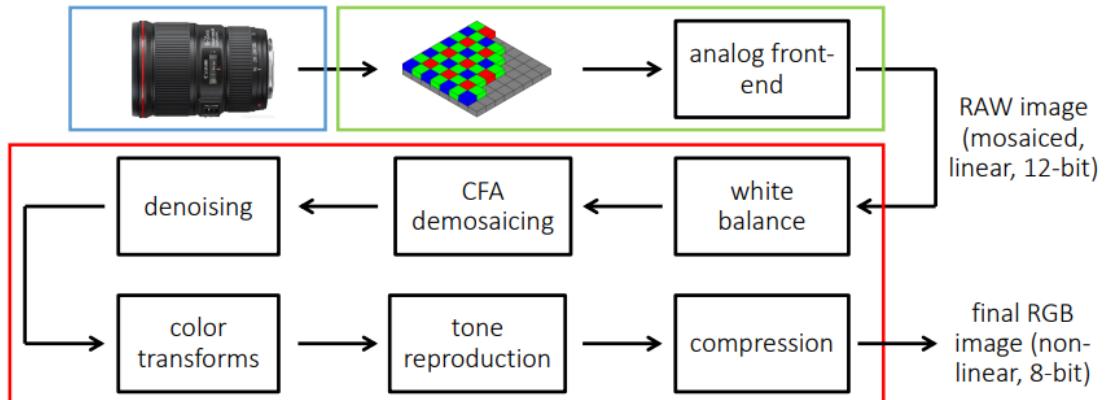
Weighted combination of light's SPD: light contributes more at wavelengths where the sensor has higher sensitivity.

:Color filter arrays (CFA)

כדי למדוד צבע עם חיישן דיגיטלי, נחקה את התאים של האדם במערכת הראייה.
חוותם תדרים שמעל/ מתחת לאורק הגל וככה ניתן להחליף צבעים.

The in-camera image processing pipeline

רצף פעולות עיבוד התמונה המושמות על ידי אות התמונה של המצלמה מעבד (אאי אס פי) להמרת תמונה גולמית לתמונה שגרתית.



:White balancing

התהlixir של הסרת יציקות הצבע כך שצבעים שונים תופסים לבנים מוגדים כלבן בתמונה הסופית.
מצלמות בימינו מגיעות עם מספר רב של הגדרות קבועות מראש: אתה יכול לבחור איזה אוור אתה
מצלמים תמונות מתחתן, והאיזון הלבן המתאים מוחלט.

Manual vs automatic white balancing

Manual white balancing:

- Select a camera preset based on lighting.
- Manually select object in photograph that is color-neutral and use it to normalize.



Automatic white balancing:

- Grey world assumption: force average color of scene to be grey.
- White world assumption: force brightest object in scene to be white.
- Sophisticated histogram-based algorithms (what most modern cameras do).

Automatic white balancing

Grey world assumption:

- Compute per-channel average.
- Normalize each channel by its average.
- Normalize by green channel average.

$$\text{white-balanced RGB} \rightarrow \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} G_{avg}/R_{avg} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & G_{avg}/B_{avg} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \leftarrow \text{sensor RGB}$$

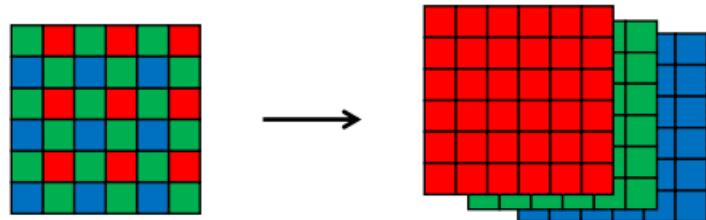
White world assumption:

- Compute per-channel maximum.
- Normalize each channel by its maximum.
- Normalize by green channel maximum.

$$\text{white-balanced RGB} \rightarrow \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} G_{max}/R_{max} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & G_{max}/B_{max} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \leftarrow \text{sensor RGB}$$

CFA demosaicing

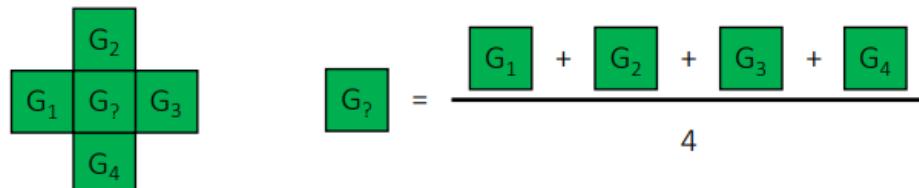
Produce full RGB image from mosaiced sensor output.



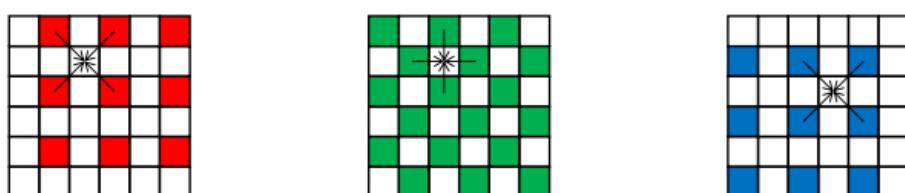
Interpolate from neighbors:

- Bilinear interpolation (needs 4 neighbors).
- Bicubic interpolation (needs more neighbors, may overblur).
- Edge-aware interpolation (more on this later).

Bilinear interpolation: Simply average your 4 neighbors.



Neighborhood changes for different channels:



Noise in images

ישנם 3 סוגי רעשים:

א. רעש צילום(פוטונים)

תדירות הגעה של פוטונים הוא תהליך רנדומלי.

כל שהזירה בהירה יותר, כך גדלה השונות בתפלגות.

ב. רעש צילום שחור

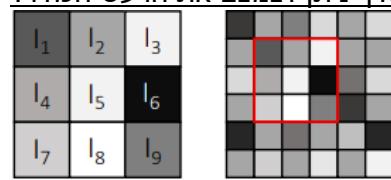
אלקטרונים הנפלטים עקב פעילות תרמית (מחמיר ככל שהחישון מתחמם).

ג. רעש קריאה

רעש קריאה מתיחס לרעש החשמלי, הלא רצוי המוכנס במהלך תהליכי קריאה האות מחישן תמונה במערכת ההדמיה (AFE) או גלי. זה נגרם בעיקר על ידי האלקטרוניקה הקריאה והאנלוגית הקדמית.

איך ניתן למצמצם את הרעש הכללי?

Look at the neighborhood around you.



- Mean filtering (take average):

$$I'_5 = \frac{I_1 + I_2 + I_3 + I_4 + I_5 + I_6 + I_7 + I_8 + I_9}{9}$$

- Median filtering (take median):

$$I'_5 = \text{median}(I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9)$$

Large area of research.

:Gamma encoding curves

עקומת קידוד הגמא המדעית תלוי בצלמה.

מאחר והעינ רגישה לטווחים מסוימים יש הבדלים בין הצבע בפועל לבין איך שהצבע מתרגם.

העינ יותר רגישה לאזורים כהים.

ازהרה: אחרי, הערכים שלנו כבר אינם לינאריים ביחס לזוהר הסצנה!

:RAW

מתיחס לפורמט קובץ המשמש בצלום דיגיטלי לאחסן נתונים תמונה מעובדים ובלתי דחוסים ישירות מחישן התמונה של המצלמה.

האם אני אי פעם צריך להשתמש?

כן natürlich!

בכל פעם שאתה משתמש באלגוריתם ראייה ממוחשבת מבוסס פיזיקה, אתה צריך מדידות של זוהר לינארי.

דוגמאות: סטריאו פוטומטר, צורה מהצללה, תאורה חדש מובוסת תמונה, תאורה...
ישום האלגוריתמים על תמונות לא ליניאריות.

אם לא מספיק לי ראייה מובוסת פיזיקה?

עדין תרצה(לא חובה) להשתמש בו.

עשה את החיצים שלך RAW, אם אתה אוהב לסייע מחדש את התמונות שלך (למשל, בפוטושופ)
הרבה יותר קלים והעריכות שלך הרבה יותר גמישות.

אם יש חסכנות לשימוש בו?

קצת תמונות הרבה יותר גדולים.

א. אתה עלול לשורף כמה כרטיסי זכרון.

ב. המצלמה עלולה לשמר נתונים בבארור בתדריות גבוהה יותר מאשר צילום במצב רציף.
ג. למחשב צריך להיות מספיק זכרון כדי לעבוד תמונות כאלה.

אם בכלל יש גישה לקבצים כאלה?

לרוב כן!

רוב המצלמות מאפשרות שבירת קובץ בצורה זו.

חלק מהפתרונותים מאפשרים גישה גם.

אם לא כוונתי את המצלמה לנע לא ניתן לקבל קובץ נع.

דוגמאות

```
# Define Generator
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.net = nn.Sequential(
            nn.ConvTranspose2d(latent_dim, 128, kernel_size=7, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 1, kernel_size=4, stride=2, padding=1, bias=False),
            nn.Tanh()
        )

    def forward(self, x):
        x = x.view(x.size(0), [latent_dim], 1, 1)
        x = self.net(x)
        return x
```

```
# Define Discriminator
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=4, stride=2, padding=1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 1, kernel_size=3, stride=1, padding=0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.net(x)
        x = x.view(x.size(0), -1)
        return x
```