Neural Networks 401

COMP 4420 / 5420 Natural Language Processing

Vladislav Lialin, Text Machine Lab

Administrative

- Python Code style quiz grades are released
  - Next step: working on your quiz mistakes
  - Complete this google form before the next class
    https://forms.gle/VB8Lmyxv11bUfS1C9
- Homework 3 is extended until the next class
  - If you already submitted, take a second look and fix your mistakes
  - Make sure you **understand** it, you will need to do similar things on your final exam
  - Extra points for implementing new activation functions.
    Try GELU, Swish, invent your own.
    Make sure to check the gradients numerically using `eval_numerical_gradient_array()`

What you should remember after this lecture

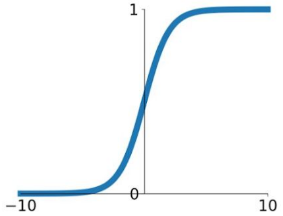
- Momentum and ADAM optimizers
- Batch Normalization
- Dropout
- Neural networks are hard to train

Neural Networks 301
Recap

Activation functions

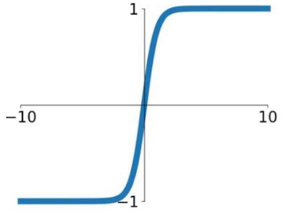
Sigmoid

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



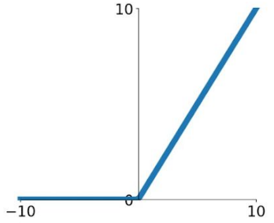
tanh

$\tanh(x)$



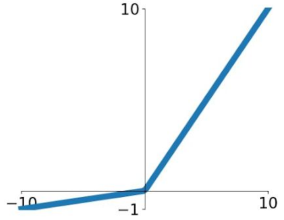
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

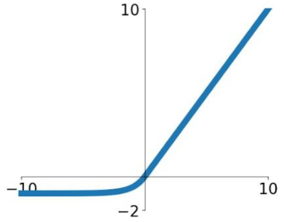


Maxout

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

ELU

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

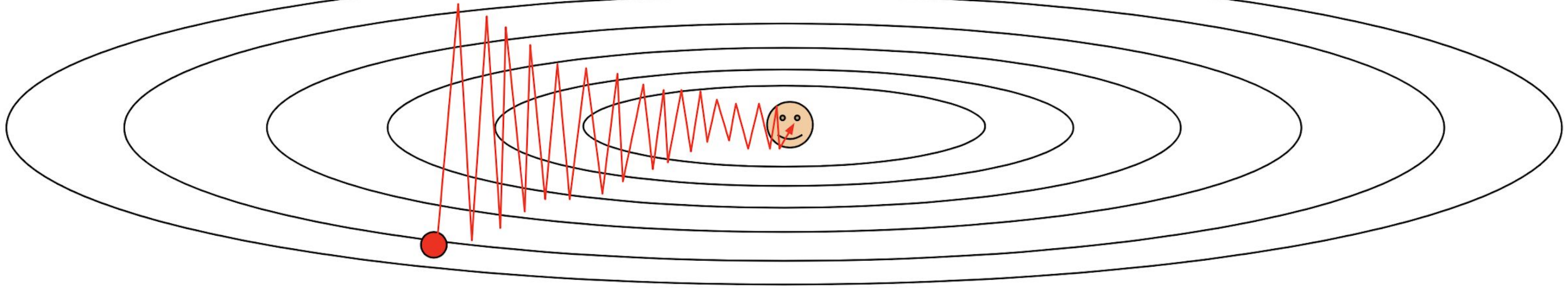


Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Optimization

SGD problems

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction

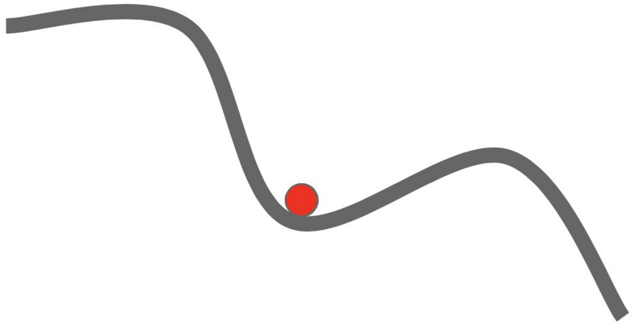


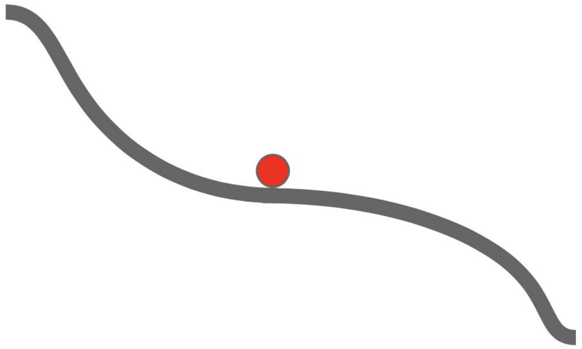
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Slide credit: Stanford CS231n
Image credit: Stanford CS231n

SGD Problems

- Saddle points
  - Gradient is near-zero, but this is not a minimum
  - Saddle points are extremely common
- Local minima
  - Not a problem in practice
  - Reasons to believe that SGD finds a global minimum for neural networks with enough parameters
- Noisy gradients
  - Just use larger batches





Momentum

SGD

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

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

SGD+Momentum

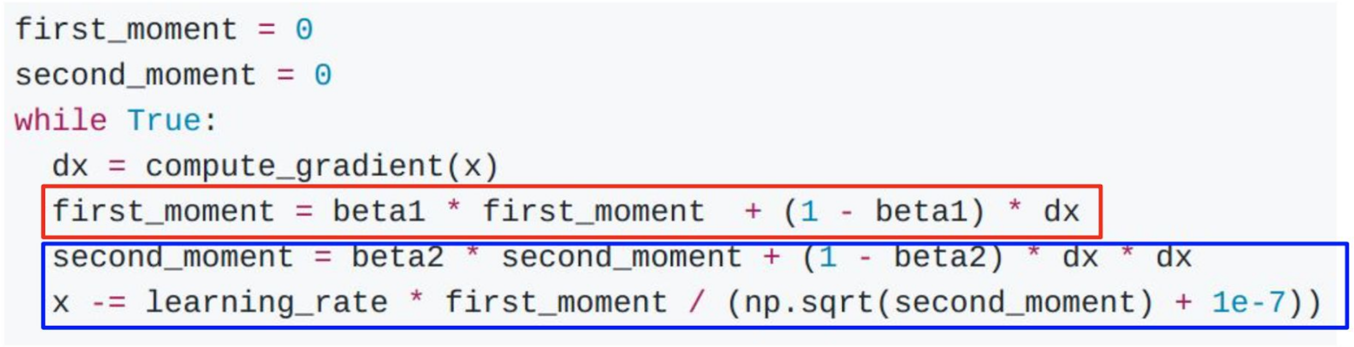
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
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

Slide credit: Stanford CS231n
Image credit: Stanford CS231n
On the importance of initialization and momentum in deep learning, Sutskever et al, 2013

Momentum + RMSProp = Adam



```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

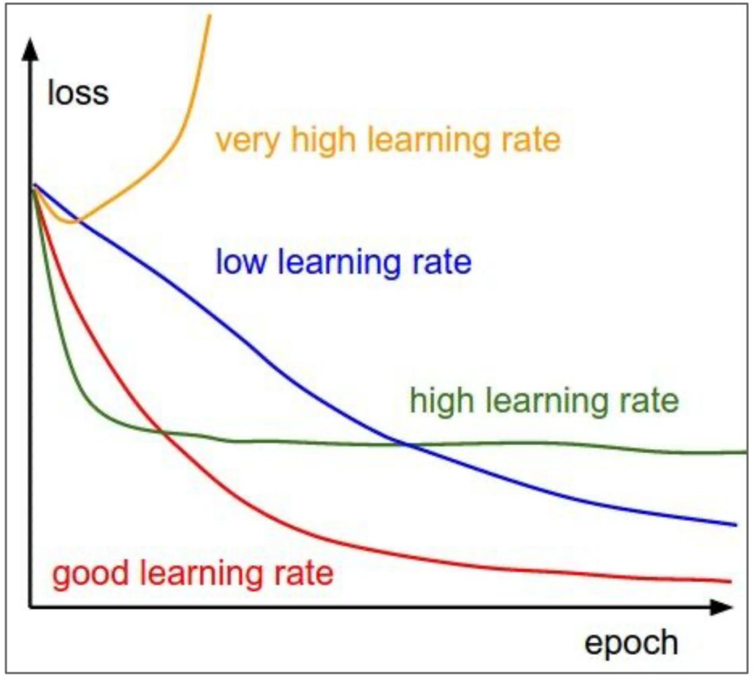
AdaGrad / RMSProp

Sort of like RMSProp with momentum

Q: What happens at first timestep?

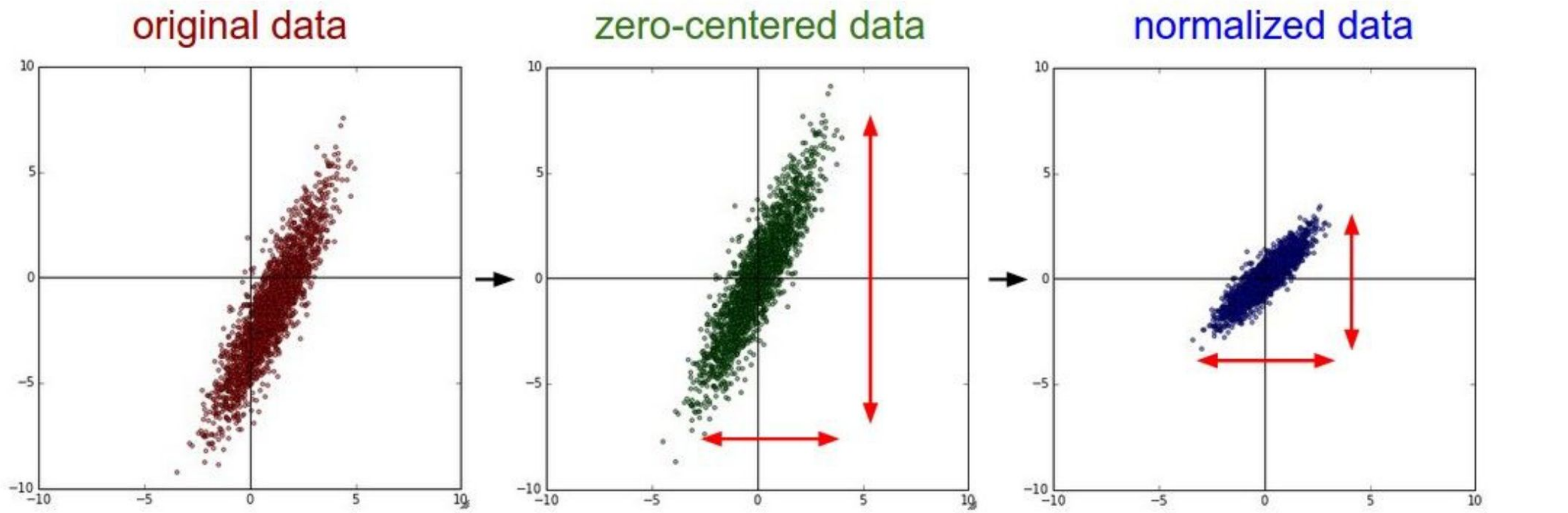
How to chose learning rate: heuristics

- Reasonable defaults:
  - 1e-3 for smaller networks (<100M parameters)
  - 1e-5 for larger networks (>100M parameters)
  - **These defaults do not always work**
  - 1e-7 and smaller is a terrible idea — too small, almost no parameter updates will be done because of numerical issues



Batch Normalization

Data preprocessing



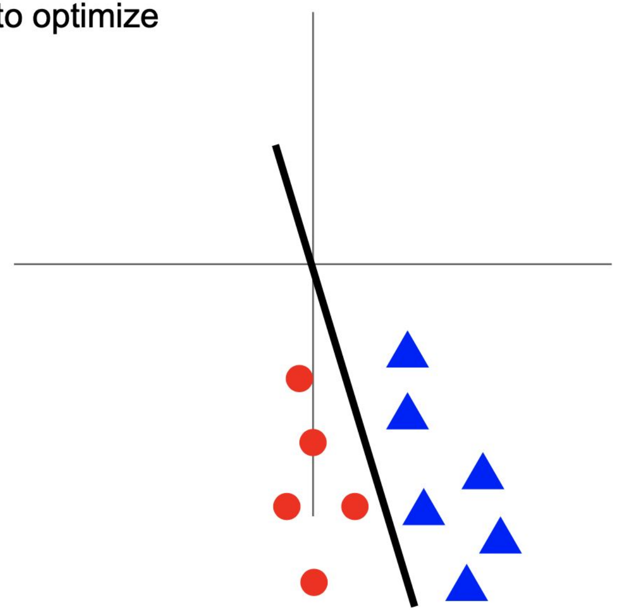
```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

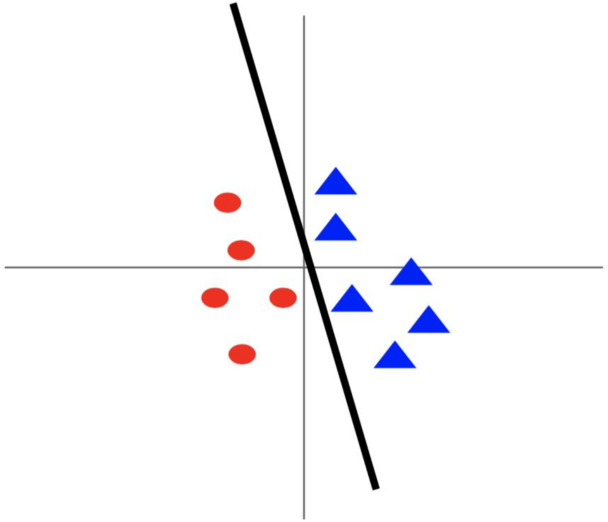
Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Data preprocessing

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



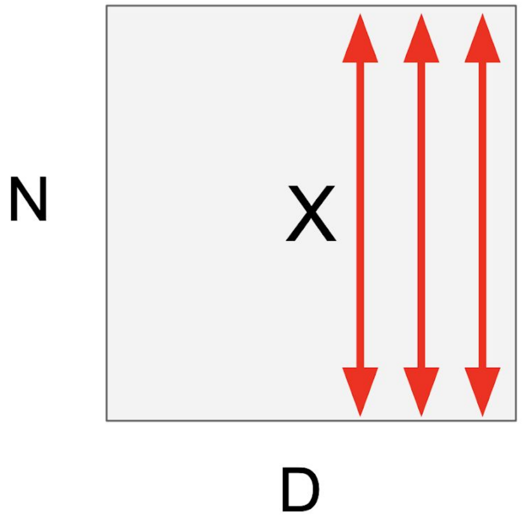
Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Batch normalization

- Preprocessing only normalizes the first layer input
- What if we can normalize the inputs of each layer?
  - How to get statistics (mean and variance)?
  - Estimate per-batch statistics during training
  - Keep running means during training
  - <u>Use these running means at test time</u>

Batch normalization

Input: $x : N \times D$



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

Per-channel mean, shape is D

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

Per-channel var, shape is D

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

Normalized x, Shape is N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Slide credit: Stanford CS231n
Image credit: Stanford CS231n
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Ioffe and Szegedy, 2015

Batch normalization

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

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

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

Per-channel mean, shape is D

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

Per-channel var, shape is D

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

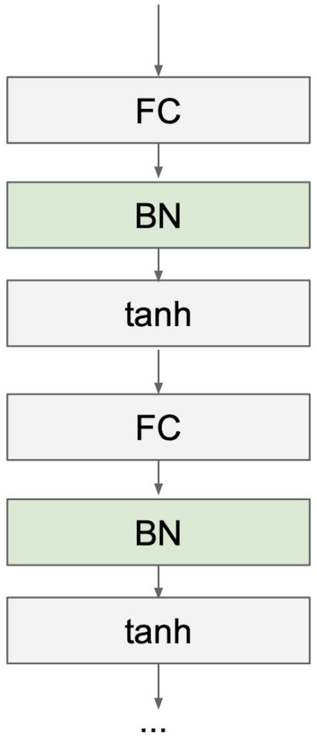
Normalized x, Shape is N x D

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

Output, Shape is N x D

Slide credit: Stanford CS231n
Image credit: Stanford CS231n
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Ioffe and Szegedy, 2015

Batch normalization



- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Slide credit: Stanford CS231n
Image credit: Stanford CS231n
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Ioffe and Szegedy, 2015

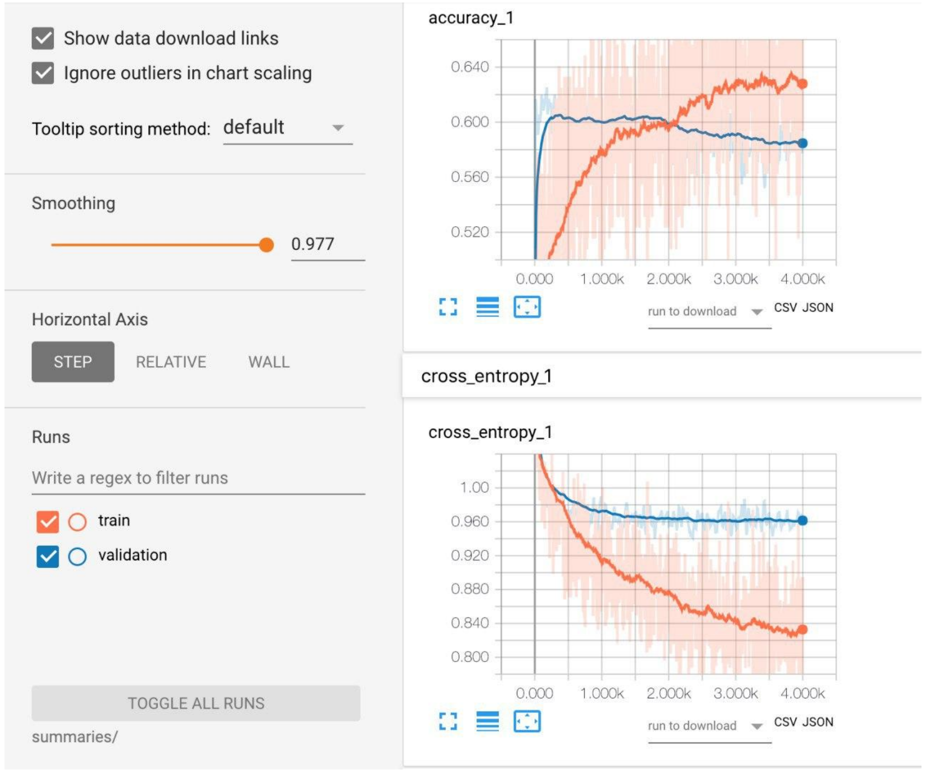
Extra

- Layer Normalization - BN for texts
  - Just batch normalization, but statistics comes from feature dimension
  - Same at train and test time — Does not require running moments
  - Layer Normalization, Lei Ba et al., 2016
- Theory of batch normalization is still controversial:
  - Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Ioffe and Szegedy, 2015
  - How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift), Santurkar et al., 2018
  - A Mean Field Theory of Batch Normalization, Yang et al., 2019

Slide credit: Stanford CS231n
Image credit: Stanford CS231n
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Ioffe and Szegedy, 2015

Regularization

~~Overfitting~~ Generalization issues



Show data download links ✓
Ignore outliers in chart scaling ✓

Tooltip sorting method: default

Smoothing 0.977

Horizontal Axis
[✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs
✓ train
✓ validation

TOGGLE ALL RUNS
summaries/

L2 regularization

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

In common use:

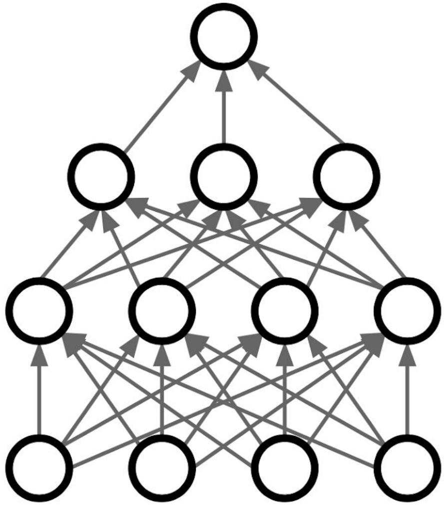
L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

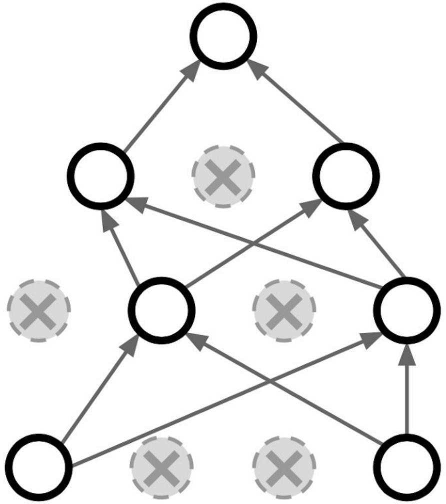
- Fully analogous to linear models
- But typical lambda is smaller: [1e-6, 1e-3]
- If you want to use it with ADAM, use AdamW instead

Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Dropout

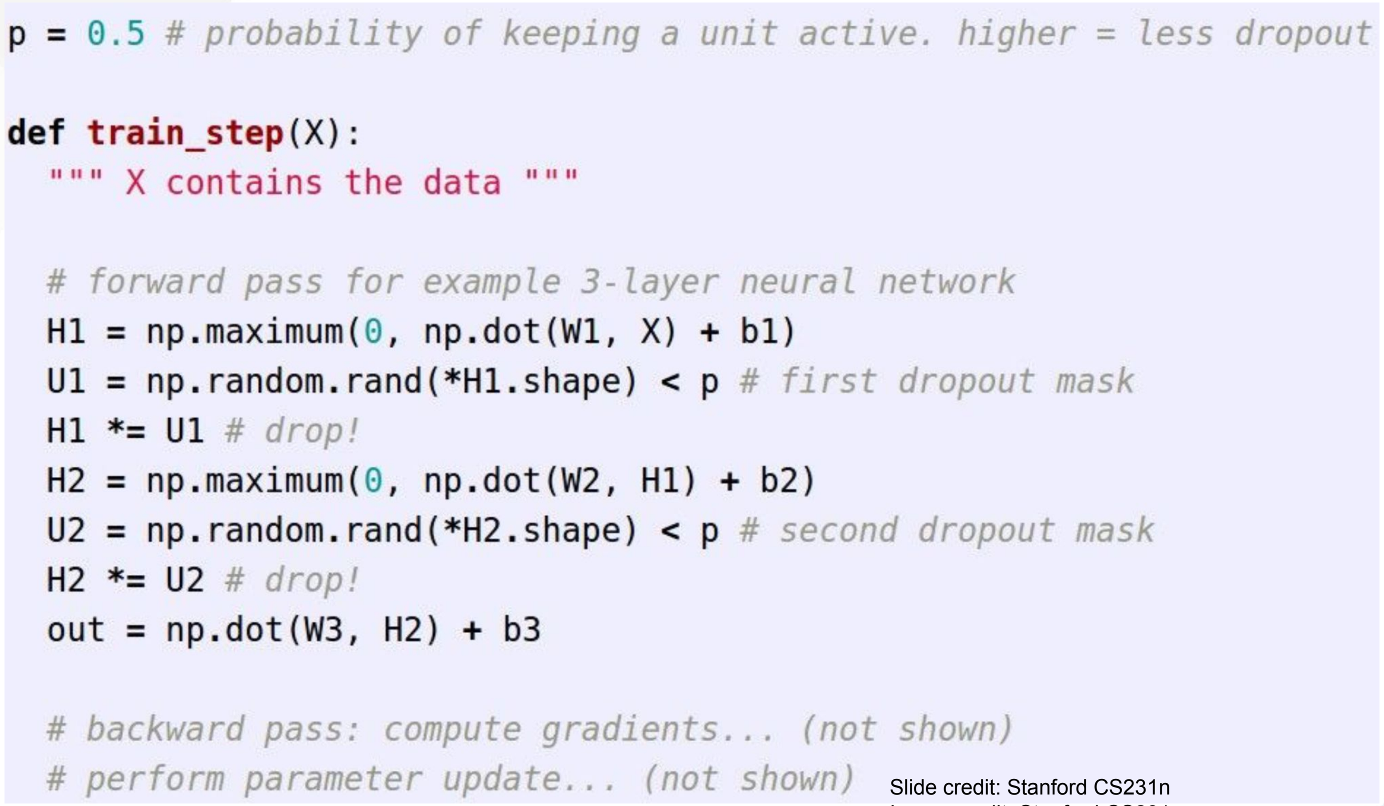
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common





Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Dropout: implementation example

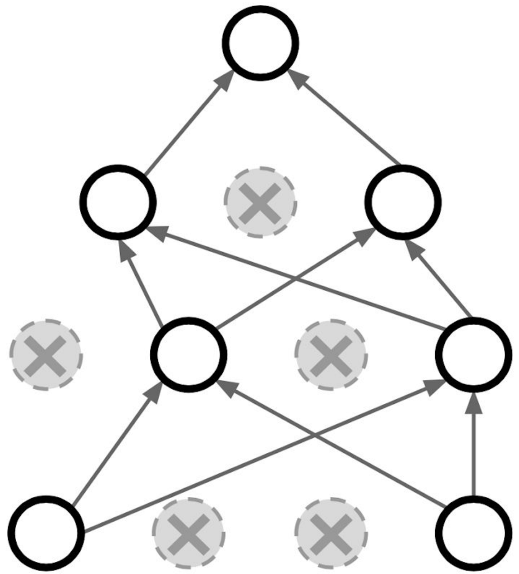


```
p = 0.5 # probability of keeping a unit active. higher = less dropout

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)
```

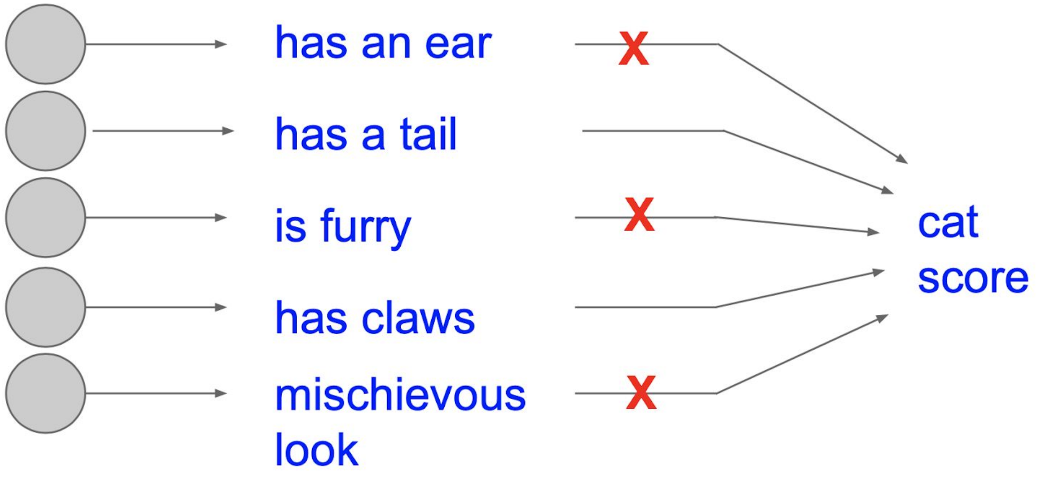
Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Dropout

How can this possibly be a good idea?



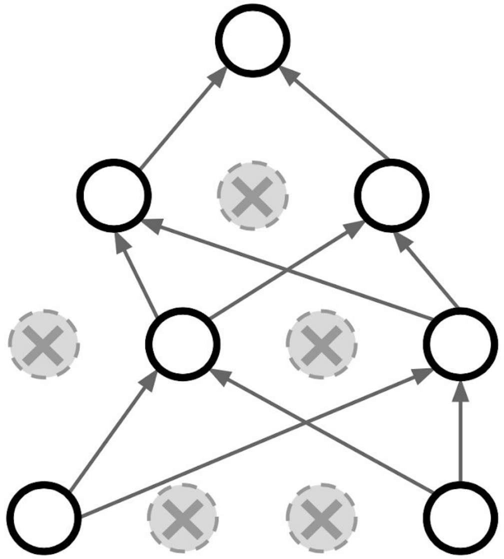
Forces the network to have a redundant representation; Prevents co-adaptation of features



Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

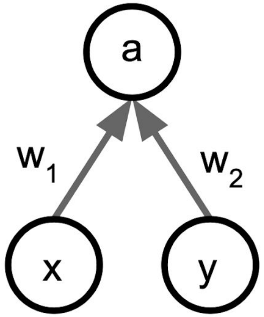
Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Dropout: test time

- Set dropout probability to zero and correct distribution mismatch

Want to approximate the integral

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



Consider a single neuron.

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

During training we have:

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

At test time, **multiply** by dropout probability

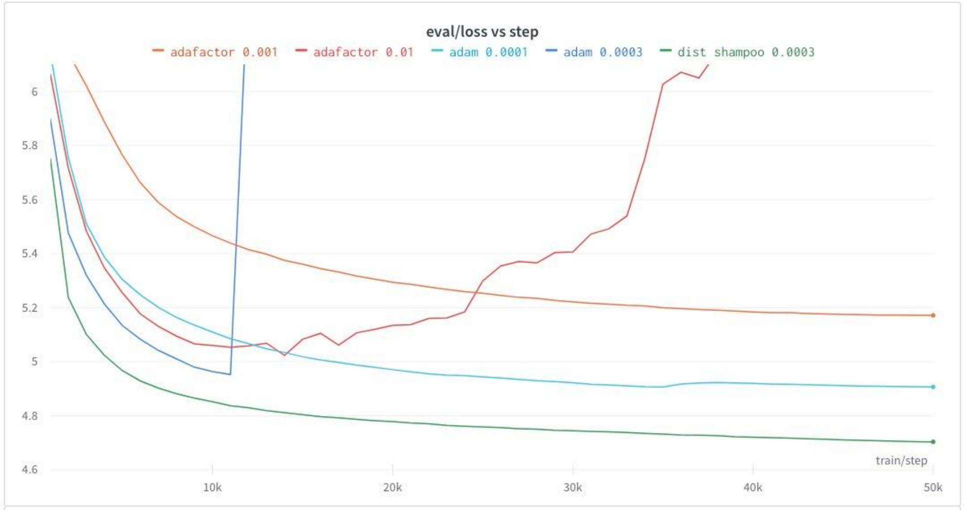
Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Babysitting the learning process

The model may diverge

Main reasons:

1. Preprocessing errors
2. LR is too high
3. Bad init
4. Small batch size
5. Other hyperparameters
6. Numerical issues
7. Bad luck 🤷‍♂️



My model does not converge checklist

- ❏ Overfit one batch of data
  - ❏ Turn off regularization
  - ❏ Reach training accuracy of 100%
- ❏ Check model inputs
  - ❏ Right before the model
  - ❏ Convert tensors back to text
  - ❏ Check class distribution
- ❏ Check optimization
  - ❏ **Find a better lr**
  - ❏ Increase batch size
  - ❏ Look into gradient histograms
  - ❏ Change your initialization

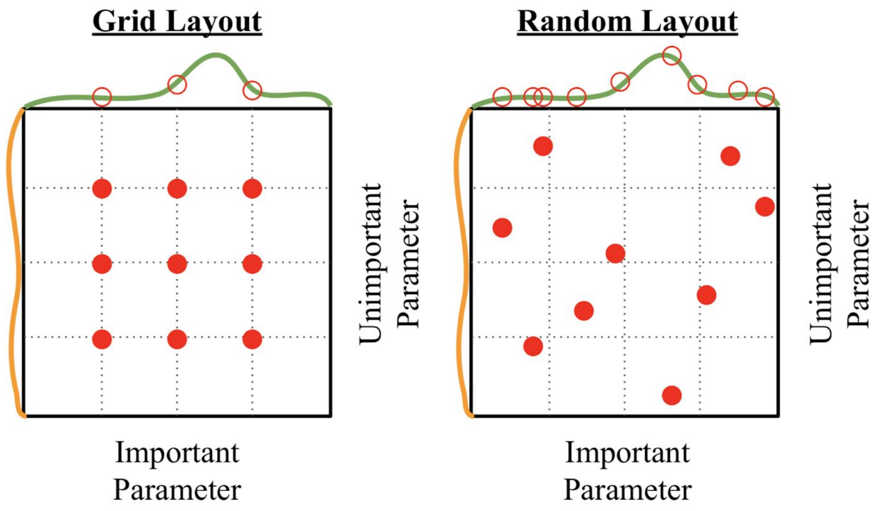
My results are bad

- ❏ **Check for bugs in data pre-processing**
- ❏ Read A Recipe for Training Neural Networks
- ❏ Use hacks (e.g. from fast.ai course)
- ❏ Use a larger model (we'll talk more about it in future lectures)
- ❏ Use ensembles
  - ❏ Train multiple models
  - ❏ Average their results at test time

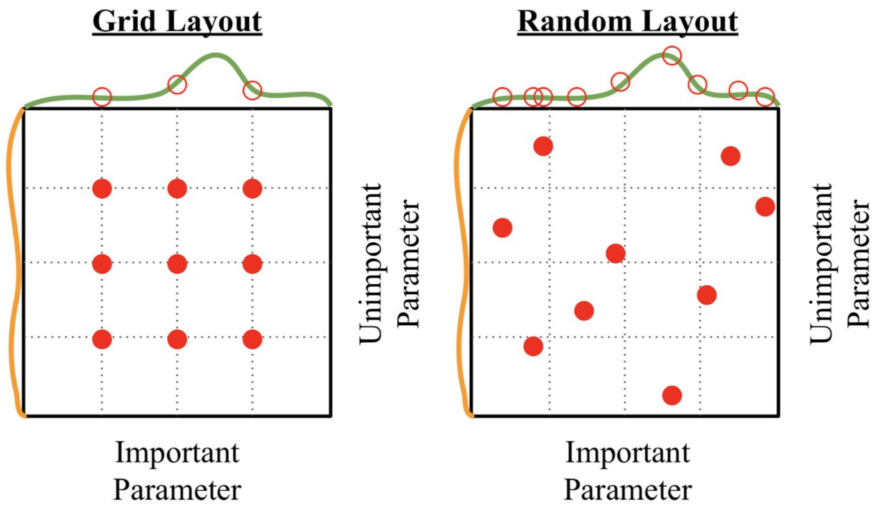
Hypeparameter search

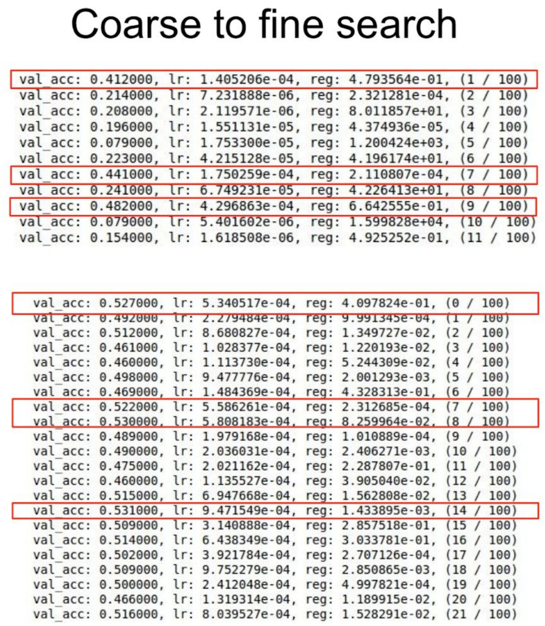
- Manual is almost always best
- Strategies for automated hparam search
  - Grid search
  - Random search
  - Bayessian optimization

Grid search vs random search



Grid search vs random search




Coarse to fine search

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

```
  val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
  val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
  val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
  val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
  val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
  val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
  val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
  val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
  val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
  val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
  val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
  val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
  val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
  val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
  val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
  val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
  val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
  val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
  val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
  val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
  val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
  val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

Hardware

GPU

- ~10-20 times faster than CPU
- Specialized for doing many similar operations at the same time
    - E.g., matrix multiplication
- Don’t like sequential operations
- Must have for deep learning
- Free GPU (with limitations):
        - Google Colab
        - Paperspace Gradient
        - Kaggle Kernels
        - CS dep labs

Specialized hardware: TPUs

- Usually is cheaper in terms of per-watt or per-dollar performance
- Scales better than GPUs (thousands of devices working in parallel)
- PyTorch TPU support is not great, use Jax instead
- Free TPU: Google Colab

Why is it so cool?

- Hardware implementation of nonlinearities
- Faster memory speeds
- Other hacky optimizations specific to DL

Homework

Homework

- Getting familiar with PyTorch
  - Pytorch is similar to numpy
  - `.backwards()` does backpropagation for you automatically
  - `torch.nn` implements common neural networks modules for you
  - Official PyTorch tutorial: https://pytorch.org/tutorials/beginner/basics/intro.html
- Training fully-connected network in PyTorch
  - IMDB
  - Count Vectors
  - Neural Network (instead of a linear model)
- Way easier than homework 2

Homework. Extra materials

- Defining a neural network in PyTorch
- What is `torch.nn` really?
- PyTorch Tools, best practices & style guide