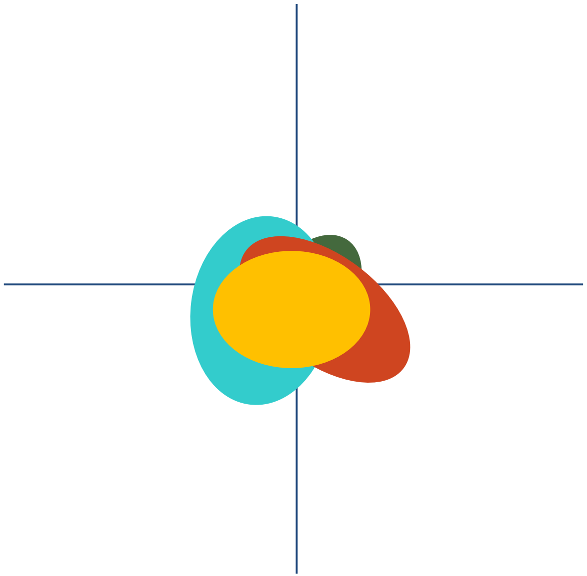
Activation, Initialization, Preprocessing, Dropout, Batch Norm

Covariate Shift and Batch Norm

Covariate Shifts

Randomly sampling mini-batches: Training assumes similar distribution!



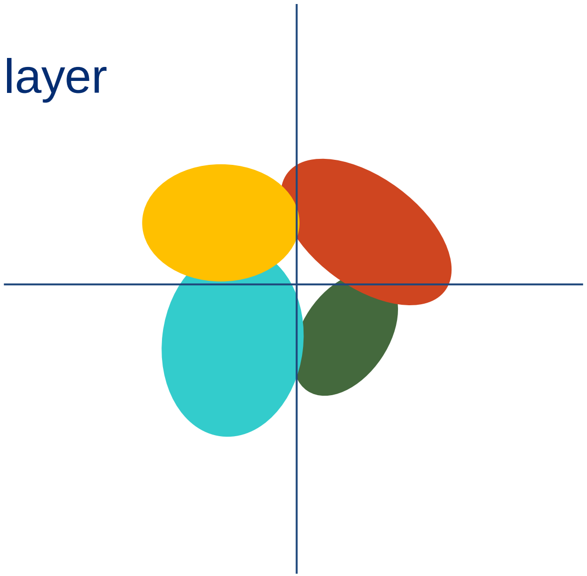
Covariate Shifts

Randomly sampling mini-batches: Training assumes similar distribution!

In practice (and although random), each mini-batch will have different distribution

→ Covariate shift

→ Can happen in **each** layer



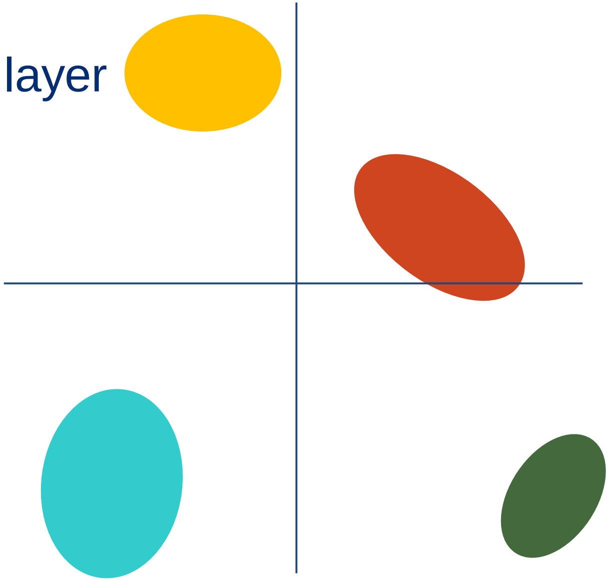
Covariate Shifts

Randomly sampling mini-batches: Training assumes similar distribution!

In practice (and although random), each mini-batch will have different distribution

→ Covariate shift

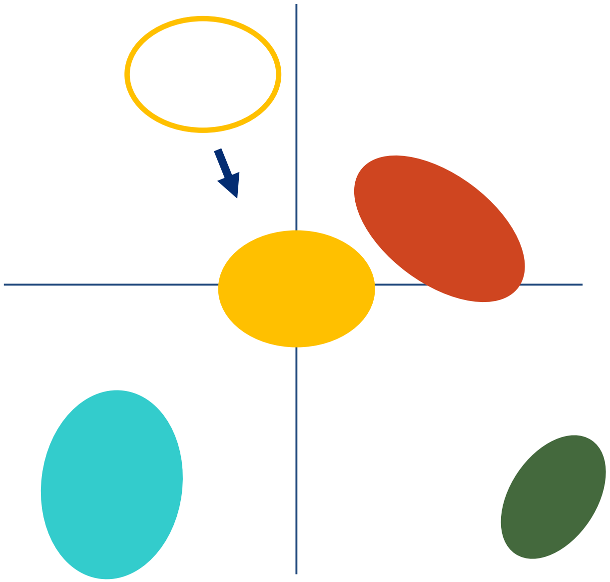
→ Can happen in **each** layer



→ Shifts can be large and can negatively affect training!

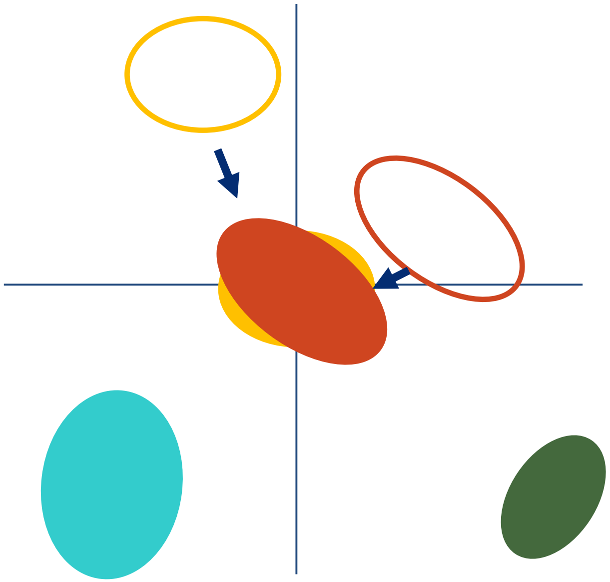
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



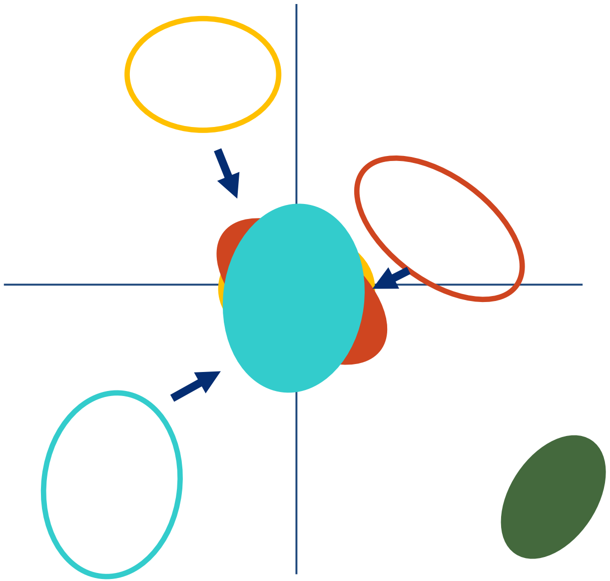
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



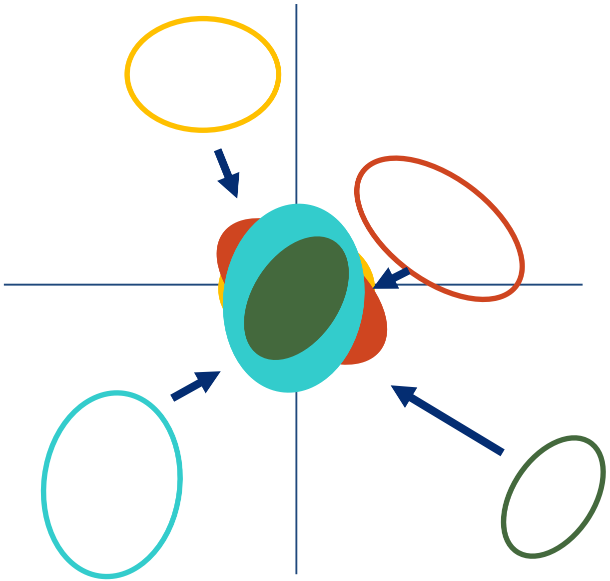
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



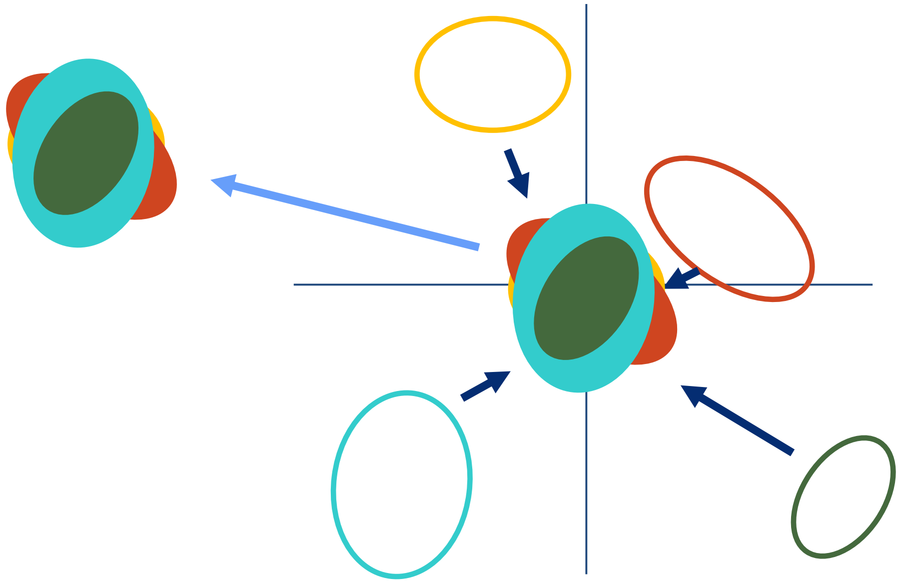
Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



Move Batches to Standard Location

Eliminate covariate shift by “moving” batches to zero mean and unit standard dev



→ Then, move entire collection to desirable location: **Batch normalization**

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization

- If we want unit Gaussian activations, let's make them that!

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

This function is differentiable (backprop!)

- Rather then pre-conditioning data and hoping that nice properties are preserved, at each layer we re-condition during every forward pass

Batch Normalization

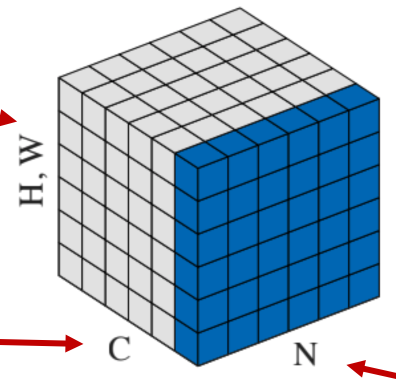
- If we want unit Gaussian activations, let's make them that!

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

This function is differentiable (backprop!)

- Rather then pre-conditioning data and hoping that nice properties are preserved, at each layer we re-condition during every forward pass

Spatial dimensions

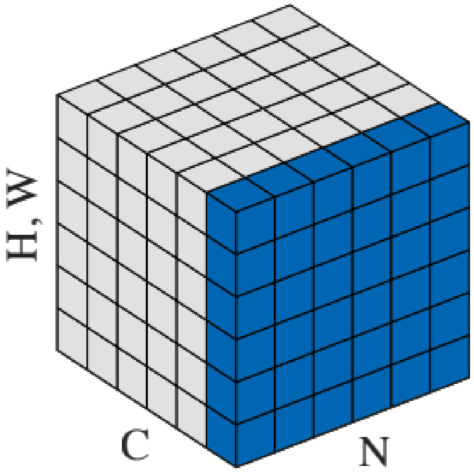


Channels

Instances in batch

Wu, Y. and He, K., 2018. Group normalization. arXiv preprint arXiv: 1803.08494.

Batch Normalization



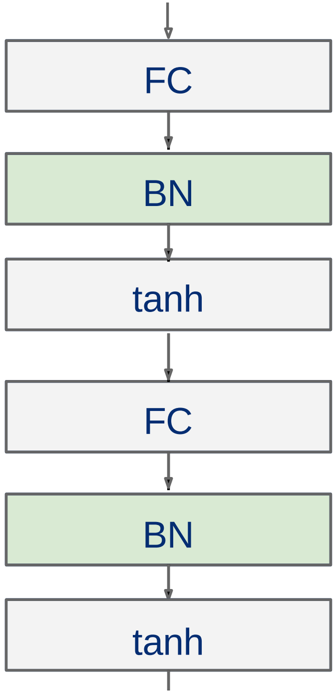
1. Compute empirical mean and variance for each channel
   $E[x^{(k)}], \mathrm{Var}[x^{(k)}]$

2. Normalize to unit Gaussian
   $$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Wu, Y. and He, K., 2018. Group normalization. arXiv preprint arXiv: 1803.08494.

Batch Normalization

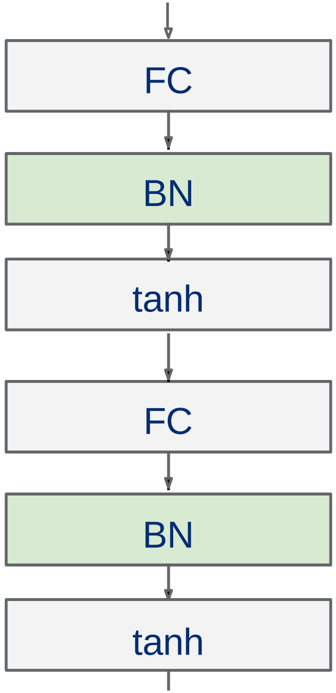
Usually inserted right after fully connected
or convolutional layers, right before activation.



1. Compute empirical mean and variance for each channel
   $$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

2. Normalize to unit Gaussian
   $$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization



Usually inserted right after fully connected or convolutional layers, right before activation.

1. Compute empirical mean and variance for each channel
   $$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

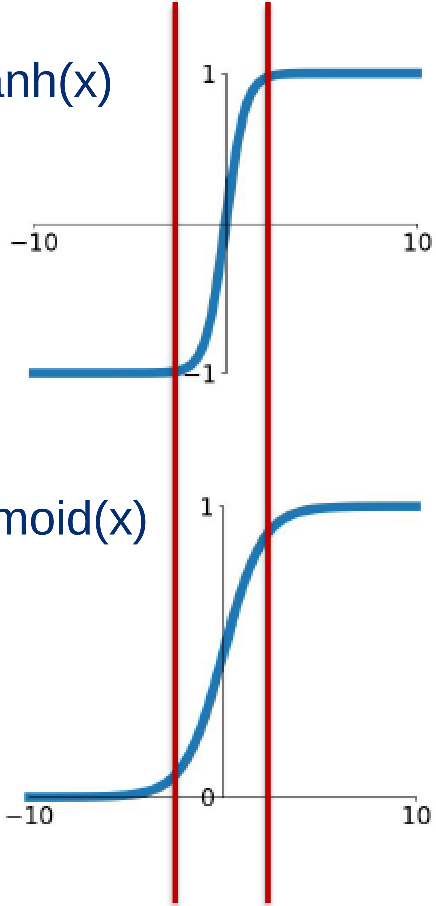
2. Normalize to unit Gaussian
   $$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Q: Is unit Gaussian activation necessarily what we want?

Batch Normalization

tanh(x)

sigmoid(x)



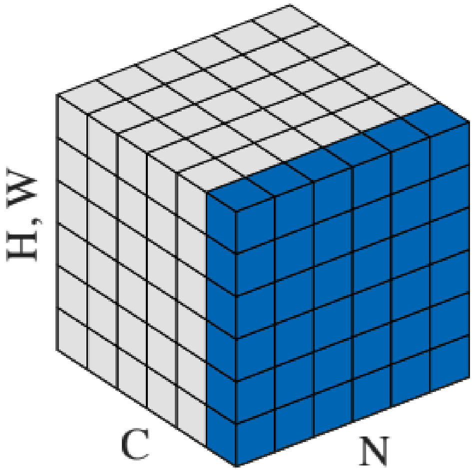
Consider tanh or sigmoid activation

→ Batch normalization will limit the activation to the linear regime of these activation functions!

→ In such case, negatively affects performance

There are other cases where you also would not want BN, e.g. when magnitude matters.

Batch Normalization



1. Compute empirical mean and variance for each channel
$$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

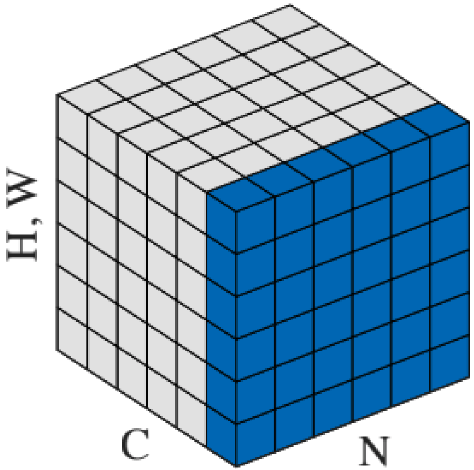
2. Normalize to unit Gaussian
$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

3. Squash output to beneficial range
$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

These are parameters and are learned during training.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization



1. Compute empirical mean and variance for each channel
$$E[x^{(k)}], \mathrm{Var}[x^{(k)}]$$

2. Normalize to unit Gaussian
$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

3. Squash output to beneficial range
$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

These are parameters and are learned during training.

Network can learn identity!

$$\gamma^{(k)} = \mathrm{Var}[x^{(k)}]$$
$$\beta^{(k)} = E[x^{(k)}]$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- Improves gradient flow through network and allows for higher learning rates
  - Avoids saturating activations
  - Avoids exploding/vanishing gradients
  - Higher learning rates usually produce larger weights leading to explosion
    → Can be avoided here since re-normalized
- Reduces strong dependence on initialization
- Acts as regularization
  - Single instance is now seen in conjuncture with other samples of the batch
  - Network outputs per sample no longer deterministic

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Batch Normalization During Testing

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Q: What to do at testing time?

Batch Normalization During Testing

Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters

for $k = 1 \ldots K$ **do**

// For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1} \mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$

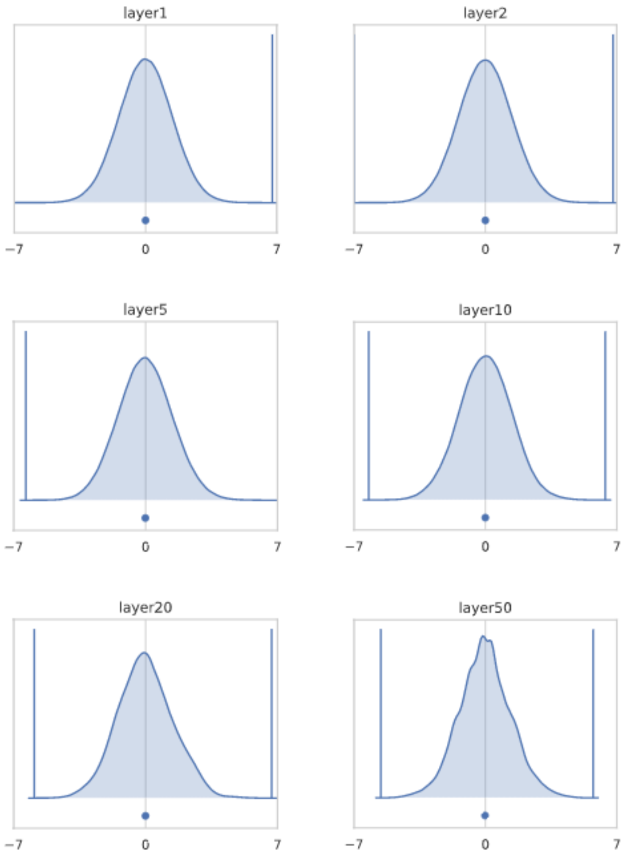
end for

Q: What to do at testing time?

Compute average mean and standard deviation across multiple batches, then save these values for inference.

Batch Norm: New Insights

- Is it really about covariate shift?
- Let's reconsider He initialization
  - Goal: Preserve mean and variance of outputs if marginalized over the weight distribution

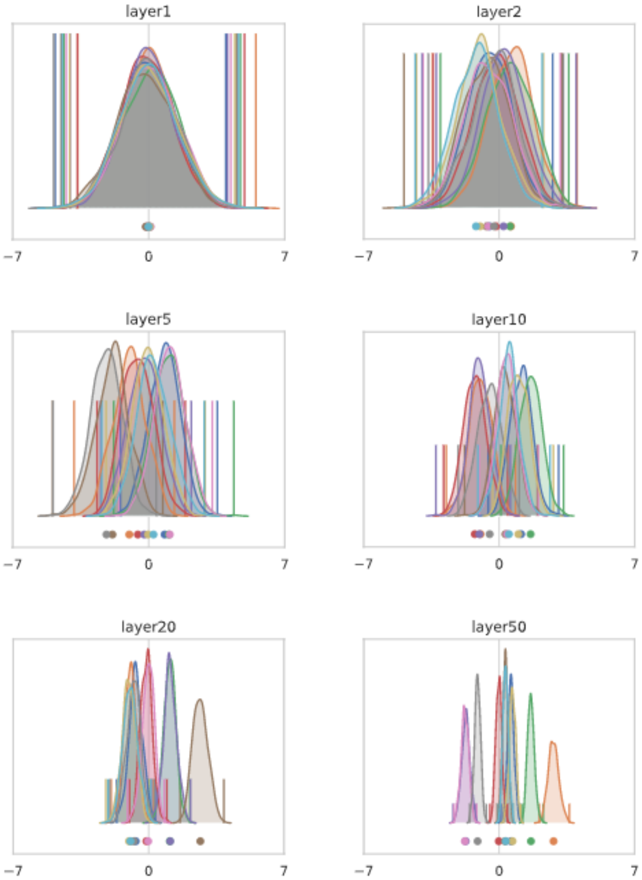


Channel activation at different depths with independent N(0,1) inputs

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- Is it really about covariate shift?
- Let's reconsider He initialization
  - Goal: Preserve mean and variance of outputs if marginalized over the weight distribution
- Every channel has chosen a constant value!
  - Peaked, narrow distribution
  - Most inputs would be classified as the orange class



Channel activation at different depths with independent N(0,1) inputs **split by channel**

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

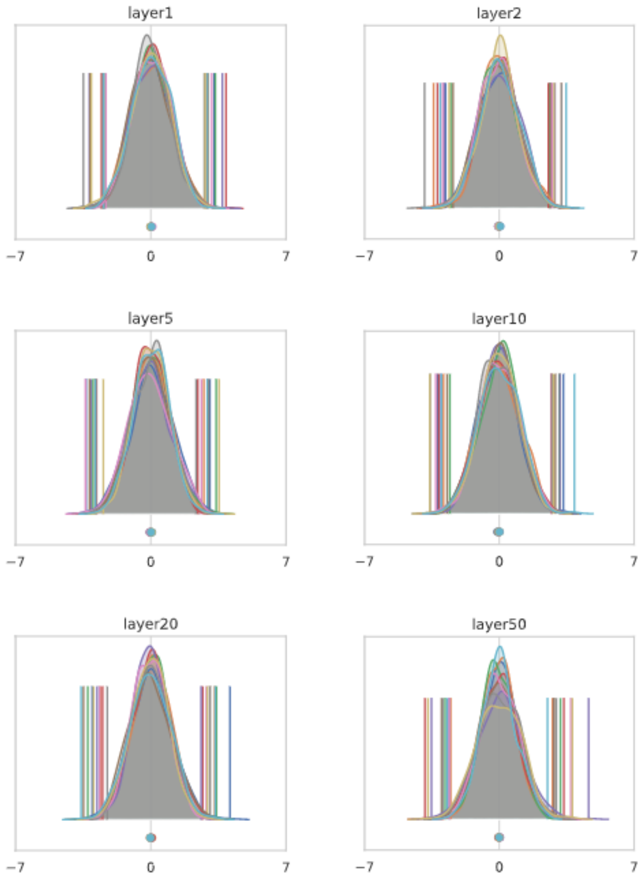
Batch Norm: New Insights

- Is it really about covariate shift?

- Let's reconsider He initialization
  - Goal: Preserve mean and variance of outputs if marginalized over the weight distribution

- Every channel has chosen a constant value!
  - Peaked, narrow distribution
  - Most inputs would be classified as the orange class

- Removing ReLU: Problem disappears
  - Non-zero channel means
  - Decreasing variance due to increasing mean (see blog)



Channel activation at different depths with independent N(0,1) inputs

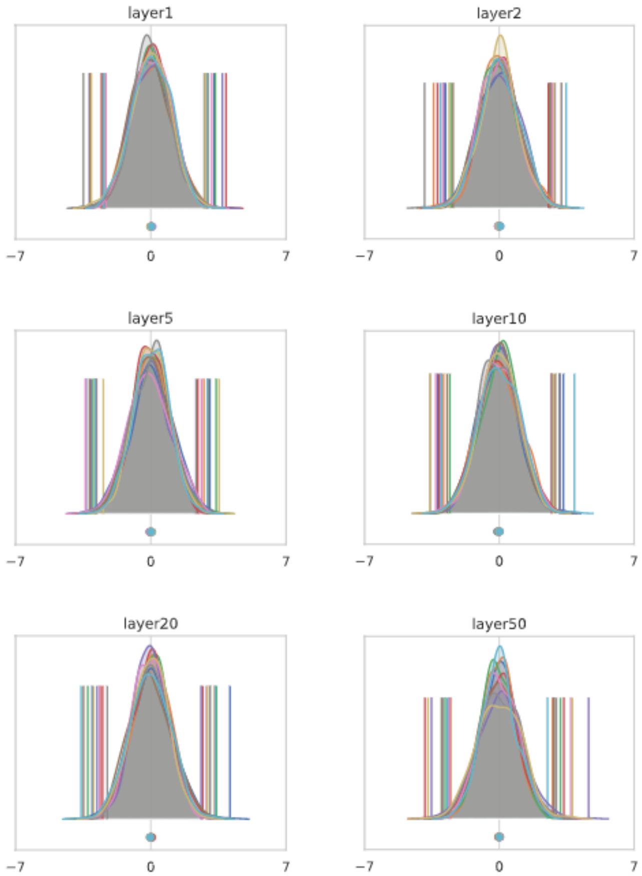
split by channel, no ReLU

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- → Without batch norm
  - → Standard initialization leads to bad configurations
  - → Network will predict near constant outputs
- → Batch norm fixes this by design

What happens during training?



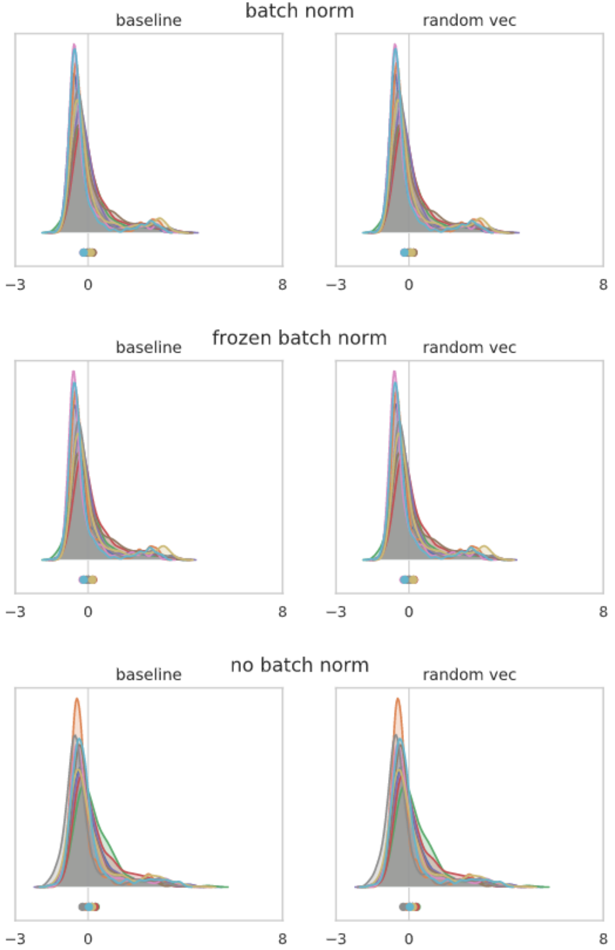
Channel activation at different depths with independent N(0,1) inputs

split by channel, no ReLU

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

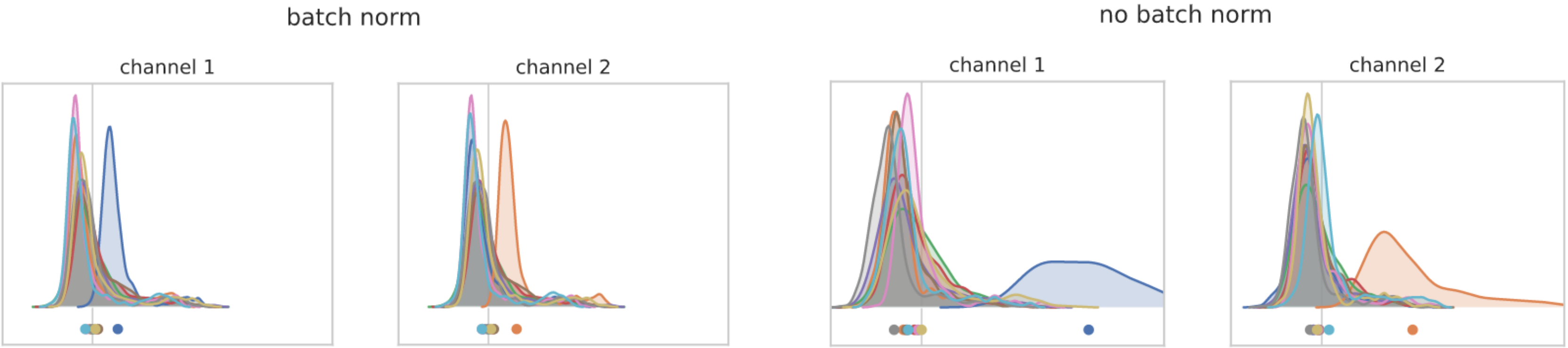
- Random perturbation of the weight
  Strength of 1% of parameter vector length
  - Similar output distributions
  - Main mode and second smaller mode: Network starting to make confident predictions



Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

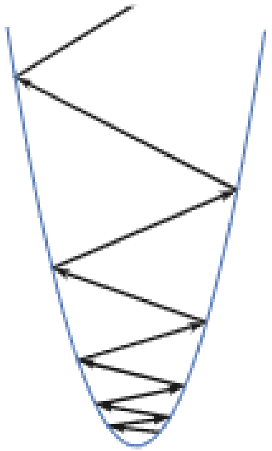
- **Targeted** perturbation of the weight
  Strength of 1% of parameter vector length
  Gradient of channel mean
  - Network will predict perturbed class in majority of inputs!
  - Internal covariate shift can propagate to external covariate shift in non-batch norm networks!



[Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).](https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/)
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- **Targeted** perturbation of the weight
  Strength of 1% of parameter vector length
  Gradient of channel mean
  - Network will predict perturbed class in majority of inputs!
  - Internal covariate shift can propagate to external covariate shift in non-batch norm networks!



- What does this mean for optimization?
  - Without batch norm, small perturbations lead to immense increases in loss!
  - This means that we are in a narrow valley-type loss landscape (see also next lecture)

[Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).](https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/)
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Batch Norm: New Insights

- Investigate the Hessian of parameters
  - **Leading eigenvector** (direction of largest curvature)
    → This direction makes SGD spiral out of control
  - Computed via a power method (not important)

- Also, compute gradients w.r.t.
  mean channel activation (as in perturbation)

→ **Compute overlap between eigenvectors and output-mean gradients**

[Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).](https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/)
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

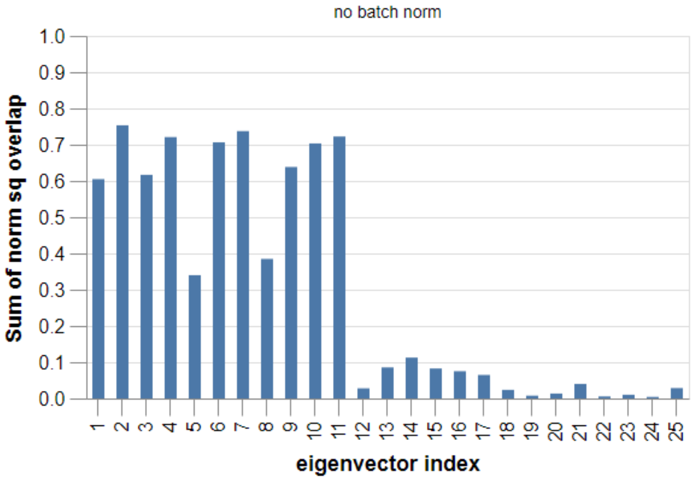
Batch Norm: New Insights

→ **Compute overlap between eigenvectors and output-mean gradients**

- Largest eigenvectors lie almost entirely in the 9-dim subspace spanned by the mean-output gradients!

- **This de-stabilizes SGD optimization!**

- **Batch norm: Smoothens the optimization landscape.**



Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. NeurIPS (pp. 2483-2493).
https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/

Activation, Initialization, Preprocessing, Dropout, Batch Norm

Regularization with Dropout

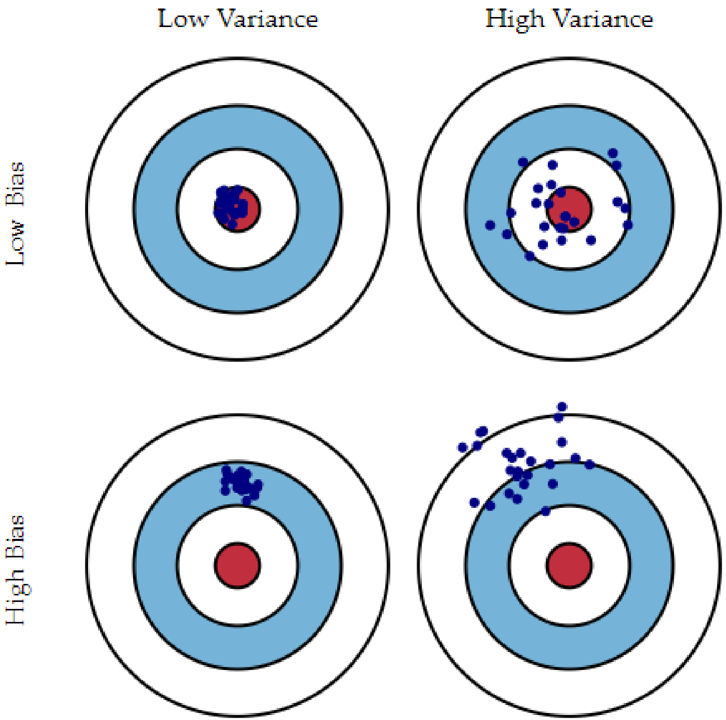
The Bias-Variance Tradeoff and Regularization

Decomposition into bias and variance

$$L(W) = \underbrace{(E[\hat{y}] - y)^2}_{\text{Bias}^2} + \underbrace{E[(\hat{y} - E[\hat{y}])^2]}_{\text{Variance}} + \sigma$$

Adding regularization

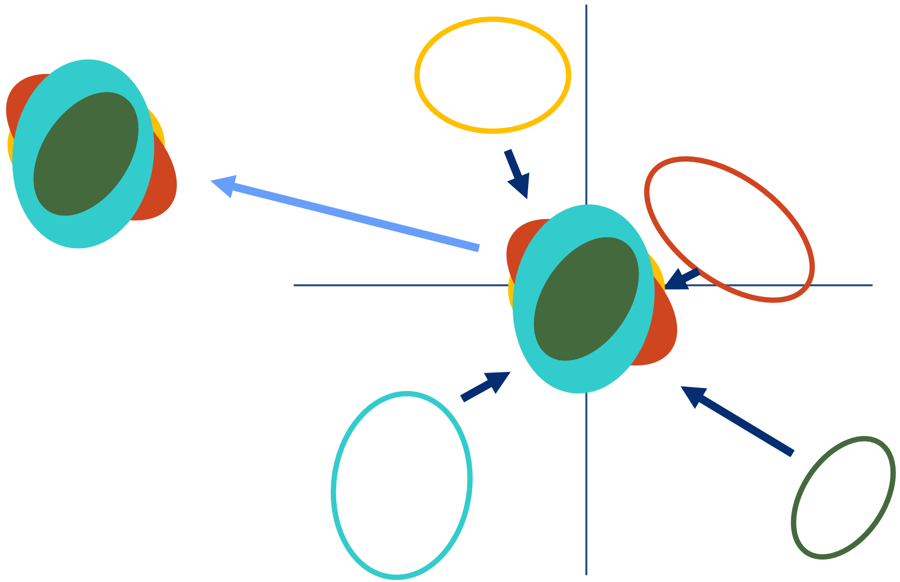
$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)}_{\textbf{Data fidelity}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$



Hastie, T., Tibshirani, R, and Friedman, J. (2017) The Elements of Statistical Learning

Batch Normalization

Regularization “in a funny way” by seeing samples in conjuncture with others



Other approaches

- L2 on weights
- L1 on activations
- Adding noise to inputs

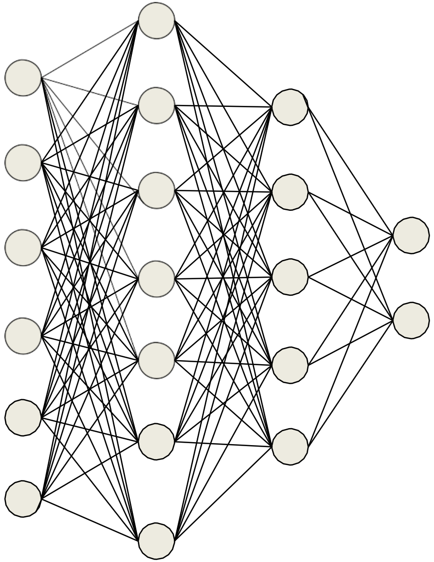
Q: Can we regularize “in a less funny” way?

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Dropout

No, not like this!

Dropout

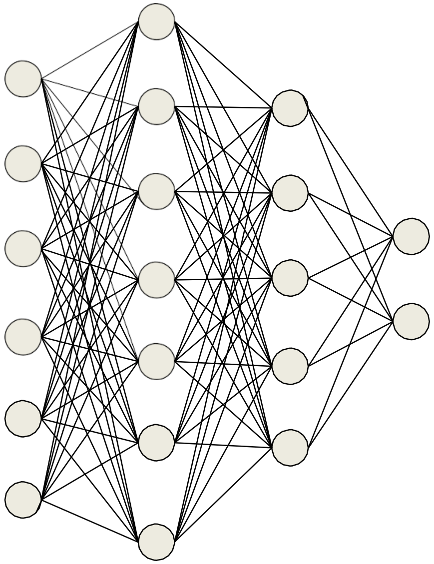


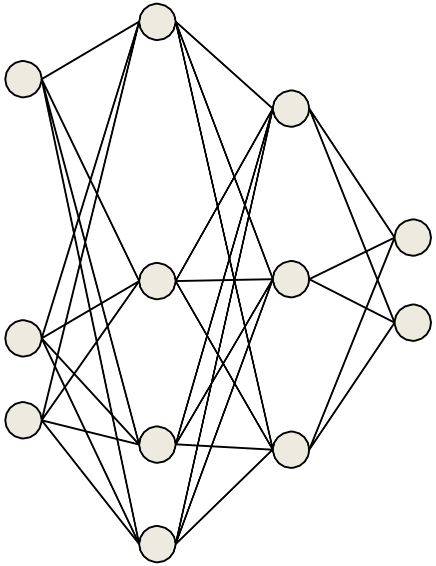
During training

- At each iteration, in each layer, “knock out” each neuron with probability 1-α

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.

Dropout



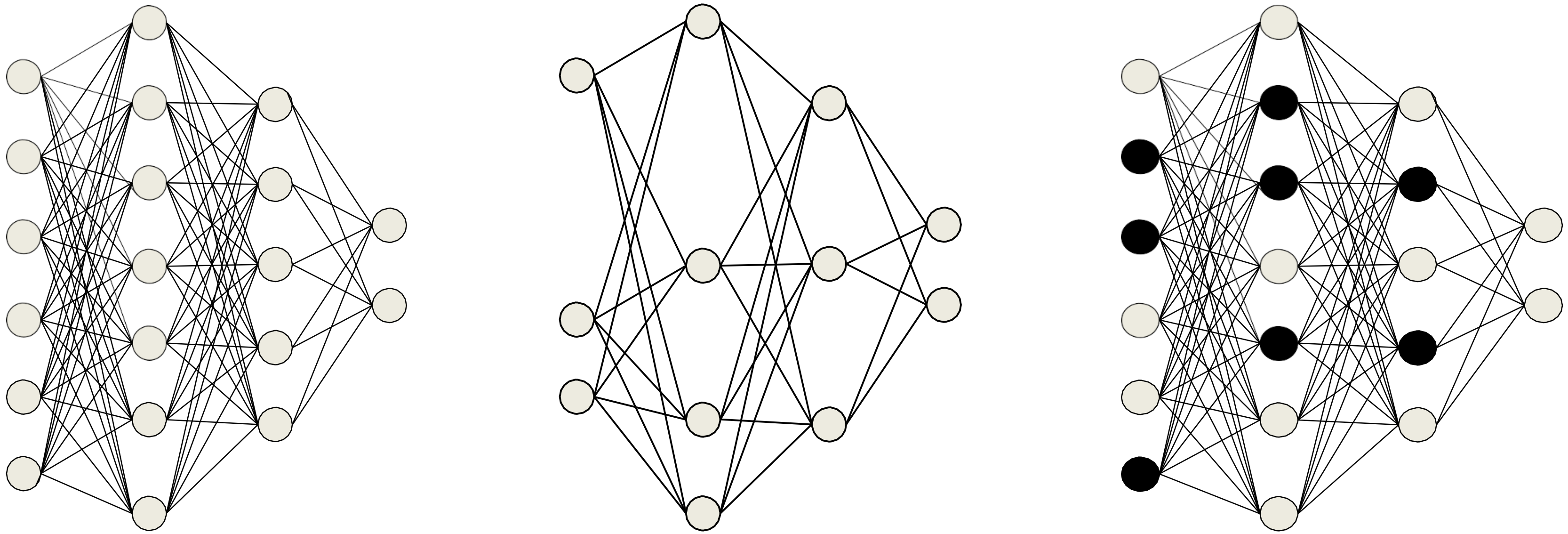


During training

- At each iteration, in each layer, “knock out” each neuron with probability 1-α

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.

Dropout



During training

- At each iteration, in each layer, “knock out” each neuron with probability 1-α
- In practice, we do not drop connections but set inputs/outputs to zero

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.

Dropout in Forward Pass

Without dropout:

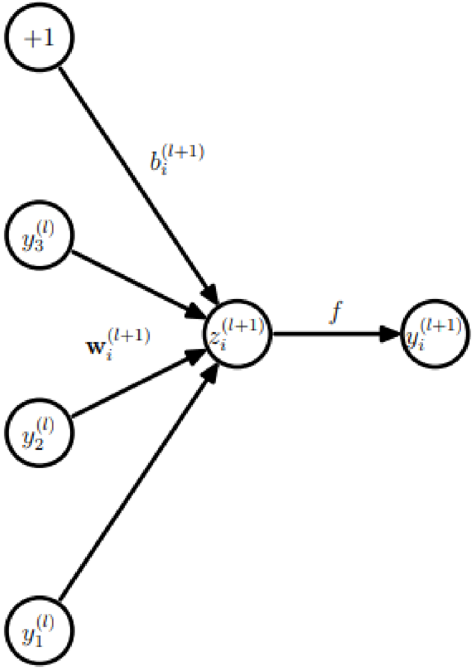
$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)},$$
$$y_i^{(l+1)} = f(z_i^{(l+1)}),$$

With dropout:

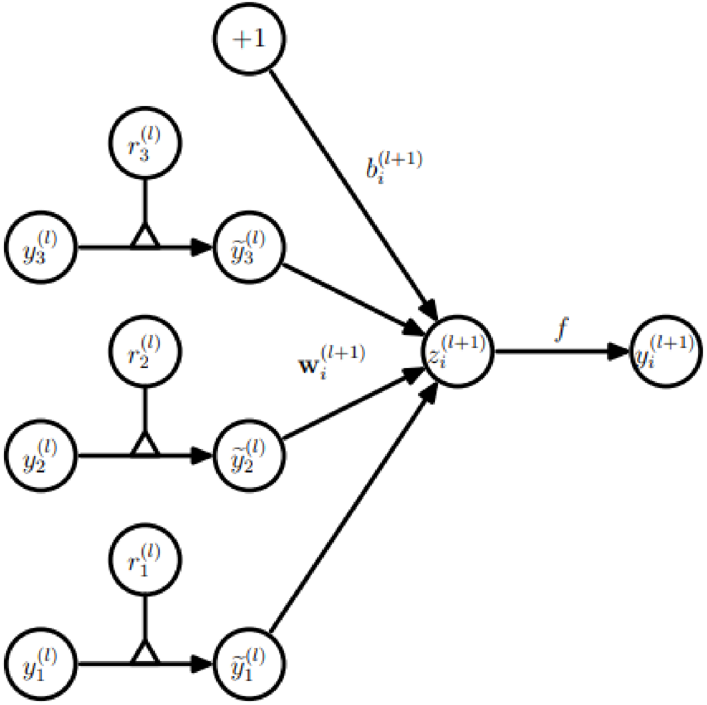
$$r_j^{(l)} \sim \text{Bernoulli}(p),$$
$$\widetilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{y}^{(l)},$$
$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)},$$
$$y_i^{(l+1)} = f(z_i^{(l+1)}).$$

1. For every node j and layer l, determine Bernoulli number {0,1}
2. Drop outputs
3. ???
4. Profit.

Dropout in Forward Pass

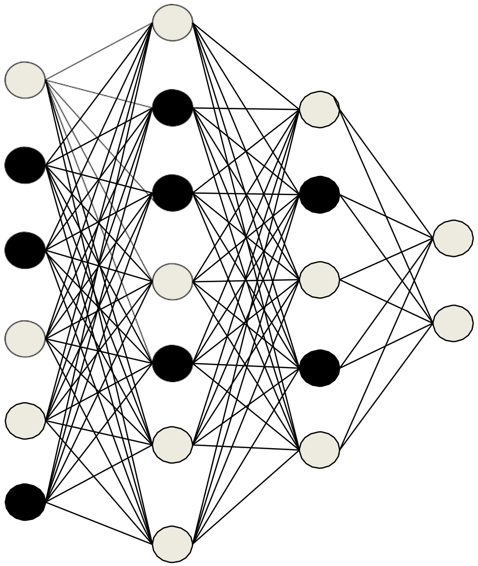


(a) Standard network



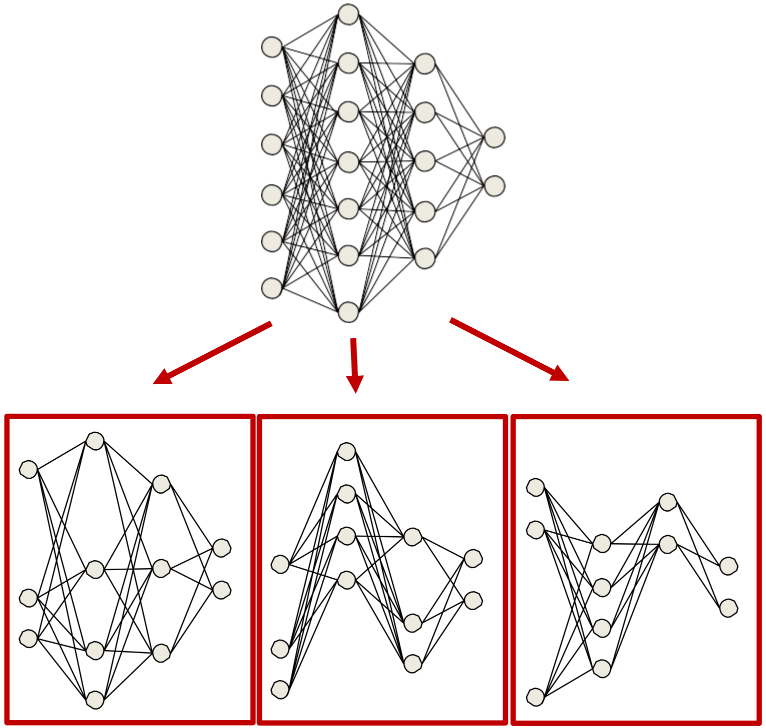
(b) Dropout network

Dropout in Backward Pass



- Backpropagation as usual, but
  Set updates to zero for dropped out weights
- Tricks of the trade still work

Dropout at Inference Time

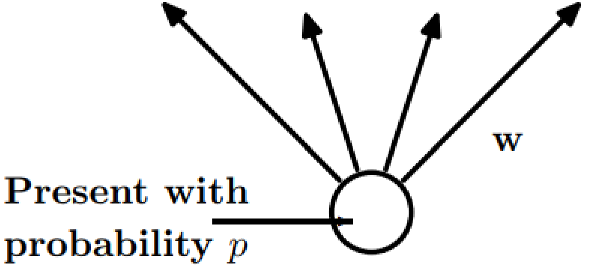


A slightly different view onto dropout

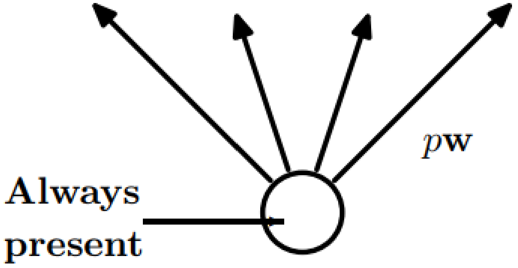
- 2^N sub-networks for N-neuron network
- Dropout samples over these sub-networks

→ Learns a network that averages over all possible networks

Dropout at Inference Time



(a) At training time



(b) At test time

During training

- Fewer activations present

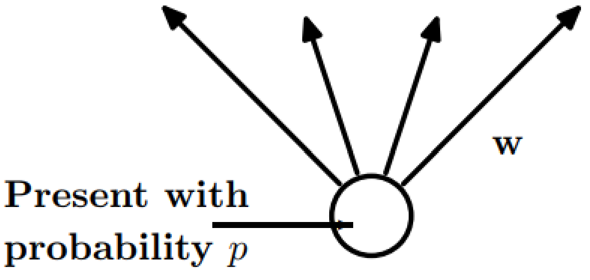
→ Overall activation smaller

During testing

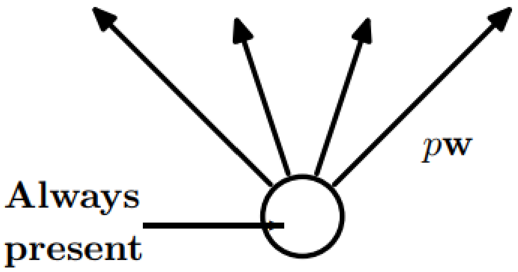
- All activations present

→ Weights or activations scaled by *p*

Dropout at Inference Time



(a) At training time



(b) At test time

During training

- Fewer activations present

→ Overall activation smaller

During testing

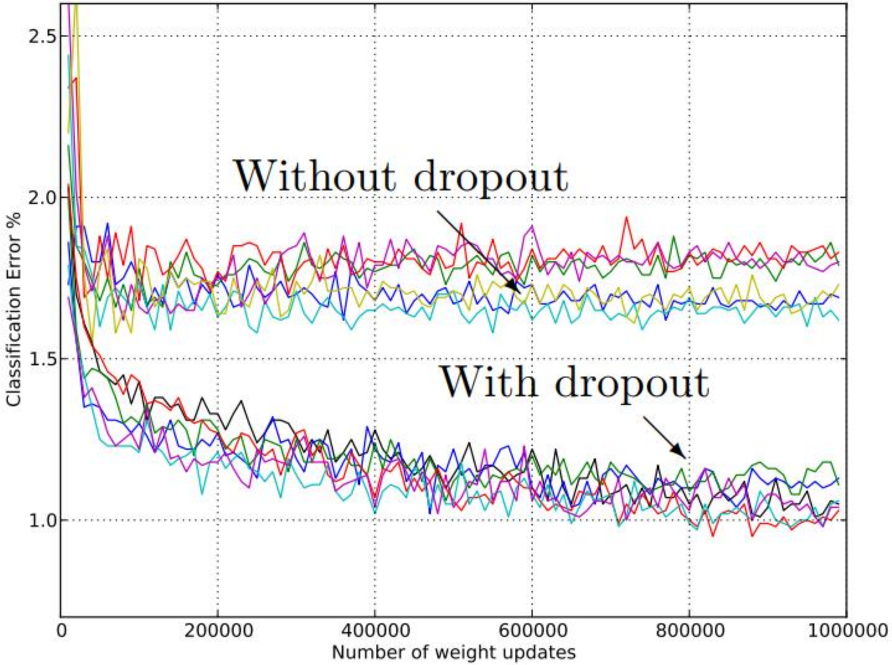
- All activations present

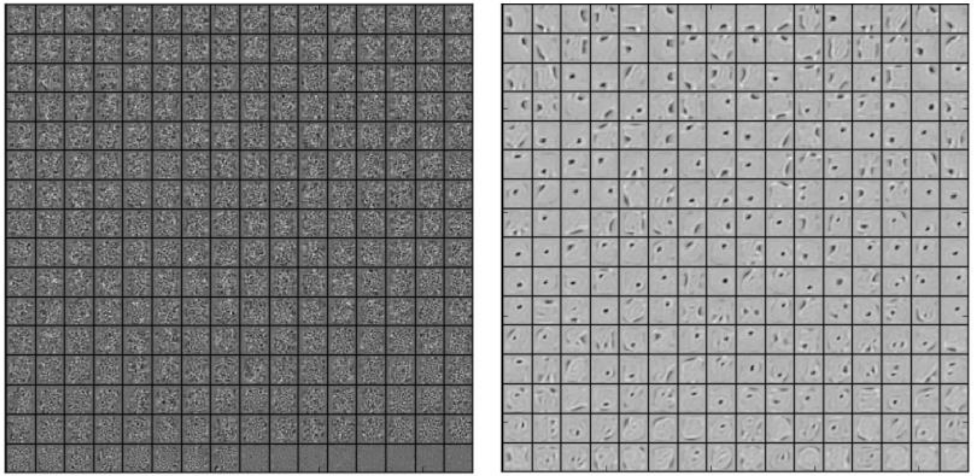
→ Weights or activations scaled by *p*

Some research on test time dropout.

Q: Why would you want to do this?

Dropout: Typical Values and Results





(a) Without dropout

(b) Dropout with $p = 0.5$.

This experiment considers an autoencoder.
We will see this behavior again later.

Typical values

- Input unit dropout: 0.2
- Hidden unit dropout: 0.5

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(1), 1929-1958.

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

EN.601.482/682 Deep Learning

Training Part II

Update Rules, Data Augmentation, Transfer Learning

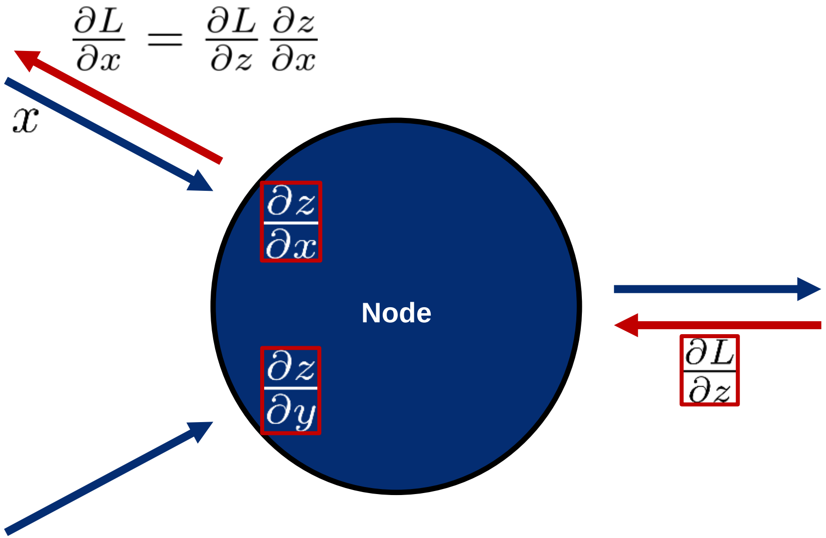
Mathias Unberath, PhD
Assistant Professor
Dept of Computer Science
Johns Hopkins University

Reminder

ConvNets

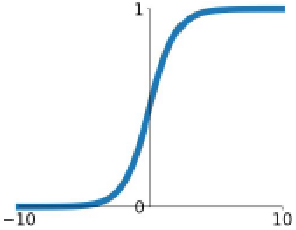
- One-time setup
  - Architecture (Lecture 12)
  - Activation functions (sigmoid, ReLU, …)
  - Regularization (batch norm, dropout)

- Training
  - Data collection: Preprocessing, Augmentation
  - Training via SGD (update rules)

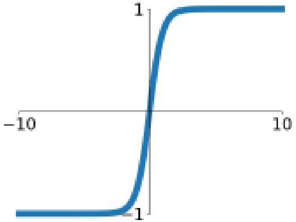


Reminder

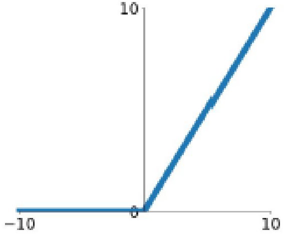
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



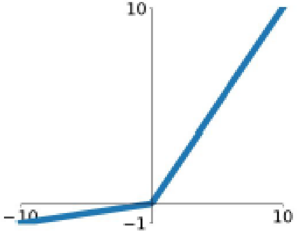
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

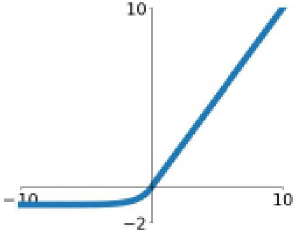


Leaky ReLU
$\max(0.1x, x)$



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

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



Reminder

Activation-related problems to keep track of

- Vanishing gradients for saturated neurons
  → Dying ReLU problem
- Linear vs. non-linear regime
- Output-range: Zero-centered?
  If not, ineffective gradient updates
- Parameters?
  Can be as easy as PReLU or as complex as Maxout

Reminder

Initialization-related problems to keep track of

- Never initialize with a constant
  → Symmetry must be broken for training to succeed

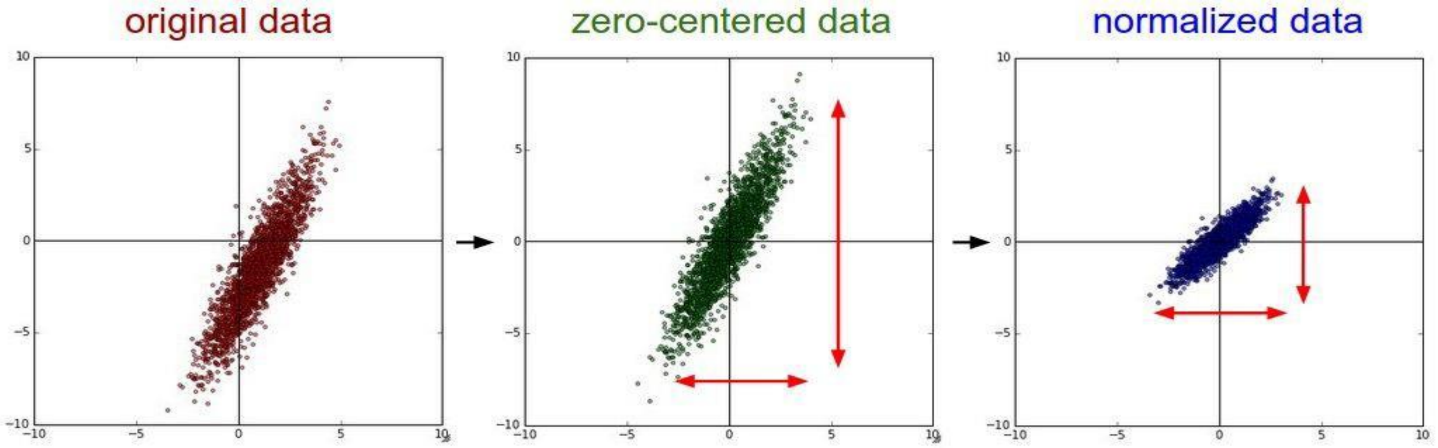
- Xavier and He initialization: Important in the success of DL

- If you are using **ReLU** as recommended: **He initialization**

Reminder

Preprocessing

- Zero-centered data for more effective gradient updates!
- Normalization not always necessary
- Consider dynamic range

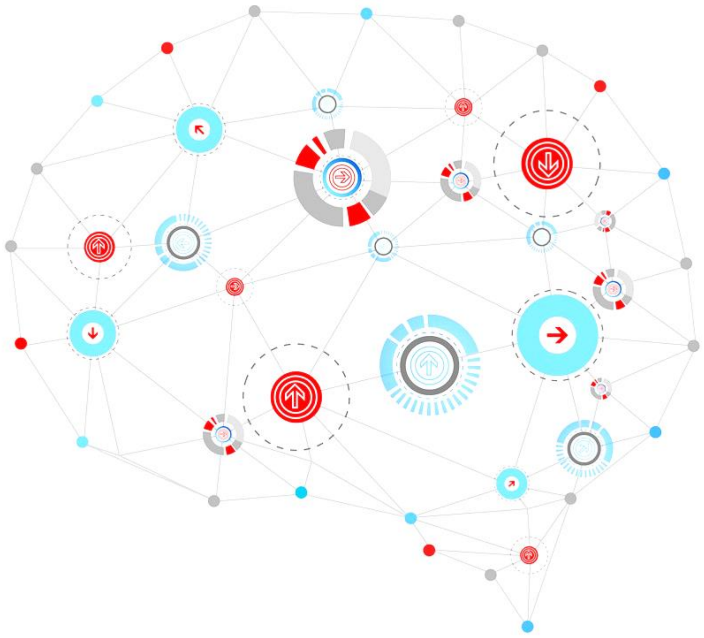


Today's Lecture

Update Rules

Data Augmentation

Transfer Learning

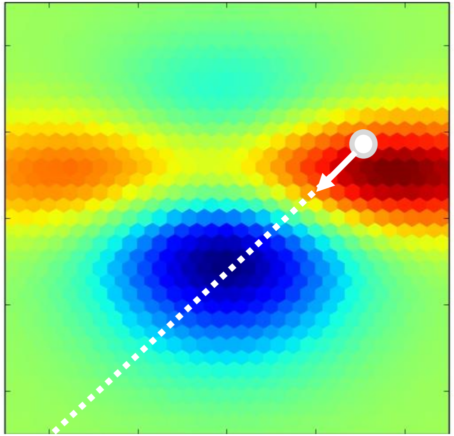


Update Rules

Optimization

Reminder: **Standard gradient descent**

Finding the lowest point: $W' = \arg\min_W L(W)$

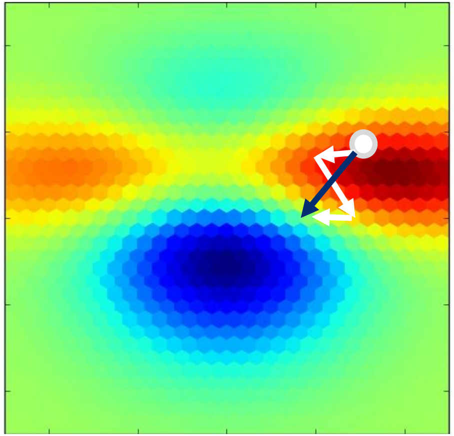


```
while not_converged:
    gradient = eval_gradient(loss, data, weights)
    weights += - step_size * gradient
```

Optimization

Reminder: **Stochastic gradient descent**

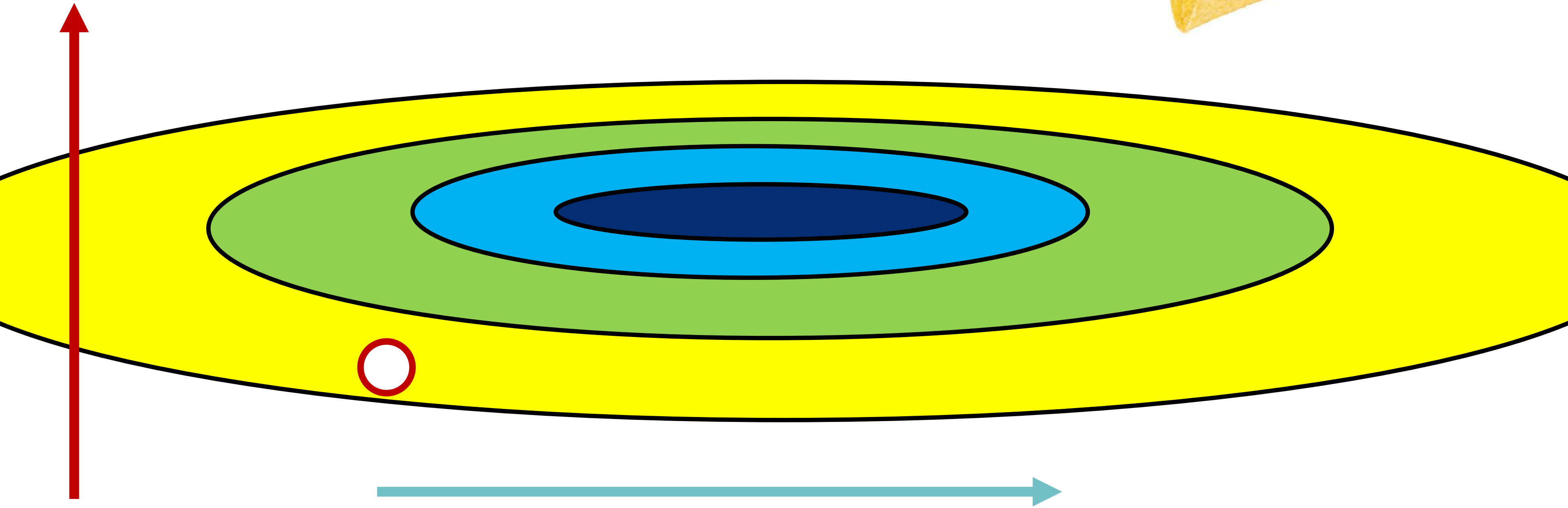
Finding the lowest point: $W' = \arg\min_W L(W)$



```
while not_converged:
    data_batch = sample_training_data(data, batch_size)
    gradient = eval_gradient(loss, data_batch, weights)
    weights += - step_size * gradient
```

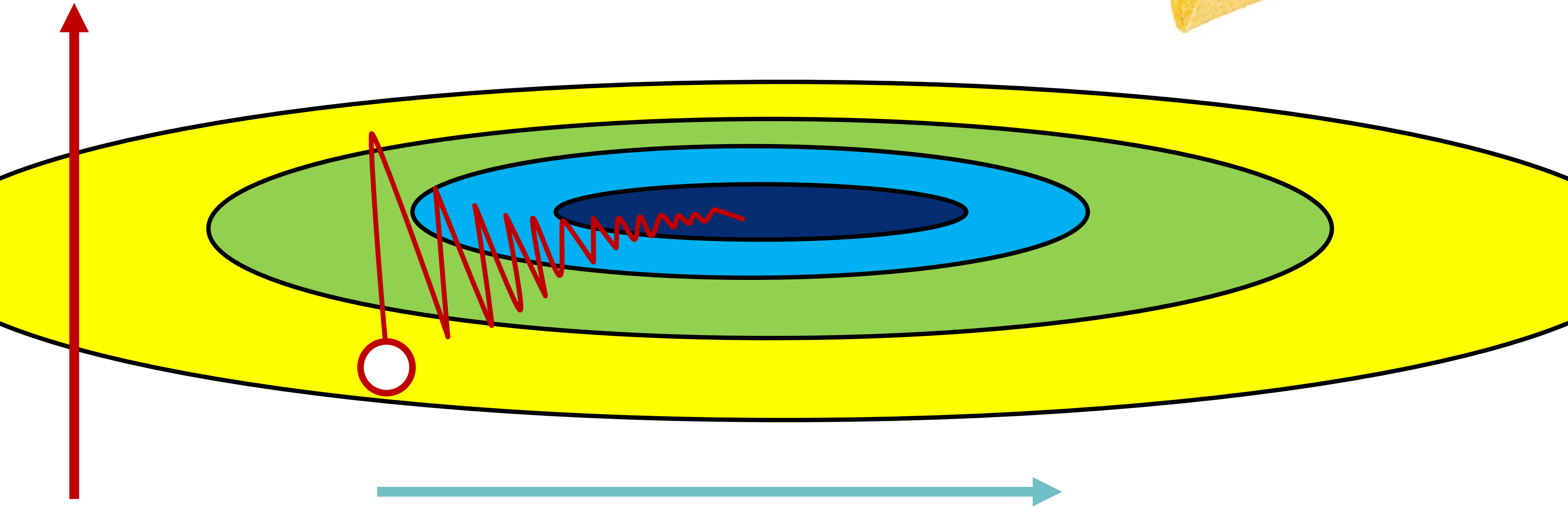
Problems with SGD Optimization

Loss changes very quickly along one direction
and very slowly along the other



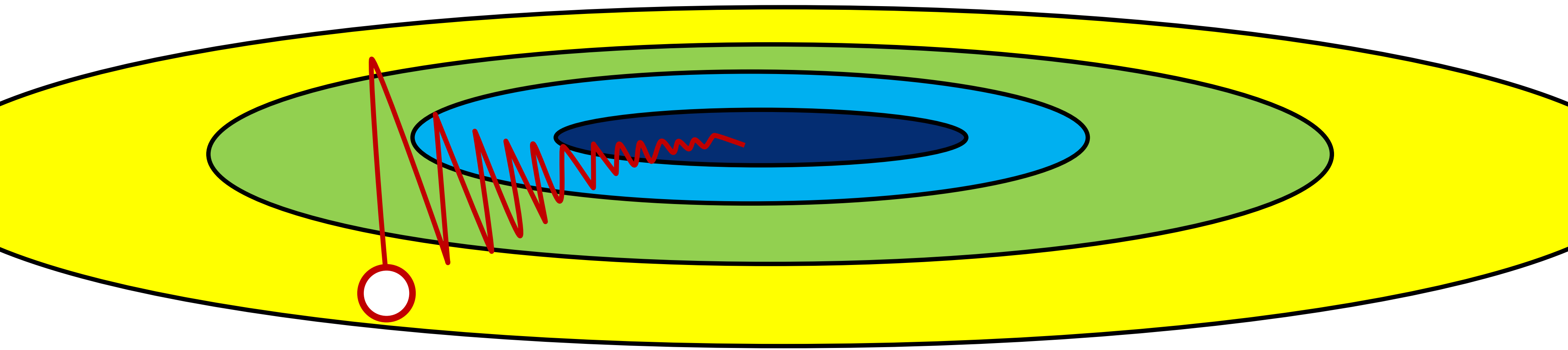
Problems with SGD Optimization

Loss changes very quickly along one direction
and very slowly along the other



Problems with SGD Optimization

→ Slow progress along shallow dimension, jitter along the other



Why is this?

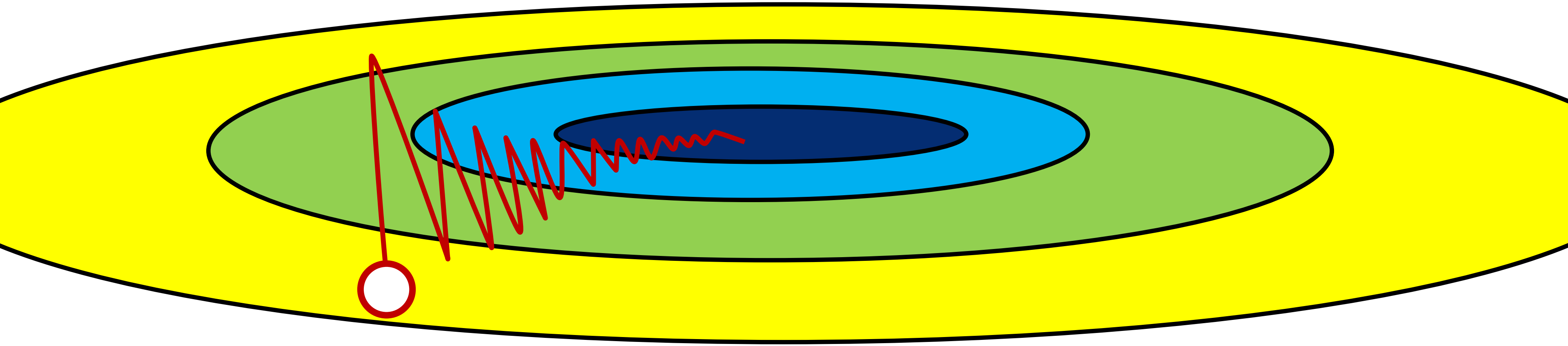
Condition number: Ratio of largest to smallest singular value of the Hessian

→ If large, then loss function at this point badly conditioned

Problems with SGD Optimization

→ Slow progress along shallow dimension, jitter along the other

→ Problematic: Neural networks have millions of parameters!



Why is this?

Condition number: Ratio of largest to smallest singular value of the Hessian

→ If large, then loss function at this point badly conditioned

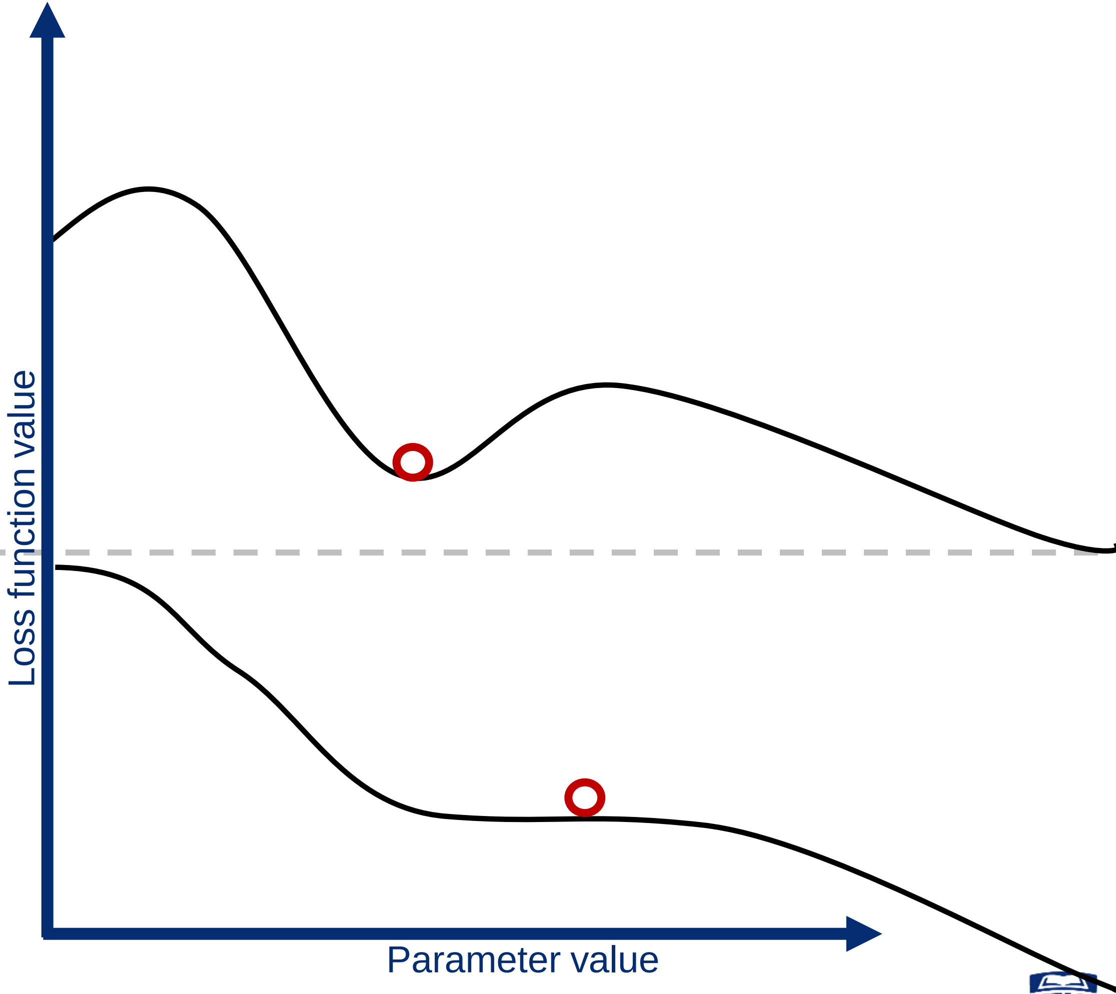
Another Problem

Local minimum

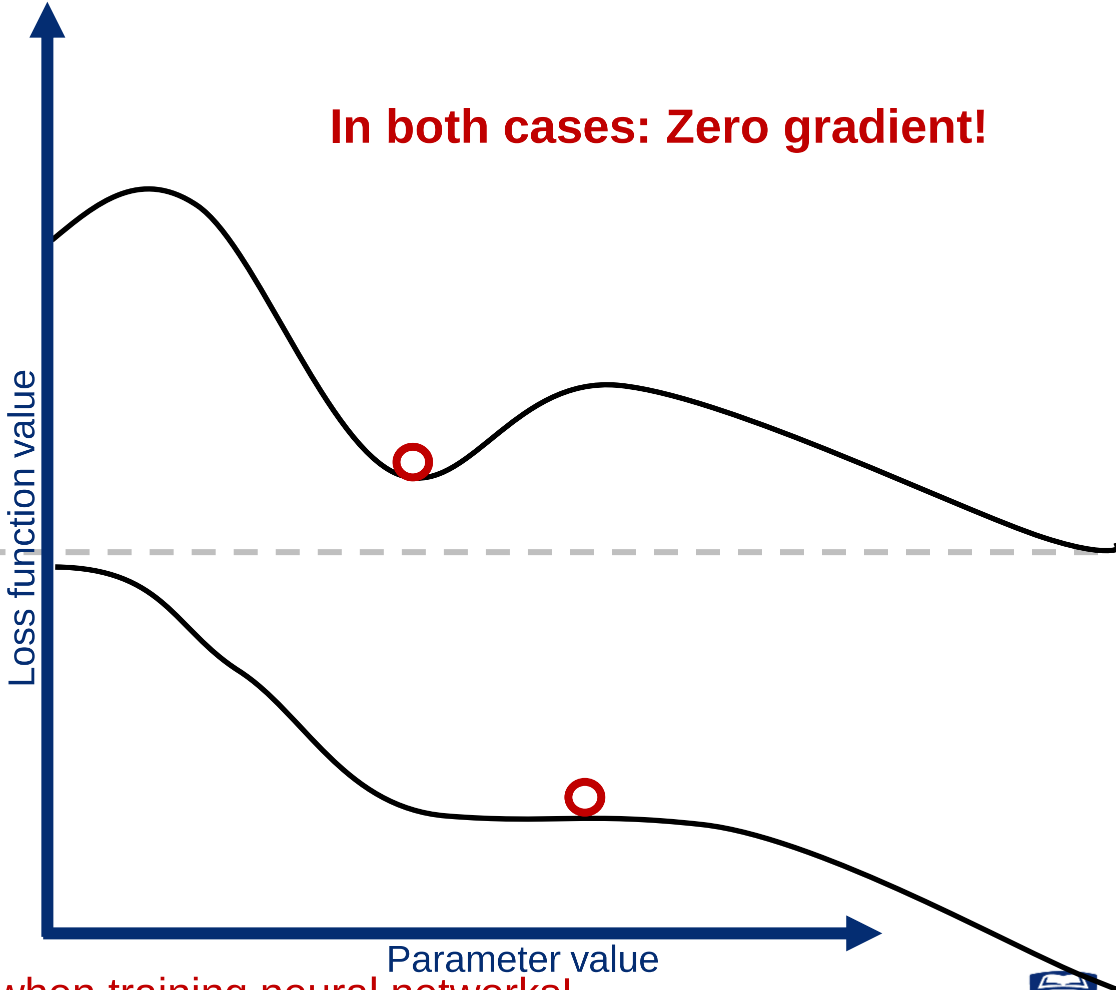
In every direction, loss will go up.

Saddle point

In some direction loss will go up,
in other direction loss will go down.



Another Problem



Local minimum

In every direction, loss will go up.

Saddle point

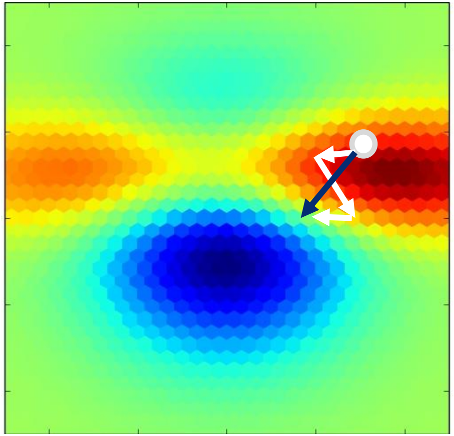
In some direction loss will go up,
in other direction loss will go down.

In high dimensional space, this scenario is much more common.

→ Saddle points are the big problem when training neural networks!

And Another Problem

```
while not_converged:
    data_batch = sample_training_data(data, batch_size)
    gradient = eval_gradient(loss, data_batch, weights)
    weights += - step_size * gradient
```

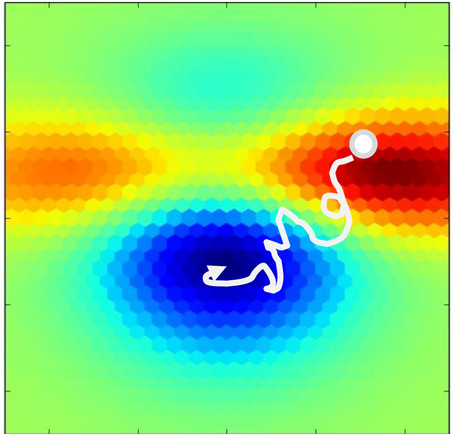


Gradient is computed over mini-batches

- Mini-batches do not necessarily represent the full dataset

And Another Problem

```
while not_converged:
    data_batch = sample_training_data(data, batch_size)
    gradient = eval_gradient(loss, data_batch, weights)
    weights += - step_size * gradient
```



Gradient is computer over mini-batches

- Mini-batches do not necessarily represent the full dataset
- **Gradients can be noisy!**

Adding Momentum $W' = \arg\min_W L(W)$

SGD

$$W_{t+1} = W_t - \alpha \nabla_W L(W_t)$$

- Update in negative gradient direction

Adding Momentum $W' = \arg\min_W L(W)$

SGD

$$W_{t+1} = W_t - \alpha \nabla_W L(W_t)$$

- Update in negative gradient direction

SGD + Momentum

$$v_{t+1} = \rho v_t + \alpha \nabla_W L(W_t)$$
$$W_{t+1} = W_t - v_{t+1}$$

- Replace gradient with *velocity*
- Velocity: Running mean of gradients
- ρ determines friction (ρ > 0.9)
- Update in negative velocity direction

Adding Momentum $W' = \arg\min_W L(W)$

SGD

$$W_{t+1} = W_t - \alpha \nabla_W L(W_t)$$

- Update in negative gradient direction

SGD + Momentum

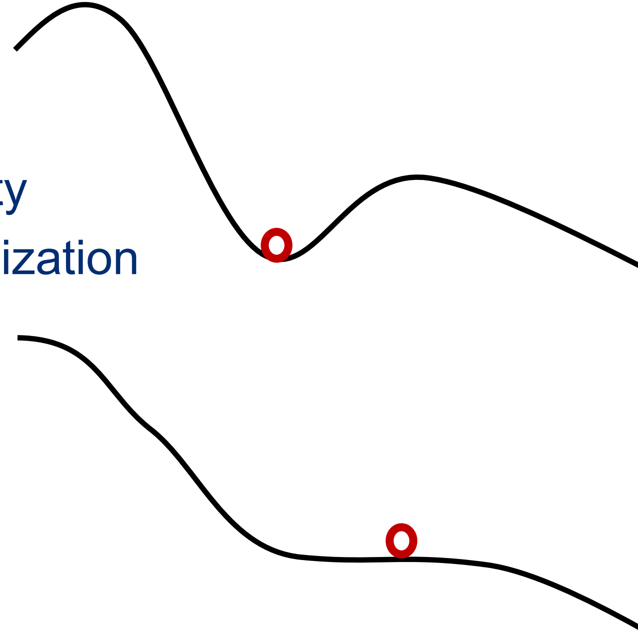
$$v_{t+1} = \rho v_t + \alpha \nabla_W L(W_t)$$
$$W_{t+1} = W_t - v_{t+1}$$

- Replace gradient with *velocity*
- Velocity: Running mean of gradients
- ρ determines friction (ρ > 0.9)
- Update in negative velocity direction

This simple strategy helps in all previous problems!

SGD + Momentum

$$v_{t+1} = \rho v_t + \alpha \nabla_W L(W_t)$$
$$W_{t+1} = W_t - \alpha v_{t+1}$$

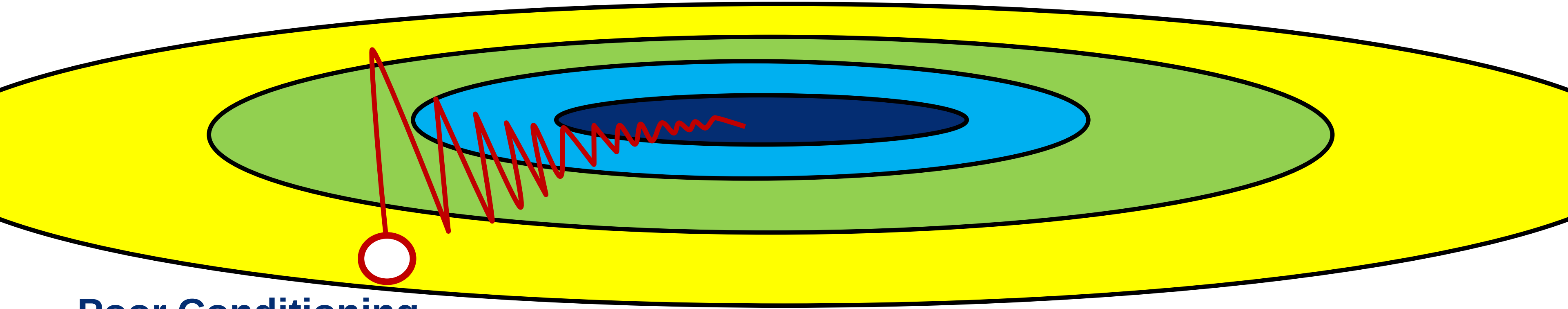


Saddle points / Local minima

- "Ball rolling down the hill" has momentum and velocity
- Even if there is zero gradient, velocity "carries" optimization

SGD + Momentum

$$v_{t+1} = \rho v_t + \alpha \nabla_W L(W_t)$$
$$W_{t+1} = W_t - \alpha v_{t+1}$$



Poor Conditioning

- Zig-zagging: Gradient contributions will cancel out
- Gradient along shallow dimension will accumulate, accelerating descent

Adding Momentum

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate O (1/k^ 2). In *Dokl. Akad. Nauk SSSR* (Vol. 269, pp. 543-547).
Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147).

SGD + Momentum

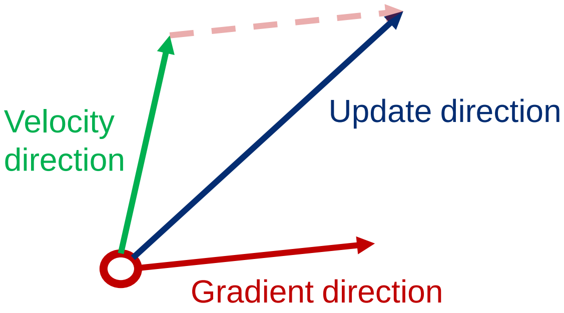
$$v_{t+1} = \rho v_t - \alpha \nabla_W L(W_t)$$
$$W_{t+1} = W_t + v_{t+1}$$

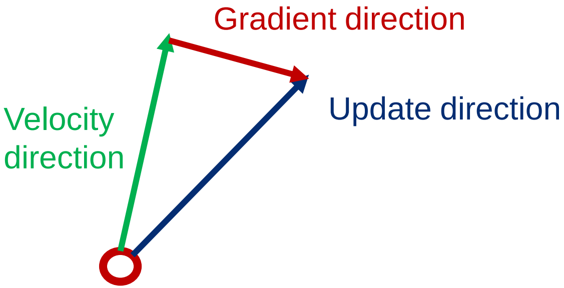
Combine gradient at current point with velocity to get update

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla_W L(W_t + \rho v_t)$$
$$W_{t+1} = W_t + v_{t+1}$$

Evaluate gradient at where velocity would take us, then mix with velocity





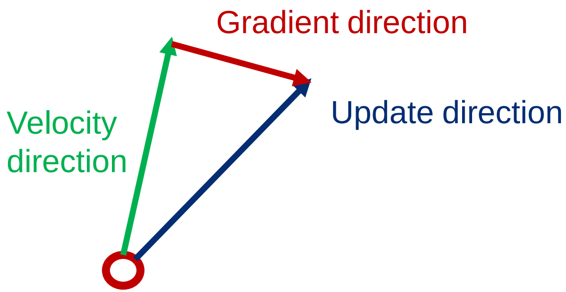
Nesterov Momentum

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla_W L(\boxed{W_t + \rho v_t})$$
$$W_{t+1} = W_t + v_{t+1}$$

This is a little unpleasant: Gradient is not computed where we want to update

Evaluate gradient at where velocity would take us, then mix with velocity



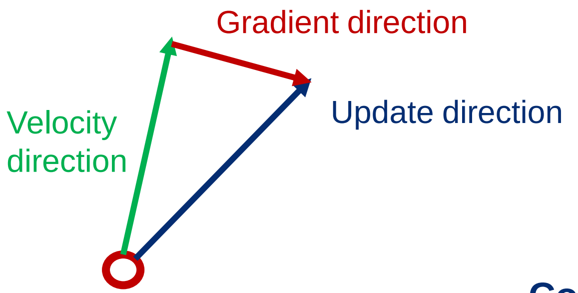
Nesterov Momentum

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla_W L\boxed{(W_t + \rho v_t)}$$
$$W_{t+1} = W_t + v_{t+1}$$

This is a little unpleasant: Gradient is not computed where we want to update

Evaluate gradient at where velocity would take us, then mix with velocity



Change of variables: $\tilde{W}_t = W_t + \rho v_t$

$$v_{t+1} = \rho v_t + \alpha \nabla_W L(\tilde{W}_t)$$

$$\begin{aligned} \tilde{W}_{t+1} &= \tilde{W}_t - \rho v_t + (1+\rho) v_{t+1} \\ &= \tilde{W}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned}$$

Conceptually: Swap the order of gradient and momentum update.

AdaGrad

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2121-2159.

$$\boxed{g_t = \nabla_W L(W_t)}$$

$$S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

1. Compute gradient

AdaGrad

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2121-2159.

$$g_t = \nabla_W L(W_t)$$

$$\boxed{S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0}$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

1. Compute gradient
2. Compute and accumulate element-wise squared gradient

AdaGrad

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2121-2159.

$$g_t = \nabla_W L(W_t)$$

$$S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0$$

$$\boxed{(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i}$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

1. Compute gradient
2. Compute and accumulate element-wise squared gradient
3. Compute gradient update with **parameter-wise** learning rate

AdaGrad

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2121-2159.

$$g_t = \nabla_W L(W_t)$$

$$S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

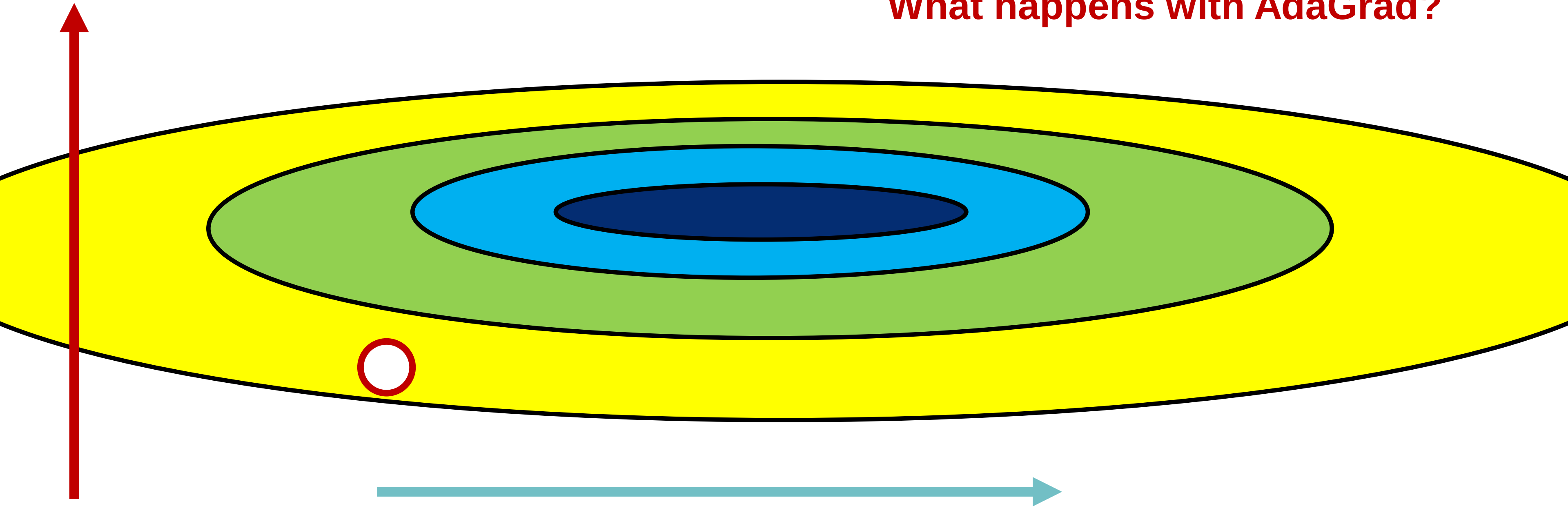
$$\boxed{W_{t+1} = W_t - \mathrm{d}W_t}$$

1. Compute gradient
2. Compute and accumulate element-wise squared gradient
3. Compute gradient update with **parameter-wise** learning rate
4. Apply gradient update

AdaGrad

Loss changes very quickly along one direction and very slowly along the other

Q: SGD will produce zig-zagging. What happens with AdaGrad?



AdaGrad

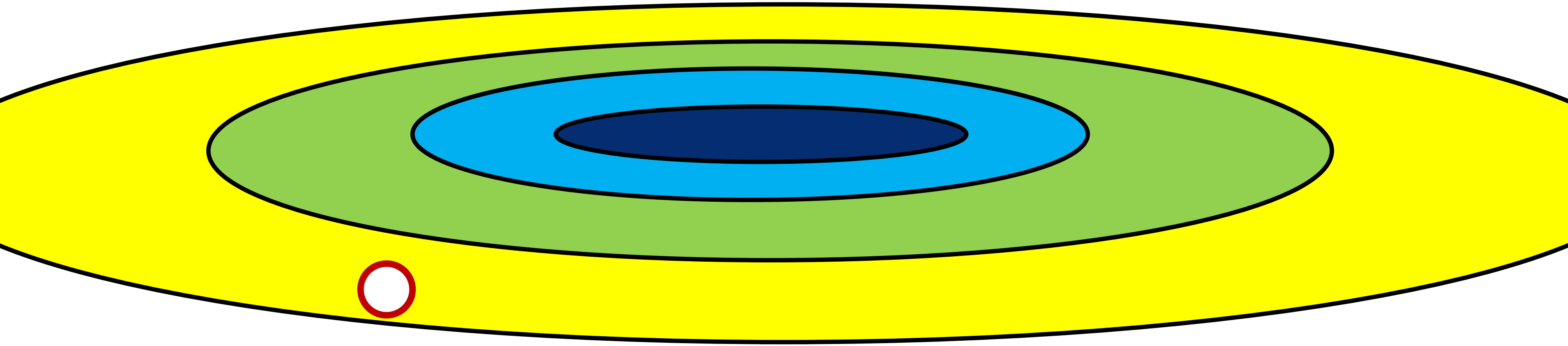
$$S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0$$

$$(\mathrm{d}W_t)_i = \boxed{\frac{\alpha}{\sqrt{S_i} + \epsilon}}(g_t)_i$$

Q: What happens with AdaGrad?

Learning rate in steep direction is damped strongly,

Learning rate in shallow direction is “accelerated”

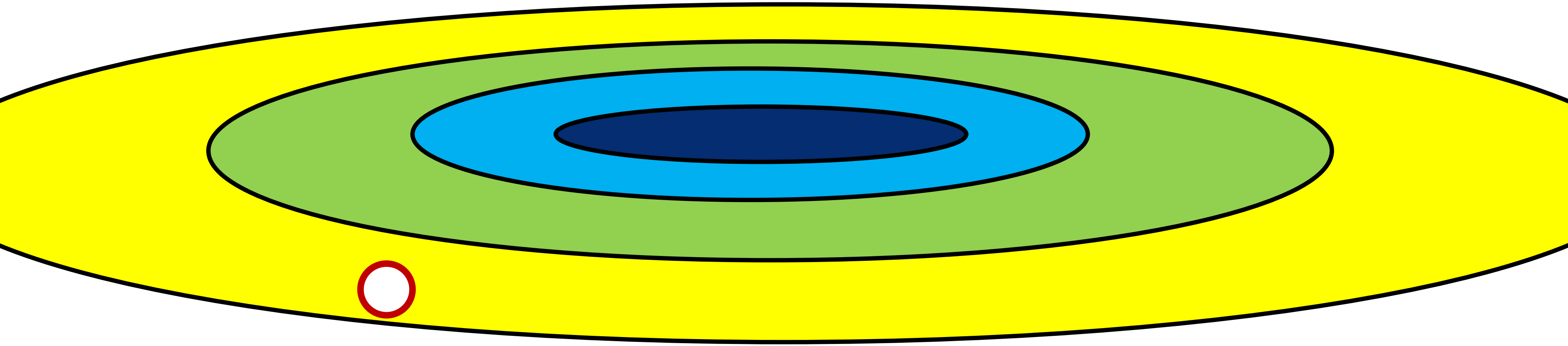


AdaGrad

$$\boxed{S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0}$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

Q: Problem?



AdaGrad

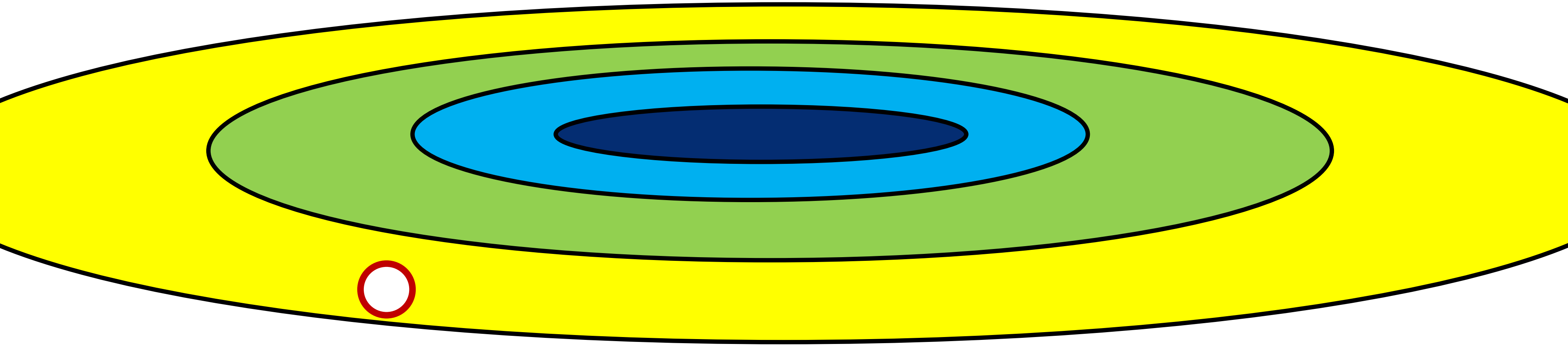
$$\boxed{S_i = S_i + (g_t)_i^2 \quad \text{with } S_i(t=0) = 0}$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

Q: Problem?

S_i term only ever increases, so the learning rate will decay to zero.

→ Slow convergence, or no convergence at all



Enter RMSProp

RMSProp is **unpublished** but has similarities to ADADELTA:
Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

$$\boxed{g_t = \nabla_W L(W_t)}$$

$$S_i = \rho \cdot S_i + (1-\rho)(g_t)_i^2 \quad \text{with } S_i(t=0) = 0$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

1. Compute gradient

Enter RMSProp

RMSProp is **unpublished** but has similarities to ADADELTA:
Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

$$S_i = S_i + \sqrt{(g_t)_i^2}$$

$$g_t = \nabla_W L(W_t)$$

$$\boxed{S_i = \rho \cdot S_i + (1-\rho)(g_t)_i^2 \quad \text{with } S_i(t=0) = 0}$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

1. Compute gradient
2. Compute “’discounted” element-wise squared gradient

Enter RMSProp

RMSProp is **unpublished** but has similarities to ADADELTA:
Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

$$S_i = S_i + \sqrt{(g_t)_i^2}$$

$$g_t = \nabla_W L(W_t)$$

$$S_i = \rho \cdot S_i + (1-\rho)(g_t)_i^2 \quad \text{with } S_i(t=0)=0$$

$$\boxed{(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i}$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

1. Compute gradient
2. Compute “’discounted” element-wise squared gradient
3. Compute gradient update with **parameter-wise** learning rate

Enter RMSProp

RMSProp is **unpublished** but has similarities to ADADELTA:
Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

$$g_t = \nabla_W L(W_t)$$

$$S_i = S_i + \sqrt{(g_t)_i^2}$$

$$S_i = \rho \cdot S_i + (1 - \rho)(g_t)_i^2 \quad \text{with } S_i(t = 0) = 0$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i} + \epsilon}(g_t)_i$$

$$\boxed{W_{t+1} = W_t - \mathrm{d}W_t}$$

1. Compute gradient
2. Compute "'discounted" element-wise squared gradient
3. Compute gradient update with **parameter-wise** learning rate
4. Apply gradient update

Recap and Take Away (if nothing else)

- Vanilla SGD has many problems
  - Noisy gradient updates
  - Zig-zagging in steep-and-shallow environments
  - Susceptible to local minima and saddle points

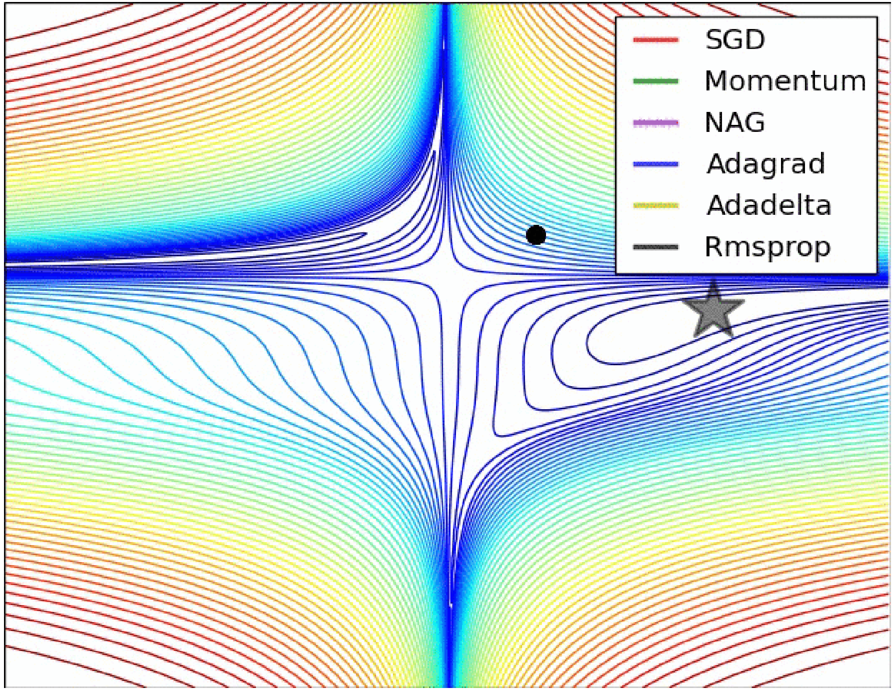
- Adding momentum (vanilla or Nesterov)
  - Stabilizes updates by mixing local gradients with “velocity”
  - Velocity: Non-zero updates even in domains with zero local gradient
  - Zig-zagging will cancel out

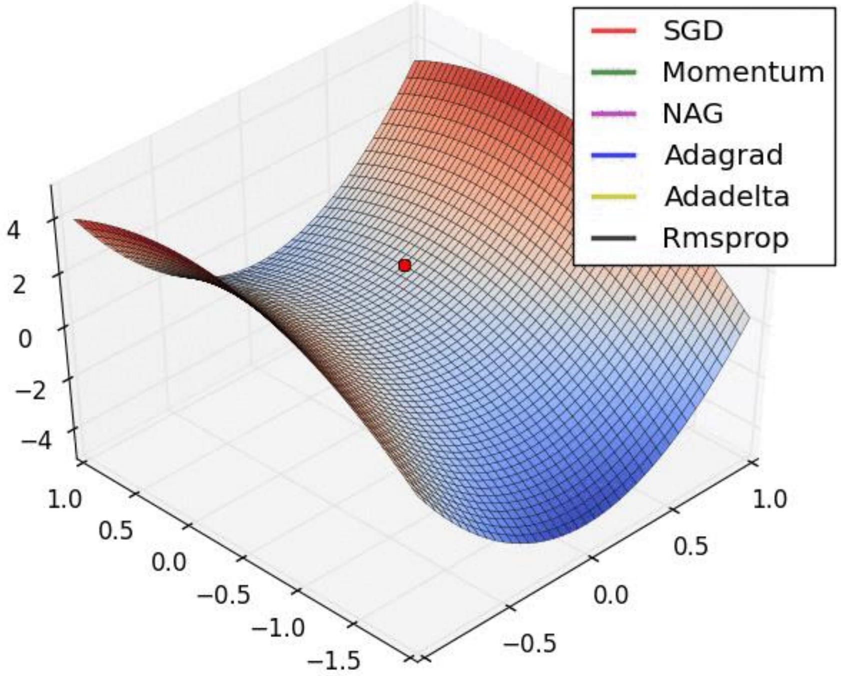
Tends to overshoot

- AdaGrad and RMSProp
  - Parameter-wise accumulation of gradient magnitude (RMSProp with discount rate)
  - Parameter-wise learning rate → “Equal steps” in every direction

Use RMSProp!
AdaGrad updates will vanish

Recap and Take Away (if nothing else)





Adam (almost)

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

$$\boxed{g_t = \nabla_W L(W_t)}$$

$$S_i^{(1)} = \rho_1 S_i^{(1)} + (1 - \rho_1)(g_t)_i$$

$$S_i^{(2)} = \rho_2 S_i^{(2)} + (1 - \rho_2)(g_t)_i^2$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i^{(2)}} + \epsilon} S_i^{(1)}$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

$$\boxed{\begin{aligned} S_i^{(1)}(t=0) &= 0 \\ S_i^{(2)}(t=0) &= 0 \end{aligned}}$$

1. Compute gradient

Adam (almost)

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

$$g_t = \nabla_W L(W_t)$$

Momentum

RMSProp

$$\boxed{\begin{aligned} S_i^{(1)} &= \rho_1 S_i^{(1)} + (1-\rho_1)(g_t)_i \\ S_i^{(2)} &= \rho_2 S_i^{(2)} + (1-\rho_2)(g_t)_i^2 \end{aligned}}$$

$$\boxed{\begin{aligned} S_i^{(1)}(t=0) = 0 \\ S_i^{(2)}(t=0) = 0 \end{aligned}}$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i^{(2)}} + \epsilon} S_i^{(1)}$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

1. Compute gradient
2. Compute first momentum (“velocity”)
3. Compute second momentum (parameter-wise normalization)

Adam (almost)

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

$$g_t = \nabla_W L(W_t)$$

$$S_i^{(1)} = \rho_1 S_i^{(1)} + (1-\rho_1)(g_t)_i$$

$$S_i^{(2)} = \rho_2 S_i^{(2)} + (1-\rho_2)(g_t)_i^2$$

$$\boxed{\begin{aligned}(\mathrm{d}W_t)_i &= \frac{\alpha}{\sqrt{S_i^{(2)}} + \epsilon} S_i^{(1)} \\ W_{t+1} &= W_t - \mathrm{d}W_t\end{aligned}}$$

$$\boxed{\begin{aligned}S_i^{(1)}(t=0) = 0 \\ S_i^{(2)}(t=0) = 0\end{aligned}}$$

Q: What happens at t=0?

1. Compute gradient
2. Compute first momentum (“velocity”)
3. Compute second momentum (parameter-wise normalization)
4. Compute update with **momentum** and **parameter-wise** learning rate
5. Apply update

Adam (almost)

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

$$g_t = \nabla_W L(W_t)$$

$$S_i^{(1)} = \rho_1 S_i^{(1)} + (1 - \rho_1)(g_t)_i$$

$$S_i^{(2)} = \rho_2 S_i^{(2)} + (1 - \rho_2)(g_t)_i^2$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i^{(2)}} + \epsilon} S_i^{(1)}$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

$$\boxed{\begin{aligned} S_i^{(1)}(t=0) &= 0 \\ S_i^{(2)}(t=0) &= 0 \end{aligned}}$$

Q: What happens at t=0?

Initialization 1st/2nd order momentum is zero; decay rates are very close to 1

2nd momentum very close to zero, step will be large!

→ Bias correction

Adam

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

$$g_t = \nabla_W L(W_t)$$

Bias correction

$$S_i^{(1)} = \left(\rho_1 S_i^{(1)} + (1-\rho_1)(g_t)_i\right)\boxed{(1-\rho_1^t)^{(-1)}}$$

$$S_i^{(2)} = \left(\rho_2 S_i^{(2)} + (1-\rho_2)(g_t)_i^2\right)\boxed{(1-\rho_2^t)^{(-1)}}$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i^{(2)}} + \epsilon} S_i^{(1)}$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

$$S_i^{(1)}(t=0) = 0$$
$$S_i^{(2)}(t=0) = 0$$

1. Compute gradient
2. Compute first momentum ("velocity")
3. Compute second momentum (parameter-wise normalization)
4. Compute update with **momentum** and **parameter-wise** learning rate
5. Apply update

Adam

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

$$g_t = \nabla_W L(W_t)$$

$$S_i^{(1)} = \left(\rho_1 S_i^{(1)} + (1-\rho_1)(g_t)_i\right)(1-\rho_1^t)^{(-1)}$$

$$S_i^{(2)} = \left(\rho_2 S_i^{(2)} + (1-\rho_2)(g_t)_i^2\right)(1-\rho_2^t)^{(-1)}$$

$$(\mathrm{d}W_t)_i = \frac{\alpha}{\sqrt{S_i^{(2)}} + \epsilon} S_i^{(1)}$$

$$W_{t+1} = W_t - \mathrm{d}W_t$$

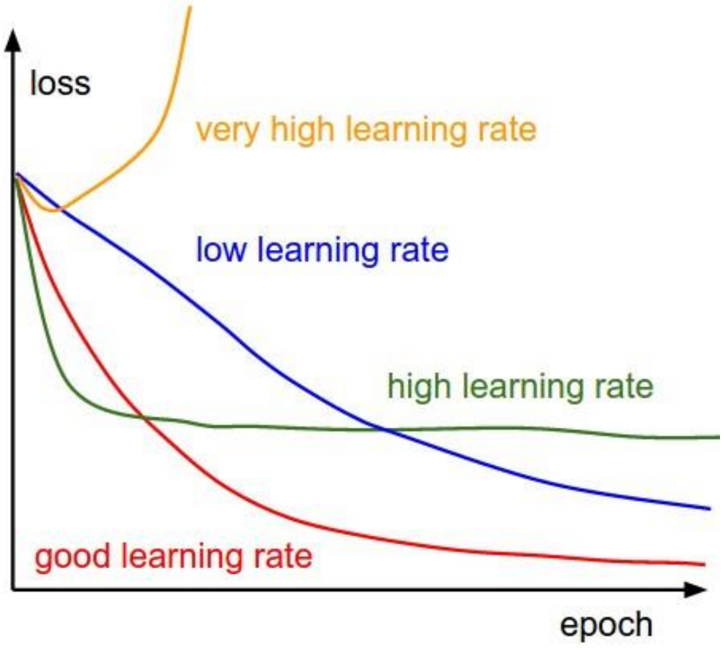
$$\boxed{\begin{array}{l} S_i^{(1)}(t=0) = 0 \\ S_i^{(2)}(t=0) = 0 \end{array}}$$

Bias correction: Compensate the fact that moments are close to zero at start.

Adam with $\rho_1 = 0.9,\ \rho_2 = 0.999,\ \alpha = 1e^{-3}$ is a good place to start!

A Note on Learning Rates

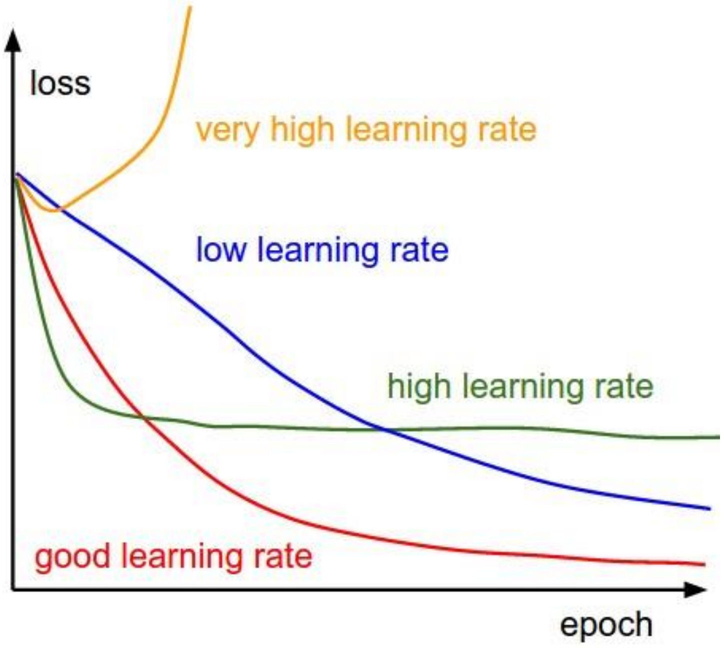
Nearly all optimization algorithms have learning rate
→ Typically, the most sensitive hyperparameter



Q: Which learning rate to chose?

A Note on Learning Rates

Nearly all optimization algorithms have learning rate
→ Typically, the most sensitive hyperparameter



→ Learning rate decay!

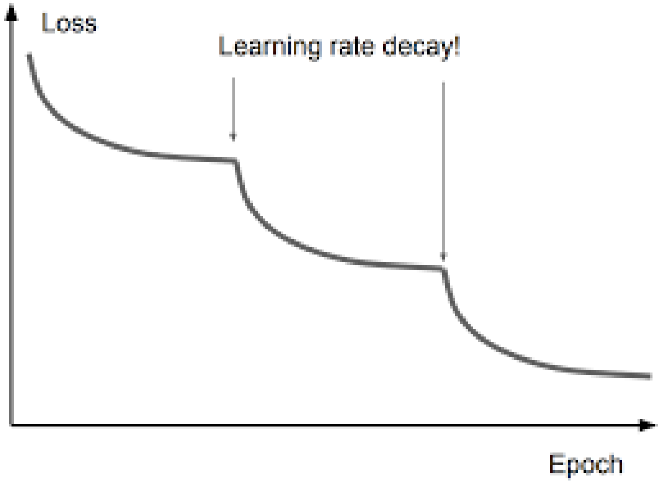
Step decay: E.g. decay by 0.1 every *X* epochs

Exponential decay: $\alpha = \alpha_0 e^{-kt}$

1/t decay: $\alpha = \frac{\alpha_0}{1+kt}$

A Note on Learning Rates

Nearly all optimization algorithms have learning rate
→ Typically, the most sensitive hyperparameter



→ Learning rate decay!

Step decay: E.g. decay by 0.1 every *X* epochs

Exponential decay: $\alpha = \alpha_0 e^{-kt}$

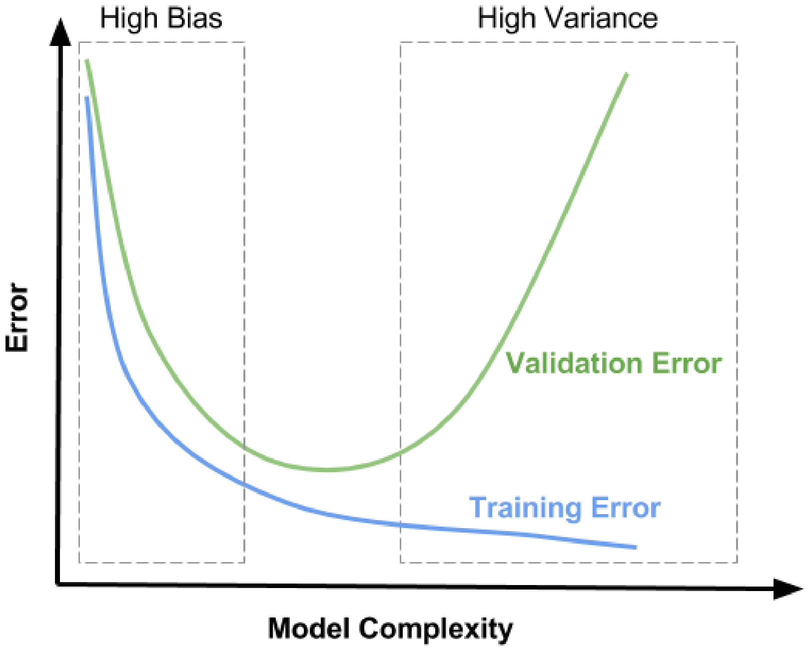
1/t decay: $\alpha = \frac{\alpha_0}{1+kt}$

Not common for Adam.

Reminder

Remember the **bias variance tradeoff** when optimizing for parameters!

→ Early stopping!



Update Rules, Data Augmentation, Transfer Learning

Data Augmentation

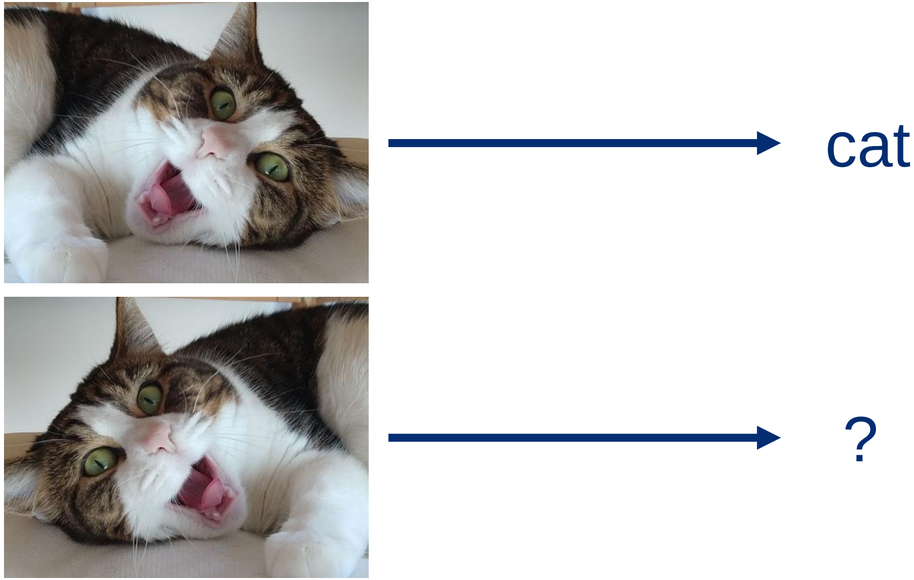
Data Augmentation

Two (primary) reasons:

1. Often, training data is very limited
2. Model should exhibit some invariance

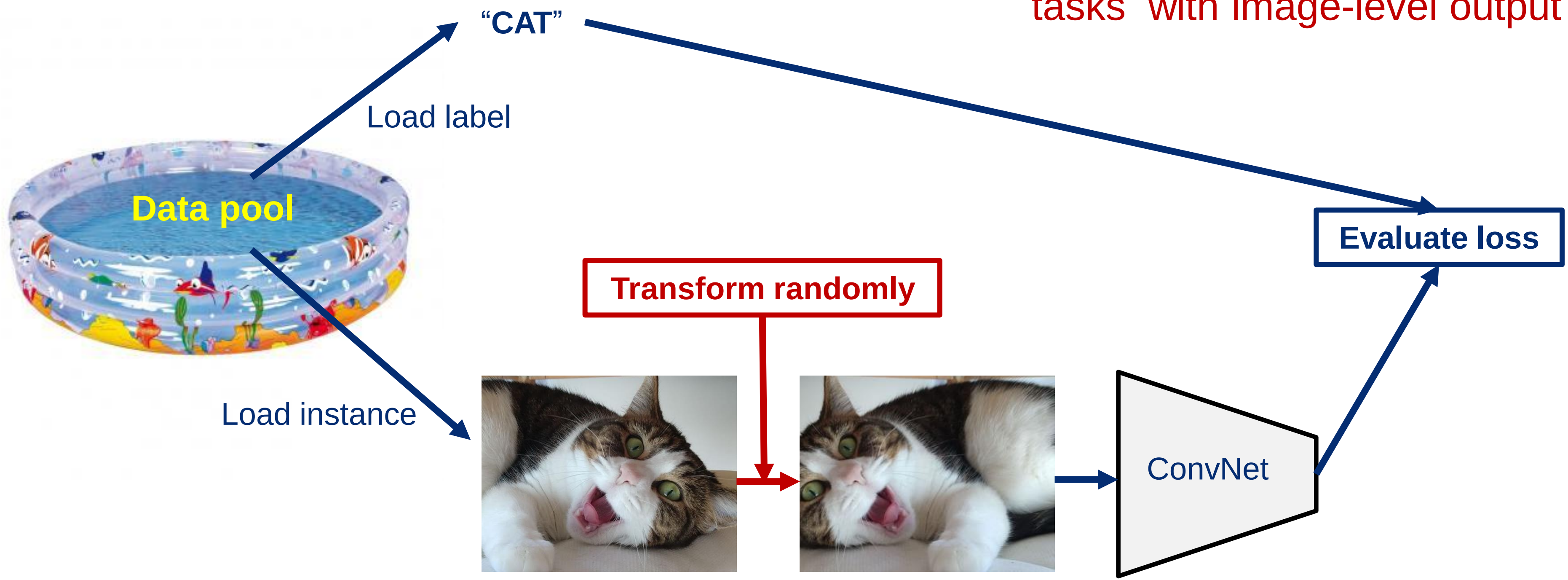
Frameworks are available:

- https://github.com/mdbloice/Augmentor
- https://github.com/aleju/imgaug



Data Augmentation: Classification

Q: How does this work for tasks with image-level output?



Data Augmentation: Segmentation …

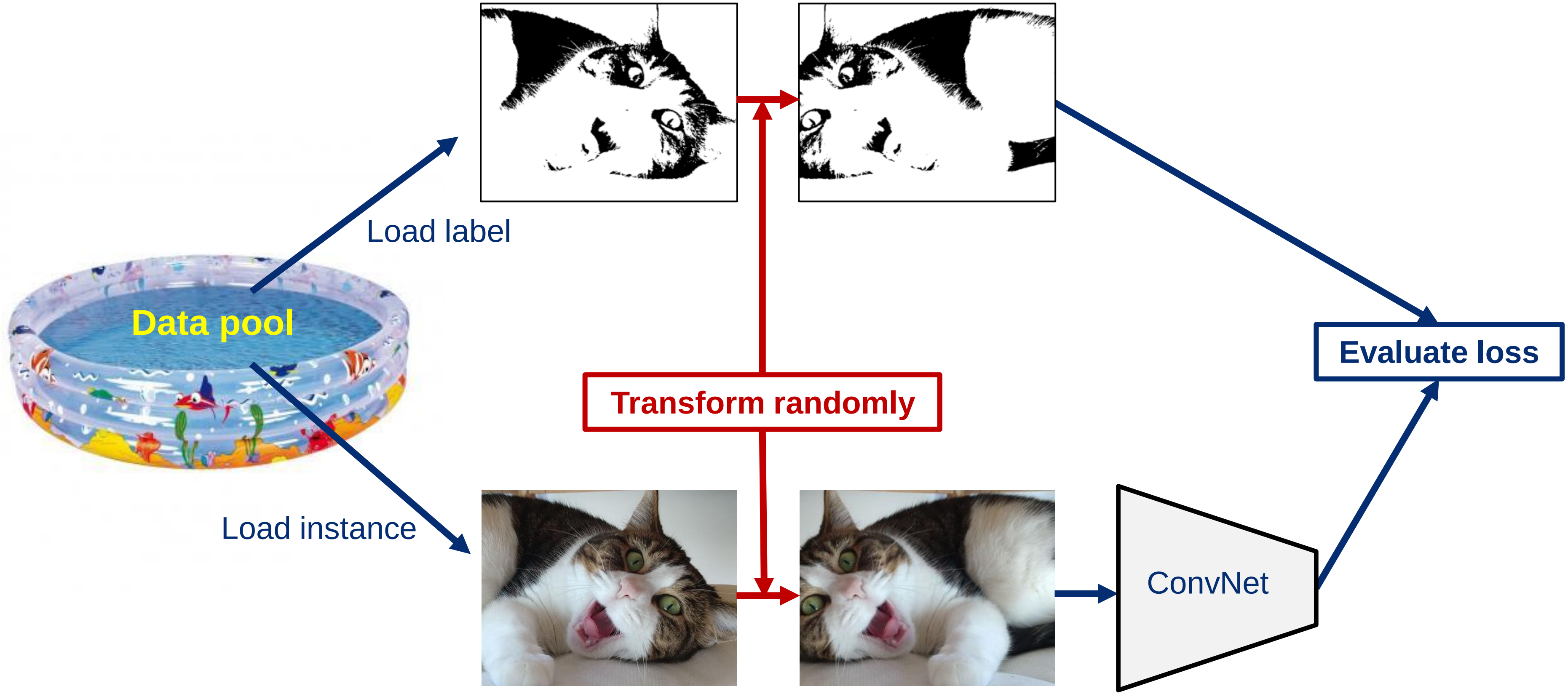
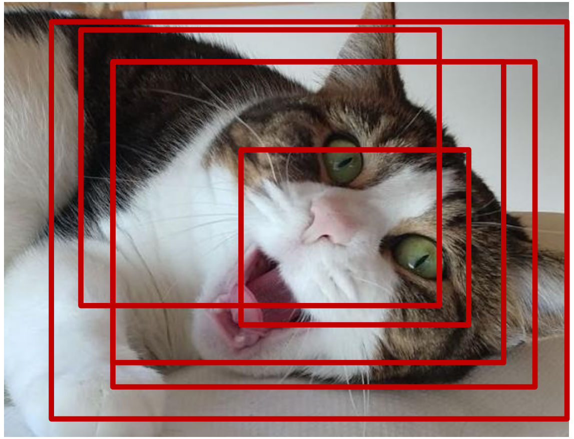


Image Transformations to Use for Augmentation

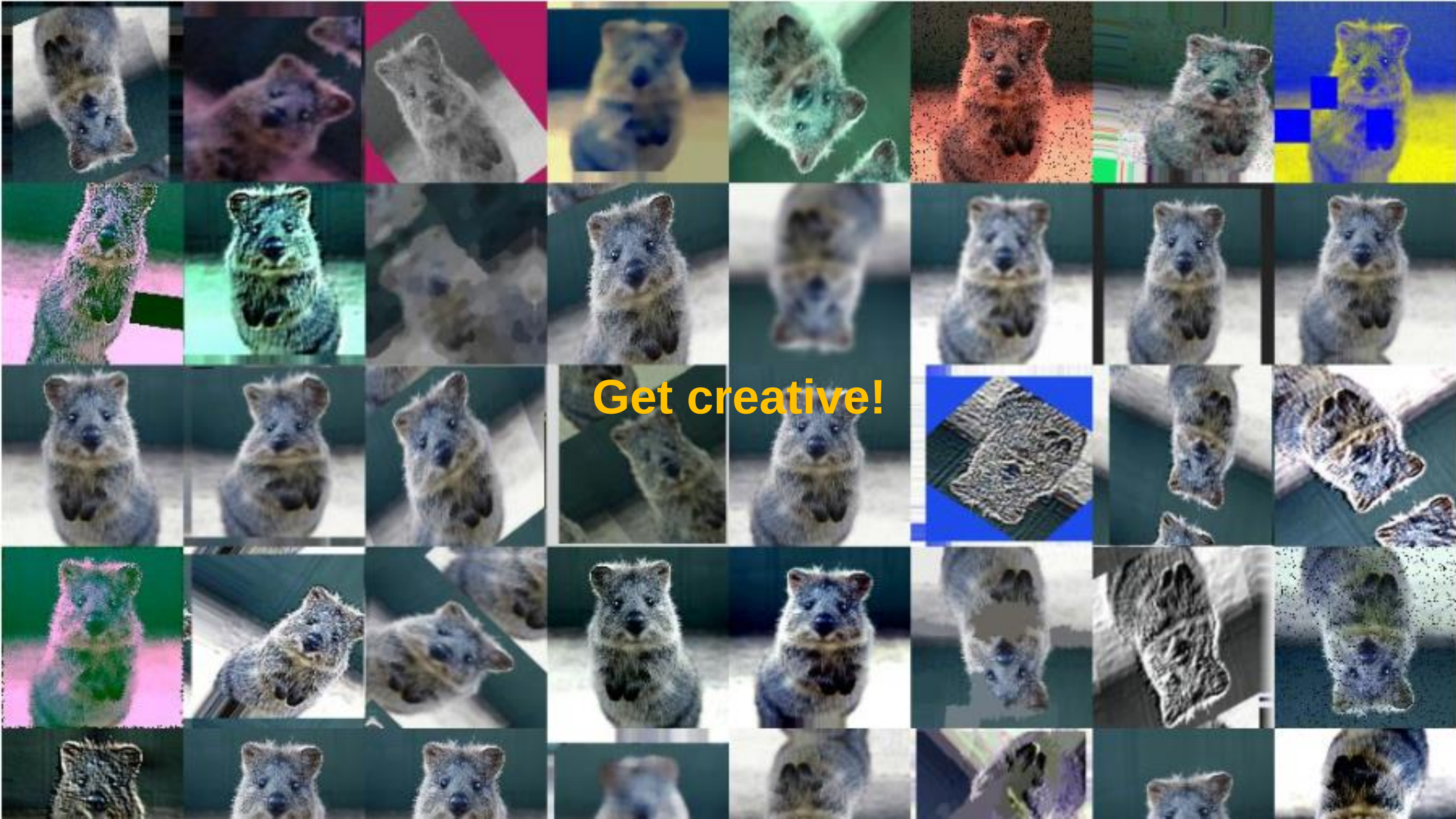
Rule of thumb
Every transformation that yields a **valid** image.

Examples: All these are random (within reasonable ranges)

- Horizontal / vertical flips
- Rotations and translations
- Noise (!)
- Scaling
- Cropping
- Color variations
- Distortions



→ We will see an interesting example of this soon!


Get creative!

A Small Aside

So far, we only discussed **training-time augmentation**

Goal: Make network invariant / robust to that particular variation in data

Remember **dropout**:

Goal: Make network invariant to feature co-adaptation

During **training**: Disturb input randomly

During **application**: Marginalize over randomness

Test time augmentation

→ Better statistics for predicted output

→ Some sense of “uncertainty”

Update Rules, Data Augmentation, Transfer Learning

Transfer Learning


Y'ALL GOT ANY MORE OF THEM
DATASETS?

Transfer Learning

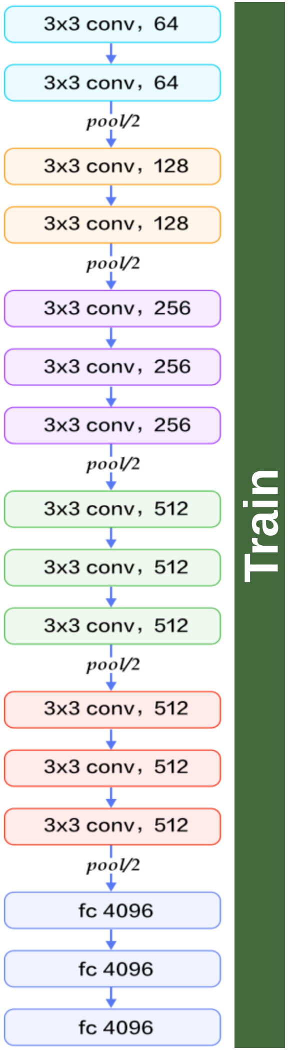
Training large models with limited data

Computer vision

- ImageNet: 1,2 Mio images
- MS-Celeb-1M: 10 Mio images

Medical imaging

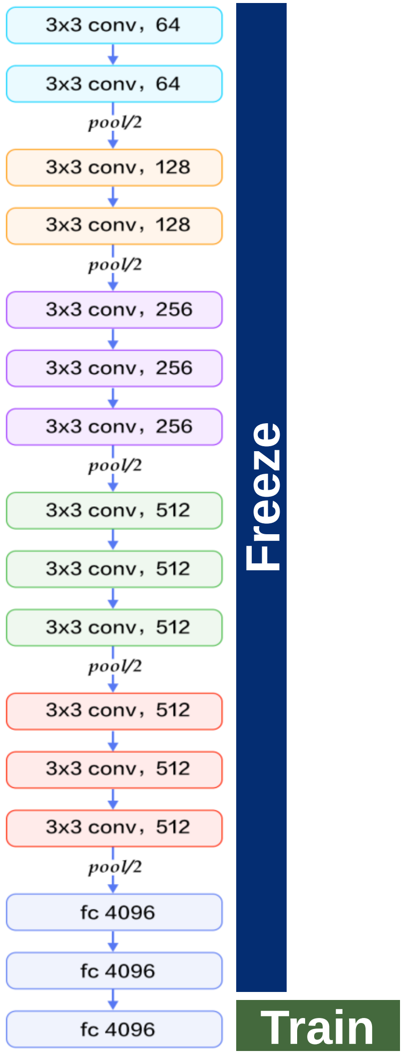
- CheXpert Chest X-ray: 224k images (14-class classification)
- Endoscopic artefact detection: ~2000 mixed resolution, multi-tissue, multi-modality, mixed population (7 class)



The regular approach to learning

A whole lot of data

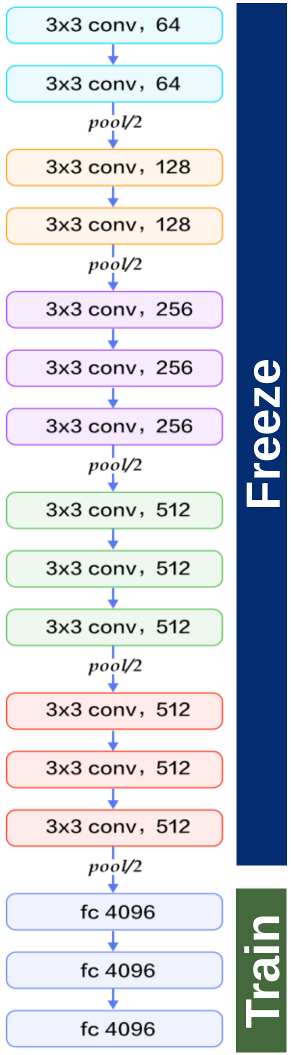
- Set-up network architecture
- Initialize randomly
- Train all parameters



Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. CVPR (pp. 806-813).

Transfer learning:
Very little data

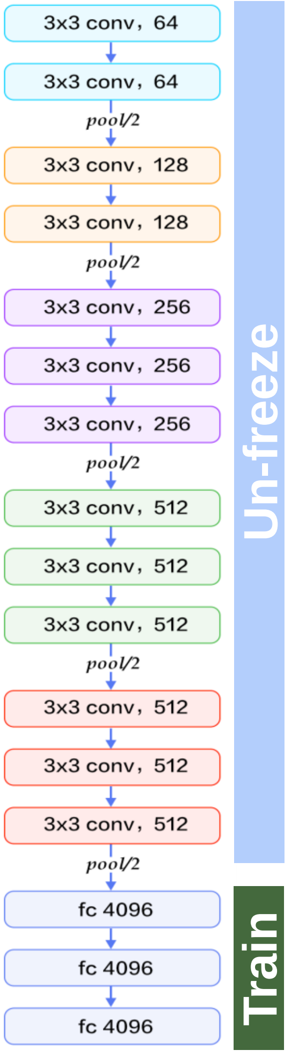
- Set-up network architecture
- Initialize very last layer randomly
- Train new parameters



Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. CVPR (pp. 806-813).

Transfer learning:
Slightly more data

- Set-up network architecture
- Initialize last layers randomly
- Train new parameters



Lower learning rate!
E.g. 1/10 of LR

[Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. CVPR (pp. 806-813).]

Transfer learning:

Slightly more data

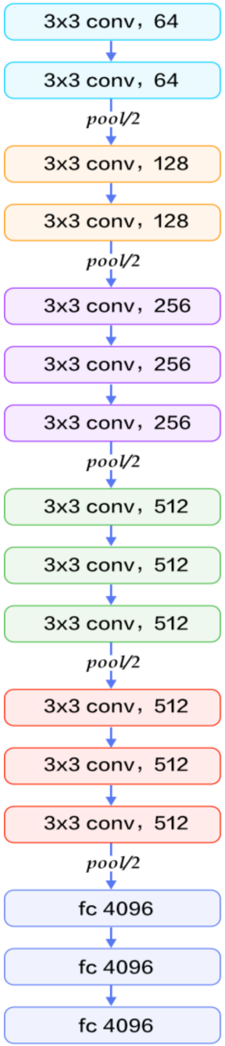
- Set-up network architecture
- Initialize last layers randomly
- Train new parameters

Second step: After some improvement in training

- Finetune complete network
- Carefully adjust LR to avoid "**forgetting**"

Q: Why does this work?

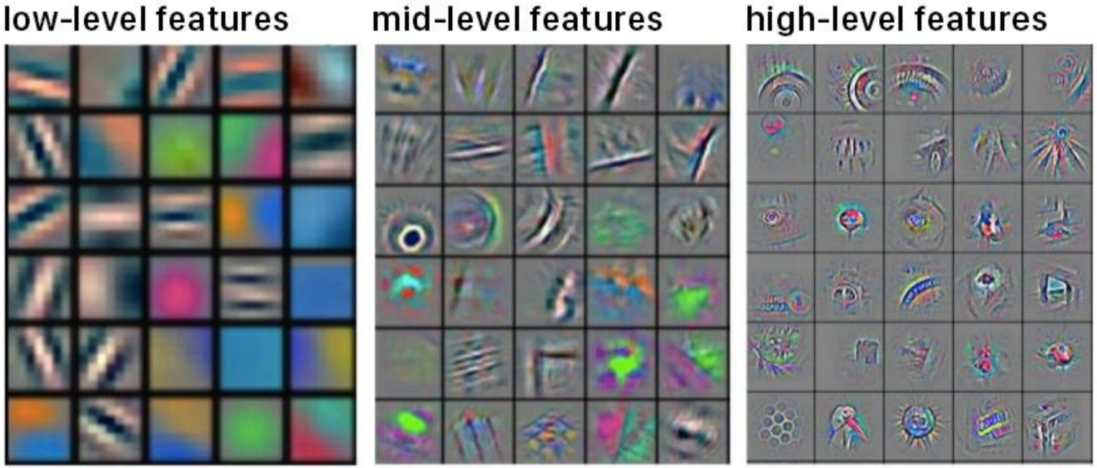
Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. CVPR (pp. 806-813).



Fairly generic

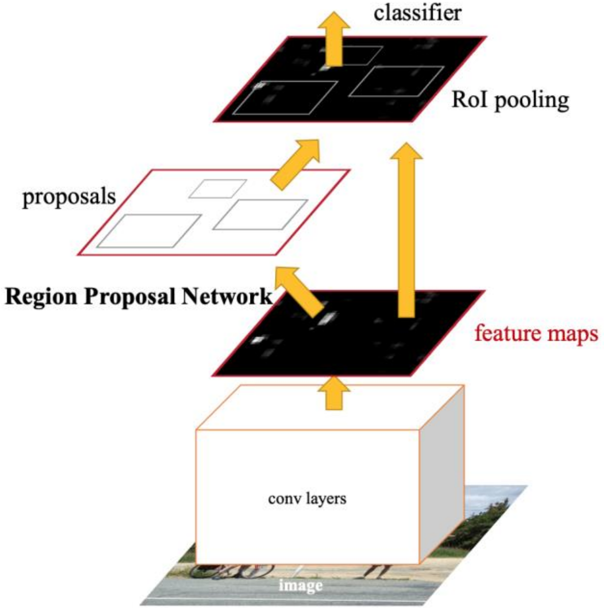
Rather specific

Why does this work?



Transfer Learning: It's the Norm!

- Transfer learning **is not a niche trick** for medical image analysis
- It is applied nearly everywhere, including in state-of-the-art CV methods



If your problem allows you to use transfer learning:

→ **Use it!**
Many tasks are very difficult to learn directly!

→ Also invest some thought in smart modeling

For many medical applications: impossible

3D data, time-series data, …

Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

Update Rules, Data Augmentation, Transfer Learning

Questions?