

Source:
Somewhere on Reddit



The components

CIS 522: Lecture 4
1/28/20

Feedback from last time, thanks

- Slides/ Notes before lecture -> Slides are posted ahead of time.
- Pytorch examples -> recitations
- More examples -> coming
-
- Too fast/ slow, technical/ intuitive, etc - mixed group

The components of DL

Linear operations

Activation functions

Loss functions

Initializations

Optimizers

Regularizations

Architectures (rest of the course)

How important are the components

Activation Function

Loss Function

Initialization

Optimizers

Regularization

Architectures



Why the parts matter

In ML: Linear regression does much of it

Generic domain tricks: e.g. convolution, transformers do most of the rest

A gazillion minor tricks do the rest

Intuition is the key to know how to make progress

Optimization of Linear Models

The components of DL

Linear operations

Activation functions

Loss functions

Initializations

Optimizers

Regularizations

Architectures (rest of the course)

When poll is active, respond at **PollEv.com/konradkordin059**

Text **KONRADKORDIN059** to **22333** once to join

Why is ReLU popular?

Runs fast

Avoids vanishing gradients

Implements polynomials

Is good with probabilities

None of the above



The components of DL

Linear operations

Activation functions

Loss functions

Initializations

Optimizers

Regularizations

Architectures (rest of the course)

Log P

What does it mean to have good predictions?

Make the stuff we predict be probable!

Why log P?

Examples

Poisson regression in Neuroscience

$$f(k; \lambda) = \Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

A lot of the others

Mean squared error

$$\mathcal{L}_N(X, Y) = \frac{1}{s} \sum_{i=1}^s \|y_i - N(x_i)\|_2^2$$

Mean squared error

$$\mathcal{L}_N(X, Y) = \frac{1}{s} \sum_{i=1}^s \|y_i - N(x_i)\|_2^2$$

- Regression
- Often used if the network is predicting a scalar
- Gaussian!

Mean Absolute error

$$\mathcal{L}_N(X, Y) = \frac{1}{s} \sum_{i=1}^s \|y_i - N(x_i)\|_1$$

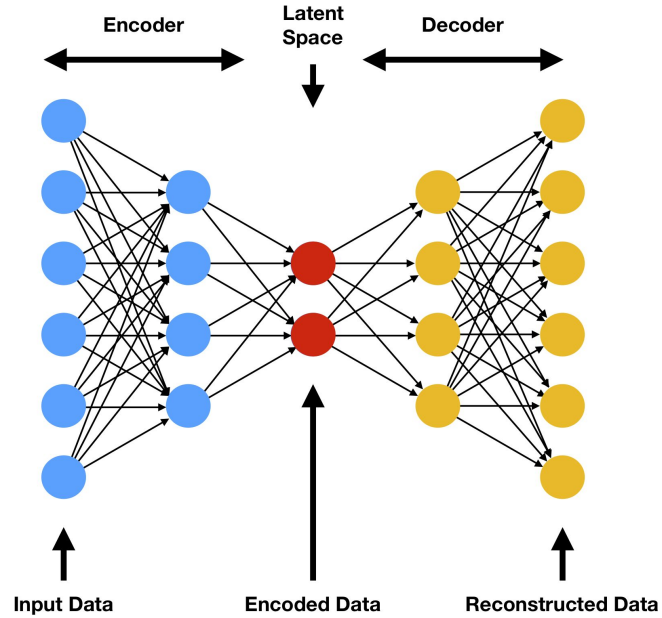
Mean Absolute error

$$\mathcal{L}_N(X, Y) = \frac{1}{s} \sum_{i=1}^s \|y_i - N(x_i)\|_1$$



- Regression
- Often used if we expect outliers
- An example in image denoising: salt-and-pepper noise (outliers) on pixels (left) and Gaussian noise (right).

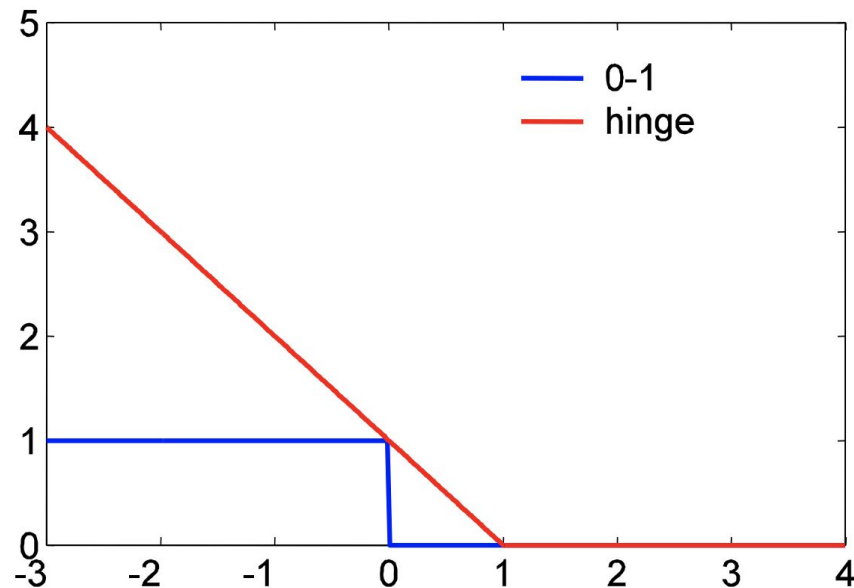
Examples



Various flavors of
Reinforcement learning

Hinge Loss

$$\ell(y) = \max(0, 1 - t \cdot y)$$



Link DL/ SVMs

Remember 520/ 519?

Same cost function as used for SVMs

Cross-entropy

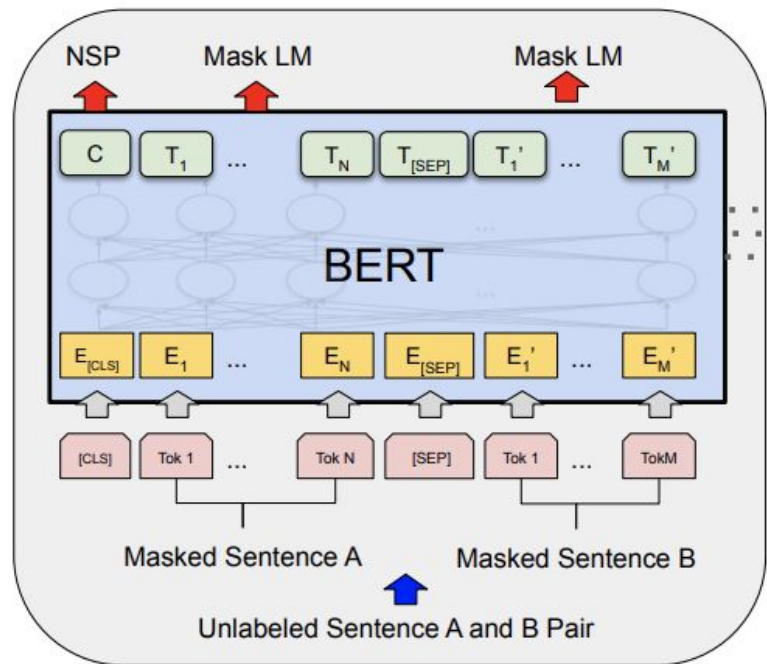
$$\mathcal{L}_N(X, Y) = \frac{1}{s} \sum_{i=1}^s \left(- \sum_j y_{ij} \log N(x_i)_j \right)$$

- Popular for classification tasks.
- Distance between two probability distributions.
- target vector is one-hot encoded:
- which means that y_{ij} is 1 where x_i belongs to class j , and is otherwise 0.
- Needs probabilities (softmax).
- In PyTorch, softmax and cross-entropy can be applied in a single step.
- **Be careful** in that case not to apply it twice :)

Examples

Typical NLP application:

Predict next word, e.g. for BERT



Cosine similarity

$$\mathcal{L}_N(X, Y) = \frac{1}{s} \sum_{i=1}^s \frac{y_i \cdot N(x_i)}{\|y_i\|_2 \|N(x_i)\|_2}$$

- Computes similarity between directions of two vectors.

Examples

- Often used in NLP contexts. For example, the two vectors might be the word-to-vec embeddings of the real sentence and a predicted one.
- E.g. used for recommender systems

Engineering loss functions

We take the image texture synthesis (neural style transfer) for example.

Main idea: using a pretrained deep network on ImageNet classification task to help with the other vision tasks. The correlation between each channels reflects the visual features at each sampling point.

Style transfer

A



B



Content Loss

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

Original

Generated

Are the features on layer l similar?


Style loss

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

Summation is over all channels in each layer.

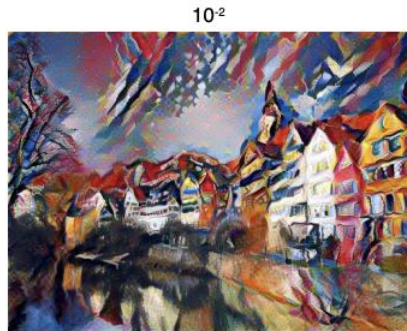
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

Difference between synthesized images with the style reference image.

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l$$


Custom Loss Functions: Style Transfer

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$



Art?

C



D



E

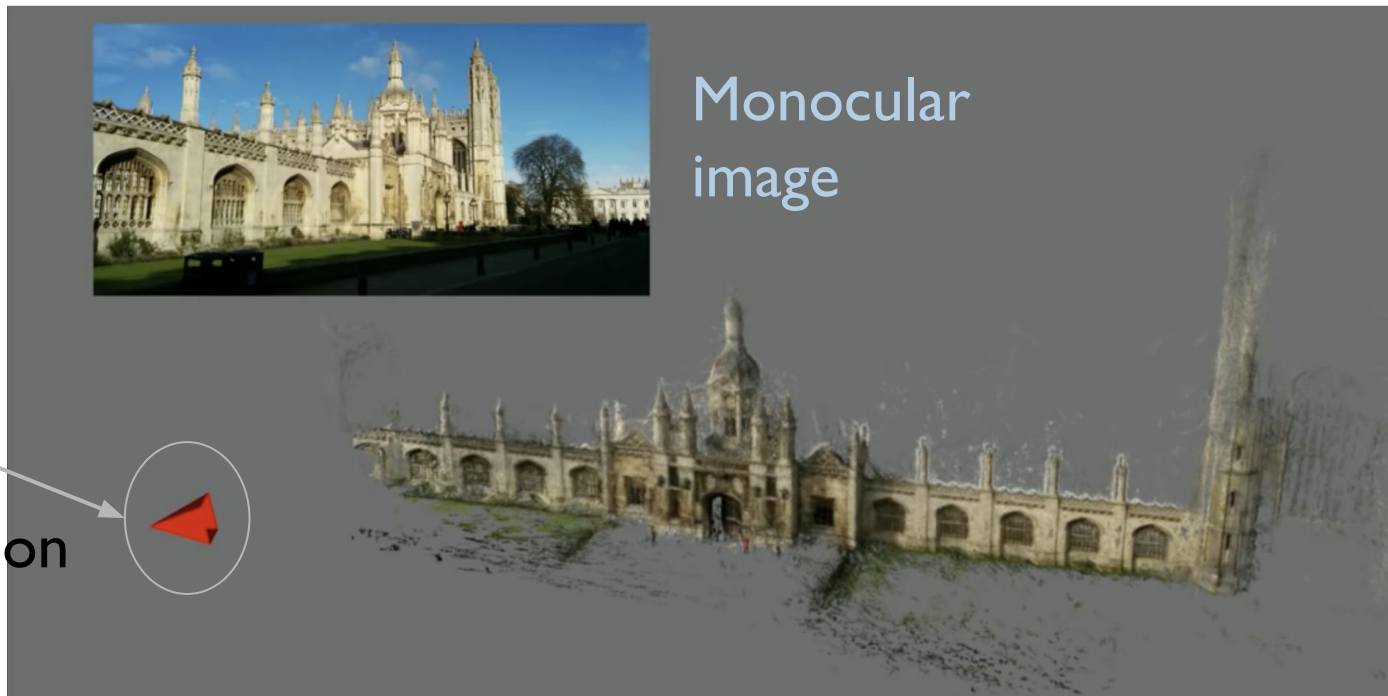


F



Domain knowledge-informed loss: Camera localization

6-DOF
camera
localization

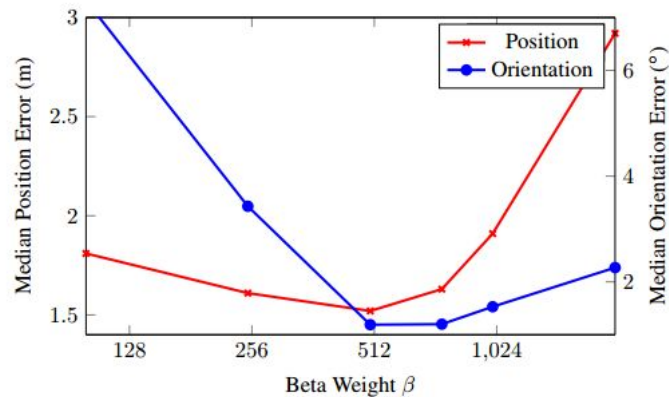
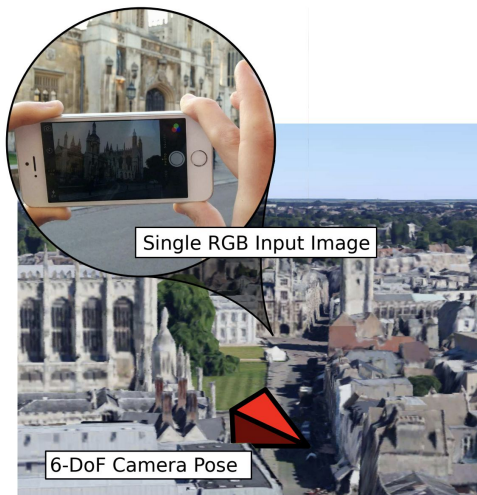


Domain knowledge-informed loss: Camera localization

how to weigh position and orientation losses?

Naive:

$$\mathcal{L}_{\beta}(I) = \overset{\text{position}}{\mathcal{L}_x(I)} + \beta \overset{\text{orientation (quaternion)}}{\mathcal{L}_q(I)}$$



Source: Kendall and Cipolla, 2017

Better loss function!

Geometry-informed loss function without hyperparameter,
combines position and orientation to reprojection error:

$$\mathcal{L}_g(I) = \frac{1}{|\mathcal{G}'|} \sum_{g_i \in \mathcal{G}'} \|\pi(\mathbf{x}, \mathbf{q}, \mathbf{g}_i) - \pi(\hat{\mathbf{x}}, \hat{\mathbf{q}}, \mathbf{g}_i)\|_{\gamma} \quad , \text{ where } \pi(\mathbf{x}, \mathbf{q}, \mathbf{g}) \mapsto \begin{pmatrix} u \\ v \end{pmatrix}$$

Subset of all 3d points in the scene

Statistically-optimal Weighting

how to weigh position and orientation losses using statistics knowledge?

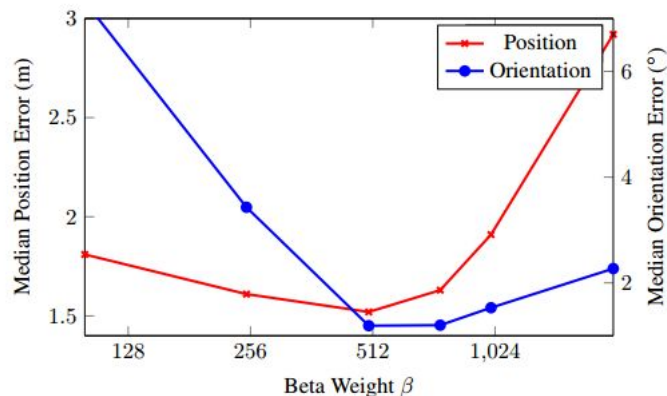
$$\mathcal{L}_\beta(I) = \mathcal{L}_x(I) + \beta \mathcal{L}_q(I)$$

reformulate the loss function as a function of the parameter of the uncertainty (Laplace distribution):

$$\mathcal{L}_\sigma(I) = \mathcal{L}_x(I) \hat{\sigma}_x^{-2} + \log \hat{\sigma}_x^2 + \mathcal{L}_q(I) \hat{\sigma}_q^{-2} + \log \hat{\sigma}_q^2$$

Then minimize w.r.t the variance parameter.

Empirical results for different weighting.



Fairness

Setting: 2 groups, special group S and background group T

Fairness?

See the new book by Michael Kearns and Aaron Roth

Fairness

Setting: 2 groups, special group S and background group T

Fairness version I:

$P(a) = Q(a)$ or

$$D_{\text{tv}}(P, Q) = \frac{1}{2} \sum_{a \in A} |P(a) - Q(a)|$$

Fairness Through Awareness

Cynthia Dwork^{*}

Moritz Hardt[†]

Toniann Pitassi[‡]

Omer Reingold[§]

Richard Zemel[¶]

Example ads for credit cards

Redlining. A well-known form of discrimination based on redundant encoding. The following definition appears in an article by [Hun05], which contains the history of the term, the practice, and its consequences: “Redlining is the practice of arbitrarily denying or limiting financial services to specific neighborhoods, generally because its residents are people of color or are poor.”

Alternative, enforce equality of outcomes

High-Level Goal: statistical parity
demographics of the “positive” group
same as the demographics of the population .

Problem Statement: Given two groups S and T , ensure statistical parity when members of S are less likely to be “qualified”.

Example: Given a set of low-income high school students (S) and a set of high-income high school students (T), ensure that proportions of students “accepted” to Penn are equal.

Curiosity

Matters a great deal for RL.

And for lifelong learning.

How would you define it?

Failure modes for loss functions

Unsupervised learning

GANs

Supervised learning (e.g. localized signals)

Loss functions

Situation	Good loss
Probabilities	Log P
Continuous	MSE
+small data	Hinge Loss
Categories	Cross Entropy
high-D	Cosine similarity
Problem	Approximation thereof

Describe a good cost function for a human being who wants to walk from one end of a room to another.



The components of DL

Linear operations

Activation functions

Loss functions

Initializations

Optimizers

Regularizations

Architectures (rest of the course)

How do we initialize weights?

Affects learning speed, convergence, generalization

- Initialize around zero
- Why? Activation functions inflect around zero
- In perceptron, we initialized the weights to zero
 - Can we do it with neural network?

Potential problems

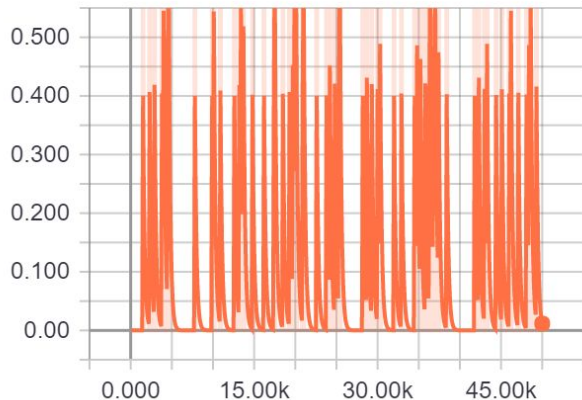
- a) No gradients
- b) Infinite gradients
- c) All gradients in (very) low dimensional space

Disaster! Zero Initialization is Terrible

Zero initialization

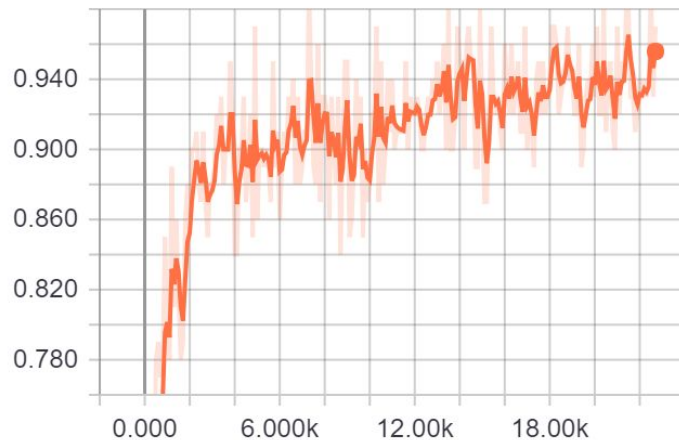
accuracy

accuracy



What we want

accuracy



BAD

If all weights are zero

Gradient is the same within layer
network with a layer width of one

Moral of the story: don't use zero
initialization. Ever.

Outputs

```
tensor(1.0000e-04 *  
  [[-1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,  
    9.0000, -1.0000, -1.0000]])  
tensor(1.0000e-04 *  
  [[-1.9999, -1.9999, -1.9999, -1.9999, -1.9999, -1.9999, -1.9999,  
    7.9991, 8.0001, -1.9999]])  
tensor(1.0000e-04 *  
  [[-2.9997, -2.9997, 7.0003, -2.9997, -2.9997, -2.9997, -2.9997,  
    6.9983, 6.9993, -2.9997]])  
tensor(1.0000e-04 *  
  [[-3.9994, -3.9994, 5.9996, -3.9994, 6.0006, -3.9994, -3.9994,  
    5.9976, 5.9986, -3.9994]])  
tensor(1.0000e-04 *  
  [[-4.9990, -4.9990, 4.9990, 5.0010, 5.0000, -4.9990, -4.9990,  
    4.9970, 4.9980, -4.9990]])  
tensor(1.0000e-04 *  
  [[-5.9985, -5.9985, 3.9985, 4.0005, 3.9995, -5.9985, -5.9985,  
    3.9965, 3.9975, 4.0015]])  
tensor(1.0000e-03 *  
  [[-0.6998, -0.6998, 0.2998, 1.3000, 0.2999, -0.6998, -0.6998,  
    0.2996, 0.2997, 0.3001]])
```

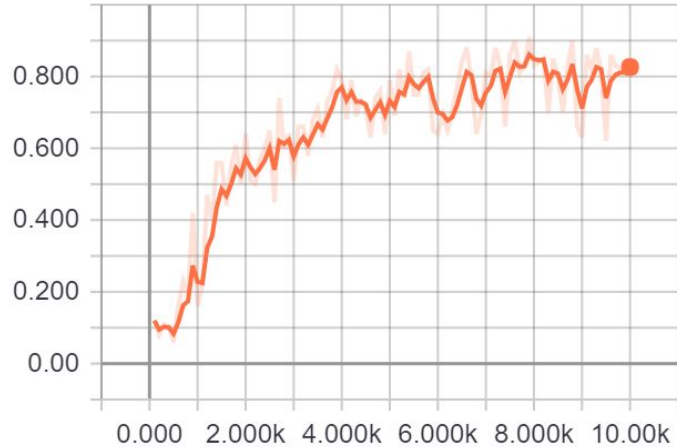
Gradients

```
tensor([-0.9000, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000,  
  0.1000, 0.1000, 0.1000])  
tensor([-0.8999, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000, 0.1000,  
  0.1000, 0.1000, 0.1000])  
tensor([ 0.1002, 0.1000, 0.1000, 0.1000, 0.1000, -0.9000, 0.1000,  
  0.1000, 0.1000, 0.1000])  
tensor([ 0.1002, 0.1000, 0.1000, 0.1000, 0.1000, -0.9000, 0.1001, 0.1000,  
  0.1000, 0.1000])  
tensor([ 0.1002, -0.9000, 0.1000, 0.1000, 0.1001, 0.1001, 0.1000,  
  0.1000, 0.1000, 0.1000])  
tensor([ 0.1002, 0.1001, 0.0999, 0.0999, -0.9000, 0.1000, 0.0999,  
  0.0999, 0.0999, 0.0999])  
tensor([ 0.1001, 0.1000, 0.0999, -0.9001, 0.1001, 0.1000, 0.0999,  
  0.0999, 0.0999, 0.0999])  
tensor([ 0.1001, -0.9000, 0.0999, 0.1000, 0.1001, 0.1000, 0.0999,  
  0.0999, 0.0999, 0.0999])  
tensor([ 0.1001, 0.1001, 0.0999, 0.1000, 0.1001, 0.1000, 0.0999,  
  0.0999, -0.9001, 0.0999])  
tensor([ 0.1001, 0.1001, 0.0999, 0.1000, 0.1001, 0.1000, -0.9001,  
  0.0999, 0.1000, 0.1000])  
tensor([ 0.1001, 0.1001, 0.0999, 0.1000, 0.1001, 0.1000, -0.9000,  
  0.0999, 0.1000, 0.1000])  
tensor([ 0.1001, 0.1001, 0.0999, 0.1000, 0.1001, -0.9000, 0.1001,  
  0.0999, 0.1000, 0.1000])  
tensor([ 0.1001, 0.1001, 0.0999, 0.1000, 0.1001, 0.1001, 0.1001,  
  0.0999, 0.1000, -0.9000])  
tensor([ 0.1001, 0.1001, 0.0999, -0.9000, 0.1001, 0.1001, 0.1001,
```

Random Initialization is Better...

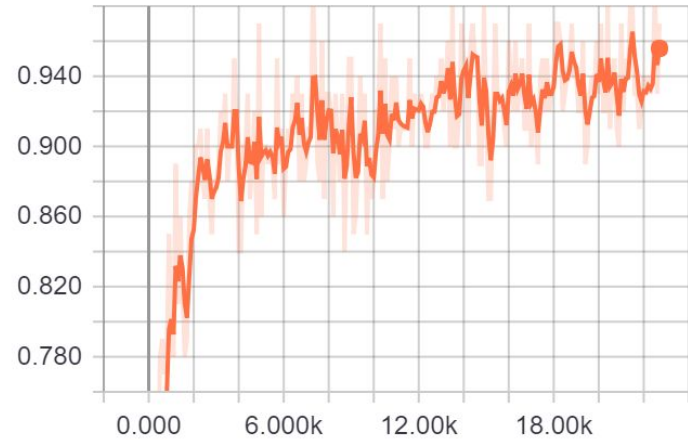
Random initialization

accuracy



What we want

accuracy



How should we scale with layer size?

Intuition:

We may want to change layer size

Should not change variance

Still Bad

Random Initialization scales poorly – with n weights in a given layer, the variance in output is:

$$\begin{aligned} \text{Var}(y) &= \text{Var}\left(\sum_i w_i x_i\right) \\ &= \sum_i \text{Var}(w_i x_i) && \text{(Independence)} \\ &= \sum_i E[x_i]^2 \text{Var}(w_i) + \text{Var}(w_i) \text{Var}(x_i) + E[w_i]^2 \text{Var}(x_i) \\ &= \sum_i \text{Var}(w_i) \text{Var}(x_i) && \text{(Assuming } E[x_i] = 0\text{)} \\ &= \sum_i \frac{1}{12} \times \text{Var}(x_i) && \text{(By variance of a uniform)} \\ &= \frac{n}{12} \times \text{Var}(x_i) \end{aligned}$$

Better Initialization

We want weight initialization methods where the variance of a single weight should scale inversely with width:

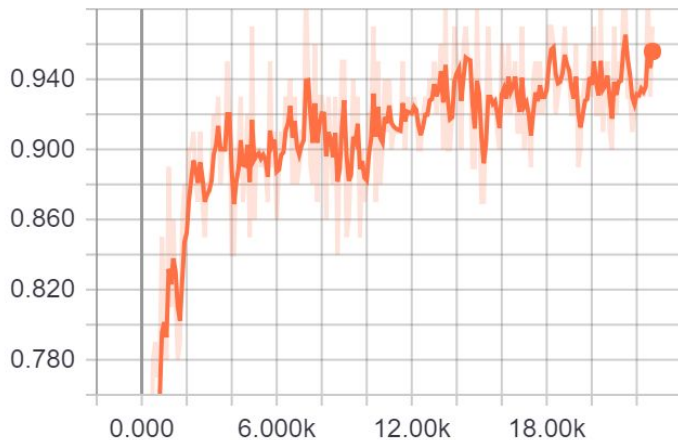
Xavier initialization (tanh, linear, sigmoid) : $w_i \sim N(0; \frac{1}{\sqrt{n}})$ | $Var(w_i) = \frac{1}{n}$

He et. al. initialization (ReLU) : $w_i \sim N(0; \frac{1}{\sqrt{2n}})$ | $Var(w_i) = \frac{2}{n}$

Xavier Initialization: PyTorch Default

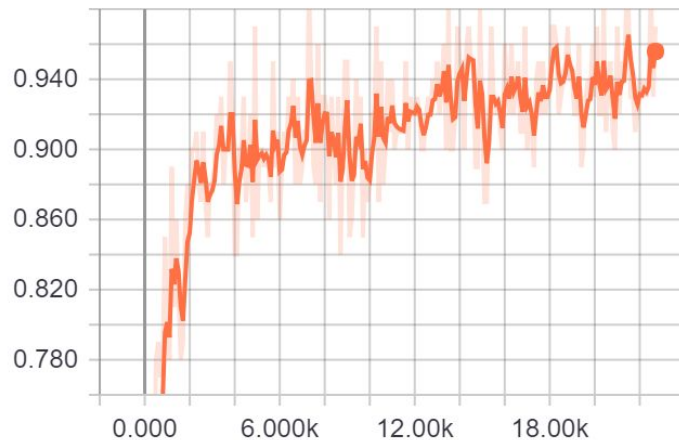
Xavier initialization

accuracy



What we want

accuracy



Briefly back to transfer functions

ReLU can produce (locally) vanishing gradients

Consider LReLU

Re-asking the question question

Bad:

NLP

Vision

Good:

Quantify how much taking a Viking boatmaking course increases lifetime earnings.

Build an NLP system that runs through a deck of google slides, crosschecks them with r/sarcasm and then makes them funny

Describe a question you would like to answer



How could lecture 4 have been more useful (also this is anonymous now)

