Search... All gists Back to GitHub Sign in Sign up

Instantly share code, notes, and snippets.

karpathy / microgpt.py

Last active now

Star 1,083 Fork 236

Code Revisions 10 Stars 1,083 Forks 236 Embed <scri Download ZIP

microgpt

microgpt.py Raw

```python
"""
The most atomic way to train and inference a GPT in pure, dependency-free Python.
This file is the complete algorithm.
Everything else is just efficiency.

@karpathy
"""

import os       # os.path.exists
import math     # math.log, math.exp
import random   # random.seed, random.choices, random.gauss, random.shuffle
random.seed(42) # Let there be order among chaos

# Let there be an input dataset `docs`: list[str] of documents (e.g. a dataset of names)
if not os.path.exists('input.txt'):
    import urllib.request
    names_url = 'https://raw.githubusercontent.com/karpathy/makemore/refs/heads/master/names.txt'
    urllib.request.urlretrieve(names_url, 'input.txt')
docs = [l.strip() for l in open('input.txt').read().strip().split('\n') if l.strip()] # list[str] of
random.shuffle(docs)
print(f"num docs: {len(docs)}")

# Let there be a Tokenizer to translate strings to discrete symbols and back
uchars = sorted(set(''.join(docs))) # unique characters in the dataset become token ids 0..n-1
BOS = len(uchars) # token id for the special Beginning of Sequence (BOS) token
vocab_size = len(uchars) + 1 # total number of unique tokens, +1 is for BOS
print(f"vocab size: {vocab_size}")

# Let there be an Autograd to apply the chain rule recursively across a computation graph
class Value:
    """Stores a single scalar value and its gradient, as a node in a computation graph."""

    def __init__(self, data, children=(), local_grads=()):
        self.data = data                # scalar value of this node calculated during forward pass
        self.grad = 0                   # derivative of the loss w.r.t. this node, calculated in bac
        self._children = children       # children of this node in the computation graph
        self._local_grads = local_grads # local derivative of this node w.r.t. its children

```

```
    def __add__(self, other):
        other = other if isinstance(other, Value) else Value(other)
        return Value(self.data + other.data, (self, other), (1, 1))

    def __mul__(self, other):
        other = other if isinstance(other, Value) else Value(other)
        return Value(self.data * other.data, (self, other), (other.data, self.data))

    def __pow__(self, other): return Value(self.data**other, (self,), (other * self.data**(other-1),
    def log(self): return Value(math.log(self.data), (self,), (1/self.data,))
    def exp(self): return Value(math.exp(self.data), (self,), (math.exp(self.data),))
    def relu(self): return Value(max(0, self.data), (self,), (float(self.data > 0),))
    def __neg__(self): return self * -1
    def __radd__(self, other): return self + other
    def __sub__(self, other): return self + (-other)
    def __rsub__(self, other): return other + (-self)
    def __rmul__(self, other): return self * other
    def __truediv__(self, other): return self * other**-1
    def __rtruediv__(self, other): return other * self**-1

    def backward(self):
        topo = []
        visited = set()
        def build_topo(v):
            if v not in visited:
                visited.add(v)
                for child in v._children:
                    build_topo(child)
                topo.append(v)
        build_topo(self)
        self.grad = 1
        for v in reversed(topo):
            for child, local_grad in zip(v._children, v._local_grads):
                child.grad += local_grad * v.grad

# Initialize the parameters, to store the knowledge of the model.
n_embd = 16     # embedding dimension
n_head = 4      # number of attention heads
n_layer = 1     # number of layers
block_size = 8  # maximum sequence length
head_dim = n_embd // n_head # dimension of each head
matrix = lambda nout, nin, std=0.02: [[Value(random.gauss(0, std)) for _ in range(nin)] for _ in ran
state_dict = {'wte': matrix(vocab_size, n_embd), 'wpe': matrix(block_size, n_embd), 'lm_head': matri
for i in range(n_layer):
    state_dict[f'layer{i}.attn_wq'] = matrix(n_embd, n_embd)
    state_dict[f'layer{i}.attn_wk'] = matrix(n_embd, n_embd)
    state_dict[f'layer{i}.attn_wv'] = matrix(n_embd, n_embd)
    state_dict[f'layer{i}.attn_wo'] = matrix(n_embd, n_embd, std=0)
    state_dict[f'layer{i}.mlp_fc1'] = matrix(4 * n_embd, n_embd)
    state_dict[f'layer{i}.mlp_fc2'] = matrix(n_embd, 4 * n_embd, std=0)
params = [p for mat in state_dict.values() for row in mat for p in row] # flatten params into a sing
print(f"num params: {len(params)}")

# Define the model architecture: a stateless function mapping token sequence and parameters to logit
# Follow GPT-2, blessed among the GPTs, with minor differences: layernorm -> rmsnorm, no biases, GeL
def linear(x, w):
    return [sum(wi * xi for wi, xi in zip(wo, x)) for wo in w]
```

```python

def softmax(logits):
    max_val = max(val.data for val in logits)
    exps = [(val - max_val).exp() for val in logits]
    total = sum(exps)
    return [e / total for e in exps]

def rmsnorm(x):
    ms = sum(xi * xi for xi in x) / len(x)
    scale = (ms + 1e-5) ** -0.5
    return [xi * scale for xi in x]

def gpt(token_id, pos_id, keys, values):
    tok_emb = state_dict['wte'][token_id] # token embedding
    pos_emb = state_dict['wpe'][pos_id] # position embedding
    x = [t + p for t, p in zip(tok_emb, pos_emb)] # joint token and position embedding
    x = rmsnorm(x)

    for li in range(n_layer):
        # 1) Multi-head attention block
        x_residual = x
        x = rmsnorm(x)
        q = linear(x, state_dict[f'layer{li}.attn_wq'])
        k = linear(x, state_dict[f'layer{li}.attn_wk'])
        v = linear(x, state_dict[f'layer{li}.attn_wv'])
        keys[li].append(k)
        values[li].append(v)
        x_attn = []
        for h in range(n_head):
            hs = h * head_dim
            q_h = q[hs:hs+head_dim]
            k_h = [ki[hs:hs+head_dim] for ki in keys[li]]
            v_h = [vi[hs:hs+head_dim] for vi in values[li]]
            attn_logits = [sum(q_h[j] * k_h[t][j] for j in range(head_dim)) / head_dim**0.5 for t in
            attn_weights = softmax(attn_logits)
            head_out = [sum(attn_weights[t] * v_h[t][j] for t in range(len(v_h))) for j in range(hea
            x_attn.extend(head_out)
        x = linear(x_attn, state_dict[f'layer{li}.attn_wo'])
        x = [a + b for a, b in zip(x, x_residual)]
        # 2) MLP block
        x_residual = x
        x = rmsnorm(x)
        x = linear(x, state_dict[f'layer{li}.mlp_fc1'])
        x = [xi.relu() ** 2 for xi in x]
        x = linear(x, state_dict[f'layer{li}.mlp_fc2'])
        x = [a + b for a, b in zip(x, x_residual)]

    logits = linear(x, state_dict['lm_head'])
    return logits

# Let there be Adam, the blessed optimizer and its buffers
learning_rate, beta1, beta2, eps_adam = 1e-2, 0.9, 0.95, 1e-8
m = [0.0] * len(params) # first moment buffer
v = [0.0] * len(params) # second moment buffer

# Repeat in sequence
num_steps = 500 # number of training steps
```

```python
for step in range(num_steps):

    # Take single document, tokenize it, surround it with BOS special token on both sides
    doc = docs[step % len(docs)]
    tokens = [BOS] + [uchars.index(ch) for ch in doc] + [BOS]
    n = min(block_size, len(tokens) - 1)

    # Forward the token sequence through the model, building up the computation graph all the way to
    keys, values = [[] for _ in range(n_layer)], [[] for _ in range(n_layer)]
    losses = []
    for pos_id in range(n):
        token_id, target_id = tokens[pos_id], tokens[pos_id + 1]
        logits = gpt(token_id, pos_id, keys, values)
        probs = softmax(logits)
        loss_t = -probs[target_id].log()
        losses.append(loss_t)
    loss = (1 / n) * sum(losses) # final average loss over the document sequence. May yours be low.

    # Backward the loss, calculating the gradients with respect to all model parameters.
    loss.backward()

    # Adam optimizer update: update the model parameters based on the corresponding gradients.
    lr_t = learning_rate * 0.5 * (1 + math.cos(math.pi * step / num_steps)) # cosine learning rate d
    for i, p in enumerate(params):
        m[i] = beta1 * m[i] + (1 - beta1) * p.grad
        v[i] = beta2 * v[i] + (1 - beta2) * p.grad ** 2
        m_hat = m[i] / (1 - beta1 ** (step + 1))
        v_hat = v[i] / (1 - beta2 ** (step + 1))
        p.data -= lr_t * m_hat / (v_hat ** 0.5 + eps_adam)
        p.grad = 0

    print(f"step {step+1:4d} / {num_steps:4d} | loss {loss.data:.4f}")

# Inference: may the model babble back to us
temperature = 0.5 # in (0, 1], control the "creativity" of generated text, low to high
print("\n--- inference ---")
for sample_idx in range(20):
    keys, values = [[] for _ in range(n_layer)], [[] for _ in range(n_layer)]
    token_id = BOS
    sample = []
    for pos_id in range(block_size):
        logits = gpt(token_id, pos_id, keys, values)
        probs = softmax([l / temperature for l in logits])
        token_id = random.choices(range(vocab_size), weights=[p.data for p in probs])[0]
        if token_id == BOS:
            break
        sample.append(uchars[token_id])
    print(f"sample {sample_idx+1:2d}: {''.join(sample)}")
```

Load earlier comments...

mgalgs commented 11 hours ago

Hello, future history books 👋

dkarapetyan-afk commented 11 hours ago

Kinda weird how the dual vector version was deleted when it's actually a lot simpler.

Nucs commented 11 hours ago

Its prettier when you called it art.

davidkimai commented 11 hours ago • edited

> uv run --python pypy microgpt.py

thanks for this

mpmisko commented 11 hours ago • edited

"final average loss over the document sequence. May yours be low." :)

davidkimai commented 10 hours ago

now everyone will be saying they trained a neural network from scratch haha **@karpathy**

bufrr commented 9 hours ago

goat

atgctagc commented 9 hours ago

just mad stuff i say !

emharsha1812 commented 9 hours ago

Legendary, keep building guys!

yitaochen commented 9 hours ago

it helps to understand a lot of fundamental stuff!

sundaram2021 commented 9 hours ago

Banger OP

GustavoNicot commented 8 hours ago

Awesome study material, thanks!

Quick design-intent questions on the latest modifications 🙏

1. Untied lm_head: clearer conceptually, but more params in tiny models. Did you see a quality/stability gain?
2. Extra rmsnorm after tok+pos embedding: measurable stability benefit, or mainly narrative consistency?

samjundi1 commented 8 hours ago

Awesome 👏

lonngxiang commented 7 hours ago

coolest

khakra commented 7 hours ago

you are the GOAT 🐐

rshrt commented 7 hours ago

Welcome to Termux!

Docs: https://termux.dev/docs
Donate: https://termux.dev/donate
Community: https://termux.dev/community

Working with packages:

- Search: pkg search
- Install: pkg install
- Upgrade: pkg upgrade

Subscribing to additional repositories:

- Root: pkg install root-repo
- X11: pkg install x11-repo

For fixing any repository issues,
try 'termux-change-repo' command.

Report issues at https://termux.dev/issues
~ $ nano minigpt.py
~ $ python minigpt.py
num docs: 32033
vocab size: 27
num params: 4064
step 1 / 500 | loss 3.2627
step 2 / 500 | loss 3.3003
step 3 / 500 | loss 3.2099
step 4 / 500 | loss 3.2991
step 5 / 500 | loss 3.2988
step 6 / 500 | loss 3.1658
step 7 / 500 | loss 3.1818
step 8 / 500 | loss 3.2133
step 9 / 500 | loss 3.0232
step 10 / 500 | loss 3.2242
step 11 / 500 | loss 3.0036
step 12 / 500 | loss 3.0079
step 13 / 500 | loss 3.2446
step 14 / 500 | loss 3.2963
step 15 / 500 | loss 3.0254
step 16 / 500 | loss 2.7848
step 17 / 500 | loss 2.8743
step 18 / 500 | loss 2.8132
step 19 / 500 | loss 2.7221
step 20 / 500 | loss 2.4210
step 21 / 500 | loss 3.3552
step 22 / 500 | loss 2.5702
step 23 / 500 | loss 2.6674
step 24 / 500 | loss 2.1306
step 25 / 500 | loss 3.1513
step 26 / 500 | loss 2.7268
step 27 / 500 | loss 3.4720
step 28 / 500 | loss 3.0404

```
step 29 / 500 | loss 2.2646
step 30 / 500 | loss 2.3740
step 31 / 500 | loss 3.1029
step 32 / 500 | loss 3.0062
step 33 / 500 | loss 2.7890
step 34 / 500 | loss 2.0912
step 35 / 500 | loss 3.2122
step 36 / 500 | loss 2.7511
step 37 / 500 | loss 2.4796
step 38 / 500 | loss 3.1469
step 39 / 500 | loss 2.5389
step 40 / 500 | loss 2.9382
step 41 / 500 | loss 2.4514
step 42 / 500 | loss 2.5784
step 43 / 500 | loss 2.9148
step 44 / 500 | loss 2.5582
step 45 / 500 | loss 2.2062
step 46 / 500 | loss 3.1859
step 47 / 500 | loss 2.5239
step 48 / 500 | loss 3.1970
step 49 / 500 | loss 2.6446
step 50 / 500 | loss 2.3823
step 51 / 500 | loss 3.2570
step 52 / 500 | loss 3.0418
step 53 / 500 | loss 2.9444
step 54 / 500 | loss 2.2813
step 55 / 500 | loss 2.7159
step 56 / 500 | loss 2.0570
step 57 / 500 | loss 2.4133
step 58 / 500 | loss 2.2158
step 59 / 500 | loss 2.7959
step 60 / 500 | loss 3.3081
step 61 / 500 | loss 2.7878
step 62 / 500 | loss 2.6010
step 63 / 500 | loss 2.6296
step 64 / 500 | loss 3.0765
step 65 / 500 | loss 2.9856
step 66 / 500 | loss 2.9788
step 67 / 500 | loss 2.9475
step 68 / 500 | loss 2.8266
step 69 / 500 | loss 2.5131
step 70 / 500 | loss 3.1322
step 71 / 500 | loss 3.0503
step 72 / 500 | loss 2.6245
step 73 / 500 | loss 2.5673
step 74 / 500 | loss 2.6100
step 75 / 500 | loss 2.3007
step 76 / 500 | loss 2.8391
step 77 / 500 | loss 3.2412
step 78 / 500 | loss 2.7156
step 79 / 500 | loss 2.4544
step 80 / 500 | loss 2.4062
step 81 / 500 | loss 2.4320
step 82 / 500 | loss 3.3576
```

```
step 83 / 500 | loss 2.7153
step 84 / 500 | loss 2.5814
step 85 / 500 | loss 2.3547
step 86 / 500 | loss 2.6444
step 87 / 500 | loss 2.8423
step 88 / 500 | loss 2.9311
step 89 / 500 | loss 2.8385
step 90 / 500 | loss 2.2634
step 91 / 500 | loss 2.5663
step 92 / 500 | loss 3.5424
step 93 / 500 | loss 2.5540
step 94 / 500 | loss 3.1618
step 95 / 500 | loss 2.6365
step 96 / 500 | loss 2.4827
step 97 / 500 | loss 2.0138
step 98 / 500 | loss 2.8764
step 99 / 500 | loss 2.3324
step 100 / 500 | loss 3.6054
step 101 / 500 | loss 2.3759
step 102 / 500 | loss 2.1912
step 103 / 500 | loss 2.8268
step 104 / 500 | loss 2.6508
step 105 / 500 | loss 2.5131
step 106 / 500 | loss 2.3524
step 107 / 500 | loss 3.1972
step 108 / 500 | loss 2.0116
step 109 / 500 | loss 2.1438
step 110 / 500 | loss 2.5797
step 111 / 500 | loss 2.8299
step 112 / 500 | loss 2.6918
step 113 / 500 | loss 2.6484
step 114 / 500 | loss 2.3999
step 115 / 500 | loss 2.1887
step 116 / 500 | loss 2.7798
step 117 / 500 | loss 2.6562
step 118 / 500 | loss 2.1591
step 119 / 500 | loss 2.4257
step 120 / 500 | loss 2.6780
step 121 / 500 | loss 2.4442
step 122 / 500 | loss 2.2111
step 123 / 500 | loss 2.9350
step 124 / 500 | loss 2.5761
step 125 / 500 | loss 3.1058
step 126 / 500 | loss 2.1808
step 127 / 500 | loss 2.5362
step 128 / 500 | loss 2.0988
step 129 / 500 | loss 2.2613
step 130 / 500 | loss 3.0241
step 131 / 500 | loss 3.1803
step 132 / 500 | loss 1.8743
step 133 / 500 | loss 2.6976
step 134 / 500 | loss 2.5843
step 135 / 500 | loss 3.0481
step 136 / 500 | loss 2.9203
```

```
step 137 / 500 | loss 2.1693
step 138 / 500 | loss 2.3851
step 139 / 500 | loss 2.7720
step 140 / 500 | loss 2.3444
step 141 / 500 | loss 3.0443
step 142 / 500 | loss 2.4185
step 143 / 500 | loss 3.1157
step 144 / 500 | loss 2.4548
step 145 / 500 | loss 2.5489
step 146 / 500 | loss 2.4013
step 147 / 500 | loss 2.2730
step 148 / 500 | loss 2.4859
step 149 / 500 | loss 2.1762
step 150 / 500 | loss 2.7279
step 151 / 500 | loss 2.8874
step 152 / 500 | loss 2.4087
step 153 / 500 | loss 2.9016
step 154 / 500 | loss 3.2959
step 155 / 500 | loss 2.7739
step 156 / 500 | loss 2.8204
step 157 / 500 | loss 3.2933
step 158 / 500 | loss 1.8516
step 159 / 500 | loss 3.3137
step 160 / 500 | loss 2.0370
step 161 / 500 | loss 1.9895
step 162 / 500 | loss 2.5055
step 163 / 500 | loss 2.6972
step 164 / 500 | loss 2.6840
step 165 / 500 | loss 2.4457
step 166 / 500 | loss 3.0005
step 167 / 500 | loss 2.1723
step 168 / 500 | loss 2.5101
step 169 / 500 | loss 3.2593
step 170 / 500 | loss 2.5678
step 171 / 500 | loss 2.4797
step 172 / 500 | loss 2.3465
step 173 / 500 | loss 2.1675
step 174 / 500 | loss 3.0380
step 175 / 500 | loss 2.9214
step 176 / 500 | loss 2.3547
step 177 / 500 | loss 3.2598
step 178 / 500 | loss 2.3522
step 179 / 500 | loss 1.9996
step 180 / 500 | loss 2.3344
step 181 / 500 | loss 2.2645
step 182 / 500 | loss 2.0609
step 183 / 500 | loss 1.8380
step 184 / 500 | loss 3.2882
step 185 / 500 | loss 2.1918
step 186 / 500 | loss 2.8884
step 187 / 500 | loss 2.4940
step 188 / 500 | loss 2.3357
step 189 / 500 | loss 1.9098
step 190 / 500 | loss 2.9308
```

```
step 191 / 500 | loss 1.8054
step 192 / 500 | loss 2.4166
step 193 / 500 | loss 2.1768
step 194 / 500 | loss 2.8127
step 195 / 500 | loss 2.7636
step 196 / 500 | loss 2.6237
step 197 / 500 | loss 2.2088
step 198 / 500 | loss 3.0606
step 199 / 500 | loss 2.4676
step 200 / 500 | loss 2.2599
step 201 / 500 | loss 2.4126
step 202 / 500 | loss 2.5267
step 203 / 500 | loss 2.0953
step 204 / 500 | loss 1.9941
step 205 / 500 | loss 2.8427
step 206 / 500 | loss 2.9501
step 207 / 500 | loss 2.9538
step 208 / 500 | loss 2.3423
step 209 / 500 | loss 2.2279
step 210 / 500 | loss 2.4999
step 211 / 500 | loss 2.0527
step 212 / 500 | loss 2.4459
step 213 / 500 | loss 4.1133
step 214 / 500 | loss 2.0608
step 215 / 500 | loss 2.5085
step 216 / 500 | loss 2.6548
step 217 / 500 | loss 2.4265
step 218 / 500 | loss 2.5753
step 219 / 500 | loss 2.2504
step 220 / 500 | loss 2.4038
step 221 / 500 | loss 3.1136
step 222 / 500 | loss 2.4151
step 223 / 500 | loss 2.2587
step 224 / 500 | loss 2.2654
step 225 / 500 | loss 2.4760
step 226 / 500 | loss 2.3599
step 227 / 500 | loss 2.4675
step 228 / 500 | loss 2.5490
step 229 / 500 | loss 2.9474
step 230 / 500 | loss 1.9947
step 231 / 500 | loss 2.8530
step 232 / 500 | loss 2.4313
step 233 / 500 | loss 2.3576
step 234 / 500 | loss 3.1088
step 235 / 500 | loss 2.2674
step 236 / 500 | loss 2.7280
step 237 / 500 | loss 2.2991
step 238 / 500 | loss 2.5833
step 239 / 500 | loss 2.2114
step 240 / 500 | loss 3.8698
step 241 / 500 | loss 3.3756
step 242 / 500 | loss 2.5886
step 243 / 500 | loss 2.3948
step 244 / 500 | loss 2.3593
```

```
step 245 / 500 | loss 2.0913
step 246 / 500 | loss 2.5268
step 247 / 500 | loss 2.4023
step 248 / 500 | loss 2.4964
step 249 / 500 | loss 1.9844
step 250 / 500 | loss 2.3555
step 251 / 500 | loss 1.8328
step 252 / 500 | loss 2.3015
step 253 / 500 | loss 2.6677
step 254 / 500 | loss 2.0878
step 255 / 500 | loss 2.3283
step 256 / 500 | loss 3.3943
step 257 / 500 | loss 2.4298
step 258 / 500 | loss 2.0373
step 259 / 500 | loss 1.9987
step 260 / 500 | loss 1.9028
step 261 / 500 | loss 2.6001
step 262 / 500 | loss 2.1300
step 263 / 500 | loss 2.9124
step 264 / 500 | loss 3.1229
step 265 / 500 | loss 1.8178
step 266 / 500 | loss 2.6388
step 267 / 500 | loss 1.9881
step 268 / 500 | loss 2.0996
step 269 / 500 | loss 2.0829
step 270 / 500 | loss 2.6435
step 271 / 500 | loss 2.2508
step 272 / 500 | loss 1.8907
step 273 / 500 | loss 2.5203
step 274 / 500 | loss 3.0554
step 275 / 500 | loss 2.1154
step 276 / 500 | loss 2.5115
step 277 / 500 | loss 3.0102
step 278 / 500 | loss 2.0989
step 279 / 500 | loss 2.2267
step 280 / 500 | loss 2.0100
step 281 / 500 | loss 3.0744
step 282 / 500 | loss 1.9592
step 283 / 500 | loss 2.3102
step 284 / 500 | loss 2.9254
step 285 / 500 | loss 3.2898
^[[D^[[D^[[D^[[Dstep 286 / 500 | loss 2.0904
step 287 / 500 | loss 2.4916
step 288 / 500 | loss 2.5988
step 289 / 500 | loss 2.3593
step 290 / 500 | loss 2.2278
step 291 / 500 | loss 3.5604
step 292 / 500 | loss 2.2703
step 293 / 500 | loss 2.0005
step 294 / 500 | loss 2.4734
step 295 / 500 | loss 2.9711
step 296 / 500 | loss 2.1368
step 297 / 500 | loss 2.3366
step 298 / 500 | loss 2.5831
```

```
step 299 / 500 | loss 2.2470
step 300 / 500 | loss 2.3347
step 301 / 500 | loss 2.5127
step 302 / 500 | loss 2.2975
step 303 / 500 | loss 2.7533
step 304 / 500 | loss 2.3857
step 305 / 500 | loss 2.8193
step 306 / 500 | loss 2.5267
step 307 / 500 | loss 1.7167
step 308 / 500 | loss 2.6465
step 309 / 500 | loss 2.0310
step 310 / 500 | loss 2.3856
step 311 / 500 | loss 2.9230
step 312 / 500 | loss 1.9773
step 313 / 500 | loss 1.9133
step 314 / 500 | loss 2.4401
step 315 / 500 | loss 3.0532
step 316 / 500 | loss 2.0706
step 317 / 500 | loss 1.9435
step 318 / 500 | loss 1.8844
step 319 / 500 | loss 2.0878
step 320 / 500 | loss 1.8774
step 321 / 500 | loss 2.5767
step 322 / 500 | loss 2.3483
step 323 / 500 | loss 2.8436
step 324 / 500 | loss 3.1205
step 325 / 500 | loss 2.6596
step 326 / 500 | loss 2.1256
step 327 / 500 | loss 2.3023
step 328 / 500 | loss 1.9138
step 329 / 500 | loss 2.0737
step 330 / 500 | loss 2.5025
step 331 / 500 | loss 2.1305
step 332 / 500 | loss 2.3430
step 333 / 500 | loss 1.9868
step 334 / 500 | loss 2.6293
step 335 / 500 | loss 3.2008
step 336 / 500 | loss 3.1295
step 337 / 500 | loss 2.7195
step 338 / 500 | loss 2.5143
step 339 / 500 | loss 2.2770
step 340 / 500 | loss 2.0494
step 341 / 500 | loss 3.1696
step 342 / 500 | loss 3.0974
step 343 / 500 | loss 1.6473
step 344 / 500 | loss 2.5096
step 345 / 500 | loss 2.7233
step 346 / 500 | loss 2.4147
step 347 / 500 | loss 3.4384
step 348 / 500 | loss 2.7714
step 349 / 500 | loss 2.7640
step 350 / 500 | loss 2.2334
step 351 / 500 | loss 3.3242
step 352 / 500 | loss 2.0447
```

```
step 353 / 500 | loss 2.3633
step 354 / 500 | loss 2.9007
step 355 / 500 | loss 2.5440
step 356 / 500 | loss 1.9885
step 357 / 500 | loss 2.7534
step 358 / 500 | loss 2.0990
step 359 / 500 | loss 1.9703
step 360 / 500 | loss 2.4822
step 361 / 500 | loss 2.4378
step 362 / 500 | loss 2.5672
step 363 / 500 | loss 1.8511
step 364 / 500 | loss 1.9484
step 365 / 500 | loss 2.4531
step 366 / 500 | loss 1.9472
step 367 / 500 | loss 3.0414
step 368 / 500 | loss 2.4095
step 369 / 500 | loss 3.1801
step 370 / 500 | loss 2.0925
step 371 / 500 | loss 3.1926
step 372 / 500 | loss 2.1203
step 373 / 500 | loss 2.9379
step 374 / 500 | loss 2.1473
step 375 / 500 | loss 2.0381
step 376 / 500 | loss 2.4270
step 377 / 500 | loss 2.5262
step 378 / 500 | loss 2.3275
step 379 / 500 | loss 2.5202
step 380 / 500 | loss 2.1807
step 381 / 500 | loss 2.3467
step 382 / 500 | loss 2.5653
step 383 / 500 | loss 2.2723
step 384 / 500 | loss 3.2872
step 385 / 500 | loss 3.3735
step 386 / 500 | loss 2.1963
step 387 / 500 | loss 3.8132
step 388 / 500 | loss 2.5678
step 389 / 500 | loss 2.3045
step 390 / 500 | loss 3.5271
step 391 / 500 | loss 2.7044
step 392 / 500 | loss 2.1275
step 393 / 500 | loss 2.0966
step 394 / 500 | loss 2.2294
step 395 / 500 | loss 2.4373
step 396 / 500 | loss 1.8814
step 397 / 500 | loss 2.4239
step 398 / 500 | loss 2.7811
step 399 / 500 | loss 2.5670
step 400 / 500 | loss 2.2285
step 401 / 500 | loss 2.7245
step 402 / 500 | loss 2.7011
step 403 / 500 | loss 1.9267
step 404 / 500 | loss 2.2126
step 405 / 500 | loss 2.7152
step 406 / 500 | loss 2.4033
```

```
step 407 / 500 | loss 2.4498
step 408 / 500 | loss 2.6116
step 409 / 500 | loss 2.8810
step 410 / 500 | loss 2.8391
step 411 / 500 | loss 2.0994
step 412 / 500 | loss 2.1234
step 413 / 500 | loss 2.2493
step 414 / 500 | loss 2.7472
step 415 / 500 | loss 2.0906
step 416 / 500 | loss 2.5594
step 417 / 500 | loss 2.8663
step 418 / 500 | loss 2.0470
step 419 / 500 | loss 2.5046
step 420 / 500 | loss 2.6694
step 421 / 500 | loss 2.4935
step 422 / 500 | loss 2.7274
step 423 / 500 | loss 1.9117
step 424 / 500 | loss 2.6567
step 425 / 500 | loss 2.6436
step 426 / 500 | loss 2.6221
step 427 / 500 | loss 2.3672
step 428 / 500 | loss 2.3567
step 429 / 500 | loss 1.9891
step 430 / 500 | loss 2.1652
step 431 / 500 | loss 2.0517
step 432 / 500 | loss 2.0038
step 433 / 500 | loss 2.2138
step 434 / 500 | loss 2.2330
step 435 / 500 | loss 2.6880
step 436 / 500 | loss 3.2736
step 437 / 500 | loss 2.7386
step 438 / 500 | loss 1.9579
step 439 / 500 | loss 1.9782
step 440 / 500 | loss 2.3770
step 441 / 500 | loss 2.8091
step 442 / 500 | loss 3.1708
step 443 / 500 | loss 3.3768
step 444 / 500 | loss 2.4385
step 445 / 500 | loss 2.4068
step 446 / 500 | loss 1.9930
step 447 / 500 | loss 1.9045
step 448 / 500 | loss 2.8939
step 449 / 500 | loss 2.1217
step 450 / 500 | loss 2.8947
step 451 / 500 | loss 2.8754
step 452 / 500 | loss 2.9844
step 453 / 500 | loss 2.4303
step 454 / 500 | loss 1.9799
step 455 / 500 | loss 2.0474
step 456 / 500 | loss 2.7910
step 457 / 500 | loss 2.6828
step 458 / 500 | loss 3.0521
step 459 / 500 | loss 1.7399
step 460 / 500 | loss 2.3532
```

```
step 461 / 500 | loss 2.0048
step 462 / 500 | loss 2.7350
step 463 / 500 | loss 1.9275
step 464 / 500 | loss 2.2453
step 465 / 500 | loss 2.6158
step 466 / 500 | loss 2.2389
step 467 / 500 | loss 1.8583
step 468 / 500 | loss 2.1809
step 469 / 500 | loss 2.3373
step 470 / 500 | loss 3.9840
step 471 / 500 | loss 2.5348
step 472 / 500 | loss 2.9083
step 473 / 500 | loss 2.4527
step 474 / 500 | loss 2.3297
step 475 / 500 | loss 2.3749
step 476 / 500 | loss 2.2957
step 477 / 500 | loss 3.5735
step 478 / 500 | loss 1.9963
step 479 / 500 | loss 2.0562
step 480 / 500 | loss 2.0096
step 481 / 500 | loss 3.6592
step 482 / 500 | loss 3.0683
step 483 / 500 | loss 2.5714
step 484 / 500 | loss 2.8548
step 485 / 500 | loss 2.5222
step 486 / 500 | loss 2.1049
step 487 / 500 | loss 3.0359
step 488 / 500 | loss 2.7135
step 489 / 500 | loss 2.3227
step 490 / 500 | loss 2.4628
step 491 / 500 | loss 2.0350
step 492 / 500 | loss 2.4197
step 493 / 500 | loss 2.4624
step 494 / 500 | loss 1.8329
step 495 / 500 | loss 2.6793
step 496 / 500 | loss 2.5143
step 497 / 500 | loss 2.7125
step 498 / 500 | loss 2.3307
step 499 / 500 | loss 2.3531
step 500 / 500 | loss 2.0160

--- inference ---
sample 1: lellen
sample 2: keles
sample 3: aylera
sample 4: kellone
sample 5: aman
sample 6: lela
sample 7: ameri
sample 8: kan
sample 9: nareena
sample 10: aliela
sample 11: seyn
sample 12: daman
```

sample 13: caaren
sample 14: ozyren
sample 15: kahiea
sample 16: anytte
sample 17: shilol
sample 18: deler
sample 19: azele
sample 20: maton

Better than I expected

okjodom commented 7 hours ago

> Now also available here on a single page :) https://karpathy.ai/microgpt.html

Makes for excellent wall art!

KarthickSelvam commented 6 hours ago

> Now also available here on a single page :) https://karpathy.ai/microgpt.html

Cool poster idea!

mario-deblock commented 6 hours ago

here for the culture

karpathy commented 6 hours ago Author

I thought about it more and realized that the autograd engine can be even more dramatically simplified. 243 lines of code is now down to 200, ~18% reduction. Surely this is it now!

PrinceKay145 commented 5 hours ago

oh wow, this is getting even more interesting 👏.

vphatak commented 5 hours ago

Oh wow. I am building 3 different AI models from the ground up using a modified version of NanoGPT and now this opens up a whole new world of possibilities!!! Dude!!! Thank you! If you are ever in Austin, TX I would love to buy you dinner. Or if you will be attending the RSA cybersecurity conference, please let me grab you a coffee as a thank you.

MkI21 commented 4 hours ago

Wow

sweet2honey commented 4 hours ago

Cool

vayqerlukashakkarainen commented 2 hours ago

> Now also available here on a single page :) https://karpathy.ai/microgpt.html

Made a quick background image to have as a desktop background, cause I am a nerd and it looks goddamn
cool.

Malko17 commented 2 hours ago

The K&R of language models.
Karpathy Sensei. Chapeau !

skannan-maf commented 1 hour ago

I will take Maton... ozyren is a second choice...

Kaartik777 commented 24 minutes ago

Only if the transformer paper had this as appendix

taomin15201212 commented 10 minutes ago

BookMark

2CoderOK commented 4 minutes ago

Wow :)
180 lines (without comments)
Goat!

Sign up for free **to join this conversation on GitHub.** Already have an account? Sign in to comment

Terms Privacy Security Status Community Docs Contact Manage cookies
Do not share my personal information

© 2026 GitHub, Inc.