Lecture 1. Intro

Questions (mine) to the lecture (to repeat; cover the notes below and answer this questions):

1.

Notes, by slides (numbers):

3. CV is
    - Building systems that process, perceive, reason about visual data.
4. CV important?
    - Everywhere. Instagram. Photos. Smartphones.
    - Massive. We don't have human power to process this data.
    - Will continue to improve.
5. Learning is 'learning from data and experience'.
6. DL. Hierarchical learning algos with many layers.
7. DL subset of ML. Intercects with CV. ML is subset of AI. CV and ML is this class focus.
8. History. Hubel and Wiesel. 1959. Wanted to understand how brain work.

Lecture 2. Image classification (I.C.).

Notes. By slides:

11. Image classification is by image getting ordered labels. Computer doesn't have our intuition about pictures. It sees giant number of bits, grid. 3 channels of RGB by 2 dim.

13-19. Challenge is variations:

```
- Viewpoint. Camera moves.
- Intraclass variation. Different cats.
- Fine-grained categories. Breeds of cats.
- Background clutter. Snow. At sand.
- Illumination changes. In dark. In light.
- Deformation. Poses.
- Occlusion. Hidden by things. A tail of cat from a cusion of a couch.
```

20. I.C. very useful: medical, whale recognition, galaxy classfication, etc. See Kaggle Challenge.

21-28. Helpful for other tasks in CV. - Object Detection: boxes around things. A way is to classify by frames of the image (moving box). - Image captioning. Describe an image. Sequence of labels. What word should I say next. - Play GO. Where to play next? (Why?) Input: position. Output: which position should I play.

29-31. How I.C.? Not obvious. Given im. Out: class.

30. Try fidning edges? They find corners? But how next? But because of variations it doesn't work.
31. ML, Data-Driven approach. Pipeline:
    A. Collect images and labels.
    B. Use ML to train classifier. Train(..)
    C. Evaluate on new images. Predict(..)

32-40. Data sources for it? - MNIST. 50k images. 28x28. Not enough - CIFAR10. 50k images. 10 classes. 32x32 rgb. - CIFAR100. 100 classes. 50k images. 32x32 rgb. 154M pixels. - ImageNet. Gold standard. ~1.3M. Top 5 accuracy: if one of 5 is correct then good. 251B pixels - MIT Places. ~8M training images. Places365. 1.6T pixels. - Omniglot dataset. Learn from little data. 1623 categories: char from 50 diff alphabets. 20 images per cat.

41-.. Nearest Neighbor.

41. Naive approach:
    - Train() : memorize all data.
    - Predict(): Compare with all set and return the label of the most similar image.
    - Need func to compare two images.
        - L1 (Manhattan) dist. Sum(Abs(img_1 - img_2))
    - Training time? O(1)
    - Testing time? O(N)
    - Bad timing.
    - But there are optimizations. facebookresearch/faiss
    - Results: not so smart. 4 out of 10 are correct for cifar10.

54-59. Dicision Boundaries. How to view this in another way? - Representation: - Picture: 2 axes, colored regions, dots there. - Axes - this is two pixels from one image. - Colored regious - learned from training set. - Dots - examples. - Boundaries between regious. - Problems: a small region inside another region; jagged or noisy boundary.

61. How to improve decison boundaries? Instead of taking from 1 nearest nebour take the color of a regious from K nearest neighbours (majority decides). Advantages? With k = 3. Smoother. Less noice on boundaries. But: now some unclassified regions.

62. Diff distance metric. Use L2 Euclidean distance. sqrt(sum((img_1 - img_2)^2))

- Choosing L1 or L2 is hyperparam. L2 might be more intuitive.
    - K-Nearest Metric can be used elsewhere. Can write our own distance metric.
- Pdf files. Uses tf-idf similarity. For freq. of words.

70. Demo for KNN.

71-79. Hyperparams: - Best val for K? Best distance metric? How? Try diff and decide

- Ideas to choose:
    - On all dataset? Bad as on new data doesn't work. Always K=1 but on new
    - Train and test? Choose on test data. Bad as learn on test data!
    - Train, validation, test. Better! Touch test data only once. Example: months work on train and validation sets and only before a week only then on the test set. This is right.
    - Cross-validation. Best idea. Split non-test data into chunks, pick up one (iteratively or randomly) for validation and others for training. Better but expensive.
- Cross validation example. Graph with accuracy for diff K.

80. Universal Approximation. If K -> infinity? If training samples to infinity? Then it represents any function (but technical issues).

- But there is a curse! Curse of dimensionality. Number of training points grows exponentially! But internet is big? No! Consider *3232 images: 2^(3232)* ~= 10^308. And particles of visible uni: 10^97.

88. KNN very rarely used.
    - Very slow at test time.
    - Hard to collect data.
    - Distance metric not informative if on raw pixels.
    - But NN works on ConvNet features ! Not on raw pixels.
        - That's a strong baseline.
        - Image captioning work with NN.

Questions

11. What's image classification (I.C) problem? 13-19. What challenges in I.C.?

12. Where I.C. used? 21-28. To what ML problems I.C. transfers?

13. Basic code template of I.C.?

    AD. Naive approach?
    AE. Main methods? 32-40. Data sources for I.C.? 41-88. NN (nearest neigbour).
14. Naive approach. Why bad?

15. Decision boundaries representation? What are colors?

    BG. Problems if K = 1?
16. How to improve?

    - Nearest K neighbors? Advantages?
17. Different distance metrics? L1? L2?

    BO. What other domain specific distance metrics?
18. How to choose hyperparameters? Name ways. And their cons. How to divide data?

19. Cross validation. Pros and cons?

20. If K -> infinitiy? Universal approximation? What's curse of dimensionality?

21. Is KNN used anywhere? On raw pixels? With other features (not raw pixels, features from ConvNet)?

Lecture 3. Linear Classifiers. Softmax / SVM classifiers. L2 regularization.

2022-07-12

/* Previously (mit 6.036): Loss func hinge = max(0, 1 - v), v = y_i*(theta.T @ X_i) Now: Loss func hinge = max(0, s_j - s_(y_i) + 1), v = y_i*(theta.T @ X_i) */

Notes, by slides:

4. Recap. CV challenges. Domain-Driven approach. kNN.

5. NNs cosists of very similar things as Linear classifiers (LC).

6. LC idea: parametric approach

7. For Cifar10.

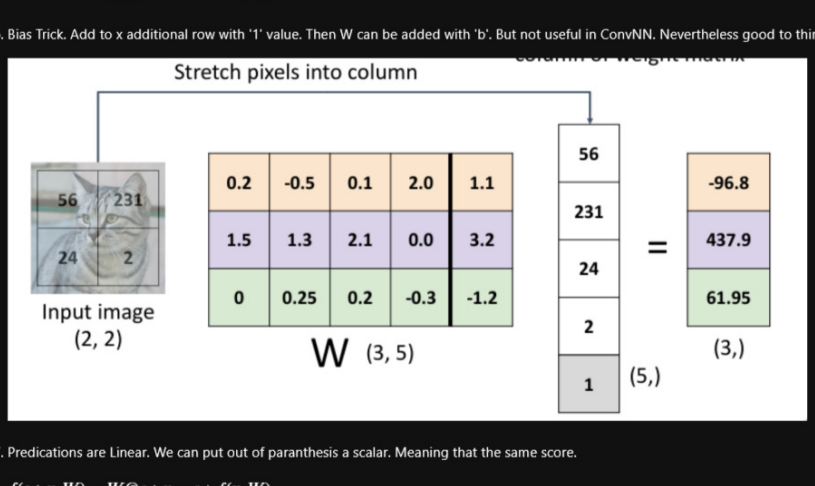
    $f(x, W) = Wx + b$

    In: image x With: W - parameters or weights Output: 10 class scores

8. Example. Cat image. x - is flattened image (2x2 into 4x1).

```
    W   @ x  +  b   =  f
  (3, 4)  (4,)   (3,)    (3,)
```

16. Bias Trick. Add to x additional row with '1' value. Then W can be added with 'b'. But not useful in ConvNN. Nevertheless good to think about it in geometric way.



Stretch pixels into column
56
231
24
2
Input image
(2, 2)
W (3, 5)
1
(5,)
=
(3,)

0.2	-0.5	0.1	2.0	1.1
1.5	1.3	2.1	0.0	3.2
0	0.25	0.2	-0.3	-1.2

-96.8
437.9
61.95

17. Predications are Linear. We can put out of paranthesis a scalar. Meaning that the same score.

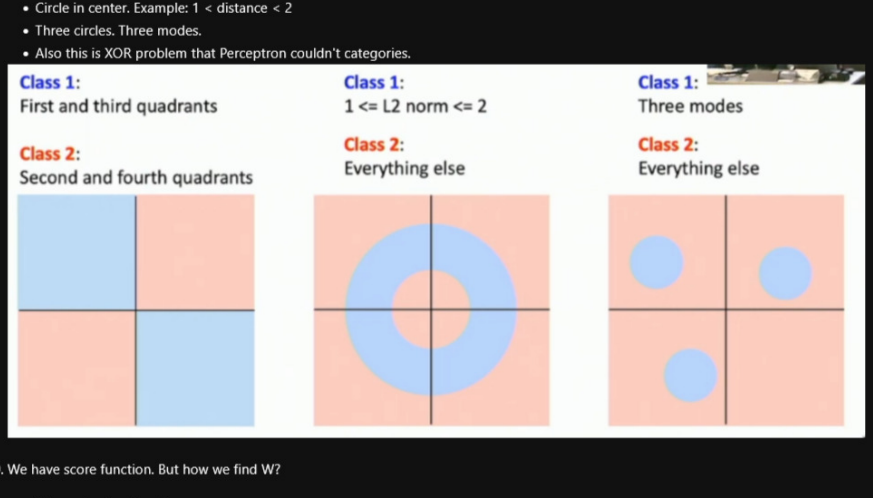
    $f(c * x, W) = W@c * x = c * f(x, W)$

    Example: is graying an image with 0.5. Then the same scores but scaled.

18. Interpreting a LC.

    S. Algebraic Viewpoint. This is when W in rows of hypothesis for each class.
    T. Visual Viewpoint. Shape rows the same as image. Then if we show this image then it shows a 'template' per category. And to get score use inner (dot) product for those 2 matrixes plus the b_i (bias).
        - Background failure. It turns out that it uses background so it might be more or less probably for the score.
        - Modes failure. Like horses looking in diff directions. Or car of diff colors.
    U. Geometric Viewpoint. Example for images: score is vertical axis, pixel value is horizontal value. Then the line shows a classifier.
        - Extends to very high dimensiona Euclidian space.
        - But in many dimensions it has flaws, not intuitive.
19. Hard Cases for LC. Examples:

    - 4 squares:
    - Circle in center. Example: 1 < distance < 2
    - Three circles. Three modes.
    - Also this is XOR problem that Perceptron couldn't categories.



Class 1:
First and third quadrants
Class 2:
Second and fourth quadrants
Class 1:
1 <= L2 norm <= 2
Class 2:
Everything else
Class 1:
Three modes
Class 2:
Everything else

20. We have score function. But how we find W?

    - We need a loss function
    - And algo that optimizes the loss function.
21. Loss func aka objective func or cost func. It is how good our classifier. Low loss - good. High loss - bad.

    - Formally:

    $(x_i, y_i), i = 1..N$

    x_i - image

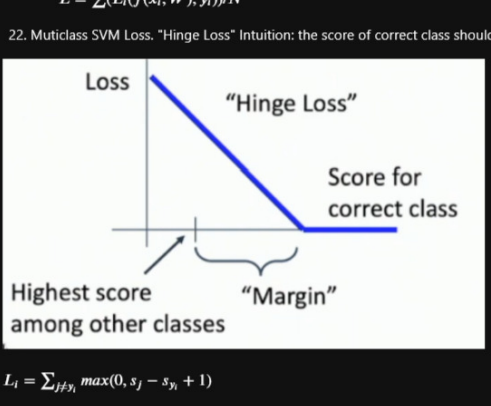
    y_i - label

    $L_i(f(x_i, W), y_i)$ for single example.

    For entire dataset:

    $L = \sum(L_i(f(x_i, W), y_i))/N$

22. Muticlass SVM Loss. "Hinge Loss" Intuition: the score of correct class should be higher, i.e. less loss.



$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$

where $s = f(x_i, W)$

SVM Loss 'wants' the score for correct class be higher than scores for other classes by some marging (δ). So that's why it is 0 when $s_j - s_{y_i}$ is negative, i.e. $s_j < s_{y_i}$

50. More about SVM loss.

- What if we change a score a little bit for correctly classified example? A: Will be 0. Compare with other loss func.
- What's max or min possible values? A: Max - inf. Min - 0.
- If scores all random but examples are standardized, then what loss to expect? A: Then we might see small loss function.
- What if sum were over all categories (including the correct class)? A: The loss will be moved by one (delta), inflated by one.
- What if the loss used a mean instead of a sum? A: The preference will be the same. Will not effect predictions.
- What if we square the loss? $L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)^2$ A: Then it will be another classifier because weights will be different.
- If we found loss that sum up to zero will it be unique? A: No. Consider if we multiply weights by 2.

So if 2W and W are with same loss how do we find the better hypothesis? Add regularization.

60. Regularization.

- L2 regularization. Multiply every element with every other element in W.
- L1.
- Other.

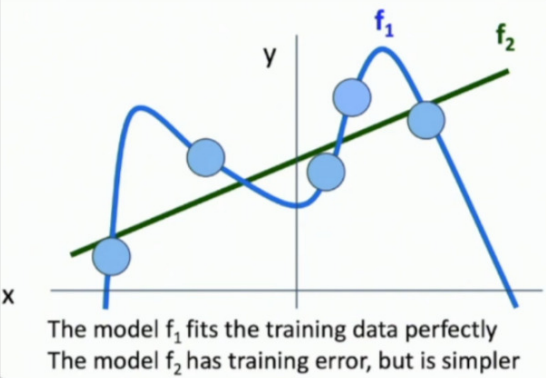
Purpose or Regula:

- Express preferences in those models that have the same loss.
- Avoid overfitting.
- Improve optimization.

64. Regularization: Expressing Preferences

Example. For $R(W) = \sum_k \sum_l W_{k,l}^2$

66. Regul. Prefer Simpler Models. Avoid overfitting.



69. Regul. Very important. Always need to add to model.

70. Another Loss func. Cross-Entropy Loss (Multinomial Logistic Regression)

Let's interpret those scores as *probabilities*.

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \quad \text{or equivalently} \quad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

See Information Theory to understand this (Cross Entropy, other).

- What CEL does? A: Maximizes probability of correct class.
- What's min/max possible loss L_i? A: Min - 0, Max - infinity. But with SVM it was possible to archive 0, but with CEL it is not practical way to get 0.
- What if all scores random? A: $-log(|Y|) = -log(10) = 2.3$. $|Y|$ - number of categories.
- What if small change? A: SVM stays the same. Cross-entropy will change.

Questions:

6. What's idea of linear classifier?
7. Formula for LC?
8. Bias trick (b vector)
9. What if we scale images? Linear?
10. How can we interpret LC? Name ways. How visual viewpoint is done?
11. Hard cases for LC? Compare with Perceptron?
12. What we need to find weights now as we have the loss func?
13. Idea for the Loss func?
14. Describe Multiclass Loss func (Hinge Loss)
    - What if we change a score a little bit for correctly classified example?
    - What's max or min possible values?
    - If scores all random but examples are standardized, then what loss to expect?
    - What if sum were over all categories (including the correct class)?
    - What if the loss used a mean instead of a sum?
    - What if we square the loss? $L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)^2$
    - If we found loss that sum up to zero will it be unique?
15. What's regularization? What kinds of it there are?
16. What purposes for the regularization? Name three
17. Cross-Entropy loss func.
    - Describe.
    - What CEL does?
    - What's min/max possible loss L_i?
    - What if all scores random?
    - What if small change?

[ ]:

Lecture 4.

Optimization. Stochastic Gradient Descent. Momentum, AdaGrad, Adam. Second-order optimizers .

9. Recap.

- Score func
- Loss function

How to find best w ? That is $w^* = argmin_w L(w)$ This is optimization problem. Let's focus only on essentials.

11. Intuition is walking on a plane and seeking a bottom of this. The peak is the highest loss.

- Sometime can find solution directly. But usually need a gradual approach.

13. Idea #1. Random Search.

- Generate random weights.
- Keep the best one
- Iterate.
- Accuracy for this is ~15.5%. Not bad thought. SOTA is 95%.

16. Idea #2. Follow the scope. Strategy is step in the direction of greatest decrease.

- In 1-dimension. It is $df(x)/dx$
- For multi-dim, it's the gradient.
- The greatest decrease is the opposite to the greatest increase. Inutitive.

19. How to compute gradient?

- Numerical gradient. That is increase a bit on dim of W then get diff for both f of those two w, div by this small bit and that tells value of gradient:

$$dL/dw = \begin{bmatrix} \frac{f(w) - f(w + [w_0 + 0.001, ...])}{0.001} \\ ... \end{bmatrix}$$

Problem with numerical grad?

- Slow: O(#dim). Inpractical in many dim.
- Approximate. Not accurate.

27. Analytic grad. So why not calc gradient? So we could get grad in one operation. It relies on backprop (Lecture 6). Exact, fast, error-prone.

In practice. Analytic grad needs a gradient check. So do it it with a numereric gradient. In assignments: `grad_check_sparse`. In PyTorch: `gradcheck` (and `gradgradcheck` for second derivatives).

35. Gradient Descent.

```
w = init_w()
for t in range(num_steps):
    dw = comp_gradient(loss_fn, data, w)
    w -= learning_rate * dw
```

Hyperparams. Weights - matter much, there is a proper way. Num of steps - common is to run for a fixed num of iterations, constrained by resourses. Learning rate - how we trust the grad, how we want to take this grad, can be variable. Those we will learn through out the lectures.

36. Representation. Heat map. Demos.

GD demo on heat map.

- Does not go straight to the bottom. Kind of curves around the optima. Step size becomes slower as it approaches min as the grad is less there.

Batch Gradient Descent. For full loss func GD is very expensive. Full loss func is sum div by num of examples.

38. Stochastic GD (SGD).

Approximate the sum using minibatches (32 / 64 / 128). Practically almost always used. Now we add batch size (use bigger as u can, not matter too much) and data sampling (can shuffle in beginning and then again; not matter too much) to our hyperparams.

40. Why stochastic?

$$L(W) = \mathbb{E}_{(x,y)\sim p_{data}}[L(x, y, W)] + \lambda R(W) \approx \frac{1}{N}\sum_{i=1}^{N} L(x_i, y_i, W) + \lambda R(W)$$

Because now our func is probabilistic.

42. Problems with SGD

If loss changes quickly in one dir but slowly in another? Slow progress along the shallow dim because it is quick in steep dir.

- 45. Another prob. Local minimum or saddle point.

In local min we might stop. And won't escape from it. In saddle point, at the tip of it the grad also the 0, so again stuck. In high dim, saddle points are common.

- 47. Third problem. It is stochastic, it is random. Might be avg towards optimum but not clear. So we need to handle this noise.

49. SGD + Momentum. To handle noise.

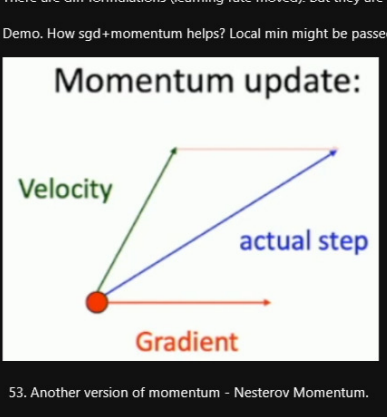
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Idea is to track of a velocity. Imaging a ball rolling down a hill. So at every point we have this velocity vector and the gradient. We update the velocity. And use this velocity to update weights. `rho` - friction or decay rate.

```
v = < zero vector >
for t in range(num_steps):
    v = rho * v + dw
    w -= learning_rate * v
```

There are diff formulations (learning rate moved). But they are similar.

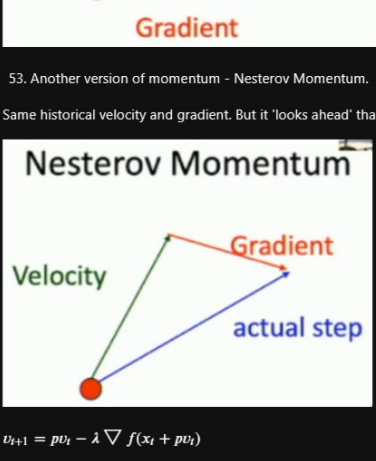
Demo. How sgd+momentum helps? Local min might be passed by using a velocity. Similar for saddle points. Also helps with going faster to an optimum. Also it smoothes noise and thus moves faster to center.



Momentum update:
Velocity
actual step
Gradient

53. Another version of momentum - Nesterov Momentum.

Same historical velocity and gradient. But it 'looks ahead' that is computes gradient for the point where the velocity will take us and then summing this gradient.



Nesterov Momentum
Velocity
Gradient
actual step

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

It is akward for us as it uses future point. And it adds velocity vector instead of subtracting it (as it is with the gradient). So it can be rewritten to allow this.

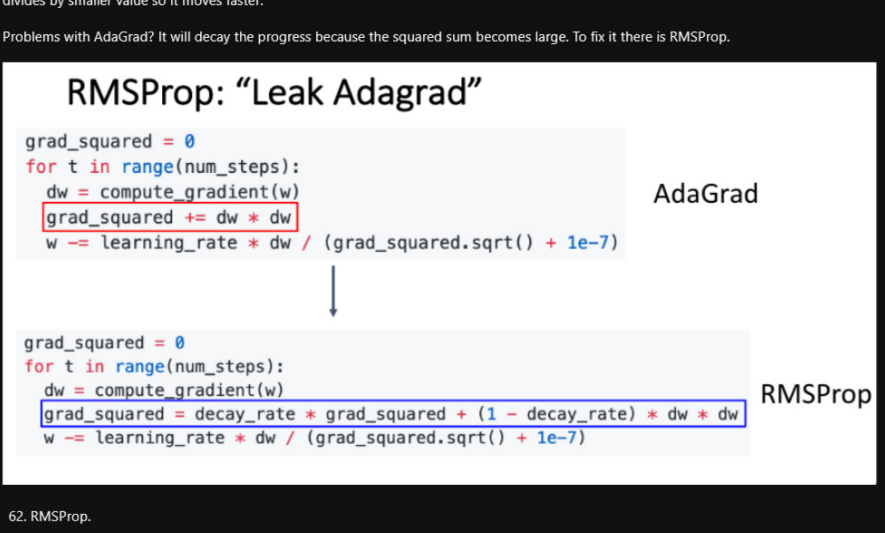
- 57. Demo. We can see that SGD+Mom and Nesterov overshoot at the bottom going a bit further.

58. AdaGrad.

Adaptive learning rates class of optimizations.

Now it keeps tracking squared gradient and when updating the weights it divides it by square root. Why? It solves the issue when grad changed very fast adagrad will divide by large num so it overcomes this ill conditions. And when grad is small it divides by smaller value so it moves faster.

Problems with AdaGrad? It will decay the progress because the squared sum becomes large. To fix it there is RMSProp.



RMSProp: "Leak Adagrad"

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared += dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

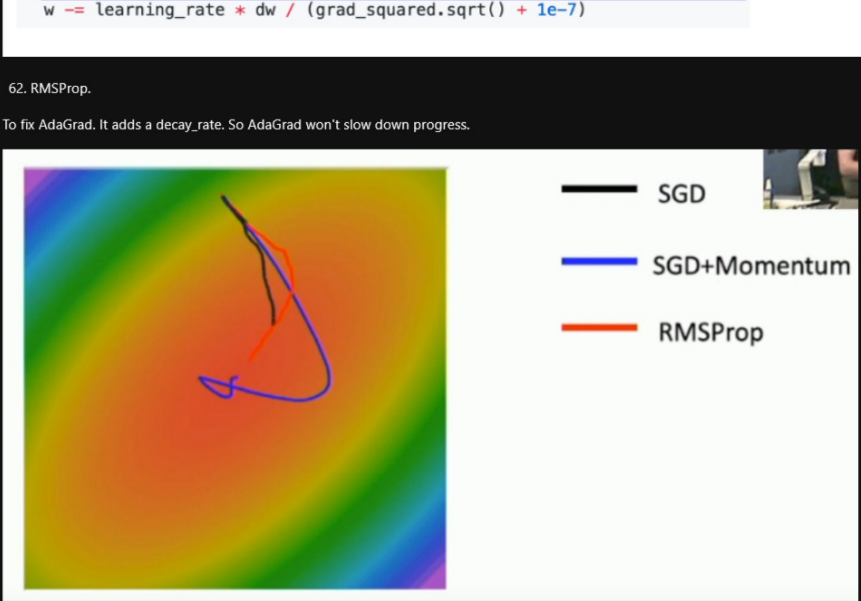
AdaGrad

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

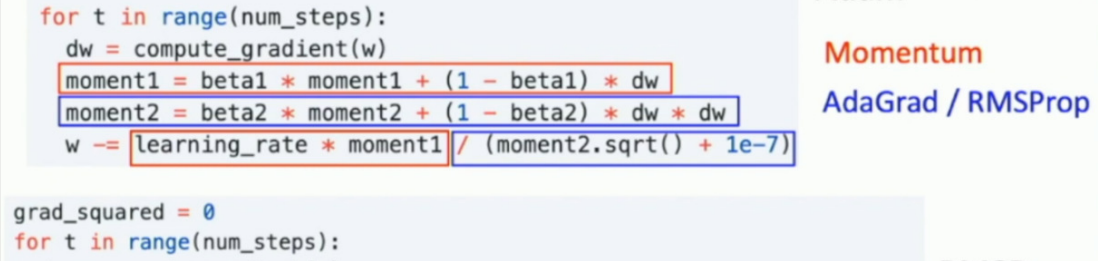
62. RMSProp.

To fix AdaGrad. It adds a decay_rate. So AdaGrad won't slow down progress.



64. Adam.

It is basically RMSProp + Momentum!



```
moment1 = 0
moment2 = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Adam
Momentum
AdaGrad / RMSProp

```
grad_squared = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
  w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

Problem. What happens at t=0? Where beta2 = 0.999. It will divide by very small number and it will take very large step. To fix it Adam adds bias correction.

This Adam works well in many systems. Used in practice.

Note that in high-dim it works very diff from 2-d as shown on demos.

74. Second Order

First-Order (первая производная) optimization. Uses grad to approximate. Steps to minimize the approximation.

Second-order optim (вторая производная). Grad + hessian (second derivative) to make quadratic approx. Step to min this approximation. So it guesses where to take large or small steps.

See formula.

Why is this imractical? Not compitible fast. O(N^3). Where N can be hunreds of millions.

82. In practice.

Adam is good choice. SGD+momentum might outperform but require tuning.

If can afford second-order, this is L-BFGS.

Questions:

11. Intuition behind how to find optimum weights. Name ideas
12. Random Search
13. Follow slope
14. How numerical gradient? What are problems with it?
15. Analytic gradient (backpropogation in lecture 6). How to check analytic gradient?
16. Gradient descent algorithm
    - What are hyperparameters now? How much should we care about those?
17. How to represent GD?
18. Stochastic GD. Describe. Two randomization sources.
19. Why is it stochastic?
20. Problems with SGD? Three problems.
21. SGD+Momentum.

- What it solves?
- Idea
- Algorithm

53. Nesterov Momentum

- Idea
- Algo

58. AdaGrad.

- Idea. What problem it solves.
- Algo

62. RMSProp

- What it fixes in AdaGrad?

64. Adam.

- Idea behind it?
- What's bias correction?

74. Second Order.

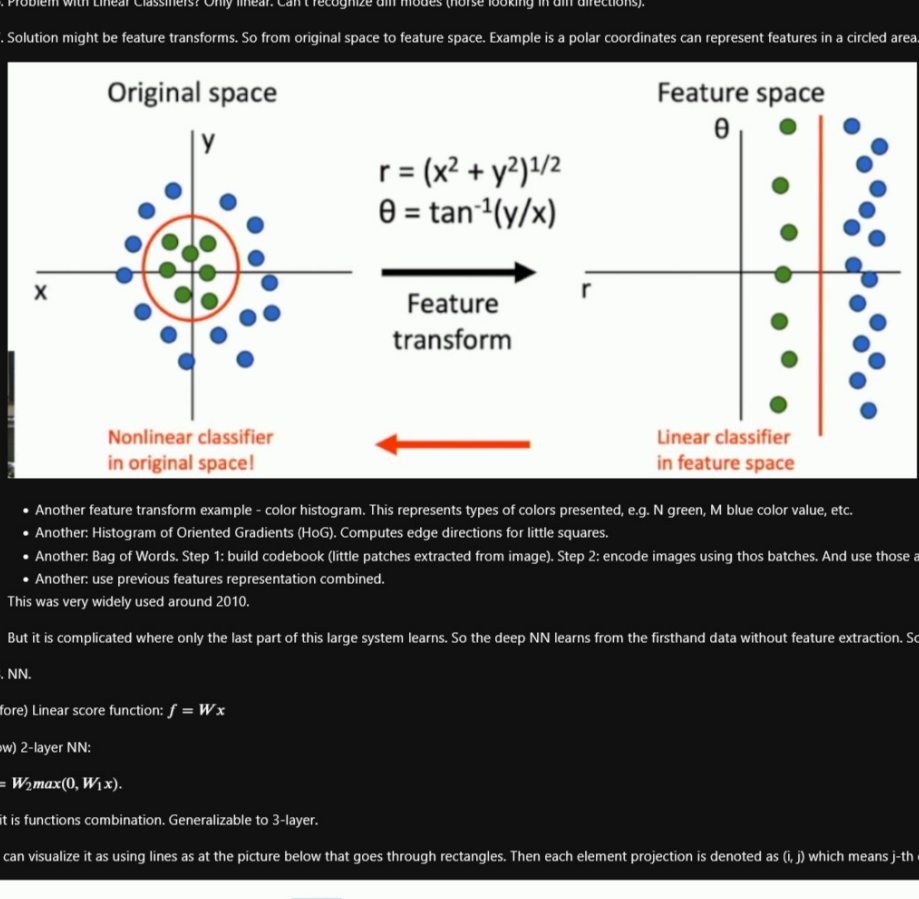
- How it is used?
- Practical? Why?
- Name algo that implements this.

[ ]:

Lecture 5. NN

6. Problem with Linear Classifiers? Only linear. Can't recognize diff modes (horse looking in diff directions).

7. Solution might be feature transforms. So from original space to feature space. Example is a polar coordinates can represent features in a circled area. So choosing smartly to feature transformation might be more meaningful.



Original space
y
x
r = (x² + y²)^1/2
θ = tan⁻¹(y/x)
Feature transform
Feature space
θ
r
Nonlinear classifier in original space!
Linear classifier in feature space

- Another feature transform example - color histogram. This represents types of colors presented, e.g. N green, M blue color value, etc.
- Another: Histogram of Oriented Gradients (HoG). Computes edge directions for little squares.
- Another: Bag of Words. Step 1: build codebook (little patches extracted from image). Step 2: encode images using thos batches. And use those as features.
- Another: use previous features representation combined.

This was very widely used around 2010.

But it is complicated where only the last part of this large system learns. So the deep NN learns from the firsthand data without feature extraction. So NN learns how to extract features and learns on them.

8. NN.

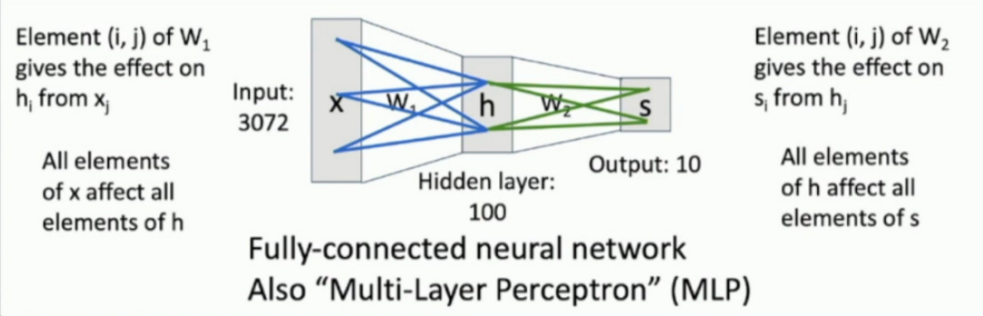
(Before) Linear score function: $f = Wx$

(Now) 2-layer NN:

$f = W_2 max(0, W_1 x)$.

So it is functions combination. Generalizable to 3-layer.

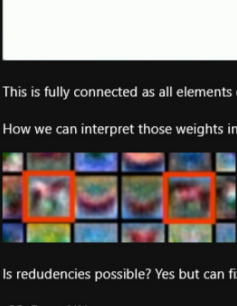
We can visualize it as using lines as at the picture below that goes through rectangles. Then each element projection is denoted as (i, j) which means j-th element from source projected to i-th element in target.



Element (i, j) of W₁ gives the effect on hᵢ from xⱼ
All elements of x affect all elements of h
Input: 3072
x
W₁
h
W₂
s
Hidden layer: 100
Output: 10
Element (i, j) of W₂ gives the effect on sᵢ from hⱼ
All elements of h affect all elements of s
Fully-connected neural network
Also "Multi-Layer Perceptron" (MLP)

This is fully connected as all elements effect all other.

How we can interpret those weights in layers? Unclear, but there is some structure. Below is an image that shows that there is two diff templates for left-headed horse and right-headed horse. But most of the time it is not interpretable.



Is redudencies possible? Yes but can fix.

32. Deep NN.

Depth - num of layers. Width - size of a layer.

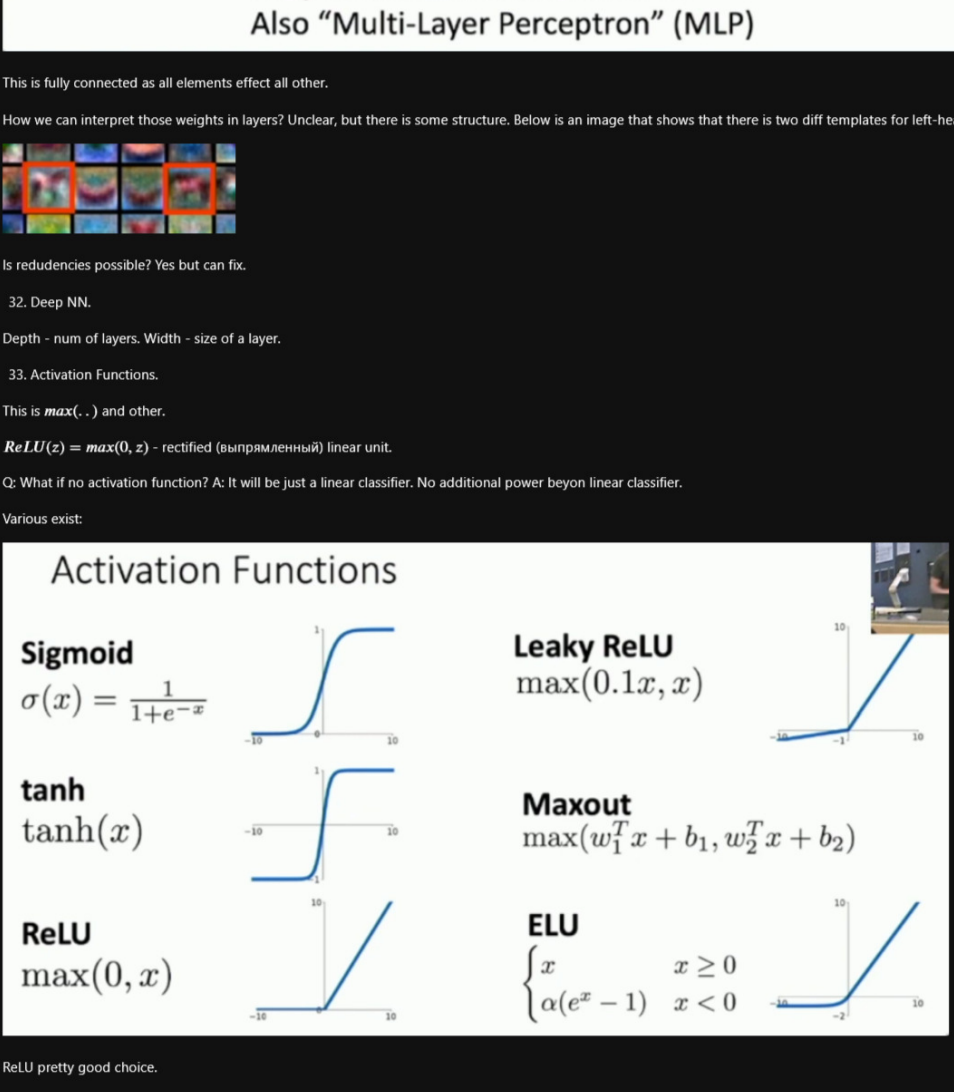
33. Activation Functions.

This is $max(.,.)$ and other.

$ReLU(z) = max(0, z)$ - rectified (выпрямленный) linear unit.

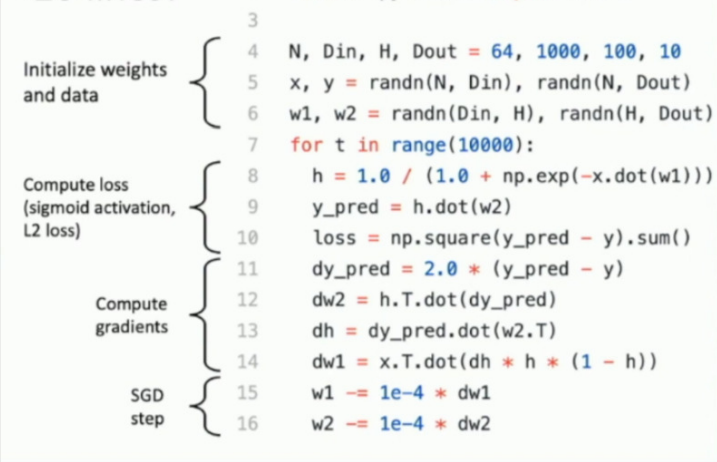
Q: What if no activation function? A: It will be just a linear classifier. No additional power beyon linear classifier.

Various exist:



ReLU pretty good choice.

39. Simplest 1 hidden layer implementation example with numpy.



<20 lines!

```
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)
for t in range(10000):
    h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
    y_pred = h.dot(w2)
    loss = np.square(y_pred - y).sum()
    dy_pred = 2.0 * (y_pred - y)
    dw2 = h.T.dot(dy_pred)
    dh = dy_pred.dot(w2.T)
    dw1 = x.T.dot(dh * h * (1 - h))
    w1 -= 1e-4 * dw1
    w2 -= 1e-4 * dw2
```

Initialize weights and data
Compute loss (sigmoid activation, L2 loss)
Compute gradients
SGD step

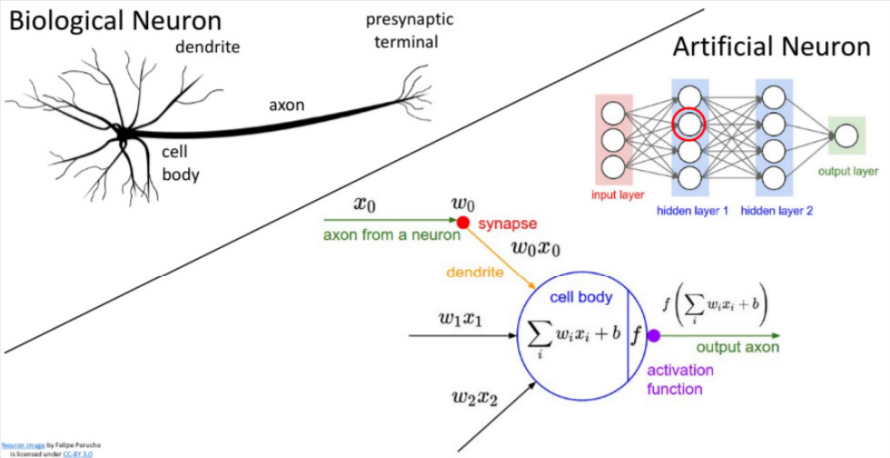
43. So what's neurons? Why this analogy?

Neuron:

- Cell body
- Axon - output
- Dendrite - input from other axon
- Synapse - connection between dendrites and axons.
- Firing rate is at what point it fires outside when received an input rates. This is similar to artificial neurons. This is where simularity ends!

There are many differences:

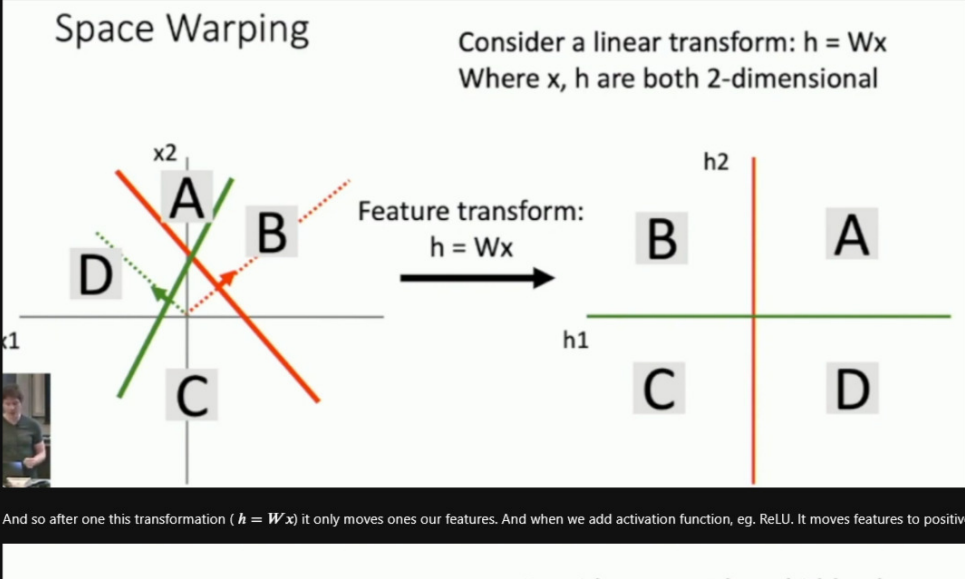
- can loop back to itself, they have very complex connectivity (topology). But artificial NN is in layers which is an abstraction. There are research on complex topologies of NN.
- Bio neurons are diff types
- Can perform non-linear computations
- Synapses are not single weight.
- Rate code might not be received.



Biological Neuron
dendrite
cell body
axon
presynaptic terminal
Artificial Neuron
input layer
hidden layer 1
hidden layer 2
output layer
x_0
axon from a neuron
w_0
synapse
$w_0 x_0$
dendrite
$w_1 x_1$
cell body
$\sum_i w_i x_i + b$
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation function
$w_2 x_2$

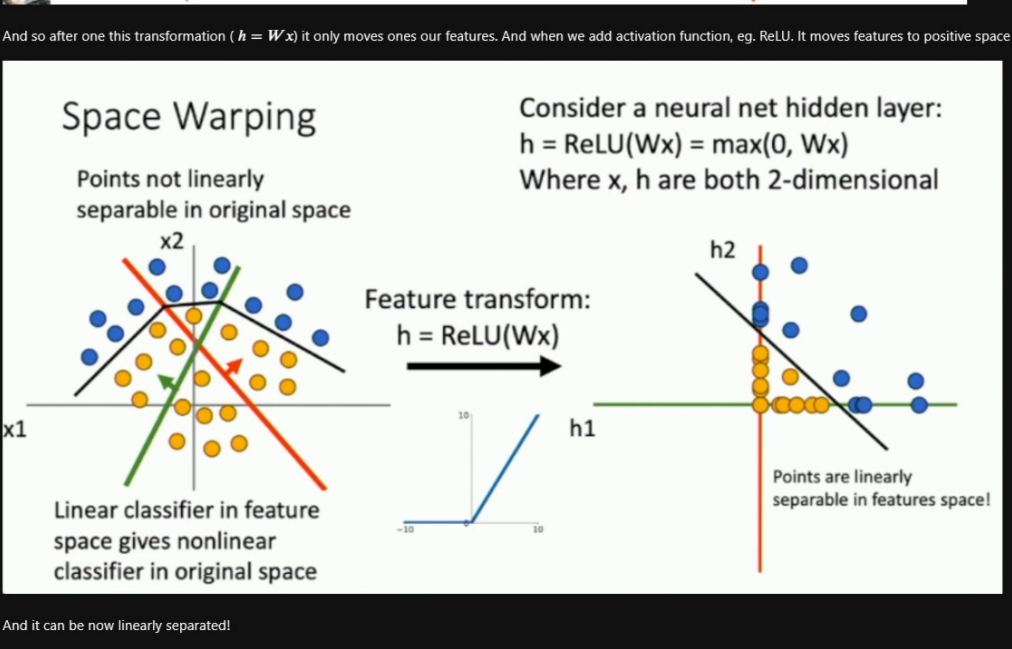
52. Space Warping.

Why NN so powerfull? Space warping. So for two classifies it can be viewed as it tranforms input into another 2-dimensional space:



Space Warping
Consider a linear transform: h = Wx
Where x, h are both 2-dimensional
x1
x2
A
B
C
D
Feature transform:
h = Wx
h1
h2
B
A
C
D

And so after one this transformation ($h = Wx$) it only moves ones our features. And when we add activation function, eg. ReLU. It moves features to positive space via projecting them on positive parts of axes:



Space Warping
Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional
Points not linearly separable in original space
x2
x1
Feature transform:
h = ReLU(Wx)
h2
h1
Points are linearly separable in features space!
Linear classifier in feature space gives nonlinear classifier in original space

And it can be now linearly separated!

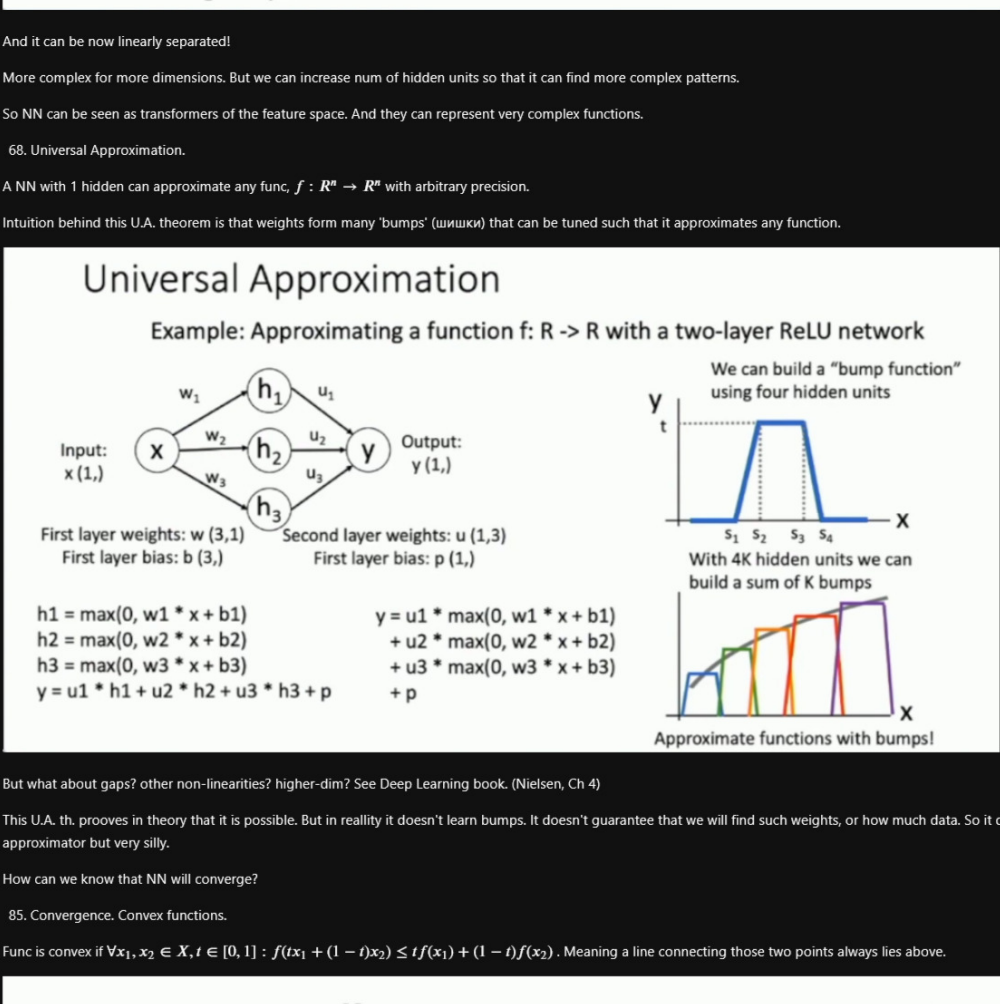
More complex for more dimensions. But we can increase num of hidden units so that it can find more complex patterns.

So NN can be seen as transformers of the feature space. And they can represent very complex functions.

68. Universal Approximation.

A NN with 1 hidden can approximate any func, $f : R^n \rightarrow R^n$ with arbitrary precision.

Intuition behind this U.A. theorem is that weights form many 'bumps' (шишки) that can be tuned such that it approximates any function.



Universal Approximation
Example: Approximating a function f: R -> R with a two-layer ReLU network
Input: x (1,)
x
w₁
w₂
w₃
h₁
h₂
h₃
u₁
u₂
u₃
y
Output: y (1,)
First layer weights: w (3,1)
First layer bias: b (3,)
Second layer weights: u (1,3)
First layer bias: p (1,)
h1 = max(0, w1 * x + b1)
h2 = max(0, w2 * x + b2)
h3 = max(0, w3 * x + b3)
y = u1 * h1 + u2 * h2 + u3 * h3 + p
y = u1 * max(0, w1 * x + b1)
+ u2 * max(0, w2 * x + b2)
+ u3 * max(0, w3 * x + b3)
+ p
We can build a "bump function" using four hidden units
y
t
x
s₁ s₂ s₃ s₄
With 4K hidden units we can build a sum of K bumps
Approximate functions with bumps!

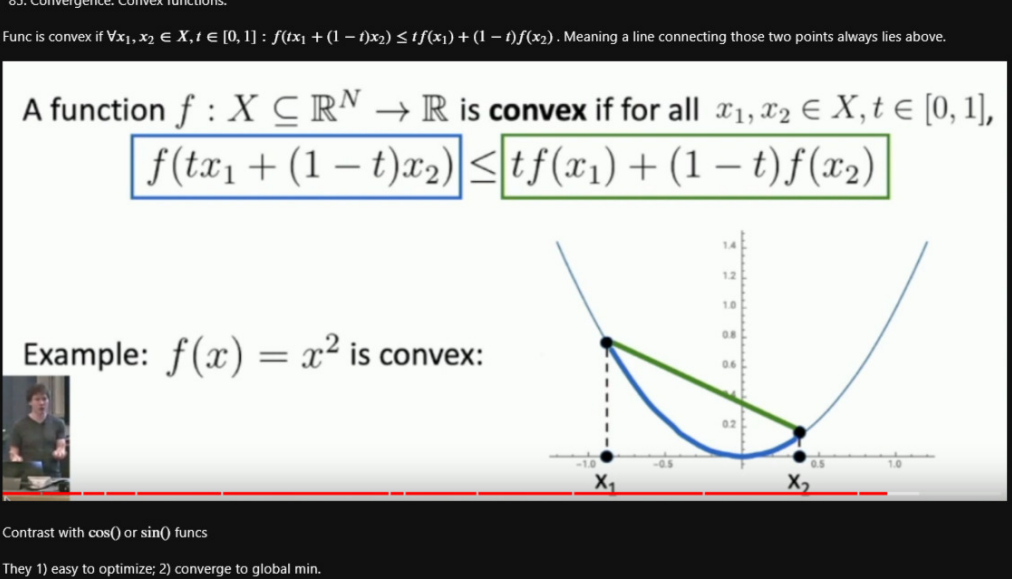
But what about gaps? other non-linearities? higher-dim? See Deep Learning book. (Nielsen, Ch 4)

This U.A. th. prooves in theory that it is possible. But in reallity it doesn't learn bumps. It doesn't guarantee that we will find such weights, or how much data. So it doesn't proove that NN is the best, silver bullet. Why? Remmber kNN is also a uni. approximator but very silly.

How can we know that NN will converge?

85. Convergence. Convex functions.

Func is convex if $\forall x_1, x_2 \in X, t \in [0, 1] : f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2)$. Meaning a line connecting those two points always lies above.

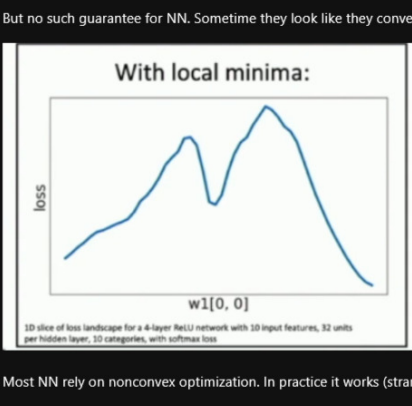


Contrast with $cos()$ or $sin()$ funcs

They 1) easy to optimize; 2) converge to global min.

- 92. Linear classifiers optimization problems are a convex functions.

But no such guarantee for NN. Sometime they look like they convex but othertime they not:



Most NN rely on nonconvex optimization. In practice it works (strange). Active area of research.

102. How to compute gradients? Backpropogation.

Questions

6. What's problem with linear classifiers?
7. Feature transformation. What's it? Name diff feature transformations.
8. NN. Basic form for 2 layers? With more layers?

- What's in representation of NN an element (i, j) means in terms of source and target.
- How can we interpret those weights in NN?
- What's depth in deep NN? What's width?

33. Activation Functions.

- What's it?
- ReLU func?
- Name other activation functions.

43. Neurons.

- Name main parts. What are they?
- Name differences with artificial neurons

52. Space warping.

- What's it? What's those h represent?
- What happens with this wraping when we add activation func?
- What's NN then?

68. Universal Approximation.

- Formulate the theorm.
- What's the intuition behind this? 'Bumps'?
- So any NN will work in practice?

85. Convergence.

- What's convex function? Formulate condition.
- Is Linear classifier a convex func?
- Is NN a convex func? What happens in practice?

[ ]:

Lecture 6. Backpropagation

5. How to compute gradients? If we compute grad we can optimize the loss func. How for arbitrarity weights.

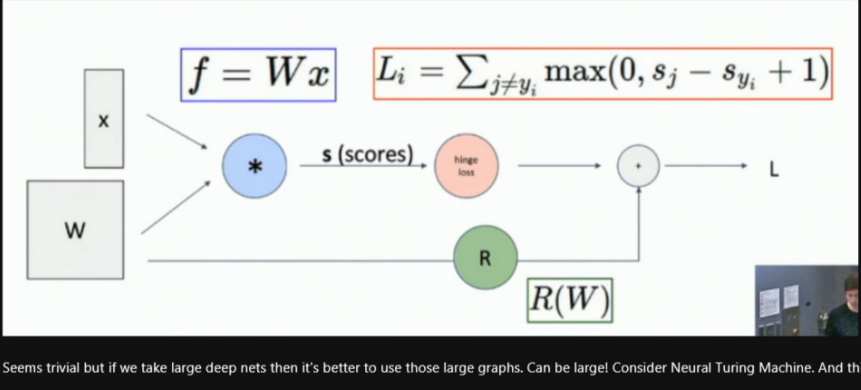
6. Naive. Derive gradient on paper. Idea is to write the derivative and then insert values.

- Why this is bad?
    - Tedious. Long to write.
    - Not feasible for complex models. E.e. softmax.
    - Not modular design. If we have done for sigmoid loss but today want ReLU? Re-write then. So we want modular approach.

Better idea is using computational graphs.

7. Computations graph.

Units are functions or variables that are combined with arithmentic operations. See picture:



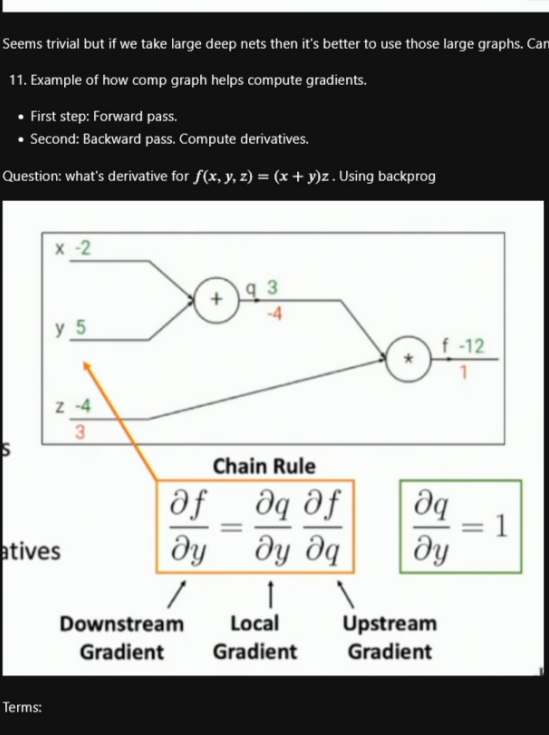
$f = Wx$
$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
x
W
*
s (scores)
hinge loss
+
L
R
$R(W)$

Seems trivial but if we take large deep nets then it's better to use those large graphs. Can be large! Consider Neural Turing Machine. And this can be even more complex when nets are recurrent.

11. Example of how comp graph helps compute gradients.

- First step: Forward pass.
- Second: Backward pass. Compute derivatives.

Question: what's derivative for $f(x, y, z) = (x + y)z$. Using backprog

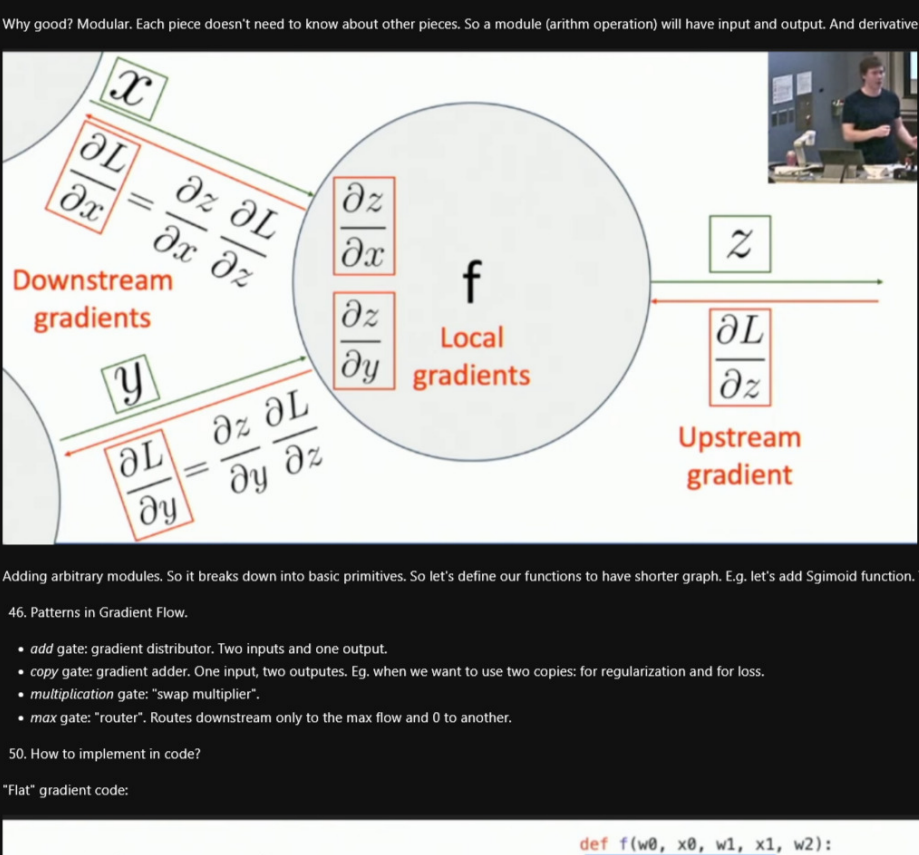


x -2
y 5
z -4
3
+
q 3
-4
*
f -12
1
s
Chain Rule
$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$
$\frac{\partial q}{\partial y} = 1$
atives
Downstream Gradient
Local Gradient
Upstream Gradient

Terms:

- Downstream grad
- Local grad
- Upstream grad

Why good? Modular. Each piece doesn't need to know about other pieces. So a module (arithm operation) will have input and output. And derivative is calculated backward way. This is because of chain rule.



x
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
Downstream gradients
y
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
z
$\frac{\partial L}{\partial z}$
Upstream gradient

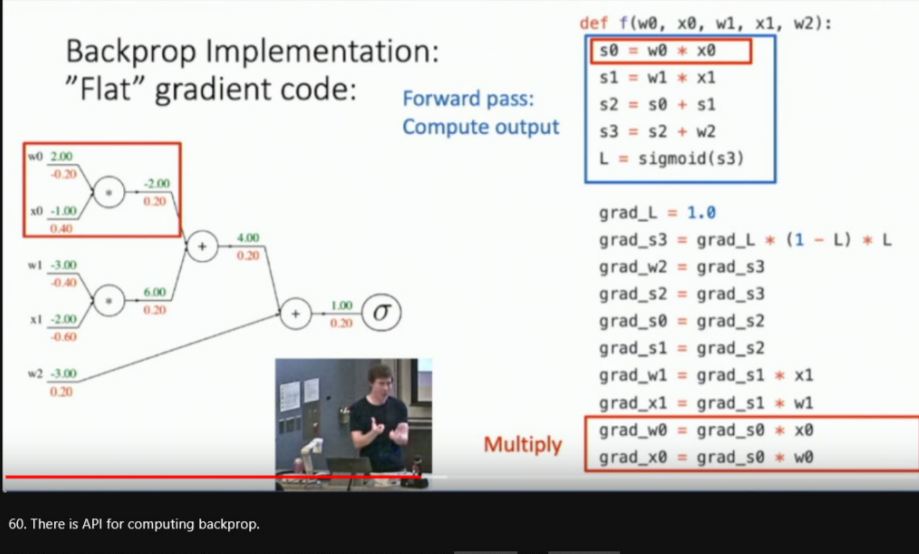
Adding arbitrary modules. So it breaks down into basic primitives. So let's define our functions to have shorter graph. E.g. let's add Sgimoid function. This is when one input and one output.

46. Patterns in Gradient Flow.

- *add* gate: gradient distributor. Two inputs and one output.
- *copy* gate: gradient adder. One input, two outputs. Eg. when we want to use two copies: for regularization and for loss.
- *multiplication* gate: "swap multiplier".
- *max* gate: "router". Routes downstream only to the max flow and 0 to another.

50. How to implement in code?

"Flat" gradient code:



Backprop Implementation:
"Flat" gradient code:

Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  grad_L = 1.0
  grad_s3 = grad_L * (1 - L) * L
  grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad_x0 = grad_s0 * w0
```

Multiply

w0 2.00
-0.20
x0 -1.00
0.40
w1 -3.00
-0.40
x1 -2.00
-0.60
w2 -3.00
0.20
-2.00
0.20
6.00
0.20
4.00
0.20
1.00
0.20
σ

60. There is API for computing backprop.

In PyTorch - autograd functions. It has a class for each operation that has `forward` and `backward`.

67. What about vector-valued func?

1 varaible - Derivative. n vector - Gradient. n times m - Jacobian.

The same picture for backprop with Vectors: downstream, upstream, etc.

75. Backprop with Vectors.

For $max(0, x)$. Jacobian matrix is diagonal then. So never form explicit form of Jacobian.

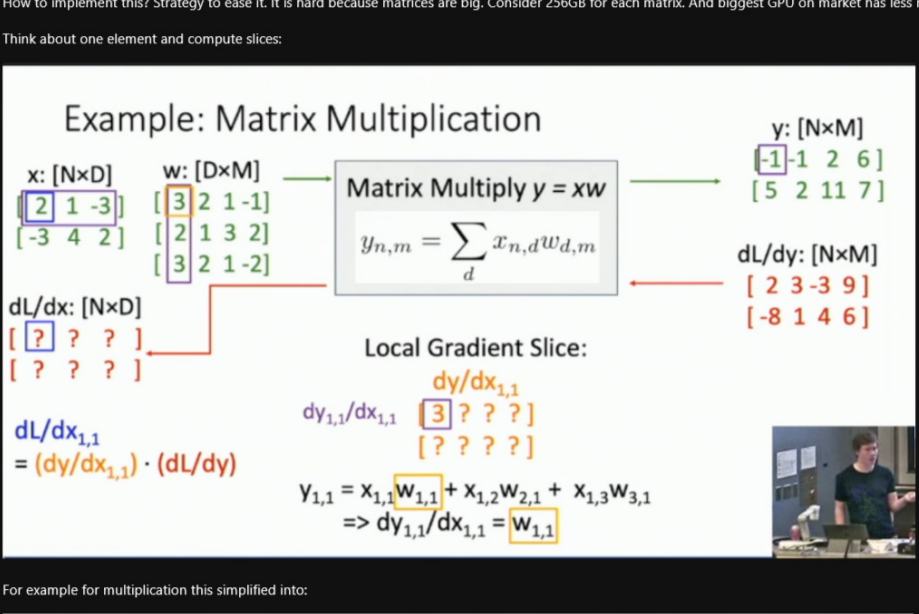
82. Backprop with Matrices (Tensor)

Upstream grad is matrix. Donwstream is matrix, tensors.

86. How to strategy to ease computations.

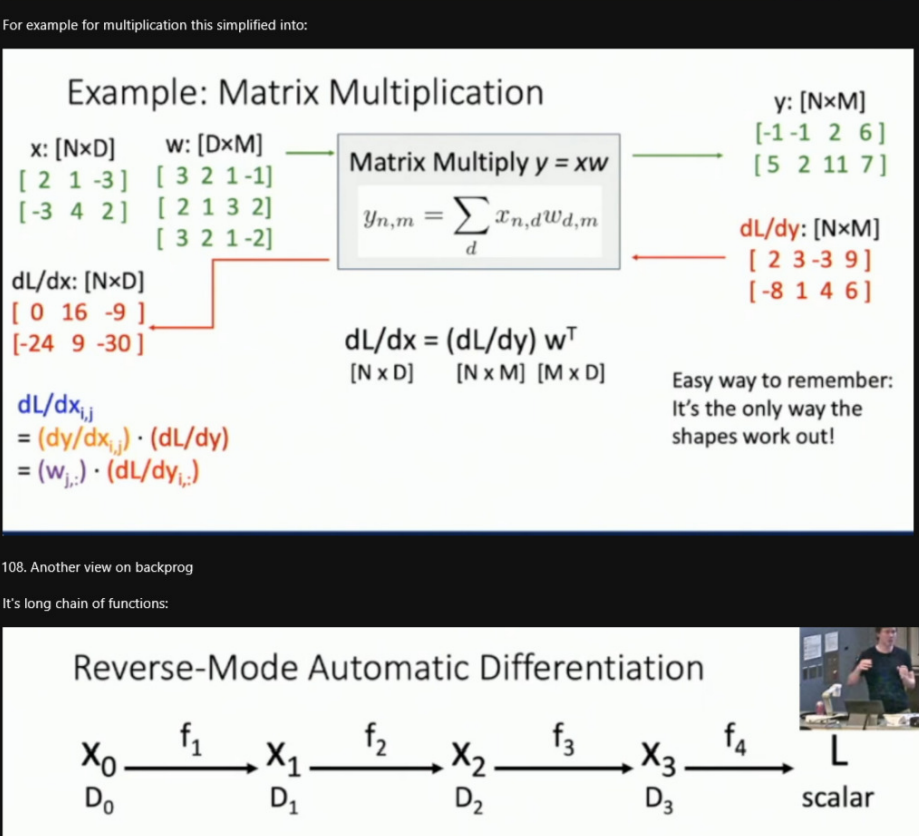
How to implement this? Strategy to ease it. It is hard because matrices are big. Consider 256GB for each matrix. And biggest GPU on market has less memory. How to compute without explicitely?

Think about one element and compute slices:



Example: Matrix Multiplication
x: [N×D]
[2 1 -3]
[-3 4 2]
w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]
Matrix Multiply y = xw
$y_{n,m} = \sum_d x_{n,d} w_{d,m}$
y: [N×M]
[-1 -1 2 6]
[5 2 11 7]
dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]
dL/dx: [N×D]
[? ? ?]
[? ? ?]
Local Gradient Slice:
dy/dx₁,₁
dy₁,₁/dx₁,₁
[3 ? ? ?]
[? ? ? ?]
dL/dx₁,₁
= (dy/dx₁,₁) · (dL/dy)
y₁,₁ = x₁,₁w₁,₁ + x₁,₂w₂,₁ + x₁,₃w₃,₁
=> dy₁,₁/dx₁,₁ = w₁,₁

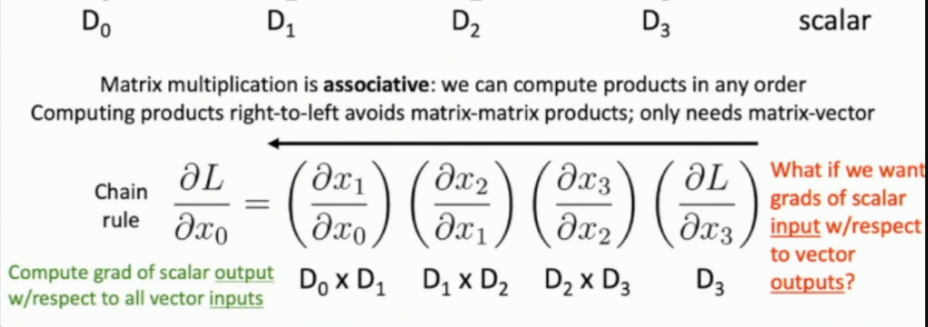
For example for multiplication this simplified into:



Example: Matrix Multiplication
x: [N×D]
[2 1 -3]
[-3 4 2]
w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]
Matrix Multiply y = xw
$y_{n,m} = \sum_d x_{n,d} w_{d,m}$
y: [N×M]
[-1 -1 2 6]
[5 2 11 7]
dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]
dL/dx: [N×D]
[0 16 -9]
[-24 9 -30]
dL/dx = (dL/dy) wᵀ
[N x D] [N x M] [M x D]
dL/dxᵢ,ⱼ
= (dy/dxᵢ,ⱼ) · (dL/dy)
= (wⱼ,:) · (dL/dyᵢ,:)
Easy way to remember: It's the only way the shapes work out!

108. Another view on backprog

It's long chain of functions:



Reverse-Mode Automatic Differentiation
x₀ D₀
f₁
x₁ D₁
f₂
x₂ D₂
f₃
x₃ D₃
f₄
L scalar
Matrix multiplication is associative: we can compute products in any order
Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector
Chain rule
$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right)\left(\frac{\partial x_2}{\partial x_1}\right)\left(\frac{\partial x_3}{\partial x_2}\right)\left(\frac{\partial L}{\partial x_3}\right)$
D₀ x D₁
D₁ x D₂
D₂ x D₃
D₃
Compute grad of scalar output w/respect to all vector inputs
What if we want grads of scalar input w/respect to vector outputs?

There is Forward-Mode Automatic Differentiation which is for diff w.r.t. a scalar. This not used in ML but might be useful elsewhere (physical simulations).

115. Higher-Order Derivatives.

The same backprop used. Hessian matrix - second-order. Tricks for multiplying Hessian by vector. This is based on first-order derivatives. So we can compute higher-order derivaties. It is implemenhted in PyTorch/TensorFlow.

122. Summary

So we have linear classifier, NN, backprop. But it doesn't respect the spatial structure of images. So let's go to Convolutional NN.

Questions:

5. Why computing gradient 'on paper' and then using it in code is bad?
6. What is an alternative? Computational graph?
7. Backprop

- Show how to do backprop on $f(x, y, z) = (x + y)z$. Where x=-2, y=5, z=-4.
- What's downstream/local/upstream gradients?
- Why it is good?

60. How is it implemented in PyTorch?
61. How to backprop with vector-valued functions?
62. With vector. If with vector, what type is Jacobian matrix then?
63. With matrices. What's Jacobian matrix?

- Computational problem with Jacobian matrices. How to simplify computation?

108. Another view on backprop? Chain of functions.
109. Is it possible to compute higher-order derivatives with Pytorch/TensorFlow?
110. Why do we need Convolutional NN now when we have loss func, backprop, linear classifier, etc.? What do we lack now?

[ ]:

Lecture 7: Convolutional NN

4. Recap. Backprop. Forward pass, backward pass, etc.

5. Problem now is that none of those classifiers (linear or NN) respect spatial structure. But it is very usefull to take into account spacial structure.

- Solution? Define new operators (computational nodes) that operate one spatial input.

7. Components of a Con. Network.

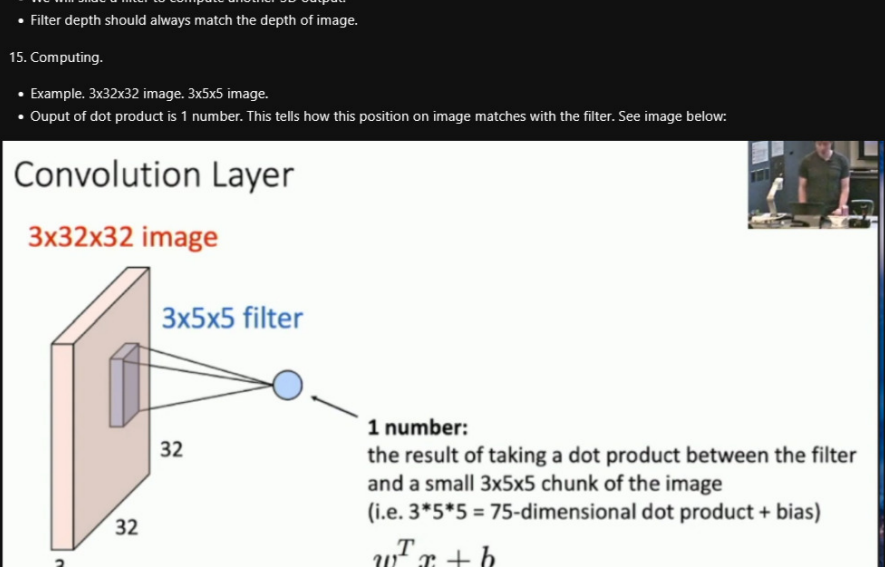
- Fully-Connected Layers
- Convolutional Layers
- Activation Function
- Pooling Layers
- Normalization

9. Convolution Layers

- It moves from Fully-Connected Layer. To recap: Fully Conn. layer takes flatten layer. And we multiply with W.
- Now, Con. Layer preserve spatial structure. E.g. 3x32x32 image. 3 - depth or channel. 32x32 width and height.
- Weight matrix now have also 3D structure. Now it is 'filter'. E.g. 3x5x5.
- We will slide a filter to compute another 3D output.
- Filter depth should always match the depth of image.

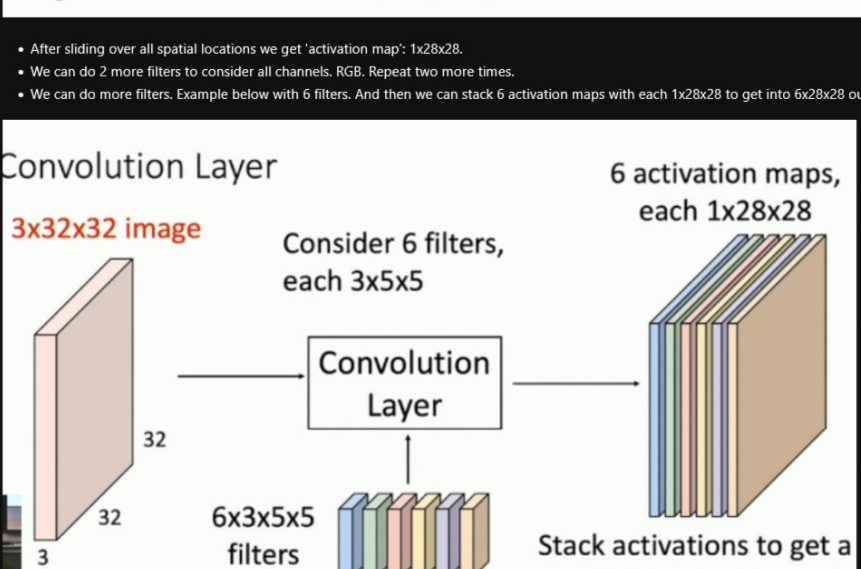
15. Computing.

- Example. 3x32x32 image. 3x5x5 image.
- Ouput of dot product is 1 number. This tells how this position on image matches with the filter. See image below:



Convolution Layer
3x32x32 image
3x5x5 filter
32
32
3
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)
$w^T x + b$

- After sliding over all spatial locations we get 'activation map': 1x28x28.
- We can do 2 more filters to consider all channels. RGB. Repeat two more times.
- We can do more filters. Example below with 6 filters. And then we can stack 6 activation maps with each 1x28x28 to get into 6x28x28 output image.

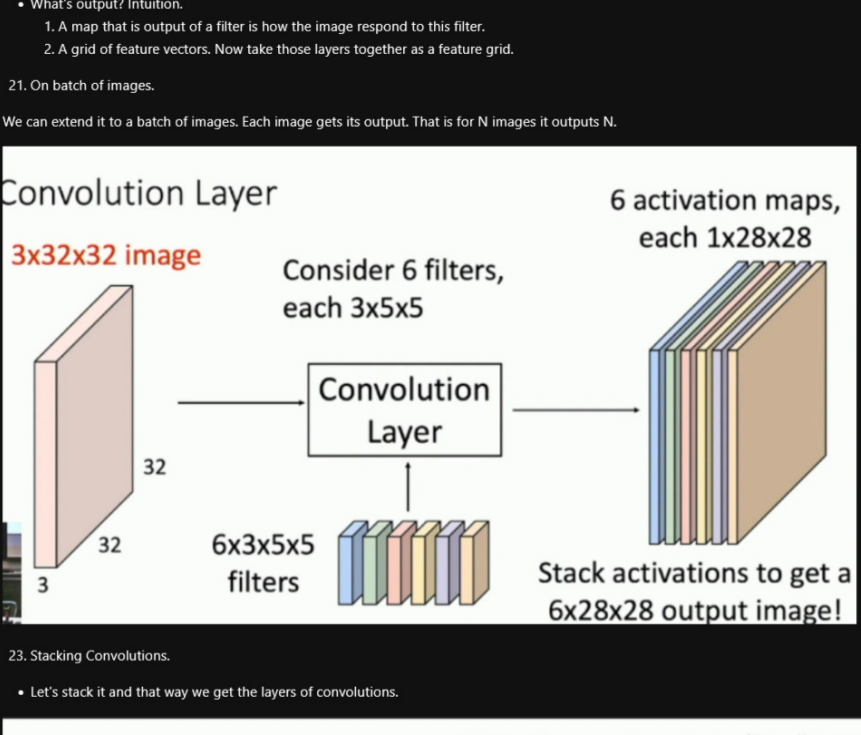


Convolution Layer
3x32x32 image
32
32
3
Consider 6 filters, each 3x5x5
Convolution Layer
6x3x5x5 filters
6 activation maps, each 1x28x28
Stack activations to get a 6x28x28 output image!

- Conv. layer always 6 dim bias vector.
- What's output? Intuition.
    1. A map that is output of a filter is how the image respond to this filter.
    2. A grid of feature vectors. Now take those layers together as a feature grid.

21. On batch of images.

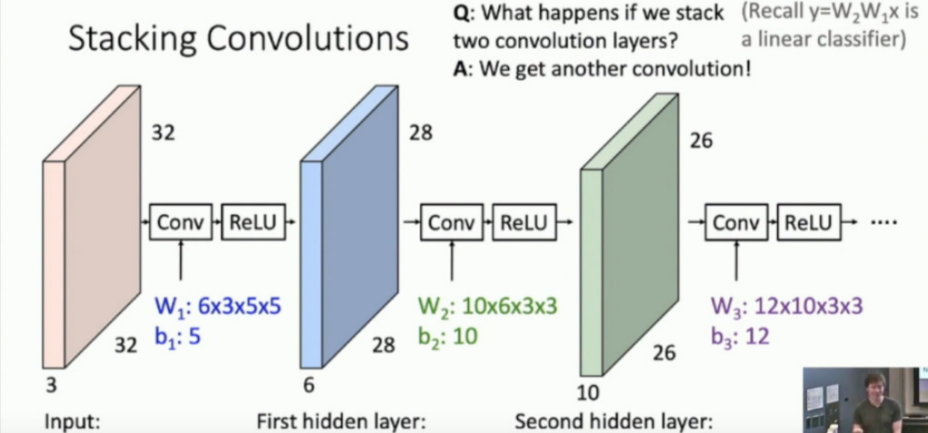
We can extend it to a batch of images. Each image gets its output. That is for N images it outputs N.



Convolution Layer
3x32x32 image
32
32
3
Consider 6 filters, each 3x5x5
Convolution Layer
6x3x5x5 filters
6 activation maps, each 1x28x28
Stack activations to get a 6x28x28 output image!

23. Stacking Convolutions.

- Let's stack it and that way we get the layers of convolutions.

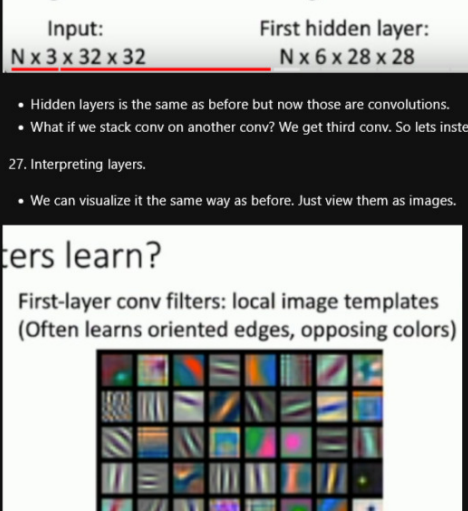


Stacking Convolutions
Q: What happens if we stack two convolution layers?
A: We get another convolution!
(Recall y=W₂W₁x is a linear classifier)
32
32
3
Conv
ReLU
W₁: 6x3x5x5
b₁: 5
28
28
6
Conv
ReLU
W₂: 10x6x3x3
b₂: 10
26
26
10
Conv
ReLU
W₃: 12x10x3x3
b₃: 12
....
Input:
N x 3 x 32 x 32
First hidden layer:
N x 6 x 28 x 28
Second hidden layer:
N x 10 x 26 x 26

- Hidden layers is the same as before but now those are convolutions.
- What if we stack conv on another conv? We get third conv. So lets instert non linear activation function, let's add 'ReLU'.

27. Interpreting layers.

- We can visualize it the same way as before. Just view them as images.



ers learn?
First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)